

FOOD MANIA WEBSITE

Projected by:

Name

Rohit Vidyadhar Goythale

dse21123035@git-india.edu.in

Om Bhushan Tambat

dse21146323@git-india.edu.in

ABSTRACT

This report documents the design, development, and implementation of a dynamic and user-friendly food blog website. In an era where culinary exploration and gastronomic experiences are shared widely, the need for an engaging online platform dedicated to food enthusiasts is paramount. This report outlines the key components, features, and considerations that went into creating a food blog website using modern web development technologies.

The report begins by highlighting the rising popularity of food blogging in the digital age, emphasizing the importance of creating a visually appealing, responsive, and content-rich website to captivate the target audience. It underscores the significance of effective content management, user engagement, and seamless navigation as fundamental principles guiding the website's development.

The core of this report delves into the technical aspects of the website's architecture, detailing the choice of the Spring Boot framework for robust backend development, Thymeleaf for dynamic HTML templates, and Spring Data JPA for efficient data management. It discusses the implementation of user authentication and authorization, allowing content creators and readers to interact securely.

Additionally, the report addresses the database design and management, highlighting the importance of structuring and storing diverse content such as recipes, user profiles, comments, and multimedia elements. It explores the utilization of relational database management systems for efficient data retrieval and storage.

User experience (UX) and user interface (UI) design considerations are also examined in this report, emphasizing the creation of visually appealing, responsive, and intuitive interfaces that enhance user engagement and satisfaction. It discusses the use of HTML, CSS, and JavaScript to craft captivating and user-centric web pages.

Furthermore, the report touches upon content creation and management, including the ability for users to publish and share recipes, comment on posts, and engage in a culinary community. It underscores the importance of user-generated content in enriching the website's appeal.

ACKNOWLEDGEMENT

The realization of this Food Mania Website project using Spring Boot has been made possible through the collective efforts and support of various individuals and resources. As we reflect on the journey that brought this platform to life, we extend our sincere gratitude to those who played a pivotal role in its development and success.

Our foremost acknowledgment goes to the dedicated members of our development team whose expertise, passion, and tireless commitment have been the driving force behind this project. Their collaborative spirit and innovative contributions have transformed the initial idea into a fully functioning and engaging culinary community.

The open-source community also deserves special recognition for its invaluable contributions. We have drawn upon a multitude of open-source libraries, frameworks, and tools that have greatly accelerated the development process. The ethos of open source, characterized by knowledge sharing and collaboration, has been instrumental in shaping this project.

DECLARATION

we here by declaring that the project entitled, "Food Mania Website" has not been any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

TABLE OF CONTENTS

INDEX	PAGE NO
Introduction	6
Features and Functionality of Food Blog Website	7
Database Design	9
Diagrams	11
Design and Implementation of Food Blog website	13
Source Code	16
Conclusion & Future Work	31

INTRODUCTION

In a world increasingly driven by culinary exploration and gastronomic adventures, the creation of a Food Blog Website has emerged as an enticing endeavor. This project embarks on a journey to develop a dynamic and user-friendly platform for food enthusiasts to share their culinary experiences, recipes, and culinary wisdom. Leveraging the power of Spring Boot, a popular Java-based framework known for its simplicity and efficiency, we aim to construct a robust and feature-rich food blog website that caters to both content creators and avid foodies.

The core objective of this project is to provide a seamless and interactive online space where users can not only discover a diverse array of recipes but also actively participate in the community by sharing their culinary innovations. The website will offer an intuitive and aesthetically pleasing user interface that allows users to browse through an extensive collection of recipes, create their profiles, submit their recipes, and engage in discussions through comments and feedback.

One of the key components of this project is the integration of Spring Data JPA to facilitate effortless data storage and retrieval. We will define essential entity classes, such as 'User,' 'Recipe,' and 'Comment,' to model our data schema. Through the power of JPA annotations, we will establish relationships between these entities, enabling smooth and efficient database operations. Furthermore, we will employ Spring Security to ensure the integrity of user data and protect sensitive information, incorporating user authentication and authorization mechanisms.

The website's frontend will be crafted using Thymeleaf templates, offering a dynamic and responsive user interface. Users will be able to interact with the site intuitively, from browsing and searching for recipes to commenting on their favorite dishes. A well-thought-out design will enhance the overall user experience, making it visually appealing and user-friendly.

Features & Functionality of the Food Blog Website

The food Mania website built using Spring Boot can offer a wide range of features and functionality to cater to both content creators and food enthusiasts. Here are some of the key features and functionalities that can be implemented in the project:

1. User Registration and Authentication:

- Users can create accounts and log in securely.
- Password hashing and encryption for data security.
- User roles (e.g., admin, regular user) for different levels of access.

2. User Profiles:

- Users can create and customize their profiles with profile pictures and information.
- View and edit their own profile information.

3. Recipe Management:

- Content creators can create, edit, and delete recipes.
- Add recipe details such as title, description, ingredients, and instructions.
- Upload images for recipes.
- Categorize recipes (e.g., appetizers, main courses, desserts).
- Search and filter recipes by various criteria (e.g., cuisine, dietary preferences).

4. Recipe Viewing:

- Users can browse and search for recipes.
- View detailed recipe information, including ingredients, instructions, and images.
- Leave ratings and reviews for recipes.
- Share recipes on social media platforms.

5. User Interactions:

- Users can follow other users to receive updates on their latest recipes and activities.
- Implement a "like" or "favorite" system for users to bookmark their favorite recipes.

6. Admin Panel:

- An admin dashboard for site administrators to manage users, recipes, and comments.
- The ability to moderate and delete inappropriate content and users.

These features and functionalities can make your food blog website using Spring Boot a comprehensive and engaging platform for both content creators and food enthusiasts, fostering a vibrant online culinary community.

Used Tools and Technologies:

Technology/Domain: Java

Front-end: HTML, CSS

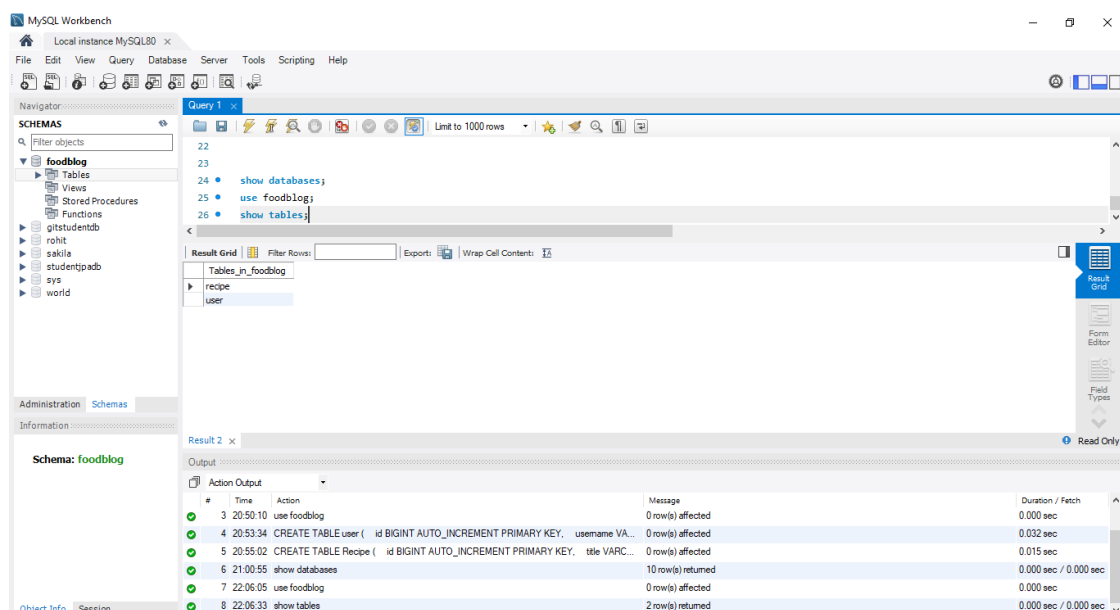
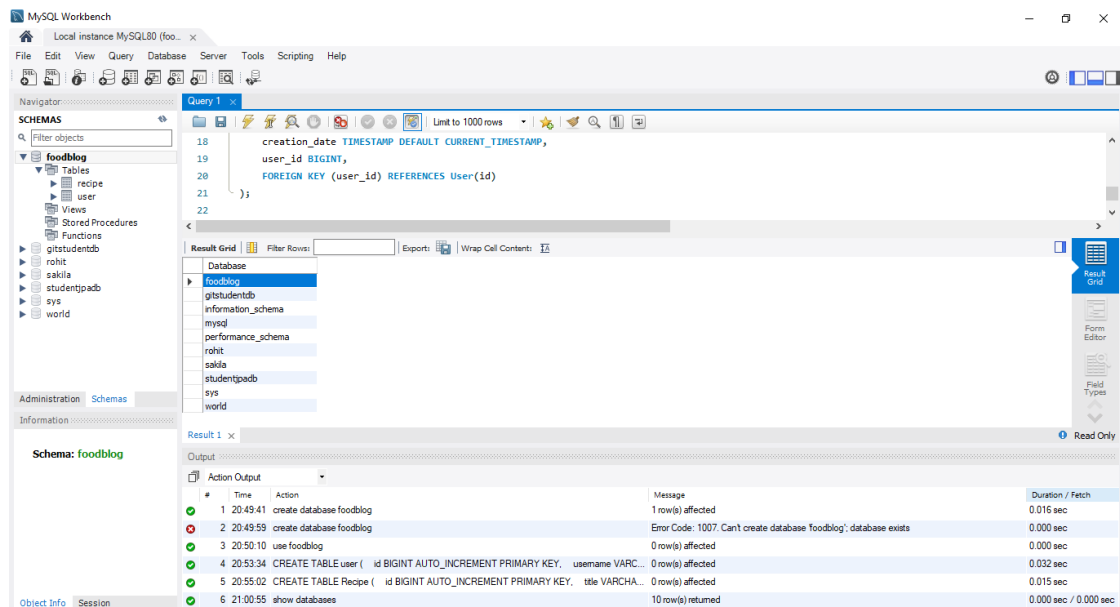
Back-end: Java

Database: MySQL

DATABASE DESIGN

We use MySQL database for this project total table. We describe the function of table below:

DATABASE TABLE: This is the database table of our Food Mania Website. To complete this Food Blog we use two effective tables.



MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator

Filter objects

SCHEMAS

foodblog

Tables

Views

Stored Procedures

Functions

gitstudentdb

rohit

sakila

studenttpadb

sys

world

Administration Schemas

Information

Schema: foodblog

Query 1

```

23
24 • show databases;
25 • use foodblog;
26 • show tables;
27 • select * from user;

```

Result Grid

id	username	email	password	profile_picture	registration_date
1	rohit	rohit@gmail.com	123456	123456	2023-07-22
2	rohit	rohit@gmail.com	123456	123456	2023-07-22
3	rohit	rohit@gmail.com	123456	123456	2023-07-22

Output

#	Time	Action	Message	Duration / Fetch
4	20:53:34	CREATE TABLE user (id BIGINT AUTO_INCREMENT PRIMARY KEY, username VA...	0 row(s) affected	0.032 sec
5	20:55:02	CREATE TABLE Recipe (id BIGINT AUTO_INCREMENT PRIMARY KEY, title VAR...	0 row(s) affected	0.015 sec
6	21:00:55	show databases	10 row(s) returned	0.000 sec / 0.000 sec
7	22:06:05	use foodblog	0 row(s) affected	0.000 sec
8	22:06:33	show tables	2 row(s) returned	0.000 sec / 0.000 sec
9	22:07:03	select * from user LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator

Filter objects

SCHEMAS

foodblog

Tables

Views

Stored Procedures

Functions

gitstudentdb

rohit

sakila

studenttpadb

sys

world

Administration Schemas

Information

Schema: foodblog

Query 4

```

24 • show databases;
25 • use foodblog;
26 • show tables;
27 • select * from user;
28 • select * from recipe;

```

Result Grid

id	title	description	ingredients	instructions	image	creation_date	user_id
1	rohit	rohit@gmail.com	123456	123456	123456	2023-07-22	2023-07-22
2	rohit	rohit@gmail.com	123456	123456	123456	2023-07-22	2023-07-22
3	rohit	rohit@gmail.com	123456	123456	123456	2023-07-22	2023-07-22

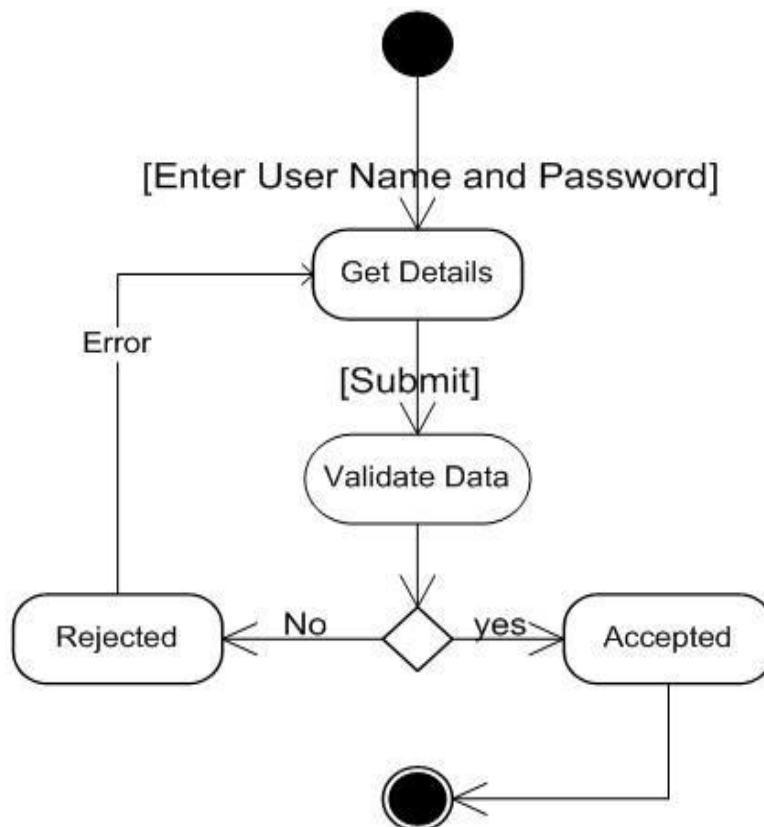
Output

#	Time	Action	Message	Duration / Fetch
5	20:55:02	CREATE TABLE Recipe (id BIGINT AUTO_INCREMENT PRIMARY KEY, title VAR...	0 row(s) affected	0.015 sec
6	21:00:55	show databases	10 row(s) returned	0.000 sec / 0.000 sec
7	22:06:05	use foodblog	0 row(s) affected	0.000 sec
8	22:06:33	show tables	2 row(s) returned	0.000 sec / 0.000 sec
9	22:07:03	select * from user LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
10	22:07:38	select * from recipe LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

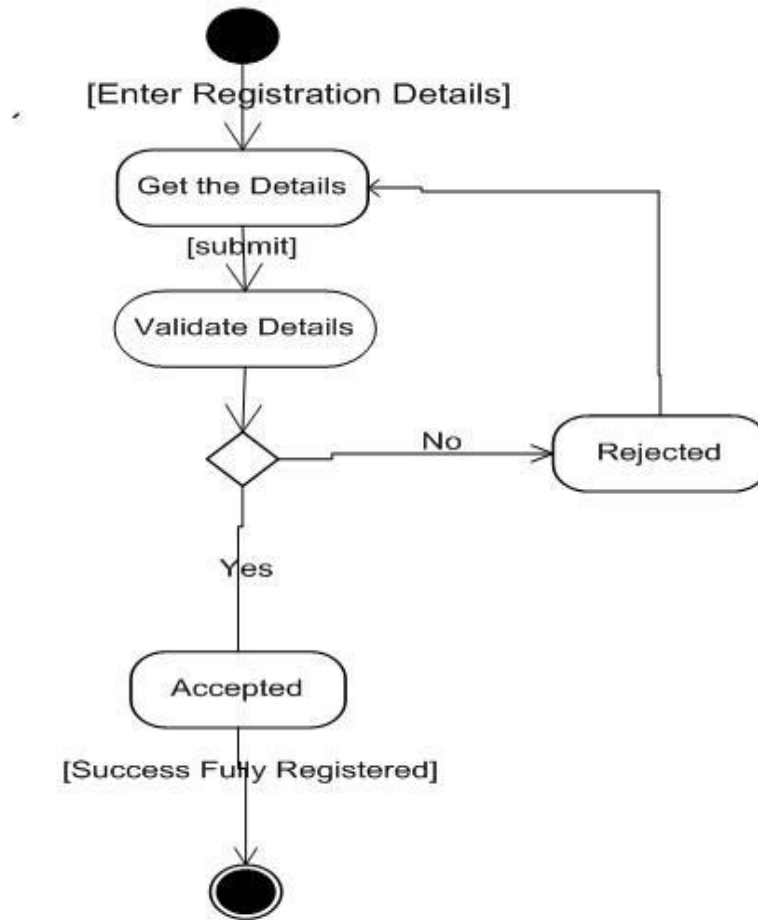
Object Info Session

DIAGRAMS FOR E-COMMERCE WEBSITE PROJECT

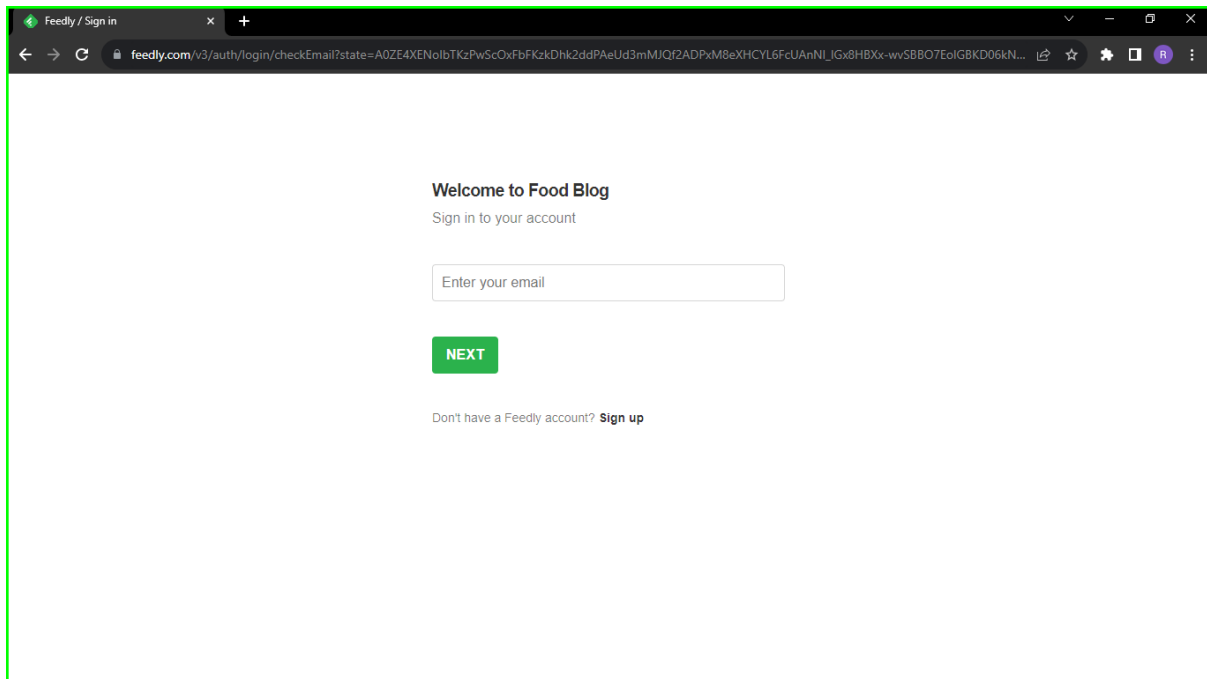
LOGIN ACTIVITY DIAGRAM:



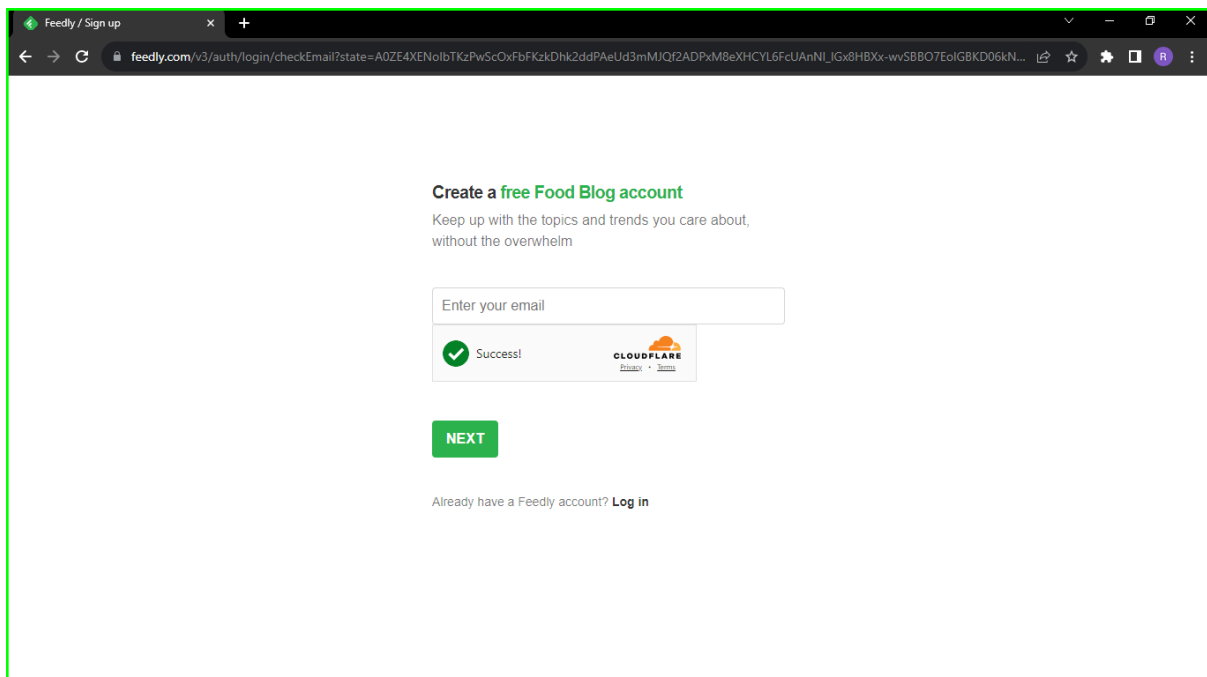
REGISTRATION ACTIVITY DIAGRAM:



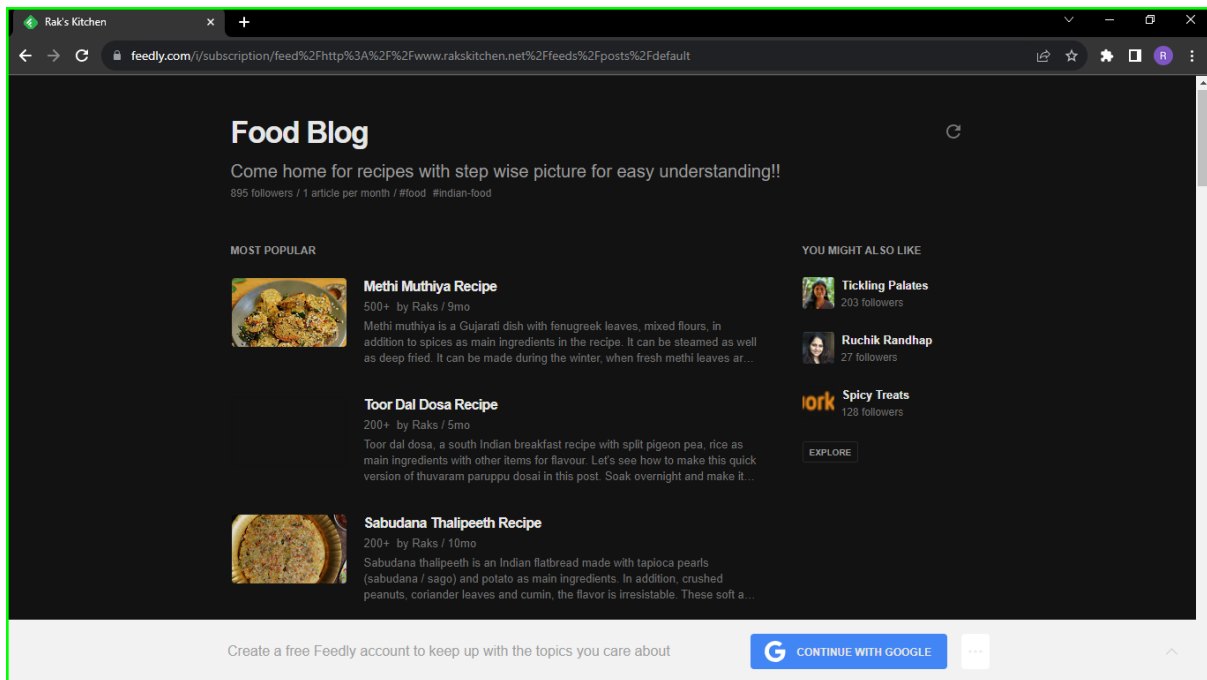
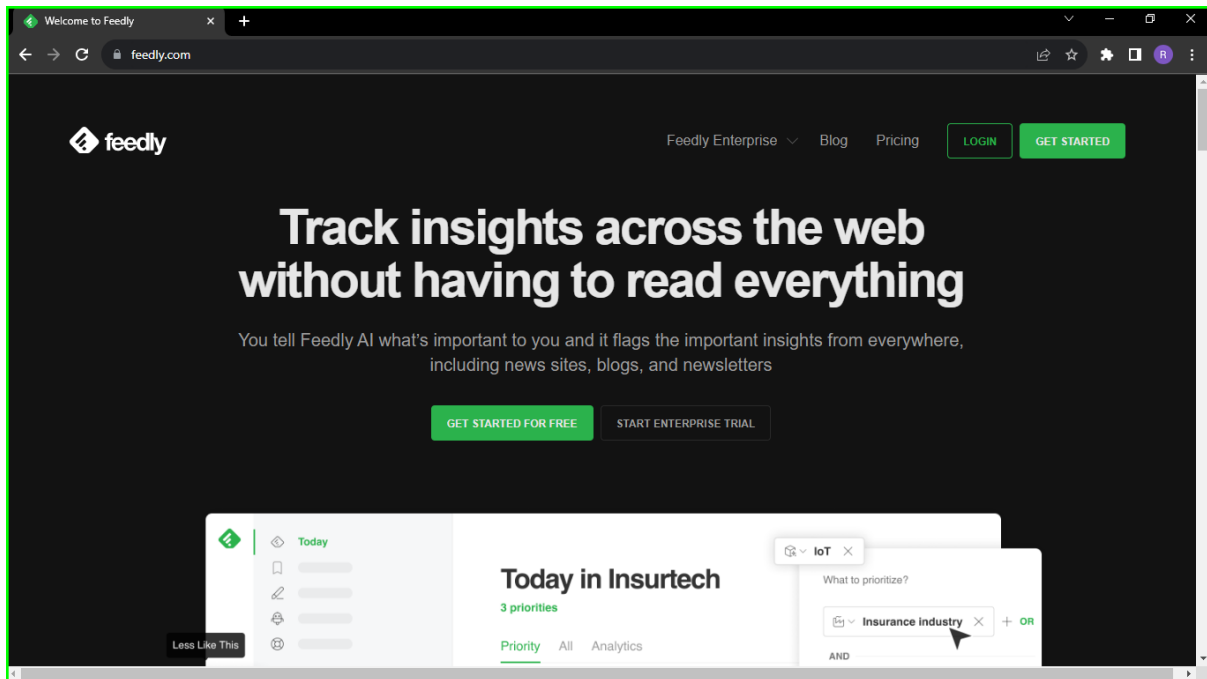
DESIGN AND IMPLEMENTATION OF E-COMMERCE WEBSITE



The screenshot shows a web browser window with the address bar displaying 'feedly.com/v3/auth/login/checkEmail?state=A0ZE4XENolbTKzPwScOxPbFKzkDhk2ddPAeUd3mMIQfZADPxM8eXHCYL6FcUAnNI_JGx8HBXx-wwSBBO7EolGBKD06kN...'. The page title is 'Feedly / Sign in'. The main content area has a heading 'Welcome to Food Blog' followed by the text 'Sign in to your account'. Below this is a text input field with the placeholder 'Enter your email'. A green button labeled 'NEXT' is positioned below the input field. At the bottom, there is a link that reads 'Don't have a Feedly account? Sign up'.



The screenshot shows a web browser window with the same address bar as the previous image. The page title is 'Feedly / Sign up'. The main content area has a heading 'Create a free Food Blog account' followed by the text 'Keep up with the topics and trends you care about, without the overwhelm'. Below this is a text input field with the placeholder 'Enter your email'. A green button labeled 'NEXT' is positioned below the input field. At the bottom, there is a link that reads 'Already have a Feedly account? Log in'. A success message is displayed above the 'NEXT' button, consisting of a green checkmark icon, the text 'Success!', and the Cloudflare logo with the text 'CLOUDFLARE' and 'Privacy Terms'.




Rak's Kitchen

feedly.com/subscription/feed%2Fhttp%3A%2F%2Fwww.rakskitchen.net%2Ffeeds%2Fposts%2Fdefault

Rak's Kitchen

FEB 21



Ragi Chapathi Recipe
by Raks / 7mo

Ragi chapathi is a gluten free flatbread, made with finger millet flour/ ragi flour. Let's see how to make a soft one easily in this post with step by step images and video. If you are looking for diabetic friendly and gluten free sw...

FEB 11

Moringa Tempura Recipe | Eggless Tempura batter
21 by Raks / 8mo

Moringa tempura is crispy as well as tasty snack, I recently tried at a restaurant and wanted to share a simple recipe you can try at home. Learn how to make a perfect and also a light tempura batter, that stays crunchy fo...

FEB 09

Arisi Paruppu Sadam Recipe
87 by Raks / 8mo

Arisi Paruppu Sadam is a basic, go-to everyday meal, popular in Coimbatore and in South Indian cuisine. This one pot meal is made with Ponni parboiled rice, boiled arisi, maida and tomatoes, gets loved by everyone. In this post, let's

Create a free Feedly account to keep up with the topics you care about

CONTINUE WITH GOOGLE


...

Rak's Kitchen

feedly.com/subscription/feed%2Fhttp%3A%2F%2Fwww.rakskitchen.net%2Ffeeds%2Fposts%2Fdefault

Rak's Kitchen

DEC 06 22



Cooking without fire recipes
by Raks / 10mo


that you can make it for competition as well as to involve kids to create the interest in them early. In this post, I have shared some fireless cooking recipes, ideas and in addition, strategies for No-cook recipes using mainly...

DEC 01 22

Bedmi Puri Recipe
21 by Raks / 10mo

Bedmi puri is a crispy deep fried food item from Delhi, made of urad dal, wheat flour and spices as main ingredients. Usually paired with aloo gravy as side dish. Perfect for winter mornings to enjoy hot poori with hot aloo sabji f...

NOV 15 22



Oats chilla recipe
by Raks / 10mo

Oats chilla or oats cheela is an Indian breakfast recipe you can make instantly with ingredients available at home. Perfect for dinner as well as for weight loss. I make oats chilla often for dinner or lunch rather than breakfas...

Create a free Feedly account to keep up with the topics you care about

CONTINUE WITH GOOGLE

...

Source Code:

```
import React, { useState } from 'react'
import { useEffect } from 'react';
import { useAuth } from '../context/AuthContext';
import { useModal } from '../hooks/useModal';
import ForgotPassword from './ForgotPassword';
import Signin from './Signin';
import Signup from './Signup';
import SigninInformation from './SignupInformation';

const LoginModal = () => {
  const [islogin, setIsLogin] = useState(true);
  const [forgotPassword, setForgotPassword] = useState(false);
  const [signinInfo, setSigninInfo] = useState(false);
  const [userInfo, setUserInfo] = useState(false);
  const { openModal, setOpenModal } = useModal();
  const { currentUser } = useAuth();

  useEffect(() => {
    if (currentUser)
      setOpenModal(false)
  }, [currentUser, setOpenModal])

  return (
    <div className="modal">
      <i onClick={() => setOpenModal(!openModal)} className="fas fa-times"></i>
      {!signinInfo ?
        (!forgotPassword ?
          (<>
            {islogin && <Signin setisLogin={setisLogin} islogin={islogin}
setForgotPassword={setForgotPassword} />}
            {!islogin && <Signup setisLogin={setisLogin} islogin={islogin}
setSigninInfo={setSigninInfo} signinInfo={signinInfo}
userInfo={userInfo} setUserInfo={setUserInfo} />}
          </>)
          : <ForgotPassword setForgotPassword={setForgotPassword} />
        )
        :
          <SigninInformation userInfo={userInfo} setUserInfo={setUserInfo} />
      }
    </div >
  )
}

export default LoginModal

import React, { useEffect, useState } from 'react';
import { useTheme } from '../hooks/useTheme';
import './css/Signin.css'
import './css/Signup.css'
```



```

import LoginWithPopUp from './LoginWithPopup';

const Signup = ({ islogin, setisLogin, signinInfo, setSigninInfo, setUserInfo,
userInfo }) => {

  const { navColor } = useTheme();
  const [values, setValues] = useState({
    email: "",
    password: "",
    confirmPassword: ""
  });

  const [message, setMessage] = useState({
    error: null
  });

  useEffect(() => {
    setMessage({
      error: null
    });
  }, [values])

  const { email, password, confirmPassword } = values;

  const handleChange = name => event => {
    setValues({ ...values, [name]: event.target.value });
  }

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (values.password !== values.confirmPassword)
      return setMessage({ ...message, error: "Password do not match" })

    setUserInfo({ ...userInfo, email: email, password: password })
    setSigninInfo(!signinInfo)
  }

  return (
    <div className="loginForm signup">
      <form onSubmit={handleSubmit}>
        <h1>Sign up</h1>
        <div className="field">
          <label htmlFor="email">Email:</label>
          <input required autoComplete="off" name="email"
onChange={handleChange("email")} value={email} type="email" id="email" />
        </div>
        <div className="field">
          <label htmlFor="password">Password:</label>
          <input required autoComplete="off" name="password"
onChange={handleChange("password")} value={password} type="password"
id="password" />
        </div>
        <div className="field">

```

```

        <label htmlFor="confirmPassword">Confirm password:</label>
        <input required autoComplete="off" name="confirmPassword"
onChange={handleChange("confirmPassword")} value={confirmPassword}
type="password" id="confirmPassword" />
      </div>
      {message.error && <div className="loginError">
<p>{message.error}</p></div>}
      <button className="signin" style={{ backgroundColor: `${navColor}`,
color: "#fff" }} >Sign up</button>
      <p className="createAcc">Already Have an Account ? <span onClick={()
=> setIsLogin(!islogin)}>Log in</span></p>
    </form>
    <div className="googleSignin">
      <p>or</p>
      <LoginWithPopUp message={message} setMessage={setMessage} />
    </div>
  </div>)
}

```

```
export default Signup
```

```

import {
  arrayRemove,
  deleteDoc,
  doc,
  getDoc,
  updateDoc,
} from "@firebase/firestore";
import { deleteObject, ref } from "@firebase/storage";
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router";
import { useAuth } from "../context/AuthContext";
import { db, storage } from "../firebase/firebaseConfig";
import { useModal } from "../hooks/useModal";
import { useTheme } from "../hooks/useTheme";
import AuthModal from "../AuthModal";
import ConfirmationAlert from "../ConfirmationAlert";
import "../css/Recipe.css";
import LoginAlertModal from "../LoginAlertModal";

```

```

function Recipe({ recipe, id }) {
  const { mode } = useTheme();
  const [deleteRecipe, setDeleteRecipe] = useState(false);
  const navigate = useNavigate();
  const { navColor } = useTheme();
  const { currentUser } = useAuth();
  const { openModal, setOpenModal, setModalContent } = useModal();

  function cookFunction() {
    navigate(`/recipe/${id}`);
  }

  useEffect(() => {

```

```

async function removeRecipe() {
  const docSnap = await getDoc(doc(db, "recipe_list", id));
  if (docSnap.exists()) {
    const recipeData = docSnap.data();
    const imgRef = ref(storage, recipeData.recipeImagePath);

    await updateDoc(doc(db, "users", recipeData.authorUid), {
      recipeAdded: arrayRemove(id),
    });

    await deleteDoc(doc(db, "recipe_list", id));
    await deleteObject(imgRef);
  }
}

if (deleteRecipe === true) {
  removeRecipe();
}
}, [deleteRecipe, id, currentUser]);

async function removeRecipe() {
  try {
    if (!currentUser) {
      setModalContent(<LoginAlertModal />);
      return setOpenModal(!openModal);
    }
    const docSnap = await getDoc(doc(db, "users", currentUser.uid));
    const userData = docSnap.data();
    if (recipe.authorUid === currentUser.uid || userData.role === "admin") {
      setModalContent(
        <ConfirmationAlert
          message="Are you sure you want to delete recipe ?"
          setDeleteRecipe={setDeleteRecipe}
        />
      );
      setOpenModal(true);
    } else {
      setModalContent(
        <AuthModal message="You can only delete your own recipe." />
      );
      setOpenModal(!openModal);
    }
  } catch (err) {
    alert("Failed to remove recipe!", err.message);
  }
}

function truncateString(inputString, noOfWords) {
  let arrayOfWords = inputString.trim().split(" ");
  if (arrayOfWords.length < noOfWords) return inputString;
  const truncated = arrayOfWords.slice(0, noOfWords).join("
").concat("...");
  return truncated;
}

```

```

return (
  <div className={`recipe_item ${mode}`}>
    <i onClick={removeRecipe} className="fas fa-trash-alt"></i>
    <div className="recipe_card">
      <div className="recipe_img">
        <img src={` ${recipe.recipeImageUrl}`} alt={` ${recipe.title}`} />
      </div>
      <p>{recipe.title}</p>
      <p className={` ${mode}`}>{recipe.time} minutes to make</p>
      <p className="recipeMethod">{truncateString(recipe.method, 20)}</p>
      <button
        onClick={cookFunction}
        style={{ backgroundColor: ` ${navColor}`, color: "#fff" }}
      >
        Cook This
      </button>
    </div>
  </div>
);
}

export default Recipe;

import React, { useEffect, useRef } from 'react';
import { useNavigate } from 'react-router';
import { useAuth } from '../context/AuthContext';
import { useModal } from '../hooks/useModal';
import { useTheme } from '../hooks/useTheme';
import './css/Navbar.css'
import LoginModal from './LoginModal';
import LoginAlertModal from './LoginAlertModal';
import Modal from './Modal';
import UserAvatar from './UserAvatar';

const Navbar = ({ setSearchTerm }) => {
  const { currentUser } = useAuth();
  const { openModal, setOpenModal, modalContent, setModalContent } =
    useModal();
  const { navColor } = useTheme();
  const navigate = useNavigate();
  const search = useRef();

  useEffect(() => {
    setModalContent(<LoginModal />)
  }, [setModalContent])

  function addRecipe() {
    navigate('/createrecipe')
    if (!currentUser) {
      setTimeout(() => {
        setModalContent(<LoginAlertModal />)

```

```

        setOpenModal(!openModal)
        setTimeout(() => navigate('/'), 300)
      }, 400)
    }
  }

  return (
    <nav style={{ backgroundColor: `${navColor}` }}>
      <div className="navbar">
        <div className="logo">
          <h1 onClick={() => navigate("/")} style={{ cursor: "pointer"
}}>Spicy Dragon</h1>
        </div>
        <div className="nav_search">
          <label htmlFor="search">Search: </label>
          <input onChange={() => { setSearchTerm(search.current.value); }}
ref={search} type="text" id="search" />
          {openModal && <Modal >{modalContent}</Modal>}
          <button className="addRecipe" style={{ backgroundColor:
`${navColor}` }} onClick={addRecipe}>Add Recipe</button>
          {!currentUser && <button style={{ backgroundColor: `${navColor}` }}
onClick={() => { setModalContent(<LoginModal />); setOpenModal(!openModal)
}}>Login</button>}
          {currentUser && < UserAvatar />}
        </div>
      </div>
    </nav >
  )
}

export default Navbar

import React, { useEffect } from 'react'
import { useLoading } from '../hooks/useLoading';
import { useRecipeList } from '../hooks/useRecipeList';
import { useTheme } from '../hooks/useTheme';
import "../css/Restaurant.css"
import DummyContent from './DummyContent';
import Recipe from './Recipe';
// import "../database.js"

const Restaurant = ({ searchTerm }) => {
  const { setContentIsReady } = useLoading();
  const { mode } = useTheme();
  const { recipeList } = useRecipeList()
  let filteredList = recipeList && recipeList.filter((recipe) =>
recipe.title.toLowerCase().includes(searchTerm.trim().toLowerCase()))

  useEffect(() => {
    setContentIsReady(false)
  }, [setContentIsReady])

```

```

useEffect(() => {
  if (recipeList)
    setContentIsReady(true)
}, [recipeList, setContentIsReady])

return (
  recipeList ? (<div className={`recipelists ${mode}`} >
    {
      filteredList.map((recipe) => {
        return <Recipe key={recipe.id} recipe={recipe} id={recipe.id} />
      })
    }
  </div >) :
  <DummyContent />
)
}

export default Restaurant

import { addDoc, arrayUnion, collection, doc, getDoc, Timestamp, updateDoc }
from '@firebase/firestore';
import { deleteObject, getDownloadURL, ref, uploadBytesResumable } from
'@firebase/storage';
import { CircularProgress } from '@mui/material';
import React, { useEffect, useRef, useState } from 'react'
import { useLocation, useParams } from 'react-router';
import { useAuth } from '../context/AuthContext';
import { db, storage } from '../firebase/firebaseConfig';
import { useLoading } from '../hooks/useLoading';
import { useTheme } from '../hooks/useTheme';
import { optimizeFile } from '../utility_js/imageOptimize';
import Alert from './Alert';
import './css/CreateRecipe.css'
import UploadProgress from './UploadProgress';

const CreateRecipe = () => {
  const location = useLocation()
  const { setContentIsReady } = useLoading();
  const { navColor, mode } = useTheme();
  const { currentUser } = useAuth()
  const { recipeid } = useParams();
  const [isLoading, setIsLoading] = useState(false);
  const ingref = useRef();
  const addRef = useRef();
  const recipeImageRef = useRef();

  const [responseImageEdit, setResponseImageEdit] = useState(null);
  const [compressedFile, setCompressedFile] = useState(null);
  const [uploadStatus, setUploadStatus] = useState({
    progress: 0,
    state: ''
  })

```

```

})

useEffect(() => {
  setContentIsReady(true)
}, [setContentIsReady, recipeid])

const [values, setValues] = useState({
  title: '',
  ingredients: [],
  method: '',
  time: 1,
  imageUrl: ''
})
const [message, setMessage] = useState({
  success: null,
  error: null,
  warning: null,
  recipeRepeat: false
});

useEffect(() => {
  var r = document.querySelector(':root');
  r.style.setProperty('--navColor', `${navColor}`);
}, [navColor])

useEffect(() => {
  if (!recipeid) {
    setValues({
      title: '',
      ingredients: [],
      method: '',
      time: 1,
      imageUrl: ''
    })
    setResponseImageEdit(null);
    setUploadStatus({
      progress: 0,
      state: ''
    })
    return
  }
  async function getRecipe() {
    const recipeData = location.state;
    setValues({
      title: recipeData.title,
      ingredients: recipeData.ingredients,
      method: recipeData.method,
      time: recipeData.time,
      imageUrl: ''
    })
  }
}

```

```

    getRecipe()

}, [recipeid, location]);

useEffect(() => {
  setMessage({
    success: null,
    error: null,
    warning: null,
    recipeRepeat: false
  });
}, [values])

const { title, method, time, ingredients, imageUrl } = values;

const handleChange = name => event => {
  if (name === 'imageUrl') {
    setUploadStatus({ ...uploadStatus, progress: 0 })
    optimizeFile(event.target.files[0], setCompressedFile)
  }
  setValues({ ...values, [name]: event.target.value });
}

function addItem(e) {
  if (e.target !== addRef.current)
    return
  e.preventDefault();
  ingref.current.focus();
  if (!ingref.current.value.trim())
    return;
  if (ingref.current.value)
    setValues({ ...values, ingredients: [...ingredients,
ingref.current.value.trim()] })
  ingref.current.value = "";
}

function deleteItem(e) {
  e.preventDefault();
  ingref.current.focus();
  ingredients.pop();
  setValues({ ...values, ingredients: ingredients })
}

async function handleSubmit(e) {
  e.preventDefault();
  if (!currentUser)
    return setMessage({ ...message, success: '', error: 'Please login to
submit the recipe.' })

  if (message.recipeRepeat)

```



```

        return setMessage({ ...message, success: '', error: 'Resubmitting same
recipe is not allowed' })

    if (!title.trim() || !method.trim() || !ingredients.length) {
        switch (false) {
            case Boolean(title.trim()):
                return setMessage({ ...message, warning: "Title field is empty" })
            case Boolean(ingredients.length):
                return setMessage({ ...message, warning: "Requires atleast one
Ingredient" })
            case Boolean(method.trim()):
                return setMessage({ ...message, warning: "Method field is empty" })
            case Boolean(time):
                return setMessage({ ...message, warning: "Time field is empty" })
            default:
        }
    }
}
const recipeRef = collection(db, 'recipe_list')
if (recipeid) {
    try {
        if (!compressedFile && responseImageEdit)
            return;
        setIsLoading(!isLoading)
        const lastEditedTime = Timestamp.fromDate(new Date())

        if (responseImageEdit) {
            const recipeImagePath = `recipe_image/${title}_${new
Date().toISOString()}.jpg`;

            const storageRef = ref(storage, recipeImagePath);

            const metadata = {
                contentType: 'image/jpeg'
            };

            const uploadTask = uploadBytesResumable(storageRef, compressedFile,
metadata);

            // Listen for state changes, errors, and completion of the upload.
            uploadTask.on('state_changed',
                (snapshot) => {
                    // Get task progress, including the number of bytes uploaded and
the total number of bytes to be uploaded
                    const progress = (snapshot.bytesTransferred /
snapshot.totalBytes) * 100;
                    setUploadStatus({ ...uploadStatus, progress, state:
snapshot.state })
                },
                (error) => {
                    throw new Error(error.code)
                }
            )
        }
    }
}

```

```

    },
    () => {
        // Upload completed successfully, now we can get the download
        URL
        (async function recipeUpload() {
            const recipeImageUrl = await
getDownloadURL(uploadTask.snapshot.ref)
            setUploadStatus({ ...uploadStatus, progress: 100, state:
'uploaded' })

            //delete file
            const docSnap = await getDoc(doc(db, "recipe_list",
recipeid));
            if (docSnap.exists()) {
                const recipeData = docSnap.data();
                const imgRef = ref(storage, recipeData.recipeImagePath);
                await deleteObject(imgRef)
            }

            await updateDoc(doc(db, 'recipe_list', recipeid), { ...values,
lastEditedTime, recipeImageUrl, recipeImagePath, editedAuthorUid:
currentUser.uid })

            setMessage({ ...message, success: "Recipe successfully
edited!", recipeRepeat: true })
            setIsLoading((loading) => !loading)

        })();
    }
  );
}
else {
    const { imageUrl, ...updatedValue } = values
    await updateDoc(doc(db, 'recipe_list', recipeid), { ...updatedValue,
lastEditedTime, editedAuthorUid: currentUser.uid })

    setMessage({ ...message, success: "Recipe successfully edited!",
recipeRepeat: true })
    setIsLoading((loading) => !loading)
  }
}
catch (err) {
    setIsLoading((loading) => !loading)
    setMessage({ ...message, error: err.message })
  }
  return;
}
try {
    if (!compressedFile)
        return;
    setIsLoading(!isLoading)
    let createTime = Timestamp.fromDate(new Date())

```

```

    const lastEditedTime = Timestamp.fromDate(new Date())
    let authorUid = currentUser.uid;

    const recipeImagePath = `recipe_image/${title}_${new
Date().toISOString()}.jpg`;
    const storageRef = ref(storage, recipeImagePath);

    const metadata = {
      contentType: 'image/jpeg'
    };
    const uploadTask = uploadBytesResumable(storageRef, compressedFile,
metadata);

    // Listen for state changes, errors, and completion of the upload.
    uploadTask.on('state_changed',
      (snapshot) => {
        // Get task progress, including the number of bytes uploaded and the
total number of bytes to be uploaded
        const progress = (snapshot.bytesTransferred / snapshot.totalBytes) *
100;
        setUploadStatus({ ...uploadStatus, progress, state: snapshot.state
})
      },
      (error) => {
        throw new Error(error.code)
      },
      () => {
        // Upload completed successfully, now we can get the download URL
        (async function recipeUpload() {
          const recipeImageUrl = await
getDownloadURL(uploadTask.snapshot.ref)
          setUploadStatus({ ...uploadStatus, progress: 100, state:
'uploaded' })
          const recipeDoc = await addDoc(recipeRef, { ...values,
createdTime, authorUid, lastEditedTime, recipeImageUrl, recipeImagePath })

          await updateDoc(doc(db, "users", currentUser.uid), {
            recipeAdded: arrayUnion(recipeDoc.id)
          })
          // await setDoc(doc(db, "recipe_list", currentUser.uid), {
...recipeData, createdTime })
          setMessage({ ...message, success: "Recipe successfully added!",
recipeRepeat: true })
          setIsLoading((loading) => !loading)
        })();
      }
    );

    // await uploadBytes(storageRef, recipeImageRef.current.files[0])
    // const recipeImageUrl = await getDownloadURL(ref(storage,
recipeImagePath))

```

```

    }
    catch (err) {
        setIsLoading((loading) => !loading)
        setMessage({ ...message, error: err.message })
    }
}
function responseImage(e) {
    e.preventDefault();
    e.target.textContent === 'Yes' ? setResponseImageEdit(true) :
setResponseImageEdit(false)
}

return (
    <div className="recipeform" onSubmit={handleSubmit}>
        <form >
            <h2 className={` ${mode} `}>Add a New Recipe</h2>
            <div className="field">
                <label className={` ${mode} `} htmlFor="title">Recipe title:</label>
                <input name="title" onChange={handleChange("title")} value={title}
type="text" id="title" />
            </div>
            <div className="field">
                <label className={` ${mode} `} htmlFor="ingredients"
name="ingredients">Recipe Ingredients</label>
                <div>
                    <input ref={ingref} type="text" id="ingredients" />
                    <button ref={addRef} className="addButton" style={{
backgroundColor: ` ${navColor} `, color: "#fff" }} onClick={addItem}>
                        add
                        {ingredients.length ?
                            (<div className="deleteButton" onClick={deleteItem} style={{
backgroundColor: ` ${navColor} `, color: "#fff" }}>
                                <i className="fas fa-minus"></i>
                            </div>) : null}
                        </button>
                    </div>
                <p className={` label ${mode} `} style={{ paddingRight: "50px"
}}>Current ingredients: {ingredients.join(', ')}</p>
            </div >
            <div className="field">
                <label className={` ${mode} `} htmlFor="method">Recipe Method:</label>
                <textarea onChange={handleChange("method")} value={method}
form="recipeform" id="method">
            </textarea>
        </div>
        <div className="field">
            <label className={` ${mode} `} htmlFor="time">Cooking time
(minutes):</label>
            <input onChange={handleChange("time")} value={time} type="number"
min="1" max="10000" id="time" />
        </div>
        {responseImageEdit === false ? null :

```

```

    <div className="field">
      <label className={` ${mode}`} htmlFor="recipeImage">Recipe
image:</label>
      <input required ref={recipeImageRef} className={` ${mode}`}
accept="image/*" onChange={handleChange("imageUrl")} value={imageUrl}
type="file" id="recipeImage" />
      {isLoading ?
        <UploadProgress uploadStatus={uploadStatus}
setUploadStatus={setUploadStatus} />
        : null}
      {recipeid && !responseImageEdit ? <div className={` imageEdit
${mode}`}>
        <p>Edit recipe image?</p>
        <div className="responseBtn" >
          <button onClick={responseImage}>Yes</button>
          <button onClick={responseImage}>No</button>
        </div>
        </div> : null}
    </div>}

    <Alert message={message} />
    {
      !isLoading ?
        <button style={{ backgroundColor: `${navColor}`, color: "#fff" }}
className="submit" >submit</button>
        : <button disabled className="signin loading" style={{ margin:
"auto", width: "72px", height: "32px", backgroundColor: `${navColor}40`,
color: "#fff" }} ><CircularProgress style={{ marginTop: "8px", width: "15px",
height: "15px" }} /></button>
    }
  </form >
</div >
)
}

export default CreateRecipe

```

CONCLUSION

The creation of a Food Mania Website using Spring Boot represents a dynamic and engaging platform for culinary enthusiasts to connect, share, and explore the world of gastronomy. Throughout the development process, we have harnessed the capabilities of Spring Boot, a powerful and versatile Java-based framework, to construct a feature-rich and user-friendly website tailored to the needs of both content creators and food enthusiasts.

This project has led to the realization of a comprehensive online culinary community where users can register, create and customize their profiles, and actively contribute to the platform's content. Content creators are empowered to share their culinary creations through detailed recipes, and users can explore a vast collection of recipes, engage in discussions through comments, and build connections with like-minded individuals.

FUTURE WORK

1. Mobile Application Development: Develop native mobile apps for iOS and Android to provide a more convenient user experience.
2. Advanced Search and Recommendation: Implement personalized recipe recommendations and intelligent search capabilities to enhance content discovery.
3. Enhanced User Engagement: Expand social features to facilitate user interactions, such as direct messaging and user-to-user following.
4. Monetization Strategies: Explore revenue generation options, including advertising, premium memberships, and sponsored content.
5. Security and Compliance: Continuously update and enhance security measures while ensuring compliance with data privacy regulations to protect user data.