Engineering Baseline for Web Clients

Scope and Inheritance

This document defines the mandatory implementation stack and conventions for browser-based applications in the suite.

It exists to:

- Prevent framework sprawl
- Enforce browser security discipline
- Ensure consistent developer experience
- Reduce cognitive load and hiring friction
- Align client behaviour with edge and internal service contracts

Web clients must comply with this document and the relevant cross-service contracts defined in the Web Suite Architecture.

Runtime and Language

- Language: TypeScript
- Package manager: pnpm
- TypeScript strict mode required
- Build must fail on type errors
- Lockfile must be committed and respected in CI

UI Framework and Tooling

- UI framework: React

- Build tool: Vite
- Router: React Router
- Data fetching: TanStack Query
- Form handling: React Hook Form
- Schema validation: Zod
- Styling: Tailwind CSS

No alternative UI framework or build tool permitted without architectural approval.

All network access must be centralised through a dedicated API client module.

Authentication

- Identity provider: Auth0
- Flow: Authorization Code with PKCE
- Access tokens stored in memory only
- No localStorage or sessionStorage token storage
- Tokens must never be logged
- Tokens must never be embedded in URLs
- Tokens must never be persisted across full page reloads
- Token refresh must follow Auth0 SDK best practices

On logout:

- In memory tokens must be cleared
- Cached user context must be cleared

The client must treat the server as authoritative for identity and permissions.

API Communication

- Client must call only polite-intervention

- No direct calls to internal services
- Base API URL environment-configurable
- All non 2xx responses treated as errors
- Cursor values treated as opaque
- Cursor values must never be parsed or inspected in the client
- request_id returned by the server must be captured and made available for diagnostics

All HTTP calls must:

- Include Authorization header when authenticated
- Fail closed if token missing for authenticated routes
- Use a shared error handling layer

Internationalisation and Translation Sources

Shared Catalogue Requirement

All browser applications must source shared translation keys from the shared-i18n repository.

Shared-i18n contains:

- Generic error messages
- Authentication messages
- Common UI labels
- Cross-application terminology

Clients must:

- Import shared-i18n as a dependency
- Load shared translations at application bootstrap

- Merge shared translations with app-specific translations
- Not duplicate shared keys locally

App-specific strings may exist locally, but must not redefine shared namespaces such as:

- errors.generic.*
- errors.validation.*
- auth.*
- common.*

The canonical key set is defined in en.

Locale and Formatting Behaviour

Source of Locale

The web client must retrieve effective locale preferences from polite-intervention:

GET /me/preferences

The response provides:

- language
- locale
- timezone

The client must not:

- Derive locale from the browser environment as authoritative
- Use navigator.language as authoritative
- Infer timezone from the system clock as authoritative
- Hardcode formatting rules

Browser defaults may be used only as a temporary fallback before /me/preferences resolves.

Once resolved:

- language, locale, and timezone must be treated as authoritative
- Formatting must update reactively

Storage and Lifetime

- Locale settings must be stored in memory only
- Locale settings must not be persisted in localStorage or sessionStorage
- Locale must be reloaded on full page refresh

Formatting Rules

All formatting of user-facing values must use:

- Intl.DateTimeFormat
- Intl.NumberFormat
- Intl.RelativeTimeFormat when applicable

Manual formatting using string concatenation is prohibited.

Formatting must use:

- locale for numeric and date conventions
- timezone for date/time display

Timezone must not affect business logic.

Timezone is display-only.

Dates and Times

- All timestamps from the API are UTC.
- Conversion to display timezone must occur in the client.
- Never assume server timezone equals user timezone.
- Never embed timezone offsets manually.

- Do not perform business logic calculations using formatted strings.

Numbers and Currency

- Numeric grouping and decimal separators must use Intl.
- Currency display must use Intl with explicit currency code.
- Currency symbols must not be hardcoded.
- Minor unit scaling must be handled based on API contract, not inferred from display formatting.

Error Localisation Rules

Backend services return:

- error.code as stable identifier
- message as non-localised diagnostic text
- message_params optional structured values

The client must:

- Localise error messages using error.code
- Map error.code to translation keys in shared-i18n
- Use message_params for interpolation when provided
- Provide a deterministic fallback for unknown error codes

The client must not:

- Render error.message directly to users
- Branch logic on error.message text
- Depend on English error text

All generic error states must use shared-i18n keys.

Rendering Discipline

Formatting and translation must be centralised.

The client must provide:

- A formatting utility layer or hook
- A translation abstraction that wraps shared-i18n
- A centralised API error mapping layer
- No component may construct date or number formats ad hoc
- No component may embed raw English strings for shared messages
- No component may branch on HTTP status alone without considering error.code

This prevents drift and inconsistent formatting across the app.

Testing Requirements (Internationalisation)

- Unit tests must verify formatting utilities with at least:
  - en-US
  - en-AU
  - de-DE

Tests must verify:

- Date formatting changes with timezone
- Decimal separator changes with locale
- Currency symbol placement changes with locale

E2E tests must verify:

- Switching locale changes visible formatting
- Generic error states render localised messages
- UI does not render backend error.message directly

- Cursor pagination behaves correctly with opaque cursors

WebSocket Usage

- Access token required at handshake
- Connection must close when token expires
- Reconnect requires fresh token
- No client-side permission inference from events
- Server remains authoritative
- Real time events must not be treated as authoritative domain data without HTTP confirmation

Security

- No secrets in client source or build artifacts
- No machine-to-machine credentials in browser
- CSP must be defined
- Dependency audit required
- No permissive debug endpoints in production
- No embedding of internal service URLs
- No client-side permission caching beyond UI convenience state

Observability

- Client error reporting required
- Redact tokens and sensitive data
- If request_id header returned, include in client logs and error reports
- Correlate client errors with server request_id when available

- Do not log raw Authorization headers

Testing

- Unit tests using Vitest
- UI tests using Testing Library
- E2E tests using Playwright
- No mocking of internal services
- Only polite-intervention endpoints may be mocked
- Security tests must include:
  - Token expiry handling
  - Unauthenticated route access
  - Error envelope handling
  - Locale fallback behaviour

Prohibited Patterns

- Storing tokens in persistent storage
- Embedding internal URLs
- Client-side permission caching as source of truth
- Console logging of sensitive data
- Rendering backend error.message directly to users
- Hardcoding shared generic messages in English
- Parsing or constructing cursor values in the client