A Lab Project Report

on

**Zomato Clone**

Submitted in partial fulfillment of the requirements for the award of the degree of

#### BACHELOR OF ENGINEERING

**COMPUTER SCIENCE AND ENGINEERING**

BY

**ITHA MANISHA(1602-19-733-024)**

**Y.Akshaya(1602-19-733-007)**

Under the guidance of

**Vinay sir**

A picture containing diagram

Description automatically generated

**Department of Computer Science Engineering**

**Vasavi College Of Engineering (Affiliated to Osmania University)**

**Ibrahimbagh, Hyderabad-31**

**2021**

**ACKNOWLEDGEMENT**

We would like to express our heartfelt gratitude to Sateesh sir, our project guide, for her valuable guidance and constant support, along with his capable instructions and persistent encouragement.

We are grateful to our Head of Department, Dr. T. Adilakshmi, for her steady support and the provision of every resource required for the completion of this project.

We would like to take this opportunity to thank our Principal, Dr. S. V. Ramana, as well as the management of the institute, for having designed an excellent learning atmosphere.

We are thankful to and fortunate enough to get constant encouragement, support and guidance which helped us in successfully completing our project work.

**TABLE OF CONTENTS**

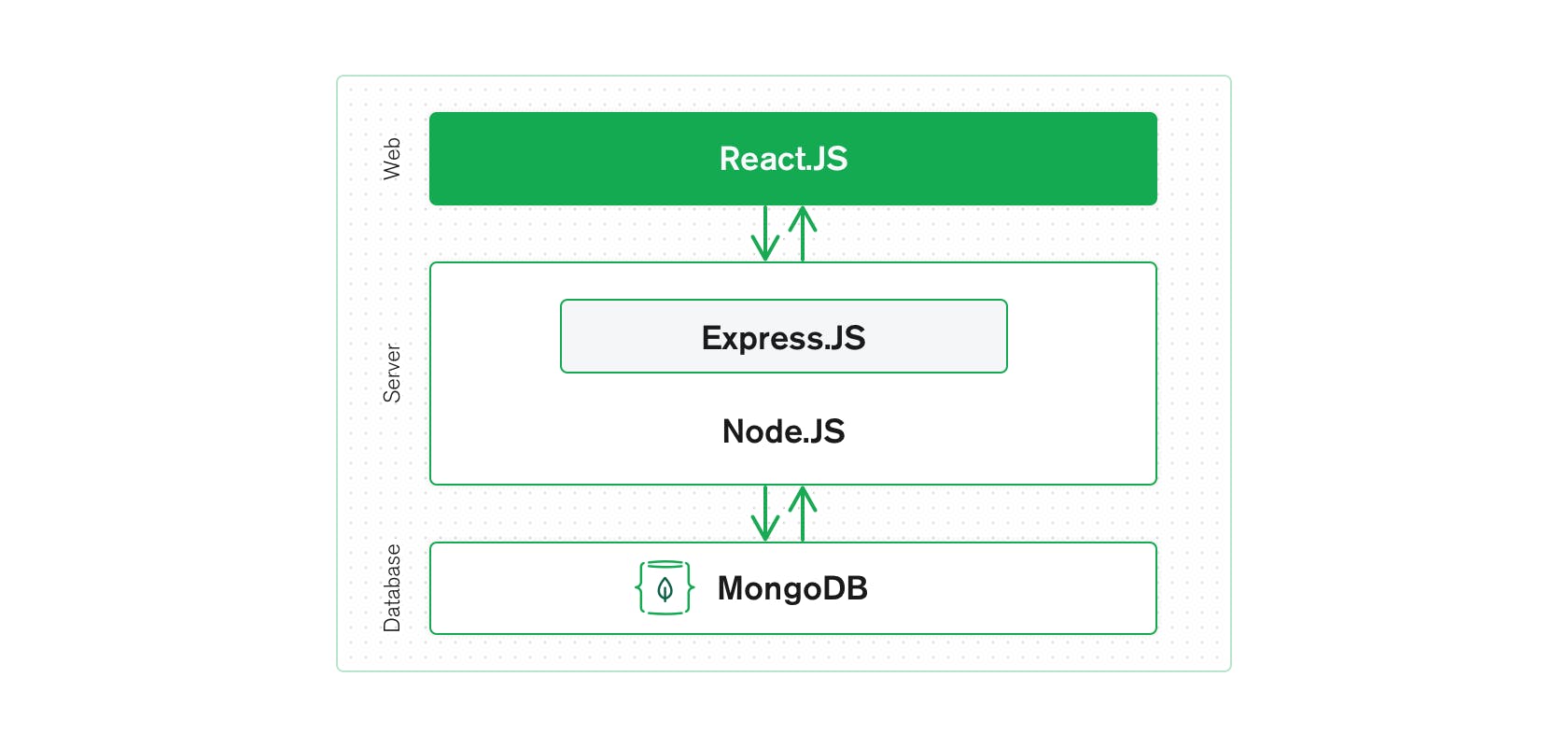
1. ABSTRACT……...……………………………………………..04
2. INTRODUCTION………………………………………………05
3. IMPLEMENTATION…………………………………………...06
4. ER DIAGRAM…………………………………………………..30
5. OUTPUT …….………………………………………………….31
6. CONCLUSION …………………………………………………40
7. REFERENCES …………………………………………………41

**1. ABSTRACT**

Zomato Clone app is a MERN stack application. This app uses API calls to get restaurants, recipes, ratings, etc around the users Location. It allows users to order their favourite food from favourite restaurant. They also have the privilege to rate the restaurant and write the review. User can order food online , search for popular restaurants, search for restaurants located nearby, delivery rating. User have to sign in or sign up before rating, checkout and make payments. User can sign in or sign up using google sign in or through email verification.

**2.INTRODUCTION**

This project is a MERN stack application where MERN Stack is a Javascript Stack that is used for easier and faster deployment of full-stack web applications. MERN Stack comprises of 4 technologies namely: MongoDB, Express, React and Node.js. It is designed to make the development process smoother and easier. Each of these 4 powerful technologies provides an end-to-end framework for the developers to work in and each of these technologies play a big part in the development of web applications. Express and Node make up the middle (application) tier. Express.js is a server-side web framework, and Node.js the popular and powerful JavaScript server platform.



**3.IMPLEMENTATION**

**Folder: Server**

**File: src\API\Auth\index.js**

// Library

import express from "express";

import passport from "passport";

// Models

import { UserModel } from "../../database/user/index";

//validation

import { ValidateSignin, ValidateSignup } from "../../validation/auth";

const Router = express.Router();

/\*

Route   - /auth/signup

Desc    - Register new user

Params  - None

Access  - Public

Methos  - POST

\*/

Router.post("/signup", async(*req*,*res*) => {

    try{

        await ValidateSignup(req.body.credentials);

        await UserModel.findByEmailAndPhone(req.body.credentials);

        const newUser = await UserModel.create(req.body.credentials);

        const token = newUser.generateJwtToken();

        return res.status(200).json({token, status: "success"});

    }

    catch(error){

        return res.status(500).json({error: error.message })

    }

});

/\*

Route   - /auth/signin

Desc    - signin with email and password

Params  - None

Access  - Public

Methos  - POST

\*/

Router.post("/signin", async(*req*,*res*) => {

    try{

        await ValidateSignin(req.body.credentials);

        const user = await UserModel.findByEmailAndPassword(req.body.credentials);

        const token = user.generateJwtToken();

        return res.status(200).json({token, status: "success"});

    } catch(error) {

        return res.status(500).json({error: error.message });

    }

});

/\*

Route   - /auth/google

Desc    - route for google authentication

Params  - None

Access  - Public

Methos  - GET

\*/

Router.get(

    "/google",

     passport.authenticate("google", {

        scope: [

            "https://www.googleapis.com/auth/userinfo.profile",

            "https://www.googleapis.com/auth/userinfo.email",

        ],

    })

);

/\*

Route   - /auth/google/callback

Desc    - google callback function

Params  - None

Access  - Public

Methos  - GET

\*/

Router.get(

    "/google/callback",

    passport.authenticate("google", {failureRedirect: "/"}),

        (*req*,*res*) => {

            return res.redirect(

                `https://zomatoclone-netlify.netlify.app/google/${req.session.passport.user.token}`

              );

    }

);

export default Router;

**File: src\API\Food\index.js**

//Libraries

import express from "express";

//Database Model

import { FoodModel } from "../../database/allModels";

import { ValidateRestaurantId, ValidateCategory } from "../../validation/food";

const Router = express.Router();

/\*

Route   food/:\_id

Des     Get specific food

Params  \_id

Access  Public

Method  GET

\*/

Router.get("/:\_id", async (*req*, *res*) => {

    try {

      const { \_id } = req.params;

      const foods = await FoodModel.findById(\_id);

      return res.json({ foods });

    } catch (error) {

      return res.status(500).json({ error: error.message });

    }

  });

/\*

Route   food/r/:\_id

Des     Get all food based on particular restaurant

Params  \_id

Access  Public

Method  GET

\*/

Router.get("/r/:\_id", async (*req*,*res*) => {

    try{

        await ValidateRestaurantId(req.params);

        const {\_id} = req.params;

        const foods= await FoodModel.find({ restaurant: \_id});

        return res.json({foods});

    } catch(error) {

        return res.status(500).json({error: error.message});

    }

});

/\*

Route   /food/c/:category

Des     Get all food based on particular category

Params  category

Access  Public

Method  GET

\*/

Router.get("/c/:category", async (*req*,*res*) => {

    try{

        await ValidateCategory(req.params);

        const {category} = req.params;

        const foods= await FoodModel.find({

            category: { $regex: category, $options: "i" },

        });

        return res.json({foods});

    } catch(error) {

        return res.status(500).json({error: error.message});

    }

});

export default Router;

**File: src\API\Image\index.js**

// Libraries

import express from "express";

import multer from "multer";

//Database Modal

import { ImageModel } from "../../database/allModels";

//upload to s3

import { s3Upload } from "../../Utils/AWS/s3";

const Router = express.Router();

// multer config

const storage = multer.memoryStorage();

const upload = multer({ storage });

/\*

Route     /

Des       Get Image details

Params    \_id

Access    Public

Method    GET

\*/

Router.get("/:\_id", async (*req*, *res*) => {

  try {

    const image = await ImageModel.findById(req.params.\_id);

    return res.json({ image });

  } catch (error) {

    return res.status(500).json({ error: error.message });

  }

});

/\*

Route           /image

Des             Uploads given image to S3 bucket, and saves file link to mongodb

Params          none

Access          Public

Method          POST

\*/

Router.post("/", upload.single("file"), async (*req*, *res*) => {

  try {

    const file = req.file;

    //s3 bucket options

    const bucketOptions = {

      Bucket: "shapeai-zomato-abcd",

      Key: file.originalname,

      Body: file.buffer,

      ContentType: file.mimetype,

      ACL: "public-read", //Access Control List

    };

    const uploadImage = await s3Upload(bucketOptions);

    return res.status(200).json({ uploadImage });

  } catch (error) {

    return res.status(500).json({ error: error.message });

  }

});

export default Router;

**File: src\API\Menu\index.js**

//Libraries

import express from "express";

//Database Model

import { MenuModel, ImageModel } from "../../database/allModels";

import { ValidateId } from "../../validation/menu";

const Router = express.Router();

/\*

Route   /menu/list/:\_id

Des     Get all list of menu based on id

Params  \_id

Access  Public

Method  GET

\*/

Router.get("/list/:\_id", async (*req*,*res*) => {

    try{

        await ValidateId(req.params);

        const {\_id} = req.params;

        const menus = await MenuModel.findById(\_id);

        return res.json({menus});

    } catch(error) {

        return res.status(500).json({error: error.message});

    }

});

/\*

Route   /menu/image/:\_id

Des     Get all menu images based on id

Params  \_id

Access  Public

Method  GET

\*/

Router.get("/image/:\_id", async (*req*,*res*) => {

    try{

        await ValidateId(req.params);

        const {\_id} = req.params;

        const menus = await ImageModel.findOne(\_id);

        return res.json({menus});

    } catch(error) {

        return res.status(500).json({error: error.message});

    }

});

export default Router;

**File: src\API\Orders\index.js**

// Libraries

import express from "express";

import passport from "passport";

// Database Schema

import { OrderModel } from "../../database/allModels";

const Router = express.Router();

/\*

Route           /order/:\_id

Des             Get all orders based on id

Params          \_id

Access          Public

Method          GET

\*/

Router.get(

  "/:\_id",

  passport.authenticate("jwt", { session: false }),

  async (*req*, *res*) => {

    try {

      const { \_id } = req.params;

      const getOrders = await OrderModel.findOne({ user: \_id });

      if (!getOrders) {

        return res.status(404).json({ error: "User not found" });

      }

      return res.status(200).json({ orders: getOrders });

    } catch (error) {

      return res.status(500).json({ error: error.message });

    }

  }

);

/\*

Route           /restaurant/new/:\_id

Des             Add new order

Params          \_id

Access          Public

Method          POST

\*/

Router.post("/new", passport.authenticate("jwt"), async (*req*, *res*) => {

  try {

    const { \_id } = req.session.passport.user.\_doc;

    const { orderDetails } = req.body;

    const addNewOrder = await OrderModel.findOneAndUpdate(

      {

        user: \_id,

      },

      {

        $push: { orderDetails },

      },

      {

        new: true,

      }

    );

    if (!addNewOrder) {

      const details = await OrderModel.create({

        user: \_id,

        orderDetails: [orderDetails],

      });

      console.log(details);

      return res.json({ order: details });

    }

    return res.json({ order: addNewOrder });

  } catch (error) {

    return res.status(500).json({ error: error.message });

  }

});

export default Router;

**File: src\API\Payments\index.js**

// Libraries

import express from "express";

import passport from "passport";

import Razorpay from "razorpay";

import { v4 as uuidv4 } from "uuid";

// Database Modal

import { MenuModel } from "../../database/allModels";

const Router = express.Router();

/\*

Route           /payments/new

Des             For creating a order/payment

Params          none

Access          Public

Method          POST

\*/

Router.post("/new", async (*req*, *res*) => {

  try {

    const instance = new Razorpay({

      key\_id: process.env.RZR\_PAY\_ID,

      key\_secret: process.env.RZR\_PAY\_SECRET,

    });

    const options = {

      amount: req.body.amount \* 100,

      currency: "INR",

      receipt: `${uuidv4()}`,

    };

    const order = await instance.orders.create(options);

    return res.json({ order });

  } catch (error) {

    return res.status(500).json({ error: error.message });

  }

});

export default Router;

**File: src\API\Restaurant\index.js**

//Libraries

import express from "express";

//Database Model

import { RestaurantModel } from "../../database/allModels";

//validation

import { ValidateRestaurantCity, ValidateRestaurantSearchString } from "../../validation/restaurant";

import { ValidateRestaurantId } from "../../validation/food";

const Router = express.Router();

/\*

Route   /restaurant/?city=

Des     Get all the restaurant details based on the city name

Params  none

Access  Public

Method  GET

\*/

Router.get("/",  async (*req*,*res*) => {

    try{;

        await ValidateRestaurantCity(req.query)

        const {city } = req.query;

        const restaurants = await RestaurantModel.find({ city });

        return res.json({ restaurants });

    } catch(error) {

        return res.status(500).json({error: error.message});

    }

});

/\*

Route   /restaurant/:\_id

Des     Get individual restaurant details based on id

Params  \_id

Access  Public

Method  GET

\*/

Router.get("/:\_id",  async (*req*,*res*) => {

    try{

        await ValidateRestaurantId(req.params);

        const { \_id } = req.params;

        const restaurant = await RestaurantModel.findById(\_id);

        if(!restaurant){

            return res.status(404).json({error: "Restaurant Not Found!!!"});

        }

        return res.json({ restaurant });

    } catch(error) {

        return res.status(500).json({error: error.message});

    }

});

/\*

Route   /search

Des     Get restaurant details based on search string

Params  none

Access  Public

Method  GET

\*/

Router.get("/search",  async (*req*,*res*) => {

    try{

        await ValidateRestaurantSearchString(req.body);

        const {searchString } = req.body;

        const restaurants = await RestaurantModel.find({

            name: {$regex: searchString, $options: "i"},

        });

        if(!restaurants){

            return res.status(404).json({error: `No restaurant matched with ${searchString}`});

        }

        return res.json({restaurants});

    } catch(error) {

        return res.status(500).json({error: error.message});

    }

});

export default Router;

**File: src\API\Reviews\index.js**

//Libraries

import express from "express";

import passport from "passport";

//Database Model

import { ReviewModel } from "../../database/allModels";

const Router = express.Router();

/\*

Route   /review/:resid

Des     Get all reviews related to a particular restaurant

Params  resid

Access  Public

Method  GET

\*/

Router.get("/:resid", async (*req*,*res*) => {

    try{

        const {resid} = req.params;

        const reviews = await ReviewModel.find({ restaurant: resid});

        return res.json({reviews});

    } catch(error) {

        return res.status(500).json({error: error.message});

    }

});

/\*

Route   /review/new

Des     Add new food review/rating

Params  none

Access  Private

Method  POST

\*/

Router.post("/new", passport.authenticate("jwt"), async (*req*, *res*) => {

    try {

      const { \_id } = req.session.passport.user.\_doc;

      const { reviewData } = req.body;

      await ReviewModel.create({ ...reviewData, user: \_id });

      return res.json({ review: "Successfully Created Review." });

    } catch (error) {

      return res.status(500).json({ error: error.message });

    }

  });

/\*

Route   /review/delete/:\_id

Des     Delete food review/rating

Params  \_id

Access  Public

Method  DELETE

\*/

Router.delete("/delete/:\_id", async (*req*,*res*) => {

    try{

        const {\_id} = req.params;

        await ReviewModel.findByIdAndDelete(\_id);

        return res.json({review: "Successfully Deleted the Review."});

    } catch(error) {

        return res.status(500).json({error: error.message});

    }

});

export default Router;

**File: src\API\User\index.js**

// Libraries

import express from "express";

// Database Schema

import { UserModel } from "../../database/allModels";

import passport from "passport";

const Router = express.Router();

/\*

Route   /

Des     Get user data

Params  null

Access  Public

Method  GET

\*/

Router.get("/", passport.authenticate("jwt"), (*req*, *res*) => {

    try {

      const { email, fullName, phoneNumber, address } =

        req.session.passport.user.\_doc;

      return res.json({ user: { email, fullName, phoneNumber, address } });

    } catch (error) {

      return res.status(500).json({ error: error.message });

    }

  });

/\*

Route   /user/:\_id

Des     Get user data

Params  \_id

Access  Public

Method  GET

\*/

Router.get("/:\_id", async (*req*, *res*) => {

    try {

      const { \_id } = req.params;

      const getUser = await UserModel.findById(\_id);

      if (!getUser) return res.status(400).json({ user: "User not found" });

      const { fullName } = getUser;

      return res.json({ user: { fullName } });

    } catch (error) {

      return res.status(500).json({ error: error.message });

    }

  });

/\*

Route   /update/:userId

Des     update user data

Params  \_id

BODY    user data

Access  Public

Method  PUT

\*/

Router.put("/update/:userId", async (*req*,*res*) => {

    try{

        const {userId} = req.params;

        const { userData } = req.body;

        const updateUserData = await UserModel.findByIdAndUpdate(userId,

            {

                $set: userData

            },

            {new: true}

            );

        if(!getUser) return res.status(400).json({user: ""});

        return res.json({user: updateUserData});

    } catch(error) {

        return res.status(500).json({error: error.message});

    }

});

export default Router;

**File: src\AWS \s3.js**

import AWS from "aws-sdk";

import dotenv from "dotenv";

dotenv.config();

//AWS s3 bucket

const s3Bucket = new AWS.S3({

    accessKeyID: process.env.AWS\_S3\_ACCESS\_KEY,

    secretAccessKey: process.env.AWS\_S3\_SECRET\_KEY,

    region: 'ap-south-1'

});

export const s3Upload = (*options*) => {

    return new Promise((*resolve*, *reject*) => s3Bucket.upload(options, (*error*, *data*) => {

        if(error) return reject(error);

        return resolve(data);

    }));

};

**File: src\config\google.config.js**

import googleOAuth from "passport-google-oauth20";

import { UserModel } from "../database/allModels";

const GoogleStrategy = googleOAuth.Strategy;

export default (*passport*) => {

    passport.use(

        new GoogleStrategy({

            clientID: process.env.GOOGLE\_CLIENT\_ID,

            clientSecret: process.env.GOOGLE\_CLIENT\_SECRET,

            callbackURL: "https://zomato-clone--heroku-backend.herokuapp.com/auth/google/callback"

        },

        //after authenticating with google you will get these values

        async (*accessToken*, *refreshToken*, *profile*, *done*) => {

            //creating a new user object

            const newUser = {

                fullName: profile.displayName,

                email: profile.emails[0].value,

                profilePic: profile.photos[0].value,

            };

            try{

                //check if the user exists

                const user = await UserModel.findOne({email: newUser.email});

                if(user){

                    //generate token

                    const token = user.generateJwtToken();

                    //return user

                    done(null, {user, token});

                } else {

                    //create new user

                    const user = await UserModel.create(newUser);

                    //generate token

                    const token = user.generateJwtToken();

                    //return user

                    done(null, {user, token});

                }

            }catch(error) {

                done(error, null)

            }

        }

        )

    );

    passport.serializeUser((*userData*, *done*) => done(null, {...userData}));

    passport.deserializeUser((*id*, *done*) => done(null, id));

}

**File: src\config\route.config.js**

import JwtPassort from "passport-jwt";

import dotenv from "dotenv";

import { required } from "joi";

dotenv.config({

    path: require('path').resolve(\_\_dirname, '../.env'),

});

//Database

import { UserModel } from "../database/user";

import passport from "passport";

const JWTStrategy = JwtPassort.Strategy;

const ExtractJwt = JwtPassort.ExtractJwt;

const options = {

    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),

    secretOrKey: "ZomatoAPP"

};

export default (*passport*) => {

    passport.use(

        new JWTStrategy(options, async(*jwt\_\_payload*, *done*)=> {

            try{

                const doesUserExist = await UserModel.findById(jwt\_\_payload.user);

                if(!doesUserExist) return done(null, false);

                return done(null, doesUserExist);

            }catch(error){

                throw new Error(error);

            }

        })

    )

}

**File: src\database\food\index.js**

import mongoose from "mongoose";

const FoodSchema = new mongoose.Schema({

    name: {type: String, required: true},

    descript: {type: String, required: true},

    isVeg: {type: Boolean, required: true},

    isContainsEgg: {type: Boolean, required: true},

    category: {type: String, required: true},

    photos: {

        type: mongoose.Types.ObjectId,

        ref: "Images"

    },

    price: { type: Number, default:150, required: true },

    addOns: [

        {

            type: mongoose.Types.ObjectId,

            ref: "Foods"

        }

    ],

    restaurant: {

        type: mongoose.Types.ObjectId,

        ref: "Restaurants",

        required: true

    }

},{

    timestamps: true

});

export const FoodModel = mongoose.model("Foods", FoodSchema);

**File: src\database\image\index.js**

import mongoose from "mongoose";

const ImageSchema = new mongoose.Schema({

    images: [

        {

            location: {type: String, required: true},

        }

    ]

},{

    timestamps: true

});

export const ImageModel = mongoose.model("Images", ImageSchema);

**File: src\database\menu\index.js**

import mongoose from "mongoose";

const MenuSchema = new mongoose.Schema({

    menu: [

        {

            name: {type: String, required: true},

            items: [

                {

                    type: mongoose.Types.ObjectId,

                    ref: "Foods"

                }

            ]

        }

    ],

    recommended: [

        {

            type: mongoose.Types.ObjectId,

            ref: "Foods",

            unique: true

        }

    ]

},{

    timestamps: true

});

export const MenuModel = mongoose.model("Menus", MenuSchema);

**File: src\database\order\index.js**

import mongoose from "mongoose";

const OrderSchema = new mongoose.Schema({

    user:{

        type: mongoose.Types.ObjectId,

        ref: "Users"

    },

    orderDetails: [

        {

            food: {

                type: mongoose.Types.ObjectId,

                ref: "Foods"

            },

            quantity: {type: Number, required: true},

            paymode: {type: String, required: true},

            status: {type: String, default: "Placed"},

            paymentDetails: {

                itemTotal: {type: Number, required: true},

                promo: {type: Number, required: true},

                tax: {type: Number, required: true},

            },

        },

    ],

},{

    timestamps: true

});

export const OrderModel = mongoose.model("Orders", OrderSchema);

**File: src\database\restaurant\index.js**

import mongoose from "mongoose";

const RestaurantSchema = new mongoose.Schema({

    name: {type: String, required: true},

    city: {type: String, required: true},

    address: {type: String, required: true},

    mapLocation: {type: String, required: true},

    cuisine: [String],

    restaurantTimings: String,

    contactNumber: Number,

    website: String,

    popularDishes: [String],

    averageCost: Number,

    amenties: [String],

    menuImages: {

        type: mongoose.Types.ObjectId,

        ref: "Images"

    },

    menu: {

        type: mongoose.Types.ObjectId,

        ref: "Menus"

    },

    reviews: [{

        type: mongoose.Types.ObjectId,

        ref: "Reviews"

    }],

    photos: {

        type: mongoose.Types.ObjectId,

        ref: "Images"

    }

},{

    timestamps: true

});

export const RestaurantModel = mongoose.model("Restaurants", RestaurantSchema);

**File: src\database\reviews\index.js**

import mongoose from "mongoose";

const ReviewSchema = new mongoose.Schema({

    food: {type: mongoose.Types.ObjectId, ref: "Foods"},

    restaurant: {type: mongoose.Types.ObjectId, ref: "Restaurants"},

    user: {type: mongoose.Types.ObjectId, ref: "Users"},

    rating: {type: Number, required: true},

    reviewText: {type: String, required: true},

    isRestaurantReview: {type: Boolean},

    isFoodReview: {type: Boolean},

    photos: [

        {

            type: mongoose.Types.ObjectId,

            ref: "Images"

        },

    ],

},{

    timestamps: true

});

export const ReviewModel = mongoose.model("Reviews", ReviewSchema);

**File: src\database\user\index.js**

import mongoose from "mongoose";

import bcrypt from "bcryptjs";

import jwt from "jsonwebtoken";

const UserSchema = new mongoose.Schema({

    fullName: {type: String, required: true},

    email: { type: String, required: true},

    password: {type: String},

    address: [{ details: {type: String}, for: {type:String} }],

    phoneNumber: [{type: Number}]

},{

    timestamps: true

});

//statics and methods

UserSchema.methods.generateJwtToken = function () {

    //generate JWT auth token

    return jwt.sign( { user: *this*.\_id.toString() },"ZomatoAPP");

}

UserSchema.statics.findByEmailAndPassword = async ({*password*, *email*}) => {

    //check whether email exists

    const user = await UserModel.findOne({email});

    if(!user)

    {

        throw new Error("User does not exist!!!");

    }

    //compare password

    const doesPasswordMatch = await bcrypt.compare(password, user.password);

    if(!doesPasswordMatch) throw new Error("Invalid Password!!!");

    return user;

}

UserSchema.statics.findByEmailAndPhone = async ({*email*, *phoneNumber*}) => {

    //check whether email exists

    const checkUserByEmail = await UserModel.findOne({email});

    const checkUserByPhone = await UserModel.findOne({phoneNumber});

    if(checkUserByEmail || checkUserByPhone)

    {

        throw new Error("User already exists!");

    }

    return false;

}

UserSchema.pre("save", function(*next*) {

    const user = *this*;

    //password is modified

    if(!user.isModified("password")) return next();

    //password bcrypt salt

    bcrypt.genSalt(8, (*error*, *salt*) => {

        if(error) return next(error);

        //hash the password

        bcrypt.hash(user.password, salt, (*error*, *hash*) => {

            if(error) return next(error);

            //assigning hashed password

            user.password = hash;

            return next();

        });

    });

    //hash password

    // const bcryptSalt = await bcrypt.genSalt(8);

    // const hashedPassword = await bcrypt.hash(password, bcryptSalt);

});

export const UserModel = mongoose.model("Users", UserSchema)

**File: src\database\allModels.js**

import {FoodModel} from "./food";

import {ImageModel} from "./image";

import {MenuModel} from "./menu";

import {OrderModel} from "./order";

import {RestaurantModel} from "./restaurant";

import {ReviewModel} from "./reviews";

import {UserModel} from "./user";

export {

    FoodModel,

    ImageModel,

    MenuModel,

    OrderModel,

    RestaurantModel,

    ReviewModel,

    UserModel,

};

**File: src\database\cnnection.js**

import mongoose from "mongoose";

export default async() => {

    return mongoose.connect(process.env.MONGO\_URI, {

        useNewUrlParser: true,

    });

};

/\*

{

        useNewUrlParser: true,

        userUnifiedTopology: true,

        useCreateIndex: true,

        useFindAndModify: false

}

\*/

**File: src\Utils\AWS\s3.js**

import AWS from "aws-sdk";

import dotenv from "dotenv";

dotenv.config();

// AWS s3 buvket

const s3Bucket = new AWS.S3({

    accessKeyId: process.env.AWS\_S3\_ACCESS\_KEY,

    secretAccessKey: process.env.AWS\_S3\_SECRET\_KEY,

    region: "ap-south-1",

});

export const s3Upload = (*options*) => {

    return new Promise((*resolve*, *reject*) =>

        s3Bucket.upload(options, (*error*, *data*) => {

            if (error) return reject(error);

            return resolve(data);

        })

    );

};

**File: src\validation\auth.js**

import joi from "joi";

export const ValidateSignup = (*userData*) => {

    const Schema = joi.object({

        fullName: joi.string().required().min(5),

        email: joi.string().email().required(),

        password: joi.string().min(5),

        address: joi.array().items(joi.object({ details: joi.string(), for: joi.string()})),

        phoneNumber: joi.number()

    });

    return Schema.validateAsync(userData);

};

export const ValidateSignin = (*userData*) => {

    const Schema = joi.object({

        email: joi.string().email().required(),

        password: joi.string().min(5).required()

    });

    return Schema.validateAsync(userData);

};

**File: src\validation\food.js**

import joi from "joi";

export const ValidateRestaurantId = (*resId*) => {

    const Schema = joi.object({

        \_id: joi.string().required()

    });

    return Schema.validateAsync(resId);

};

export const ValidateCategory = (*category*) => {

    const Schema = joi.object({

        category: joi.string().required()

    });

    return Schema.validateAsync(category);

};

**File: src\validation\menu.js**

import joi from "joi";

export const ValidateId = (*resId*) => {

    const Schema = joi.object({

        \_id: joi.string().required()

    });

    return Schema.validateAsync(resId);

};

**File: src\validation\orders.js**

import joi from "joi";

export const ValidateOrdersId = (*resId*) => {

    const Schema = joi.object({

        \_id: joi.string().required()

    });

    return Schema.validateAsync(resId);

};

**File: src\validation\restaurant.js**

import joi from "joi";

export const ValidateRestaurantCity = (*restaurantObj*) => {

    const Schema = joi.object({

        city: joi.string().required()

    });

    return Schema.validateAsync(restaurantObj);

}

export const ValidateRestaurantSearchString = (*restaurantObj*) => {

    const Schema = joi.object({

        searchString: joi.string().required()

    });

    return Schema.validateAsync(restaurantObj);

}

**File: src\index.js**

require("dotenv").config();

import express from "express";

import cors from "cors";

import helmet from "helmet";

import passport from "passport";

//configs

import googleAuthConfig from "./config/google.config";

import routerConfig from "./config/route.config";

//Routes

import Auth from "./API/Auth/index";

import Restaurant from "./API/Restaurant/index";

import Food from "./API/Food/index";

import Menu from "./API/Menu/index";

import Image from "./API/Image/index";

import Order from "./API/Orders/index";

import Review from "./API/Reviews/index";

import User from "./API/User/index";

import MailService from "./API/Mail/index";

import Payments from "./API/Payments";

//Database connection

import ConnectDB from "./database/connection";

const zomato = express();

//passport config

googleAuthConfig(passport);

routerConfig(passport);

zomato.use(express.json());

zomato.use(express.urlencoded({ extended: false }));

zomato.use(cors());

zomato.use(helmet());

zomato.use(passport.initialize());

zomato.use(passport.session());

zomato.get("/", (*req*, *res*) => {

  res.json({ message: "Setup Success" });

});

zomato.use("/auth", Auth);

zomato.use("/restaurant", Restaurant);

zomato.use("/food", Food);

zomato.use("/menu", Menu);

zomato.use("/image", Image);

zomato.use("/order", Order);

zomato.use("/review", Review);

zomato.use("/user", User);

zomato.use("/mail", MailService);

zomato.use("/payments", Payments);

zomato.listen(process.env.PORT || 4000, () =>

  ConnectDB()

    .then(() => console.log("Server is up and running"))

    .catch((*error*) => {

      console.log(error);

      console.log("Server is running, but database connection failed ...");

    })

);

**4. ER Diagram**

**Graphical user interface

Description automatically generated**

**5.OUTPUT**

**Home Page:**

Graphical user interface, application

Description automatically generated

**Login/signup:**

**Graphical user interface, application

Description automatically generated**

**Graphical user interface, application

Description automatically generated**

**Dining out:**

**Graphical user interface, application

Description automatically generated**

**Night life:**

**Graphical user interface, application

Description automatically generated**

**Nutrition:  
Graphical user interface, website

Description automatically generated**

**Restaurant page:**

**Graphical user interface, application, PowerPoint

Description automatically generated**

**Reviews page:**

**Graphical user interface, application

Description automatically generated**

**Overview:**

**Graphical user interface, application

Description automatically generated**

**Order-Online:**

**Graphical user interface, text, email

Description automatically generated**

**Checkout:**

**Graphical user interface, application

Description automatically generated**

**Payments Page:(Razorpay)**

**Graphical user interface, application

Description automatically generated**

**Mobile view:(Responsive)**

Graphical user interface, application, website

Description automatically generated Graphical user interface, text, application, chat or text message

Description automatically generated

**MongoDB:**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**6. CONCLUSION**

The main objective of the project is to develop a Full fledge Zomato clone app. It is used for online food delivery. We have successfully used MongoDB database and successfully integrated it with our web application.

We have been retrieving data from the database without any errors and difficulties during the application execution.

We have learnt a lot of new things while building this application especially about MongoDB, Promises and various other database related topics.

This project meets all the requirements- authentication, integrity and confidentiality.

The future work for this might be including sort the items by popularity-high to low, and also filter item according to cuisines, cost and ratings.

**7.REFERENCES**

**React Documentation:**<https://reactjs.org/docs/getting-started.html>

**Node js Documentation:**

<https://nodejs.org/en/docs/>

**MongoDB Documentation:**

<https://www.mongodb.com/lp/video/awareness/getting-started?utm_content=rlsapostreg&utm_source=google&utm_campaign=gs_apac_rlsamulti_search_brand_dsa_atlas_desktop_rlsa_postreg&utm_term=&utm_medium=cpc_paid_search&utm_ad=&utm_ad_campaign_id=14412646494&adgroup=131761134852&gclid=Cj0KCQiA2ZCOBhDiARIsAMRfv9ITsFp75ol0b_vPAvw-8UnEcjHQeWAuXBFHpAw6n_rk7eI7tbY7UmUaAh9rEALw_wcB>

**Zomato Clone App:**

<https://zomatoclone-netlify.netlify.app/>