**Document for recipeController**

In this file we have described all the logic for backend for each get and post request we have added in routes.js file.

**require('../models/database'):** In this line, we are telling the recipeController to use database.js file which is inside models folder.

**const Category = require('../models/Category'):** In this line we are telling recipeController to use Category.js file as Category means in recipeController whenever we type Category it refers to the model and Schema we have defined inside Category.js file.

**const Recipe = require('../models/Recipe'):**  In this line we are telling recipeController to use Recipe.js file as Recipe means in recipeController whenever we type Recipe it refers to the model and Schema we have defined inside Recipe.js file.

One thing which I have used everywhere in this module is “**exports”.** It is an object which is implicitly created by nodejs. We can attach properties (functions, objects, values, etc.) and these properties will be available to other module which require or use this module.

**Code explanation for homepage:**

**exports.homepage = async(req, res)=>{**

**try{**

**const limitNumber = 5;**

**const categories = await Category.find({}).limit(limitNumber)**

**const latest = await Recipe.find({}).sort({\_id:-1}).limit(limitNumber);**

**const thai = await Recipe.find({category: "Thai"}).limit(limitNumber);**

**const american = await Recipe.find({category: "American"}).limit(limitNumber);**

**const chinese = await Recipe.find({category: "Chinese"}).limit(limitNumber);**

**const food = { latest, thai, american, chinese };**

**res.render('index', {title: 'Cooking Blog -Home', categories, food});**

**}catch(err){**

**res.status(500).send({message: err.message || "Error Occured"});**

**}**

**}**

**exports. homepage = async(req, res)=>{ }:** This line defines an async function “homepage” which is available to be exported to other modules which requires this module.

Now we are using try catch for error handling.

**const limitNumber = 5:**  This constant is used to limit the number of documents returned when we find() some documents.

**const categories = await Category.find({}).limit(limitNumber):**  Here, we are finding all the documents inside the Category model. This method returns an array of object containing documents which are getting stored in “categories” constant which we can say it to be an array.

This “categories” is used inside index.ejs to define categories section of webpage.

**const latest = await Recipe.find({}).sort({\_id:-1}).limit(limitNumber):** Here, we are finding all the documents on the basis of \_id which is given to them by mongoDB. We are using sort() method to get them in a sorted manner and ({\_id: -1}) means that we need to sort them in descending order.

Since, we are using this to display the latest recipe means those recipies which are added newly. So those recipes \_id will be larger than the \_id of previously uploaded recipes . Now we have sorted them in descending order so, they will store in “latest” constant which is a type of array and in “index.ejs” we are iterating through this array to display the recipe hence, the recipe which is added newly will be at 0th index and while iterating through this array we can display it at first.

This “latest” is used inside index.ejs to define latest section of webpage.

**const thai = await Recipe.find({category: "Thai"}).limit(limitNumber);**

**const american = await Recipe.find({category: "American"}).limit(limitNumber);**

**const chinese = await Recipe.find({category: "Chinese"}).limit(limitNumber);**

We are finding all the documents which matches the category and storing them in their respective arrays.

We are using these in index.ejs to define particular category sections.

**const food = { latest, thai, american, chinese }:**  In this line we are creating an object food which have all the constant that we are going to use in index.ejs.

**res.render('index', {title: 'Cooking Blog -Home', categories, food}):** In this line we are sending all the constant which we have declared and defined here to use in index.ejs . We have made food object so that we can pass all the constant at one go instead of passing one by one.

res.status(500).send({ message: err.message || "Error Occurred" }): If any errors occur within the try block, this line sends a response with a status code of 500 (Internal Server Error) along with an error message or a default message if no error message is available.

**Code explanation for exploreCategories:**

**exports.exploreCategories = async(req, res)=>{**

**try{**

**const limitNumber = 20;**

**const categories = await Category.find({}).limit(limitNumber);**

**res.render('categories', {title: 'Cooking Blog -Categories', categories});**

**}catch(err){**

**res.status(500).send({message: err.message || "Error Occured"});**

**}**

**}**

The explanation for this code is similar to the above code. This code is used to show page when we click on “View All” button in categories section at top.

**Code explanation for exploreCategoriesById:**

**exports.exploreCategoriesById = async(req, res)=>{**

**try{**

**let categoryId = req.params.id;**

**const limitNumber = 20;**

**const categoriesById = await Recipe.find({'category': categoryId}).limit(limitNumber);**

**res.render('categories', {title: 'Cooking Blog -Categories', categoriesById});**

**}catch(err){**

**res.status(500).send({message: err.message || "Error Occured"});**

**}**

**}**

This explanation for this code is also same as first code. This code is used to show page for particular categories when we click on them.

**Code explanation for exploreCategoriesById:**

exports.exploreRecipe = async(req, res)=>{

try{

let recipeId = req.params.id;

const recipe = await Recipe.findById(recipeId);

res.render('recipe', {title: 'Cooking Blog -Recipe', recipe});

}catch(err){

res.status(500).send({message: err.message || "Error Occured"});

}

}

The explanation for this code is same as previously explained code. This code is used to display the details of each recipe when we click on them.

**Code explanation for search:**

**exports.searchRecipe = async(req, res)=>{**

**try{**

**let searchTerm = req.body.searchTerm;**

**let recipe = await Recipe.find({$text: {$search: searchTerm, $diacriticSensitive: true}});**

**res.render('search', { title: 'Cooking Blog -Search', recipe});**

**}catch(err){**

**res.status(500).send({message: err.message || "Error Occured"});**

**}**

**}**

**let searchTerm = req.body.searchTerm:** This line is used to get the text which the user inputs into the search box and store it in searchTerm constant.

**let recipe = await Recipe.find({$text: {$search: searchTerm, $diacriticSensitive: true}}):** This line is used to search for the term in the Recipe model to find that recipe for which we have searched in database.

**$text:-** This is a special operator used for text-based searches in some databases like MongoDB. It allows for efficient full-text search capabilities.

**$search: searchTerm:-** Here, $search is a modifier in the $text operator signifies that we are performing search operation. searchTerm is the actual text for which we want to search.

**$diacriticSensitive: true:-** $diacriticSensitive is also a modifier on $text which is set to true which means it will treat text with diacritic and text without diacritic as separate text.

Diacritic means the means the accent, or mark on the word. For e.g.: ‘n’ and ‘ñ’. These both ‘n’ are different so database will also treat them as different.

**Code explanation for exploreLatest:**

**exports.exploreLatest = async(req, res)=>{**

**try{**

**const limitNumber = 20;**

**const recipe = await Recipe.find({}).sort({\_id:-1}).limit(limitNumber);**

**res.render('exploreLatest', {title: 'Cooking Blog -Explore Latest', recipe});**

**}catch(err){**

**res.status(500).send({message: err.message || "Error Occured"});**

**}**

**}**

This explanation of this code is same as previously explained code. This code is used to display the page when we click on explore latest button given on homepage.

**Code explanation for exploreRandom:**

**exports.exploreRandom = async(req, res)=>{**

**try{**

**let count = await Recipe.find().countDocuments();**

**let random = Math.floor(Math.random()\*count);**

**let recipe = await Recipe.findOne().skip(random).exec();**

**res.render('exploreRandom', {title: 'Cooking Blog -Explore Random', recipe});**

**}catch(err){**

**res.status(500).send({message: err.message || "Error Occured"});**

**}**

**}**

The explanation for this code is same as previously explained code. This code is used to display the random recipe page after we click on random recipe button available on homepage by first counting the number of documents in Recipe model and then choosing a random number between 1 and count.

**Code explanation for submitRecipeOnPost:**

**exports.submitRecipeOnPost = async (req, res) => {**

**try {**

**let imageUploadFile;**

**let uploadPath;**

**let newImageName;**

**if (!req.files || Object.keys(req.files).length === 0) {**

**console.log("No files were uploaded");**

**} else {**

**imageUploadFile = req.files.image;**

**newImageName = Date.now() + imageUploadFile.name;**

**uploadPath = require('path').resolve('./') + '/public/uploads/' + newImageName;**

**imageUploadFile.mv(uploadPath, function(err) {**

**if (err) {**

**return res.status(500).send(err);**

**}**

**});**

**}**

**const newRecipe = new Recipe({**

**name: req.body.name,**

**description: req.body.description,**

**email: req.body.email,**

**ingredients: req.body.ingredients,**

**category: req.body.category,**

**image: newImageName**

**});**

**await newRecipe.save();**

**req.flash('infoSubmit', 'Recipe has been added.');**

**res.redirect('/submitRecipe');**

**} catch (err) {**

**req.flash('infoError', err);**

**res.redirect('/submitRecipe');**

**}**

**}**

let imageUploadFile: Holds the uploaded image file.

let uploadPath**:** This specifies the path where the uploaded image file will be saved.

let newImageName**:** This holds the new name of the uploaded image.

req.files: This refers to the property of a request object in an Express application. When a client submits a form withf files this property holds information about the uploaded files.

Here, in if statement we are checking that if req.files == false which means no files were uploaded.

Object.keys(req.files): This part retrieves an array of keys(property names) from the req.files object. The Object.keys() function returns an array containing the names of all enumerable properties if the object.

Object.keys(req.files).length === 0: This part checks the length of retuned array if it is 0 it means no files were uploaded.

req.files.image: This line retrieves the image and store it in variable imageUploadFile.

Date.now() + imageUploadFile.name**:** This line retrieves the name of uploaded image and concatenate the current date to it and store it in newImageName.

**require('path').resolve('./') + '/public/uploads/' + newImageName:** In this whole line we are setting the whole path of the directory where we want to save the uploaded image.

**imageUploadFile.mv(uploadPath, function(err)):** imageUploadFile is the variable in which we have stored the uploaded image. uploadPath is the variable in which we have stored the path where we want to save the uploaded image.

.mv is a method which is used to move the uploaded file to desired location. Here, it is uploadPath.

Also we have a callback for errors if there will be an error it log the status and error message.

In next lines of code we are making a document by name “newRecipe” which will get saved to database into collection name “Recipe”.

await newRecipe.save(): In this line we save the document to our collection.

req.flash('infoSubmit', 'Recipe has been added.'): If the recipe is successfully submitted, a success flash message is stored using req.flash indicating that the recipe has been added.

The response is then redirected to the /submitRecipe page.

req.flash('infoError', err): If there is an error in submitting the recipe then a flash message will be stored in infoError object.

**Code explanation for submitRecipe flash messages:**

**exports.submit = async(req, res)=>{**

**const infoErrorsObj = req.flash('infoError');**

**const infoSubmitObj = req.flash('infoSubmit');**

**res.render('submitRecipe', {title: 'Cooking Blog -Submit Recipe', infoErrorsObj, infoSubmitObj});**

}

**const infoErrorsObj = req.flash('infoError');**

**const infoSubmitObj = req.flash('infoSubmit');**

These both lines retrieve the flash messages related to key infoError and infoSubmit objects and store them in infoErrorsObj and infoSubmitObj and later on these are used in ejs file to display the flash message.