ExpressJS – Routes, Params, Query Params & URL Building

Question: Write an ExpressJS program to define a route, handle route parameters, query parameters, and URL building.

File: app.js

```
// Import express
const express = require('express');
const app = express();

// Define a simple route
app.get('/', (req, res) => {
  res.send('Welcome to the Home Page!');
});

// Route with a parameter (e.g., /user/John)
app.get('/user/:name', (req, res) => {
  const userName = req.params.name;
  res.send(`Hello, ${userName}!`);
});

// Route with query parameters (e.g., /search?item=phone&price=10000)
app.get('/search', (req, res) => {
  const { item, price } = req.query;
  res.send(`You searched for ${item} priced at ₹${price}`);
});

// URL building example
app.get('/profile/:username', (req, res) => {
  const profileUrl = `/user/${req.params.username}`;
  res.send(`Profile URL is: ${profileUrl}`);
});

// Start the server
app.listen(3000, () => {
  console.log('Server running on http://localhost:3000');
});
```

How to Run:

npm init -y

npm install express

node app.js

Test:

GET http://localhost:3000/
GET http://localhost:3000/user/John
GET http://localhost:3000/search?item=phone&price=10000
GET http://localhost:3000/profile/sathwvik

Expected Output:

/ -> Welcome to the Home Page!
/user/John -> Hello, John!
/search?item=phone&price=10000 -> You searched for phone priced at ₹10000
/profile/sathwvik -> Profile URL is: /user/sathwvik

Terminal: Server running on http://localhost:3000

ExpressJS – Middleware Logging (Exact)

Question: Write an ExpressJS program to show the working of middleware.

File: app.js

```
const express = require('express');
const app = express();
const port = 3000;

// Middleware function
const logger = (req, res, next) => {
  console.log(`${req.method} request for '${req.url}'`);
  next(); // Pass control to the next middleware/route
};

// Use middleware for all routes
app.use(logger);

// Route 1
app.get('/', (req, res) => {
  res.send('Hello! This is the homepage.');
});

// Route 2
app.get('/about', (req, res) => {
  res.send('This is the about page.');
});

// Route 3
app.get('/contact', (req, res) => {
  res.send('Contact us at contact@example.com');
});

// Start server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

How to Run:

npm init -y

npm install express

node app.js

GET http://localhost:3000/

GET http://localhost:3000/about

GET http://localhost:3000/contact

Expected Output:

Browser:
'/' -> Hello! This is the homepage.
'/about' -> This is the about page.
'/contact'-> Contact us at contact@example.com

Terminal:
GET request for '/'
GET request for '/about'
GET request for '/contact'
Server running at http://localhost:3000

ExpressJS – User Authentication (Exact)

Question: Write an ExpressJS program for user authentication.

File: app.js

```
const express = require('express');
const app = express();
const port = 3000;

app.use(express.json());

// Dummy user data
const user = {
  username: 'shamith',
  password: '12345'
};

// Middleware to check authentication
function authenticate(req, res, next) {
  const { username, password } = req.body;
  if (username === user.username && password === user.password) {
    next();
  } else {
    res.status(401).send('Authentication failed');
  }
}

// Login route
app.post('/login', authenticate, (req, res) => {
  res.send(`Welcome ${req.body.username}, you are authenticated!`);
});

// Protected route
app.get('/dashboard', authenticate, (req, res) => {
  res.send('This is the dashboard. Only authenticated users can see this.');
});

app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

How to Run:

npm init -y
npm install express
node app.js

Test with Postman:
POST http://localhost:3000/login
Body (JSON):
{ "username":"shamith", "password":"12345" }

Expected Output:

POST /login (correct)-> Welcome shamith, you are authenticated!
POST /login (wrong) -> 401 Authentication failed
Terminal: Server running at http://localhost:3000

ExpressJS – Accept, Retrieve, Delete (HTTP methods)

Question: Write an ExpressJS program to accept data, retrieve data and delete a specified resource using HTTP methods.

File: app.js

```
const express = require('express');
const app = express();
const port = 3000;

app.use(express.json()); // to read JSON data

// Sample data (acts like a small database)
let users = [
  { id: 1, name: 'Sathwvik' },
  { id: 2, name: 'Lavanya' }
];

// ✅GET - Retrieve all users
app.get('/users', (req, res) => {
  res.json(users);
});

// ✅POST - Accept data and add a new user
app.post('/users', (req, res) => {
  const newUser = req.body;
  users.push(newUser);
  res.send('User added successfully!');
});

// ✅DELETE - Delete a user by id
app.delete('/users/:id', (req, res) => {
  const id = parseInt(req.params.id);
  users = users.filter(user => user.id !== id);
  res.send(`User with ID ${id} deleted.`);
});

// Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

How to Run:

npm init -y
npm install express
node app.js

GET http://localhost:3000/users
POST http://localhost:3000/users Body: { "id":3, "name":"Ravi" }
DELETE http://localhost:3000/users/2

Expected Output:

GET -> [{ id:1, name:'Sathwvik' }, { id:2, name:'Lavanya' }]
POST -> User added successfully!
DELETE -> User with ID 2 deleted.
Terminal: Server running at http://localhost:3000

ExpressJS – Working with Form Data (Exact)

Question: Write an ExpressJS program to work with form data.

File: app.js

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const port = 3000;

// Middleware to parse form data
app.use(bodyParser.urlencoded({ extended: true }));

// Serve a simple HTML form
app.get('/', (req, res) => {
  res.send(`
    <h2>Simple Form Example</h2>
    <form action="/submit" method="POST">
      Name: <input type="text" name="name" /><br><br>
      Email: <input type="email" name="email" /><br><br>
      <button type="submit">Submit</button>
    </form>
  `);
});

// Handle form submission
app.post('/submit', (req, res) => {
  const { name, email } = req.body;
  res.send(`<h3>Form Submitted Successfully!</h3>
            <p>Name: ${name}</p>
            <p>Email: ${email}</p>`);
});

// Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

How to Run:

npm init -y
npm install express body-parser
node app.js
Open http://localhost:3000/ and submit the form

Expected Output:

Displays submitted Name and Email on the response page.
Terminal: Server running at http://localhost:3000

ExpressJS – Templating Engine (EJS) (Exact)

Question: Write an ExpressJS program using templating engine.

File: views/index.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>home page </title>
  <style>
    body { font-family: sans-serif; padding: 20px; }
    li { color: navy; }
  </style>
</head>
<body>

  <h1><%= message %> </h1>

  <p>Here is a list of items:</p>
  <ul>
    <% items.forEach(function(item){ %>
      <li><%= item %></li>
    <% }); %>
  </ul>

</body>
</html>
```

File: app.js

```
const express = require('express');
const app = express();
const port = 3000;

app.set('view engine', 'ejs');

app.get('/', (req, res) => {

  const data = {
    title: 'Home Page',
    message: 'Hello from EJS!',
    items: ['Apple', 'Banana', 'Cherry']
  };

  res.render('index', data);
});
```

```
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

How to Run:

npm init -y
npm install express ejs
put index.ejs inside ./views
node app.js
GET http://localhost:3000/

Expected Output:

Renders:
Hello from EJS!
Here is a list of items:
- Apple
- Banana
- Cherry

ReactJS – Props & State (Exact)

Question: Write a React JS program to work with props and states.

File: src/App.js

```
import React, { useState } from 'react';
import Greeting from './Greeting';

function App() {
  const [name, setName] = useState("Sathwvik");

  return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      <h1>React Props and State Example</h1>
      <Greeting user={name} />
      <button onClick={() => setName("Lavanya")}>Change Name</button>
    </div>
  );
}

export default App;
```

File: src/Greeting.js

```
import React from 'react';

function Greeting(props) {
  return <h2>Hello, {props.user}!</h2>;
}

export default Greeting;
```

How to Run:

npx create-react-app props-state-demo
cd props-state-demo
npm start

Expected Output:

Initial shows Hello, Sathwvik! then after click shows Hello, Lavanya!

ReactJS – Event Handling (Exact)

Question: Write a React JS program for responding to events.

File: src/App.js

```
import React, { useState } from 'react';

function App() {
  const [text, setText] = useState("");

  const handleClick = () => alert("Button Clicked!");
  const handleChange = (e) => setText(e.target.value);
  const handleSubmit = (e) => { e.preventDefault(); alert(`You entered:
${text}`); };

  return (
    <div style={{ textAlign: "center", marginTop: "40px" }}>
      <h2>React Event Handling Example</h2>
      <button onClick={handleClick}>Click Me</button>
      <hr />
      <form onSubmit={handleSubmit}>
        <input value={text} onChange={handleChange} placeholder="Type..." />
        <button type="submit">Submit</button>
      </form>
      <p>You typed: {text}</p>
    </div>
  );
}

export default App;
```

How to Run:

npx create-react-app event-demo
cd event-demo
npm start

Expected Output:

Button alert on click; input mirrors text; submit alerts the entered text.

ReactJS – JSX Markup (Exact)

Question: Write a React JS program for writing markup with JSX.

File: src/App.js

```
import React from 'react';

function App() {
  const name = "Sathwvik";
  const branch = "CSE";

  return (
    <div style={{ textAlign: "center", marginTop: "40px" }}>
      <h1>JSX Example</h1>
      <p>Hello, <strong>{name}</strong>!</p>
      <p>You are studying in the <em>{branch}</em> branch.</p>
      <p>JSX lets us write HTML-like code inside JavaScript!</p>
    </div>
  );
}

export default App;
```

How to Run:

npx create-react-app jsx-demo
cd jsx-demo
npm start

Expected Output:

Displays headings and lines with name and branch.

ReactJS – Routing with React Router (Exact)

Question: Write a React JS program for routing to different pages using react router.

File: src/App.js

```
import React from "react";
import { BrowserRouter as Router, Routes, Route, Link } from "react-router-
dom";
import Home from "./Home";
import About from "./About";
import Contact from "./Contact";

function App() {
  return (
    <Router>
      <div style={{ textAlign: "center", marginTop: "40px" }}>
        <h2>React Router Example</h2>
        <nav>
          <Link to="/">Home</Link> | <Link to="/about">About</Link> | <Link
to="/contact">Contact</Link>
        </nav>
        <hr />
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/contact" element={<Contact />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

File: src/Home.js

```
import React from "react";
export default function Home(){ return <h3>Welcome to the Home Page!</h3>; }
```

File: src/About.js

```
import React from "react";
export default function About(){ return <h3>This is the About Page.</h3>; }
```

File: src/Contact.js

```
import React from "react";
export default function Contact(){ return <h3>Contact us at
contact@example.com</h3>; }
```

How to Run:

npx create-react-app router-demo
cd router-demo
npm install react-router-dom
npm start

Expected Output:

Clicking links swaps content without page reload.

ReactJS – Updating Screen (useState) (Exact)

Question: Write a React JS program for updating the screen.

File: src/App.js

```
import React, { useState } from 'react';

function App() {
  const [count, setCount] = useState(0);
  return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      <h2>React Screen Update Example</h2>
      <p>Current Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase Count</button>
    </div>
  );
}

export default App;
```

How to Run:

npx create-react-app update-demo
cd update-demo
npm start

Expected Output:

Count increments on each click.

ReactJS – Conditional Rendering (Exact)

Question: Write a React JS program for conditional rendering.

File: src/App.js

```
import React, { useState } from 'react';

function App() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h2>Conditional Rendering Example</h2>
      {isLoggedIn ? <h3>Welcome, Sathwvik!</h3> : <h3>Please log in to
continue.</h3>}
      <button onClick={() => setIsLoggedIn(!isLoggedIn)}>
        {isLoggedIn ? "Logout" : "Login"}
      </button>
    </div>
  );
}

export default App;
```

How to Run:

npx create-react-app conditional-demo
cd conditional-demo
npm start

Expected Output:

Shows login prompt then welcome text after toggle.

ReactJS – Rendering Lists (Exact)

Question: Write a React JS program for rendering lists.

File: src/App.js

```
import React from 'react';

function App() {
  const students = ["Sathwvik", "Lavanya", "Ravi", "Teja"];
  return (
    <div style={{ textAlign: "center", marginTop: "40px" }}>
      <h2>Rendering List Example</h2>
      <ul>{students.map((name, i) => <li key={i}>{name}</li>)}</ul>
    </div>
  );
}

export default App;
```

How to Run:

npx create-react-app list-demo
cd list-demo
npm start

Expected Output:

List of names rendered as bullet points.

ReactJS – CSS & Sass Styling (Exact)

Question: Write a React JS program to add styles (CSS & Sass) and display data.

File: src/App.js

```
import React from 'react';
import './App.css';
import './App.scss';

function App() {
  const student = { name: "Sathwvik", branch: "CSE", year: "2nd Year" };
  return (
    <div className="container">
      <h1 className="title">Student Details</h1>
      <div className="card">
        <p><b>Name:</b> {student.name}</p>
        <p><b>Branch:</b> {student.branch}</p>
        <p><b>Year:</b> {student.year}</p>
      </div>
    </div>
  );
}
export default App;
```

File: src/App.css

```
.container { text-align: center; margin-top: 40px; }
.title { color: blue; }
```

File: src/App.scss

```
.card { background-color:#f5f5f5; border:2px solid #ddd; padding:20px;
margin:20px auto; width:250px; border-radius:10px; box-shadow:2px 2px 6px
gray;
  p { color:#333; font-size:16px; } }
```

How to Run:

npx create-react-app style-demo

cd style-demo

npm install sass

npm start

Expected Output:

Shows styled card with student details.

ReactJS – Creating & Nesting Components (Exact)

Question: Write a React JS program for creating and nesting components (function and class).

File: src/App.js

```
import React from "react";
import Welcome from "./Welcome";
import Message from "./Message";

function App() {
  return (
    <div style={{ textAlign: "center", marginTop: "40px" }}>
      <h2>React Components Example</h2>
      <Welcome />
      <Message />
    </div>
  );
}
export default App;
```

File: src/Welcome.js

```
import React from "react";
function Welcome(){ return <h3>Hello! I am a Function Component.</h3>; }
export default Welcome;
```

File: src/Message.js

```
import React, { Component } from "react";
class Message extends Component { render(){ return <p>This message is from a
Class Component.</p>; } }
export default Message;
```

How to Run:

npx create-react-app components-demo

cd components-demo

npm start

Expected Output:

Shows both function and class components.

MongoDB – CRUD (Exact User Code)

Question: Write MongoDB queries to perform CRUD operations on document using insert(), find(), update(), remove().

File: mongo_crud.js

```
const { MongoClient } = require("mongodb");
const url = "mongodb://127.0.0.1:27017";
const client = new MongoClient(url);

async function run() {
  await client.connect();
  const db = client.db("college");
  const students = db.collection("students");

  // CREATE
  await students.insertOne({ name: "Ravi", age: 20, course: "CSE" });
  console.log("Inserted!");

  // READ
  console.log(await students.find().toArray());

  // UPDATE
  await students.updateOne({ name: "Ravi" }, { $set: { course: "Data Science"
} });
  console.log("Updated!");

  // DELETE
  await students.deleteOne({ name: "Ravi" });
  console.log("Deleted!");

  await client.close();
}

run();
```

How to Run:

mongod
node mongo_crud.js

Expected Output:

Inserted!
[{ _id: <ObjectId>, name: 'Ravi', age: 20, course: 'CSE' }, ...]
Updated!
Deleted!

MongoDB – Create/Drop DB & Collections (Exact)

Question: Write MongoDB queries to Create and drop databases and collections.

File: mongo_db_ops.js

```
const { MongoClient } = require("mongodb");

const url = "mongodb://127.0.0.1:27017";
const client = new MongoClient(url);

async function run() {
  await client.connect();

  // ✅CREATE DATABASE
  const db = client.db("school");
  console.log("✅Database 'school' created!");

  // ✅CREATE COLLECTION
  await db.createCollection("students");
  console.log("✅Collection 'students' created!");

  // ✅INSERT SAMPLE DOCUMENT
  await db.collection("students").insertOne({ name: "Sathwvik", class: "CSE"
});
  console.log("📄 Sample document inserted!");

  // ✅DROP COLLECTION
  await db.collection("students").drop();
  console.log("🗑 Collection 'students' dropped!");

  // ✅DROP DATABASE
  await db.dropDatabase();
  console.log("🗑 Database 'school' dropped!");

  await client.close();
}

run();
```

How to Run:

mongod
node mongo_db_ops.js

Expected Output:

✅Database 'school' created!
✅Collection 'students' created!

 Sample document inserted!
 Collection 'students' dropped!
 Database 'school' dropped!

MongoDB – find(), limit(), sort(), createIndex(), aggregate() (Exact)

Question: Write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate().

File: mongo_records.js

```
const { MongoClient } = require("mongodb");

const url = "mongodb://127.0.0.1:27017";
const client = new MongoClient(url);

async function run() {
  await client.connect();
  const db = client.db("school");
  const students = db.collection("students");

  // Insert sample data
  await students.insertMany([
    { name: "Sathwvik", marks: 85, branch: "CSE" },
    { name: "Lavanya", marks: 90, branch: "ECE" },
    { name: "Ravi", marks: 75, branch: "IT" },
    { name: "Teja", marks: 95, branch: "CSE" },
  ]);
  console.log("✅Sample records inserted");

  // 1️⃣ FIND - Display all records
  console.log("📋 All Students:");
  console.log(await students.find().toArray());

  // 2️⃣ LIMIT - Show only 2 records
  console.log("📋 First 2 Students:");
  console.log(await students.find().limit(2).toArray());

  // 3️⃣ SORT - Sort by marks descending
  console.log("📋 Sorted by Marks (High to Low):");
  console.log(await students.find().sort({ marks: -1 }).toArray());

  // 4️⃣ CREATE INDEX - Create index on name
  await students.createIndex({ name: 1 });
  console.log("✅Index created on 'name' field");

  // 5️⃣ AGGREGATE - Average marks by branch
  const avgMarks = await students.aggregate([
    { $group: { _id: "$branch", avgMarks: { $avg: "$marks" } } }
  ]).toArray();
  console.log("📋 Average Marks by Branch:");
  console.log(avgMarks);
```

```
  await client.close();
}

run();
```

How to Run:

mongod
node mongo_records.js

Expected Output:

✅Sample records inserted
 All Students: [...4 docs...]
 First 2 Students: [first two docs]
 Sorted by Marks (High to Low): [Teja, Lavanya, Sathwvik, Ravi]
✅Index created on 'name' field
 Average Marks by Branch: [{ _id: 'CSE', avgMarks: 90 }, { _id: 'ECE', avgMarks: 90 }, { _id: 'IT', avgMarks: 75 }]