CS346 – Project Deliverable 2 (Week 8): Interactive Pages & Basic Server Logic

---

Overview

This second deliverable expands your static EJS site into an interactive, data-driven web app.
You'll use **JavaScript** for dynamic client-side behavior and **Node + Express** for basic server routes and controllers.
By the end of this iteration, your project should feel *alive* — responding to user actions and rendering information dynamically instead of being purely static

Deliverable 1 Clarity

Project Deliverable 1

.

Goals

- Connect the **front end (EJS + JS)** with the **back end (Express controllers)**.
- Add at least one interactive feature that changes the DOM or displays dynamic data.
- Practice passing variables from routes → controllers → views.
- Reinforce Git workflow and collaborative development.
- Demonstrate incremental progress toward a functioning full-stack monolith.

Requirements

1. Server Logic and Routing

- Add at least **two new routes** handled by Express (`src/routes/`).
- Each route must be connected to a **controller** (`src/controllers/`) that:
  - Processes a request, performs logic (e.g., read sample data or mock a form submission).
  - Renders an EJS view with data passed via `res.render()`.
- Use the **MVC structure** from Deliverable 1 — controllers contain logic; views contain markup.

2. Client-Side JavaScript (Interactivity)

- Include at least one JS file in `/public/js/` and link it to your EJS pages.
- Demonstrate basic interactivity, such as:
  - Responding to a button click or form submission.
  - Dynamically updating DOM content.
  - Using `fetch()` to call your Express route and display a response.
- Client JS must use event listeners (`addEventListener`) and avoid inline `onclick`.

3. Dynamic Views with EJS

- Update existing EJS pages to display server-side data (e.g., titles, arrays, mock objects).
- Use EJS syntax properly:
  - `<%= variable %>` for escaped output.
  - `<% for (...) { } %>` for loops or conditional blocks.
- Confirm that routes and views work together end-to-end.

4. Code Organization & Project Structure

Maintain the template layout from Deliverable 1:

```
src/
 ├── routes/
 ├── controllers/
 ├── views/
 └── public/
      ├── css/
      ├── js/
      └── img/
```

- No server logic inside EJS.
- No inline JavaScript in HTML.
- All assets served statically via Express (`app.use(express.static('public'))`).

5. Git Workflow and Collaboration

- Branch name: `feature/week8-interactivity`.
- Use meaningful commit messages that describe functional changes.
- Open a Pull Request (PR) to `main` by Friday (so it can be reviewed by peers).
- Request a peer review from the assigned team and leave constructive feedback.
- Merge after you have reviewed the feedback and are confident in your final product.
- Update `README.md` with a short summary of new interactive features.

Deliverables (by end of week)

You should have:

- A functioning Express app with server routes rendering dynamic EJS pages.
- At least one client-side JavaScript file that adds interactivity to the UI.
- Proper MVC separation and organized codebase.
- A merged PR showing this work in `main`.
- An updated README documenting new behavior and verification steps.

Grading (30 points × partner multiplier)

Category	**Points**	**Focus**
Implementation of Topics	15	Working routes, controllers, and interactive client JS
Process & Professionalism	15	Git workflow, PR quality, peer review, documentation
Partner Collaboration Multiplier	—	×1.0 to ×0.6 based on participation

---

Summary (Simplified)

Goal: Integrate JavaScript and Express to make your pages interactive and dynamic.

By end of Deliverable 2 you should be able to:

- Pass data from Express to EJS and render it on the page.
- Add basic front-end interactivity with client-side JS.
- Demonstrate clean code organization and collaboration through Git and peer reviews.