EXPRESS.JS

DEVELOPER

NOTES

👨🏻‍💻 Name: Mr. Rupesh Lal

🌐 Website: www.itmodem.com

📞 WhatsApp: +91 7870886355

💼 Profession: Software Engineer

**Subject: Express.js – Basics to Advanced**

**Chapter 1. What is Express?**

* Why use Express with Node?
* Setting up a project
* Basic Express server

**Chapter 2. Express Routing**

* app.get, app.post, app.put, app.delete
* Route parameters & query strings
* Express Router

**Chapter 3. Middleware**

* Built-in vs Custom middleware
* express.json(), express.urlencoded()
* Error-handling middleware
* next() usage

**Chapter 4. Templates & Static Files**

* Serving static files
* View engines (e.g., EJS)

**Chapter 5. REST API with Express**

* Building CRUD APIs
* Status codes
* Validation and error messages
* Postman testing

**Chapter 6. Express + MongoDB Integration**

* Connecting MongoDB with Mongoose
* Routes + DB CRUD Operations
* MVC Pattern

**Chapter 7. Authentication (Intro)**

* Sessions & Cookies
* JWT Basics

**🔸 Exercises: Simple routing, custom middleware**

**🔸 Mini Project: To-Do App API**

**🔸 Interview Qs: Difference between middleware and route handler?**

**📘 Chapter 1. What is Express?**

**✅ What is Express?**

* **Express.js** is a **minimal and flexible Node.js web application framework** that provides a robust set of features to build web and mobile applications.
* It’s a wrapper over Node’s native HTTP module—makes server-side development **faster and cleaner**.

🔸 Think of Express as the **“backend framework”** for Node.js, similar to how React is for the frontend.

**✅ Why Use Express with Node?**

| **Feature** | **Benefit** |
| --- | --- |
| 🚀 Simplicity | Clean and readable syntax |
| 🛠 Middleware Support | Easily add custom and third-party functions to the request/response cycle |
| 📦 Routing | Built-in support for URL routing |
| 📁 Static Files | Built-in way to serve static files |
| 🌐 REST APIs | Ideal for building RESTful APIs quickly |

🔥 **Real-world use:** Used in MERN stack (MongoDB, Express, React, Node.js)

**✅ Setting Up an Express Project**

**1. Initialize a new project**

mkdir express-app

cd express-app

npm init -y

**2. Install Express**

npm install express

**✅ Basic Express Server**

**Create index.js:**

const express = require('express');

const app = express();

*// Route*

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

res.send('Hello from Express!');

});

*// Start server*

app.listen(3000, () => {

console.log('Server running at http://localhost:3000');

});

**Run the server:**

node index.js

🟢 Open http://localhost:3000 in browser to see your response.

**✅ Explanation:**

| **Code** | **Purpose** |
| --- | --- |
| express() | Initializes the app |
| app.get() | Handles GET request at route / |
| res.send() | Sends back a response to the client |
| app.listen() | Starts the server on a given port |

**🧠 Pro Tip:**

Use nodemon for auto-reload on changes:

npm install --save-dev nodemon

npx nodemon index.js

**📘Chapter 2. Express Routing**

**✅ What is Routing?**

Routing defines how your app responds to **client requests** to a particular **URL path** and **HTTP method** (GET, POST, etc.).

**✅ 1. HTTP Methods in Express**

| **Method** | **Purpose** |
| --- | --- |
| GET | Read/Retrieve data |
| POST | Create new data |
| PUT | Update existing data |
| DELETE | Delete data |

**🔹 Basic Syntax:**

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

app.post('/path', (req, res) => { ... });

app.put('/path', (req, res) => { ... });

app.delete('/path', (req, res) => { ... });

**✅ Example:**

const express = require('express');

const app = express();

app.use(express.json()); *// to parse JSON request body*

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

res.send('Get all users');

});

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

res.send('Create a user');

});

app.put('/users/:id', (req, res) => {

res.send(`Update user with ID: ${req.params.id}`);

});

app.delete('/users/:id', (req, res) => {

res.send(`Delete user with ID: ${req.params.id}`);

});

app.listen(3000);

**✅ 2. Route Parameters**

Used to capture values from the URL (dynamic routing).

**Example:**

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

res.send(`Product ID: ${req.params.id}`);

});

🔹 :id is a route parameter. You can access it via req.params.id.

**✅ 3. Query Strings**

Passed in the URL after ? to filter or sort data.

**URL:** http://localhost:3000/search?term=node&page=2

**Route Example:**

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

const { term, page } = req.query;

res.send(`Search term: ${term}, Page: ${page}`);

});

✅ Access query string values via req.query.

**✅ 4. Express Router – Modular Routing**

Helps organize routes into separate files.

**Step 1: Create routes/userRoutes.js**

const express = require('express');

const router = express.Router();

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

res.send('All users');

});

router.post('/', (req, res) => {

res.send('Create user');

});

module.exports = router;

**Step 2: Import and use it in app.js**

const express = require('express');

const app = express();

const userRoutes = require('./routes/userRoutes');

app.use('/users', userRoutes);

app.listen(3000);

**🧠 Summary**

| **Concept** | **Key Point** |
| --- | --- |
| HTTP Methods | Define action: GET, POST, PUT, DELETE |
| Route Params | :id for dynamic values |
| Query Strings | req.query for filters/sorting |
| Express Router | Organize routes modularly |

**📘Chapter 3. Middleware in Express**

**✅ What is Middleware?**

* **Middleware** functions are functions that have access to the **request** and **response** objects, and the **next** function.
* They are executed in sequence during the request-response cycle.

**Basic Structure:**

(req, res, next) => {

*// Logic here*

next(); *// Pass control to next middleware*

}

**✅ 1. Built-in Middleware**

| **Middleware** | **Purpose** |
| --- | --- |
| express.json() | Parses incoming JSON payloads |
| express.urlencoded() | Parses URL-encoded form data |

**Example:**

const express = require('express');

const app = express();

app.use(express.json()); *// to parse JSON*

app.use(express.urlencoded({ extended: true })); *// for form data*

🔸 Always put .use() before routes.

**✅ 2. Custom Middleware**

You can write your own middleware to log, validate, or manipulate requests.

**Example: Logger Middleware:**

app.use((req, res, next) => {

console.log(`${req.method} ${req.url}`);

next(); *// call next middleware*

});

🔹 Custom middleware must call next() to proceed.

**✅ 3. Error-handling Middleware**

Express identifies error middleware by the **4 arguments**:

(err, req, res, next) => {

console.error(err.message);

res.status(500).send('Something went wrong!');

}

**Use case:**

app.use((err, req, res, next) => {

res.status(500).json({ error: err.message });

});

🔸 Always define **after all routes**.

**✅ 4. Using next()**

| **Situation** | **Use** |
| --- | --- |
| Normal middleware | next() moves to next handler |
| In error | next(err) passes error to error-handling middleware |

**Example with error:**

app.use((req, res, next) => {

const err = new Error('Not found');

next(err); *// Go to error middleware*

});

**✅ Middleware Execution Flow Example**

app.use((req, res, next) => {

console.log('Middleware 1');

next();

});

app.use((req, res, next) => {

console.log('Middleware 2');

next();

});

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

res.send('Final handler');

});

**Console output:**

Middleware 1

Middleware 2

Final handler

**🧠 Summary**

| **Type** | **Purpose** |
| --- | --- |
| Built-in | express.json(), express.urlencoded() |
| Custom | Logging, validation, auth |
| Error-handling | Handle thrown/caught errors |
| next() | Moves request through middleware stack |

**📘 4. Templates & Static Files**

**✅ 1. Serving Static Files in Express**

Static files include images, CSS, JS, HTML—files that don’t change dynamically.

**Step 1: Create a public/ folder:**

project/

├── public/

│ ├── index.html

│ ├── style.css

│ └── script.js

**Step 2: Serve static files:**

const express = require('express');

const app = express();

app.use(express.static('public'));

app.listen(3000);

🟢 Now you can access files like:

* http://localhost:3000/index.html
* http://localhost:3000/style.css

**✅ 2. What is a View Engine?**

* A **View Engine** lets you write HTML templates that include **dynamic data** using logic (loops, conditionals, etc.).
* Popular engines: **EJS**, Pug, Handlebars.

**✅ Using EJS with Express**

**Step 1: Install EJS**

npm install ejs

**Step 2: Set the view engine**

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

**Step 3: Create a views/ folder with .ejs files**

**views/home.ejs**

<!DOCTYPE html>

<html>

<head>

<title><%= title %></title>

</head>

<body>

<h1>Welcome, <%= user %>!</h1>

</body>

</html>

**Step 4: Render a template from route**

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

res.render('home', { title: 'Home Page', user: 'John' });

});

✅ res.render() injects dynamic values into your .ejs template.

**✅ EJS Syntax Summary**

| **Syntax** | **Purpose** |
| --- | --- |
| <%= variable %> | Output value |
| <% if () {} %> | Conditional rendering |
| <% for (...) { } %> | Loop over array |

**🧠 Summary**

| **Feature** | **Code** |
| --- | --- |
| Static files | app.use(express.static('public')) |
| Set view engine | app.set('view engine', 'ejs') |
| Render EJS | res.render('file', data) |

**📘 5. REST API with Express**

**✅ What is a REST API?**

A **REST API** (Representational State Transfer) is an architecture that uses standard HTTP methods to perform **CRUD operations**:

| **Method** | **Action** | **Example URL** |
| --- | --- | --- |
| GET | Read | /api/users |
| POST | Create | /api/users |
| PUT | Update | /api/users/:id |
| DELETE | Delete | /api/users/:id |

**✅ Basic Express REST API Example**

const express = require('express');

const app = express();

app.use(express.json());

let users = [

{ id: 1, name: 'Alice' },

{ id: 2, name: 'Bob' }

];

*// GET - Read all users*

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

res.json(users);

});

*// POST - Create user*

app.post('/api/users', (req, res) => {

const { name } = req.body;

const newUser = { id: users.length + 1, 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).json({ error: 'User not found' });

user.name = req.body.name;

res.json(user);

});

*// DELETE - Delete user*

app.delete('/api/users/:id', (req, res) => {

users = users.filter(u => u.id != req.params.id);

res.json({ message: 'User deleted' });

});

app.listen(3000, () => console.log('API running on port 3000'));

**✅ HTTP Status Codes (Important!)**

| **Code** | **Meaning** | **Use in REST** |
| --- | --- | --- |
| 200 | OK | GET, PUT, DELETE success |
| 201 | Created | POST success |
| 400 | Bad Request | Validation failed |
| 404 | Not Found | Resource not found |
| 500 | Internal Server Error | Unexpected issue |

**✅ Input Validation Example**

app.post('/api/users', (req, res) => {

if (!req.body.name || req.body.name.length < 3) {

return res.status(400).json({ error: 'Name must be at least 3 characters' });

}

const user = { id: users.length + 1, name: req.body.name };

users.push(user);

res.status(201).json(user);

});

**✅ Using Postman to Test APIs**

1. **Download Postman:** <https://www.postman.com/downloads/>
2. **Start your Express server**
3. In Postman:
   * Choose method: GET, POST, PUT, DELETE
   * Set URL: http://localhost:3000/api/users
   * For POST/PUT, go to **Body → raw → JSON**, and input:
   * { "name": "Charlie" }
   * Click **Send** and view response.

**✅ REST API Summary**

| **Task** | **Method** | **Endpoint** | **Code Example** |
| --- | --- | --- | --- |
| Read users | GET | /api/users | app.get() |
| Add user | POST | /api/users | app.post() |
| Update user | PUT | /api/users/:id | app.put() |
| Delete user | DELETE | /api/users/:id | app.delete() |

**📘 6. Express + MongoDB Integration**

**✅ 1. Connecting MongoDB with Mongoose**

**Mongoose** is an ODM (Object Data Modeling) library for MongoDB and Node.js. It simplifies schema creation, validation, and interaction with MongoDB.

**🔹 Install Mongoose**

npm install mongoose

**🔹 Connect to MongoDB**

const mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1:27017/expressdb', {

useNewUrlParser: true,

useUnifiedTopology: true

})

.then(() => console.log('MongoDB connected'))

.catch(err => console.error('MongoDB connection error:', err));

**✅ 2. Create Mongoose Model**

**models/User.js**

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

name: { type: String, required: true, minlength: 3 },

email: { type: String, required: true, unique: true },

age: { type: Number }

});

module.exports = mongoose.model('User', userSchema);

**✅ 3. Routes + CRUD Operations with MongoDB**

**routes/userRoutes.js**

const express = require('express');

const router = express.Router();

const User = require('../models/User');

*// GET all users*

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

const users = await User.find();

res.json(users);

});

*// POST new user*

router.post('/', async (req, res) => {

try {

const user = new User(req.body);

await user.save();

res.status(201).json(user);

} catch (err) {

res.status(400).json({ error: err.message });

}

});

*// PUT update user*

router.put('/:id', async (req, res) => {

try {

const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });

res.json(user);

} catch (err) {

res.status(400).json({ error: err.message });

}

});

*// DELETE user*

router.delete('/:id', async (req, res) => {

await User.findByIdAndDelete(req.params.id);

res.json({ message: 'User deleted' });

});

module.exports = router;

**✅ 4. Register Router in Express App**

**app.js**

const express = require('express');

const mongoose = require('mongoose');

const userRoutes = require('./routes/userRoutes');

const app = express();

app.use(express.json());

*// Connect DB*

mongoose.connect('mongodb://127.0.0.1:27017/expressdb')

.then(() => console.log('MongoDB connected'));

*// Use Routes*

app.use('/api/users', userRoutes);

app.listen(3000, () => console.log('Server running'));

**✅ 5. MVC Pattern in Express Apps**

| **Layer** | **Responsibility** | **Folder** |
| --- | --- | --- |
| **Model** | MongoDB Schema/Logic | /models |
| **View** | (Optional) EJS, etc. | /views |
| **Controller** | Request handling logic | /controllers |
| **Route** | Route definitions | /routes |
| **App** | Entry file | app.js |

**Example Folder Structure:**

/models/User.js

/routes/userRoutes.js

/controllers/userController.js

/app.js

🔸 MVC keeps your code **organized and scalable**.

**🧠 Summary**

| **Feature** | **Code / Tool** |
| --- | --- |
| ODM | mongoose |
| Connect to DB | mongoose.connect() |
| Schema/Model | mongoose.Schema() |
| CRUD | find(), save(), findByIdAndUpdate(), findByIdAndDelete() |
| Structure | MVC (Model-View-Controller) |

**📘 7. Authentication (Intro)**

**✅ What is Authentication?**

Authentication is the process of **verifying a user’s identity** (e.g., login with username/email and password).

In Express apps, common authentication methods include:

* **Sessions & Cookies**
* **JWT (JSON Web Tokens)**

**✅ 1. Sessions & Cookies**

**🔹 What are Cookies?**

* Cookies are small pieces of data stored on the **client (browser)**.
* They store session IDs or tokens used for authentication.

**🔹 What are Sessions?**

* Sessions store data **on the server**, tied to a session ID (stored in a cookie on the client).
* When a user logs in, a **session** is created and linked to that user.

**🔸 How to Use Sessions in Express**

**🧩 Install express-session**

npm install express-session

**🔐 Setup Example**

const session = require('express-session');

app.use(session({

secret: 'your-secret-key',

resave: false,

saveUninitialized: true,

cookie: { secure: false } *// Use `true` with HTTPS*

}));

**🔐 Create & Use Sessions**

*// Login route*

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

const { username } = req.body;

req.session.user = username;

res.send('Logged in');

});

*// Protected route*

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

if (req.session.user) {

res.send(`Welcome ${req.session.user}`);

} else {

res.status(401).send('Unauthorized');

}

});

**🔓 Logout**

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

req.session.destroy();

res.send('Logged out');

});

**✅ 2. JWT (JSON Web Tokens)**

JWT is a **stateless** authentication method using tokens instead of sessions.

**🔹 How JWT Works:**

1. User logs in.
2. Server creates a token (signed with secret key) and sends it to client.
3. Client stores it (localStorage or cookie).
4. On each request, client sends token in the Authorization header.
5. Server verifies token.

**🔸 JWT Example with jsonwebtoken**

**🧩 Install:**

npm install jsonwebtoken

**🔐 Generate Token**

const jwt = require('jsonwebtoken');

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

const user = { id: 1, name: 'John' };

const token = jwt.sign(user, 'secret123', { expiresIn: '1h' });

res.json({ token });

});

**🧾 Middleware to Verify Token**

const authenticate = (req, res, next) => {

const authHeader = req.headers['authorization'];

const token = authHeader?.split(' ')[1];

if (!token) return res.sendStatus(401);

jwt.verify(token, 'secret123', (err, user) => {

if (err) return res.sendStatus(403);

req.user = user;

next();

});

};

*// Protected route*

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

res.json({ message: 'Welcome to profile', user: req.user });

});

**🔐 Sessions vs JWT – Comparison**

| **Feature** | **Sessions** | **JWT** |
| --- | --- | --- |
| Storage | Server | Client (token) |
| Stateless | ❌ No | ✅ Yes |
| Scalable | ❌ Less | ✅ More |
| Security | Secure with HTTPS | Needs token protection |
| Use case | Web apps (server-rendered) | APIs (RESTful) |

**🧠 Summary**

| **Concept** | **Notes** |
| --- | --- |
| Cookie | Stored on browser |
| Session | Stored on server |
| JWT | Token stored on client; verified by server |
| Package | express-session, jsonwebtoken |

**📘 Exercises, Mini Project & Interview Questions**

**✅ Exercises**

**1. Simple Routing**

const express = require('express');

const app = express();

app.get('/', (req, res) => res.send('Home Page'));

app.get('/about', (req, res) => res.send('About Page'));

app.get('/contact', (req, res) => res.send('Contact Page'));

app.listen(3000, () => console.log('Server running on port 3000'));

**2. Custom Middleware**

const logger = (req, res, next) => {

console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);

next(); *// continue to next middleware/route*

};

app.use(logger);

Use with a route:

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

res.send('Profile Page');

});

**✅ Mini Project – To-Do App API (No DB)**

**🧩 File: todoApp.js**

const express = require('express');

const app = express();

app.use(express.json());

let todos = []; *// in-memory array*

*// Get all todos*

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

res.json(todos);

});

*// Add a new todo*

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

const todo = { id: Date.now(), task: req.body.task, done: false };

todos.push(todo);

res.status(201).json(todo);

});

*// Update a todo*

app.put('/todos/:id', (req, res) => {

const todo = todos.find(t => t.id == req.params.id);

if (!todo) return res.status(404).json({ error: 'Not found' });

todo.task = req.body.task ?? todo.task;

todo.done = req.body.done ?? todo.done;

res.json(todo);

});

*// Delete a todo*

app.delete('/todos/:id', (req, res) => {

todos = todos.filter(t => t.id != req.params.id);

res.json({ message: 'Deleted' });

});

app.listen(3000, () => console.log('To-Do API running'));

🧪 Test with Postman or browser:

* GET /todos
* POST /todos with JSON { "task": "Buy milk" }
* PUT /todos/:id
* DELETE /todos/:id

**✅ Interview Question**

**❓ What’s the difference between middleware and a route handler?**

| **Feature** | **Middleware** | **Route Handler** |
| --- | --- | --- |
| Purpose | Intercept & modify requests/responses | Respond to a specific HTTP request |
| Executes via | app.use() or before route | app.get(), app.post() etc. |
| Next() usage | Must call next() to continue chain | Ends request by sending a response |
| Example | Logger, auth checker, body parser | /api/users – returns users list |

**Example:**

*// Middleware*

app.use((req, res, next) => {

console.log('Middleware ran');

next();

});

*// Route handler*

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

res.send('Welcome Home');

});