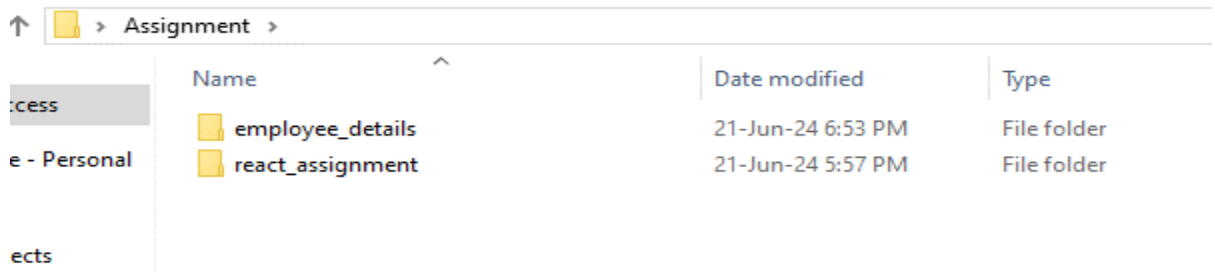
React Assignment

GitHub Link: Frontend & Backend

Project folder Looks like



Process of creating frontend project.

Step 1: Initialize the project

By creating the applicating using below command,

npx create-react-app react_assignment

where react_assignment is the name of the folder for our application. This may take a few minutes to create the React application and install its dependencies.

Step 2: Deleting all Unnecessary Files

Now deleted all unnecessary files in our folder structure. We also need to make some changes in our index.js & index.html as it contains these unwanted element links.

Step 3: Installing necessary libraries

npm install react-router-dom &

npm install @mui/material @emotion/react @emotion/styled

Step 4: Modifying index.js file and Creating routes in app.js file.

Index.js file:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.js';
import { BrowserRouter } from 'react-router-dom';
import axios from 'axios';
```

```
const root =
ReactDOM.createRoot(document.getElementById('root'));
axios.defaults.baseURL = "http://localhost:5000/api";
//Instead of using full URL in each request, we can managed
here.
root.render(
    <BrowserRouter>
        <App />
    </BrowserRouter>
);
```

App.js file:

```
import React from "react";
import { Route, Routes } from "react-router-dom";
import Employee from "./components/Employee.jsx";
import AddEmployee from "./components/AddEmployee.jsx";
import HomePage from "./components/HomePage.jsx";

function App() {
  return (
    <Routes>
      <Route path="/" element={<HomePage />} />
      <Route path="/list" element={<Employee />} />
      <Route path="/add" element={<AddEmployee />} />
    </Routes>
  );
}

export default App;
```

Specifying path address and re-directing the pages.

Step 5: Creating components

File name: EmployeeList.jsx

Designed the page to display each data using material ui.

```
import { Card, CardContent, Typography } from
'@mui/material'
import React from 'react'

function EmployeeList(props) {
    return (
        <Card
            sx={{
                margin: 2,
                width: 180,
                height: "auto",
                borderRadius: 5,
                ":hover": {
                    boxShadow: "10px 10px 20px #ccc",
                },
            }}>
            <CardContent>
                <Typography gutterBottom variant="h6" >
                    ID: {props.EmployeeId}
                </Typography>
                <Typography gutterBottom variant="h6" >
                    Name: {props.EmployeeName}
                </Typography>
                <Typography gutterBottom variant="h6" >
                    Departmet: {props.Department}
                </Typography>
```

```
                <Typography gutterBottom variant="h6" >
                    Salary: {props.Salary}
                </Typography>

            </CardContent>

        </Card>
    )
}

export default EmployeeList
```

File name: Employee.jsx

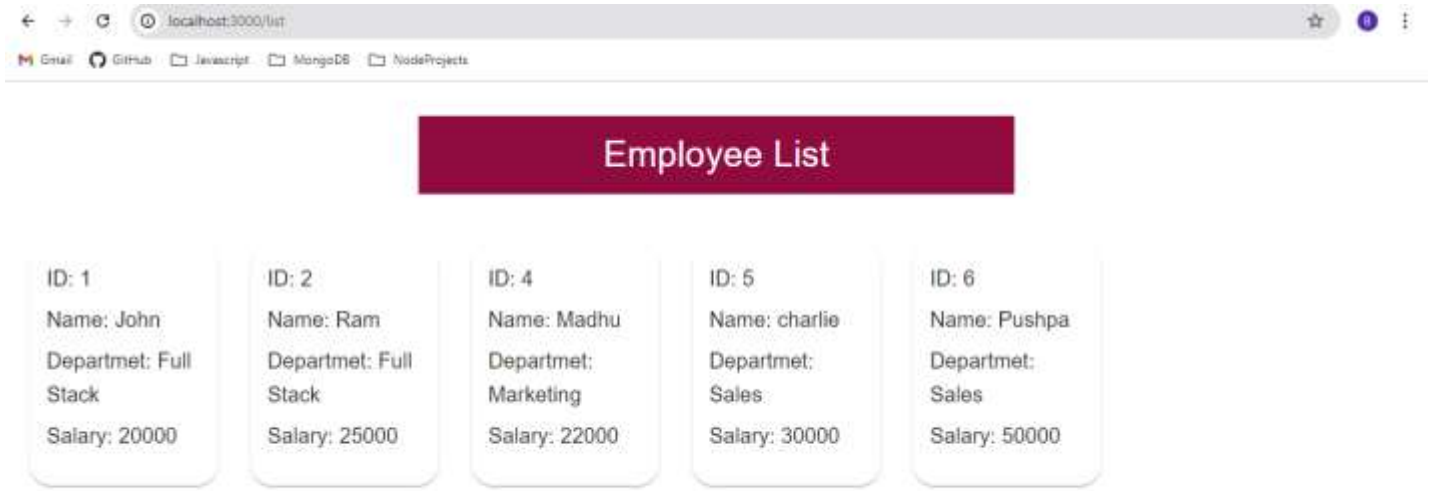
In this file getting/retrieving all data from the database. Display's all data retrieved from database in the web page.

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { Box, Typography } from '@mui/material';
import EmployeeList from './EmployeeList';

function Employee() {
    const [employees, setEmployees] = useState();

    async function getEmployeeList() {
        //Useing axios to send request and get respose from
the backend
        const res = await axios.get("/list") //.get is used
to get all data from database
            .catch((err) => console.log(err));

        //Checking error when no data found at database
        if (res.status !== 200) {
```

```
            return console.log("No Data");
        }

        let data = null;
        if (res) {
            data = await res.data;
        }
        return data;
    };

    useEffect(() => {
        getEmployeeList()
            .then(data => setEmployees(data.employees));
//Updates the initial state value, Once data retrieved
    }, []);

    return (
        <Box margin={"auto"} marginTop={4}>
            <Typography
                margin={"auto"}
                variant="h4"
                padding={2}
                width="40%"
                bgcolor={"#900C3F"}
                color="white"
                textAlign={"center"}
            >
                Employee List
            </Typography>
            <Box
                width={"100%"}
                margin="auto"
```

```
                marginTop={4}
                display={"flex"}
                justifyContent="flex-start"
                flexWrap={"wrap"}
            >
                {employees &&
                    employees.map((Employee, index) => (

                        <EmployeeList
                            key={index}
                            //Retrieving data from database
and displaying in the page
                            EmployeeId={Employee.EmployeeId
}
                            EmployeeName={Employee.Employee
Name}
                            Department={Employee.Department
}
                            Salary={Employee.Salary}
                        />


                    ))}
            </Box>

        </Box>
    )
}

export default Employee
```

Output:



File Name: AddEmployee.jsx

This file to Inserting data to the database present in MongoDB Atlas by send the post request. Create form to insert data using input fields.

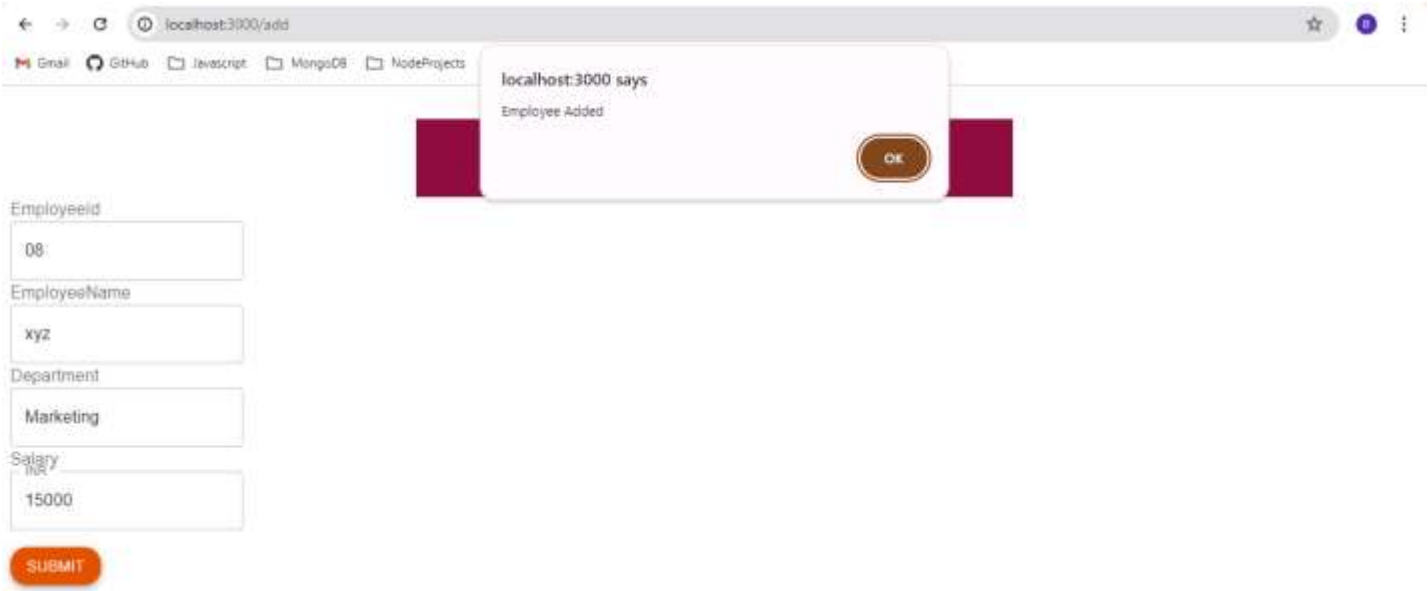
```
import { Button, InputLabel, TextField, Typography } from
'@mui/material';
import axios from 'axios';
import React, { useState } from 'react'

function AddEmployee() {

    const [inputs, setInputs] = useState({
        EmployeeId: "", EmployeeName: "", Department: "",
Salary: "" //Variables declared with initially no values to
update values from web page
    })

    //Function to handle changes and update it
    function handleChange(e) {
        setInputs((prevState) => ({
            ...prevState, //Taking or Managing the previous
values as it is
```

```
            [e.target.name]: e.target.value  //To insert
new values
        }));
    }

    //async and await to handle promisees
    async function sendRequest() {
        //Useing axios to send request and get respose from
the backend
        const res = await axios.post(`/add`, { //.post is
used to insert data to the database
            //Inserting values to the variable to save in
database
            EmployeeId: inputs.EmployeeId,
            EmployeeName: inputs.EmployeeName,
            Department: inputs.Department,
            Salary: inputs.Salary,
        }).catch(err => console.log(err))

        const data = await res.data;
        return data;
    }

    function handleSubmit(e) {
        e.preventDefault(); //To control the default
actions
        console.log(inputs);
        sendRequest()
            .then(data => console.log(data)) //Prents the
inserted data in the console
            .then(() => alert("Employee Added")) //If data
added successfully
```

```
    }

    return (
        <div>
            <form onSubmit={handleSubmit} >
                <Typography
                    //Using CSS properties to style the
content of the page
                    margin={"auto"}
                    marginTop={4}
                    variant="h4"
                    padding={2}
                    width="40%"
                    bgcolor={"#900C3F"}
                    color="white"
                    textAlign={"center"}>
                    Add Employee
                </Typography>
                <InputLabel
variant='h6'>EmployeeId</InputLabel>
                <TextField name='EmployeeId'
onChange={handleChange} value={inputs.EmployeeId}
variant='outlined' required/>
                <InputLabel
variant='h6'>EmployeeName</InputLabel>
                <TextField name='EmployeeName'
onChange={handleChange} value={inputs.EmployeeName}
variant='outlined' required/>
                <InputLabel
variant='h6'>Department</InputLabel>
```

```
                <TextField name='Department'
onChange={handleChange} value={inputs.Department}
variant='outlined' required/>
                <InputLabel
variant='h6'>Salary</InputLabel>
                <TextField name='Salary'
onChange={handleChange} value={inputs.Salary}
variant='outlined' label='INR' required/><br />
                <Button type="submit" sx={{ mt: 2,
borderRadius: 4 }} variant='contained'
color='warning'>Submit</Button>
            </form>
        </div>
    )
}

export default AddEmployee
```

Output:



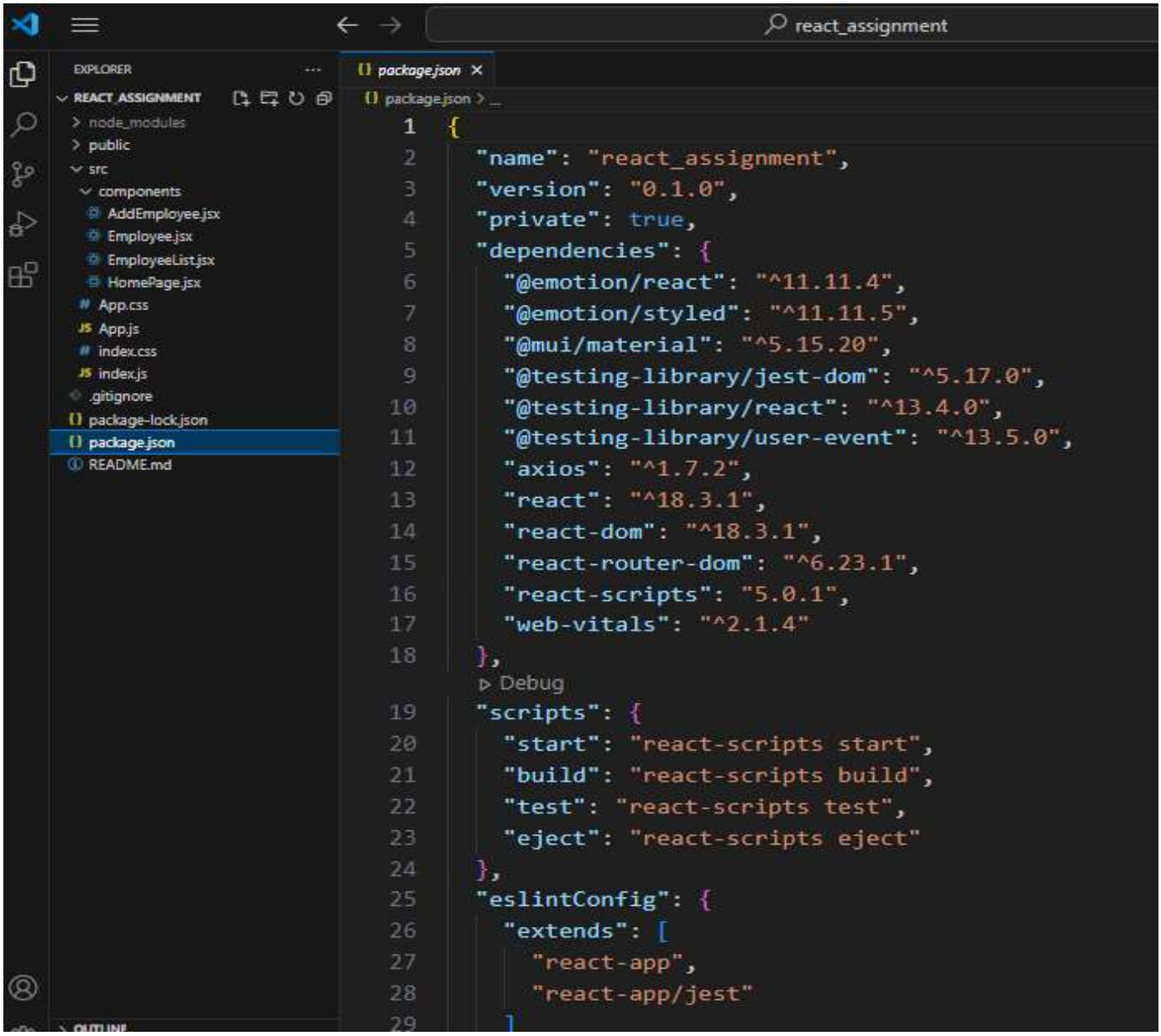
File Name: HomePage.jsx

Created to make the application in correct format. Home page will be display first with two buttons they are view all employees and add employee. Which renders to particular page once button clicked.

```
import { Box, Button } from '@mui/material';
import React from 'react'
import { Link } from 'react-router-dom';

function HomePage() {
    return (
        <div >
            <Box display={"flex"} padding={5}
margin={"auto"} variant="h1" >
                <Button LinkComponent={Link} to="/list"
variant='outlined' sx={{ margin: "auto", width: "30%",
color: "#2b2d42" }}>View All Employees</Button>
            </Box>
            <Box display={"flex"} padding={5}
margin={"auto"}>
                <Button LinkComponent={Link} to="/add"
variant='outlined' sx={{ margin: "auto", width: "30%",
color: "#2b2d42" }}>Add Employees</Button>
            </Box>
        </div>
    )
}

export default HomePage
```

Output:

After the project completed, The folder Structure and package.json file looks like,



Run the project using the command,

```
npm start
```

Once the command run the web page automatically opens in the browser.

Process of creating backend

Step 1: Setting up the project

Create folder for the backend and initializing the project process,

```
npm init
```

Step 2: Installing necessary libraries,

Installing nodemon for the developer dependencies using below command

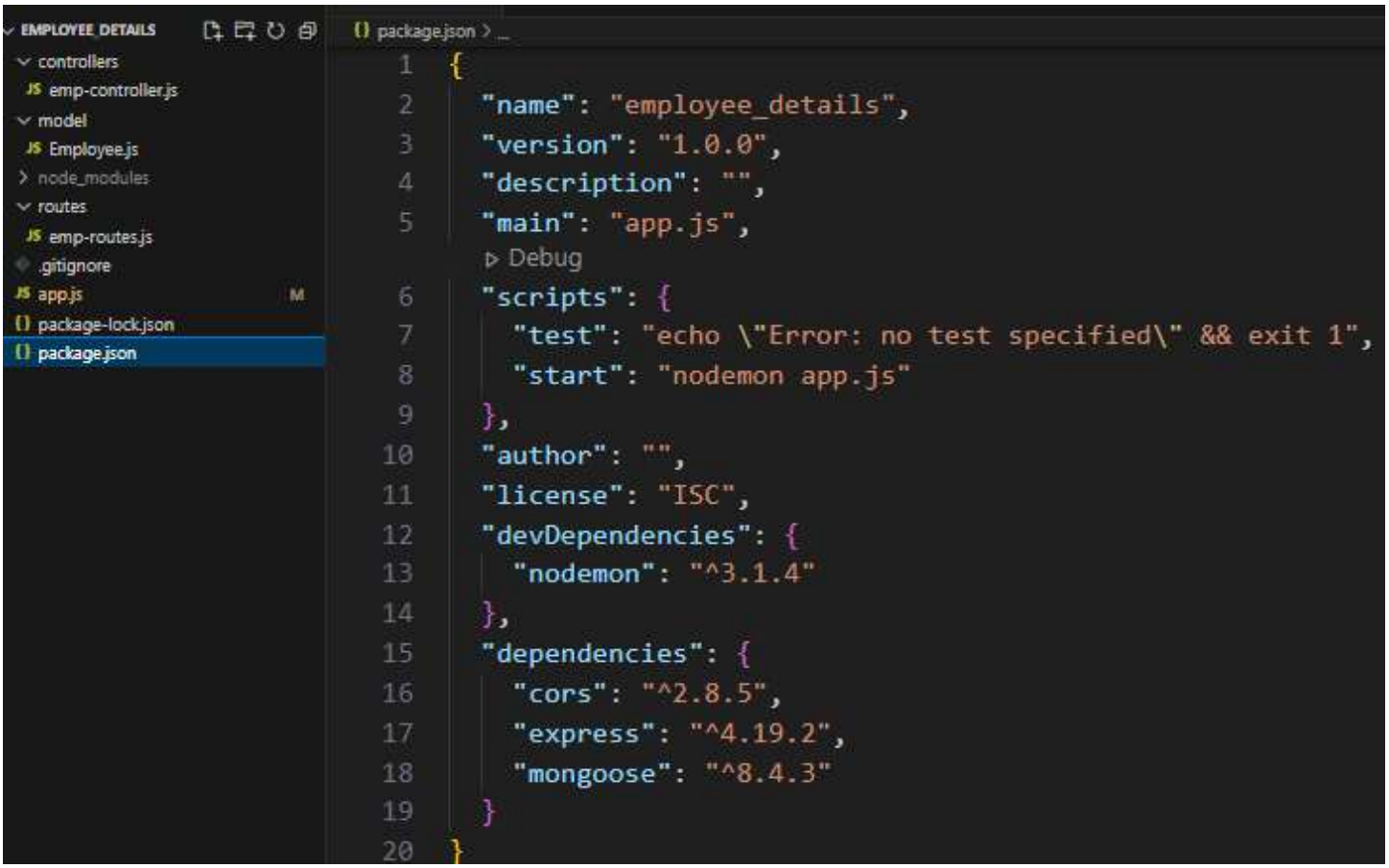
```
npm I nodemon –save-dev
```

&

```
npm i express mongoose cors
```

Step 3: Creating project using MVC (Model View Controller) architecture. Which helps us to organize our web application and make it more manageable.

Folder structure and package.json file looks like,



Step 4: Create project in MongoDB Atlas

1. Login to MongoDB Atlas free trail.
2. Create new project
3. Click on Create Cluster to deploy the cluster. Once you deploy your cluster, it can take up to 5-10 min for your cluster to provision and become ready to use.

Step 5: Connecting to MongoDB Atlas

1. Go to the Database tab, select your database, and click Connect. Then choose the method of connection.
2. Then copy your URL link and paste in your express server directly (app.js file).

App.js file,

```
const express = require('express');
const mongoose = require('mongoose');
const employeeRouter = require('./routes/emp-routes.js')
const cors = require('cors'); //Installed cors to interact
with different domain
```

```
const app = express();
app.use(cors()); //Used to allow access from the Frontend
app.use(express.json());
app.use("/api", employeeRouter)

//Connecting to database from MongoDB Atlas
mongoose.connect("mongodb+srv://bhoomikahm18:Eb9pD8rgZxthvt
sy@cluster0.p374nah.mongodb.net/?retryWrites=true&w=majorit
y&appName=Cluster0")
    .then(() => app.listen(5000)) //Giving the port number.
    .then(() => console.log("Connected to Database and
listening to localhost 5000"))
    .catch((err) => console.log(err))
```

Step 6: Create Database.

Schema is designed to create the database at mongodb atlas with required properties.

Model named as Employee.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

// Creating the Database
const employeeSchema = new Schema({
    EmployeeId: {
        type: Number,
        required: true,
        unique: true,
    },
    EmployeeName: {
        type: String,
```

```
        required: true,
        unique: true,
    },
    Department: {
        type: String,
        required: true,
    },
    Salary: {
        type: Number,
        required: true,
    },
})

module.exports = mongoose.model("Employee",
employeeSchema);
```

Step 7: Create routes

emp-routes.js

```
const express = require("express");
const { getAllEmployee, addEmployee } =
require("../controllers/emp-controller.js");

const router = express.Router();

router.get("/list", getAllEmployee);
router.post("/add", addEmployee);

module.exports = router;
```

Step 8: Create controller methods

Write logic to operate data from database, like getting all data from the database and inserting data to the database.

emp-controller.js

```
const Employee = require("../model/Employee.js");

module.exports.getAllEmployee = async (req, res) => {
    let employees;
    try {
        employees = await Employee.find(); //Retrieving all
employees from database
    } catch (err) {
        console.log(err);
    }
    if (!employees) { //Checking database, is has employee
data or not
        return res.status(404).json({ message: "Employee
Not Found" });
    }
    return res.status(200).json({ employees });
};

module.exports.addEmployee = async (req, res) => {
    const { EmployeeId, EmployeeName, Department, Salary }
= req.body;

    let existingEmployee; //Checking for already existing
employee by name
    try {
        existingEmployee = await Employee.findOne({
EmployeeName }); //Checking employee name
    } catch (err) {
```

```
        return console.log(err);
    }

    if (existingEmployee) {
        return res.status(400).json({ message: "Employee
already exists" });
    }

    const employee = new Employee({
        EmployeeId,
        EmployeeName,
        Department,
        Salary,
    });

    try {
        await employee.save(); //Storing new data to the
database
    } catch (err) {
        return console.log(err);
    }

    return res.status(200).json({ employee });
}
```

All the scenario's are checked in postman. Before directly access from the web page.

Completed the project with given task,

- Show employee list in Data table

Displayed all employee list in the web page by retrieving it from the employee database

- Add Employee page in separate page

Separate page is designed to insert new employee to the database

- Handle Validation of each field

Maintained validation like EmployeeId and EmployeeName should be unique, data type is defined to each field and each field should contain the value, not to be null value.