Online Food Recipe Web

Application

**By**

**Ramdas Sakpal– AF04959590**

**Aniket Londhe– AF04958065**

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr.no** | **Topic** | **Page no** |
| 1 | Title of Project | 1 |
| 2 | Acknowledgement | 3 |
| 3 | Abstract | 4 |
| 4 | Introduction | 5 |
| 5 | Objective | 6 |
| 6 | System Analysis | 7 |
| 7 | System Design | 22 |
| 8 | Screenshots | 32 |
| 9 | Coding | 39 |
| 10 | Testing | 46 |
| 11 | Report | 47 |
| 12 | Future Scope | 48 |
| 13 | Conclusion | 48 |
| 14 | Bibliography | 49 |
| 15 | References | 49 |

# Acknowledgement

The project **“Online News Portal with Sentiment Analysis”** is the Project work carried out by

|  |  |
| --- | --- |
| **Name** | **Enrollment No** |
| **Ramdas Sakpal** | **AF04959590** |
| **Aniket Londhe** | **AF04958065** |

Under the Guidance.

We are thankful to my project guide for guiding me to complete the Project.

His suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.

# Abstract

# The Food Recipe App is a dynamic and user-friendly web application designed to provide users with a wide variety of food recipes categorized by cuisine, meal type, and dietary preferences. This application allows users to explore recipes with detailed ingredients, step-by-step preparation methods, and cooking tips. It also offers features like user authentication, personalized recipe suggestions, and the ability to bookmark or share favorite recipes.

# The system is developed using React.js for the frontend, ensuring a responsive and interactive user experience, while Node.js and Express.js handle the backend operations. The application uses MySQL as the database to securely store user data and recipe information. The primary aim of this project is to simplify the cooking process by offering a digital platform where users can easily access, manage, and enjoy recipes anytime, anywhere.

# This documentation outlines the objectives, system requirements, architecture, and workflow of the Food Recipe App, providing a comprehensive overview of its design and functionality.

# Introduction

Cooking has always been an essential part of daily life, but in today’s fast-paced world, people often look for convenient ways to explore and prepare new dishes. The Food Recipe App has been developed to address this need by providing a digital platform where users can easily search, view, and manage a wide range of recipes.

The app not only helps beginners learn cooking through step-by-step instructions but also caters to experienced users by offering diverse recipe collections across multiple cuisines and dietary preferences. Users can log in to the system, access personalized recipe suggestions, and save their favorite dishes for future reference.

Technically, the application is built using React.js for creating an interactive and responsive user interface, while Node.js with Express.js manages backend services such as authentication, API handling, and recipe management. MySQL serves as the database, ensuring efficient storage and retrieval of user details and recipe data.

The main objective of this project is to simplify the process of discovering and preparing meals by delivering an intuitive, organized, and accessible recipe management system. This documentation provides a detailed overview of the app’s design, architecture, workflow, and implementation to highlight how it meets user requirements.

## Objective of the Present Work

The objectives of this project are as follows:

* + - To develop a web application for an Online News Portal website that can keep people informed.
    - To provide daily news updates.
    - To provide breaking news updates in real-time.
    - To deliver news in an easy-to-understand, visually appealing, and engaging way.
    - To ensure wide coverage and faster dissemination of information through the portal.
    - To make it possible for anyone, anywhere, and anytime to access news at a low cost via the internet.
    - To create a dynamic platform that allows adding new information easily without any complex

**System analysis**

The Food Recipe App is developed to provide users with an easy-to-use platform for exploring, managing, and saving recipes.

**Existing System**

Users rely on cookbooks, blogs, or scattered online sources, which lack personalization, categorization, and bookmarking features.

**Proposed System**

**The app provides:**

* User authentication (login & registration)
* Categorized and searchable recipes
* Step-by-step instructions with ingredients
* Bookmarking favorite recipes
* Responsive and user-friendly interface

**Feasibility Study**

* Technical: Built with React.js, Node.js, Express.js, and MySQL (all open-source).
* Operational: Easy to use with basic internet access.
* Economic: Cost-effective due to free technologies.

**Requirements**

**Functional**: Login/Signup, recipe browsing, recipe details, search, bookmarking.

**Non-Functional**: Secure, fast, scalable, and mobile-friendly.

* 1. **PROBLEM DEFINITION**

In today’s busy lifestyle, people struggle to find reliable, well-organized, and easy-to-follow recipes. Existing sources like cookbooks, blogs, or videos are scattered, lack personalization, and do not provide features such as bookmarking or categorization. There is a need for a centralized platform where users can quickly search, view, and save recipes in a simple and user-friendly way.

* 1. **Preliminary Investigation Purpose**

The purpose of the Food Recipe App is to provide a centralized and user-friendly platform where users can easily search, explore, and manage recipes. The app aims to simplify cooking by offering step-by-step instructions, categorized recipes, and bookmarking features. It also ensures a personalized experience through user authentication, making it easier for individuals to access and save their favorite recipes anytime, anywhere

**Benefits**

* Easy Access: Users can quickly find recipes anytime, anywhere.
* Categorization: Recipes are organized by cuisine, meal type, and dietary needs.
* Step-by-Step Guidance: Clear instructions help both beginners and experienced cooks.
* Time-Saving: Eliminates the need to search across multiple sources.
* User-Friendly: Responsive and simple interface accessible on different devices.

**Proposed System**

The **Food Recipe App** offers a user-friendly and centralized platform to access recipes

with step-by-step instructions. It includes features like secure login, categorized recipes,

search and filter options, bookmarking favorites, and a responsive design. This system

simplifies cooking, provides personalized access, and ensures scalability for future

updates.

* 1. **Feasibility Study**
     + - The feasibility study ensures that the Food Recipe App is practical, cost-effective, and user-friendly.
       - Technical Feasibility:
       - The app is built using React.js, Node.js, Express.js, and MySQL – all open-source, reliable, and widely supported technologies.
       - Operational Feasibility:
       - The system is simple to use with an intuitive interface. Users only need basic internet access and a device (mobile/PC).
       - Economic Feasibility:
       - Development and maintenance costs are minimal as the app uses free technologies, making it affordable and sustainable.
       - Modular development ensures different parts of the system are built incrementally, improving flexibility.
  2. Project Planning

Purpose of Project Planning

Project planning ensures that the development of the **Online Food Recipe Web Application** follows a structured approach. It helps define the workflow, resource allocation, timelines, and risks involved to deliver the system efficiently.

Phases Covered in the Plan

The Food Recipe App is planned and developed in different phases to ensure smooth progress and timely completion.

Phases of Project Planning:

Requirement Analysis:

Collect user needs such as login, recipe search, categorization, and bookmarking.

System Design:

Plan architecture, database structure (MySQL), and UI/UX layout using React.js.

Development:

Frontend: Build responsive interface with React.js, HTML, CSS, JS.

Backend: Implement APIs and authentication with Node.js & Express.js.

Database: Store user and recipe data in MySQL.

Testing:

Test functionality, security, and performance to ensure a smooth user experience.

Deployment:

Launch the app on a server for user access.

Maintenance & Updates:

Regularly update recipes, fix bugs, and add new features.

Project Planning Goal:

To deliver a user-friendly, scalable, and reliable recipe platform within the planned timeframe and resources..



**Start**

Preliminary

System Analysis

System Design

Coding

Testing

Security

Implementation

**Stop**

* 1. Software Requirement Specification (SRS)

The **Software Requirement Specification (SRS)** outlines the fundamental requirements of the **Online Food Recipe Web Application** to ensure efficient functionality, usability, and maintainability.

System Overview

The Food Recipe App is a web-based platform designed to help users easily find, view, and manage recipes. It provides secure login, categorized recipe browsing, step-by-step cooking instructions, and bookmarking features.

**Frontend: Developed using React.js, HTML, CSS, and JavaScript for a responsive and interactive user interface.**

**Backend: Built with Node.js and Express.js to handle authentication, recipe management, and API requests.**

**Database: MySQL is used for storing user details, login credentials, and recipe information.**

**Users:**

**Registered Users – can log in, search, view, and bookmark recipes.**

**Guests – can browse recipes with limited access.**

Software & Hardware Requirements Software Requirements

* + **Frontend**: HTML, CSS, React JS for responsive UI
  + **Backend**: Node JS for data handling and security
  + **Database**: MySQL for storing articles, comments, and user details

Hardware Requirements

* + **Processor**: Intel i5 or higher
  + **RAM**: Minimum 8GB
  + **Storage**: At least 100GB for database and media files
  + **Connectivity**: Internet access for real-time updates
  1. Functional Requirements

The **Food Recipe App** must support the following key functions:

1. **User Authentication** – Allow users to register, log in, and log out securely.
2. **Recipe Browsing** – View recipes categorized by cuisine, meal type, and dietary preferences.
3. **Search & Filter** – Search recipes by name, ingredients, or category.
4. **Recipe Details** – Display ingredients, step-by-step preparation instructions, and cooking time.
5. **Bookmarking** – Save favorite recipes for quick future access.
6. **User Profile Management** – Update personal details and manage saved recipes.
7. **Responsive Design** – Ensure smooth access across devices (desktop & mobile).
   1. ​Software Engineering Paradigm

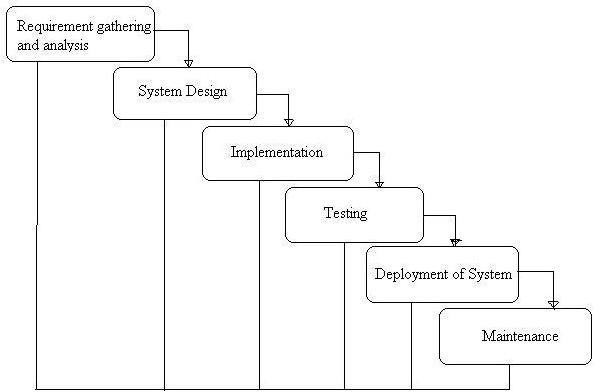
The development of the **Online Food Recipe Web Application** follows a structured approach to ensure efficiency, reliability, and maintainability. The chosen paradigm helps streamline the project by defining clear phases while allowing iterative improvements.

Development Model: Adapted Waterfall Model

The **Waterfall Model** is traditionally a linear approach, but for this project, an **iterative feedback mechanism** is incorporated. This helps refine earlier phases based on insights gathered during implementation.

Key Adaptations in the Waterfall Model:

1. **Structured Phase Progression** – Each stage follows a defined sequence, ensuring clarity in execution.
2. **Iterative Refinements** – Feedback loops allow adjustments, especially between testing and coding.
3. **Defined Milestones** – Each stage reaches completion before moving to the next phase.
4. **Flexible Adjustments** – Overlapping is permitted when necessary to enhance efficiency.



Phases of Development

### Requirement Analysis & System Study

* + Identifying project goals, challenges, and functional specifications.
  + Gathering stakeholder requirements and defining core functionalities.

### System Design

* + Structuring the **database, modules, and architecture**.
  + Designing **user interfaces** for optimal accessibility.

### Implementation (Coding)

* + Backend development using **Node JS**
  + Frontend design using **HTML, CSS, React JS**.
  + Database integration with **MySQL**.

### Testing & Debugging

* + Unit testing, integration testing, and usability checks.
  + Debugging for performance improvements.

### Deployment & Maintenance

* + Hosting on a scalable environment.
  + Continuous updates for feature enhancements.
  1. Data Flow Diagram:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

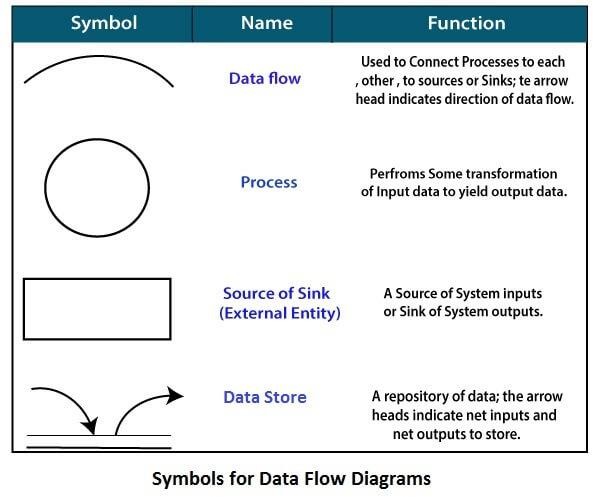
It shows how data enter and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

**The following observations about DFDs are essential:**

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

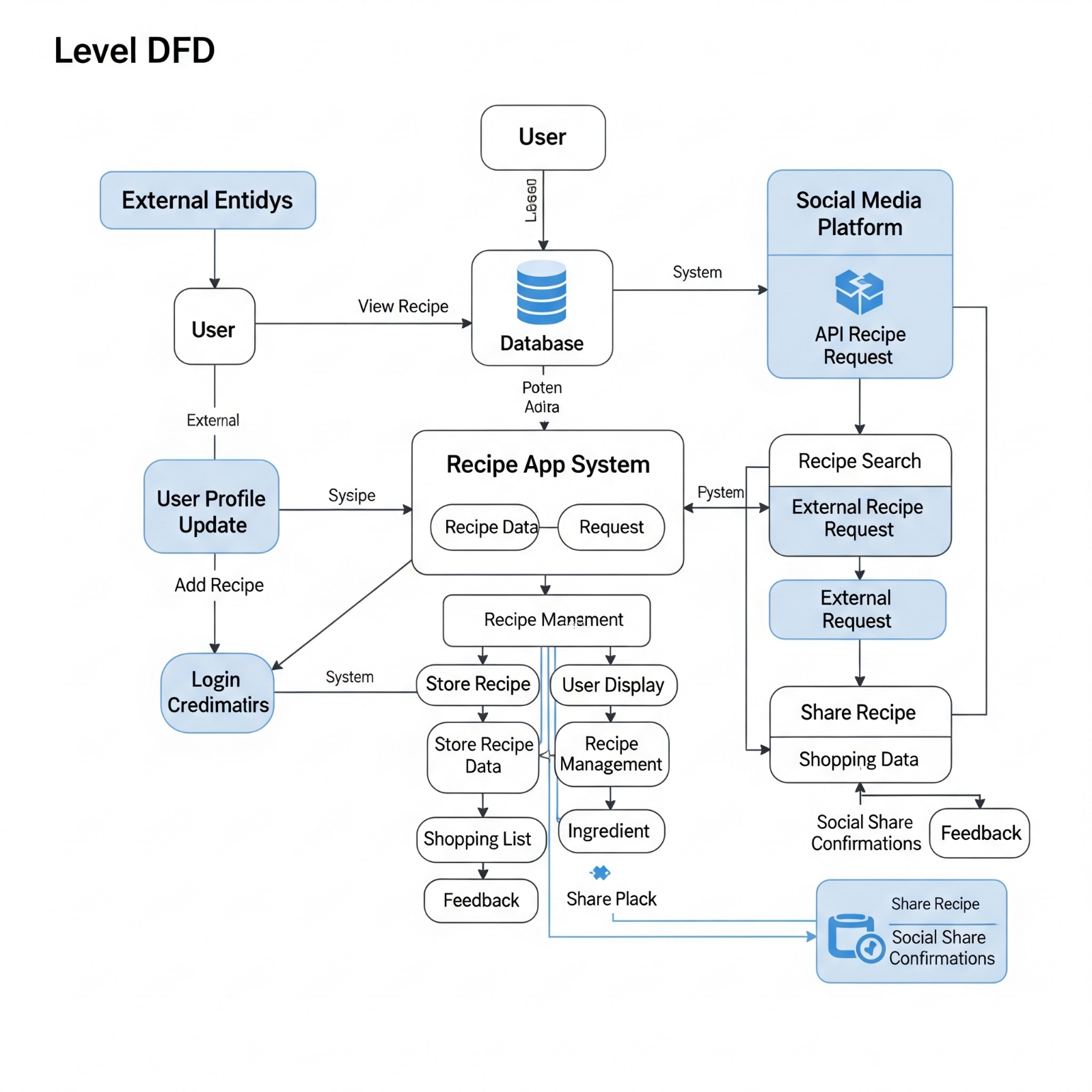


**Circle:** A circle (bubble) shows a process that transforms data inputs into data outputs.

**Data Flow:** A curved line shows the flow of data into or out of a process or data store.

**Data Store:** A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

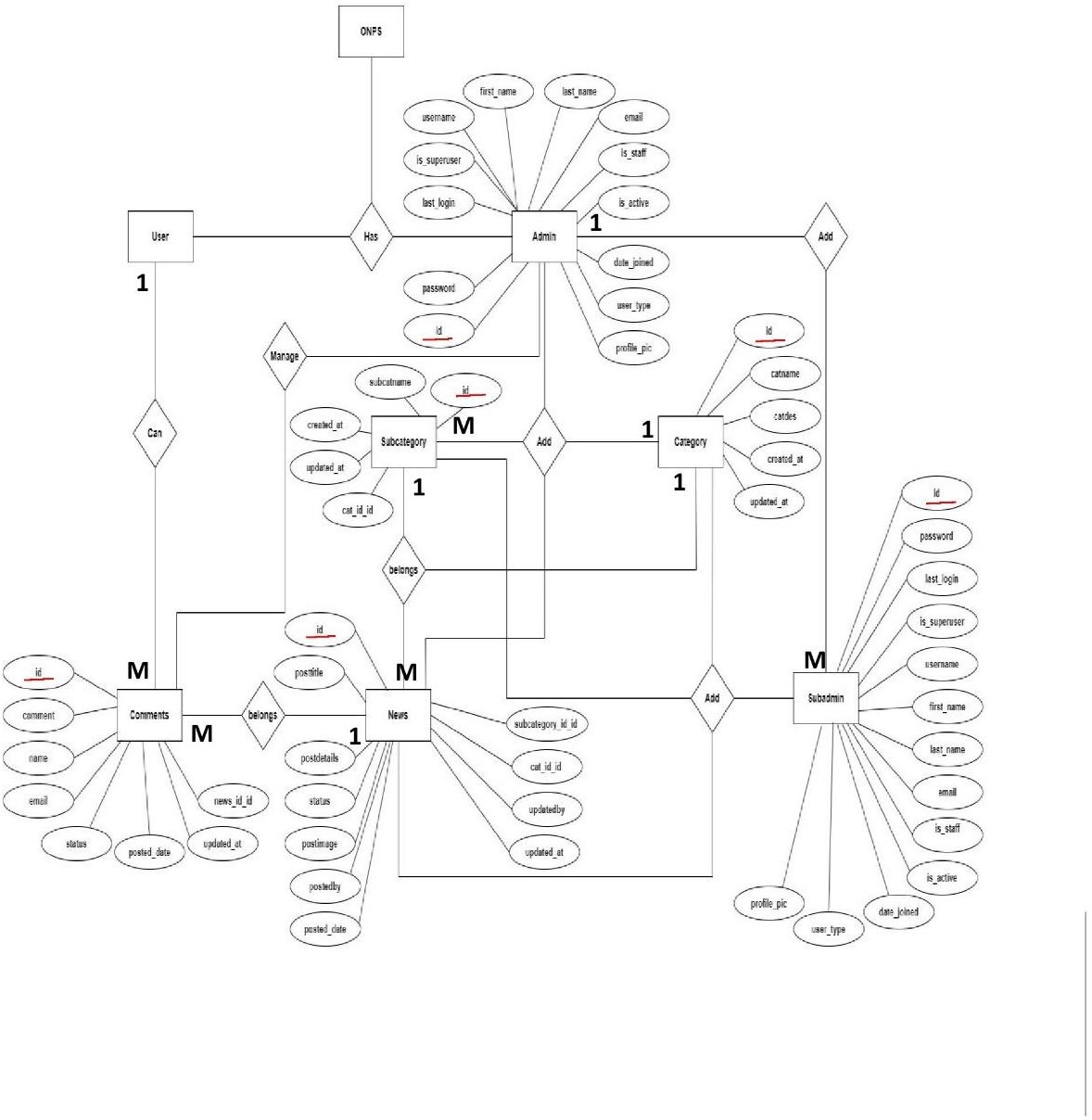
**Source or Sink**: Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.



The **Zero Level DFD** of the Online News Portal System (ONPS) shows the main processes interacting with the system, such as Admin, Sub-Admin, Category, Subcategory, News, Comments, Login, Password, Authorization, and Webpage Management. It highlights the flow of data between the central system (ONPS) and these modules, representing how each core function is managed within the portal.

ER diagram

The **ER Diagram** of the **Online News Portal System** shows key relationships between **Users, Admins, News Articles, Categories, Comments, and Sentiment Analysis**, ensuring structured data management and insightful moderation.



1. System design
   1. Module
      * The system design defines the architecture, components, and interactions of the **Food Recipe App**. It ensures that the application is efficient, scalable, and easy to use. Read news articles categorized into Politics, Business, Technology, Sports, Entertainment, etc.

**User Module**

* User Registration and Login
* Profile management
* Secure authentication

### ****Recipe Module****

* Browse recipes by category (cuisine, meal type, diet)
* Search and filter recipes
* View recipe details (ingredients, steps, cooking time)

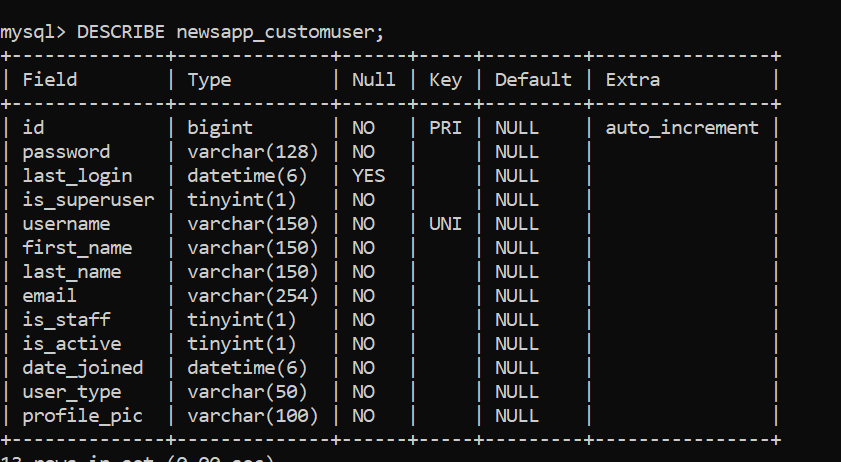
### ****Admin Module (Optional / Future Scope)****

* Add, update, or delete recipes
* Manage recipe categories
* Monitor user activity
  1. DATA STRUCTURE OF ALL MODULES:

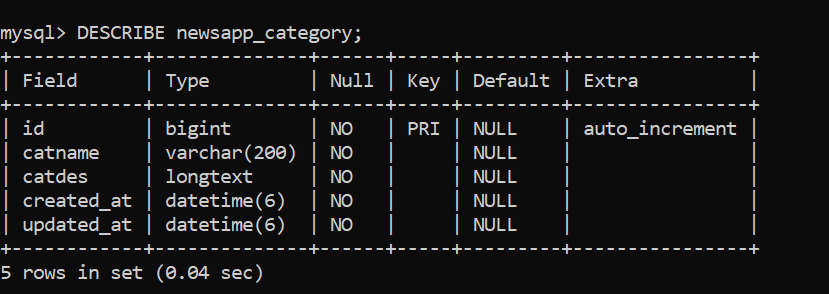
We have organized one database **Food Recipes** for system design. It can be accessed directly or sequentially by registered. The database determines files, record, fields, and characters. It can be easily controlled and updated. **“Online Food Recipe Web Application” contains 15 MySQL tables(In this MySQL 6 table is customized and 9 table) :**

### Customized Tables Details

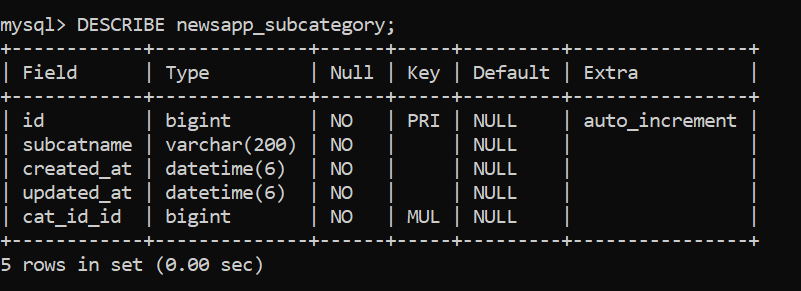
**Admin and Sub-Admin Table:**(newsapp\_customuser) This store admin personal and login details.



**Category Table: Table name(**newsapp\_category) This table store the details of category of news.

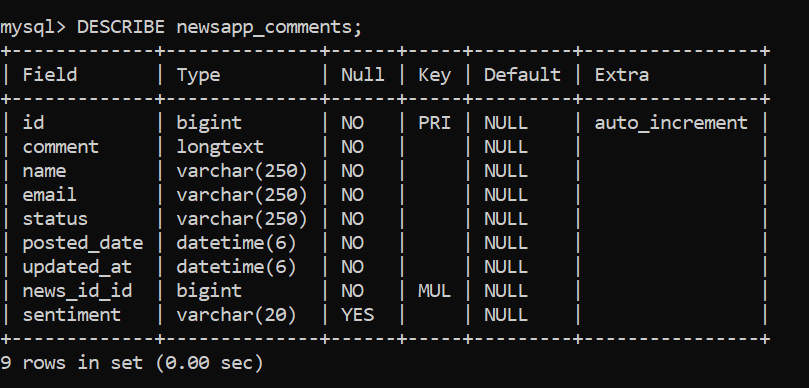


**Subcategory Table: Table name(**newsapp\_subcategory) This table store sub-category of news.



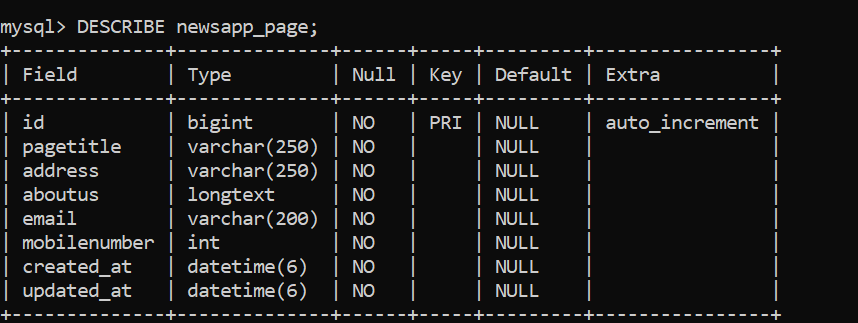
**Comments Table: Table name(**newsapp\_comments)

This table store the comments details which is given by news reader.

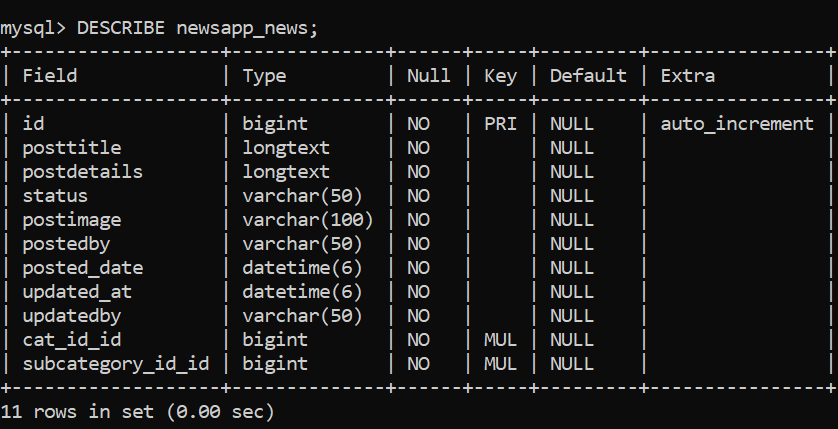


**Website Setting Table: Table name(**newsapp\_page)

This table store the website details which is manage by admin.

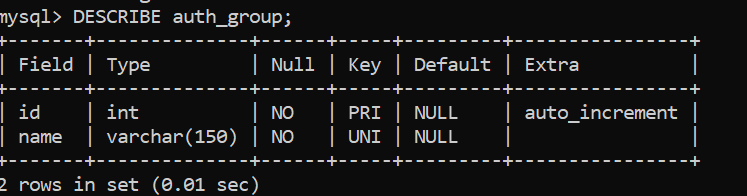


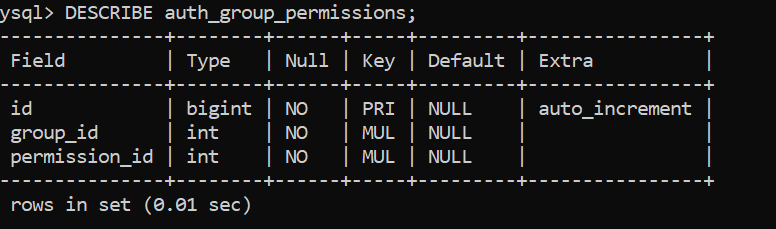
**News Table: Table name(**newsapp\_news) This table store art products details.



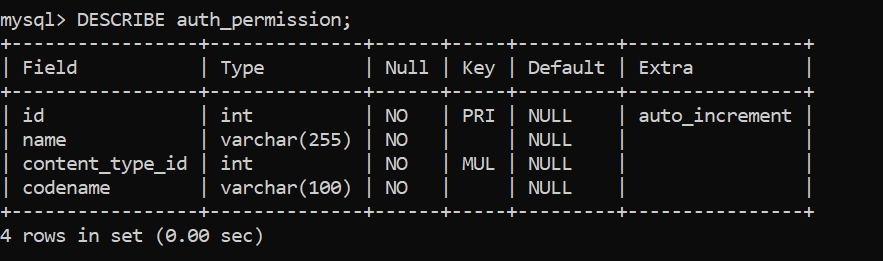
### Default Tables Details

**auth\_group table Structure :** This table in Django is part of the Django authentication system and is used to represent groups of users.

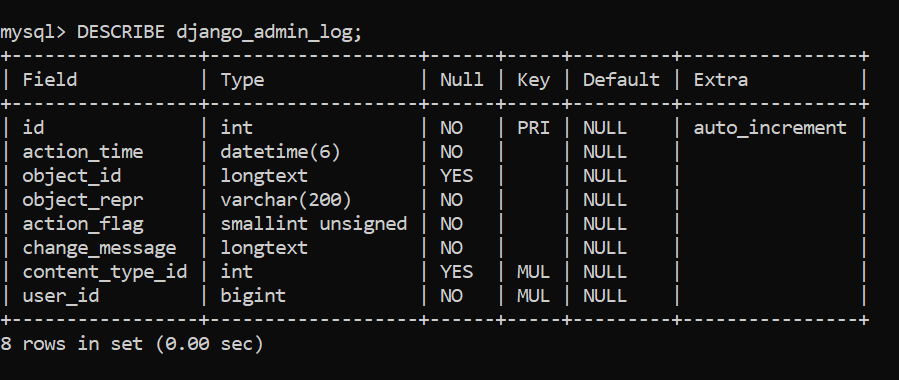


**auth\_group\_permissions table Structure :** This table in Django is a part of the permission system and is used to manage the relationship between user groups and permissions.

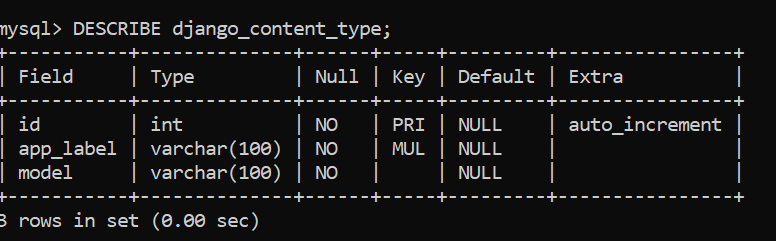
**auth\_permission table Structure :** This table in Django is used to store information about the permissions defined.



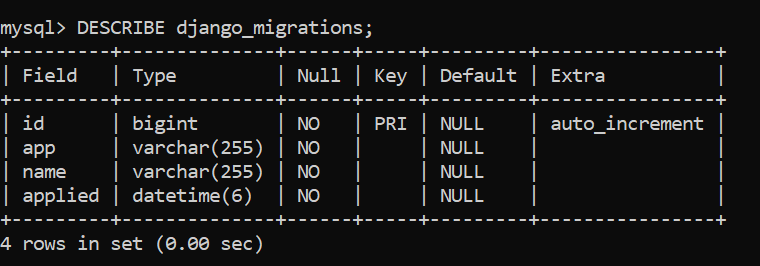
**django\_admin\_log table Structure :** This table is used to store records of actions taken by administrators or users through the Django admin interface.



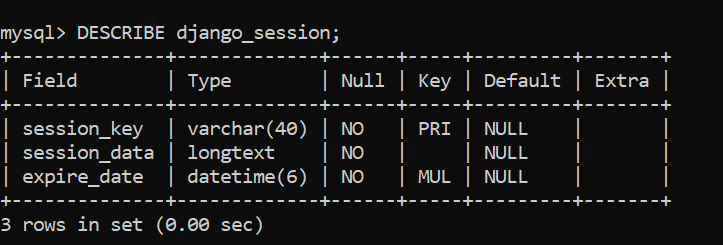
**django\_content\_type table Structure :** This table in Django is a system table that is used to store information about each model (database table) in your Django project.



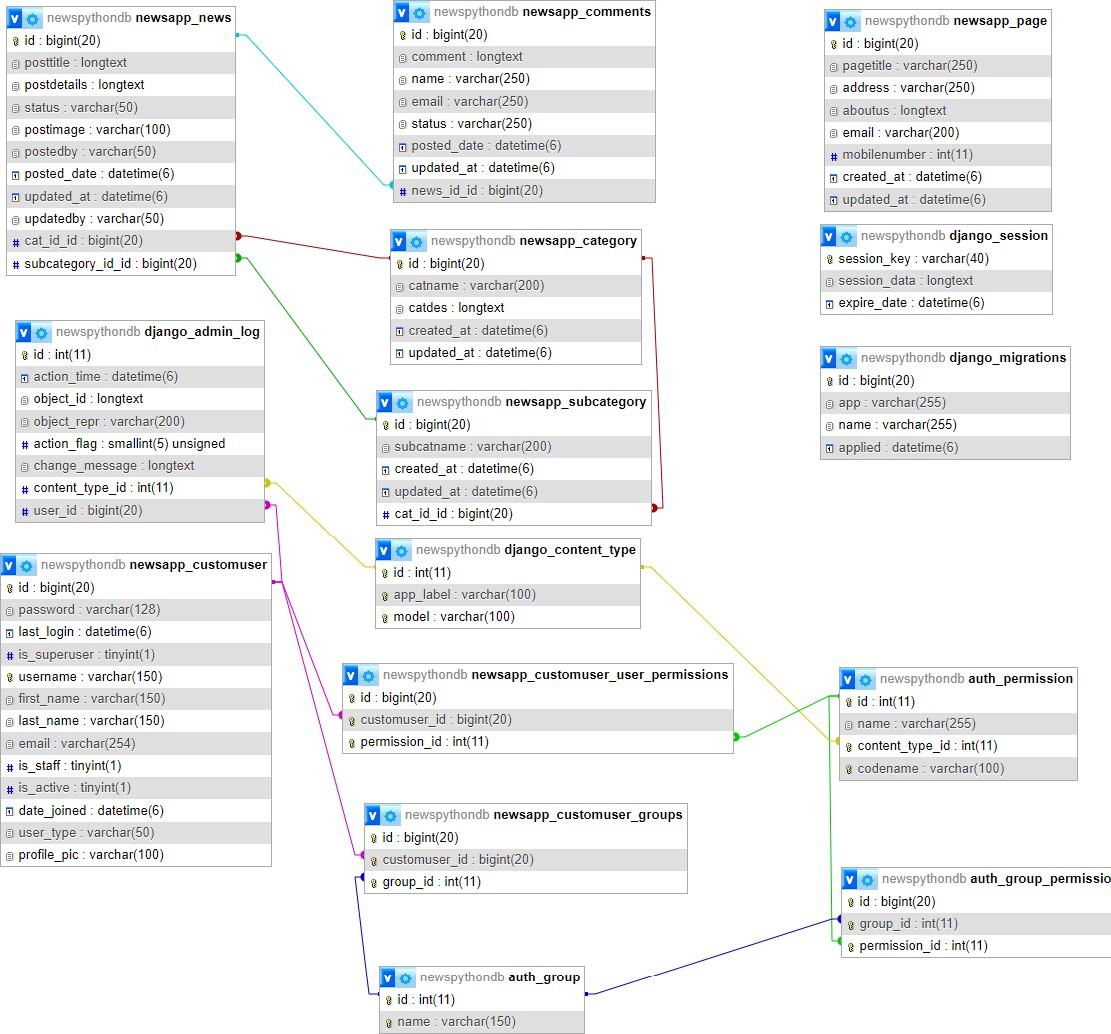
**django\_migrations table Structure :** This table table is part of the database schema and is used to keep track of which migrations have been applied to the database.



**django\_session table Structure :** This table store the session data for user.



## Relationship between tables (Class Diagram)



* 1. PROCEDURAL DESIGN:

Process logic (flowchart ) of each module

## User Panel Design

The **User Panel** is designed to provide an easy and interactive interface for users to explore and manage recipes. It includes the following components:

### ****Features:****

1. **Login & Registration** – Secure authentication for new and existing users.
2. **Dashboard/Homepage** – Displays popular and recommended recipes.
3. **Recipe Browsing** – View recipes by categories such as cuisine, meal type, and diet.
4. **Search & Filter** – Quickly find recipes based on ingredients or keywords.
5. **Recipe Details Page** – Step-by-step cooking instructions with ingredients.
6. **Bookmark/Favorites** – Save and manage preferred recipes.
7. **User Profile** – Manage personal details and view saved recipes.

### ****Design Goals:****

* Simple and user-friendly navigation.
* Responsive interface for desktop and mobile.
* Personalized experience with bookmarks and recommendations.

## Admin Panel Design

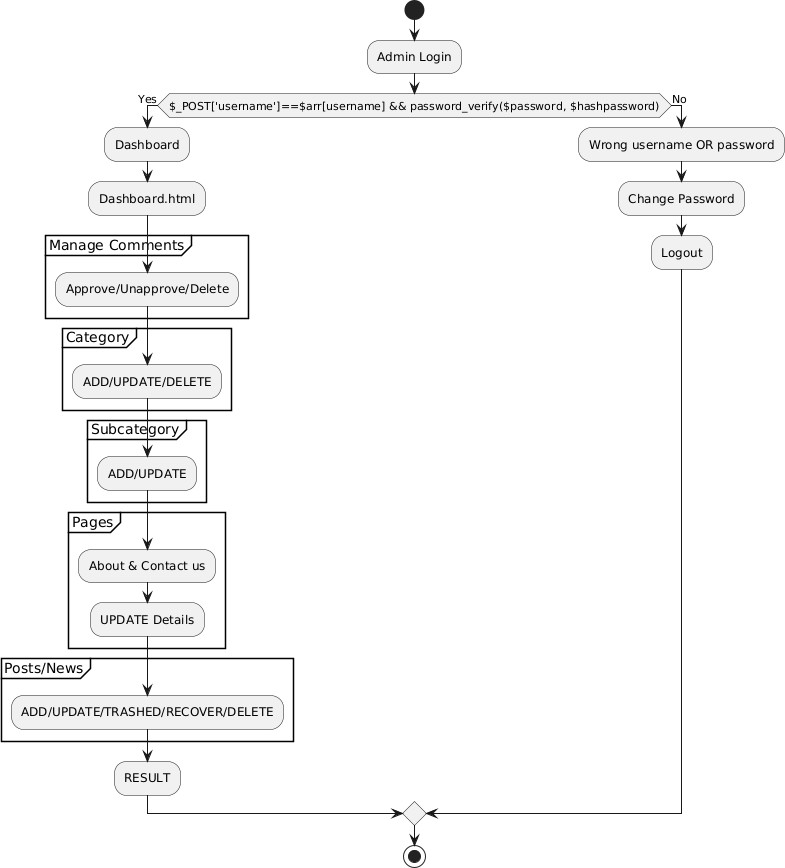
The **Admin Panel** is designed to manage the overall functioning of the Food Recipe App. It provides tools for handling recipes, users, and categories.

### ****Features:****

1. **Admin Login** – Secure access for administrators.
2. **Recipe Management** – Add, edit, update, or delete recipes.
3. **Category Management** – Create and manage recipe categories (e.g., Indian, Italian, Vegan).
4. **User Management** – View and manage registered users.
5. **Bookmark/Feedback Monitoring** – Track user interactions and saved recipes.
6. **Dashboard** – Overview of recipes, users, and activity statistics.

### ****Design Goals:****

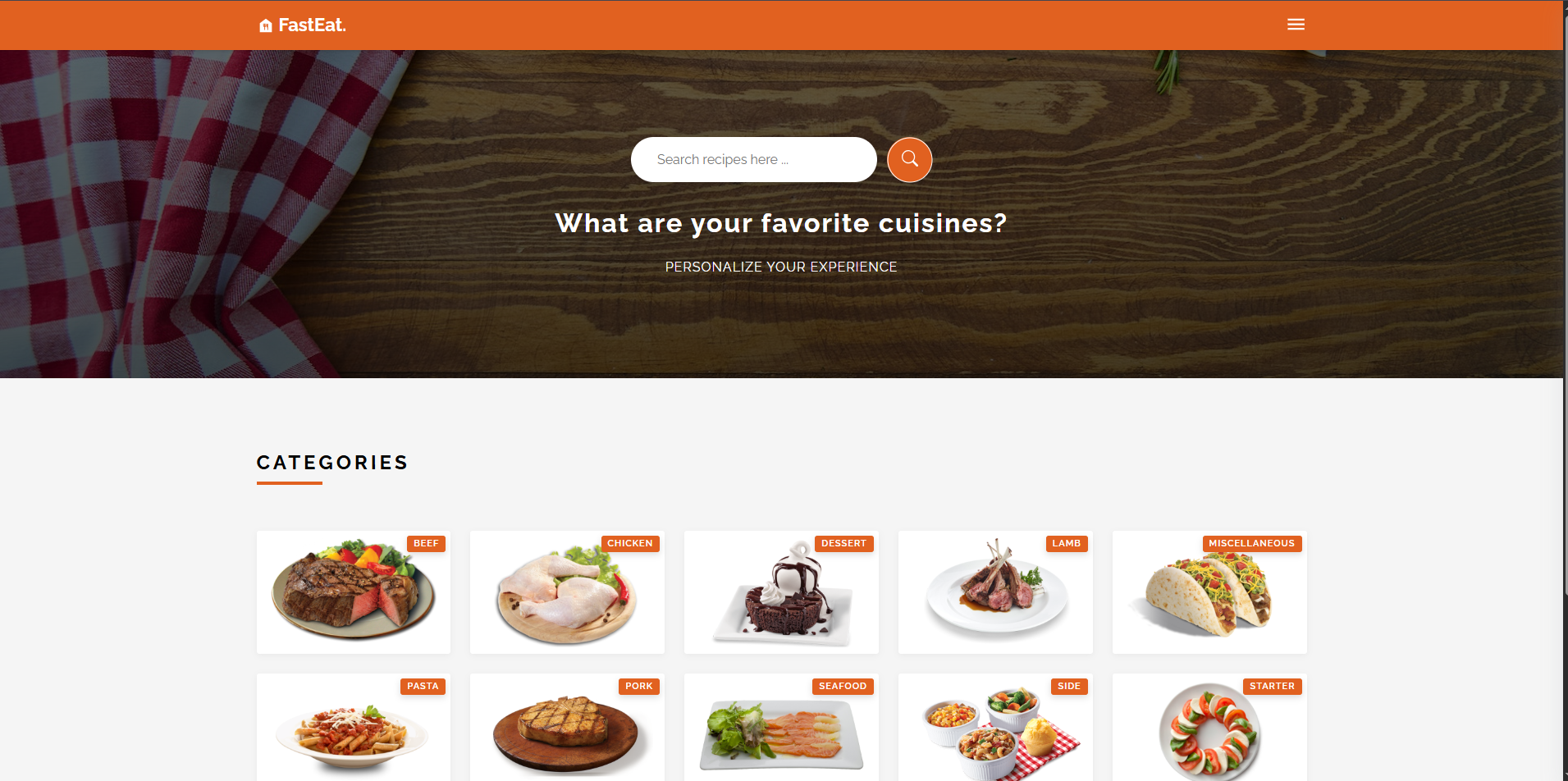
* Provide easy recipe and user management.
* Ensure system security with admin-only access.
* Maintain an organized and scalable recipe database.

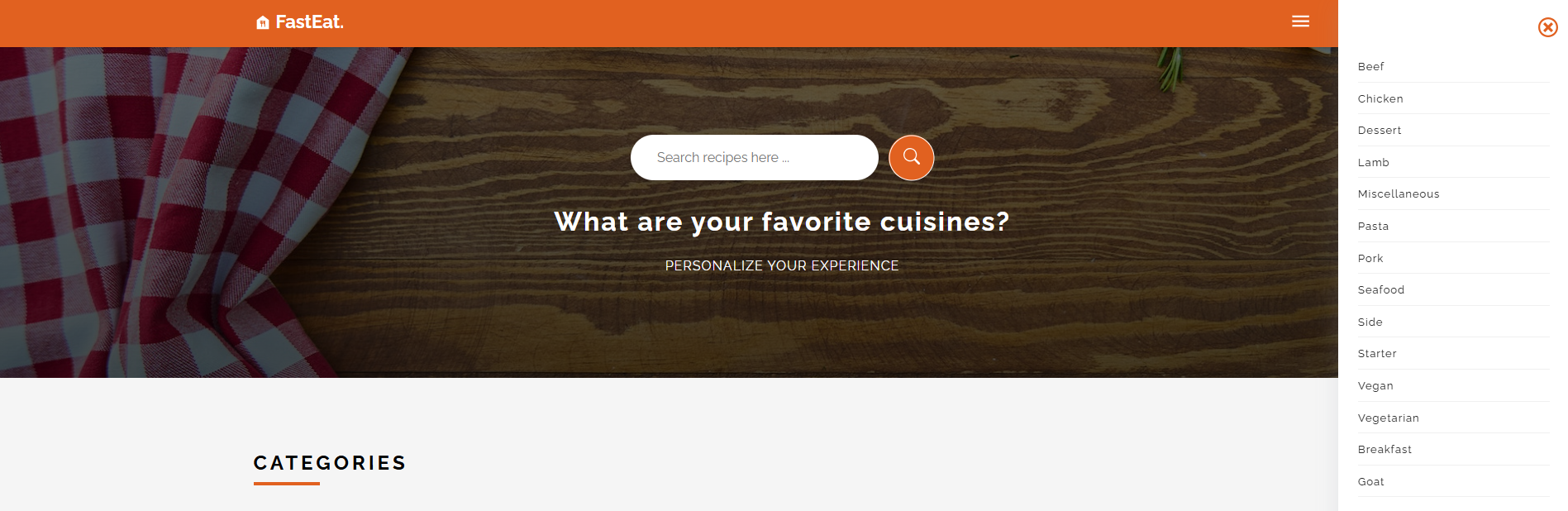


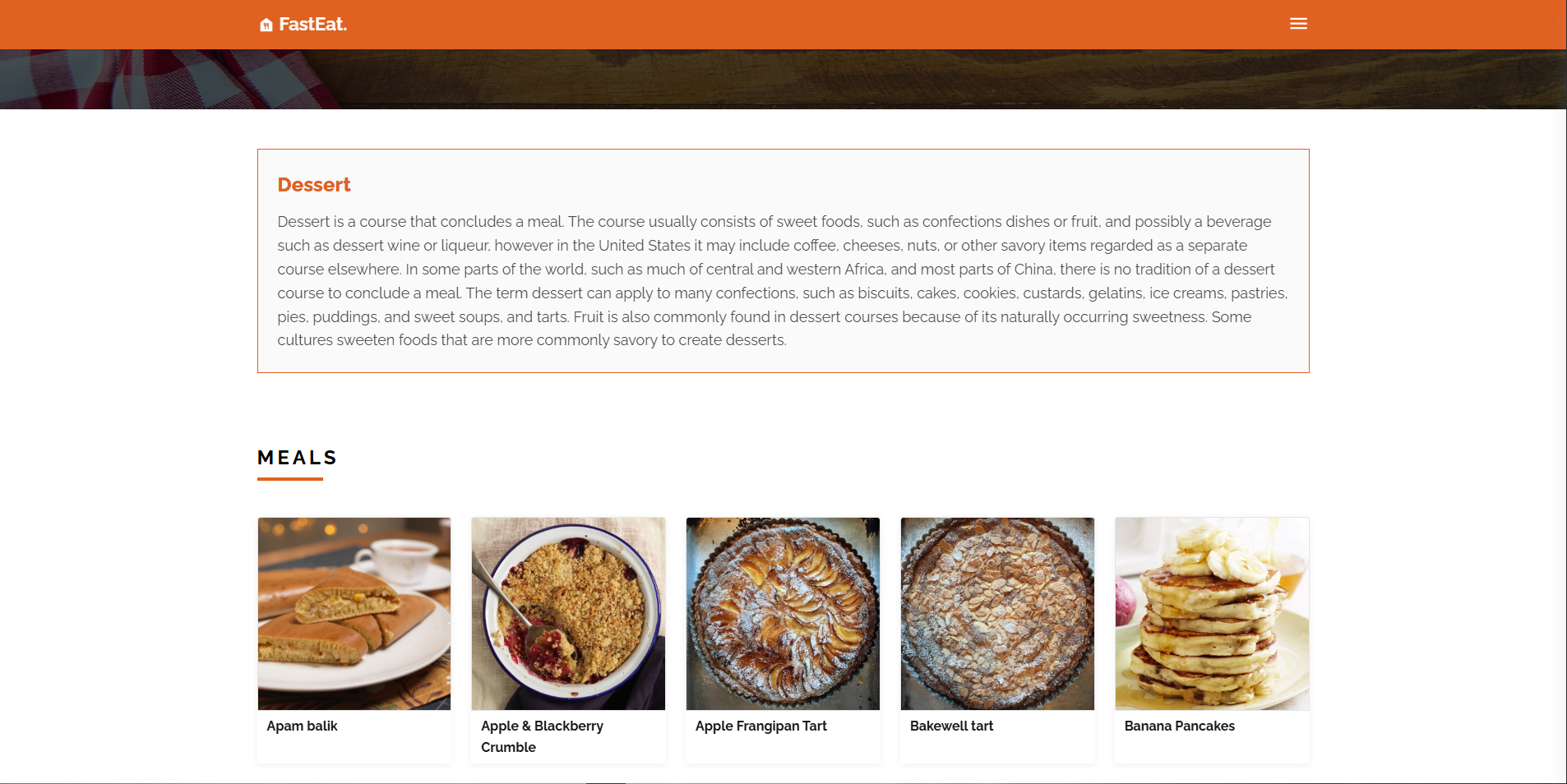
**Fig. 4.2: Admin Login Flow Chart.**

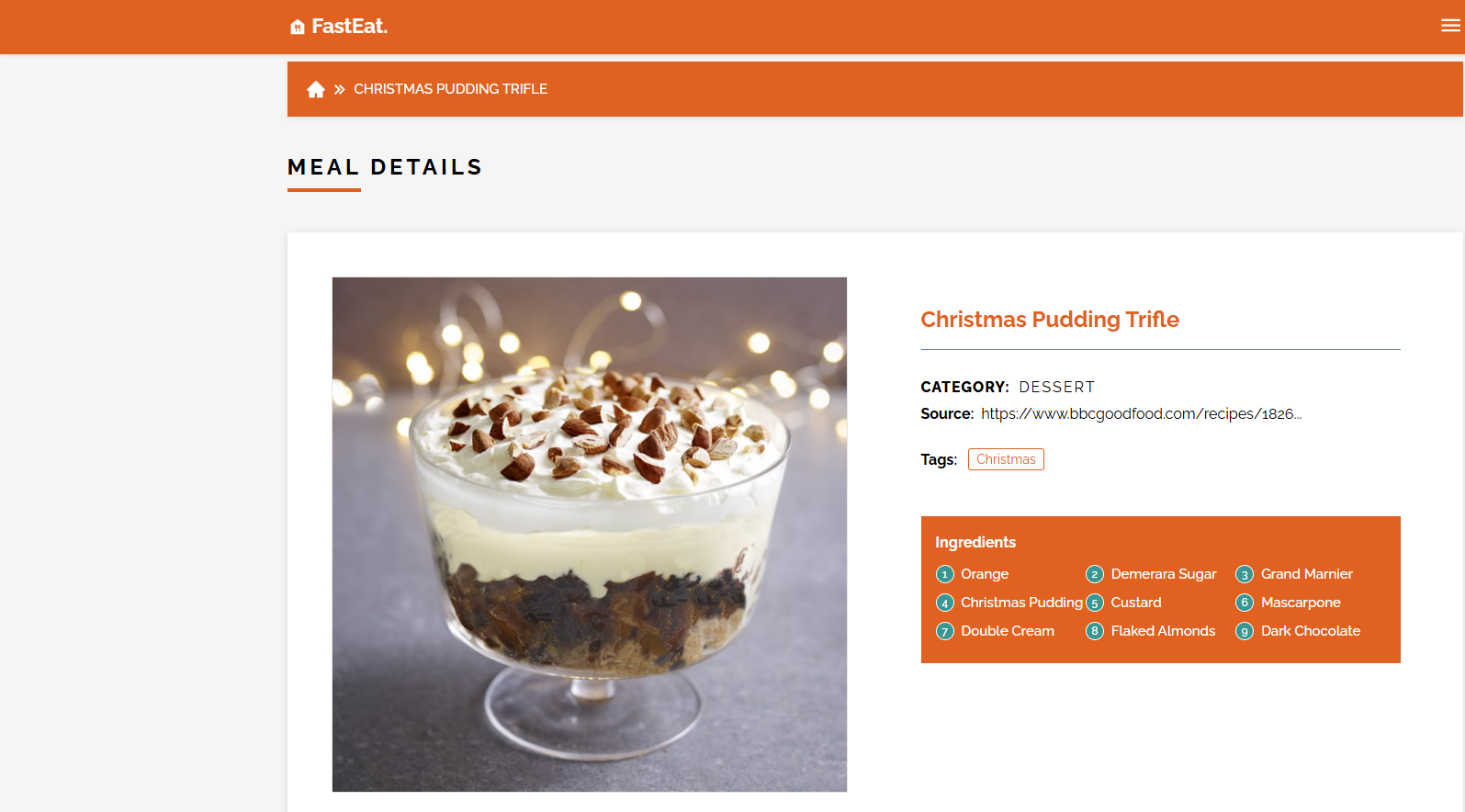
**SCREENSHOTS**

**Home Page**

****

****

****

****

****

Coding

App.js

import './App.scss';

// react router dom

import {

BrowserRouter,

Routes,

Route

} from "react-router-dom";

// pages

import { Home, MealDetails, Error, Category } from "./pages/index";

// components

import Header from "./components/Header/Header";

import Sidebar from "./components/Sidebar/Sidebar";

function App() {

return (

<BrowserRouter>

<Header />

<Sidebar />

<Routes>

<Route path = "/" element = {<Home />} />

<Route path = "/meal/:id" element = {<MealDetails />} />

<Route path = "/meal/category/:name" element = {<Category />} />

<Route path = "\*" element = {<Error />} />

</Routes>

</BrowserRouter>

);

}

export default App;

Auth.js

import React, { useState } from "react";

import axios from "axios";

const API\_URL = "http://localhost:5000"; // backend server

export default function Auth() {

const [registerData, setRegisterData] = useState({

username: "",

email: "",

password: "",

});

const [loginData, setLoginData] = useState({

email: "",

password: "",

});

const handleRegisterChange = (e) => {

setRegisterData({ ...registerData, [e.target.name]: e.target.value });

};

const handleLoginChange = (e) => {

setLoginData({ ...loginData, [e.target.name]: e.target.value });

};

const register = async () => {

try {

const res = await axios.post(`${API\_URL}/register`, registerData);

alert(res.data.message);

} catch (err) {

alert(err.response?.data?.message || "Registration failed!");

}

};

const login = async () => {

try {

const res = await axios.post(`${API\_URL}/login`, loginData);

alert(res.data.message);

} catch (err) {

alert(err.response?.data?.message || "Login failed!");

}

};

return (

<div className="flex flex-col items-center justify-center min-h-screen bg-gray-100">

{/\* Register Box \*/}

<div className="bg-white p-6 rounded-2xl shadow-md w-80 mb-6">

<h2 className="text-xl font-bold mb-4">Register</h2>

<input

type="text"

name="username"

placeholder="Username"

value={registerData.username}

onChange={handleRegisterChange}

className="w-full mb-3 p-2 border rounded"

/>

<input

type="email"

name="email"

placeholder="Email"

value={registerData.email}

onChange={handleRegisterChange}

className="w-full mb-3 p-2 border rounded"

/>

<input

type="password"

name="password"

placeholder="Password"

value={registerData.password}

onChange={handleRegisterChange}

className="w-full mb-3 p-2 border rounded"

/>

<button

onClick={register}

className="w-full bg-blue-600 text-white py-2 rounded hover:bg-blue-700"

>

Register

</button>

</div>

{/\* Login Box \*/}

<div className="bg-white p-6 rounded-2xl shadow-md w-80">

<h2 className="text-xl font-bold mb-4">Login</h2>

<input

type="email"

name="email"

placeholder="Email"

value={loginData.email}

onChange={handleLoginChange}

className="w-full mb-3 p-2 border rounded"

/>

<input

type="password"

name="password"

placeholder="Password"

value={loginData.password}

onChange={handleLoginChange}

className="w-full mb-3 p-2 border rounded"

/>

<button

onClick={login}

className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

>

Login

</button>

</div>

</div>

);

}

Index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

import { SidebarProvider } from './context/sidebarContext';

import { MealProvider } from './context/mealContext';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<SidebarProvider>

<MealProvider>

<App />

</MealProvider>

</SidebarProvider>

);

Mealaction.js

import axios from "../api/axios";

import {

FETCH\_CATEGORY\_BEGIN,

FETCH\_CATEGORY\_ERROR,

FETCH\_CATEGORY\_MEALS\_BEGIN,

FETCH\_CATEGORY\_MEALS\_ERROR,

FETCH\_CATEGORY\_MEALS\_SUCCESS,

FETCH\_CATEGORY\_SUCCESS,

FETCH\_MEALS\_BEGIN,

FETCH\_MEALS\_ERROR,

FETCH\_MEALS\_SUCCESS,

FETCH\_SINGLE\_MEAL\_BEGIN,

FETCH\_SINGLE\_MEAL\_ERROR,

FETCH\_SINGLE\_MEAL\_SUCCESS

} from "./actions";

import { CATEGORIES\_URL, MEAL\_CATEGORIES\_URL, MEAL\_SINGLE\_URL, SEARCH\_URL } from "../utils/constants";

export const startFetchCategories = async(dispatch) => {

try{

dispatch({ type: FETCH\_CATEGORY\_BEGIN});

const response = await axios.get(`${CATEGORIES\_URL}`);

dispatch({type: FETCH\_CATEGORY\_SUCCESS, payload: response.data.categories});

} catch(error){

dispatch({ type: FETCH\_CATEGORY\_ERROR, payload: error.message});

}

}

export const startFetchSingleMeal = async(dispatch, id) => {

try{

dispatch({ type: FETCH\_SINGLE\_MEAL\_BEGIN});

const response = await axios.get(`${MEAL\_SINGLE\_URL}${id}`);

dispatch({type: FETCH\_SINGLE\_MEAL\_SUCCESS, payload: response.data.meals});

} catch(error){

dispatch({ type: FETCH\_SINGLE\_MEAL\_ERROR, payload: error.message});

}

}

export const startFetchMealByCategory = async(dispatch, category) => {

try{

dispatch({type: FETCH\_CATEGORY\_MEALS\_BEGIN});

const response = await axios.get(`${MEAL\_CATEGORIES\_URL}${category}`);

dispatch({type: FETCH\_CATEGORY\_MEALS\_SUCCESS, payload: response.data.meals})

} catch(error){

dispatch({ type: FETCH\_CATEGORY\_MEALS\_ERROR, payload: error.message});

}

}

export const startFetchMealsBySearch = async(dispatch, searchTerm) => {

try{

dispatch({ type: FETCH\_MEALS\_BEGIN});

const response = await axios.get(`${SEARCH\_URL}${searchTerm}`);

dispatch({ type: FETCH\_MEALS\_SUCCESS, payload: response.data.meals});

} catch(error){

dispatch({type: FETCH\_MEALS\_ERROR, payload: error.message});

}

}

Categories.js

import React from 'react';

import { Link } from 'react-router-dom';

import "./Category.scss";

const CategoryList = ({categories}) => {

return (

<div className='section-wrapper bg-whitesmoke'>

<div className='container'>

<div className='sc-title'>categories</div>

<section className='sc-category grid'>

{

categories.map(category => {

const { idCategory: id, strCategory: title, strCategoryThumb: thumbnail} = category;

return (

<Link to = {`/meal/category/${title}`} className = "category-itm align-center justify-center" key = {id}>

<div className='category-itm-img h-100 w-100 flex align-center justify-center'>

<img src = {thumbnail} alt = {title} />

<div className='category-itm-title bg-orange'>

<h3 className='text-white fs-11 fw-6 ls-1 text-uppercase'>{title}</h3>

</div>

</div>

</Link>

)

})

}

</section>

</div>

</div>

)

}

export default CategoryList

MealContext.js

import React, {createContext, useContext, useEffect, useReducer} from "react";

import { mealReducer } from "../reducers/mealReducer";

import { startFetchCategories } from "../actions/mealsActions";

const initialState = {

categories: [],

categoryLoading: false,

categoryError: false,

categoryMeals: [],

categoryMealsLoading: false,

categoryMealsError: false,

meals: [],

mealsLoading: false,

mealsError: false,

meal: [],

mealLoading: false,

mealError: false

}

const MealContext = createContext({});

export const MealProvider = ({children}) => {

const [state, dispatch] = useReducer(mealReducer, initialState);

useEffect(() => {

startFetchCategories(dispatch);

}, []);

return (

<MealContext.Provider value = {{

...state,

dispatch,

startFetchCategories

}}>

{children}

</MealContext.Provider>

)

}

export const useMealContext = () => {

return useContext(MealContext);

}

Testing

**Unit Testing:** Unit testing where individual program units or object classes are tested. Here by using this testing we have focused on testing the functionality of methods.

**Module Testing:** Where this is the combination of unit program is called module. Here we tested the unit program (5-6 programs) is where the module programs have dependency.

**Sub-system Testing:** Then we combined some module for the Preliminary System Testing in our Project.

**System Testing:** Where it is the combination of two or more sub-system and then it is tested. Here we tested the Entire system as per the requirements.

**Acceptance Testing:** Normally this type of testing is done to verify if system meets the customer specified requirements. After submitting this project to User then they tested it and to determine whether to accept application. It is the system testing performed by the customer(s) to determine whether they should accept the delivery of the system

# 6. Future scope

The **Food Recipe App** can be further enhanced with additional features and improvements to increase usability and user engagement. Some possible future enhancements include:

1. **Recipe Upload by Users** – Allow users to share their own recipes with the community.
2. **AI-based Recommendations** – Suggest recipes based on user preferences, history, and available ingredients.
3. **Meal Planner** – Weekly or monthly meal planning with automated shopping lists.
4. **Nutrition Tracking** – Display nutritional values (calories, proteins, fats, etc.) for each recipe.
5. **Multimedia Support** – Add recipe videos and step-by-step image guides.
6. **Social Integration** – Share recipes on social media or within the app community.
7. **Mobile App Version** – Launch Android and iOS apps for broader accessibility.
8. **Voice Assistant Integration** – Step-by-step cooking guidance via voice commands.

**Conclusion**

The **Food Recipe App** is a simple yet powerful platform that brings together a wide range of recipes in one place. It eliminates the limitations of traditional cooking resources by providing features such as secure login, recipe categorization, step-by-step instructions, search and filter options, and bookmarking. The use of **React.js, Node.js, Express.js, and MySQL** ensures a responsive, secure, and scalable system.

This project not only simplifies the process of discovering and preparing meals but also enhances the overall cooking experience for both beginners and experienced users. With potential future enhancements like AI-based recommendations, meal planning, and mobile app integration, the Food Recipe App can grow into a comprehensive digital cooking assistant.

# REFERENCES

## ****References****

1. **Tailwind CSS Documentation** – [https://tailwindcss.com/docs](https://tailwindcss.com/docs?utm_source=chatgpt.com)
2. **React Documentation** – https://react.dev
3. **React DOM Documentation** – https://react.dev/reference/react-dom
4. **Node.js Documentation** – https://nodejs.org/en/docs
5. **MySQL Documentation** – https://dev.mysql.com/doc

## ****API Reference****

* **TheMealDB API** – https://www.themealdb.com/api.php