Literary Lighthouse Frontend – Angular Implementation Overview

> Presentation deck (convertible to PPT) describing how Angular is used in this project.
>
> Suggested slide ratio: 16:9 – Each H2 becomes a slide. Speaker notes under **Speaker Notes** blocks.

1. Title / Opening

Literary Lighthouse – Angular Frontend Architecture
Building a performant, modular, server-aware digital library UI with Angular

Presenter: (Manish Shaw)
Date: (01-10-2025)

Objectives:

- Show core Angular patterns in the codebase
- Explain why these choices improve DX & UX
- Prepare team for future feature extensions (Billing, Analytics, etc.)

Speaker Notes: Introduce scope: not every line of code—focus on structure, patterns, scalability.

2. Tech Stack Snapshot

Layer	Technology
Framework	Angular (standalone APIs)
Rendering	CSR + Prerender (static routes)
Styling	SCSS (component-scoped)
Data	REST (Node/Express API + SQLite / JSON fallback)
State	Signals + service singletons
Tooling	HMR, lazy loading, local caching

Key Angular Features Used: standalone components, functional route guards, signals, DI services, lazy-loaded routes, template directives, prerender build.

Speaker Notes: Emphasize adoption of *modern Angular* (no NgModules, functional

3. High-Level Architecture

```
frontend/
  src/app/
    auth.service.ts
    book.service.ts
    cart.service.ts
    main-content/ (smart container)
    booklist/ (presentational)
    checkout/ (multi-step orchestrator)
    billing/ (lazy admin page)
    login-modal/ (UX service + component)
```

Separation of Concerns:

- Services = data & state boundaries
- Components = UI rendering + minimal orchestration
- Guards = access policy (admin-only)
- Config (API_BASE) centralizes backend integration

4. Routing & Navigation

Highlights:

- Defined in `app.routes.ts` using `Routes` array
- Lazy routes: `login`, `billing`
- Guard: `adminGuard` using `inject()` (SSR compatible)
- 404 strategy: catch-all redirect to home

Example:

```
export const routes: Routes = [
  { path: '', component: MainContentComponent },
  { path: 'billing', canActivate: [adminGuard], loadComponent: () => import('./billing/billing').then(m => m.BillingComponent) },
];
```

Speaker Notes: Underscore zero NgModule overhead—faster startup and simpler mental model.

5. Standalone Components Strategy

Why standalone:

- Faster boot: no NgModule compilation cost
- Co-located imports → explicit dependencies
- Encourages granular lazy loading

Patterns Used:

- Container vs Presentational (e.g., `main-content` vs `booklist`)
- Reusable directives (e.g., image fallback)
- Inline template composition with structural directives (`*ngIf`, `@if`, `*ngFor`)

Speaker Notes: Emphasize maintainability & testability improvements.

6. State Management with Signals

We use Angular signals (no external lib):

- `AuthService`: `userSignal`
- `CartService`: `cartItems` signal

Why Signals:

- Reactive reads in templates without manual subscriptions
- Avoids global mutable stores / over-engineering

Example:

```
private _user = signal<UserRecord | null>(null);
get currentUser() { return this._user(); }
```

Speaker Notes: Stress simplicity vs. redux/ngrx for current scale.

7. Data Access & Caching

`BookService` responsibilities:

- LocalStorage caching (versioned via `/booksVersion` endpoint)
- Forced refresh on admin stock changes
- Server-side filtering via query params: `GET /booksData?category=1,2`

Flow:

1. Check cached copy + version
2. Fetch if stale / filtered
3. Normalize categories → number[]

Example:

```
async fetchByCategories(ids: (number|string)[]) {
  const qs = ids.length ? `?category=${ids.join(',')}` : '';
```

8. Auth & Role Handling

Auth Pattern:

- Phone/email lookup against backend users
- Admin inferred (temporary logic: specific phone)
- Persistent session via localStorage hydrate

Guard:

```
export const adminGuard: CanActivateFn = () => {
  const auth = inject(AuthService); const router = inject(Router);
  return auth.isAdmin ? true : router.parseUrl('/');
};
```

Speaker Notes: Future: move to backend-issued JWT & roles column.

9. Cart & Server Synchronization

Features:

- Optimistic UI add/remove
- Server reconciliation if request fails
- Out-of-stock prevention (frontend + backend 409 handling)

Snippet:

```
if (res.status === 409) {
  // revert optimistic increment
  // show toast
}
```

Speaker Notes: Stress resilience + consistent multi-device state.

10. Checkout & Orders

Checkout UX Enhancements:

- Step-based validation & toasts
- Auto-prefill from user profile
- Real order POST to `/orders` (clears cart server-side)

Returns:

```
{ "orderId": "ORD...", "ack": "ACK123456", "total": 450, ... }
```

Speaker Notes: Foundation for future billing & invoice generation.

11. Admin Features

Implemented:

- Toggle book stock (PATCH)
- Orders retrieval
- Billing route placeholder (guarded)

Mechanism:

- Book updates bump `booksVersion` → clients invalidate stale cache

Speaker Notes: Will evolve into reporting dashboard (charts, metrics).

12. Performance & DX

Techniques:

- Local caching + version invalidation
- Lazy loading non-critical routes
- Prerender of static routes (faster first contentful paint)
- HMR for rapid iteration (reduced reload disruption)

Speaker Notes: Mention potential additions (image optimization, bundle analysis).

13. Accessibility & UX

Practices:

- Semantic buttons & ARIA labels (theme toggle, status messages)
- Toast feedback for cart / auth gating
- Disabled interactions for out-of-stock items

Speaker Notes: Roadmap: keyboard trap audit, focus outlines, high contrast mode.

14. Extensibility Roadmap

Planned / Easy Next Steps:

- Server-side search (query param `search=`)
- Role-based access (DB-backed roles)
- Billing: revenue aggregation & CSV exports
- Notifications: WebSocket or SSE for low-stock alerts
- Observability: Log enrichment + client perf telemetry

Speaker Notes: Highlight prioritization: search + roles first unlock richer admin tooling.

15. Risks & Mitigations

Risk	Mitigation
Hard-coded admin phone	Add roles column + backend auth
LocalStorage tampering	Server validation & signed tokens
Cache staleness	Version endpoint already in place
Growing bundle size	Continue lazy splitting; analyze source maps

Speaker Notes: Encourage periodic audits each release.

16. Summary / Call to Action

We leveraged modern Angular patterns to build:

- Reactive, cached, role-aware UI
- Resilient cart & order flows
- Extensible admin surface (billing coming)

Next: finalize billing analytics + hardened auth.

Thank You – Questions?

Speaker Notes: Invite feedback on architectural choices before billing build-out.

17. Appendix (Optional Code Deep Dives)

Include only if time permits:

- Detailed Cart optimistic update flow
- Stock toggle + cache invalidation path
- Order creation + cart clearing sequence diagram

Conversion Tips

1. Copy this Markdown into PowerPoint / Google Slides via a Markdown-to-slides tool (Marp, Deckset, Pandoc, VSCode Marp extension).
2. Use one visual (diagram or screenshot) per dense slide for balance.
3. Highlight code with monospace + subtle background.
4. Add a live demo after Slide 10 if possible.

Tools / Commands (Internal Only – Remove for Final Deck)

```
# Rebuild frontend
npm run build
# Start backend & frontend (dev)
node backend/api-server.js &
cd frontend && npm start
```