Sajib-Bhattacharjee / ejs-documentation-2025

Code Issues Pull requests Actions Projects Security Insights

📘 Complete EJS Documentation from Beginner to Advanced! Learn EJS syntax, loops, conditionals, partials, layouts, forms, and more with real examples. 🧩 Built by Sajib Bhattacharjee – MERN Stack Specialist 🚀 Perfect for Node.js + Express developers. #EJS #WebDev 🛠️

MIT license

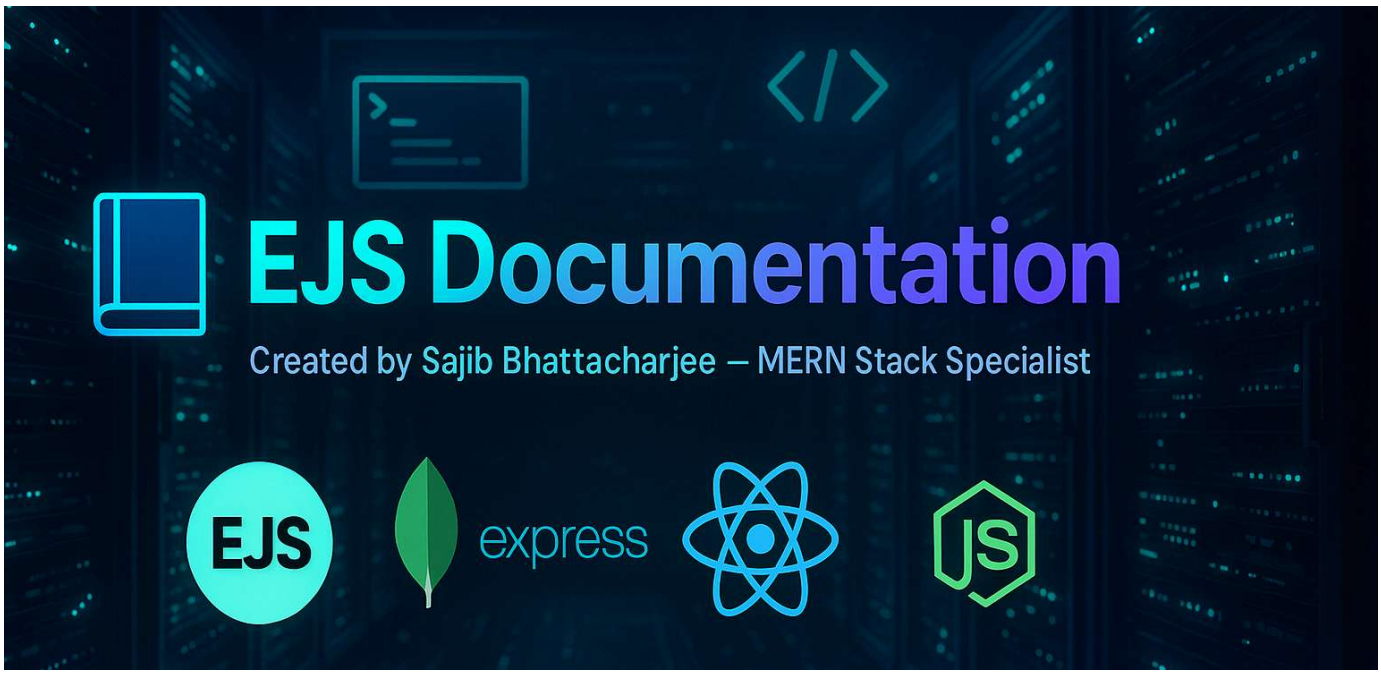
0 stars 0 forks 0 watching Branches Activity
Tags

Public repository

main 1 Branch 0 Tags Go to file t Go to file Add file Code

Sajib-Bhattacharjee Everything Uploaded Successfully. 8f316af · 2 months ago

images	Everything Uploaded Successfully.	2 months ago
LICENSE	Initial commit	2 months ago
README.md	Everything Uploaded Successfully.	2 months ago
infographic.html	Everything Uploaded Successfully.	2 months ago



☀️ **Infographic Preview → EJS** 💞

🎉 Click to explore the fun and laughter! 😄

📘 EJS Documentation – Complete Guide

🎯 Table of Contents

🔰 Beginner Level

🛠️ Intermediate Level

🚀 Advanced Level

README MIT license

🔰 Beginner Level

🎯 Introduction to EJS

📝 What is EJS?

EJS (Embedded JavaScript) is a simple templating language that lets you generate HTML markup with plain JavaScript.
Think of it as a way to "fill in the blanks" in your HTML using data and logic from your server.

> **Analogy:**
> Imagine you're sending out party invitations. You have a template, but you want to personalize each one with the guest's name.
> EJS lets you do this for web pages—combining a template with data to create custom HTML for each user.

📝 Why use EJS?

- ✅ **Familiar Syntax:** If you know HTML and JavaScript, you already know EJS.
- 🚀 **Rapid Development:** Mix server-side data and logic directly into your HTML.
- 🔍 **Easy Debugging:** EJS provides clear JavaScript errors with line numbers.
- ⚡ **High Performance:** EJS caches templates in production for fast rendering.
- 🔄 **Flexible:** Use any JavaScript logic—no need to learn a new language.

📝 EJS vs Other Templating Engines

Feature	EJS (Embedded JavaScript)	Pug (formerly Jade)	Handlebars
Syntax	Plain HTML & JavaScript. What you see is what you get.	Indentation-based, abstract. No HTML tags.	Mustache-like `{{ }}` syntax.
Logic	Full power of JavaScript. Anything you can do in JS, you can do here.	Limited logic, focuses on structure.	Limited logic, requires "helpers" for anything complex.
Best For	Developers who love HTML and want maximum flexibility.	Developers who prefer minimalism and concise syntax.	Projects needing logic-less templates and cross-language support.

🎯 Getting Started

📝 Installation & Setup

1. **Install Node.js** (if you haven't already):
   Download Node.js

2. **Initialize your project:**

```
npm init -y
```

3. **Install Express and EJS:**

```
npm install express ejs
```

📝 Integrating EJS with Node.js & Express

```
// server.js
import express from "express";
import path from "path";
import { fileURLToPath } from "url";

const app = express();
const PORT = process.env.PORT || 3000;

// Modern way to get __dirname in ES modules
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

// Set the view engine to EJS
app.set("view engine", "ejs");

// Set the views directory
app.set("views", path.join(__dirname, "views"));

// Middleware to parse POST data (important for forms)
app.use(express.urlencoded({ extended: true }));

// Render a basic template
app.get("/", (req, res) => {
  res.render("index", { title: "Welcome to EJS!" });
});

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

> 2025 Tip:
> Use ES modules (`import / export`) for all new Node.js projects.
> Always use `express.urlencoded` middleware for form handling.

📝 Rendering a Basic Template

Project Structure:

```
/my-project
|-- /views
|   |-- index.ejs   <-- Your template files go here
|-- server.js       <-- Your Express server logic
|-- package.json
```

views/index.ejs:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title><%= title %></title>
</head>
<body>
  <h1><%= title %></h1>
  <p>Welcome to your first EJS page! 🎉</p>
</body>
</html>
```

🎯 Basic Syntax

📝 Outputting Data: <%= %>

- Use `<%= variable %>` to output and escape data (prevents XSS).
- Example:

```
<h1>Hello, <%= user.name %>!</h1>
```

📝 Executing JS: <% %>

- Use `<% %>` for JavaScript logic (no output).
- Example:

```
<% if (user.isAdmin) { %>
  <p>Welcome, admin!</p>
<% } %>
```

📝 Including HTML Comments

- Use standard HTML comments: `<!-- This is a comment -->`
- For EJS-only comments (not rendered in HTML), use `<%# This is a comment %>`

📝 Escaping HTML: <%- %>

- Use `<%- %>` to output unescaped (raw) HTML.
  ⚠️ **Only use with trusted content!**
- Example:

```
<div><%- trustedHtml %></div>
```

🎯 Variables and Expressions

📝 Passing Data to Templates

- Data is passed from your Express route handler to your EJS template using `res.render`.
- Example:

```
// server.js
app.get("/profile", (req, res) => {
  res.render("profile", { user: { name: "Alex", age: 25 } });
});
```

```
<!-- views/profile.ejs -->
<h2>Welcome, <%= user.name %>!</h2>
<p>Age: <%= user.age %></p>
```

- You can pass any JavaScript object, array, or value.

📝 Using JavaScript in Templates

- EJS lets you use any valid JavaScript inside `<% %>` tags.
- You can declare variables, use expressions, and perform calculations.
- Example:

```
<% const year = new Date().getFullYear(); %>
<footer>© <%= year %> My Website</footer>
```

- You can also use expressions directly in output tags:

```
<p>Next year: <%= year + 1 %></p>
```

🎯 Conditional Rendering

📝 if / else statements

- Use standard JavaScript `if`, `else if`, and `else` inside EJS:

```
<% if (user.isAdmin) { %>
  <span>👑 Admin Panel</span>
<% } else { %>
  <span>🙋 User Dashboard</span>
<% } %>
```

📝 Ternary Operators

- For inline conditions, use the ternary operator:

```
<p>Status: <%= user.active ? "🟢 Active" : "🔴 Inactive" %></p>
```

🎯 Loops

📝 for Loops

- Use classic JavaScript `for` loops for full control:

```
<ul>
  <% for (let i = 0; i < items.length; i++) { %>
    <li><%= items[i] %></li>
```

```
  <% } %>
</ul>
```

📝 forEach in Arrays

- The most common way to loop through arrays:

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

📝 Looping with Index and Keys

- Access both value and index:

```
<ul>
  <% items.forEach((item, idx) => { %>
    <li>#<%= idx + 1 %>: <%= item %></li>
  <% }); %>
</ul>
```

- For objects:

```
<ul>
  <% Object.entries(user).forEach(([key, value]) => { %>
    <li><b><%= key %></b>: <%= value %></li>
  <% }); %>
</ul>
```

🛠️ Intermediate Level

🎯 Partials

📝 Creating and Using Partials

- **Partials** are reusable EJS files (like navbars, footers, cards) included in other templates.
- Convention: Store partials in `/views/partials` and prefix filenames with `_` (e.g., `_header.ejs`).
- **Including a partial:**

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

- **Passing data to a partial:**

```
<%- include('partials/_header', { user: loggedInUser }) %>
```

📝 Use Cases (Navbar, Footer, etc.)

- **Navbar:**
  `/views/partials/_navbar.ejs`

```
<nav>
  <a href="/">Home</a>
  <a href="/profile">Profile</a>
  <% if (user?.isAdmin) { %>
```

```
    <a href="/admin">Admin</a>
  <% } %>
</nav>
```

- **Footer:**
  `/views/partials/_footer.ejs`

```
<footer>
  &copy; <%= new Date().getFullYear() %> MyApp
</footer>
```

- **Usage in a page:**

```
<body>
  <%- include('partials/_navbar', { user }) %>
  <main> ... </main>
  <%- include('partials/_footer') %>
</body>
```

🎯 Layout and Reusability

📝 Template Inheritance with Includes

- EJS doesn't have true inheritance, but you can simulate it with includes.
- **Example:**
  - `/views/layouts/main.ejs`

```
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
</head>
<body>
  <%- include('../partials/_navbar', { user }) %>
  <main>
    <%- body %>
  </main>
  <%- include('../partials/_footer') %>
</body>
</html>
```

  - In your route:

```
// server.js
app.get("/about", (req, res) => {
  res.render("pages/about", {
    layout: "layouts/main",
    title: "About",
    user,
    body: "<h2>About us</h2>",
  });
});
```

  - Or use a library like express-ejs-layouts for more advanced layout support.

📝 Modularizing UI Components

- Break complex UIs into small, reusable partials (cards, alerts, forms).
- Example:
  - `/views/partials/_productCard.ejs`

```
<div class="card">
  <h3><%= product.name %></h3>
```

```
    <p>Price: <%= formatCurrency(product.price) %></p>
  </div>
```

- Usage:

```
<% products.forEach(product => { %>
  <%- include('partials/_productCard', { product }) %>
<% }); %>
```

🎯 Forms & EJS

📝 Rendering Form Fields Dynamically

- Use EJS to generate form fields based on data or configuration.

```
<% fields.forEach(field => { %>
  <label><%= field.label %></label>
  <input type="<%= field.type %>" name="<%= field.name %>" value="<%= formData[f
<% }); %>
```

📝 Handling Form Input Display

- Pre-fill form fields with submitted data to improve UX:

```
<input type="text" name="username" value="<%= formData?.username || '' %>">
```

📝 Validation Error Rendering

- Show errors above the form:

```
<% if (errors && errors.length) { %>
  <ul class="error-list">
    <% errors.forEach(error => { %>
      <li>❌ <%= error %></li>
    <% }); %>
  </ul>
<% } %>
```

🎯 Advanced Conditional Logic

📝 Nested Conditions

- Use nested `if` statements for complex logic:

```
<% if (user) { %>
  <% if (user.isAdmin) { %>
    <p>Welcome, admin!</p>
  <% } else { %>
    <p>Welcome, user!</p>
  <% } %>
<% } else { %>
  <p>Please log in.</p>
<% } %>
```

📝 Combining Logic and HTML Output

- Mix logic and output for dynamic UIs:

```
<% if (cart.length) { %>
  <ul>
    <% cart.forEach(item => { %>
      <li><%= item.name %> - <%= item.qty %></li>
```

```
    <% }); %>
  </ul>
<% } else { %>
  <p>Your cart is empty.</p>
<% } %>
```

🎯 Arrays and Objects

📄 Iterating Complex Data Structures

- Loop through arrays of objects:

```
<% users.forEach(user => { %>
  <div>
    <h4><%= user.name %></h4>
    <p>Email: <%= user.email %></p>
  </div>
<% }); %>
```

📄 Accessing Nested Data

- Access properties deeply:

```
<p>City: <%= user.address?.city || "Unknown" %></p>
```

🎯 Custom Helper Functions

📄 Creating Utility Functions for Views

- Define helpers in a separate file (e.g., `/utils/helpers.js`):

```
export function formatCurrency(amount) {
  return new Intl.NumberFormat("en-US", {
    style: "currency",
    currency: "USD",
  }).format(amount);
}
```

📄 Formatting Dates, Currency, etc.

- Register helpers globally:

```
// server.js
import * as helpers from "./utils/helpers.js";
Object.assign(app.locals, helpers);
```

- Use in EJS:

```
<p>Price: <%= formatCurrency(product.price) %></p>
```

🚀 Advanced Level

🎯 Advanced Template Structuring

📄 Folder Structure Best Practices

- Organize your `views` directory for scalability:

```
/views
├── /pages         # Top-level page templates
│   ├── home.ejs
│   ├── about.ejs
│   └── /products
│       ├── index.ejs
│       └── show.ejs
├── /partials      # Reusable components
│   ├── _header.ejs
│   ├── _footer.ejs
│   └── _productCard.ejs
├── /layouts       # Base layout skeletons (optional)
│   └── main.ejs
└── 404.ejs        # Error page
```

📝 Organizing Partials and Views

- Use clear, consistent naming.
- Group related partials (e.g., `/partials/forms`, `/partials/cards`).
- Keep layouts in a dedicated folder for easy reuse.

🎯 Performance Optimization

📝 Caching Templates

- EJS automatically caches templates in production mode.
- For custom caching, use the `cache` option:

```
res.render("index", { cache: true });
```

📝 Minimizing Re-renders

- Only re-render templates when data changes.
- Use partials for static content to avoid unnecessary processing.

🎯 Security

📝 Preventing XSS with EJS

- Always use `<%= %>` for user-generated content (it escapes HTML).
- Never use `<%- %>` with untrusted data.

📝 Using escapeHTML Properly

- EJS escapes HTML by default with `<%= %>`.
- If you need custom escaping, use a helper:

```
export function escapeHTML(str) {
  return String(str)
    .replace(/&/g, "&amp;")
    .replace(/</g, "&lt;")
    .replace(/>/g, "&gt;")
    .replace(/"/g, "&quot;")
    .replace(/'/g, "&#39;");
}
```

```
<%= escapeHTML(user.comment) %>
```

🎯 Integrating with Front-End Frameworks

📝 EJS with Bootstrap or Tailwind

- Use EJS to output classes and structure for any CSS framework:

```
<div class="card <%= product.featured ? 'border-primary' : '' %>">
  <h3 class="text-lg font-bold"><%= product.name %></h3>
</div>
```

📝 Mixing Client-Side JavaScript

- You can include `<script>` tags in your EJS templates for client-side logic.
- Pass server data to the client:

```
<script>
  const user = <%- JSON.stringify(user) %>;
</script>
```

🎯 Debugging EJS Templates

📝 Common Errors and Fixes

- **Undefined variable:**
  Check spelling and ensure you pass all required data to `res.render`.
- **Syntax error:**
  EJS will show the line number and error in the console.

📝 Logging Data to Console/Browser

- Log data in the template for debugging:

```
<% console.log(user); %>
```

- Or output as JSON in the browser:

```
<pre><%= JSON.stringify(user, null, 2) %></pre>
```

🎯 Localization (i18n) with EJS

📝 Rendering Multi-Language Support

- Pass a translation function to your templates:

```
// i18n.js
export const translations = {
  en: { welcome: "Welcome" },
  es: { welcome: "Bienvenido" },
};
export function getTranslator(lang = "en") {
  return (key) => translations[lang][key] || key;
}
```

```
// server.js
import { getTranslator } from "./i18n.js";
app.use((req, res, next) => {
  const lang = req.query.lang || "en";
  res.locals.t = getTranslator(lang);
```

```
  next();
});
```

📝 Locale-Based Content Display

- Use the translation function in EJS:

```
<h1><%= t('welcome') %>, <%= user.name %>!</h1>
```

🎯 Deploying EJS Apps

📝 Preparing Templates for Production

- Minify HTML and assets.
- Set `NODE_ENV=production` for best performance.

📝 Hosting on Render/Vercel/Heroku

- All major Node.js hosts support EJS/Express.
- For Vercel/Render, use a `vercel.json` or `render.yaml` for configuration.
- Always use environment variables for secrets and config.

🎯 Real Project Example

📝 Full CRUD App Using EJS

- Build a simple product manager:
  - List, create, edit, and delete products.
  - Use EJS for all views.
  - Store data in a database (e.g., MongoDB, PostgreSQL).

📝 Folder Structure + Routing + Views

- Example structure:

```
/views
  /products
    index.ejs
    show.ejs
    edit.ejs
    new.ejs
  /partials
    _header.ejs
    _footer.ejs
/routes
  products.js
server.js
```

🎯 EJS Best Practices

📝 Clean Code Principles

- Keep logic out of views—do calculations in controllers.
- Use partials for repeated UI.

📝 Code Reusability

- Modularize helpers and partials.
- Use layouts for consistent structure.

📝 Separation of Concerns

- Views: display only.
- Controllers: handle logic and data.
- Models: manage data storage.

Happy coding! 🚀

👨‍💻 About the Author

Sajib Bhattacharjee MERN Stack Specialist | Full-Stack Web Developer

- 🌐 Portfolio & Projects
- 💼 LinkedIn
- 📧 Contact Me

Releases

No releases published

Packages

No packages published

Languages

- **HTML** 100.0%