

JavaScript Full-Stack Web Development — Course Guide

Course Contract · L3 LPECMN — Fall 2026

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-14

Abstract The course contract for the 18-hour JavaScript full-stack web development course: prerequisites, the one-app spine, the review-layer capstone, the 6-session schedule, the 10/60/30 assessment, and the 1-month home stretch. Consult this reference document throughout the course.

Table of contents

1	Prerequisites	1
2	The Contract	2
3	The Capstone = the review layer	3
3.1	Objective	3
3.2	The canonical review schema	3
3.3	Per-session milestones	4
3.4	Frontend → backend flow (in-book)	4
4	Schedule (6 sessions)	6
5	Assessment	6
5.1	Attendance & Participation (10%) ...	7
5.2	Final Quiz (60%)	7
5.3	Group Project: the review layer, extended (30%)	7
5.3.a	Optional Group Extension	8
6	Home stretch (1 month)	8
7	FAQ	9

1 Prerequisites

This course assumes you can already program; it is **not** a first-language course.

- **Assumed — programming background:** logical thinking, variables, control flow, functions, and basic data structures in **any** language (PHP, Python, Java, ...).
- **Assumed — a prior PHP lecture:** several HTML/CSS and server-side topics here are **refreshers** for developers who have already seen server-side web development (request/response, routing, database queries). We build on that exposure rather than re-teaching it from zero.
- **Not assumed — JavaScript:** no prior JavaScript experience is required; we start from JavaScript fundamentals.
- **New in this course:** Flexbox, CSS Grid, responsive design, and Tailwind CSS

(**build-time**, **PostCSS**) are introduced as **new skills** — they are *not* refreshers.

- **Mindset:** willingness to work with **AI-assisted** development workflows (GitHub Copilot, ChatGPT) while verifying and understanding every line you keep.

RECALL / if needed — what counts as “programming background” You should be able to:

- read and modify a working program without step-by-step guidance;
- explain what a loop, a function, and a condition do;
- follow an error message back to its source;
- use a terminal and a code editor (VS Code) to open, run, and commit code.

If you have built anything in a course before (a PHP page, a Python script, a Java class), you have the baseline this course expects. What we add is the **JavaScript full-stack layer** and the **AI-assisted workflow** on top of that baseline.

2 The Contract

This is the **contract** between you and the course. Read it once, then keep it handy — it is the reference you return to whenever you need to know what is expected.

- **Duration:** 18 hours, delivered as 6 sessions × 3 h (S0–S5).
- **Session shape:** each session = 2 h lecture + 1 h hands-on practice.
- **One app, one spine:** a single running application anchors the whole course — the `develop` branch of `ebpro/fullstack-minimal-app`, referenced **by URL** (it is *not* vendored as a submodule). You **understand** the working catalog first, then **extend it layer-by-layer** until you have built the review layer.
- **AI-assisted, fundamentals first:** you use LLMs from day one as **productivity tools**, never as a substitute for understanding.

The S0–S2 fundamentals (HTML, CSS, JavaScript) are built **by hand** with chat-assisted **CRISP** prompting — no autonomous agents. From **S3 onward**, agents (Claude Code, OpenCode, Copilot agents) become an **optional accelerator**, and you remain the **quality gate**: you verify and can explain every line you keep.

! The one-app spine

There is **one** running example all term: `fullstack-minimal-app` (`develop` branch). You do **not** juggle parallel demo apps. The catalog it already ships is the **given end state**; your work is to understand it, then extend it — presentation → list → React components → routes → persistence — until the review layer is wired end-to-end.

if needed — the running layout (ports) When the app is running, you will see these local ports:

Layer	Port	Notes
Frontend (Vite dev server)	:5173	React UI
Backend (Express API)	:4000	REST API
MySQL	:3306	persistence
Scraper (mock)	:5000	home-stretch only — not wired in the 18 h

The first three rows are part of the in-book spine. The `:5000` scraper mock is only brought

online during the 1-month home stretch (see the Home stretch section).

3 The Capstone = the review layer

The capstone of the course is **the review layer** added on top of the catalog: a **reviews** table (MySQL), a set of **review routes** (Express API), and a **React review UI**. These three pieces share one **canonical review schema**, identical in the table, the routes, and the React components.

The review layer **completes in-book by S5**: by the end of the last session the reviews table, the review routes, and the React review UI are wired end-to-end and working. What is **deliberately NOT** built in the 18 h is the **scraper connection** (fetching reviews on-demand from external e-commerce sources); that is the **1-month home stretch** (see the Home stretch section below).

3.1 Objective

Build the **review layer** of a full-stack web application that stores and displays product reviews from multiple sources (Amazon, Best-Buy, Walmart). The catalog is the given end state (see the real app); you extend it by adding the review layer layer-by-layer across S1–S5. In the in-book capstone, reviews are **seeded directly** into the `reviews` table; the on-demand pull from external sources is the home stretch.

3.2 The canonical review schema

The same 8-field shape is used **identically** in the Express review routes and the React review components. The MySQL `reviews` table stores those same 8 fields **plus a `product_id` column** (a foreign key to `products.id`) so the review routes can run `WHERE product_id = ?`; that

column is a join detail, not part of the review's data shape:

```
review_id      // primary key
source         // Amazon | BestBuy
               | Walmart
author         // string
rating         // integer 1–5
title          // string
body          // string
created_at     // timestamp
verified_purchase // boolean
```

The **product** schema is the given end-to-end `snake_case` shape already shipped by the catalog (do **not** change it):

```
id, name, description, price, image_url,
category_id, category_name, created_at
```

In the **API and the frontend**, a review is associated with its product through the **route path** (`/api/products/:id/reviews`) — clients see only the 8 fields above. In the **database**, the `reviews` table additionally carries a `product_id` column (FK → `products.id`) so the review routes can filter with `WHERE product_id = ?`.

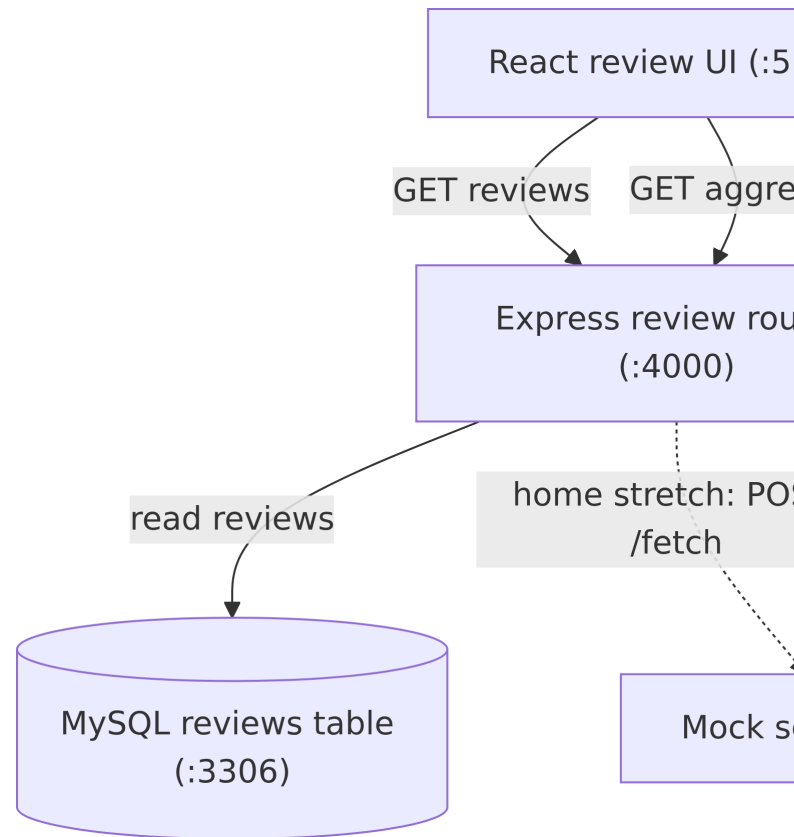
i What is deliberately NOT built in the 18 h

The **review table, the review routes, and the React review UI ARE** the capstone, and they are built **S1 → S5** (see the milestone table below). What is **NOT** connected in the 18 h is the **scraper**: the on-demand `POST /fetch` that pulls reviews from external e-commerce sources is the **1-month home stretch**. In-book, you never wire the scraper — review data is seeded directly into the `reviews` table.

3.3 Per-session milestones

Session	Capstone added	layer
S1	Review card — a static presentation of a single review (HTML/CSS)	
S2	Review list — render + filter a list of reviews from JSON (vanilla JS)	
S3	ReviewCard / ReviewList / ReviewForm — React components	
S4	Review routes — Express API endpoints (in-memory Map — Framing A ; no NEW DB for reviews, the app's existing MySQL is used for the product check)	
S5	reviews table + full frontend → backend → DB wiring — move from in-memory (Framing A) to the MySQL reviews table; capstone works in-book	

3.4 Frontend → backend flow (in-book)



Explanation:

1. The React review UI asks the Express review routes for the persisted reviews and for aggregated statistics.
2. The routes read the reviews from the MySQL **reviews** table.
3. The UI renders the review list, per-source averages, and the rating histogram.
4. The dashed path (**POST /fetch** → mock scraper) is the **home stretch** — it is not connected during the 18 h.

RECALL / if needed — full technical overview
Frontend (React) — suggested components:

1. **ProductPage** — displays product name/info; shows the overall average rating and total number of reviews.
2. **SourceBreakdown** — a table or bar chart of the average rating per platform.
3. **ReviewList** — individual reviews (author, rating, comment, date) with filtering by source.
4. **ReviewStats** — a histogram of 1–5 star ratings.
5. **Data-fetching hooks** — e.g. `useProductReviews(productId)` calling:
 - `GET /api/products/:id/reviews` → persisted reviews
 - `GET /api/products/:id/aggregate` → overall average, per-source averages, histogram

Backend (Node.js + Express) — in-book endpoints:

Method	Endpoint	Description
GET	<code>/api/products/:id/reviews</code>	Return all persisted reviews for the product
GET	<code>/api/products/:id/aggregate</code>	Return overall average, per-source averages, and rating histogram
POST	<code>/api/products/:id/reviews</code>	Fetch reviews from the scraper and store them

Reference DDL for the `reviews` table, derived from the canonical review schema (column

lengths are illustrative — the contract is the shape, i.e. the names + types). Note the extra `product_id` column: it is the foreign key that lets the review routes filter by product (`WHERE product_id = ?`); it is **not** one of the 8 fields the API and frontend expose.

```
CREATE TABLE reviews (
  review_id          VARCHAR(64)
  PRIMARY KEY,
  product_id         INT          NOT NULL,
  -- FK -> products.id (join detail,
  -- not part of the 8-field shape)
  source             VARCHAR(32) NOT
  NULL,              -- Amazon | BestBuy | Walmart
  author             VARCHAR(100)
  NOT NULL,
  rating             INT          NOT
  NULL CHECK (rating BETWEEN 1 AND 5),
  title              VARCHAR(255),
  body               TEXT,
  created_at         TIMESTAMP    NOT
  NULL DEFAULT CURRENT_TIMESTAMP,
  verified_purchase  BOOLEAN      NOT
  NULL DEFAULT FALSE,
  FOREIGN KEY (product_id) REFERENCES
  products(id) ON DELETE CASCADE
);

-- Index for the dominant query
-- pattern: "all reviews for product X"
-- (WHERE product_id = ?)
CREATE INDEX idx_reviews_product ON
reviews(product_id);
```

Pedagogical goals

- **Backend:** Node + Express, REST APIs, MySQL persistence, data aggregation.
- **Frontend:** React, state management, async data fetching, simple visualizations.
- **Full-stack integration:** connecting frontend, backend, and database.

- **AI-assisted development:** effective prompting and critical code evaluation.
- **Real-world skills:** e-commerce data, multi-source integration patterns.

4 Schedule (6 sessions)

The course is **18 h = 6 sessions × 3 h** (S0–S5). Each session = **2 h lecture + 1 h practice**. Every file link below is a real, relative link to a course document.

Session	Theme	Lecture file(s)	Practice file
S0	Orien- tation, Web Arch, Con- tract, AI	Intro- duction · Course Guide (this doc) · AI Quick Start	AI Quick Start
S1	HTML/ CSS (by hand)	HTML & CSS Es- sentials	HTML/ CSS
S2	JavaScript (for Re- act)	JavaScript for React	JavaScript for React
S3	React (fron- tend)	React Fron- tend	React Review
S4	Node/ Express (back- end)	Node & Express	Node & Express
S5	MySQL (CAP- STONE)	Data- base In- tegra- tion	Data- base In- tegra- tion

if needed — per-session goals & deliverables

- **S0 — Orientation, Web Arch, Contract, AI:** read this contract; set up the environment and the app from the URL; use an AI assistant (e.g. GitHub Copilot, Cursor, or similar) + LLMs to verify your first changes. Deliverable: working environment, app running on [:5173/:4000](#).
- **S1 — HTML/CSS (by hand):** refresh semantic HTML/CSS (a refresher), then add the new skills — Flexbox, Grid, responsive, Tailwind (build-time, PostCSS). Deliverable: the **review card** (static presentation of one review).
- **S2 — JavaScript (for React):** JavaScript fundamentals (variables, functions, async/await, DOM, [fetch](#)) and why React. Deliverable: the **review list** — render + filter reviews from JSON.
- **S3 — React (frontend):** components, JSX, props/state, hooks, data fetching. Deliverable: **ReviewCard / ReviewList / ReviewForm** React components.
- **S4 — Node/Express (backend):** Express fundamentals, routing, middleware, REST. Deliverable: the **review routes** (GET [/api/products/:id/reviews](#), GET [/api/products/:id/aggregate](#); POST [/api/products/:id/fetch](#) is the home stretch).
- **S5 — MySQL (CAPSTONE):** MySQL integration, connection pools, the [reviews](#) table. Deliverable: the **reviews table + full frontend → backend → DB wiring** — the capstone works in-book.

5 Assessment

The final grade is based on three components: **Attendance & Participation (10%)**, a **Final Quiz (60%)**, and a **Group Project (30%)**.

5.1 Attendance & Participation (10%)

- **Attendance tracking:** 6 sessions × 3 h = 18 h total.
- **Grading:**

Sessions attended	Points
6 / 6	10%
5	8%
4	6%
3 or fewer	0%

- **Participation bonus:** active engagement during lectures and practices (questions, helping peers, demonstrating understanding).

5.2 Final Quiz (60%)

- **Format:** individual written quiz (30 minutes).
- **Timing:** beginning of S5 (the last session) — before the capstone wiring is finalized. (Fall 2026 running: exact date announced in the course calendar.)
- **Coverage:** fundamental concepts with **emphasis on full-stack integration**.
- **Topics** (may be adjusted depending on class progress):

Topic	Weight
Full-stack integration (React ↔ Express ↔ MySQL, REST, data flow)	20%
Backend & databases (Express routing, middleware, SQL queries)	15%
Frontend frameworks (React com-	10%

Topic	Weight
ponents, hooks, state, JSX)	
JavaScript fundamentals (async/await, promises, DOM, ES6+)	8%
HTTP & REST concepts (methods, status codes, request/response)	5%
HTML/CSS & accessibility (semantic HTML, responsive design)	2%

- **Question types:** multiple choice, code-snippet analysis.
- **Focus:** understanding how the different layers communicate and integrate.

5.3 Group Project: the review layer, extended (30%)

- **Format:** group project (5 students per team). Teams merge and extend the work completed individually throughout the course, polish it, and add optional features.
- **Timeline:**
 - ▶ **S1–S4:** students work **individually** on the progressive milestones.
 - ▶ **End of S4 / start of S5:** teams form (instructor assigns team numbers).
 - ▶ **S5:** team consolidation — merging individual codebases, finishing the in-book wiring, planning the home stretch.
 - ▶ **Submission deadline:** 1 month after S5. (Fall 2026 running: exact deadline announced in the course calendar.)
- **Submission requirements** (code + `README.md` + `LLM_Usage.md` + video demo):

- Source code repository (GitHub) — or a `.zip` archive if there is no repository — including:
 - a clear `README.md` describing the functionalities;
 - a one-page `LLM_Usage.md` explaining LLM usage (prompts, verification, challenges);
 - a ≤ 3 -minute **demo video** via a private/unlisted YouTube link (screencast).

Grading breakdown:

Component	Weight
Functionality (catalog, review layer, aggregation/statistics, MySQL persistence, full-stack integration)	15%
Code quality & documentation (structure, README, naming/comments, error handling, Git history)	8%
LLM usage documentation (prompts with context, verification/corrections, critical reflection)	4%
Demo video (features demo, full-stack integration, technical challenges)	3%

5.3.a Optional Group Extension

If you are motivated, you can implement additional features beyond the core requirements for bonus points — e.g. the **real scraper integration** (this is exactly the home stretch), an

analytics dashboard with charts, or an admin panel for review moderation.

6 Home stretch (1 month)

After the 18 h, the course’s **home stretch** is to connect the **scraper** — the piece that was deliberately left unwired in-book.

- **What it is:** the on-demand `POST /api/products/:id/fetch` endpoint that calls a **mock scraper server** (`:5000`) to pull reviews from external e-commerce sources (Amazon, BestBuy, Walmart), then persists them into the `reviews` table. This turns the seeded in-book capstone into a truly on-demand aggregator.
- **Why it is after the course:** wiring an external source is the step that separates a working capstone from a production-style integration. It is the natural, scoped extension of exactly what you built in S1–S5 — not new material.

RECALL / if needed — known traps in the home stretch Two concrete mismatches to expect when you connect the scraper to the seeded database:

1. **Product IDs / names do not match the DB seed.** The mock scraper’s product IDs and names are not guaranteed to line up with the product rows already seeded in your MySQL database. Map them explicitly (or normalize) before persisting, or reviews will land on the wrong product.
2. **Only ids 1–3 have mock reviews.** The mock scraper only returns reviews for product **ids 1, 2, and 3**. Any other product will return no reviews — that is expected, not a bug.

Plan your work around these two facts before you start wiring.

7 FAQ

Q: Can I use a different tech stack (Vue instead of React, PostgreSQL instead of MySQL)? A: No. The course is designed around **React + Express + MySQL**. Assessment and support are based on this stack.

Q: What is the “one-app spine,” and where does it live? A: The single running example is the `develop` branch of `ebpro/fullstack-minimal-app`, referenced by **URL** (not vendored as a submodule). You understand it, then extend it layer-by-layer. There are no parallel demo apps.

Q: Is the scraper part of the in-book capstone? A: No. The in-book capstone is the review table + review routes + React review UI, built S1 → S5 with reviews **seeded** into the table. The scraper connection is the **1-month home stretch**.

Q: What if my team member isn’t contributing? A: Document the issue and contact the instructor immediately. In severe cases, individual grades may be adjusted.

Q: Can I submit the project individually instead of with my team? A: No. The project is team-based only. If you want individual bonus work, focus on exceeding the requirements in the main group project and document it in the README.

Q: How much should I rely on LLMs? A: LLMs are tools, not replacements for learning. In **S0–S2** the fundamentals are built **by hand** — a chat assistant may explain or check, but no autonomous agents. From **S3** onward, agents become an **optional accelerator**; you remain the **quality gate** and must understand and verify everything. **You’ll be tested individually on the quiz.**

Q: How is Tailwind used in the course? A: Tailwind (build-time, via PostCSS) — not a

CDN `<script>` tag. You add Tailwind to the build pipeline and use its utility classes in the markup.

Q: What if Docker doesn’t work on my computer? A: Use the local-installation path (Node.js + MySQL running natively). Bring your laptop if you need setup help, or ask on Discord.

Q: Will solutions be provided? A: The reference app is available on GitHub (`fullstack-minimal-app`). You should not copy it directly — the goal is to learn by doing and to extend it for your project.