Restaurant Management System

Backend API Documentation

1. File Handling

The Backend development includes mainly two folders **Prisma** which includes the database schema whereas the other folder **Src** includes the API development codes. The **Src** folder also has been divided into two subfolders **handlers and modules**. The **handlers** folder includes the JavaScript file of task operation for employee, employee authentication, food items, user, order, and get Total. The **module** folder includes the authentication and authentication checker. Beside that **Src** folder also includes the database(db) , server, and route JavaScript Files.

Server :

D: > level 5 > sem2 > Resturant management > src > JS server.js > ...

```
1   import express from "express";
2   import morgan from "morgan";
3   import cors from "cors";
4   import { signIn, signUp } from "./handlers/user.js";
5   import router from "./route.js";
6   import { signInEmployee } from "./handlers/employeeAuth.js";
7   import {
8     getTotalEmployees,
9     getTotalFoodItems,
10    getTotalSales,
11  } from "./handlers/getTotal.js";
12
13  const app = express();
14
15  app.use(cors());
16  app.use(express.json());
17  app.use(morgan("dev"));
18  app.use(express.urlencoded({ extended: true }));
19
20  // For admin
21  app.post("/signup", signUp);
22  app.post("/signin", signIn);
23  app.get("/getTotalEmployees/:id", getTotalEmployees);
24  app.get("/getTotalFoodItems/:id", getTotalFoodItems);
25  app.get("/getTotalSales/:id", getTotalSales);
26  // For emloyee
27  app.post("/signinemployee", signInEmployee);
28
29  app.use("/order", router);
30  app.use("/employee", router);
31  app.use("/fooditem", router);
32
33  app.use("/file", router);
```

In the **server**, JavaScript file the API for the Sign up and Sign in has been created .

In Details,

```
import express from "express";
import morgan from "morgan";
import cors from "cors";
import { signIn, signUp } from "./handlers/user.js";
import router from "./route.js";
import { signInEmployee } from "./handlers/employeeAuth.js";
import {
  getTotalEmployees,
  getTotalFoodItems,
  getTotalSales,
} from "./handlers/getTotal.js";
```

Express, morgan, and cors are imported which are the node.js frameworks. Express is used for the HTTPS server and router handling, Morgan for logging HTTP requests and responses, and the Cors for the Cross-Origin Resource Sharing requests.

The SignIn and signUp function is imported from the file path of handelers/user.js for user login and user registration.

The signInEmployee function is imported from the path handlers/employeeAuth.js for authenticating the email and password of the user.

The imports of functions getTotalEmployee, getTotalFoodItems, and getTotalSales from the path handelers/getTotal.js for the total numbers of row data stored in the respective table.

```
app.post("/signup", signUp);
```

In this the data from the user that is taken is inserted into the database table with the app. post method. The signUp function will perform to register a new admin user account.

- Request Body

Field		Type	Description
Id	(Required)	String	Unique Id of the admin @default(uuid())
Email	(Required)	String	Admin email
restaurant_name	(Required)	String	Restaurant name to register
Username	(Required)	String	Username for log in
password	(Required)	String	Password of the employee account log in
phone_number	(Required)	String	Phone number of an employee
Employee	(Required)	String	Employee designation or Role
createdAt	(Required)	DateTime	Date and time when the account was created.

			@default(now( ))
Adminref	(Required)	User	Reference of the admin
Adminid	(Required)	String	Id of the admin

```
app.post("/signin", signIn);
```

- Parameter

Field		Type	Description
Id	(Required)	String	Unique Id of the Order

```
app.get("/getTotalEmployees/:id", getTotalEmployees);
app.get("/getTotalFoodItems/:id", getTotalFoodItems);
app.get("/getTotalSales/:id", getTotalSales);
```

The above code uses the app. get method to get the total number of row data from the database tables using the function name.

```
app.use("/order", router);
app.use("/employee", router);
app.use("/fooditem", router);

app.use("/file", router);
```

app.use() with a specific URL pattern tells Express.js to use the corresponding route handler for any request that matches that pattern.

Router :

```
import { Router } from "express";

import {
  createEmployee,
  deleteEmployee,
  getEmployee,
  getEmployees,
  updateEmployee,
} from "./handlers/employee.js";
import {
  createBill,
  createOrder,
  getBilling,
  getOrder,
  getUnserved,
  updateOrder,
  updateServe,
  getComplete,
```

```
} from "./handlers/order.js";
import {
  createFoodItem,
  updateFoodItem,
  getFoodItems,
  deleteFoodItem,
} from "./handlers/fooditem.js";

const router = Router();

router.post("/placeorder", createOrder);
router.get("/getorder/:id", getOrder);
router.put("/updateorder", updateOrder);
router.put("/updateServe", updateServe);
router.put("/createBill", createBill);
router.get("/getBilling/:id", getBilling);
router.get("/unserved/:id", getUnserved);
router.get("/complete/:id", getComplete);

router.post("/createemployee", createEmployee);
router.get("/getemployee", getEmployee);
router.get("/getemployees/:id", getEmployees);
router.delete("/deleteemployee/:id", deleteEmployee);
router.put("/updateemployee", updateEmployee);

router.post("/createfooditem", createFoodItem);
router.put("/updatefooditem", updateFoodItem);
router.get("/getfooditems/:id", getFoodItems);
router.delete("/deletefooditem/:id", deleteFoodItem);

export default router;
```

In the **Router** , JavaScript file the CRUD operation is performed for employee, order and food items .

In Details,

```
import {
  createEmployee,
  deleteEmployee,
  getEmployee,
  getEmployees,
  updateEmployee,
} from "./handlers/employee.js";
import {
```

```
  createBill,
  createOrder,
  getBilling,
  getOrder,
  getUnserved,
  updateOrder,
  updateServe,
  getComplete,
} from "./handlers/order.js";
import {
  createFoodItem,
  updateFoodItem,
  getFoodItems,
  deleteFoodItem,
} from "./handlers/fooditem.js";
```

First, we imported various functions from the various handlers into the code above. primarily from the employee, order, and fooditems JavaScript files of the three handlers file to perform the CRUD operation.

```
router.post("/placeorder", createOrder);
router.get("/getorder/:id", getOrder);
router.put("/updateorder", updateOrder);
router.put("/updateServe", updateServe);
router.put("/createBill", createBill);
router.get("/getBilling/:id", getBilling);
router.get("/unserved/:id", getUnserved);
router.get("/complete/:id", getComplete);
router.post("/createemployee", createEmployee);
router.get("/getemployee", getEmployee);
router.get("/getemployees/:id", getEmployees);
router.delete("/deleteemployee/:id", deleteEmployee);
router.put("/updateemployee", updateEmployee);

router.post("/createfooditem", createFoodItem);
router.put("/updatefooditem", updateFoodItem);
router.get("/getfooditems/:id", getFoodItems);
router.delete("/deletefooditem/:id", deleteFoodItem);
```

In the above code the router.post , router.put, router.get methods are used to create, update and get the data from the database table respectively.

The method includes two parameters the first one inside “ ” is the URL path and when the method is called the URL will be trigger this route handler., the second parameter is the function name which will be executed then the request is made to that route.

- **Order List Management**

```
router.post("/placeorder", createOrder);
```

The line of code will take place a new order by the waiter from the customer and will insert into the database table .

- Request Body

Field		Type	Description
Id	(Required)	String	Unique Id of the Order @default(uuid())
food_name	(Required)	String	Name of the food item
Quantity	(Required)	Int	Quantity of the food item
Description	(Required)	String	Food Item description
Timestamp	(Required)	DateTime	Date when order was placed @default(now( ))
table_number	(Required)	Int	Table number where the order was placed
Price	(Required)	Float	Total Food item price
isCompleted	(Required)	Boolean	Is the order completed? @default(false)
isServed	(Required)	Boolean	Is the order served to the customer? @default(false)
isBilled	(Required)	Boolean	Is the order billing is done ? @default(false)
Adminref	(Required)	User	Reference of the admin
Adminid	(Required)	String	Id of the admin

```
router.get("/getorder/:id", getOrder);
```

This line of code will get the order detail data from the database table to the employee.

- Parameter

Field		Type	Description
Id	(Required)	String	Unique Id of the Order

```
router.put("/updateorder", updateOrder);
```

This line of code is used to update the order that had been already placed.

- Request Body

Field		Type	Description
Id	(Required)	String	Unique Id of the Order

			@default(uuid())
food_name	(Required)	String	Name of the food item
Quantity	(Required)	Int	Quantity of the food item
Description	(Required)	String	Food Item description
Timestamp	(Required)	DateTime	Date and time when order was placed @default(now( ))
table_number	(Required)	Int	Table number where the order was placed
Price	(Required)	Float	Total Food item price
isCompleted	(Required)	Boolean	Is the order completed? @default(false)
isServed	(Required)	Boolean	Is the order served to the customer? @default(false)
isBilled	(Required)	Boolean	Is the order billing is done ? @default(false)
Adminref	(Required)	User	Reference of the admin
Adminid	(Required)	String	Id of the admin

```
router.put("/updateServe", updateServe);
```

This line of code is used to update whether the order is served to the customer or not.

- Request Body

Field		Type	Description
isServed	(Required)	Boolean	Is the order served to the customer? @default(false)

```
router.put("/createBill", createBill);
```

This line of code will update the bill status.

- Request Body

Field		Type	Description
isBilled	(Required)	Boolean	Is the order billing is done ? @default(false)

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

This line of code will view the total bill to the screen from the database table using the unique id of the customer order.

- Parameter

Field		Type	Description
Id	(Required)	String	Unique Id of the Order

```
router.get("/unserved/:id", getUnserved);
```
This line of code is used to view the unserved food order from the database table to the employee screen.

- Parameter

Field		Type	Description
Id	(Required)	String	Unique Id of the Order

```
router.get("/complete/:id", getComplete);
```
This line of code will view all the completed food order items to the employee screen.

- Parameter

Field		Type	Description
Id	(Required)	String	Unique Id of the Order

- **Employee Management**

```
router.post("/createemployee", createEmployee);
```
The above code is used to add or insert the new employee to the restaurant database table.

- Request Body

Field		Type	Description
Id	(Required)	String	Unique Id of the Employee @default(uuid())
Email	(Required)	String	Employee email for log in
f_name	(Required)	String	First name of the employee
l_name	(Required)	String	Last name of the employee
password	(Required)	String	Password of the employee account log in
phone_number	(Required)	String	Phone number of an employee

work_as	(Required)	String	Employee designation or Role
createdAt	(Required)	DateTime	Date and time when the account was created. @default(now( ))
Adminref	(Required)	User	Reference of the admin
Adminid	(Required)	String	Id of the admin

```
router.get("/getemployee", getEmployee);
```

The above code is used to get the total employee data from the database table.

- Request Body

Field		Type	Description
Id	(Required)	String	Unique Id of the Employee
Email	(Required)	String	Employee email for log in
f_name	(Required)	String	First name of the employee
l_name	(Required)	String	Last name of the employee
password	(Required)	String	Password of the employee account log in
phone_number	(Required)	String	Phone number of an employee
work_as	(Required)	String	Employee designation or Role
createdAt	(Required)	DateTime	Date and time when the account was created. @default(now( ))
Adminref	(Required)	User	Reference of the admin
Adminid	(Required)	String	Id of the admin

```
router.get("/getemployees/:id", getEmployees);
```

The above code is used to get the employees data using their unique ID from the database table.

- Parameter

Field		Type	Description
Id	(Required)	String	Unique Id of the Employee

```
router.delete("/deleteemployee/:id", deleteEmployee);
```

The above code is to delete a specific employee from the restaurant database table through their unique employee id.

- Parameter

Field		Type	Description
Id	(Required)	String	Unique Id of the Employee


```
router.put("/updateemployee", updateEmployee);
```

The above code is for the updating of the data of existing employees into the database table.

- Request Body

Field		Type	Description
Id	(Required)	String	Unique Id of the Employee @default(uuid())
Email	(Required)	String	Employee email for log in
f_name	(Required)	String	First name of the employee
l_name	(Required)	String	Last name of the employee
password	(Required)	String	Password of the employee account log in
phone_number	(Required)	String	Phone number of an employee
work_as	(Required)	String	Employee designation or Role
createdAt	(Required)	DateTime	Date and time when the account was created @default(now( ))
Adminref	(Required)	User	Reference of the admin
Adminid	(Required)	String	Id of the admin

- **Food Item Management**

```
router.post("/createfooditem", createFoodItem);
```

The above code is used to add the new food item to the database table.

- Request Body

Field		Type	Description
Id	(Required)	String	Unique Id of the Food Item @default(uuid())
food_name	(Required)	String	Name of the food item
category	(Required)	String	Category of the food item
price	(Required)	Float	Price for the food item
ingredients		String	Ingredients used in the food item
imgSrc		String	Picture of the food item
isTrending		Boolean	Is the food item trending? @default(false)
Adminref	(Required)	User	Reference of the admin

Adminid	(Required)	String	Id of the admin

```
router.put("/updatefooditem", updateFoodItem);
```

The above code is used to update the existing food item data to the database table.

- Request Body

Field		Type	Description
Id	(Required)	String	Unique Id of the Food Item @default(uuid())
food_name	(Required)	String	Name of the food item
category	(Required)	String	Category of the food item
price	(Required)	Float	Price for the food item
ingredients		String	Ingredients used in the food item
imgSrc		String	Picture of the food item
isTrending		Boolean	Is the food item trending? @default(false)
Adminref	(Required)	User	Reference of the admin
Adminid	(Required)	String	Id of the admin

```
router.get("/getfooditems/:id", getFoodItems);
```

The above code will show the specific food item details using its unique food item id from the database.

- Parameter

Field		Type	Description
Id	(Required)	String	Unique Id of the food item

```
router.delete("/deletefooditem/:id", deleteFoodItem);
```

The above code will delete the specific food Item from the database of the restaurant using its unique food item id.

- Parameter

Field		Type	Description
Id	(Required)	String	Unique Id of the food item