LECTURE 24

Response Generation Using Templates

Building real HTML on the server by filling reusable views with data — EJS, loops, partials, and the XSS trap to avoid.

CSC336 Web Technologies

OVERVIEW

In This Lecture

1. SSR vs. client-side rendering — a quick recap
2. What a template engine is, and EJS in action
3. Passing data from an Express route into a view, and interpolating it
4. Loops, conditionals, partials/includes, and layouts inside templates
5. Output escaping (XSS prevention), form redisplay, and flash messages

RECAP

SSR vs. Client-Side Rendering

1. Server-side rendering (SSR): the server builds the complete HTML page — including the data — and sends it ready to display
2. Client-side rendering (CSR): the server sends a near-empty page + JavaScript; the browser builds the visible content after load
3. CSR is how single-page apps built with React work — covered later in this course
4. This lecture is entirely about the SSR side: Express + a template engine

SSR vs. CSR, Side by Side

What Is a Template Engine?

1. A library for writing HTML files with placeholders and logic (loops, conditionals) embedded inside them
2. At request time, it combines your template file with real data and produces plain HTML — a template file is called a view
3. EJS (Embedded JavaScript) — real JavaScript inside <% %> tags, right alongside HTML — our primary example
4. Pug (indentation-based), Handlebars ("logic-light"), Django templates — same idea, different ecosystems

Installing and Configuring EJS

```
npm install ejs

const express = require("express");
const app = express();

app.set("view engine", "ejs"); // tells Express to use EJS
app.set("views", "./views");   // folder where .ejs files live
```

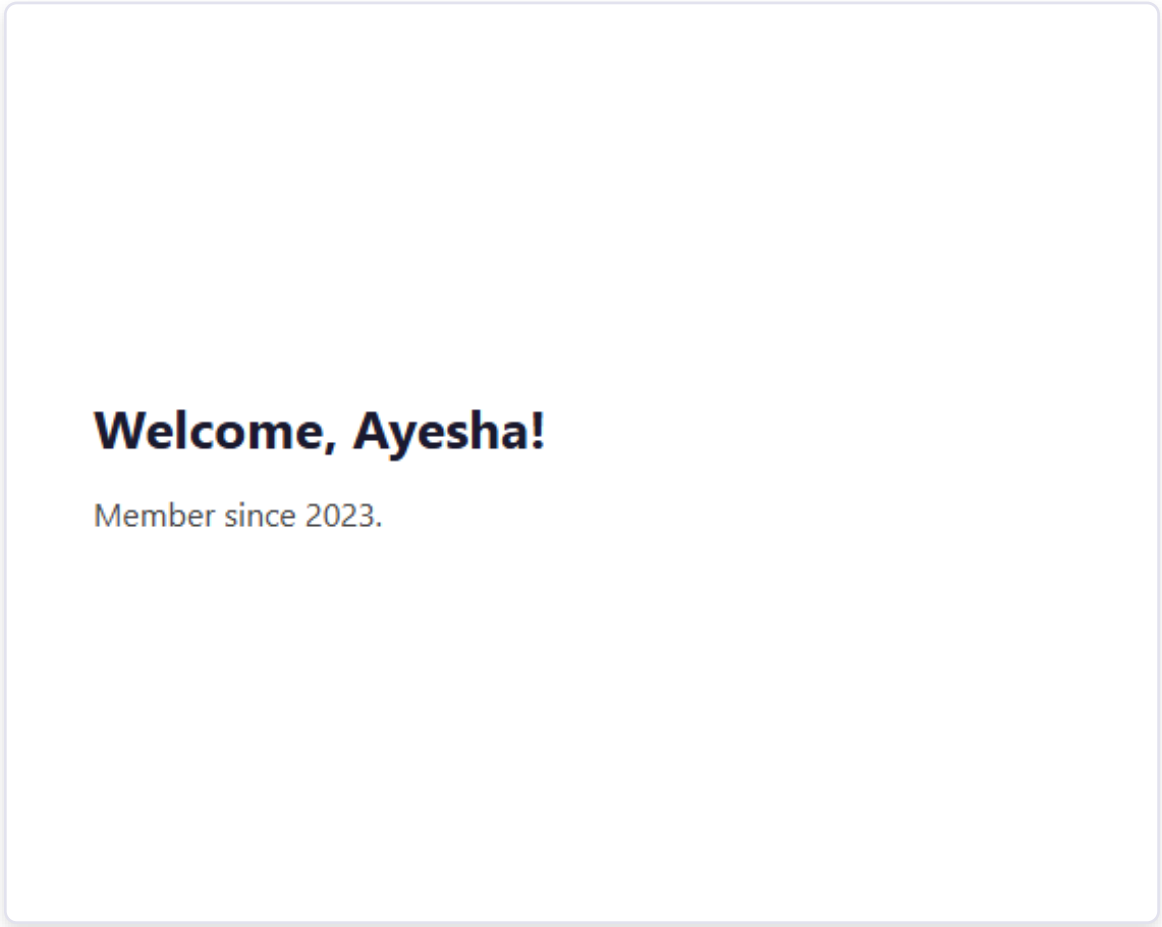
Express now looks for files ending in .ejs inside the views folder whenever you call res.render().

res.render(view, data)

```
app.get("/profile", (req, res) => {
  res.render("profile", {
    username: "Ayesha",
    joinYear: 2023,
  });
});
```

profile.ejs, Rendered

```
<!-- views/profile.ejs -->
<h1>Welcome,
  <%= username %>!
</h1>
<p>
  Member since
  <%= joinYear %>.
</p>
```



<%= %> Escapes by Default

NOTE

<%= expression %> is interpolation — it evaluates the JavaScript expression and inserts the result into the HTML, automatically escaping it (why that matters comes up shortly).

Data From the Route

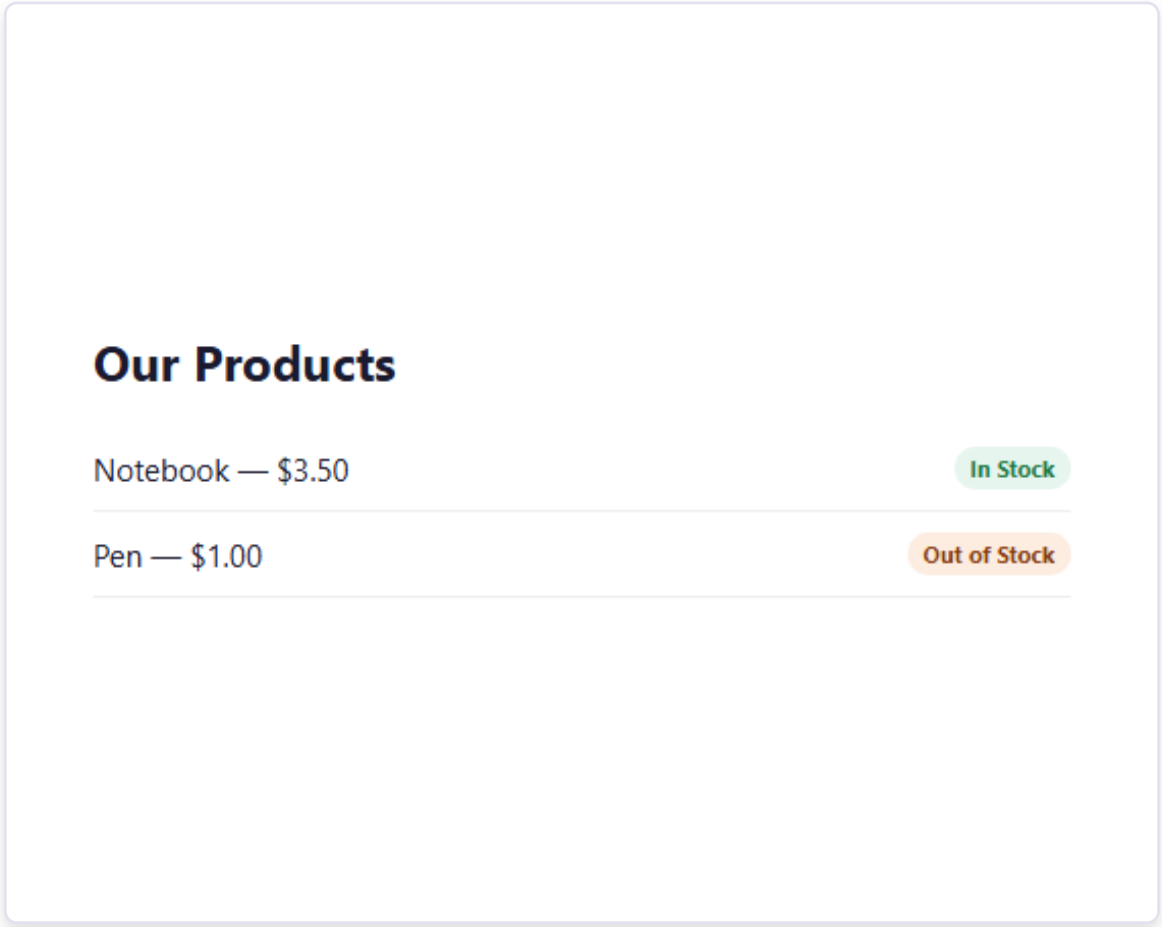
```
app.get("/products", (req, res) => {
  res.render("products", {
    products: [
      { name: "Notebook", price: 3.5, inStock: true },
      { name: "Pen", price: 1.0, inStock: false },
    ],
  });
});
```

<% %> Runs Raw JavaScript

```
<% if (products.length === 0) { %>
  <p>No products available.</p>
<% } else { %>
  <ul>
    <% products.forEach(function(product) { %>
      <li>
        <%= product.name %> — $<%= product.price.toFixed(2) %>
        <% if (product.inStock) { %>
          <span class="in-stock">In Stock</span>
        <% } else { %>
          <span class="out-of-stock">Out of Stock</span>
        <% } %>
      </li>
    <% }) %>
  </ul>
<% } %>
```

products.ejs, Rendered

```
<% products.forEach(p => { %>
  <li>
    <%= p.name %> —
    $<%= p.price
      .toFixed(2) %>
    <% if (p.inStock) { %>
      In Stock
    <% } else { %>
      Out of Stock
    <% } %>
  </li>
<% }) %>
```



Partials / Includes

1. A small, reusable template fragment inserted into other templates — a header, nav bar, or footer

2. Avoids repeating the same markup in every single template file as pages grow

Defining and Including a Partial

```
<!-- views/partials/header.ejs -->
<header>
  <h1>My Website</h1>
  <nav><a href="/">Home</a> | <a href="/products">Products</a></nav>
</header>

<!-- views/products.ejs -->
<%- include('partials/header') %>
<h1>Our Products</h1>
<%- include('partials/footer') %>
```

Why <%- %> Here, Not <%= %>

NOTE

include() returns raw HTML, so we use <%- %> (the unescaped output tag) instead of <%= %> — otherwise the header's own HTML tags would show up as literal text like <header> instead of being rendered.

Layouts: A Shared Page Shell

1. A layout takes partials a step further: one "shell" template (<html>, <head>, header, footer already in place) wrapping each page's unique content

2. EJS has no built-in layout support like some engines — achieve the same effect with includes, or the express-ejs-layouts package

express-ejs-layouts

```
const expressLayouts = require("express-ejs-layouts");
app.use(expressLayouts);
app.set("layout", "layout"); // uses views/layout.ejs

<!-- views/layout.ejs -->
<body>
  <%- include('partials/header') %>
  <%- body %>  <!-- each page's unique content injected here -->
  <%- include('partials/footer') %>
</body>
```

Cross-Site Scripting (XSS)

1. An attack where an attacker gets their own JavaScript to run inside your page
2. Usually via submitted text (a comment, a username) inserted directly into HTML without being cleaned up
3. If a comment contains <script>stealCookies()</script> and your template inserts it unescaped, that script runs in every visitor's browser

Escaped vs. Unescaped Output

```
<!-- comment.text is: <script>alert('hacked')</script> -->

<p><%= comment.text %></p>
<!-- Renders SAFELY as visible text -->

<p><%- comment.text %></p>
<!-- DANGEROUS: actually executes the script -->
```

<%= %> converts <, >, &, " into safe HTML entities (<, >, ...) so the browser displays them as text instead of running them.

Never Use <%- %> on User Data

WARNING

Only use the unescaped <%- %> tag for content you trust completely — your own partials, or HTML you've deliberately sanitized. Default to escaped output (<%= %>) and treat unescaped output as an exception that needs a specific reason.

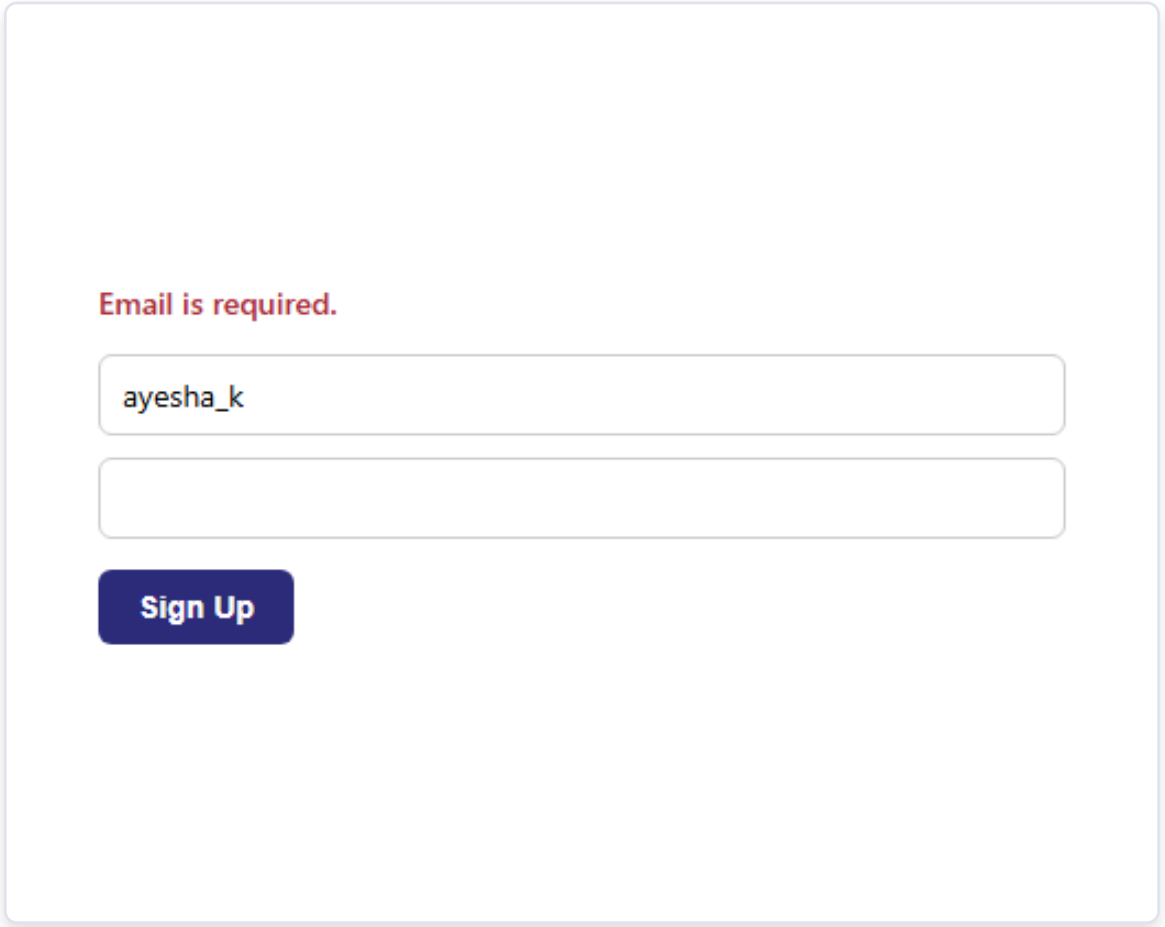
Redisplaying an Invalid Submission

```
app.post("/signup", (req, res) => {
  const { username, email } = req.body;

  if (!email) {
    return res.render("signup", {
      error: "Email is required.",
      username, // send back what they already typed
      email,
    });
  }
  // ... otherwise, save the user and redirect ...
});
```

Better to show the form again with what the user already typed than to make them start over from a blank form.

signup.ejs, Rendered With an Error

```
<% if (typeof error
     !== 'undefined') { %>
  <p class="error">
    <%= error %>
  </p>
<% } %>

<input name="username"
  value="<%= username %>">
<input name="email"
  value="<%= email %>">
```



One-Time Messages After a Redirect

1. A flash message is a short message ("Login successful", "Item deleted") shown immediately after an action
2. Typically right after a redirect, then automatically discarded so it doesn't reappear on refresh
3. Relies on sessions to temporarily hold the message across the redirect

Setting Up connect-flash

```
npm install connect-flash express-session

app.use(session({ secret: "some-secret-key", resave: false, saveUninitialized: false }));
app.use(flash());

// Make flash messages available to every template automatically
app.use((req, res, next) => {
  res.locals.successMessage = req.flash("success");
  res.locals.errorMessage = req.flash("error");
  next();
});
```

Setting and Reading the Message

```
app.post("/items/:id/delete", (req, res) => {
  // ... delete the item ...
  req.flash("success", "Item deleted successfully.");
  res.redirect("/items");
});

<!-- views/items.ejs -->
<% if (successMessage.length > 0) { %>
  <p class="flash-success"><%= successMessage[0] %></p>
<% } %>
```

The message is stored when req.flash() is called, read (and removed) on the very next render, then gone — refreshing /items again shows nothing.

KEY TAKEAWAYS

Lecture 24 in Six Points

1. SSR builds complete HTML on the server before sending it; client-side rendering sends a near-empty page and builds content with JavaScript in the browser.
2. A template engine (EJS, Pug, Handlebars, Django templates) fills HTML views with real data at request time via res.render().
3. EJS uses <%= %> for escaped interpolation, <% %> for control flow, and <%- %> for unescaped raw HTML output.
4. Partials/includes avoid repeating shared markup; layouts wrap a shared page shell around each view's unique content.
5. Escaping output by default is your main defense against XSS — never render untrusted, user-submitted data with the unescaped tag.
6. Form redisplay shows a submitted form again with the user's input and an error; flash messages show a one-time message after a redirect, then discard it.