

Building the Static Review Card (S1 Practice)

A static review card in semantic HTML + CSS, prompted with CRISP in an AI chat

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

2026-10-14

Abstract Build the static review card — the S1 milestone of the WebLang capstone — as a 100% static page (semantic HTML + CSS, no JS, no framework): one featured review card laid out with Flexbox and a responsive grid of review cards (CSS Grid, mobile-first). You drive a chat assistant with the CRISP prompting pattern to generate and understand the CSS; you are the quality gate: you write the code, you verify it in the browser, and you iterate. The app’s real styling approach — Tailwind 3.4 utility classes — is referenced throughout.

Table of contents

1 Objective	1
2 The review schema	2
3 Starting point	3
4 The AI assistant: a chat, not an autonomous agent	3
5 Task 1 — Build the featured review card (HTML)	4
6 Task 2 — Build the review grid (HTML) . .	5
7 Task 3 — Style with CSS (box model + Flexbox + Grid)	6
7.1 Tailwind: the app’s real styling approach	7
8 Task 4 — Make the page responsive (media query)	7
9 The trap	8
10 Critique (what would you check in the assistant’s output?)	9

11 Quality gate (before committing)	9
12 Self-assessment / completion criteria	9

1 Objective

Build a **static review card** — one page, no JS, no framework — and drive it with a chat assistant in **CRISP** prompting:

1. a **featured review card** laid out with **Flexbox** (header: source badge + author + rating; body; footer: date + verified-purchase);
2. a **responsive grid** of 3–4 review cards (CSS Grid, mobile-first breakpoints) — “reviews for this product”;
3. a **shared stylesheet** (`styles.css`) that applies the box model, **Flexbox** on the featured card and **CSS Grid** on the grid, and

makes the whole thing **responsive** (at least one @media).

The deliverable is a page that you open in the browser and that **renders correctly on a wide screen and on a narrow screen**. It is the S1 milestone of the WebLang capstone: fully static, presenting the review layer that students will build across S1–S5.

The product context (name, image, price) is **given** — it comes from the app’s existing product catalog. You are not building the catalog; you are building the first piece of the review layer on top of it.

2 The review schema

Every review in the app carries these canonical fields. Do not invent your own, do not add others.

Field	Type	Display?	Notes
<code>review_id</code>	STRING	No	Internal identifier.
<code>source</code>	Amazon Best-Buy Walmart	Yes	Rendered as a badge.
<code>author</code>	string	Yes	Reviewer name.
<code>rating</code>	int 1–5	Yes	Integer — e.g. “5/5”. Never a decimal on a single review.
<code>title</code>	string	Yes	Short headline.

Field	Type	Display?	Notes
<code>body</code>	string	Yes	The review text.
<code>created_at</code>	date	Yes	e.g. “2026-03-14”.
<code>verified_purchase</code>	boolean	Yes	e.g. “✓ Verified purchase”.

`product_id` (the FK) is only added in the S5 database table; it is not part of the S1 card.

Four demo reviews serve as the data:

#	Source	Author	Rating	Title	Body	Created	Verified
1	Amazon	M. Dubois	5	“Excellent!”	“Battery lasts all day, sound is crisp...”	2026-03-14	Yes
2	Best-Buy	A. Chen	4	“Great for the price”	“Solid build, slight rattle at high volume...”	2026-02-28	Yes
3	Walmart	J. Okamoto	5	“My go-to phone”	“Comfortable, clear, pairs instantly...”	2026-01-02	No
4	Amazon	L. Rossi	3	“Decent, but not fine”	“Audio is fine”	2026-01-19	Yes

#	Source	Author	Rating	Title	Body	Created	Verified
				amaz-but ing” the case is flimsy...”			

3 Starting point

Create a `reviews/` folder containing two files. Here is the starting skeleton — copy it as-is, and the tasks will fill it in.

`reviews/index.html`:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width,
initial-scale=1.0">
  <title>Reviews - Wireless
Headphones</title>
  <link rel="stylesheet"
href="styles.css">
</head>
<body>
  <header>
    <h1>Reviews for Wireless
Headphones</h1>
    <p class="product-context">
      
      Wireless Headphones - $199.99
    </p>
  </header>
  <main>
    <!-- Featured review card goes
here (Task 1) -->
    <!-- Review grid goes here (Task
```

```
2) -->
</main>
<footer>
  <p>© 2026 - Demonstration review
layer</p>
</footer>
</body>
</html>
```

`reviews/styles.css` (shared sheet, empty for now):

```
/* styles.css - to be filled in Tasks
3 and 4 */
```

Open `reviews/index.html` in your browser (double-click, or `python3 -m http.server` run in `reviews/` if you prefer): you see the unstyled skeleton. That is your **baseline**.

4 The AI assistant: a chat, not an autonomous agent

Stated position (defer-agents). This practice is built on a **chat assistant** used with **CRISP** prompting. It is **not** an autonomous agent: the assistant cannot see your files, does not open the browser, executes nothing and commits nothing. **You** are the author and the **quality gate**: you write the code the assistant proposes, you verify it in the browser, and you iterate. The assistant *proposes*; you *decide*.

CRISP is the prompting pattern used at every step:

- **Context** — where you are, which files, which stack (static, no JS).
- **Role** — who the assistant should be (e.g. “senior front-end developer in CSS”).

- **Inputs** — exactly what it must produce (the file, the portion).
- **Steps** — the concrete steps, in order.
- **Parameters / constraints** — the limits (no JS, no framework, mobile-first, “output ONLY the code”).

For each task below, you will find a **CRISP prompt in a copy-paste box**. Paste it into your assistant, read its answer, **write it yourself** into the right file, then move to the **Verification**.

5 Task 1 — Build the featured review card (HTML)

Goal: fill the `<main>` of `reviews/index.html` with ONE featured review card, in semantic HTML.

1. Open `reviews/index.html`.
2. In your assistant, paste the CRISP prompt below.
3. Write the assistant’s answer inside `<main>` (before the grid comment). Do not change the `<header>` or the `<footer>`.
4. Move to the Verification.

CRISP — Task 1 (featured review card)

Context:

- I am building a 100% static review card (HTML + CSS, NO JS, NO framework).
- Files: ``index.html`` (reviews page) and ``styles.css`` (shared sheet).
- The page presents reviews for ONE product: "Wireless Headphones" (given context — do not build a product listing).

Role:

- Senior web developer specialized in accessible semantic HTML.
- You propose ONLY HTML — no CSS,

no JS.

Input:

- Write the HTML for ONE featured review card to place INSIDE `<main>`, before the `<!-- Review grid -->` comment.

Steps:

1. An `<article class="review-card featured">` containing:

- A `<header class="review-card-header">` with:
 - A `<span class="review-source">` for the source badge (e.g. "Amazon").
 - A `<span class="review-author">` for the author name.
 - A `<span class="review-rating">` for the rating as an integer out of 5 (e.g. "5/5").
- A `<h2 class="review-title">` for the review title.
- A `<p class="review-body">` for the review body text.
- A `<footer class="review-card-footer">` with:
 - A `<time class="review-date">` for `created_at` (e.g. "2026-03-14").
 - A `<span class="review-verified">` for `verified_purchase` (e.g. "✓ Verified purchase").

2. Use the FIRST canonical review: source "Amazon", author "M. Dubois", rating 5, title "Exceeded expectations",

body "Battery lasts all day, sound is crisp and the noise cancelling is impressive for the price.",
`created_at` "2026-03-14",
`verified_purchase` true.

Parameters / constraints:

- Semantic HTML only (`<article>`, `<header>`, `<footer>`, `<h2>`, `<p>`, `<span>`, `<time>`).
- NO JS, NO framework, NO inline style

```
attributes.  
- Do NOT display review_id or  
product_id.  
- Output ONLY the HTML to insert into  
<main> (no explanation).
```

Verification (browser). Open `reviews/index.html`:

- The featured card displays with source badge, author, rating “5/5”, title, body, date, and “✓ Verified purchase”.
- In DevTools → Elements, the structure is semantic: `<article class="review-card featured">` → `<header>` → `<h2>` → `<p>` → `<footer>`.
- The rating is an **integer** (“5/5”), not a decimal.
- No `review_id` or `product_id` is displayed.

6 Task 2 — Build the review grid (HTML)

Goal: add a responsive grid of 3–4 review cards after the featured card, in semantic HTML.

1. Open `reviews/index.html`.
2. Paste the CRISP prompt below into your assistant.
3. Write the answer into `<main>` (replacing the `<!-- Review grid -->` comment).
4. Move to the Verification.

```
CRISP – Task 2 (review grid)  
=====
```

Context:

- Static reviews page (HTML + CSS, NO JS, NO framework).
- Files: ``index.html`` and ``styles.css``.
- The page already has ONE featured review card. Now add a grid of 3 more review cards (total 4 on the page).

Role:

- Senior web developer in accessible semantic HTML. HTML only.

Input:

- Write the HTML for the review grid to replace the

`<!-- Review grid -->` comment inside `<main>`.

Steps:

1. A `<section class="review-grid" aria-label="All reviews">` containing 3 `<article class="review-card">` elements (the grid; the featured card above is separate).

2. Each review card has the same internal structure as the featured card:

```
<header class="review-card-header"> (source badge, author, rating),
```

```
<h2 class="review-title">, <p class="review-body">,
```

```
<footer class="review-card-footer"> (date, verified).
```

3. Use reviews 2, 3, and 4 from the canonical set:

- BestBuy, A. Chen, 4, "Great for the price",

```
"Solid build, slight rattle at high volume but overall good value.",  
2026-02-28, verified true.
```

- Walmart, J. Okafor, 5, "My go-to headphones",

```
"Comfortable, clear sound, pairs instantly with every device.",  
2026-04-02, verified false.
```

- Amazon, L. Rossi, 3, "Decent, not amazing",

```
"Audio is fine but the case is flimsy and the mic is weak.",  
2026-01-19, verified true.
```

Parameters / constraints:

- Semantic HTML, NO JS, NO framework, NO inline styles.

- Ratings are INTEGERS (1–5), never decimals.
- Do NOT display review_id or product_id.
- Output ONLY the HTML to insert (no explanation).

Verification (browser). Open `reviews/index.html`:

- The featured card is followed by 3 more cards (4 total).
- Each card shows source, author, integer rating, title, body, date, and verified status.
- In DevTools → Elements, the grid is: `<section class="review-grid">` → `<article class="review-card">` × 3.
- The unverified card (Walmart) does NOT show “✓ Verified purchase”.

7 Task 3 — Style with CSS (box model + Flexbox + Grid)

Goal: fill `reviews/styles.css` with the base styling: the box model on the cards, **Flexbox** on the featured card’s header, **CSS Grid** for the review grid, and distinct styles for each review field. **Mobile-first** approach.

1. Open `reviews/styles.css`.
2. Paste the CRISP prompt below into your assistant.
3. Write the answer into `styles.css`.
4. **Understand one property:** pick a CSS property you do not yet master (e.g. `grid-template-columns` or `gap`). Ask the assistant:

“Explain the CSS property `grid-template-columns` simply: what it is for, what its values do, and an example.”

Re-read its answer, then **re-explain it out loud** without looking at it — that is your quality gate: you do not keep what you do not understand.

5. Move to the Verification.

CRISP – Task 3 (styling)

=====

Context:

- Static reviews page; ``styles.css`` is the shared sheet.
- The page has a featured `<article class="review-card featured">` (Flexbox layout) and a `<section class="review-grid">` of 3 `<article class="review-card">` elements (CSS Grid).

Role:

- Senior front-end developer in CSS (box model + layout).

You propose ONLY CSS – no HTML, no JS.

Input:

- Write the COMPLETE content of ``styles.css``.

Steps:

1. Base: ``box-sizing: border-box`` everywhere; readable font; muted colors.
2. The review cards: a visible box model (padding, border, border-radius, margin, background) on `.review-card`.
3. The FEATURED card header: Flexbox on `.review-card-header` (`display: flex; align-items: center; gap`) – source badge, author, rating in a row.
4. The FEATURED card footer: Flexbox on `.review-card-footer`

```

    (display: flex; justify-content:
space-between; align-items: center).
5. The GRID: a mobile-first CSS grid
on .review-grid
    (display: grid; grid-template-
columns: 1fr; gap) – one column by
default.
6. Distinct styles for .review-source
(badge: background, color,
border-radius, padding), .review-
rating (bold), .review-verified
(green color).
Parameters / constraints:
- PURE CSS, NO framework, NO
preprocessor, NO JS.
- MOBILE-FIRST: the base style
(without @media) must be correct on
a narrow
screen.
- Output ONLY the CSS (no
explanation).

```

Verification (browser). Re-open `reviews/index.html`:

- The cards have a visible box model (padding, border, rounded corners, background).
- The featured card's header shows source badge, author, and rating in a **row** (Flexbox).
- On a narrow screen (DevTools → device mode, ~375 px), the grid is in **one column**.
- The source badge has a distinct style (background color, rounded corners).
- The “✓ Verified purchase” text is green.

7.1 Tailwind: the app's real styling approach

The `fullstack-minimal-app` uses **Tailwind CSS 3.4** — a utility-first framework. The same layout you just wrote by hand maps to Tailwind utilities directly in the markup. Here is how the featured card header you just styled translates:

```

<!-- Hand-written CSS (what you
built) -->
<header class="review-card-header">
    <span class="review-
source">Amazon</span>
    <span class="review-author">M.
Dubois</span>
    <span class="review-rating">5/5</
span>
</header>

<!-- Same layout, Tailwind 3.4
utilities (what the app uses) -->
<header class="flex items-center
gap-2">
    <span class="rounded-full bg-
blue-100 px-2 py-0.5 text-xs font-
medium text-blue-800">Amazon</span>
    <span class="text-sm text-
gray-600">M. Dubois</span>
    <span class="ml-auto text-sm font-
bold">5/5</span>
</header>

```

The concepts are identical — Flexbox (`flex items-center gap-2`), spacing (`gap-2, px-2`), colors (`bg-blue-100, text-blue-800`). You wrote the CSS by hand so you can **read** the Tailwind the app ships. You do not need to install Tailwind for this practice; the point is that you now recognise the mapping.

8 Task 4 — Make the page responsive (media query)

Goal: extend `reviews/styles.css` with at least one `@media` that moves the grid to multiple columns on wide screens. Still **mobile-first** (start from the narrow base style, then widen).

1. Open `reviews/styles.css` (the Task 3 content is already there).

2. Paste the CRISP prompt below into your assistant.
3. Write the answer **as an addition** to `styles.css`.
4. **Debug the layout:** if the grid does not behave as expected, do not panic — ask the assistant to help you diagnose, giving it the exact symptom:

“In `styles.css`, my `.review-grid` (display: grid) displays in one column even above 768 px. Here is my CSS: ... Here is what I see in the browser: ... HELP me identify the cause (do not rewrite the whole file).”

The assistant *proposes* a diagnosis; **you** apply it and re-verify.

5. Move to the Verification.

CRISP – Task 4 (responsive)

=====

Context:

- `'styles.css'` already contains the base, the box model, the Flexbox on `.review-card-header` / `.review-card-footer`, and the mobile-first grid of `.review-grid` (1 column).

Role:

- Senior front-end developer in responsive design. CSS only.

Input:

- Write the rules to **ADD** to `'styles.css'` to scale up.

Steps:

1. A `@media` (min-width: 768px) that moves `.review-grid` to multiple columns (e.g. `grid-template-columns: repeat(2, 1fr)`).
2. A `@media` (min-width: 1024px) that moves `.review-grid` to 3 columns

```
(grid-template-columns: repeat(3, 1fr)).
```

3. Adjust font sizes / spacing if needed on wide screens.

Parameters / constraints:

- At LEAST one `@media` ; MOBILE-FIRST (min-width, start from the narrow style).
- PURE CSS, NO JS, NO framework.
- Output **ONLY** the CSS to add (no explanation).

Verification (browser). Resize the window (or DevTools → device mode):

- At ~375 px: grid in **one column**.
- Above 768 px: grid in **two columns**.
- Above 1024 px: grid in **three columns**.
- **No horizontal scrolling** at any size.
- The featured card remains full-width at all sizes.

9 The trap

A media query in the wrong direction.

With **mobile-first**, the base style is the **narrow** screen, and the `@media` (min-width: 768px) **widens** it. If you write `@media` (max-width: 768px) thinking “small screen” while your base is already the small screen, the rule does nothing visible (or inverts the behavior). Symptom: “nothing changes when I resize.” Check min-width against max-width, and that the breakpoint is above the size you are testing.

A badly linked CSS file. If your page displays **without style**, check that the `<link rel="stylesheet" href="styles.css">` is in the `<head>` and that `styles.css` is **in the same folder** as the `html` file (same relative path). A wrong relative path = a healthy but unstyled page.

A decimal rating. A single review has an **integer** rating 1–5. “4.5/5” is an *average* across

many reviews — it does not belong on one card. If the assistant outputs a decimal, fix it: the schema says `int 1-5`.

10 Critique (what would you check in the assistant's output?)

Read the assistant's output through the lens of **common AI errors**. For each point, note whether it is clean or needs fixing:

- **div-soup vs semantic**. Does it use `<article>`, `<header>`, `<footer>`, `<time>` — or a pile of `<div>` without landmarks?
- **Missing alt**. Does the product thumbnail have a descriptive `alt`?
- **Inline styles**. Did it slip in a `style="..."` attribute or a script block? Everything must live in `styles.css`.
- **Desktop-first drift**. Is the **base** style already correct on a narrow screen (1 column grid), without the mobile depending on it?
- **Phantom fields**. Did it invent a field that is not in the 8-field schema (e.g. a “helpful” count, a “reply” button, a `product_id`)?
- **Decimal rating**. Did it output “4.5/5” or “4.7 stars” on a single review? The rating is an integer 1–5.
- **Displaying internal fields**. Did it render `review_id` or `product_id`? These are internal/association only — never displayed.

11 Quality gate (before committing)

Check **everything** before considering the practice done:

- ☐ The page opens in the browser **without errors** (no error message in the console).
- ☐ Semantic HTML: `<header>`, `<main>`, `<article>`, `<footer>`, `<time>` present; **one**

`<h1>` per page; every `<img>` has a descriptive `alt`.

- ☐ **No JS**, **no framework**, **no inline style**="..." (search in the two files).
- ☐ The **box model** is visible on the cards (padding, border, margin, background).
- ☐ The **featured card header** uses **Flexbox**; the **grid** uses **CSS Grid**.
- ☐ **At least one @media**; the grid adapts at 768 px; **no horizontal scrolling** on mobile.
- ☐ The rating is an **integer** (“5/5”), not a decimal.
- ☐ No `review_id` or `product_id` is displayed.
- ☐ You can **re-explain** each CSS property you used without looking at it (the assistant helped, but you are the one who understands).
- ☐ You can **read** the Tailwind equivalent of your CSS and identify the mapping.

12 Self-assessment / completion criteria

- ☐ `reviews/` contains `index.html` and `styles.css`.
- ☐ The page presents **one featured review card** (Flexbox header: source badge, author, integer rating; title; body; Flexbox footer: date, verified) + **a grid of 3 more review cards** (CSS Grid).
- ☐ The 8-field schema is respected: `source`, `author`, `rating` (int), `title`, `body`, `created_at`, `verified_purchase` displayed; `review_id` and `product_id` NOT displayed.
- ☐ The CSS is **mobile-first**: `base` = narrow screen, `@media (min-width: 768px)` widens (multi-column grid).
- ☐ You referenced the **Tailwind 3.4** equivalent of at least one layout rule.
- ☐ You used the assistant in **CRISP** at every step, you **explained** at least one property,

and you **debugged** (or diagnosed) at least one layout.

- ☐ You are the **quality gate**: everything was verified in the browser, nothing was kept “blind”.