**Challenge Overview**  
Implementing SmartFit’s end-to-end JSON‐backed workflow required balancing simplicity and robustness. We needed a zero-dependency data store (plain JSON) yet still handle file corruption and concurrent writes gracefully. On the front end, we strove for a fast, user-friendly experience—from form submission to real-time order display—while keeping our code modular and maintainable.

**Key Trade-Offs**

1. **Persistence Layer: JSON vs. Database**
   * *Why JSON?* Easy to configure, human-readable, no external services.
   * *Downside:* No built-in transaction support; risk of file corruption under concurrent writes.
   * *Mitigation:* Wrapped all loads in try/catch, reset on JSON decode errors, and designed writes to overwrite atomically.
2. **Styling: Inline vs. Cascading CSS**
   * *Quick Prototype:* Inline styles—fast but scattered.
   * *Better Practice:* Central CSS classes for. order-card,. error, etc., improving consistency and ease of future redesigns.
3. **CORS Policy**
   * *Open CORS (“*”)\* for rapid local testing versus *Restricted CORS* in production.
   * *Decision:* Start with wildcard during development; lock down to the live front-end origin when deploying.

**Lessons Learned**

* **Anticipate Failure Modes:** Even a simple JSON file can become unreadable. Defensive coding around file I/O is non-negotiable.
* **User Feedback Is Key:** Immediate success/error messages and a “Loading…” state turn an opaque system into a trusting experience for end users.
* **Separation of Concerns:** Moving styles into CSS and logic into standalone JS functions pays dividends in code readability and maintenance.
* **Graceful Degradation:** Designing for network errors and empty data ensures the app never leaves a user stranded with a blank screen.