🧑‍💻 Day 13 Lab Sheet – Express.js Routing, Middleware & REST APIs

Objective

By the end of this lab, you will:

- Create routes in **Express.js** for different HTTP methods
- Implement **middleware** for logging and request handling
- Build a basic **CRUD REST API** (Create, Read, Update, Delete)
- Test APIs using **Postman**

Setup

1. Create a new folder:
2. mkdir Day13_ExpressAPI
3. cd Day13_ExpressAPI
4. Initialize npm and install Express:
5. npm init -y
6. npm install express
7. Create a file named app.js.

Task 1: Create a Basic Express Server

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

app.get('/', (req, res) => {
  res.send('Welcome to Express.js!');
});

app.listen(4000, () => {
  console.log('Server running at http://localhost:4000');
});
```

✅ Run:

```
node app.js
```

Visit http://localhost:4000 → "Welcome to Express.js!" appears.

Task 2: Basic Routes (GET, POST, PUT, DELETE)

Add the following in app.js:

```
app.get('/home', (req, res) => {
  res.send('This is Home Page');
});

app.post('/create', (req, res) => {
  res.send('Data created successfully');
});

app.put('/update', (req, res) => {
  res.send('Data updated successfully');
});

app.delete('/delete', (req, res) => {
  res.send('Data deleted successfully');
});
```

✅ Test using **Postman** → change HTTP method each time (GET, POST, PUT, DELETE).

Task 3: Route Parameters

```
app.get('/user/:id', (req, res) => {
  const id = req.params.id;
  res.send(`User ID: ${id}`);
});
```

✅ Visit http://localhost:4000/user/101 → shows User ID: 101.

Task 4: Query Parameters

```
app.get('/search', (req, res) => {
  const query = req.query.q;
  res.send(`You searched for: ${query}`);
});
```

✅ Visit /search?q=javascript → shows “You searched for: javascript”.

Task 5: Using Middleware

Step 1 – Create a logger middleware:

```
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next();
});
```

✅ Every request logs method + URL in console.

Step 2 – Parse JSON data:

```
app.use(express.json());
```

✅ Now Express can handle JSON input (useful for APIs).

Task 6: Build a CRUD REST API

Replace or add below code to app.js:

```
let users = [
  { id: 1, name: 'Alice' },
  { id: 2, name: 'Bob' }
];

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

// GET user by ID
app.get('/api/users/:id', (req, res) => {
  const user = users.find(u => u.id == req.params.id);
  if (!user) return res.status(404).send('User not found');
  res.json(user);
});

// POST - add new user
app.post('/api/users', (req, res) => {
  const newUser = {
    id: users.length + 1,
    name: req.body.name
  };
  users.push(newUser);
  res.status(201).json(newUser);
});

// PUT - update user
app.put('/api/users/:id', (req, res) => {
  const user = users.find(u => u.id == req.params.id);
  if (!user) return res.status(404).send('User not found');
  user.name = req.body.name;
  res.json(user);
});

// DELETE - remove user
app.delete('/api/users/:id', (req, res) => {
  users = users.filter(u => u.id != req.params.id);
  res.send('User deleted');
});
```

✅ Test each route in **Postman**:

Method	URL	Description	Body (for POST/PUT)
GET	/api/users	Get all users	—
GET	/api/users/1	Get user by ID	—
POST	/api/users	Add a new user	{ "name": "Charlie" }

Method	**URL**	**Description**	**Body (for POST/PUT)**
PUT	/api/users/1	Update user name	{ "name": "Updated Alice" }
DELETE	/api/users/2	Delete a user	—

Task 7: Error Handling Middleware

Add at the bottom of app.js:

```
app.use((err, req, res, next) => {
 console.error(err.stack);
 res.status(500).send('Something went wrong!');
});
```

✅ Simulate error (for example, throw an error inside any route) and check custom message.

Task 8: Organize Routes (Bonus Task)

1. Create a folder routes.
2. Inside, make userRoutes.js:
3. const express = require('express');
4. const router = express.Router();
5.
6. let users = [{ id: 1, name: 'Alice' }];
7.
8. router.get('/', (req, res) => res.json(users));
9. router.post('/', (req, res) => {
10. users.push({ id: users.length + 1, name: req.body.name });
11. res.status(201).send('User added');
12. });
13.
14. module.exports = router;
15. In app.js:
16. const userRoutes = require('./routes/userRoutes');
17. app.use('/api/users', userRoutes);

✅ Visit /api/users → should work using modular routing.

✅ Deliverables

- app.js (main app file)
- routes/userRoutes.js (if bonus attempted)
- Working CRUD API tested in Postman

Each route should correctly handle its function and return JSON responses.

💡 Optional Challenge

👉 Create a products API with routes /api/products and /api/products/:id implementing all 4 CRUD operations — similar to users, but using productName instead of name.