# Step-by-Step Backend Development Roadmap with Express.js

## Step 1: Create a Basic Route

- Define a route that responds to a GET request at `/greet` and sends back "Hello, World!".  
- Example: Accessing `/greet` in the browser or using a tool like Postman should return the message "Hello, World!".  
- Hint: Use `app.get()` with the correct path and callback.

## Step 2: Handle Query Parameters

- Create a route `/search` that takes query parameters like `term` and `page`.  
- Example: Visiting `/search?term=express&page=1` should respond with "Searching for 'express' on page 1".  
- Hint: Use `req.query` to access the parameters from the URL.

## Step 3: Add Route Parameters

- Create a route `/users/:id` that accepts a dynamic user ID.  
- Example: Visiting `/users/42` should return "User ID: 42".  
- Hint: Use `req.params` to retrieve the parameter from the URL.

## Step 4: Implement Middleware

- Add a middleware that logs the HTTP method and URL of incoming requests.  
- Example: When accessing any route, you should see a log like "GET /users/42" in the console.  
- Hint: Middleware functions use `req`, `res`, and `next()`.

## Step 5: Process POST Data

- Create a route `/data` that accepts JSON data in a POST request.  
- Example: Sending a POST request with `{ "name": "Noam" }` should respond with "Received: Noam".  
- Hint: Use a body parser and access the `req.body` object.

## Step 6: Error Handling

- Implement a route that intentionally throws an error.  
- Add an error-handling middleware to catch and respond with "Something went wrong!".  
- Example: Visiting `/error` should return a 500 status with the error message.  
- Hint: Pass an error to `next()` in your route handler.

## Step 7: Handle Missing Routes

- Add a fallback route for handling undefined paths.  
- Example: Visiting `/nonexistent` should return "Route not found." with a 404 status.  
- Hint: Define a catch-all route using `app.use()`.

## Step 8: Serve Static Files

- Serve a directory of static files (e.g., images or HTML files).  
- Example: Placing an `index.html` in the directory should allow access via `/index.html`.  
- Hint: Use `express.static()` and point it to your static files folder.

## Step 9: Add Validation for Query Parameters

- Update the `/search` route to check if `term` and `page` are provided. Return an error message if they are missing.  
- Example: Accessing `/search?term=express` without `page` should return "Missing required parameters: page".  
- Hint: Validate `req.query` values and send a response if conditions are not met.

## Step 10: Enhance with Dynamic Responses

- Create a route `/time` that returns the current server time in a user-friendly format.  
- Example: Visiting `/time` should return "Current time: 14:30".  
- Hint: Use JavaScript's `Date` object to generate the time.

# Enhanced Backend Development Roadmap

## Project Folder Structure

For a scalable Express.js application, use the following folder structure:  
  
src/  
├── routes/ # Define route files  
├── controllers/ # Business logic for each route  
├── middleware/ # Custom middleware functions  
├── models/ # Database models (if applicable)  
├── utils/ # Utility/helper functions  
├── app.js # Express app and middleware setup  
└── server.js # Starts the server

## Steps to Implement Routers and Controllers

1. Define routes in the `routes/` directory for modularity.

Example: `src/routes/userRoutes.js`  
```javascript  
const express = require('express');  
const { getUser, createUser } = require('../controllers/userController');  
const router = express.Router();  
  
router.get('/:id', getUser);  
router.post('/', createUser);  
  
module.exports = router;  
```

2. Separate business logic into `controllers/` to keep routes clean.

Example: `src/controllers/userController.js`  
```javascript  
exports.getUser = (req, res) => {  
 const { id } = req.params;  
 res.send(`User ID: ${id}`);  
};  
  
exports.createUser = (req, res) => {  
 const { name } = req.body;  
 res.send(`User Created: ${name}`);  
};  
```