Node, Express and MongoDB

Section 1 – NodeJS Modules

- NodeJS Introduction , Architecture , Installation and Setup
- Path Module
- Url Module
- FS Module
- Http Module
- Create Server App and apply routing.
- Attach css and image in page

*1. Explain the steps to install Node.js on your computer. Create a simple "Hello World" program and run it using Node.js.
*2. Write a Node.js program that uses the `path` module to join and normalize file paths. Also, create a program that utilizes the `url` module to parse and manipulate URLs.
*3. Develop a Node.js script to read a text file asynchronously using the `fs` module and display its contents.
*4. Create a basic HTTP server using Node.js. When a client makes a request to the server, respond with "Hello, World!".
*5. Expand on the previous HTTP server by implementing basic routing. Depending on the URL path, return different responses (e.g., "Welcome to Home Page" for '/', "About Us" for '/about', etc.).

#1. Create Multiple Pages (Home , About and Services) and also attach images and boot-strap in it. Prepare Menu Bar for routing.
#2. Enhance your server to handle POST requests. Create a form in HTML that sends data to the server, which then processes and displays it.
#3. Modify your server to serve static HTML, CSS, and image files. When a user accesses '/style.css' or '/image.jpg', they should see the respective content.
#4. Implement a logging system in your Node.js server. Log each incoming request along with a timestamp and the requested URL.
#5. Extend your routing system to handle URL parameters. For example, create a route like '/user/:id' that displays user information based on the 'id' parameter.
#6. Build a simple RESTful API using Node.js. Create endpoints for CRUD operations (e.g., GET, POST, PUT, DELETE) on a resource like "tasks" or "products". Test your API using tools like Postman or curl.

Section 2 – Express and Routing

- Express Introduction
- Create Server via Express
- EJS Introduction
- Send Response from Express
- Routing in Express
- Handle Get, Post, Put and Delete Request Data
- Path Variable in Express
- Response json() function
- Mail Sending
- OTP Verfication
- Payment Intergation

*1. Take input 5 subjects marks and send Get request to server and show total , percentage and grade in response page.

*2. Set up an Express.js application on your local machine, and create a basic server that listens on port 3000. Test the server to ensure it's running.

*3. Create an Express application that uses the EJS template engine to render a dynamic webpage. Pass data from the server to the template and display it.

*4. Build an Express app with three different routes: '/home', '/about', and '/contact'. Each route should render a separate HTML page.

*5. Implement an Express route that handles POST requests. Create a simple HTML form that allows users to submit data, and then display the submitted data on a new page.

*6. Extend your Express app to handle PUT requests. Design a route that updates existing data, such as user information, based on an 'id' parameter.

*7. Take employee income details such as basic salary , incentives , bonus and tax from user and send Post request to server and show off result on response page.

#1. Patient Records Management Application with this menu :

Home Add Patient Show Patient

1. Add New Patient
2. Show All Patient in Table
3. Delete Patient

Note : Data is stored in a JSON File.

#2. Develop an Express route to handle DELETE requests. Define a route that removes data, such as user records, based on an 'id' parameter.

#3. Build an Express API endpoint at '/api/users' that responds with a JSON array of user objects. Populate the JSON data with mock user data.

#4. Implement a custom middleware function in your Express app. Log a message to the console for each incoming request.

#5. Create an error-handling middleware in Express. Test it by intentionally triggering an error in one of your routes and ensuring that the error handler responds appropriately.

Section 3 – Mysql Connectivity

- Install Mysql2 Module
- Mysql Connectivity
- Perform CRUD Operation

*1. Patient Records Management Application with this menu :

Home Add Patient Show Patient

1. Add New Patient
2. Show All Patient in Table
3. Delete Patient

#1. Manage Multiple Students Exam Details via Mysql :
Exam Details : RollNumber , Name , Branch , Physics , Chemistry , Maths

Operations : 1. Add Exam Details
2. Delete Exam
3. Filter Exam Records Via Branch , Name and Roll Number

Section 4. Sequelize

- Sequelize Introduction
- Sequelize Model and Creation
- Sequelize Migration
- CRUD Operation in Sequelize
- Table Column Constraints and Validation
- Association : One-To-One , One-To-Many , May-To-Many
- Fetch Data from table

*1. Create REST API for Patient Records Management Application.

#1. Create REST API Manage Student CRUD.
#2. Create Three Tables :

Department
departmentId , departmentName
Employee
empId , empName , empEmail ,
empDepartment , empSalary
EmpSalary
salId , employee , month , amount , bonus

1. Add , Update and List Department Api
2. Add , Update and List Employee Api
3. Add and List Employee Salary Api

Section 5. MongoDB

- MongoDB Install
- Create Database and Collection
- Insert , Update and Delete Document
- Delete Collection
- Fetch Records from Collection
- Install Mongoose Module
- Mongoose Connectivity
- Perform CRUD Operation

*1. Patient Records Management Application Api:

1. Add New Patient
2. Show All Patient in Table
3. Delete Patient

Note : Data is stored in a Mongoose Table.

#1. Manage Multiple Students Exam Details via MongoDB:
Exam Details : RollNumber , Name , Branch , Physics , Chemistry , Maths

Operations : 1. Add Exam Details
2. Delete Exam
3. Filter Exam Records Via Branch , Name and Roll Number

Section 6. JWT , CORS and File Uploading

- JWT Introduction
- Setup JWT : Create Token , Parse Token
- Create Token Parsing Middleware
- Apply CORS
- Uploading File into Server

*1. Create a Node.js script that generates a JWT token. The token should include a user's unique identifier and a custom claim (e.g., "role" with a value of "user"). After generating the token, print it to the console.
*2. Develop an API endpoint that allows users to upload files (e.g., images) to your Node.js server. Describe how to handle file uploads, store them on the server, and provide a link to access the uploaded files.

#1. Develop a Node.js script that verifies the authenticity of a given JWT token. You should provide a secret key used for signing the token. The script should take a token as input and verify whether it's valid or not. If the token is valid, print a success message; otherwise, print an error message.

#2. Implement functionality for users to download and manage their uploaded files. Create routes that list, download, and delete user-specific files stored on the server.

Section 7. Mini Project

College Project in which three users are there such as :

1. Admin
    1. Login
    2. View Students
    3. View Faculty
    4. Block Any other User.
2. Student
    1. Register and Login
    2. Update Profile
    3. Ask Question
    4. Download Files
    5. View Messages
3. Faculty
    1. Register and Login
    2. Update Profile
    3. Solve Student Question
    4. Upload Files
    5. Send Messages

Note : Using Sequelize and Mongoose

Section 8. Minor Project

1. ChatBuddy
2. Emall
3. Library
4. CodeBetter Center Management

Section 9. API Testing Tool