Complete Express.js Guide

A Comprehensive Tutorial with Practical Examples

Table of Contents

---

1. Introduction to Express.js

What is Express.js?

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications. Think of it as a toolkit that makes creating web servers much easier than using plain Node.js.

Key Benefits:

- Simple and easy to learn
- Fast development
- Large community and ecosystem
- Flexible and unopinionated
- Great for building APIs and web applications

Real-World Analogy: If Node.js is like having basic building blocks, Express.js is like having pre-made templates and tools that help you build faster and better.

---

2. Creating Your First Express App

2.1 Project Setup

Step 1: Create Project Directory

```bash
# Create a new folder for your project
mkdir my-express-app
cd my-express-app

# Initialize Node.js project
npm init -y
```

What this does:

- Creates a new folder
- Initializes a package.json file (stores project information)

Step 2: Install Express

```bash
npm install express
```

This downloads Express and adds it to your project's dependencies.

2.2 Understanding Dependencies

When you install Express, your `package.json` looks like this:

```json
{
  "name": "my-express-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

Additional Useful Dependencies:

```bash
# For auto-reloading during development
npm install --save-dev nodemon

# For parsing JSON data
npm install body-parser

# For handling file uploads
npm install multer

# For environment variables
npm install dotenv
```

2.3 Basic App Initialization

Create a file named `app.js`:

```javascript
// Import Express
const express = require('express');

// Create Express application
const app = express();

// Define a port number
const PORT = 3000;

// Create a simple route
app.get('/', (req, res) => {
  res.send('Hello, Express!');
});

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

Explanation Line by Line:

1. **`require('express')`**: Imports the Express library
2. **`express()`**: Creates a new Express application instance
3. **`PORT = 3000`**: Defines which port the server will listen on

4. `app.get()`: Creates a route that responds to GET requests
5. `app.listen()`: Starts the server and listens for requests

Run Your Server:

```bash
node app.js
```

Visit `http://localhost:3000` in your browser to see "Hello, Express!"

2.4 Understanding the Listen Method

The `listen()` method is what actually starts your server.

```javascript
// Basic syntax
app.listen(port, [hostname], [callback]);

// Example 1: Simple
app.listen(3000);

// Example 2: With callback
app.listen(3000, () => {
  console.log('Server started successfully!');
});

// Example 3: With hostname
app.listen(3000, 'localhost', () => {
  console.log('Server is running on localhost:3000');
});

// Example 4: Using environment variable
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

2.5 App Object Methods

The `app` object has many useful methods:

```javascript
```

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

// 1. HTTP Methods
app.get('/users', (req, res) => {
  res.send('Get all users');
});

app.post('/users', (req, res) => {
  res.send('Create a new user');
});

app.put('/users/:id', (req, res) => {
  res.send('Update user');
});

app.delete('/users/:id', (req, res) => {
  res.send('Delete user');
});

// 2. use() - Apply middleware
app.use(express.json()); // Parse JSON bodies

// 3. set() - Configure application settings
app.set('view engine', 'ejs');
app.set('port', 3000);

// 4. get() - Retrieve setting value
const port = app.get('port');

// 5. locals - Store application-wide variables
app.locals.title = 'My Express App';
app.locals.email = 'admin@example.com';

// 6. route() - Create chainable routes
app.route('/book')
  .get((req, res) => res.send('Get book'))
  .post((req, res) => res.send('Add book'))
  .put((req, res) => res.send('Update book'));

app.listen(3000);
```

3. Understanding Routes

3.1 Route Definition

A route is a way to respond to client requests at specific endpoints (URLs).

Basic Structure:

```
javascript

app.METHOD(PATH, HANDLER);
```

- **METHOD**: HTTP method (get, post, put, delete, patch)
- **PATH**: URL path
- **HANDLER**: Function to execute when route is matched

3.2 GET Routes

GET requests retrieve data from the server.

```
javascript
```

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

// Simple GET route
app.get('/', (req, res) => {
  res.send('Welcome to Homepage');
});

// GET route with HTML response
app.get('/about', (req, res) => {
  res.send('<h1>About Us</h1><p>This is the about page</p>');
});

// GET route with JSON response
app.get('/api/users', (req, res) => {
  const users = [
    { id: 1, name: 'John Doe', email: 'john@example.com' },
    { id: 2, name: 'Jane Smith', email: 'jane@example.com' }
  ];
  res.json(users);
});

// GET route with status code
app.get('/success', (req, res) => {
  res.status(200).send('Operation successful');
});

app.listen(3000);
```

3.3 POST Routes

POST requests send data to create new resources.

```javascript
```

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

// Middleware to parse JSON bodies
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// POST route to create user
app.post('/api/users', (req, res) => {
  const newUser = {
    id: Date.now(),
    name: req.body.name,
    email: req.body.email
  };

  console.log('Received:', newUser);

  res.status(201).json({
    message: 'User created successfully',
    user: newUser
  });
});

// POST route with validation
app.post('/api/login', (req, res) => {
  const { username, password } = req.body;

  if (!username || !password) {
    return res.status(400).json({
      error: 'Username and password are required'
    });
  }

  // Check credentials (simplified example)
  if (username === 'admin' && password === 'password') {
    res.json({
      message: 'Login successful',
      token: 'abc123xyz'
    });
  } else {
    res.status(401).json({
      error: 'Invalid credentials'
    });
  }
});
```

```
app.listen(3000);
```

Test with curl:

```bash
curl -X POST http://localhost:3000/api/users \
  -H "Content-Type: application/json" \
  -d '{"name":"John Doe","email":"john@example.com"}'
```

3.4 PUT Routes

PUT requests update existing resources.

```javascript
```

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

app.use(express.json());

// Sample data
let users = [
  { id: 1, name: 'John', email: 'john@example.com' },
  { id: 2, name: 'Jane', email: 'jane@example.com' }
];

// PUT route to update user
app.put('/api/users/:id', (req, res) => {
  const userId = parseInt(req.params.id);
  const userIndex = users.findIndex(u => u.id === userId);

  if (userIndex === -1) {
    return res.status(404).json({ error: 'User not found' });
  }

  // Update user
  users[userIndex] = {
    id: userId,
    name: req.body.name,
    email: req.body.email
  };

  res.json({
    message: 'User updated successfully',
    user: users[userIndex]
  });
});

app.listen(3000);
```

3.5 DELETE Routes

DELETE requests remove resources.

```javascript
```

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

let users = [
  { id: 1, name: 'John', email: 'john@example.com' },
  { id: 2, name: 'Jane', email: 'jane@example.com' }
];

// DELETE route
app.delete('/api/users/:id', (req, res) => {
  const userId = parseInt(req.params.id);
  const userIndex = users.findIndex(u => u.id === userId);

  if (userIndex === -1) {
    return res.status(404).json({ error: 'User not found' });
  }

  const deletedUser = users.splice(userIndex, 1)[0];

  res.json({
    message: 'User deleted successfully',
    user: deletedUser
  });
});

app.listen(3000);
```

3.6 PATCH Routes

PATCH requests partially update resources.

```javascript
```

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

app.use(express.json());

let users = [
  { id: 1, name: 'John', email: 'john@example.com', age: 30 }
];

// PATCH route - update only specific fields
app.patch('/api/users/:id', (req, res) => {
  const userId = parseInt(req.params.id);
  const user = users.find(u => u.id === userId);

  if (!user) {
    return res.status(404).json({ error: 'User not found' });
  }

  // Update only provided fields
  if (req.body.name) user.name = req.body.name;
  if (req.body.email) user.email = req.body.email;
  if (req.body.age) user.age = req.body.age;

  res.json({
    message: 'User updated partially',
    user: user
  });
});

app.listen(3000);
```

Difference between PUT and PATCH:

- **PUT**: Replaces entire resource (all fields required)
- **PATCH**: Updates specific fields (only changed fields needed)

3.7 Route Parameters

Route parameters are named URL segments used to capture values.

```javascript
```

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

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

// Multiple parameters
app.get('/users/:userId/posts/:postId', (req, res) => {
  const { userId, postId } = req.params;
  res.json({
    message: 'Fetching post',
    userId: userId,
    postId: postId
  });
});

// Optional parameters using regex
app.get('/users/:id(\\d+)', (req, res) => {
  // Only matches if id is a number
  res.send(`User ID (number only): ${req.params.id}`);
});

// Parameter with custom name
app.get('/files/:filename.:format', (req, res) => {
  const { filename, format } = req.params;
  res.send(`File: ${filename}.${format}`);
});

app.listen(3000);
```

Examples:

- `/users/123` → userId = "123"
- `/users/5/posts/42` → userId = "5", postId = "42"
- `/files/report.pdf` → filename = "report", format = "pdf"

3.8 Query Strings

Query strings pass data via URL after the `?` symbol.

```
javascript
```

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

// Simple query string
app.get('/search', (req, res) => {
  const query = req.query.q;
  res.send(`Searching for: ${query}`);
});
// URL: /search?q=express

// Multiple query parameters
app.get('/products', (req, res) => {
  const { category, minPrice, maxPrice, sort } = req.query;

  res.json({
    category: category || 'all',
    priceRange: {
      min: minPrice || 0,
      max: maxPrice || 'unlimited'
    },
    sortBy: sort || 'name'
  });
});
// URL: /products?category=electronics&minPrice=100&maxPrice=500&sort=price

// Query with defaults
app.get('/api/users', (req, res) => {
  const page = parseInt(req.query.page) || 1;
  const limit = parseInt(req.query.limit) || 10;
  const sortBy = req.query.sortBy || 'name';

  res.json({
    message: 'Fetching users',
    pagination: {
      page: page,
      limit: limit,
      sortBy: sortBy
    }
  });
});
// URL: /api/users?page=2&limit=20&sortBy=email

app.listen(3000);
```

Difference between Params and Query:

- **Params**: `/users/:id` → Part of the path, required
- **Query**: `/users?id=123` → After `?`, optional

3.9 Route Matching

Express matches routes in the order they are defined.

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

// Specific route (checked first)
app.get('/users/admin', (req, res) => {
  res.send('Admin user page');
});

// Parameter route (checked second)
app.get('/users/:id', (req, res) => {
  res.send(`User ID: ${req.params.id}`);
});

// Wildcard route (checked last)
app.get('/users/*', (req, res) => {
  res.send('Some user-related page');
});

// Pattern matching with regex
app.get(/^\/users\/([0-9]+)$/, (req, res) => {
  res.send(`User with numeric ID: ${req.params[0]}`);
});

// Multiple paths for same handler
app.get(['/about', '/about-us'], (req, res) => {
  res.send('About Us page');
});

app.listen(3000);
```

Important Rule: Always put specific routes before generic ones!

3.10 Method Chaining

Handle multiple HTTP methods on the same route.

```javascript
```

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

app.use(express.json());

// Method 1: Individual routes
app.get('/user', (req, res) => res.send('Get user'));
app.post('/user', (req, res) => res.send('Create user'));
app.put('/user', (req, res) => res.send('Update user'));
app.delete('/user', (req, res) => res.send('Delete user'));

// Method 2: Using app.route() - Cleaner!
app.route('/product')
  .get((req, res) => {
    res.json({ message: 'Get product' });
  })
  .post((req, res) => {
    res.json({ message: 'Create product' });
  })
  .put((req, res) => {
    res.json({ message: 'Update product' });
  })
  .delete((req, res) => {
    res.json({ message: 'Delete product' });
  });

// Method 3: With parameters
app.route('/books/:id')
  .get((req, res) => {
    res.send(`Get book ${req.params.id}`);
  })
  .put((req, res) => {
    res.send(`Update book ${req.params.id}`);
  })
  .delete((req, res) => {
    res.send(`Delete book ${req.params.id}`);
  });

app.listen(3000);
```

Benefits of Method Chaining:

- Less code repetition
- Better organization
- Easier to maintain

3.11 Complete CRUD Operations Example

javascript

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

app.use(express.json());

// In-memory database (array)
let books = [
  { id: 1, title: '1984', author: 'George Orwell', year: 1949 },
  { id: 2, title: 'To Kill a Mockingbird', author: 'Harper Lee', year: 1960 }
];

// CREATE - Add new book
app.post('/api/books', (req, res) => {
  const newBook = {
    id: books.length + 1,
    title: req.body.title,
    author: req.body.author,
    year: req.body.year
  };

  books.push(newBook);

  res.status(201).json({
    message: 'Book created successfully',
    book: newBook
  });
});

// READ - Get all books
app.get('/api/books', (req, res) => {
  res.json({
    count: books.length,
    books: books
  });
});

// READ - Get single book
app.get('/api/books/:id', (req, res) => {
  const book = books.find(b => b.id === parseInt(req.params.id));

  if (!book) {
    return res.status(404).json({ error: 'Book not found' });
  }

  res.json(book);
});
```

```
// UPDATE - Update entire book
app.put('/api/books/:id', (req, res) => {
  const bookIndex = books.findIndex(b => b.id === parseInt(req.params.id));

  if (bookIndex === -1) {
    return res.status(404).json({ error: 'Book not found' });
  }

  books[bookIndex] = {
    id: parseInt(req.params.id),
    title: req.body.title,
    author: req.body.author,
    year: req.body.year
  };

  res.json({
    message: 'Book updated successfully',
    book: books[bookIndex]
  });
});

// UPDATE - Partial update
app.patch('/api/books/:id', (req, res) => {
  const book = books.find(b => b.id === parseInt(req.params.id));

  if (!book) {
    return res.status(404).json({ error: 'Book not found' });
  }

  if (req.body.title) book.title = req.body.title;
  if (req.body.author) book.author = req.body.author;
  if (req.body.year) book.year = req.body.year;

  res.json({
    message: 'Book updated partially',
    book: book
  });
});

// DELETE - Remove book
app.delete('/api/books/:id', (req, res) => {
  const bookIndex = books.findIndex(b => b.id === parseInt(req.params.id));

  if (bookIndex === -1) {
    return res.status(404).json({ error: 'Book not found' });
  }
```

```
  const deletedBook = books.splice(bookIndex, 1)[0];

  res.json({
    message: 'Book deleted successfully',
    book: deletedBook
  });
});

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

---

4. Working with Middleware

4.1 What is Middleware?

Middleware functions are functions that have access to the request (`req`), response (`res`), and the next middleware function (`next`) in the application's request-response cycle.

Think of middleware as checkpoints: Just like security checkpoints at an airport, middleware checks and processes requests before they reach their final destination (your route handler).

Simple Example:

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

// This is middleware!
app.use((req, res, next) => {
  console.log('Request received at:', new Date().toISOString());
  next(); // Pass control to next middleware
});

app.get('/', (req, res) => {
  res.send('Hello World');
});

app.listen(3000);
```

4.2 Understanding the Middleware Stack

Requests flow through middleware in order:

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

// Middleware 1
app.use((req, res, next) => {
  console.log('1. First middleware');
  next();
});

// Middleware 2
app.use((req, res, next) => {
  console.log('2. Second middleware');
  next();
});

// Middleware 3
app.use((req, res, next) => {
  console.log('3. Third middleware');
  next();
});

// Route handler (final destination)
app.get('/', (req, res) => {
  console.log('4. Route handler');
  res.send('Response sent');
});

app.listen(3000);

// Console output when visiting /:
// 1. First middleware
// 2. Second middleware
// 3. Third middleware
// 4. Route handler
```

4.3 The next() Function

The `next()` function passes control to the next middleware.

```javascript
```

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

// Middleware that calls next()
app.use((req, res, next) => {
  console.log('Processing...');
  next(); // Continue to next middleware
});

// Middleware that stops the chain
app.use((req, res, next) => {
  if (req.query.stop === 'true') {
    return res.send('Stopped here!');
    // next() is NOT called, so chain stops
  }
  next();
});

app.get('/', (req, res) => {
  res.send('Reached the route!');
});

app.listen(3000);

// Try: http://localhost:3000/ → "Reached the route!"
// Try: http://localhost:3000/?stop=true → "Stopped here!"
```

Important Rules:

1. Always call `next()` unless you're sending a response
2. Don't call `next()` after sending a response
3. Don't call `next()` multiple times

4.4 Types of Middleware

4.4.1 Application-Level Middleware

Bound to the app object, runs for all routes.

```
javascript
```

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

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

// Runs for routes starting with /api
app.use('/api', (req, res, next) => {
  console.log('API request received');
  next();
});

app.get('/', (req, res) => res.send('Home'));
app.get('/api/users', (req, res) => res.send('Users API'));

app.listen(3000);
```

4.4.2 Router-Level Middleware

Works same as application-level but bound to router.

```javascript
```

```javascript
const express = require('express');
const app = express();
const router = express.Router();

// Middleware for this router only
router.use((req, res, next) => {
  console.log('Router middleware');
  next();
});

router.get('/products', (req, res) => {
  res.send('Products list');
});

router.get('/products/:id', (req, res) => {
  res.send(`Product ${req.params.id}`);
});

// Mount router
app.use('/api', router);

app.listen(3000);
```

4.4.3 Built-in Middleware

Express provides some built-in middleware:

```javascript
javascript
```

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

// 1. Parse JSON bodies
app.use(express.json());

// 2. Parse URL-encoded bodies (form data)
app.use(express.urlencoded({ extended: true }));

// 3. Serve static files
app.use(express.static('public'));

// 4. Serve static files with prefix
app.use('/assets', express.static('public'));

app.post('/api/data', (req, res) => {
  console.log(req.body); // Now you can access POST data
  res.json(req.body);
});

app.listen(3000);
```

4.4.4 Third-Party Middleware

Popular middleware from npm:

```javascript
```

```javascript
const express = require('express');
const morgan = require('morgan'); // Logging
const cors = require('cors'); // Cross-origin requests
const helmet = require('helmet'); // Security headers
const compression = require('compression'); // Compress responses

const app = express();

// Install: npm install morgan cors helmet compression

// Logging middleware
app.use(morgan('dev'));

// Enable CORS
app.use(cors());

// Security headers
app.use(helmet());

// Compress responses
app.use(compression());

// Your routes here
app.get('/', (req, res) => {
  res.send('Hello with middleware!');
});

app.listen(3000);
```

4.4.5 Route-Specific Middleware

Apply middleware to specific routes only:

```javascript
```

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

// Authentication middleware
const authenticate = (req, res, next) => {
  const token = req.headers.authorization;

  if (token === 'secret-token') {
    next(); // Authenticated, continue
  } else {
    res.status(401).json({ error: 'Unauthorized' });
  }
};

// Logging middleware
const logRequest = (req, res, next) => {
  console.log(`${req.method} ${req.url} at ${new Date().toISOString()}`);
  next();
};

// Public route (no middleware)
app.get('/', (req, res) => {
  res.send('Public homepage');
});

// Protected route (with authentication)
app.get('/dashboard', authenticate, (req, res) => {
  res.send('Private dashboard');
});

// Route with multiple middleware
app.get('/admin', [authenticate, logRequest], (req, res) => {
  res.send('Admin panel');
});

// Route-specific middleware for POST
app.post('/api/data', logRequest, (req, res) => {
  res.send('Data received');
});

app.listen(3000);
```

4.5 Error-Handling Middleware

Error-handling middleware has 4 parameters: `err, req, res, next`.

javascript

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

app.use(express.json());

// Regular routes
app.get('/', (req, res) => {
  res.send('Homepage');
});

app.get('/error', (req, res) => {
  throw new Error('Something went wrong!');
});

app.post('/api/user', (req, res, next) => {
  if (!req.body.name) {
    const error = new Error('Name is required');
    error.status = 400;
    return next(error); // Pass error to error handler
  }
  res.json({ message: 'User created' });
});

// 404 Handler (no route matched)
app.use((req, res, next) => {
  res.status(404).json({
    error: 'Route not found',
    path: req.url
  });
});

// Error handling middleware (MUST be last)
app.use((err, req, res, next) => {
  console.error('Error:', err.message);

  res.status(err.status || 500).json({
    error: err.message || 'Internal Server Error',
    path: req.url,
    timestamp: new Date().toISOString()
  });
});

app.listen(3000);
```

Key Points:

- Error middleware must have 4 parameters
- Must be defined AFTER all other routes and middleware
- Catches errors from all previous middleware and routes

4.6 Middleware Ordering

CRITICAL: Order matters! Middleware executes in the order it's defined.

```javascript
```

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

// ❌ WRONG - This won't work!
app.get('/api/data', (req, res) => {
  res.json(req.body); // req.body is undefined!
});

app.use(express.json()); // Too late!

// ✅ CORRECT - Middleware before routes
const express = require('express');
const app = express();

// 1. Body parsing middleware first
app.use(express.json());

// 2. Logging middleware
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next();
});

// 3. Routes come after middleware
app.get('/api/data', (req, res) => {
  res.json(req.body); // Now req.body works!
});

// 4. 404 handler
app.use((req, res) => {
  res.status(404).send('Not found');
});

// 5. Error handler (always last!)
app.use((err, req, res, next) => {
  res.status(500).send('Server error');
});

app.listen(3000);
```

Correct Order:

1. Built-in middleware (express.json(), etc.)
2. Third-party middleware (morgan, cors, etc.)
3. Custom middleware

4. Routes

5. 404 handler

6. Error handler

4.7 Creating Custom Middleware

Example 1: Request Logger

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

// Custom logging middleware
const requestLogger = (req, res, next) => {
  const startTime = Date.now();

  // Log when request arrives
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);

  // Log when response is sent
  res.on('finish', () => {
    const duration = Date.now() - startTime;
    console.log(`Response sent in ${duration}ms with status ${res.statusCode}`);
  });

  next();
};

app.use(requestLogger);

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

app.listen(3000);
```

Example 2: Authentication Middleware

```javascript
```

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

// Authentication middleware
const requireAuth = (req, res, next) => {
  const token = req.headers['authorization'];

  if (!token) {
    return res.status(401).json({
      error: 'No token provided'
    });
  }

  // Simple token check (in real app, verify JWT)
  if (token === 'Bearer valid-token-123') {
    req.user = { id: 1, name: 'John Doe' }; // Attach user to request
    next();
  } else {
    res.status(403).json({
      error: 'Invalid token'
    });
  }
};

// Public route
app.get('/', (req, res) => {
  res.send('Public page');
});

// Protected route
app.get('/profile', requireAuth, (req, res) => {
  res.json({
    message: 'This is your profile',
    user: req.user
  });
});

app.listen(3000);
```

Example 3: Request Validation Middleware

```javascript
```

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

app.use(express.json());

// Validation middleware factory
const validateBody = (requiredFields) => {
  return (req, res, next) => {
    const missingFields = [];

    requiredFields.forEach(field => {
      if (!req.body[field]) {
        missingFields.push(field);
      }
    });

    if (missingFields.length > 0) {
      return res.status(400).json({
        error: 'Missing required fields',
        fields: missingFields
      });
    }

    next();
  };
};

// Email validation middleware
const validateEmail = (req, res, next) => {
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

  if (req.body.email && !emailRegex.test(req.body.email)) {
    return res.status(400).json({
      error: 'Invalid email format'
    });
  }

  next();
};

// Use validation middleware
app.post('/api/register',
  validateBody(['name', 'email', 'password']),
  validateEmail,
  (req, res) => {
    res.json({
```

```
    message: 'User registered successfully',
    user: {
      name: req.body.name,
      email: req.body.email
    }
  });
 }
);

app.listen(3000);
```

Example 4: Rate Limiting Middleware

javascript

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

// Simple rate limiter
const rateLimiter = () => {
  const requests = new Map();

  return (req, res, next) => {
    const ip = req.ip;
    const now = Date.now();
    const windowMs = 60000; // 1 minute
    const maxRequests = 10;

    if (!requests.has(ip)) {
      requests.set(ip, []);
    }

    const userRequests = requests.get(ip);

    // Remove old requests
    const recentRequests = userRequests.filter(
      time => now - time < windowMs
    );

    if (recentRequests.length >= maxRequests) {
      return res.status(429).json({
        error: 'Too many requests, please try again later'
      });
    }

    recentRequests.push(now);
    requests.set(ip, recentRequests);

    next();
  };
};

app.use(rateLimiter());

app.get('/api/data', (req, res) => {
  res.json({ message: 'Data retrieved' });
});

app.listen(3000);
```

5. Serving Static Files

5.1 Understanding Static Files

Static files are files that don't change: images, CSS, JavaScript, PDFs, etc.

Project Structure:

```
my-app/
├── app.js
├── public/
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── script.js
│   ├── images/
│   │   └── logo.png
│   └── index.html
```

5.2 Using express.static()

javascript

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

// Serve files from 'public' directory
app.use(express.static('public'));

// Now accessible:
// http://localhost:3000/css/style.css
// http://localhost:3000/js/script.js
// http://localhost:3000/images/logo.png
// http://localhost:3000/index.html

app.listen(3000);
```

5.3 Setting Up Public Directories

Basic Setup

javascript

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

// Method 1: Relative path
app.use(express.static('public'));

// Method 2: Absolute path (recommended)
app.use(express.static(path.join(__dirname, 'public')));

// Method 3: Multiple static directories
app.use(express.static('public'));
app.use(express.static('files'));
app.use(express.static('uploads'));

app.listen(3000);
```

With URL Prefix

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

// Serve static files with '/static' prefix
app.use('/static', express.static('public'));

// Now accessible at:
// http://localhost:3000/static/css/style.css
// http://localhost:3000/static/js/script.js

// Different folders with different prefixes
app.use('/assets', express.static('public'));
app.use('/uploads', express.static('uploads'));
app.use('/downloads', express.static('downloads'));

app.listen(3000);
```

5.4 Static File Caching

```javascript
```

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

// Cache static files for 1 day
app.use(express.static('public', {
  maxAge: '1d' // 1 day
}));

// Advanced caching options
app.use(express.static('public', {
  maxAge: '1d',          // Cache duration
  etag: true,            // Enable ETag
  lastModified: true,    // Enable Last-Modified header
  setHeaders: (res, path) => {
    if (path.endsWith('.html')) {
      res.setHeader('Cache-Control', 'no-cache');
    }
  }
}));

app.listen(3000);
```

5.5 Complete Static Files Example

Project Structure:

```
my-app/
├── app.js
├── public/
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── app.js
│   ├── images/
│   │   └── logo.png
│   └── index.html
```

app.js:

javascript

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

// Serve static files
app.use(express.static(path.join(__dirname, 'public')));

// API routes
app.get('/api/hello', (req, res) => {
  res.json({ message: 'Hello from API' });
});

// Fallback for SPA (Single Page Application)
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

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

public/index.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Express Static Files</title>
  <link rel="stylesheet" href="/css/style.css">
</head>
<body>
  <div class="container">
    <img src="/images/logo.png" alt="Logo">
    <h1>Welcome to Express</h1>
    <p>This page is served as a static file!</p>
    <button id="fetchBtn">Fetch API Data</button>
    <div id="result"></div>
  </div>

  <script src="/js/app.js"></script>
</body>
</html>
```

public/css/style.css:

```
css
```

```css
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 20px;
  background-color: #f0f0f0;
}

.container {
  max-width: 800px;
  margin: 0 auto;
  background: white;
  padding: 30px;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

img {
  max-width: 200px;
  display: block;
  margin: 0 auto 20px;
}

h1 {
  color: #333;
  text-align: center;
}

button {
  background-color: #007bff;
  color: white;
  border: none;
  padding: 10px 20px;
  border-radius: 4px;
  cursor: pointer;
  font-size: 16px;
}

button:hover {
  background-color: #0056b3;
}

#result {
  margin-top: 20px;
  padding: 15px;
  background-color: #e9ecef;
```

```
  border-radius: 4px;
}
```

public/js/app.js:

```javascript
document.getElementById('fetchBtn').addEventListener('click', async () => {
  try {
    const response = await fetch('/api/hello');
    const data = await response.json();

    document.getElementById('result').innerHTML = `
      <strong>API Response:</strong><br>
      ${data.message}
    `;
  } catch (error) {
    console.error('Error:', error);
    document.getElementById('result').innerHTML =
      '<strong>Error fetching data</strong>';
  }
});
```

---

6. Template Engines

6.1 What are Template Engines?

Template engines allow you to generate HTML dynamically by injecting data into templates. Think of them as HTML with superpowers!

Popular Template Engines:

- **EJS** (Embedded JavaScript) - Easy to learn, uses JavaScript
- **Pug** (formerly Jade) - Clean syntax, no closing tags
- **Handlebars** - Logic-less templates

6.2 Setting Up EJS

Install EJS:

```bash
npm install ejs
```

Project Structure:

```
my-app/
├── app.js
├── views/
│   ├── index.ejs
│   ├── about.ejs
│   └── partials/
│       ├── header.ejs
│       └── footer.ejs
```

app.js:

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

// Set EJS as template engine
app.set('view engine', 'ejs');

// Set views directory (optional, default is 'views')
app.set('views', path.join(__dirname, 'views'));

// Render EJS template
app.get('/', (req, res) => {
  res.render('index', {
    title: 'Home Page',
    message: 'Welcome to Express with EJS!',
    user: { name: 'John Doe', age: 30 }
  });
});

app.get('/about', (req, res) => {
  res.render('about', {
    title: 'About Us',
    company: 'My Company'
  });
});

app.listen(3000);
```

6.3 EJS Syntax and Examples

views/index.ejs:

```html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title><%= title %></title>
  <style>
    body {
      font-family: Arial, sans-serif;
      max-width: 800px;
      margin: 50px auto;
      padding: 20px;
    }
    .user-card {
      background: #f0f0f0;
      padding: 20px;
      border-radius: 8px;
      margin: 20px 0;
    }
  </style>
</head>
<body>
  <h1><%= message %></h1>

  <!-- Output variable -->
  <p>Page title: <%= title %></p>

  <!-- Output with HTML escaping (safe) -->
  <p>User name: <%= user.name %></p>

  <!-- Output without escaping (dangerous if user input!) -->
  <p>Raw HTML: <%- "<strong>Bold text</strong>" %></p>

  <!-- JavaScript code (no output) -->
  <% const greeting = "Hello"; %>
  <% const items = ['Apple', 'Banana', 'Orange']; %>

  <!-- Conditionals -->
  <% if (user.age >= 18) { %>
    <p><%= user.name %> is an adult</p>
  <% } else { %>
    <p><%= user.name %> is a minor</p>
  <% } %>

  <!-- Loops -->
  <h2>Fruits List:</h2>
```

```
  <ul>
    <% items.forEach(item => { %>
      <li><%= item %></li>
    <% }); %>
  </ul>

  <!-- User card -->
  <div class="user-card">
    <h3>User Information</h3>
    <p><strong>Name:</strong> <%= user.name %></p>
    <p><strong>Age:</strong> <%= user.age %></p>
  </div>
</body>
</html>
```

6.4 EJS Partials (Reusable Components)

views/partials/header.ejs:

```
html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title><%= title %></title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
    }
    nav {
      background: #333;
      color: white;
      padding: 15px;
    }
    nav a {
      color: white;
      text-decoration: none;
      margin: 0 15px;
    }
    .container {
      max-width: 1200px;
      margin: 0 auto;
      padding: 20px;
    }
  </style>
</head>
<body>
  <nav>
    <a href="/">Home</a>
    <a href="/about">About</a>
    <a href="/contact">Contact</a>
  </nav>
  <div class="container">
```

views/partials/footer.ejs:

```
html
```

```
  </div>
  <footer style="background: #333; color: white; text-align: center; padding: 20px; margin-top: 50px;">
    <p>&copy; 2024 My Website. All rights reserved.</p>
  </footer>
</body>
</html>
```

views/index.ejs (using partials):

```
html

<%- include('partials/header', { title: 'Home Page' }) %>

<h1>Welcome to Home Page</h1>
<p>This is the main content area.</p>

<%- include('partials/footer') %>
```

6.5 Complete EJS Example with Data

app.js:

```
javascript
```

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

app.set('view engine', 'ejs');
app.use(express.static('public'));

// Sample data
const products = [
  { id: 1, name: 'Laptop', price: 999, inStock: true },
  { id: 2, name: 'Mouse', price: 29, inStock: true },
  { id: 3, name: 'Keyboard', price: 79, inStock: false },
  { id: 4, name: 'Monitor', price: 299, inStock: true }
];

// Home page
app.get('/', (req, res) => {
  res.render('home', {
    title: 'Home',
    heading: 'Welcome to Our Store'
  });
});

// Products page
app.get('/products', (req, res) => {
  res.render('products', {
    title: 'Products',
    products: products
  });
});

// Single product page
app.get('/products/:id', (req, res) => {
  const product = products.find(p => p.id === parseInt(req.params.id));

  if (!product) {
    return res.status(404).render('error', {
      title: 'Error',
      message: 'Product not found'
    });
  }

  res.render('product-detail', {
    title: product.name,
    product: product
  });
});
```

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

views/products.ejs:

```
html

<%- include('partials/header', { title: title }) %>

<h1>Our Products</h1>

<div style="display: grid; grid-template-columns: repeat(auto-fill, minmax(250px, 1fr)); gap: 20px;">
  <% products.forEach(product => { %>
    <div style="border: 1px solid #ddd; padding: 20px; border-radius: 8px;">
      <h3><%= product.name %></h3>
      <p><strong>Price:</strong> $<%= product.price %></p>

      <% if (product.inStock) { %>
        <p style="color: green;">✓ In Stock</p>
      <% } else { %>
        <p style="color: red;">✗ Out of Stock</p>
      <% } %>

      <a href="/products/<%= product.id %>" style="display: inline-block; background: #007bff; color: white; padding:
        View Details
      </a>
    </div>
  <% }); %>
</div>

<% if (products.length === 0) { %>
  <p>No products available at the moment.</p>
<% } %>

<%- include('partials/footer') %>
```

6.6 Setting Up Pug

Install Pug:

```
bash

npm install pug
```

app.js:

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

// Set Pug as template engine
app.set('view engine', 'pug');
app.set('views', './views');

app.get('/', (req, res) => {
  res.render('index', {
    title: 'Home Page',
    message: 'Welcome to Pug!'
  });
});

app.listen(3000);
```

views/index.pug:

```pug
```

```
doctype html
html(lang="en")
  head
    meta(charset="UTF-8")
    meta(name="viewport" content="width=device-width, initial-scale=1.0")
    title= title
    style.
      body {
        font-family: Arial, sans-serif;
        max-width: 800px;
        margin: 50px auto;
        padding: 20px;
      }
  body
    h1= message
    p This is a Pug template

    //- Conditionals
    - const isLoggedIn = true
    if isLoggedIn
      p Welcome back, user!
    else
      p Please log in

    //- Loops
    ul
      each item in ['Apple', 'Banana', 'Orange']
        li= item
```

---

7. Practical Project: Blog Server

Let's build a complete blog server with all the concepts we've learned!

Project Structure:

```
blog-server/
├── app.js
├── package.json
├── public/
│   ├── css/
│   │   └── style.css
│   └── images/
│       └── logo.png
├── views/
│   ├── index.ejs
│   ├── post.ejs
│   ├── create.ejs
│   └── partials/
│       ├── header.ejs
│       └── footer.ejs
└── middleware/
    ├── logger.js
    └── validation.js
```

7.1 Initialize Project

bash

```
mkdir blog-server
cd blog-server
npm init -y
npm install express ejs
npm install --save-dev nodemon
```

7.2 Main Application File (app.js)

javascript

```
const express = require('express');
const path = require('path');
const loggerMiddleware = require('./middleware/logger');
const { validatePost } = require('./middleware/validation');

const app = express();
const PORT = 3000;

// Middleware setup
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(express.static('public'));
app.use(loggerMiddleware);

// View engine setup
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// In-memory blog posts database
let posts = [
  {
    id: 1,
    title: 'Getting Started with Express',
    content: 'Express.js is a minimal and flexible Node.js web application framework...',
    author: 'John Doe',
    createdAt: new Date('2024-01-15'),
    likes: 42
  },
  {
    id: 2,
    title: 'Understanding Middleware',
    content: 'Middleware functions are functions that have access to the request object...',
    author: 'Jane Smith',
    createdAt: new Date('2024-02-20'),
    likes: 38
  },
  {
    id: 3,
    title: 'Working with Template Engines',
    content: 'Template engines allow you to generate HTML dynamically...',
    author: 'Bob Johnson',
    createdAt: new Date('2024-03-10'),
    likes: 27
  }
];
```

```
// Routes

// Home page - List all posts
app.get('/', (req, res) => {
  res.render('index', {
    title: 'Blog Home',
    posts: posts.sort((a, b) => b.createdAt - a.createdAt)
  });
});

// View single post
app.get('/posts/:id', (req, res) => {
  const postId = parseInt(req.params.id);
  const post = posts.find(p => p.id === postId);

  if (!post) {
    return res.status(404).render('error', {
      title: 'Error',
      message: 'Post not found'
    });
  }

  res.render('post', {
    title: post.title,
    post: post
  });
});

// Create post form
app.get('/create', (req, res) => {
  res.render('create', {
    title: 'Create New Post'
  });
});

// Handle post creation
app.post('/posts', validatePost, (req, res) => {
  const newPost = {
    id: posts.length + 1,
    title: req.body.title,
    content: req.body.content,
    author: req.body.author,
    createdAt: new Date(),
    likes: 0
  };

  posts.push(newPost);
```

```
  res.redirect('/');
});

// Update post
app.put('/api/posts/:id', validatePost, (req, res) => {
  const postId = parseInt(req.params.id);
  const postIndex = posts.findIndex(p => p.id === postId);

  if (postIndex === -1) {
    return res.status(404).json({ error: 'Post not found' });
  }

  posts[postIndex] = {
    ...posts[postIndex],
    title: req.body.title,
    content: req.body.content,
    author: req.body.author
  };

  res.json({
    message: 'Post updated successfully',
    post: posts[postIndex]
  });
});

// Like a post
app.patch('/api/posts/:id/like', (req, res) => {
  const postId = parseInt(req.params.id);
  const post = posts.find(p => p.id === postId);

  if (!post) {
    return res.status(404).json({ error: 'Post not found' });
  }

  post.likes += 1;

  res.json({
    message: 'Post liked',
    likes: post.likes
  });
});

// Delete post
app.delete('/api/posts/:id', (req, res) => {
  const postId = parseInt(req.params.id);
  const postIndex = posts.findIndex(p => p.id === postId);
```

```
  if (postIndex === -1) {
    return res.status(404).json({ error: 'Post not found' });
  }

  const deletedPost = posts.splice(postIndex, 1)[0];

  res.json({
    message: 'Post deleted successfully',
    post: deletedPost
  });
});

// Search posts
app.get('/search', (req, res) => {
  const query = req.query.q?.toLowerCase() || '';

  const results = posts.filter(post =>
    post.title.toLowerCase().includes(query) ||
    post.content.toLowerCase().includes(query) ||
    post.author.toLowerCase().includes(query)
  );

  res.render('index', {
    title: `Search Results for "${query}"`,
    posts: results,
    searchQuery: query
  });
});

// API endpoint - Get all posts as JSON
app.get('/api/posts', (req, res) => {
  res.json({
    count: posts.length,
    posts: posts
  });
});

// 404 handler
app.use((req, res) => {
  res.status(404).render('error', {
    title: '404 - Not Found',
    message: 'The page you are looking for does not exist.'
  });
});

// Error handler
```

```
app.use((err, req, res, next) => {
  console.error('Error:', err.message);

  res.status(err.status || 500).render('error', {
    title: 'Error',
    message: err.message || 'Something went wrong!'
  });
});

// Start server
app.listen(PORT, () => {
  console.log(`Blog server running on http://localhost:${PORT}`);
});
```

7.3 Logger Middleware (middleware/logger.js)

javascript

```javascript
const fs = require('fs');
const path = require('path');

// Logger middleware
const logger = (req, res, next) => {
  const timestamp = new Date().toISOString();
  const method = req.method;
  const url = req.url;
  const ip = req.ip;

  // Console logging
  console.log(`[${timestamp}] ${method} ${url} - IP: ${ip}`);

  // File logging
  const logMessage = `[${timestamp}] ${method} ${url} - IP: ${ip}\n`;
  const logFile = path.join(__dirname, '..', 'access.log');

  fs.appendFile(logFile, logMessage, (err) => {
    if (err) console.error('Error writing to log file:', err);
  });

  // Measure response time
  const startTime = Date.now();

  res.on('finish', () => {
    const duration = Date.now() - startTime;
    console.log(`Response sent with status ${res.statusCode} in ${duration}ms`);
  });

  next();
};

module.exports = logger;
```

7.4 Validation Middleware (middleware/validation.js)

javascript

```
// Validation middleware for post creation/update
const validatePost = (req, res, next) => {
  const { title, content, author } = req.body;
  const errors = [];

  // Validate title
  if (!title || title.trim().length === 0) {
    errors.push('Title is required');
  } else if (title.length < 5) {
    errors.push('Title must be at least 5 characters long');
  } else if (title.length > 100) {
    errors.push('Title must not exceed 100 characters');
  }

  // Validate content
  if (!content || content.trim().length === 0) {
    errors.push('Content is required');
  } else if (content.length < 20) {
    errors.push('Content must be at least 20 characters long');
  }

  // Validate author
  if (!author || author.trim().length === 0) {
    errors.push('Author name is required');
  } else if (author.length < 2) {
    errors.push('Author name must be at least 2 characters long');
  }

  // If errors exist, send error response
  if (errors.length > 0) {
    return res.status(400).json({
      error: 'Validation failed',
      details: errors
    });
  }

  // Sanitize inputs (basic)
  req.body.title = title.trim();
  req.body.content = content.trim();
  req.body.author = author.trim();

  next();
};

module.exports = { validatePost };
```

7.5 Views Templates

views/partials/header.ejs:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title><%= title %></title>
  <link rel="stylesheet" href="/css/style.css">
</head>
<body>
  <nav class="navbar">
    <div class="nav-container">
      <div class="nav-brand">
        <img src="/images/logo.png" alt="Logo" class="logo">
        <span>My Blog</span>
      </div>
      <ul class="nav-links">
        <li><a href="/">Home</a></li>
        <li><a href="/create">New Post</a></li>
        <li><a href="/api/posts">API</a></li>
      </ul>
      <form action="/search" method="GET" class="search-form">
        <input type="text" name="q" placeholder="Search posts..." class="search-input">
        <button type="submit" class="search-btn">Search</button>
      </form>
    </div>
  </nav>
  <div class="container">
```

views/partials/footer.ejs:

```html
```

```
  </div>
  <footer class="footer">
    <p>&copy; 2024 My Blog. Built with Express.js and EJS</p>
    <p>
      <a href="/">Home</a> |
      <a href="/create">Create Post</a> |
      <a href="/api/posts">API</a>
    </p>
  </footer>
</body>
</html>
```

views/index.ejs:

```
html
```

```
<%- include('partials/header', { title: title }) %>

<h1><%= title %></h1>

<% if (typeof searchQuery !== 'undefined' && searchQuery) { %>
  <p class="search-info">
    Showing results for "<strong><%= searchQuery %></strong>"
    (<%= posts.length %> <%= posts.length === 1 ? 'result' : 'results' %>)
  </p>
<% } %>

<div class="posts-grid">
  <% if (posts.length === 0) { %>
    <div class="no-posts">
      <h2>No posts found</h2>
      <p>Be the first to create a post!</p>
      <a href="/create" class="btn btn-primary">Create Post</a>
    </div>
  <% } else { %>
    <% posts.forEach(post => { %>
      <article class="post-card">
        <h2 class="post-title">
          <a href="/posts/<%= post.id %>"><%= post.title %></a>
        </h2>

        <div class="post-meta">
          <span class="author">By <%= post.author %></span>
          <span class="date"><%= post.createdAt.toLocaleDateString() %></span>
          <span class="likes">❤️ <%= post.likes %></span>
        </div>

        <p class="post-excerpt">
          <%= post.content.substring(0, 150) %>...
        </p>

        <div class="post-actions">
          <a href="/posts/<%= post.id %>" class="btn btn-secondary">Read More</a>
          <button class="btn btn-like" onclick="likePost(<%= post.id %>)">
            Like ( ❤️ <%= post.likes %>)
          </button>
        </div>
      </article>
    <% }); %>
  <% } %>
</div>
```

```
<script>
  async function likePost(postId) {
    try {
      const response = await fetch(`/api/posts/${postId}/like`, {
        method: 'PATCH'
      });

      const data = await response.json();

      if (response.ok) {
        alert(`Post liked! Total likes: ${data.likes}`);
        location.reload();
      } else {
        alert('Error liking post');
      }
    } catch (error) {
      console.error('Error:', error);
      alert('Error liking post');
    }
  }
</script>

<%- include('partials/footer') %>
```

views/post.ejs:

```
html
```

```
<%- include('partials/header', { title: title }) %>

<article class="post-detail">
  <h1 class="post-title"><%= post.title %></h1>

  <div class="post-meta">
    <span class="author">By <%= post.author %></span>
    <span class="date"><%= post.createdAt.toLocaleDateString('en-US', {
      year: 'numeric',
      month: 'long',
      day: 'numeric'
    }) %></span>
    <span class="likes">❤️ <%= post.likes %> likes</span>
  </div>

  <div class="post-content">
    <%= post.content %>
  </div>

  <div class="post-actions">
    <button class="btn btn-primary" onclick="likePost(<%= post.id %>)">
      ❤️ Like (<%= post.likes %>)
    </button>
    <button class="btn btn-danger" onclick="deletePost(<%= post.id %>)">
      🗑️ Delete
    </button>
    <a href="/" class="btn btn-secondary">← Back to Home</a>
  </div>
</article>

<script>
  async function likePost(postId) {
    try {
      const response = await fetch(`/api/posts/${postId}/like`, {
        method: 'PATCH'
      });

      const data = await response.json();

      if (response.ok) {
        alert(`Post liked! Total likes: ${data.likes}`);
        location.reload();
      }
    } catch (error) {
      alert('Error liking post');
    }
```

```
  }

  async function deletePost(postId) {
    if (!confirm('Are you sure you want to delete this post?')) {
      return;
    }

    try {
      const response = await fetch(`/api/posts/${postId}`, {
        method: 'DELETE'
      });

      if (response.ok) {
        alert('Post deleted successfully');
        window.location.href = '/';
      }
    } catch (error) {
      alert('Error deleting post');
    }
  }
</script>

<%- include('partials/footer') %>
```

views/create.ejs:

```
html
```

```
<%- include('partials/header', { title: title }) %>

<div class="create-form-container">
  <h1>Create New Post</h1>

  <form action="/posts" method="POST" class="create-form">
    <div class="form-group">
      <label for="title">Post Title *</label>
      <input
        type="text"
        id="title"
        name="title"
        required
        minlength="5"
        maxlength="100"
        placeholder="Enter post title"
        class="form-control">
      <small>Must be between 5-100 characters</small>
    </div>

    <div class="form-group">
      <label for="author">Author Name *</label>
      <input
        type="text"
        id="author"
        name="author"
        required
        minlength="2"
        placeholder="Enter your name"
        class="form-control">
    </div>

    <div class="form-group">
      <label for="content">Content *</label>
      <textarea
        id="content"
        name="content"
        required
        minlength="20"
        rows="10"
        placeholder="Write your post content here..."
        class="form-control"></textarea>
      <small>Must be at least 20 characters</small>
    </div>

    <div class="form-actions">
```

```
    <button type="submit" class="btn btn-primary">Create Post</button>
    <a href="/" class="btn btn-secondary">Cancel</a>
   </div>
  </form>
</div>

<%- include('partials/footer') %>
```

views/error.ejs:

```
html

<%- include('partials/header', { title: title }) %>

<div class="error-container">
  <h1><%= title %></h1>
  <p class="error-message"><%= message %></p>
  <a href="/" class="btn btn-primary">← Go Home</a>
</div>

<%- include('partials/footer') %>
```

7.6 CSS Stylesheet (public/css/style.css)

```
css
```

```css
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  line-height: 1.6;
  color: #333;
  background-color: #f4f4f4;
}

/* Navbar */
.navbar {
  background-color: #2c3e50;
  color: white;
  padding: 1rem 0;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.nav-container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 20px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  flex-wrap: wrap;
}

.nav-brand {
  display: flex;
  align-items: center;
  font-size: 1.5rem;
  font-weight: bold;
}

.logo {
  height: 40px;
  margin-right: 10px;
}

.nav-links {
  display: flex;
  list-style: none;
```

```
  gap: 2rem;
}

.nav-links a {
  color: white;
  text-decoration: none;
  transition: color 0.3s;
}

.nav-links a:hover {
  color: #3498db;
}

.search-form {
  display: flex;
  gap: 0.5rem;
}

.search-input {
  padding: 0.5rem 1rem;
  border: none;
  border-radius: 4px;
  min-width: 200px;
}

.search-btn {
  padding: 0.5rem 1rem;
  background-color: #3498db;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.search-btn:hover {
  background-color: #2980b9;
}

/* Container */
.container {
  max-width: 1200px;
  margin: 2rem auto;
  padding: 0 20px;
  min-height: calc(100vh - 200px);
}
```

```css
h1 {
  color: #2c3e50;
  margin-bottom: 1.5rem;
  font-size: 2.5rem;
}

/* Search Info */
.search-info {
  background-color: #e8f4f8;
  padding: 1rem;
  border-radius: 4px;
  margin-bottom: 1.5rem;
  border-left: 4px solid #3498db;
}

/* Posts Grid */
.posts-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(350px, 1fr));
  gap: 2rem;
  margin-top: 2rem;
}

.post-card {
  background: white;
  border-radius: 8px;
  padding: 1.5rem;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
  transition: transform 0.3s, box-shadow 0.3s;
}

.post-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 4px 12px rgba(0,0,0,0.15);
}

.post-title {
  font-size: 1.5rem;
  margin-bottom: 0.5rem;
}

.post-title a {
  color: #2c3e50;
  text-decoration: none;
  transition: color 0.3s;
}
```

```css
.post-title a:hover {
  color: #3498db;
}

.post-meta {
  display: flex;
  gap: 1rem;
  font-size: 0.9rem;
  color: #7f8c8d;
  margin-bottom: 1rem;
  flex-wrap: wrap;
}

.post-excerpt {
  color: #555;
  margin-bottom: 1rem;
  line-height: 1.8;
}

.post-actions {
  display: flex;
  gap: 0.5rem;
  flex-wrap: wrap;
}

/* Post Detail */
.post-detail {
  background: white;
  border-radius: 8px;
  padding: 2rem;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
  max-width: 800px;
  margin: 0 auto;
}

.post-content {
  font-size: 1.1rem;
  line-height: 1.8;
  color: #444;
  margin: 2rem 0;
  white-space: pre-wrap;
}

/* Buttons */
.btn {
  display: inline-block;
  padding: 0.6rem 1.2rem;
```

```
  border-radius: 4px;
  text-decoration: none;
  cursor: pointer;
  border: none;
  font-size: 1rem;
  transition: all 0.3s;
}

.btn-primary {
  background-color: #3498db;
  color: white;
}

.btn-primary:hover {
  background-color: #2980b9;
}

.btn-secondary {
  background-color: #95a5a6;
  color: white;
}

.btn-secondary:hover {
  background-color: #7f8c8d;
}

.btn-danger {
  background-color: #e74c3c;
  color: white;
}

.btn-danger:hover {
  background-color: #c0392b;
}

.btn-like {
  background-color: #e91e63;
  color: white;
}

.btn-like:hover {
  background-color: #c2185b;
}

/* Forms */
.create-form-container {
  background: white;
```

```
  border-radius: 8px;
  padding: 2rem;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
  max-width: 800px;
  margin: 0 auto;
}

.create-form {
  margin-top: 2rem;
}

.form-group {
  margin-bottom: 1.5rem;
}

.form-group label {
  display: block;
  margin-bottom: 0.5rem;
  font-weight: bold;
  color: #2c3e50;
}

.form-control {
  width: 100%;
  padding: 0.75rem;
  border: 1px solid #ddd;
  border-radius: 4px;
  font-size: 1rem;
  font-family: inherit;
}

.form-control:focus {
  outline: none;
  border-color: #3498db;
  box-shadow: 0 0 0 3px rgba(52, 152, 219, 0.1);
}

textarea.form-control {
  resize: vertical;
  min-height: 150px;
}

.form-group small {
  display: block;
  margin-top: 0.3rem;
  color: #7f8c8d;
  font-size: 0.875rem;
```

```
}

.form-actions {
  display: flex;
  gap: 1rem;
  margin-top: 2rem;
}

/* Error Page */
.error-container {
  background: white;
  border-radius: 8px;
  padding: 3rem;
  text-align: center;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
  max-width: 600px;
  margin: 0 auto;
}

.error-message {
  font-size: 1.2rem;
  color: #e74c3c;
  margin: 2rem 0;
}

/* No Posts */
.no-posts {
  text-align: center;
  padding: 3rem;
  background: white;
  border-radius: 8px;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
}

.no-posts h2 {
  color: #7f8c8d;
  margin-bottom: 1rem;
}

.no-posts p {
  color: #95a5a6;
  margin-bottom: 1.5rem;
}

/* Footer */
.footer {
  background-color: #2c3e50;
```

```
  color: white;
  text-align: center;
  padding: 2rem;
  margin-top: 3rem;
}

.footer p {
  margin: 0.5rem 0;
}

.footer a {
  color: #3498db;
  text-decoration: none;
}

.footer a:hover {
  color: #5dade2;
}

/* Responsive */
@media (max-width: 768px) {
  .nav-container {
    flex-direction: column;
    gap: 1rem;
  }

  .posts-grid {
    grid-template-columns: 1fr;
  }

  .search-form {
    width: 100%;
  }

  .search-input {
    flex: 1;
  }
}
```

7.7 Package.json

```
json
```

```
{
  "name": "blog-server",
  "version": "1.0.0",
  "description": "A complete blog server with Express.js",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js"
  },
  "keywords": ["express", "blog", "ejs", "middleware"],
  "author": "Your Name",
  "license": "MIT",
  "dependencies": {
    "express": "^4.18.2",
    "ejs": "^3.1.9"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

7.8 Running the Blog Server

```bash
# Development mode (auto-reload)
npm run dev

# Production mode
npm start
```

Visit: `http://localhost:3000`

---

8. Review and Best Practices

8.1 Express.js Core Concepts Summary

1. Application Structure

```javascript
```

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

// Middleware
app.use(express.json());
app.use(express.static('public'));

// Routes
app.get('/', handler);
app.post('/api/data', handler);

// Start server
app.listen(3000);
```

2. Request-Response Cycle

```
Client → Middleware 1 → Middleware 2 → Route Handler → Response → Client
```

3. Common Methods

- `app.get()` - Retrieve data
- `app.post()` - Create data
- `app.put()` - Update entire resource
- `app.patch()` - Update partial resource
- `app.delete()` - Delete resource

8.2 Best Practices

1. Project Structure

```
project/
├── app.js              # Main application file
├── routes/             # Route handlers
├── middleware/         # Custom middleware
├── controllers/        # Business logic
├── models/             # Data models
├── views/              # Templates
├── public/             # Static files
├── config/             # Configuration
└── package.json
```

2. Error Handling

javascript

```javascript
// Always use try-catch for async operations
app.get('/data', async (req, res, next) => {
  try {
    const data = await fetchData();
    res.json(data);
  } catch (error) {
    next(error); // Pass to error handler
  }
});

// Global error handler
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: err.message });
});
```

3. Security

javascript

```javascript
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');

// Security headers
app.use(helmet());

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // Limit each IP to 100 requests per window
});

app.use('/api/', limiter);
```

4. Environment Variables

javascript

```javascript
require('dotenv').config();

const PORT = process.env.PORT || 3000;
const DB_URL = process.env.DATABASE_URL;

app.listen(PORT);
```

5. Validation

```javascript
// Always validate user input
const validateEmail = (email) => {
  const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
};

app.post('/register', (req, res) => {
  if (!validateEmail(req.body.email)) {
    return res.status(400).json({ error: 'Invalid email' });
  }
  // Process registration
});
```

6. Response Standards

```javascript
// Consistent API responses
const sendSuccess = (res, data, message = 'Success') => {
  res.json({
    success: true,
    message,
    data
  });
};

const sendError = (res, message, status = 500) => {
  res.status(status).json({
```