Frontend Handbook

🎯 Purpose

A practical, opinionated handbook for building **production-grade frontend** apps. It standardizes architecture, DX, quality, security, performance, and delivery so any engineer can ship with confidence.

🧭 Core Principles

- **Clarity over cleverness**: readable > magical.
- **Fail loudly in dev; fail gracefully in prod**: error boundaries + fallbacks.
- **Small, composable pieces**: feature-first foldering, pure UI components.
- **Data flows are explicit**: single API layer, predictable state.
- **Automate quality**: lint, tests, preview deploys.
- **Performance is a feature**: budget, measure, optimize.

🏗️ Project Setup & Tooling

- **Framework**: React + Vite (preferred) or Next.js if SSR/SEO critical.
- **Lang**: TypeScript "strict": true.
- **Styling**: Tailwind CSS + design tokens; shadcn/ui where helpful.
- **Forms**: React Hook Form + Zod resolvers.
- **State**: Redux Toolkit (+ Thunk). Optional: RTK Query or React Query for server cache.
- **HTTP**: Axios instance with interceptors, retries, & cancel tokens.
- **Routing**: React Router (Vite) or Next Router (Next.js). Lazy-loaded routes.

- **Testing**: Vitest/Jest (unit), Testing Library (component), Playwright/Cypress (e2e).
- **Quality**: ESLint, Prettier, Husky + lint-staged, Type-check in CI.
- **Monitoring**: Sentry (errors) + Web Vitals to analytics.
- **i18n**: i18next (optional), always externalize copy.
- **Env**: .env per env + schema-validated with Zod.

🗂️ Folder Structure (Feature-Sliced Architecture)

```
src/
  app/          # app shell, providers, router, store, error boundary
  shared/        # tokens, icons, ui primitives, utils
  entities/       # reusable domain entities (User, Course, Payment)
  features/       # feature slices (Auth, Checkout, Uploads, Board)
  pages/         # route-level screens (Home, Dashboard, BoardPage)
  widgets/        # composed blocks used across pages
```

Rules:

- No cross-feature imports except via shared/ or declared public API of a slice.
- Each slice: index.ts, ui/, model/ (state, thunks), api/, lib/, types/.

🧩 UI & Styling

- Tailwind utility-first; extract repeated patterns into components.
- Use **design tokens**: colors, spacing, radii, typography in shared/tokens.
- Accessible primitives only: buttons are <button>, links are <a>.
- Dark mode via class strategy; respect prefers-color-scheme.
- Loading UX: skeletons/shimmers; empty states; retry CTAs on failure.

🔗 API Layer (Axios)

- Single axios instance in shared/api/http.ts with:
    - Base URL from env
    - JSON timeouts (10s)
    - Auth header from secure store
    - Retries (exponential backoff) only for idempotent methods
    - Request cancellation on route change

Example: axios instance

```
import axios from 'axios';
import { getAccessToken, refresh } from '@/shared/auth';

export const http = axios.create({
  baseURL: import.meta.env.VITE_API_URL,
  timeout: 10000,
});

http.interceptors.request.use((cfg) => {
  const t = getAccessToken();
  if (t) cfg.headers.Authorization = `Bearer ${t}`;
  return cfg;
});

let refreshing: Promise<string | null> | null = null;
http.interceptors.response.use(undefined, async (err) => {
  const { config, response } = err;
  if (response?.status === 401 && !config._retry) {
    config._retry = true;
    refreshing = refreshing ?? refresh();
    const newTok = await refreshing.finally(() => (refreshing = null));
    if (newTok) {
      config.headers.Authorization = `Bearer ${newTok}`;
      return http(config);
    }
  }
  return Promise.reject(err);
});
```

🧠 State Management

- Local UI state: useState/useReducer first.
- Server cache: React Query or TanStack Query (preferred over manual fetch + Redux store).
- Global app state: Redux Toolkit slices; async via Thunks.
- Keep store minimal; derive with selectors; normalize by ID for lists.

✅ Forms & Validation

- React Hook Form + Zod schemas per form in features/<x>/model/validation.ts.
- Show field-level errors; disable submit during pending; optimistic UI when safe.

🧯 Errors & Boundaries

- App-level ErrorBoundary with reset to home.
- Feature-level boundaries where risky (uploads, payments).
- Log to Sentry with user context; never leak secrets.

Error Boundary skeleton

```
function App() {
  return (
    <ErrorBoundary fallback={<CrashScreen/>}>
      <RouterProvider router={router} />
    </ErrorBoundary>
  );
}
```

🔒 Security Checklist

- Escape/encode all user content; use dangerouslySetInnerHTML only with sanitizer.
- Content Security Policy via meta/headers, restrict script-src.
- Cookies: HttpOnly on server; in FE, store tokens in memory; refresh via secure API.
- Prevent CSRF for cookie-based auth (SameSite/CSRF token).
- Avoid leaking envs in client bundles; whitelist only VITE_* that are safe.

♿ Accessibility (A11y)

- Semantic HTML; labels for inputs; alt text for images.
- Focus states visible; trap focus in dialogs; aria-* where needed.
- Color contrast ≥ 4.5:1; never convey info by color alone.
- Keyboard nav tests in PR (Tab/Shift+Tab path).

⚡ Performance

- **Budgets**: LCP < 2.5s, CLS < 0.1, FID/INP good, TTI < 3.5s.
- Code-split by route; lazy components; prefetch next route on hover.
- Image optimization: responsive srcset, next-gen formats, lazy loading.
- Cache: HTTP caching, SWR staleness, CDN.
- Avoid heavy libs; measure with bundle analyzer; tree-shake.
- Memoization: React.memo, useMemo/useCallback only for hotspots.

🔍 Logging & Analytics

- shared/log.ts thin wrapper; levels: info/warn/error.
- Sentry for errors; console noise stripped in prod.
- Track Web Vitals to analytics provider.

🧪 Testing Strategy

- **Unit**: pure functions, reducers, hooks.
- **Component**: Testing Library – test behavior over implementation.
- **Integration**: API hooks + components + router.
- **E2E**: Playwright happy paths (auth, critical flows like payments/uploads).
- Minimum coverage gates for critical paths; snapshot tests sparingly.

🔁 Release Hygiene

- Conventional Commits; auto-changelog.
- PR template includes: scope, screenshots, tests, a11y notes, perf notes.
- Definition of Done:
  - Types complete
  - Errors handled
  - Tests added/updated
  - Perf checked (bundle, Web Vitals)
  - Docs/Storybook updated (if component)

📦 Reusable Patterns (Snippets)

Skeleton Loader

```
export const Skeleton = ({ className = '' }) ⇒ (
  <div className={`animate-pulse bg-muted/50 rounded ${className}`} /
>
);
```

Lazy Route

```
const BoardPage = lazy(() ⇒ import('@/pages/BoardPage'));
<Route path="/board/:id" element={
  <Suspense fallback={<Skeleton className="h-64"/>}>
```

```
    <BoardPage/>
  </Suspense>
}/>
```

Zod + RHF

```
const schema = z.object({ email: z.string().email(), password: z.string().min(8) });
const { register, handleSubmit, formState:{errors} } = useForm({ resolver: zodResolver(schema) });
```

Feature Slice Example

```
features/auth/
  api/login.ts
  model/slice.ts
  model/thunks.ts
  model/selectors.ts
  ui/LoginForm.tsx
  index.ts
```

🛡️ Edge Cases & Playbooks

- **Payments**: “processing” state until webhook confirms; idempotent actions.
- **Uploads**: use presigned/ImageKit; show progress; handle cancellations.
- **Offline**: basic PWA; queue writes if needed.
- **Timezones**: always store UTC; format with locale on client.

📋 Frontend PR Checklist (copy into repo)

- Feature follows FSA foldering
- Types complete (no any)
- API calls via shared axios instance

- Error & empty states implemented
- Keyboard & screen reader paths verified
- Images optimized/lazy
- Route lazy-loaded if heavy
- Unit/Component tests added
- No console noise; logs via logger only
- ENV vars documented

📎 Appendices

- **Recommended libs**: axios, react-hook-form, zod, redux-toolkit, react-query/rtk-query, clsx, jotai/zustand (optional), date-fns, i18next, msw (tests), playwright.
- **VSCode setup**: format on save, eslint, tailwind intellisense.
- **Storybook** (optional): for shared components.

✅ How to Use This Handbook

1. Scaffold repo with the stack above.
2. Keep this doc in /docs/frontend-handbook.md.
3. Enforce via CI (lint, type, test) and PR template.
4. Review against the PR checklist before merging.