Tunis Business School
IT325 Web Services Course

IT 325 Final Project

Recipe Provider RESTful API

FORKy

Author

Rahma Charfddeine

Supervisor: DR. Montassar Ben Messaoud

date: January 15, 2023

Table of Contents

List of Figures

Abstract

A recipe-providing application is a tool that allows users to search for and discover new recipes based on criteria such as ingredient and cuisine type. The application can also provide users with detailed instructions on how to prepare and cook the recipes, as well as nutritional information and suggestions for ingredient substitutions.

Users can also save and organize their favorite recipes, and even create and share their own recipes with the community. The application can be accessed through a web-browser or a mobile app, making it easy for users to find recipes and cook them at home.

With its wide variety of recipes, detailed instructions, and convenient features, the recipe-provider application is a useful tool for both experienced and novice cooks.

Chapter 1

Introduction

My final Project for the IT325 Web Services course this semester consists of a RESTful API named “FORKy” developed using Node.js, Express, JWT.

FORKy name is inspired from the word fork , a reference to the experience of eating

The resources were stored thanks to the implementation of MongosDB I have also used technologies such as Postman, VSCode, and Git-Version Control to maximize the project’s quality. This project contains all the CRUD Operations, secured with JWT token authentication

Its main aim is to provide a set of recipes designed to help in cooking with available ingredients

The project’s function is returning a desired recipe to the user based on the ingredients that he already has at his kitchen and avoiding the burden of going out to buy groceries just to cook something familiar.

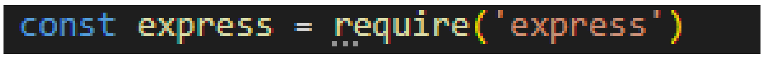
Chapter 2

Explanation of the work carried out

2.1 Node.js/EXPRESS Contribution

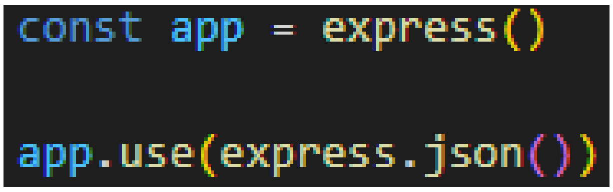
Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. [1] Express is a back end web application framework for Node.js, released as free and open source software under the MIT License. It is designed for building web applications andAPIs. [2]
I have used the Express framework for Node.js to develop my app.

Implementation code :



```
const express = require('express')
```

Figure 2.1: importing express



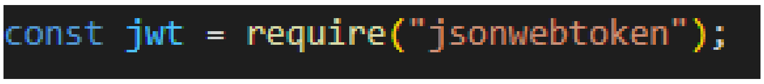
```
const app = express()

app.use(express.json())
```

Figure 2.2: express use

2.2 JWT Contribution

JSON Web Token is a proposed Internet standard for creating data with optional signature and/or optional encryption whose payload holds JSON that asserts some number of claims. The tokens are signed either using a private secret or a public/private key. [3]

I have used JSON Web Token authentication to secure the API access.
Implementation code :

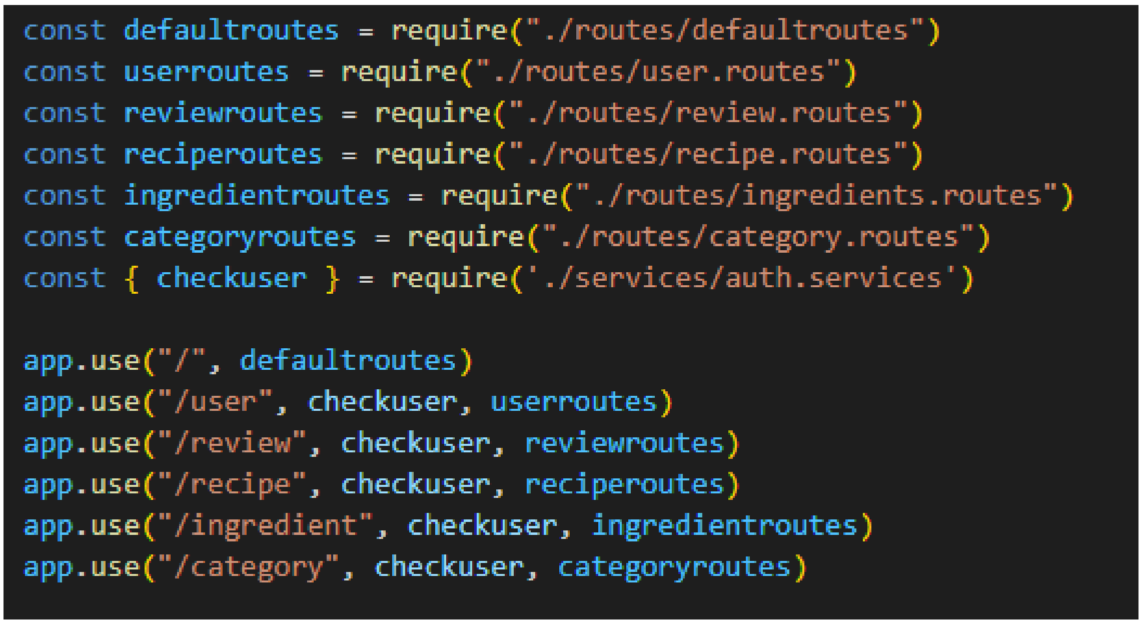


```
const jwt = require("jsonwebtoken");
```

Figure 2.3: Json Web Token

2.3 Routing in Express.js

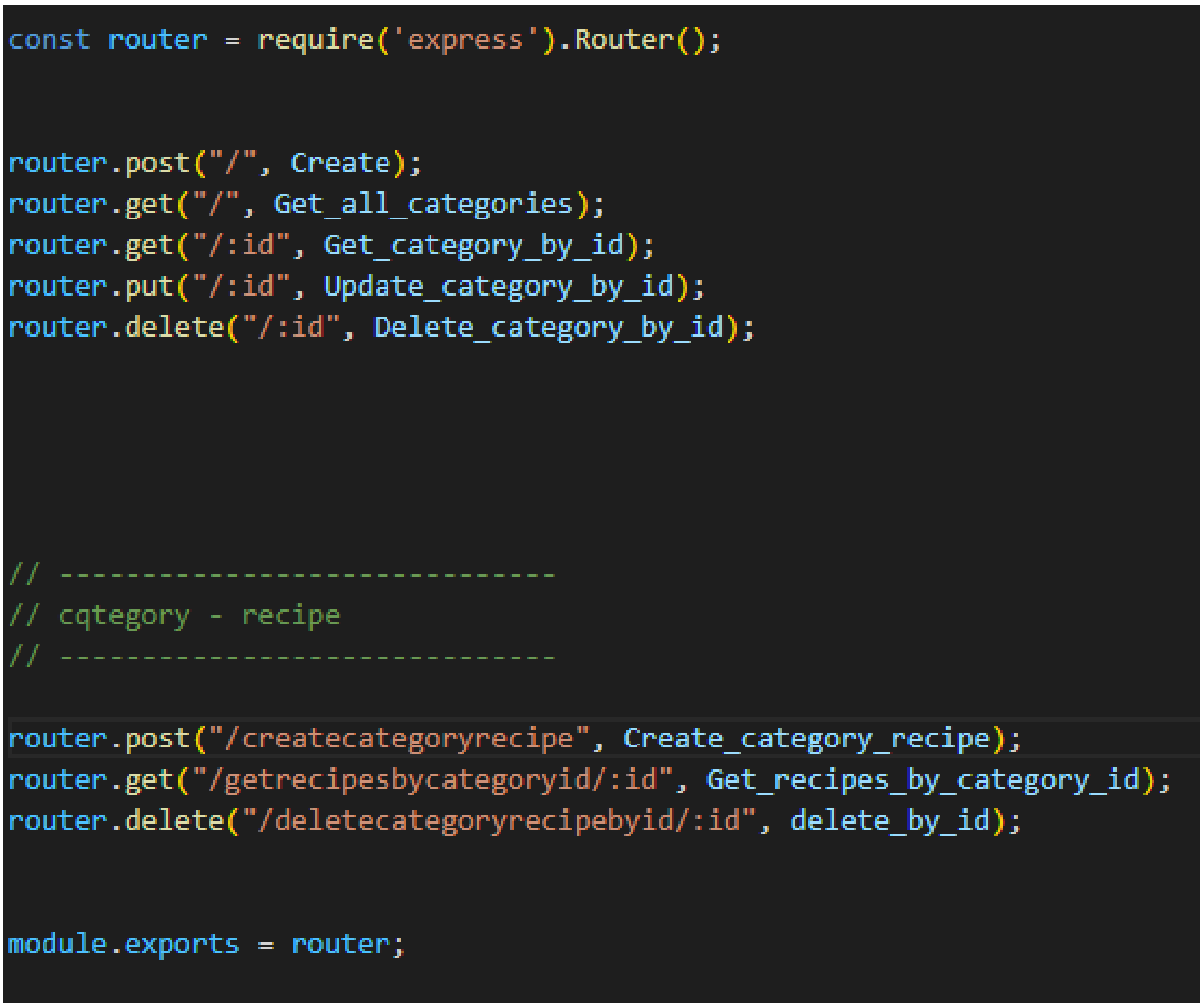
Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).Each route can have one or more handler functions, which are executed when the route is matched.[4] Implementation code :



```
const defaultroutes = require("./routes/defaultroutes")
const userroutes = require("./routes/user.routes")
const reviewroutes = require("./routes/review.routes")
const reciperoutes = require("./routes/recipe.routes")
const ingredientroutes = require("./routes/ingredients.routes")
const categoryroutes = require("./routes/category.routes")
const { checkuser } = require('./services/auth.services')

app.use("/", defaultroutes)
app.use("/user", checkuser, userroutes)
app.use("/review", checkuser, reviewroutes)
app.use("/recipe", checkuser, reciperoutes)
app.use("/ingredient", checkuser, ingredientroutes)
app.use("/category", checkuser, categoryroutes)
```

Figure 2.4: routing section in server.js

In each Route I created a set of end points related to a specific class
Here an example of routing in “./routes/category.routes”



```
const router = require('express').Router();


router.post("/", Create);
router.get("/", Get_all_categories);
router.get("/:id", Get_category_by_id);
router.put("/:id", Update_category_by_id);
router.delete("/:id", Delete_category_by_id);





// ------------------------------
// cqtegory - recipe
// ------------------------------

router.post("/createcategoryrecipe", Create_category_recipe);
router.get("/getrecipesbycategoryid/:id", Get_recipes_by_category_id);
router.delete("/deletecategoryrecipebyid/:id", delete_by_id);


module.exports = router;
```

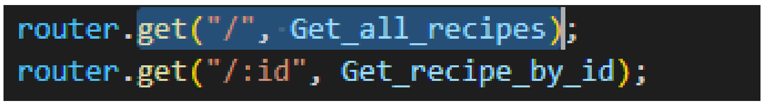
Figure 2.5: example of routing "./routes/category.routes".

Chapter 3

the HTTP Methods used

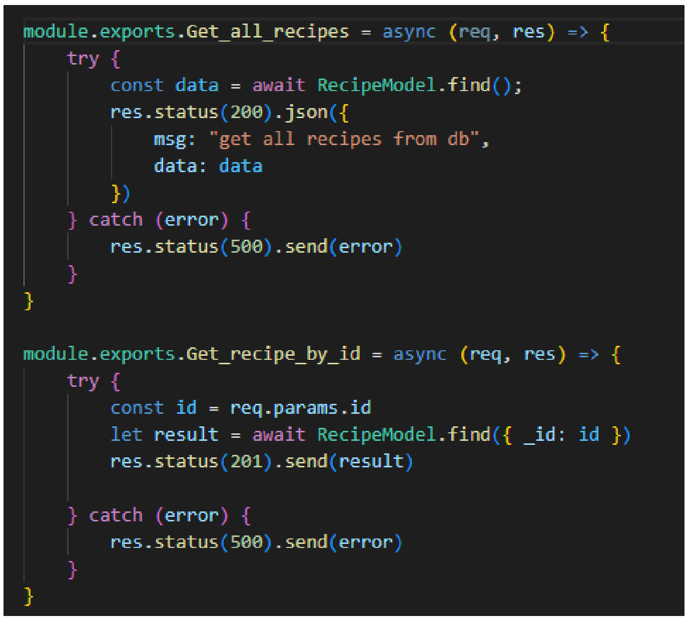
3.1 GET Request Functions

Example : recipe
The get (Get-all-recipes) endpoint will call the function (Get-all-recipes) from recipe.services file. We will set the entire set of the existent recipes with all detailed attributes.
The get (Get-recipe-by-id) endpoint will call the function (Get-recipe-by-id) from recipe.services file. We will get the recipe-id ,recipe-name, time, difficulty and instructions of a specific recipe by specifying its id. . .



```
router.get("/", Get_all_recipes);
router.get("/:id", Get_recipe_by_id);
```

Figure 3.1: Get endpoint



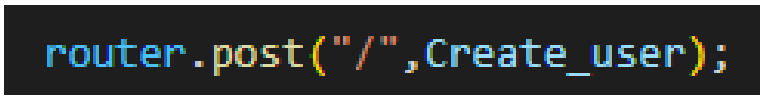
```
module.exports.Get_all_recipes = async (req, res) => {
    try {
        const data = await RecipeModel.find();
        res.status(200).json({
            msg: "get all recipes from db",
            data: data
        })
    } catch (error) {
        res.status(500).send(error)
    }
}

module.exports.Get_recipe_by_id = async (req, res) => {
    try {
        const id = req.params.id
        let result = await RecipeModel.find({ _id: id })
        res.status(201).send(result)

    } catch (error) {
        res.status(500).send(error)
    }
}
```

Figure 3.2: get functions

3.2 Post Request Functions

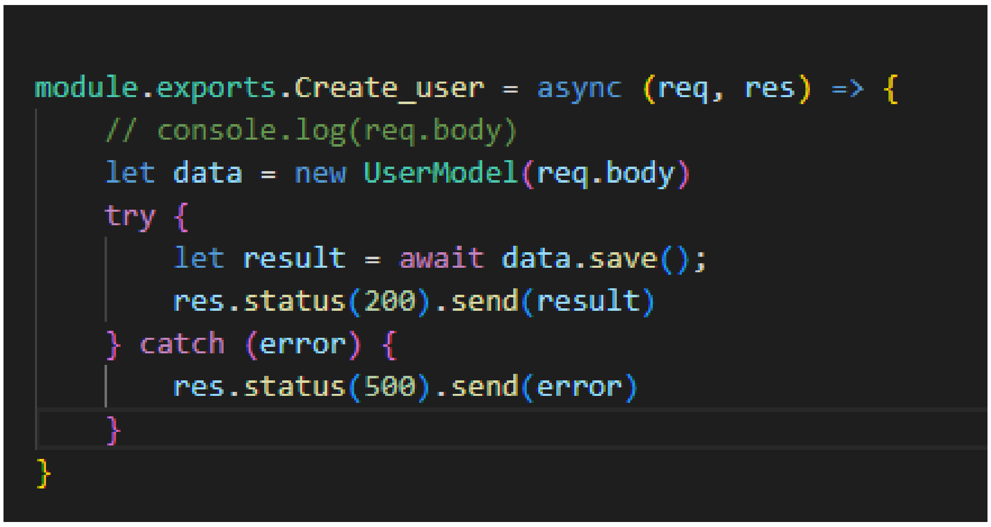
Example : user

The post (Create-user) endpoint will call the function (Create-user) from user.services file. We will create a new user by entering all attributes and it will be saved in the data base



```
router.post("/",Create_user);
```

Figure 3.3: Post endpoint



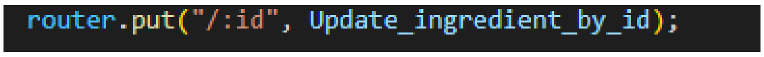
```
module.exports.Create_user = async (req, res) => {
    // console.log(req.body)
    let data = new UserModel(req.body)
    try {
        let result = await data.save();
        res.status(200).send(result)
    } catch (error) {
        res.status(500).send(error)
    }
}
```

Figure 3.4: Create function

3.3 Put Request Functions

Example : user

The put (Update-ingredient-by-id) endpoint will call the function (Update-ingredient-by-id) from ingredient.services file. We will update an existing ingredient by entering its id then changing An attribute or many .

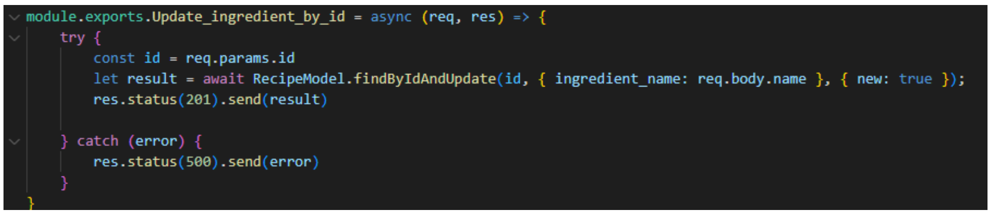


```
router.put("/:id", Update_ingredient_by_id);
```

Figure 3.5: Put endpoint

3.4 Delete Request Functions

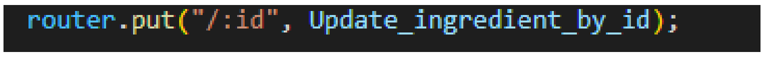
Example : ingredient-recipe (the class linking the ingredient and the recipe)



```
module.exports.Update_ingredient_by_id = async (req, res) => {
    try {
        const id = req.params.id
        let result = await RecipeModel.findByIdAndUpdate(id, { ingredient_name: req.body.name }, { new: true });
        res.status(201).send(result)

    } catch (error) {
        res.status(500).send(error)
    }
}
```

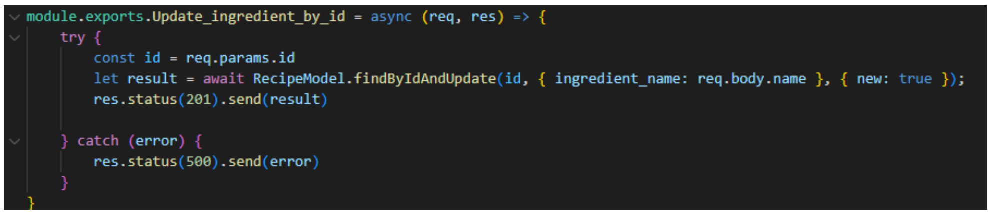
Figure 3.6: Update function

The delete (delete-by-id) endpoint in ingredient.routes file will call the function (delete-by-id) from ingredient-recipe.services file.This function will delete any linkage between a recipe and an ingredient using the id of the linkage



```
router.put("/:id", Update_ingredient_by_id);
```

Figure 3.7: Delete endpoint



```
module.exports.Update_ingredient_by_id = async (req, res) => {
    try {
        const id = req.params.id
        let result = await RecipeModel.findByIdAndUpdate(id, { ingredient_name: req.body.name }, { new: true });
        res.status(201).send(result)

    } catch (error) {
        res.status(500).send(error)
    }
}
```

Figure 3.8: Delete function

Chapter 4

Security Layer

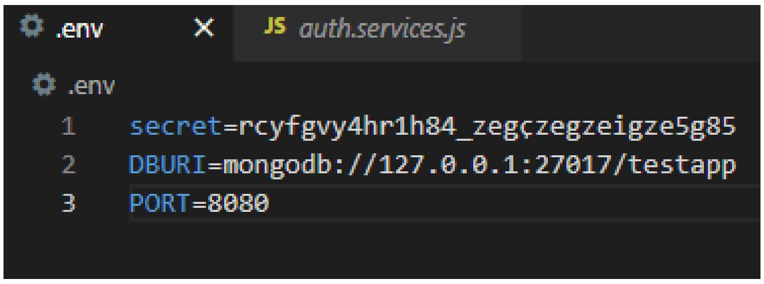
4.1 environment variables

You can access environment variables in Node. js right out of the box. When your Node. js process boots up, it'll automatically provide access to all existing environment variables by creating an env object within the process global object

For large projects with even larger teams, code-base changes constantly and so does the configuration. So documentation is key to keep anyone who uses the code-base later.[5]

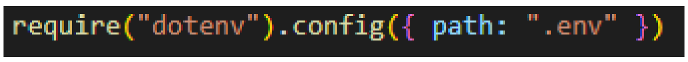
.env file contains all the environment variables (credentials) in key-value format used by the developer that would rather stay unseen for others, especially when the project is used or developed by many developers . We find here the secret key, the port used and the connection to the database

The dotenv package is required for handling environment variables



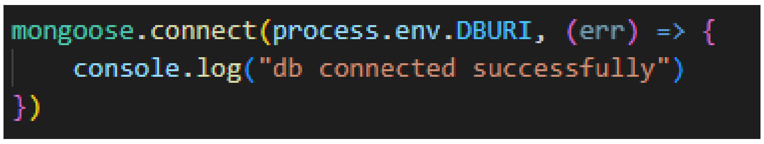
```
secret=rcyfgvy4hr1h84_zegçzegzeigze5g85
DBURI=mongodb://127.0.0.1:27017/testapp
PORT=8080
```

Figure 4.1: .env file



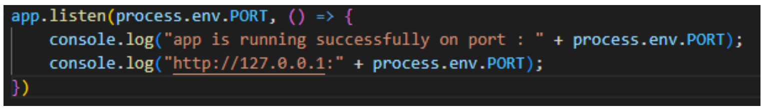
```
require("dotenv").config({ path: ".env" })
```

Figure 4.2: enviroment part 1



```
mongoose.connect(process.env.DBURI, (err) => {
    console.log("db connected successfully")
})
```

Figure 4.3: enviroment part 2



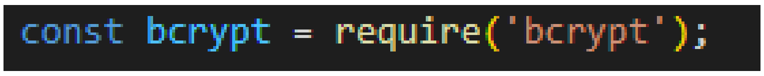
```
app.listen(process.env.PORT, () => {
    console.log("app is running successfully on port : " + process.env.PORT);
    console.log("http://127.0.0.1:" + process.env.PORT);
})
```

Figure 4.4: enviroment part 3

4.2 generating id and hashing password

bcrypt is a password-hashing function designed by Niels Provos and David Mazières. Besides incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power.[6]

The bcrypt function was used twice:first, to generate the id of the user using genSalt(). Second to hash the password with a combination of the password and the id



```
const bcrypt = require('bcrypt');
```

Figure 4.5: bcrypt library

4.3 Token's cookies using JWT

The cookies contained in an HTTP request's request headers are parsed using a cookie parser.

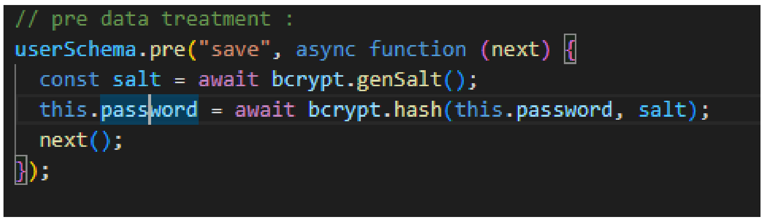
A client (such a web browser) can include information about any cookies it has previously obtained from a server in the form of name-value pairs when the client sends a request to the server. This information is extracted by the cookie parser and provided to the server-side program for use in maintaining user sessions, monitoring user preferences, and other functions.

On the client side, session data is likewise stored in cookies and is used to keep track of the user across requests. Also, it is used to monitor user behavior on the website and customize their experience.

The library cookieParser is used

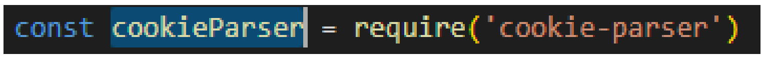
To login , the user will provide his email and password . The email will be checked if it is already existent or not . Once, the email is found , a comparison between the provided password hashed and the email's password will verify if the user is really who he claims to be. Using JWT, a token with the id , secret key and expiration period is received setting a cookie on the server-side.

Once logged, the token is cleared.



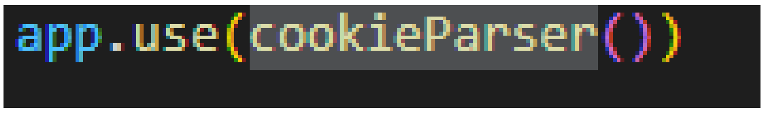
```
// pre data treatment :
userSchema.pre("save", async function (next) {
  const salt = await bcrypt.genSalt();
  this.password = await bcrypt.hash(this.password, salt);
  next();
});
```

Figure 4.6: Generating id and password with bcrypt



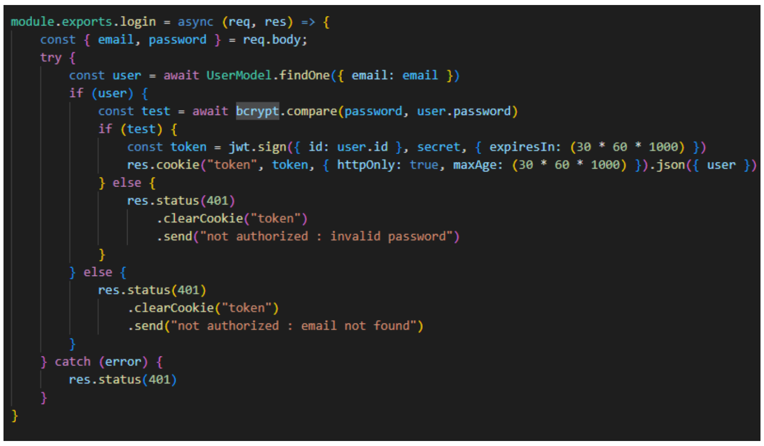
```
const cookieParser = require('cookie-parser')
```

Figure 4.7: CookieParser library



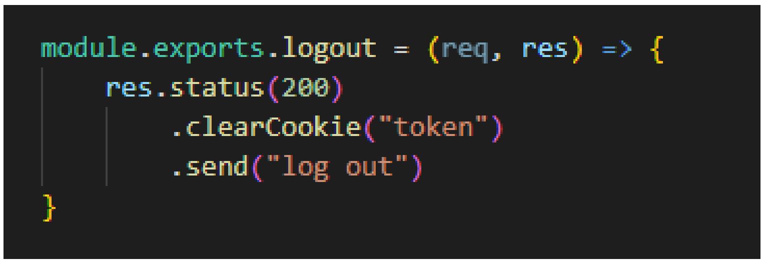
```
app.use(cookieParser())
```

Figure 4.8: CookieParser use



```
module.exports.login = async (req, res) => {
    const { email, password } = req.body;
    try {
        const user = await UserModel.findOne({ email: email })
        if (user) {
            const test = await bcrypt.compare(password, user.password)
            if (test) {
                const token = jwt.sign({ id: user.id }, secret, { expiresIn: (30 * 60 * 1000) })
                res.cookie("token", token, { httpOnly: true, maxAge: (30 * 60 * 1000) }).json({ user })
            } else {
                res.status(401)
                    .clearCookie("token")
                    .send("not authorized : invalid password")
            }
        } else {
            res.status(401)
                .clearCookie("token")
                .send("not authorized : email not found")
        }
    } catch (error) {
        res.status(401)
    }
}
```

Figure 4.9: Authorization function



```
module.exports.logout = (req, res) => {
    res.status(200)
        .clearCookie("token")
        .send("log out")
}
```

Figure 4.10: Logging out

Chapter 5

Postman Contribution

Postman is an application used for API testing. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated. [8] I have used Postman for the automatic testing of my API requests

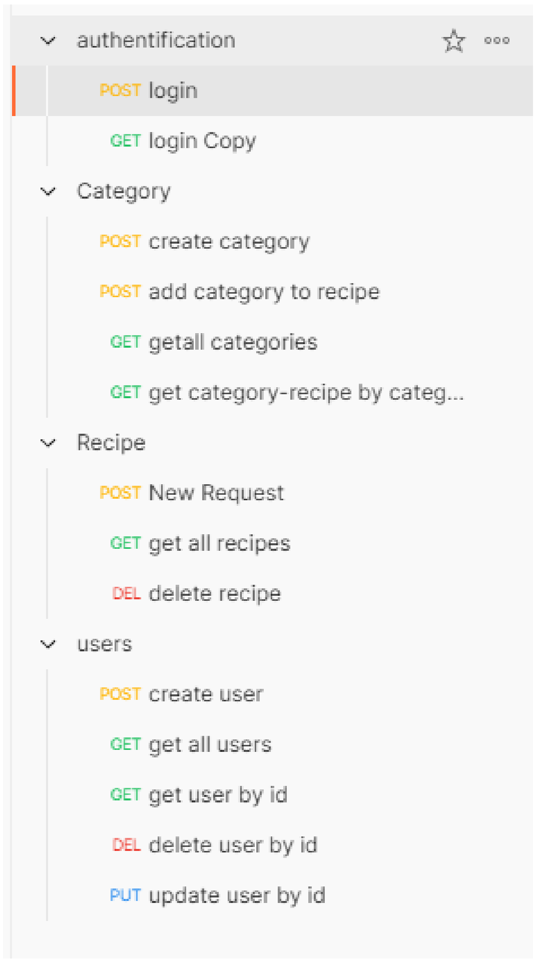


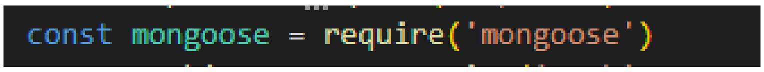
Figure 5.1: Postman tests

Chapter 6

Mongoose Contribution

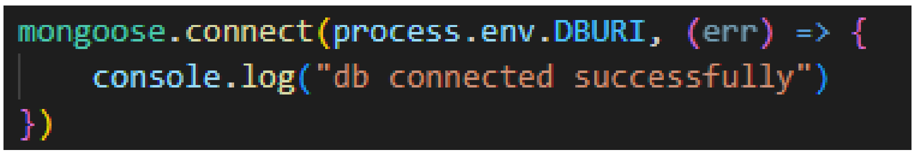
Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node. js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB. MongoDB is a schema-less NoSQL document database

MongoDB is a popular NoSQL database, Flexible and easy to use



```
const mongoose = require('mongoose')
```

Figure 6.1: importing Mongoose



```
mongoose.connect(process.env.DBURI, (err) => {
    console.log("db connected successfully")
})
```

Figure 6.2: connecting to monogoDB

Chapter 7

UML (class diagram)

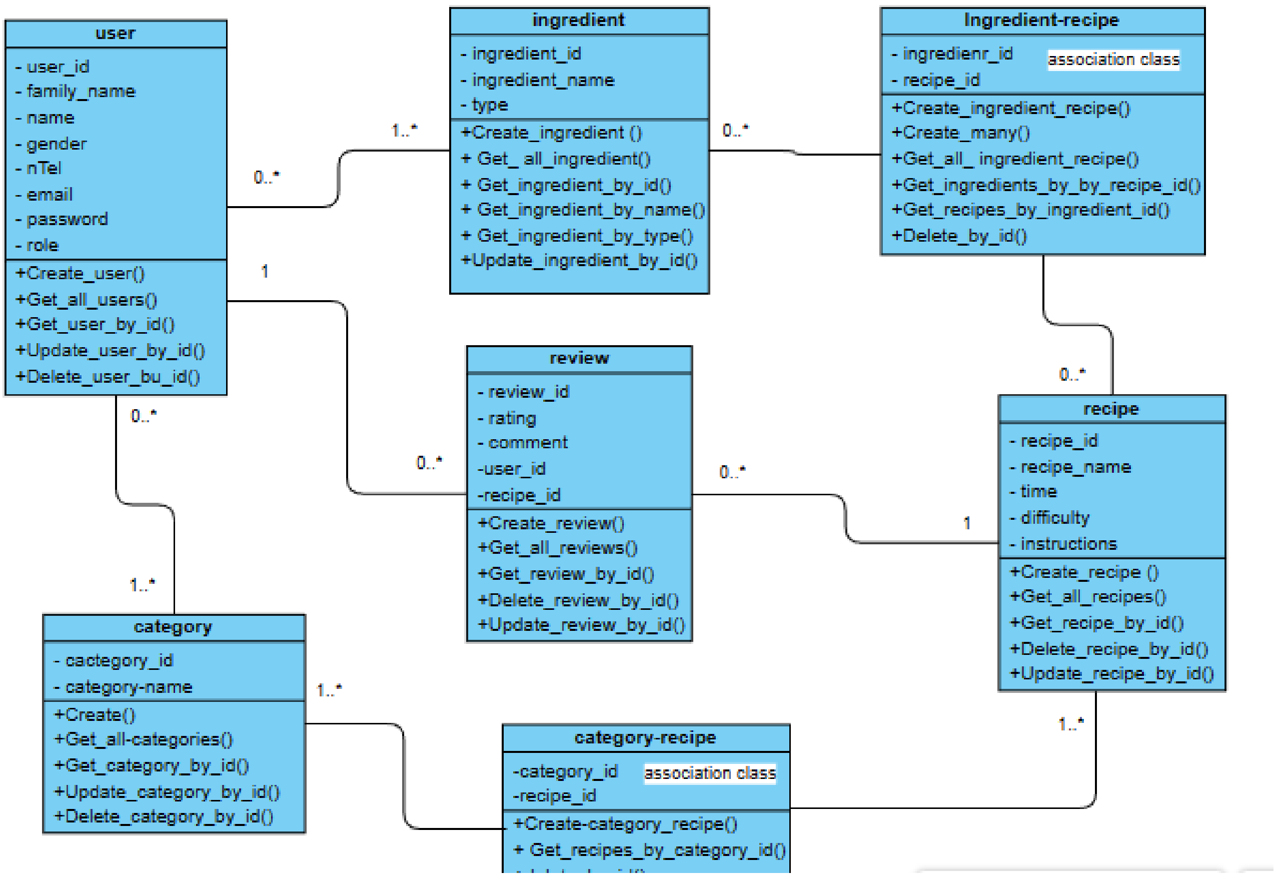


Figure 7.1: Class Diagram of FORKy

Chapter 8

Conclusion

This recipe-providing app's goal is to give users access to a selection of cooking recipes as well as step-by-step instructions for preparing and cooking the meals.

Also, it may offer functions like meal planning, ingredient lists and the capacity to save and arrange favorite recipes.

As a student living on my own, I personally had a difficult in preparing my daily meals depending on what i have in my fridge or not having enough time to go shopping for groceries With such an app , preparing a dish with what you already has never been this easy and became a delightful experience.

This project is sill in the first phase and can be improved using different techniques .
FORKy has the potential to become one of the best food recipes applications .

References

- (1) “Node.js.” https://en.wikipedia.org/wiki/Node.js. [Online; accessed 01-Jan-2023].
- (2) “Express.” https://en.wikipedia.org/wiki/Express.js.[Online; accessed 01-Jan-2023].
- (3) “Jwt.” https://en.wikipedia.org/wiki/JSON-Web-Token. [Online; accessed 01-Jan-2023].
- (4) “Routing .” https://expressjs.com/en/starter/basic-routing.html. [Online; accessed 01-Jan-2023].
- (5) “.env .”https://dev.to/mrsauravsahu/documenting-env-files-for-nodejs-projects-3a9j.[Online; accessed 01-Jan-2023].
- (6) “.bcrypt .”https://en.wikipedia.org/wiki/Bcrypt. [Online; accessed 01-Jan-2023].
- (7) G. Romero, “What is postman api test,” encora, p. 1, June 2021.
- (8) Nick Karnik “Introduction to Mongoose for MongoDB “www.freecodecamp.org /FEBRUARY 11, 2018