

Team Project: Multi-Source Product Review Aggregator

Full-Stack Web Application Project

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

2026-10-14

Abstract Course lectures and practices for JavaScript full-stack web development with AI-assisted workflows.

Table of contents

1 Objective	1
2 Real-World Context: E-Commerce & Digital Marketing	1
3 Business Value	2
4 Project Description	2
4.1 Core Functionality	2
5 Development Approach	2
5.1 LLM-Assisted Development	2
6 Technical Overview	3
6.1 Frontend Requirements (React)	3
6.2 Backend Requirements (Node.js + Express)	3
6.3 MySQL Database Schema (Simplified)	4
6.4 Frontend → Backend → Scraper Flow	4
7 Pedagogical Goals	4

1 Objective

Build a full-stack web application that fetches, stores, and displays product reviews from multiple sources.

2 Real-World Context: E-Commerce & Digital Marketing

In today's competitive e-commerce landscape, **customer reviews are critical decision-making factors**. Studies show that:

- **93% of consumers** read online reviews before making a purchase
- Products with reviews have **270% higher conversion rates** than those without
- **Multi-source review aggregation** increases buyer confidence by providing diverse perspectives

However, products are often sold across multiple platforms (Amazon, BestBuy, Walmart, etc.), making it difficult for:

- **E-commerce managers** to monitor brand reputation across channels
- **Marketing teams** to analyze customer sentiment and identify pain points
- **Product managers** to prioritize feature improvements based on feedback
- **Customers** to compare reviews from different sources in one place

Your mission: Build a **Multi-Source Product Review Aggregator** that solves this problem by centralizing reviews, calculating meaningful statistics, and presenting actionable insights.

3 Business Value

This type of application is valuable for:

1. **E-commerce platforms** — Enrich product pages with cross-platform reviews
2. **Brand monitoring tools** — Track product reputation across multiple retailers
3. **Market research firms** — Analyze customer sentiment at scale
4. **Comparison websites** — Provide comprehensive product insights

4 Project Description

You are tasked with developing a web application using **React (frontend)** and **Node.js (backend)** that aggregates product reviews from multiple e-commerce platforms. A **simulated scraper server** will be provided to mimic external sources (Amazon, BestBuy, Walmart). A scraper is a program that extracts data from websites, here it extracts reviews from e-commerce sites.

4.1 Core Functionality

The system works **on-demand**: reviews are fetched **only when a user explicitly requests them** (avoiding unnecessary API calls and respecting rate limits). Once fetched, re-

views are **persisted in a MySQL database** so that:

- Aggregated statistics can be calculated and displayed instantly
- Review lists remain available without repeated external calls
- Historical data can be tracked over time
- Duplicate reviews are automatically prevented

5 Development Approach

i Progressive Learning: Step-by-Step Construction

This project will be built **incrementally across 6 practice sessions**, where each session builds upon the previous one:

1. **Practice 1** — Static HTML/CSS prototypes
2. **Practice 2** — JavaScript fundamentals and DOM manipulation
3. **Practice 3** — React components and frontend interactivity
4. **Practice 4** — Node.js/Express backend and REST APIs
5. **Practice 5** — MySQL database integration and persistence
6. **Practice 6** — Full-stack integration with external scraper service

You won't build everything at once! Each practice focuses on specific skills and gradually assembles the complete application.

5.1 LLM-Assisted Development

Throughout the course, you'll use **Large Language Models** (GitHub Copilot, ChatGPT, Claude, etc.) to accelerate development. Key skills:

- **Write clear prompts** with context and requirements
- **Critically evaluate** generated code for correctness and security
- **Iterate and refine** outputs to match specifications
- **Understand** the code you're using (LLMs help, but don't replace learning)

! Your Role

LLMs are **productivity tools**, not replacements for thinking. You must:

- Specify **what** to build clearly
- Evaluate **if** the generated code meets requirements
- Debug **when** things don't work as expected
- Integrate components into a **coherent system**

Success = **Human judgment + AI assistance**

6 Technical Overview

The following sections provide a **complete technical reference** for the project. Remember: you'll implement these **incrementally** across the 6 practices, not all at once!

6.1 Frontend Requirements (React)

The frontend is responsible for displaying products, fetching reviews on-demand, and visualizing aggregated statistics. Suggested components:

1. ProductPage

- Displays product name and info.
- Button **"Fetch Reviews"** triggers `POST /api/products/:id/fetch`.

- Shows **overall average rating** and **total number of reviews**.

2. SourceBreakdown

- Table or bar chart showing **average rating per platform**.

3. ReviewList

- Displays individual reviews with **author, rating, comment, and date**.
- Supports **pagination** or filtering by source.

4. ReviewStats

- Histogram of 1–5 star ratings.
- Optional: timeline chart showing trends over time.

5. Data fetching hooks

- Example:
`useProductReviews(productId)` to call backend endpoints:
 - `POST /api/products/:id/fetch` → fetch reviews
 - `GET /api/products/:id/reviews` → get persisted reviews
 - `GET /api/products/:id/aggregate` → get aggregated statistics

6.2 Backend Requirements (Node.js + Express)

1. **Fetch on-demand** from the simulated scraper server.
2. **Persist reviews in MySQL** for future use.
3. Provide the following API endpoints:

Method	Endpoint	Description
POST	<code>/api/products/:id/fetch</code>	Fetch reviews from scraper and store in MySQL
GET	<code>/api/products/:id/reviews</code>	Return all persisted reviews for the product

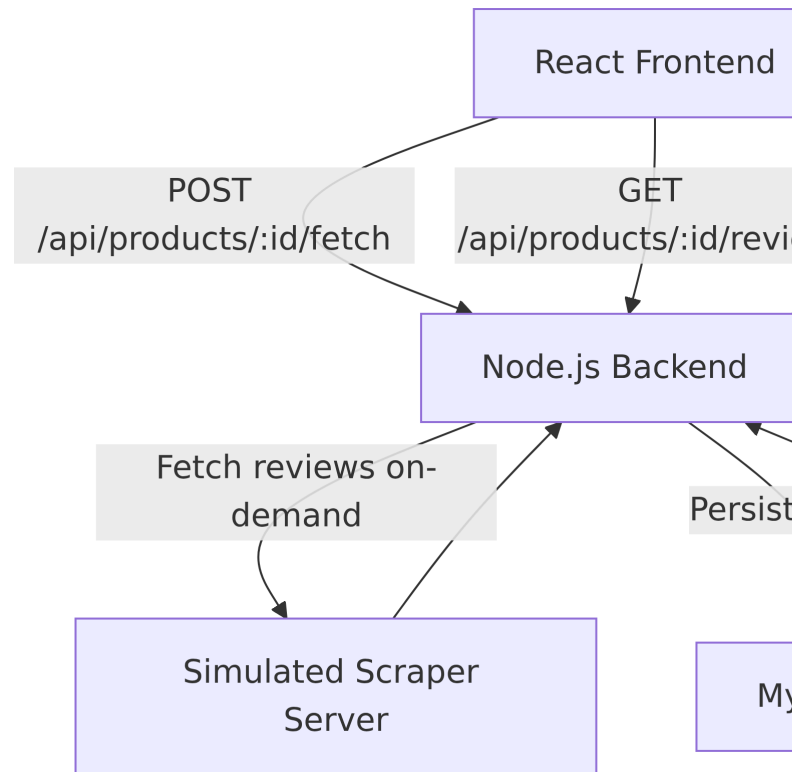
Method	Endpoint	Description
GET	/api/products/:id/aggregate	Return overall average, per-source averages, and rating histogram

6.3 MySQL Database Schema (Simplified)

```
CREATE TABLE reviews (
  id INT AUTO_INCREMENT PRIMARY KEY,
  product_id VARCHAR(50) NOT NULL,
  source VARCHAR(50) NOT NULL,
  review_id VARCHAR(50) NOT NULL,
  author VARCHAR(100),
  rating INT CHECK (rating BETWEEN 1
AND 5),
  title VARCHAR(255),
  body TEXT,
  created_at DATETIME,
  fetched_at DATETIME,
  UNIQUE KEY(product_id, source,
review_id)
);
```

- `review_id + source + product_id` ensures uniqueness and avoids duplicates.
- `fetched_at` tracks when the review was retrieved.

6.4 Frontend → Backend → Scraper Flow



Explanation:

1. Frontend triggers backend to fetch reviews **on-demand**.
2. Backend calls the simulated scraper to get reviews from multiple platforms.
3. Reviews are persisted in MySQL.
4. Frontend fetches reviews or aggregated stats from the backend to display charts, tables, and lists.

7 Pedagogical Goals

- **Backend:** Node + Express, REST APIs, MySQL persistence, data aggregation
- **Frontend:** React, state management, async data fetching, visualizations

- **Full-Stack Integration:** Connecting frontend, backend, database, and external services
- **LLM-Assisted Development:** Effective prompt engineering, critical code evaluation
- **Real-World Skills:** E-commerce data analysis, multi-source integration patterns