Lab07: App Development with Oracle Database

Objective

This Lab will serve as an introductory guide to Web Development for you to start working on your semester projects for Database Systems. It entails the necessary installations required, the crucial components of a web-app, how we use Oracle database to connect to our application and perform CRUD (Create, Read, Update, and Delete) operations, how does a backend work, and how do we connect it to our front-end. A compact OLTP System that communicates with an Oracle Database has been provided.

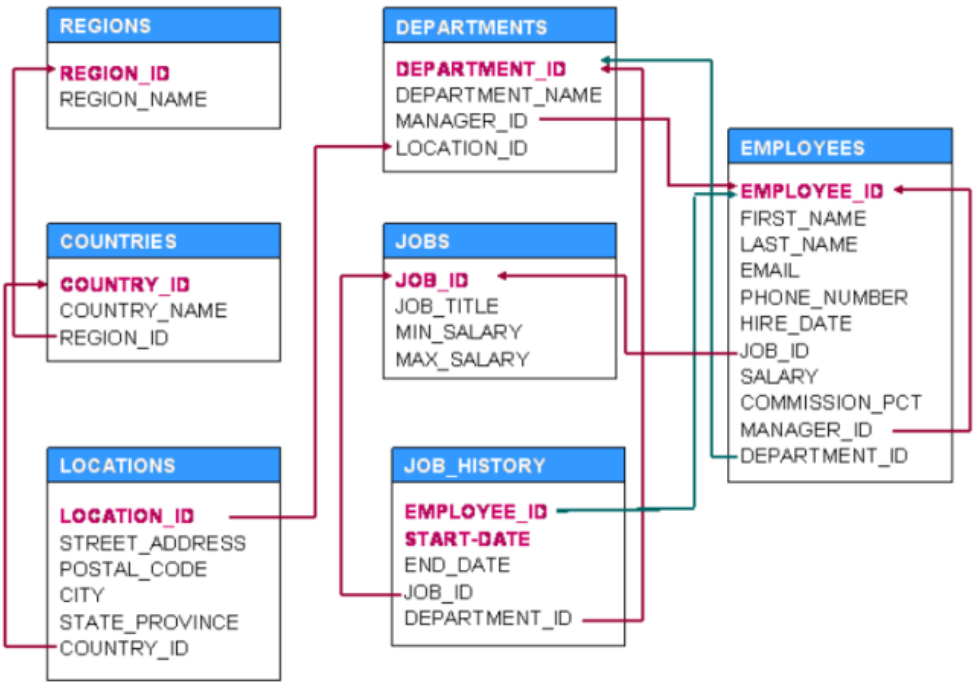
Acknowledgement

This lab was developed in collaboration with our course teaching assistants, Muhammad Shehzad, Saad Lakhani, and Abdul Rafay.

Business Scenario

The business scenario for this Lab will be an Employee-Management System, using the HR Schema provided in the previous labs. This simple application includes CRUD operations on employees for a HR User.

Relational Schema



Prerequisites

Install Node.js (v20 LTS)

Follow these steps to install Node.js version 20 LTS:

Windows:

1. Go to the official Node.js website: https://nodejs.org
2. Click the **Download** button for the **LTS version (v20)**.
3. Once downloaded, run the installer.
4. Follow the prompts in the setup wizard, ensuring you select "Add to PATH".

After installation, verify by opening Command Prompt and typing:

```
node -v
```

You should see version **20.x.x**.

MacOS:

1. Open the Terminal.
2. Install Node.js via Homebrew by running the following commands:

```
brew update
brew install node@20
```

After installation, verify by opening Command Prompt and typing:

```
node -v
```

You should see version **20.x.x**.

Install Git

Windows:

1. Download Git from the official website: https://git-scm.com
2. Run the downloaded installer and follow the setup wizard.

To verify the installation, open Command Prompt and type:

```
git --version
```

You should see the Git version installed.

MacOS:

1. Open the Terminal.

Install Git using Homebrew by typing:

```
brew install git
```

2. Verify the installation by typing:

```
git --version
```

Install Postman

1. Visit the official Postman website: https://www.postman.com/downloads/
2. Download the version for your operating system (Windows, MacOS, or Linux).
3. Install the downloaded file.
4. Launch Postman and sign in with your account.

Setup Oracle Database

Assuming Oracle Database is already installed. Make sure you have a User already setup with HR schema loaded into it. If not, please refer to the steps mentioned in Lab 1.

Theory

We use a lot of web-applications in our daily lives, we interact with those applications through the user interface, and then it processes our request and gives us a result. We will do a deep dive into the working of our application, what happens behind the scenes when you click on a button?

When you click on a button in the front-end, it sends an API call to the backend server with the relevant information required to fulfill that request, such as an Employee ID.

For our application, we are following the MVC Architecture, so for example, if we sent an update employee request, it would go to our application, and then it would route it to the employee route as the request concerns an employee, then there, the employee router checks the request and identifies it as an updateEmployee request and routes it forward to the updateEmployee controller, the updateEmployee controller calls the Model function for updating the employee (written in the EmployeeModel file in the

models folder), and returns a confirmation if the employee was updated, and gives an error if the employee is not updated.

How does the Model function work?

First and foremost, the function establishes a connection with our oracle database, and then it runs an SQL Query on the Oracle database to update the employee on the given employee id that was passed along with the request from the frontend as a parameter, and it returns a confirmation on how many rows were updated, which is passed on to the frontend as a confirmation for you to see.

Backend

We are creating our backend using Node.js and Express. We will learn how to setup the backend, how backend APIs are used and created and how they interact with the database.

Step 1: Cloning the Repository & opening it in code editor

1. Open your terminal, make sure git is installed.
2. **Clone the Repository:** To get started, you'll need to clone the repository containing the backend code. Open your terminal and run the following command:

```
git clone https://github.com/ShehzadAslamOza/lab7-dbms-app
```

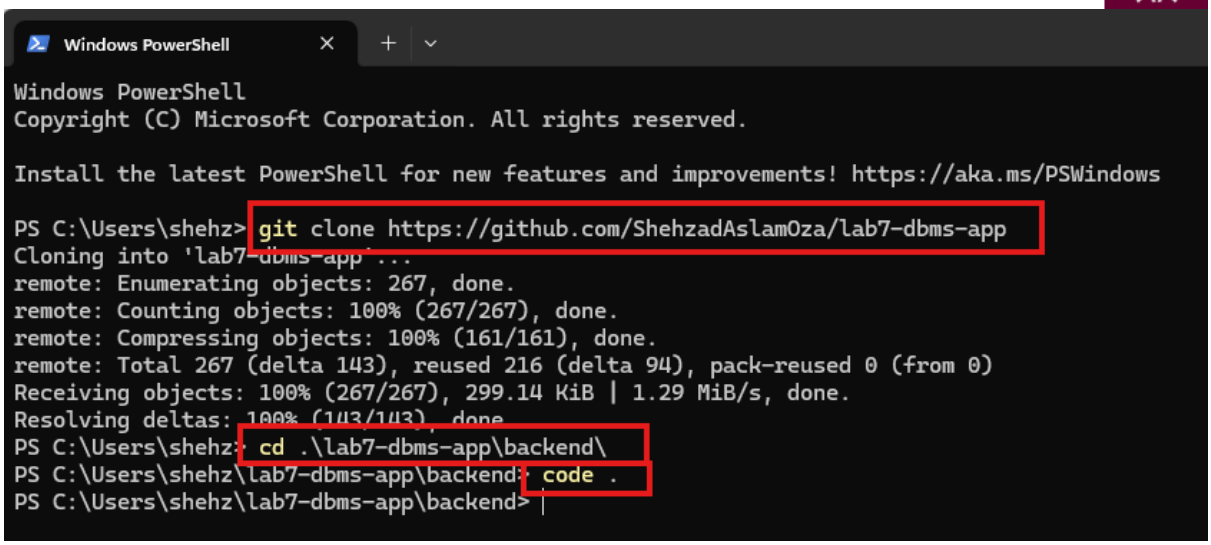
3. **Navigate to the Backend Folder:** After cloning the repository, navigate to the backend folder where all the backend code resides:

```
cd lab7-dbms-app/backend
```

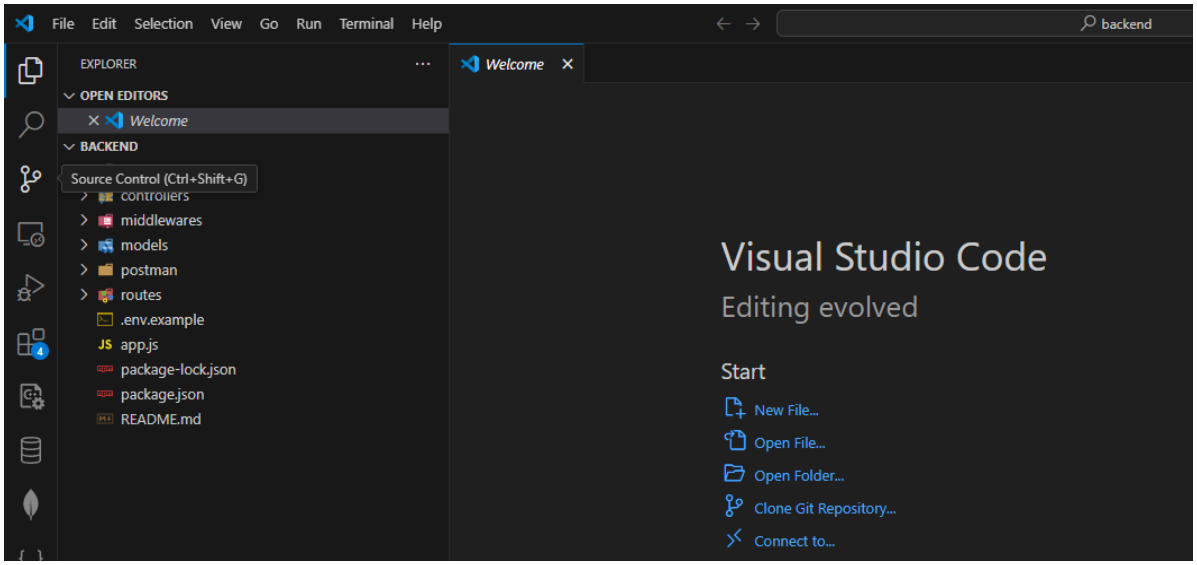
4. Open the folder in Vscode or any editor: To open it in Vscode type

```
code .
```

5. **Demo:** Running the commands



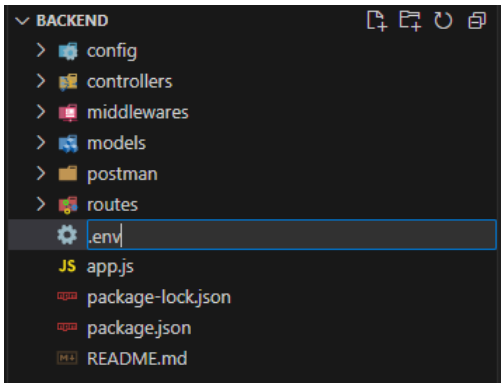
6. **Demo**: Backend Folder opened in Vscode.



Step 2: Rename .env.sample to .env

.env contains credentials of our backend so it is best practice never to push it on github. Hence we only provide the .env.sample to be later converted to .env

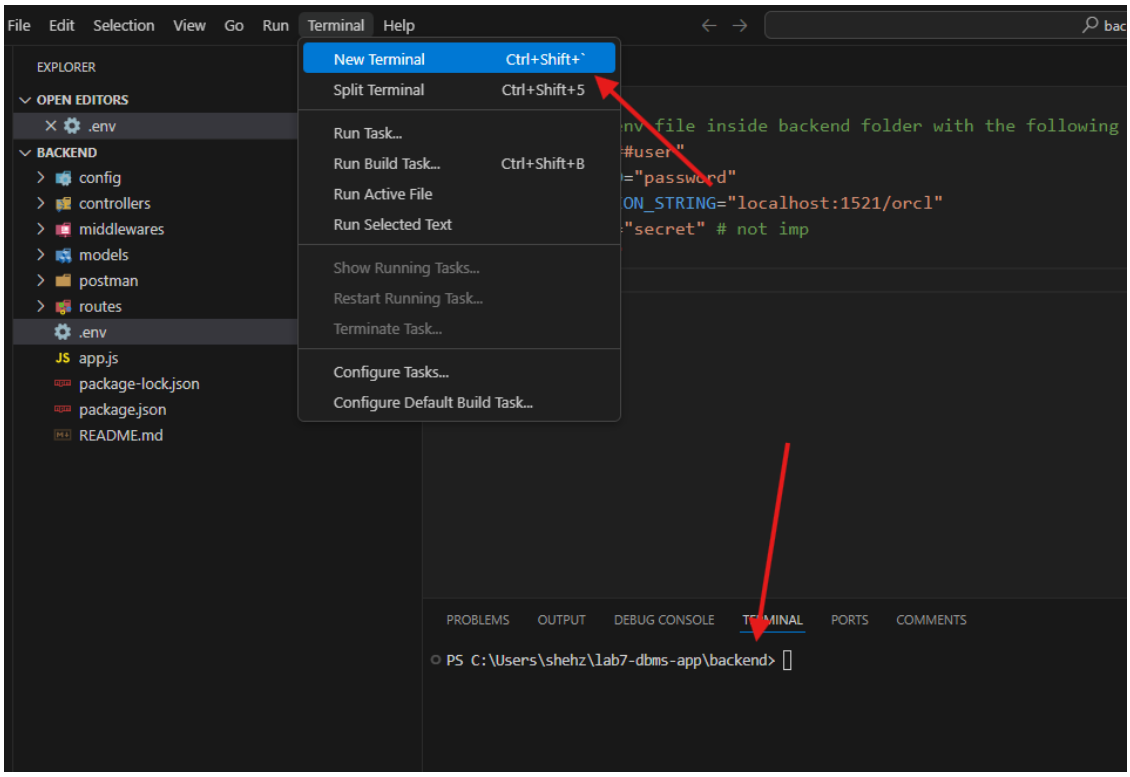
1. **Renamed**



2. **Change .env variables:** Change **DB_USER**, **DB_PASSWORD**, **DB_CONNECTION_STRING** with the values you use for connection with oracle.

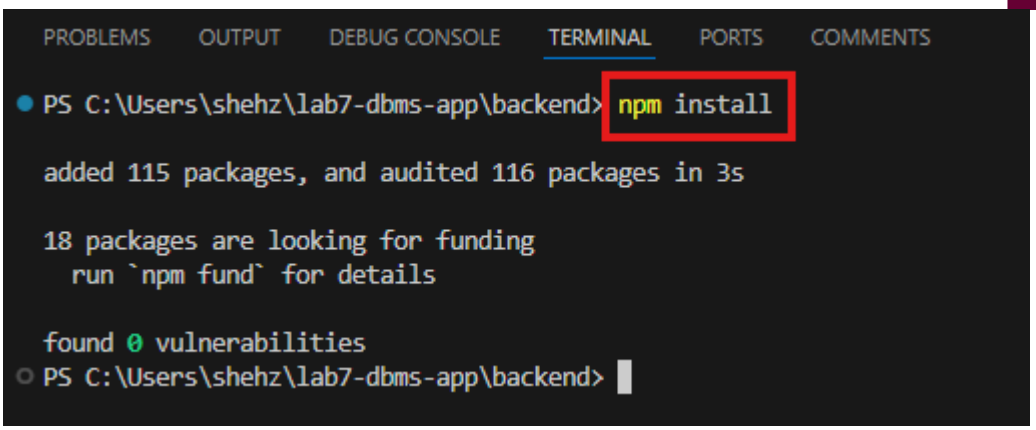
Step 3: Starting the Backend

1. Open terminal within vscode

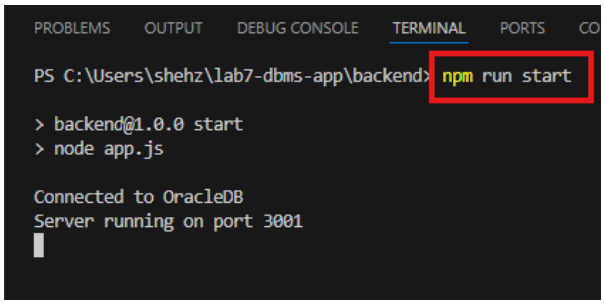


2. **Type 'npm install',** this installs all the node libraries required by the backend. Make sure you have node installed

```
npm install
```



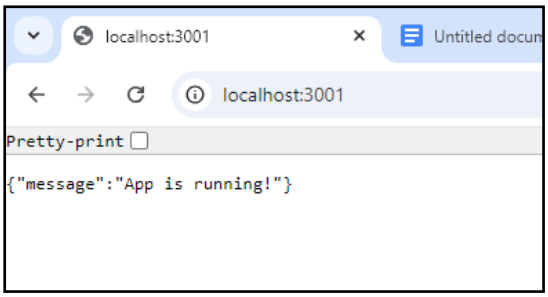
3. **Type 'npm run start'** on terminal to start the backend



The backend has now been started, lets play with the apis.

Step 4: The App is running

1. **Navigate to any browser and type** http://localhost:3001/



The backend is up and running

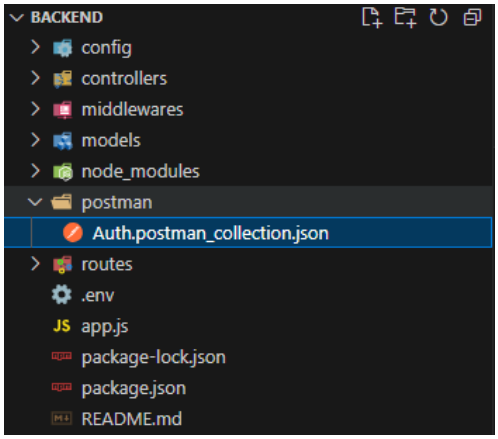
2. **Now type** http://localhost:3001/api/employees to see list of all employees in hr schema.

So you were interacting with the backend APIs. Every frontend uses these APIs to exchange information. But while developing the app, it is better that we use POSTMAN to interact with the

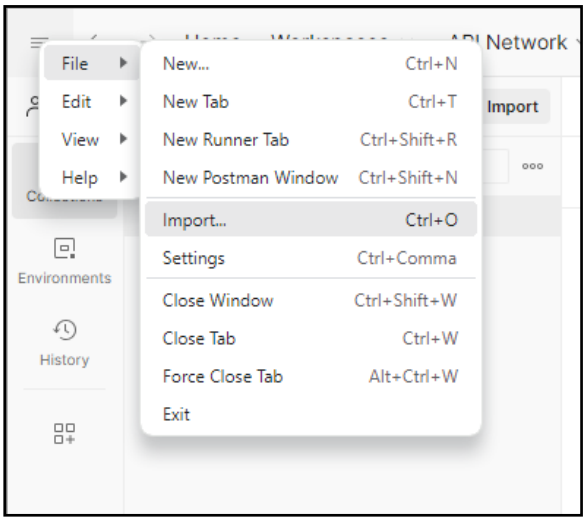
Step 5: Using Postman to Interact with the APIs

Now that your backend is set up, we'll use Postman to interact with the APIs. Postman allows you to send requests to your backend and receive responses, which makes it easy to test the functionality of each API.

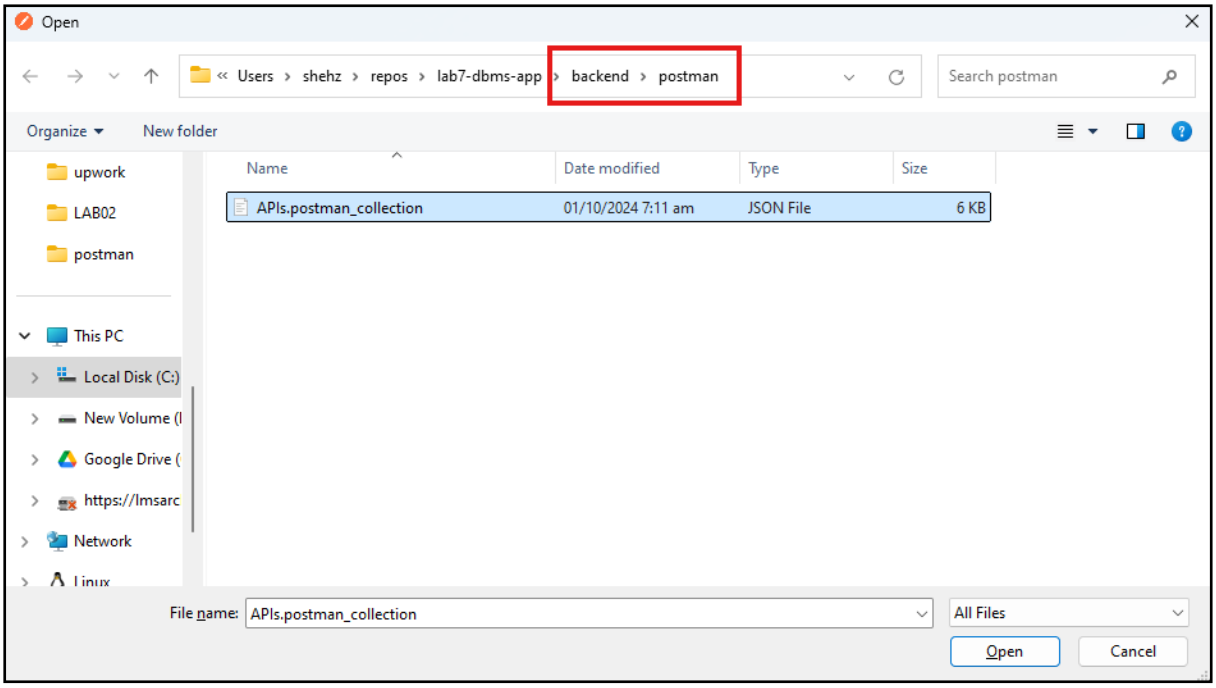
1. Make sure you have postman installed. Open it and Login/Sign In with your gmail.
2. There is a postman folder in the backend folder, we need to import the configuration in that to the postman



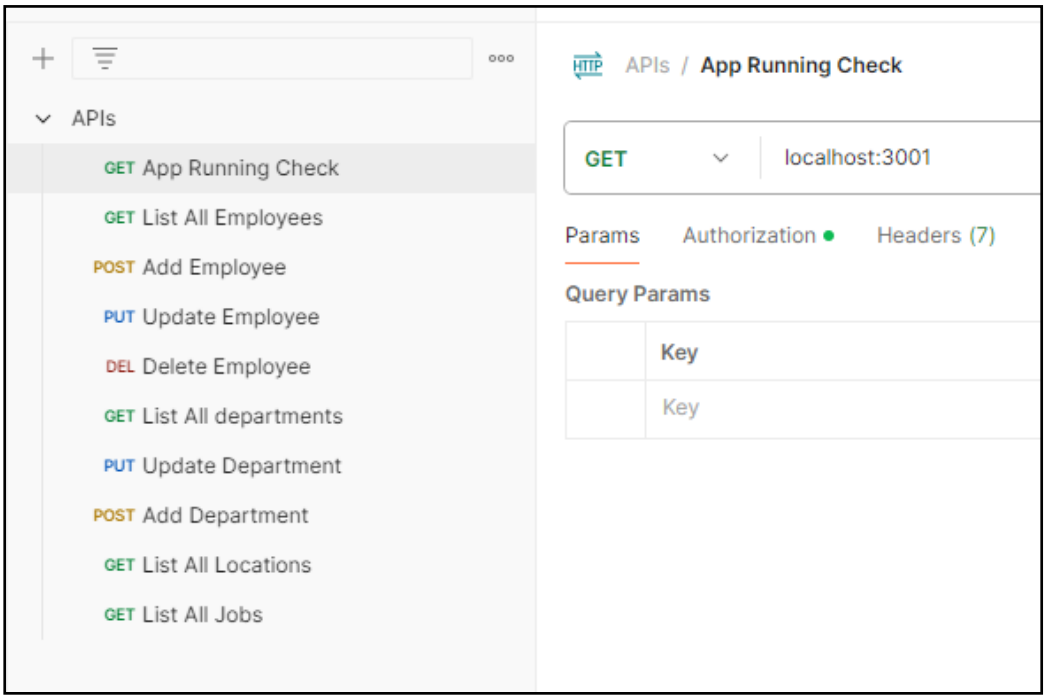
3. Select Top Left Button on Postman, then Select File>Import



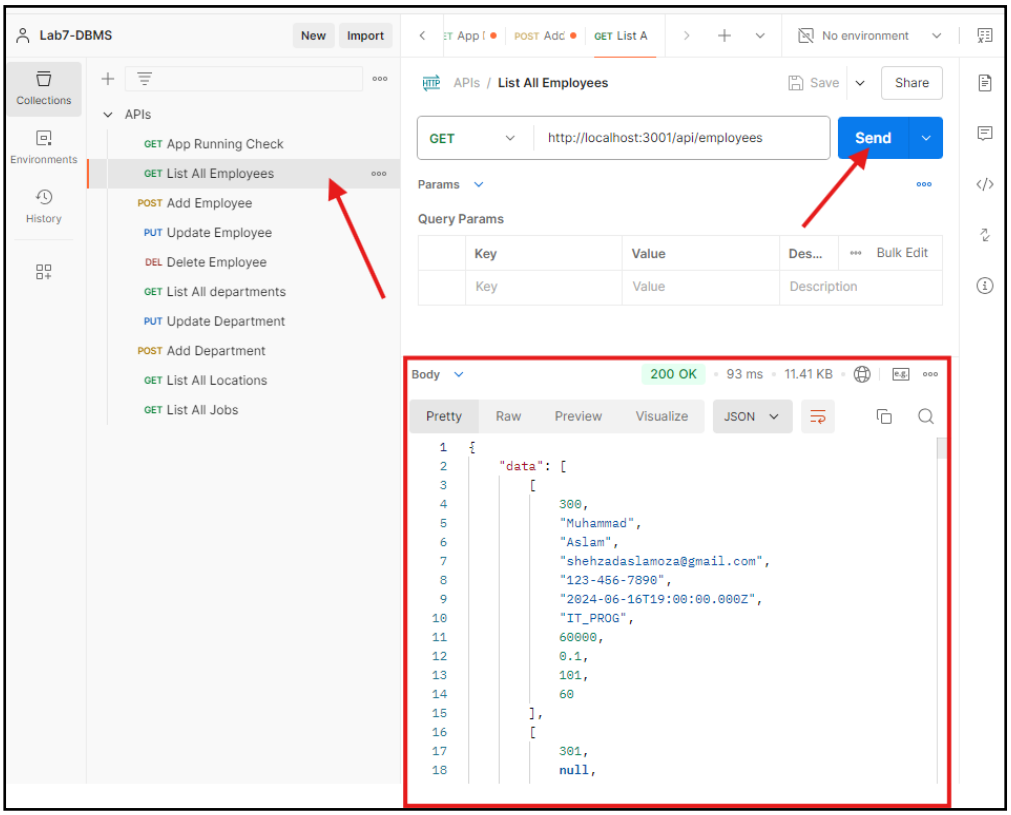
4. Navigate to the postman folder and import the collection



5. After importing you will see all the apis of this app listed



6. Select List All Employees and run



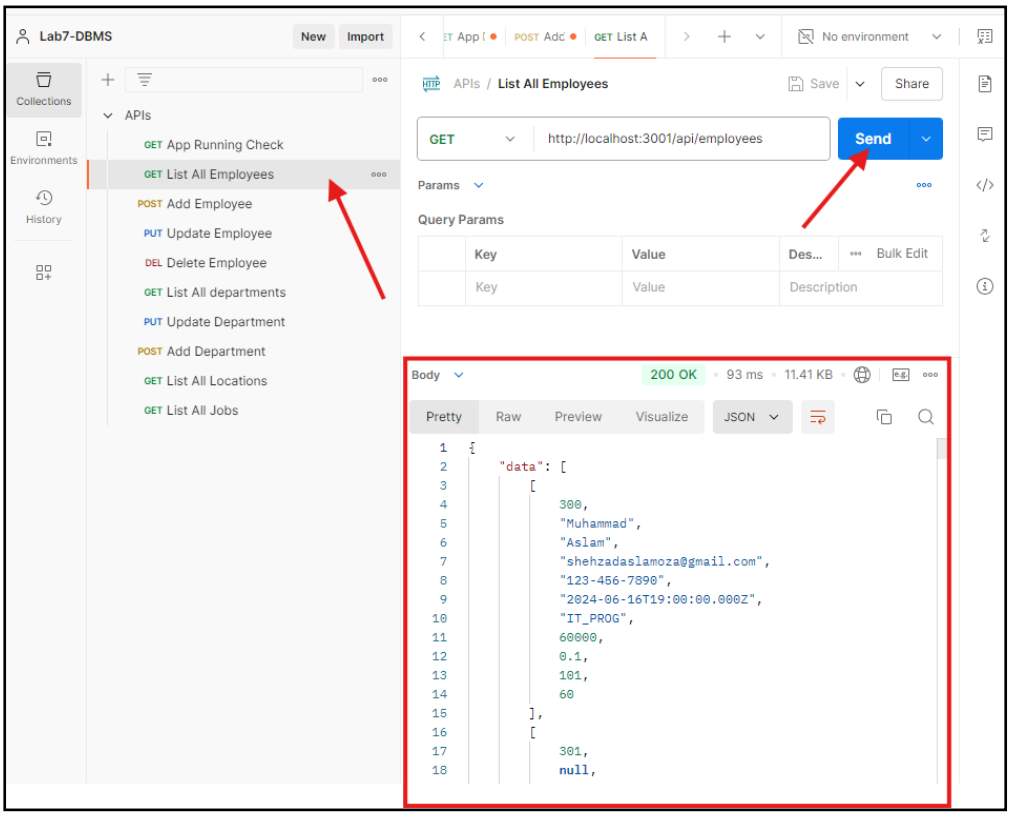
You will see the same data as you saw while typing http://localhost:3001/api/employees in the browser.

Step 6: Understanding APIS

In this backend, you'll interact with four types of APIs: **GET**, **POST**, **PUT**, and **DELETE**. These are the core HTTP methods used for CRUD (Create, Read, Update, Delete) operations, which correspond to how data is managed in a database.

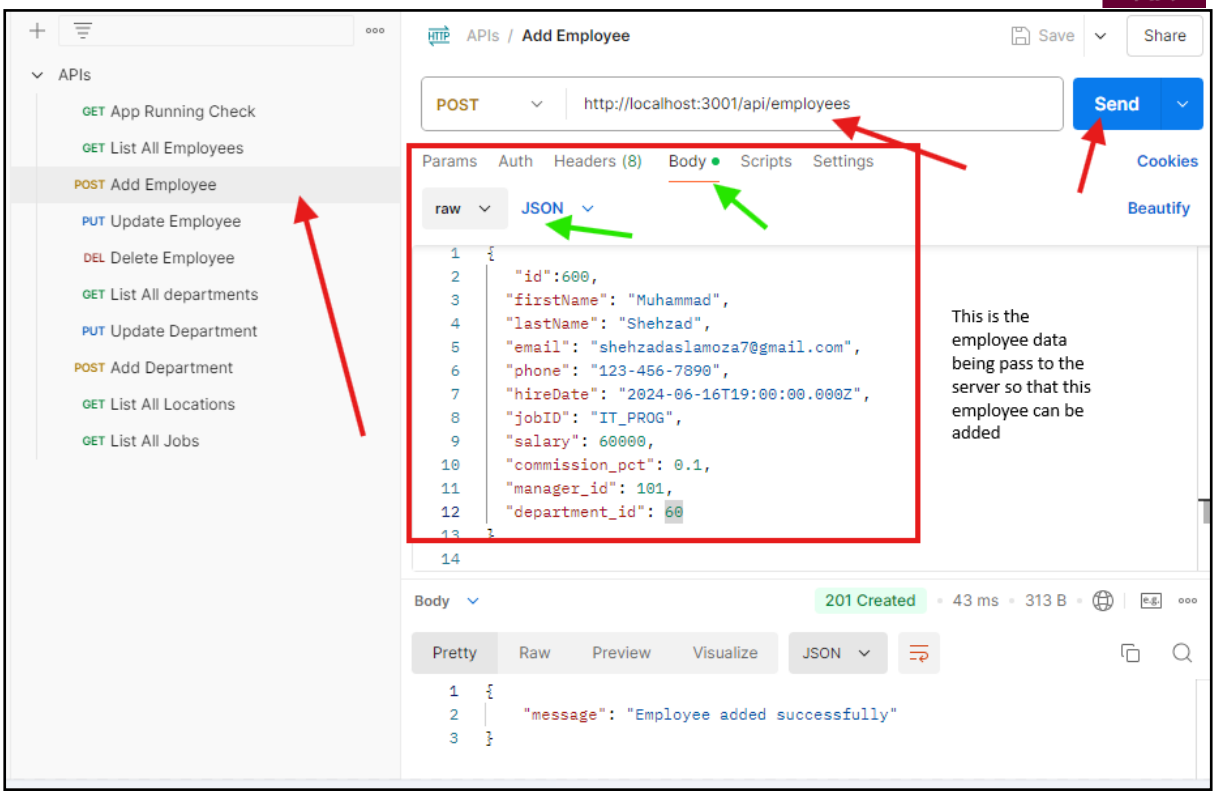
GET Request: Retrieving Data

- **Purpose:** The GET method is used to retrieve information from the server. This is typically used to read or view existing data.
- **Use Case:** Fetching a list of employees or retrieving details of a specific employee.



POST Request: Creating New Data

- **Purpose:** The POST method is used to send data to the server to create a new resource. It is typically used to add new records to the database.
- **Use Case:** Creating a new employee by sending their details (name, job, salary, etc.) to the server.



PUT Request: Updating Existing Data

- **Purpose:** The PUT method is used to update existing data on the server. This method requires specifying the resource you want to update, and you usually send the entire updated resource.
- **Use Case:** Updating an employee's salary or job title.

DELETE Request: Deleting Data

- **Purpose:** The DELETE method is used to remove data from the server.
- **Use Case:** Deleting an employee record from the database.

Step 7: Understanding the MVC Pattern

READ THIS – Walkthrough in next step

The **MVC (Model-View-Controller)** pattern is a popular architectural pattern used in web development to separate the concerns of the application into three interconnected components:

1. **Model** – Represents the data and business logic.
2. **View** – Represents the user interface (UI), although in a Node.js backend, this part might be minimal.

3. **Controller** – Acts as the intermediary between the model and the view, processing input and returning output.

Models (models folder)

- **Purpose**: The **models** define the structure of the data and the interaction with the database. In this backend, the models likely handle how employee data is structured and interact with OracleDB.
- **Example**: A model might define an **Employee** with properties such as first_name, last_name, email, phone_number, and so on.
- **Typical Operations**:
    - Define the schema for the data.
    - Handle database queries (like fetching, inserting, updating, and deleting data).

Controllers (controllers folder)

- **Purpose**: The **controllers** contain the logic that handles incoming requests from the client. They interact with the models to get or update data and return responses to the client.
    - **Example**: The employeeController.js file might contain functions like:
    - getAllEmployees: Fetch all employees from the database.
    - createEmployee: Add a new employee.
    - updateEmployee: Update details of an existing employee.
    - deleteEmployee: Remove an employee from the database.
- **Typical Workflow**:
    - A request comes from the client (via Postman or a frontend).
    - The controller processes the request, interacts with the model to get or modify the data, and sends a response back.

Routes (routes folder)

- **Purpose**: Routes define the endpoints and map them to specific controller functions. They serve as the entry point for client requests.
- **Example**: The routes/employees.js file might define routes such as:
    - GET /employees: Fetch all employees.
    - POST /employees: Add a new employee.

- PUT /employees/:id: Update an employee.
- DELETE /employees/:id: Delete an employee.

- **Typical Workflow**:
  - The routes file maps the URL and HTTP method (e.g., GET, POST) to the corresponding controller function that handles the request.

Configuration (config folder)

- **Purpose**: This folder typically contains configuration files related to database connections or any third-party services the app might use.
- **Example**: You might have a file such as db.js that manages the connection to OracleDB using the environment variables specified in your .env file.

App Initialization (app.js)

- **Purpose**: This is the main entry point of your backend application. It sets up the Express app, middleware, routes, and any other required components.
- **Functionality**:
  - **Initialize Express**: Starts the Express server.
  - **Set Up Middleware**: Adds middleware like body parsers, authentication checks, etc.
  - **Define Routes**: Imports routes and connects them to the Express application.

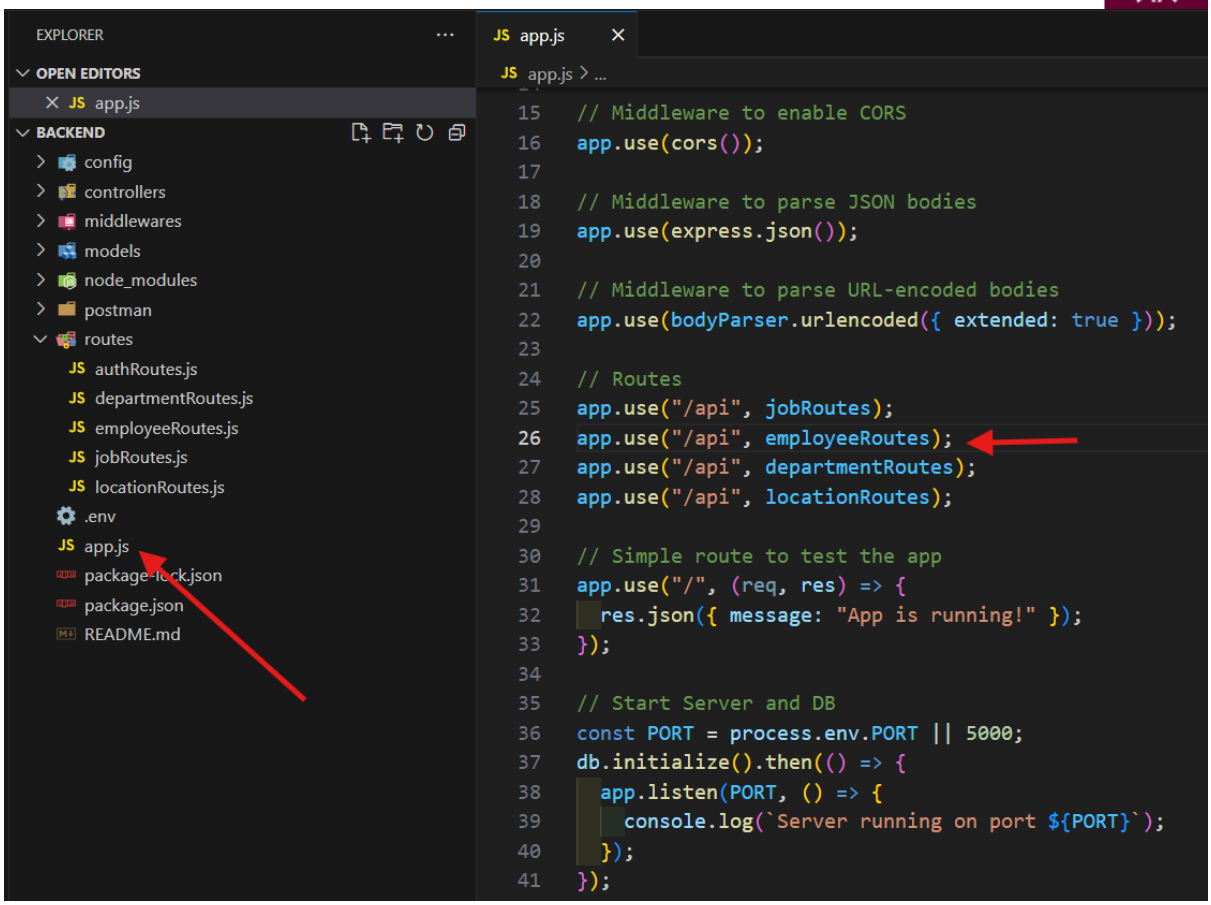
MVC Flow in Action:

- **Client Request (Postman) → Routes → Controller → Model → Database**
- After interacting with the database, the response travels back: **Database → Model → Controller → Client Response**

Step 8: Walkthrough listAllEmployees api

Let's see what happens when you hit http://localhost:3001/api/employees

App.js

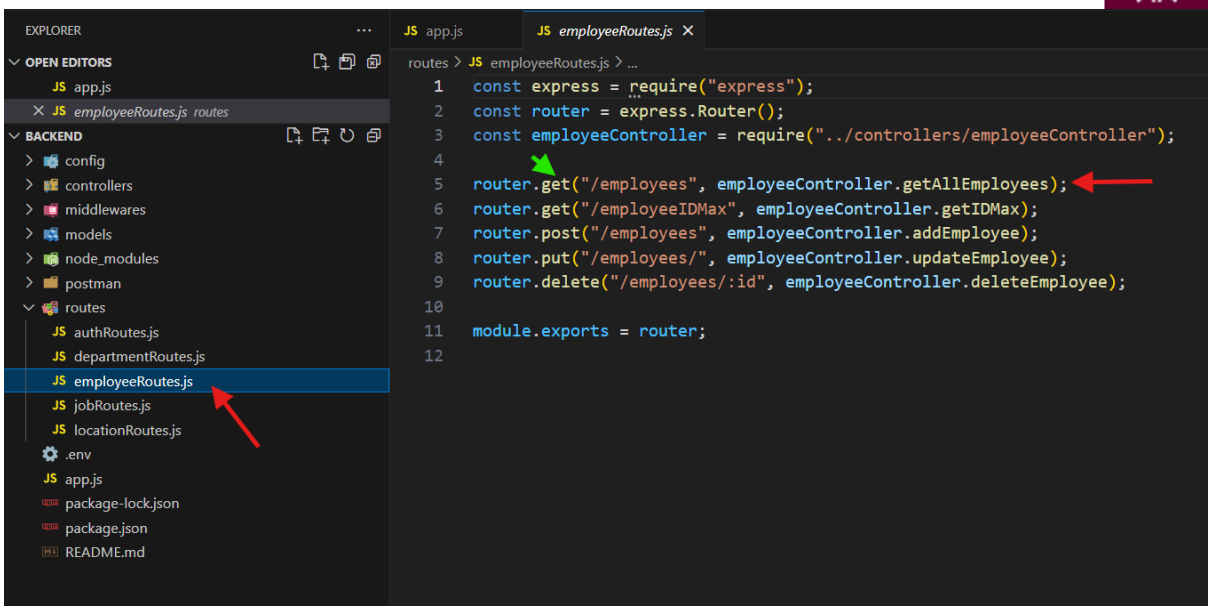


The request is received, http://localhost:3001/api/ is matched against the each of the routes in

```
app.use("/api", jobRoutes);
app.use("/api", employeeRoutes);
app.use("/api", departmentRoutes);
app.use("/api", locationRoutes);
```

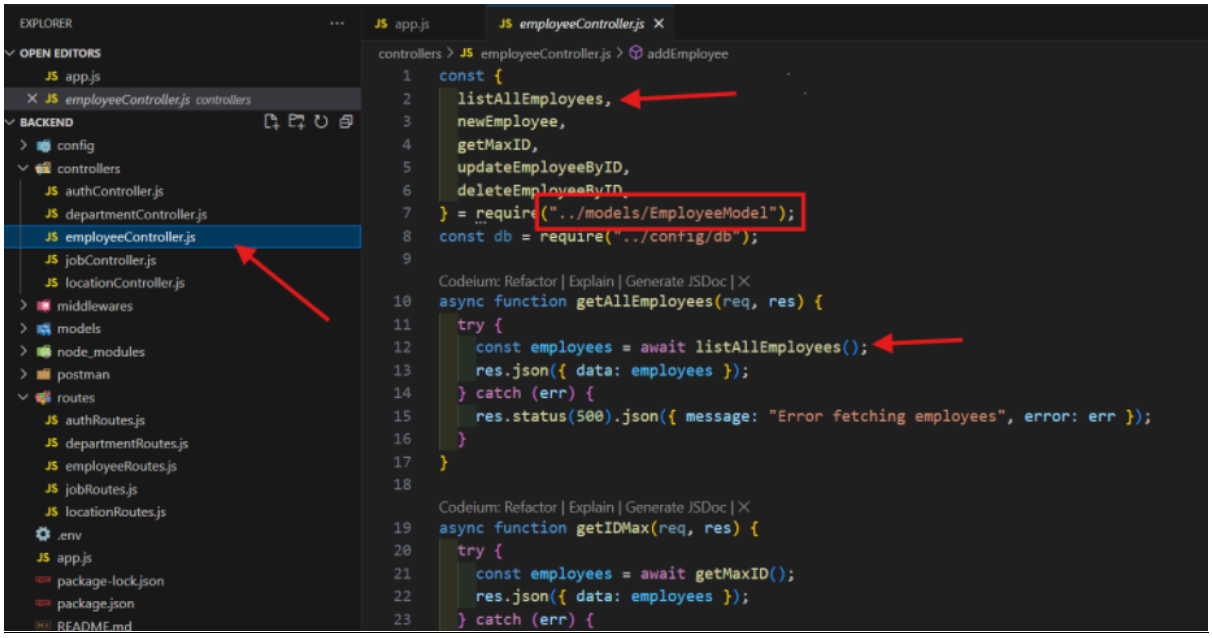
but since the complete path http://localhost:3001/api/employees is in **employeesRoutes**, request will be passed there.

employeesRoutes.js



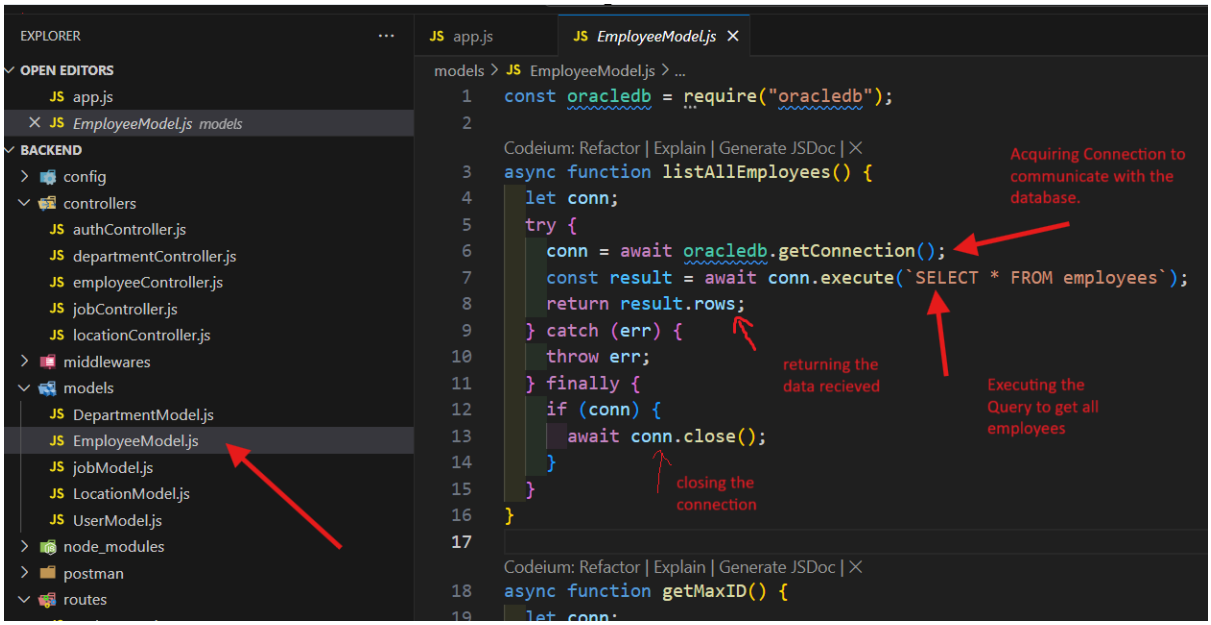
In employeesRoute, we have all the routes belonging to employees. Since we made a **GET** request to http://localhost:3001/api/employees, the request is passed to the getAllEmployees function in EmployeesController

employeesController.js



In Employees Controller, the getAllEmployees function is called. This function calls listAllEmployees function that is made in **EmployeeModel.js**. It's present in try catch block to handle errors.

EmployeeModel.js.



In ListAllEmployees, we first acquire the connection from oracle. Then execute the query and return the result. We also close the connection that was created.

Then the result is propagated back **Database → EmployeesModel → EmployeesController → Response to Client (You)**

READ STEP 7 AND 8 MULTIPLE TIMES TO UNDERSTAND THE FLOW.

Frontend

Now that the backend is set up and working, we will connect it to the front end.

Step 1: Starting the frontend

1. While the backend window is running, open the frontend folder in a new instance of VSCode

```
❯ ls
backend  db-init.sql  frontend

❯ cd frontend/

❯ code .
```

2. Open the terminal
3. **Type 'npm install',** this installs all the node libraries required by the frontend. Make sure you have node installed

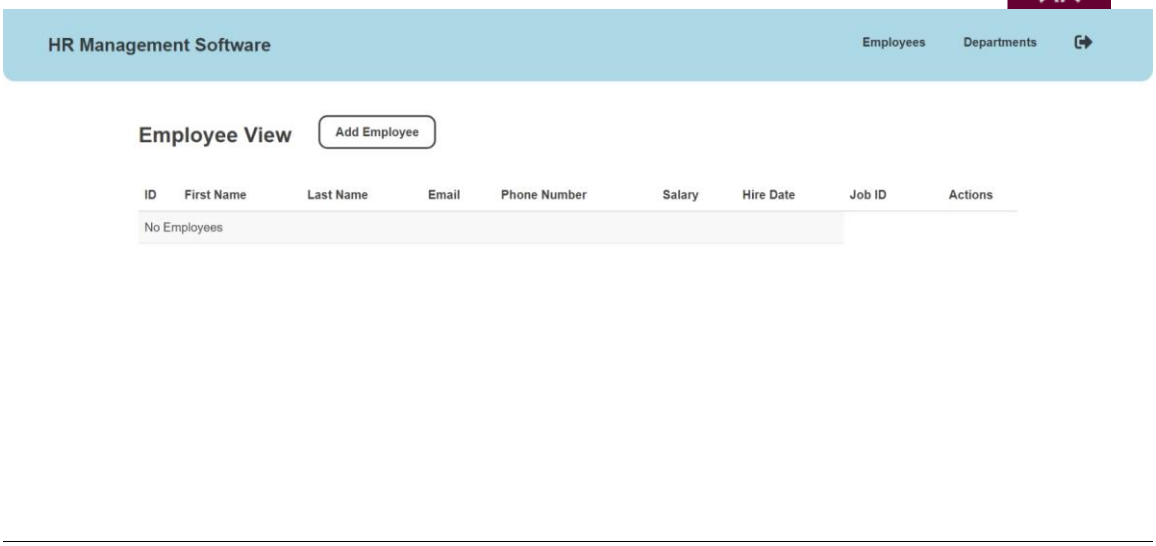
```
npm install
```

```
❯ ls
LICENSE  README.md  node_modules  package-lock.json  package.json  public  src

❯ npm install

up to date, audited 1494 packages in 12s

264 packages are looking for funding
  run `npm fund` for details

11 vulnerabilities (1 low, 2 moderate, 7 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

4. Type npm start on the terminal to start the frontend.

```
npm start
```

5. The frontend should open in a browser tab automatically, If it doesn't ,type localhost:3000 in a new browser tab.



Step 2: Adding data to the frontend

We will now populate frontend.

1. First, we will check what is the route to get the list of employees.

```
const express = require("express");
const router = express.Router();
const employeeController = require("../controllers/employeeController");

router.get("/employees", employeeController.getAllEmployees);
router.get("/employeeIDMax", employeeController.getIDMax);
router.post("/employees", employeeController.addEmployee);
router.put("/employees/", employeeController.updateEmployee);
router.delete("/employees/:id", employeeController.deleteEmployee);

module.exports = router;
```

   From EmployeeRoutes.js we can see that it is "/api/employees"

2. Next, we will add the API call to Employees.js.

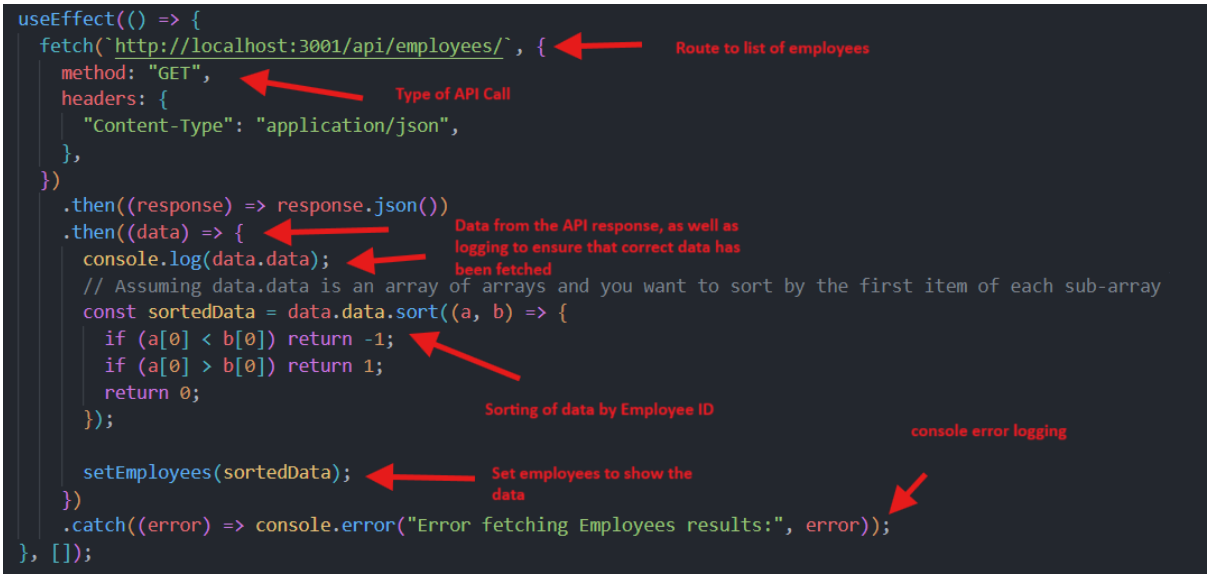
```
const Dashboard = ({ setIsAuthenticated }) => {
  const [employees, setEmployees] = useState("");
  const [selectedEmployee, setSelectedEmployee] = useState(null);
  const [isAdding, setIsAdding] = useState(false);
  const [isEditing, setIsEditing] = useState(false);

  //LAB DEMO: Add call to fetch employees here.

  const handleEdit = (employee) => {
    setSelectedEmployee(employee)
    setIsEditing(true);
  };
```

The “employees” variable is what stores the list of employees. As we can see right now, it is empty (set to “”). To populate it with data from the database, add the following useEffect hook:

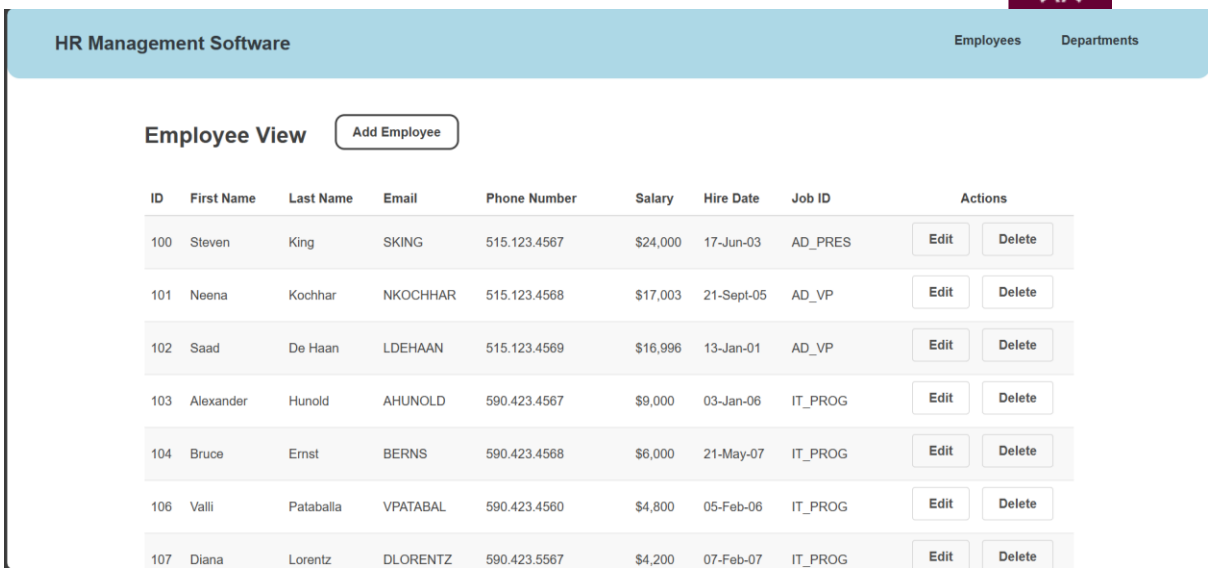
```
  useEffect(() => {
    fetch(`http://localhost:3001/api/employees/`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
    })
      .then((response) => response.json())
      .then((data) => {
        console.log(data.data);
        // Assuming data.data is an array of arrays and you want to sort by
the first item of each sub-array
        const sortedData = data.data.sort((a, b) => {
          if (a[0] < b[0]) return -1;
          if (a[0] > b[0]) return 1;
          return 0;
        });

        setEmployees(sortedData);
      })
      .catch((error) => console.error("Error fetching Employees results:",
error));
  }, []);
```

useEffect is a React hook that runs on every render.

3. Make sure that the parameters in the hook are correct.



4. Now the employees are appearing in the frontend



Task

The front end has two parts to it, the employees view and the departments view.

Data for the employees view is already added in the lab demo. Use it to perform the following tasks:

1. Create an API to list all departments from the database. You will have to edit the following files
    a. DepartmentModel.js
    b. DepartmentController.js
    c. DepartmentRoutes.js
2. Test your newly created API with Postman. Attach screenshots to show that it works
3. Call your API from the frontend. You will be modifying the following files:

a. Departments.js

Comments have been left in the files for your ease. Share screenshots of all the departments listed.

Optional Task

APIs have been made for ADD and EDIT Departments. Explore the app and see how these tasks are performed for the Employees Table. Add these for the Departments Table as well.