***Amicable settlement system (Weddi portal)***

***Code Document***

***version 1***

**TABLE OF CONTENTS**

[● EXECUTIVE SUMMARY & PROJECT OVERVIEW 1](#_heading=)

[DOCUMENT SCOPE 1](#_heading=)

[INTENDED AUDIENCE: 1](#_heading=)

[DOCUMENT STRUCTURE: 1](#_heading=)

[PURPOSE 2](#_heading=h.e1nurpnqd0a5)

[PRODUCT CONTEXT: 2](#_heading=h.ntylw9371hzq)

[TECHNICAL VALUE: 2](#_heading=h.2xietki1qlu6)

[● PURPOSE & SCOPE 4](#_heading=h.i0vvcg5jwb62)

[SCOPE (WHAT THIS HANDBOOK COVERS): 4](#_heading=)

[OUT-OF-SCOPE (EXPLICIT EXCLUSIONS): 4](#_heading=)

[PRIMARY AUDIENCES & GUIDANCE: 4](#_heading=)

[TYPICAL USAGE SCENARIOS: 5](#_heading=h.npvhzgjwe9j2)

[● SYSTEM ARCHITECTURE (COMPREHENSIVE) 6](#_heading=)

[APPLICATION ARCHITECTURE OVERVIEW 6](#_heading=h.woqfbj20gz29)

[STATE MANAGEMENT & DATA FLOW LAYER 6](#_heading=h.zdjr0rhvp53e)

[DATA ACCESS & API ABSTRACTION 6](#_heading=h.wf0zwhdu9ial)

[PRESENTATION LAYER & THEMING SYSTEM 7](#_heading=h.qz55t3frf822)

[ٍBUILD & DEPLOYMENT PIPELINE 7](#_heading=h.b29ufuify8ii)

[RUNTIME INTERACTION DIAGRAM 7](#_heading=h.rqrk55oyre5)

[● FOLDER & FILE STRUCTURE (DEEP INSPECTION) 10](#_heading=h.uo9y8vdfunxx)

[OVERVIEW 10](#_heading=h.ngvsklow6dtw)

[● TECHNOLOGY STACK & DEPENDENCIES 14](#_heading=h.etcassqta20w)

[OVERVIEW 14](#_heading=h.yuvldnv0tj9n)

[● ENVIRONMENT & SYSTEM REQUIREMENTS 16](#_heading=h.39m64ih3ot7g)

[NODE.JS & PACKAGE MANAGER 16](#_heading=h.rptbd5apigjz)

[BROWSER SUPPORT 16](#_heading=h.ng5wxlo33yy6)

[CRITICAL ENVIRONMENT VARIABLES 16](#_heading=h.1p3fst4kfdio)

[MANAGING SECRETS & .ENV FILES 17](#_heading=h.l3o5kflak9m)

[ADDITIONAL SYSTEM REQUIREMENTS & PRACTICAL NOTES 17](#_heading=h.jph2l863477l)

[● CODE STRUCTURE & COMPONENT HIERARCHY 18](#_heading=h.mmyjms3lrbka)

[INDEX.TSX 18](#_heading=h.nk4josuyfti)

[APP.TSX 18](#_heading=h.sfs4v3gifswt)

[MAINLAYOUT.TSX 18](#_heading=h.syjtn37uvhpj)

[OVERVIEW DIAGRAM 19](#_heading=h.sklr6cjva8ye)

[NAMING CONVENTIONS & MODULARITY 21](#_heading=h.wmvimamg3qe)

[● STYLING AND THEMING 22](#_heading=h.e945yvai8d68)

[● ROUTING AND NAVIGATION 23](#_heading=h.ur21ezs2od7d)

[OVERVIEW 23](#_heading=h.5f0exarx1zmy)

[● DATA MANAGEMENT AND API INTEGRATION 25](#_heading=h.uhisaphp8h2)

[● TYPESCRIPT USAGE AND TYPE SAFETY 26](#_heading=h.uauwfckky227)

[PURPOSE. 26](#_heading=h.8c80b83tcmbs)

[CONFIGURATION 26](#_heading=h.8c80b83tcmbs)

[RUNTIME GUARDS 26](#_heading=h.8c80b83tcmbs)

[UTILITY ENFORCEMENT 26](#_heading=h.8c80b83tcmbs)

[SUMMARY / KEY INSIGHT. 26](#_heading=h.8c80b83tcmbs)

[● LINTING, FORMATTING & QUALITY GATES 27](#_heading=h.11n7dxa151dk)

[LINTING 27](#_heading=h.8c80b83tcmbs)

[FORMATTING 27](#_heading=h.8c80b83tcmbs)

[CONTINUOUS INTEGRATION 27](#_heading=h.8c80b83tcmbs)

[SUMMARY / KEY INSIGHT: 27](#_heading=h.8c80b83tcmbs)

[● TESTING FRAMEWORK & COVERAGE 28](#_heading=h.9x3i154k4cwg)

[PURPOSE 28](#_heading=h.8c80b83tcmbs)

[CURRENT STATE 28](#_heading=h.8c80b83tcmbs)

[FUTURE DIRECTION 28](#_heading=h.8c80b83tcmbs)

[SUMMARY / KEY INSIGHT: 28](#_heading=h.8c80b83tcmbs)

[● BUILD AND DEPLOYMENT PIPELINE 29](#_heading=h.9bxibp4r5dvf)

[PURPOSE. 29](#_heading=h.6dxmjl4mz6m8)

[BUILD PROCESS 29](#_heading=h.6dxmjl4mz6m8)

[DEPLOYMENT WORKFLOW 29](#_heading=h.6dxmjl4mz6m8)

[SUMMARY / KEY INSIGHT: 30](#_heading=h.6dxmjl4mz6m8)

[● TROUBLESHOOTING / FAQ 31](#_heading=h.af7ncmurdito)

[PURPOSE. 31](#_heading=h.d757g5l5jdve)

[● SECURITY AND PERFORMANCE NOTES 34](#_heading=h.gdrwp4sokb2l)

[SECURITY 34](#_heading=h.xsge6kkatg87)

[PERFORMANCE 34](#_heading=h.xsge6kkatg87)

[ACCESSIBILITY AND RESILIENCE 35](#_heading=h.xsge6kkatg87)

[SUMMARY / KEY INSIGHT: 35](#_heading=h.xsge6kkatg87)

[● CONTRIBUTION GUIDELINES 36](#_heading=h.62l92dpdcaw1)

[PURPOSE. 36](#_heading=h.xsge6kkatg87)

[COMMIT AND CODE STYLE 36](#_heading=h.xsge6kkatg87)

[REVIEW STANDARDS 36](#_heading=h.xsge6kkatg87)

[SUMMARY / KEY INSIGHT: 37](#_heading=h.xsge6kkatg87)

[● CHANGE LOG & VERSION HISTORY 38](#_heading=h.4qr8iv40enbn)

[PURPOSE: 38](#_heading=h.noes2khfewlj)

[SUMMARY / KEY INSIGHT: 38](#_heading=h.1l83cct9q7e4)

[● FUTURE IMPROVEMENTS AND NEXT STEPS 39](#_heading=h.9lvw38d34a3d)

[PURPOSE: 39](#_heading=h.ilgjpnns51c1)

[SUMMARY / KEY INSIGHT: 39](#_heading=h.famjisvwfdn2)

[● APPENDICES 40](#_heading=h.ns3dzmt8vnf2)

[PURPOSE: 40](#_heading=h.8bkjkqh2i2lh)

[GLOSSARY 40](#_heading=h.8bkjkqh2i2lh)

[EXTERNAL REFERENCES 40](#_heading=h.8bkjkqh2i2lh)

[SUMMARY / KEY INSIGHT 40](#_heading=h.8bkjkqh2i2lh)

[● FINAL NOTES AND RECOMMENDATIONS FOR NEW DEVELOPERS 41](#_heading=h.mogld4jdwbiz)

[PURPOSE: 41](#_heading=h.8bkjkqh2i2lh)

[START WITH PROVIDER COMPOSITION: 41](#_heading=h.8bkjkqh2i2lh)

[TRACE PERSONA DATA END-TO-END: 41](#_heading=h.8bkjkqh2i2lh)

[REVIEW SHARED COMPONENTS AND HOOKS: 41](#_heading=h.8bkjkqh2i2lh)

[BE MINDFUL OF ENVIRONMENT DEPENDENCIES: 42](#_heading=h.7ww8qru49uzy)

[COORDINATE WITH QA EARLY: 42](#_heading=h.8bkjkqh2i2lh)

[SUMMARY / KEY INSIGHT 42](#_heading=h.imt90wislj42)

# EXECUTIVE SUMMARY & PROJECT OVERVIEW

*This technical documentation provides a comprehensive overview of the Weddi Frontend – Clean Edition, covering its architectural structure, design rationale, and implementation strategy.*

*It is intended for developers, QA engineers, DevOps teams, and technical stakeholders involved in maintaining, extending, or integrating the Weddi frontend system.*

*The goal is to establish a clear understanding of how different layers of the application — from providers and routing to state management and localization — interact to deliver a reliable and scalable user experience.*

## DOCUMENT SCOPE

*The documentation focuses exclusively on the frontend architecture of the Weddi platform. It details:*

* *Application bootstrapping and provider composition.*
* *Routing, state orchestration, and localization strategy.*
* *Feature modularity and reusable UI patterns.*
* *API communication, authentication flows, and lifecycle management.*
* *Development conventions and scalability considerations.*
* *Each section builds upon the previous one — starting with a high-level executive overview, followed by deeper technical analysis of individual modules and their design dependencies.*

## INTENDED AUDIENCE:

* *Frontend Developers: For onboarding, code navigation, and feature implementation guidelines.*
* *Tech Leads / Architects: For maintaining architectural integrity and evolution planning.*
* *QA & Test Engineers: To understand expected behavior and state transitions across flows.*
* *DevOps / CI Engineers: For environment setup, deployment, and runtime configuration alignment.*

## DOCUMENT STRUCTURE:

*The Weddi Portal is designed with the following objectives:*

* *Executive Summary & Project Overview – Strategic and architectural context.*
* *Scope & Audience – Boundaries and reader expectations.*
* *Architecture Overview – Core structure and component relationships.*
* *Feature Modules – Functional areas and reusable elements.*
* *Integration & API Layer – Backend communication and data flow.*
* *Error Handling & Performance – Resilience and optimization patterns.*
* *Development Workflow & Guidelines – Conventions and best practices.*

## PURPOSE

*Define the strategic and technical foundation for the Weddi Frontend – Clean Edition, highlighting architectural priorities, business objectives, and engineering rationale behind major design choices.*

*The Weddi Frontend is a modular React Single-Page Application (SPA) initialized via index.tsx, composed of layered providers, and rendered through a routing tree in App.tsx.*

*This bootstrapping flow ensures that localization, token lifecycle monitoring, global state management (Redux), and shared cross-feature utilities are activated at the earliest render stage — forming a consistent and reliable environment for all routes.*

*【F:src/index.tsx】【F:src/app/App.tsx】【F:src/providers/AppProvider.tsx】*

## PRODUCT CONTEXT:

*The portal is built for the Saudi Ministry of Human Resources and Social Development (HRSD) under the Amicable Settlement (Weddi) program.*

*The Clean Edition streamlines the interface for major personas — workers, establishments, legal representatives, and embassy delegates — through refined dashboards, guided hearing initiation flows, and hearing management tools.*

*The refactored provider and routing lattice centralizes shared services, minimizes domain coupling, and supports fast onboarding for new developers, while maintaining regulatory compliance and localization fidelity through strongly typed configuration and context models.*

*【F:src/features/dashboard/index.tsx】*

## TECHNICAL VALUE:

*This document is intended for:*

* *Predictable state orchestration using Redux Toolkit and RTK Query.*
* *Dynamic module loading enabling scalable, isolated feature delivery.*
* *Custom error boundaries that maintain application resilience during API or routing failures.*

*These patterns ensure flexible feature evolution while preserving robustness in hearing submission workflows, persona switching, and analytics dashboards.*

*The modular architecture — structured around providers, hooks, and feature directories — follows principles outlined in tech-design.md, enabling new modules to integrate seamlessly without destabilizing the core system.*

*【F:src/app/store/index.ts】【F:src/services/apiClient.ts】*

***SUMMARY***

*The Weddi Frontend – Clean Edition modernizes the platform into a modular, provider-driven React SPA designed for multi-persona usability, scalability, and regulatory alignment.*

*Its architecture emphasizes separation of concerns, predictable state flows, and high maintainability — allowing development teams to extend and evolve the system confidently while meeting HRSD’s compliance and localization standards.*

# PURPOSE & SCOPE

*Clarify the handbook’s boundaries, identify its primary stakeholders, and list the practical scenarios in which engineers and operators should consult this documentation.*

## SCOPE (WHAT THIS HANDBOOK COVERS):

*This documentation targets the entire frontend codebase and its immediate build configuration. It documents architectural reasoning, module responsibilities, environment setup, and operational guardrails. Specifically, the handbook describes the /src tree, build scripts, and configuration assets, such as vite.config.ts, tailwind.config.js, and TypeScript compiler settings — including how they impact dev, CI, and production builds.【F:vite.config.ts】【F:tailwind.config.js】*

## OUT-OF-SCOPE (EXPLICIT EXCLUSIONS):

*To keep the handbook focused and maintainable, the following are intentionally excluded: backend API design beyond the surface-level REST interactions consumed by the frontend, infrastructure-as-code manifests stored outside this repository, identity provider internals, third-party analytics control planes, and HRSD-owned legal/process documents. When referenced, these artifacts are contextualized only from the frontend’s consumption perspective.*

| ***Included Deliverables*** | ***Excluded Assets*** |
| --- | --- |
| *Frontend architecture, routing models, provider composition, state orchestration, styling conventions, Vite build behavior, troubleshooting heuristics, and collaboration workflows.* | *Backend service specifications (beyond observed REST/graph interactions), external IaC manifests, identity provider internals, analytics platform control, and legal/process manuals owned by HRSD.* |

## PRIMARY AUDIENCES & GUIDANCE:

* *Frontend Engineers: Implementation patterns, state breakdowns, and module responsibilities for feature development and refactors.*
* *Tech Leads / Architects: Architectural rationale, integration boundaries, and evolution guidelines.*
* *QA / Test Engineers: Route inventories, state transition expectations, localization and edge-case behavior to inform regression suites.*
* *DevOps / CI Engineers: Build, environment, and runtime configuration details required for pipeline integration and deployment validation.*
* *Each audience is directed to relevant sections (e.g., state deep-dives for engineers, route inventories for QA, build/deploy instructions for DevOps).*

## TYPICAL USAGE SCENARIOS:

* *Onboarding: New engineers review provider layering and routing before modifying feature code.*
* *Release Review: Product and engineering teams verify that new endpoints and payload shapes preserve existing contracts and error handling.*
* *Incident Triage & Retrospective: Teams trace token-refresh flows, cookie synchronization, and API client behavior to diagnose production issues.*
* *Feature Design: Architects and feature owners consult modular boundaries to decide where new functionality should live and how it should be exposed.*

*Start with the high-level executive overview (Section 1) to align on business and architectural goals. For implementation work, refer to the Architecture Overview and Feature Modules sections. For environment and deployment concerns, proceed directly to the Development Workflow & Guidelines and Integration & API Layer sections.*

***SUMMARY***

*This handbook documents the frontend’s runtime composition and developer-facing contracts while deliberately excluding backend internals and external infrastructure artifacts. By setting these boundaries up front, the document remains a practical, audience-focused reference: concise for onboarding and authoritative for release reviews and incident analysis.*

# SYSTEM ARCHITECTURE (COMPREHENSIVE)

*Provide a rigorous analysis of the architectural layers, runtime interactions, and build/deployment flow that together define the Weddi Frontend runtime.*

## APPLICATION ARCHITECTURE OVERVIEW

*The Weddi Frontend functions as a client-rendered React Single-Page Application (SPA).*

*At its core, App.tsx defines a hierarchical router tree nested under MainLayout, while AppProvider composes the provider stack responsible for Redux state, error boundaries, cookie context, and language direction awareness—all initialized before the router mounts.*

*React’s Suspense and route-level lazy() imports optimize first-paint performance and split bundle costs while ensuring that cross-cutting services such as toast notifications, token-expiry handling, and offline detection remain globally available.*

*【F:src/app/App.tsx】【F:src/providers/AppProvider.tsx】*

## STATE MANAGEMENT & DATA FLOW LAYER

*At the infrastructure layer, Redux Toolkit (RTK) forms the canonical store.*

*It integrates loading indicators, form states, cached options, and default values alongside the RTK Query API slice defined in apiClient.ts.*

*Middleware coordinates deterministic async flows:*

* *Dispatching loading signals automatically.*
* *Retrying operations upon authentication expiry.*
* *Centralizing API error handling.*

*This deterministic orchestration ensures that multi-step case creation wizards, persona-based dashboards, and cross-tab synchronization remain consistent without prop drilling or local side effects.*

*【F:src/app/store/index.ts】【F:src/services/apiClient.ts】*

## DATA ACCESS & API ABSTRACTION

*The custom RTK Query base query injects authentication and persona metadata into every request, normalizes errors via toast-based notifications, and encapsulates both legacy and OAuth token management.*

*Feature teams remain fully decoupled from header injection, token refresh logic, and error serialization—they simply define endpoints via api.injectEndpoints() and inherit unified resilience and telemetry behavior.*

*When combined with cookie-aware providers and session hooks, this design maintains session persistence across reloads, tabs, and persona switches without explicit coordination.*

*【F:src/services/apiClient.ts†L115-L240】*

*【F:src/services/apiClient.ts†L240-L382.】*

## PRESENTATION LAYER & THEMING SYSTEM

*The presentation tier is powered by Tailwind CSS, configured with an extended theme supporting RTL-first layouts, granular spacing, typography scales, and safelisted dynamic classes up to 8K viewport widths.*

*The styling foundation is defined in tailwind.config.js, while component-level styles reside in modular directories within /src/assets/styles/.*

*This structure enables rapid iteration for designers while enforcing bilingual layout consistency and design-system reusability.*

*【F:tailwind.config.js†L1-L190】【F:src/assets/styles/index.css†L1-L140】*

## ٍBUILD & DEPLOYMENT PIPELINE

*The build-to-deploy flow is orchestrated through Vite, configured in vite.config.ts.*

*During the build process:*

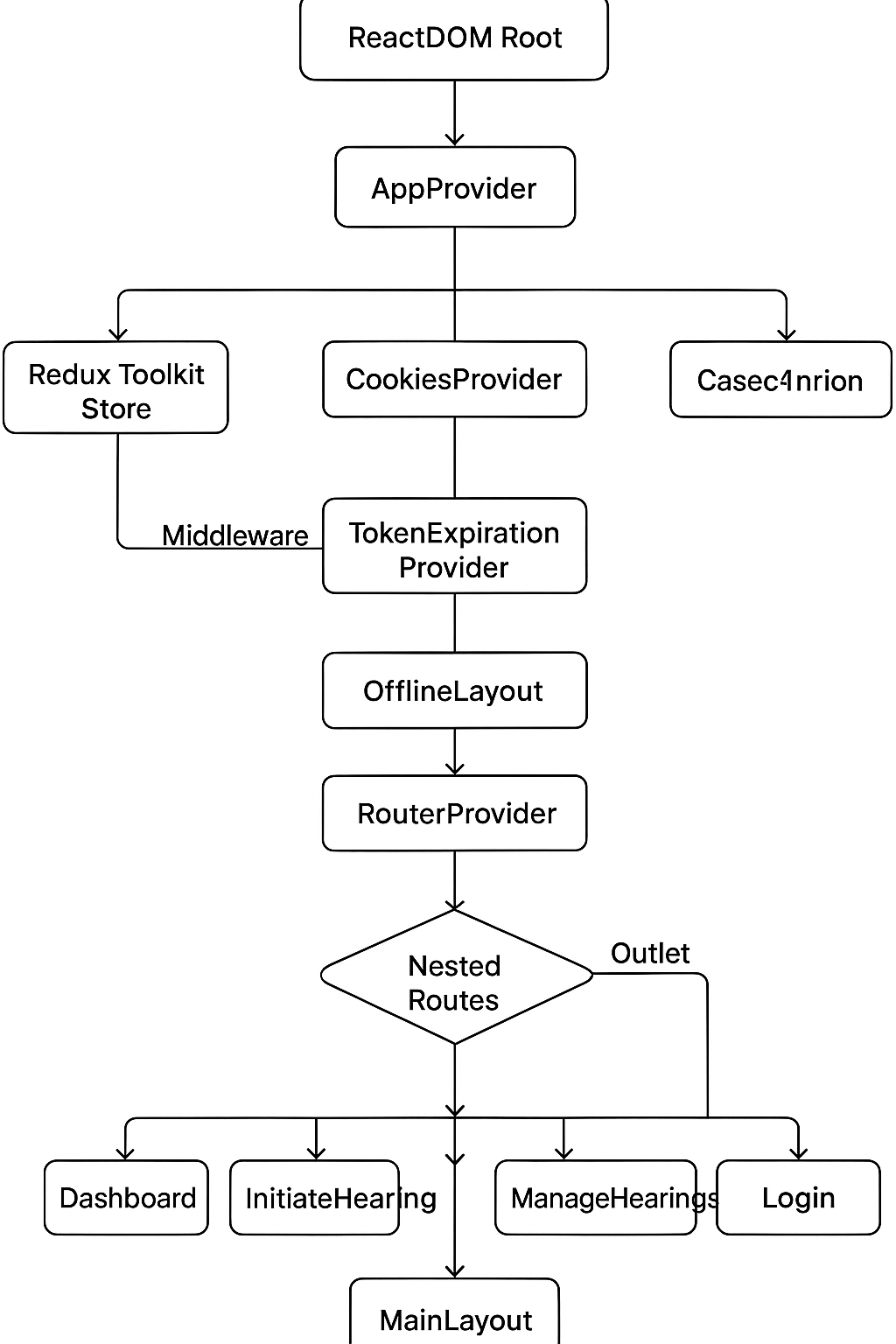
* *TypeScript undergoes a full compilation pass before bundling.*
* *React and static-copy plugins handle JSX transformation and asset relocation.*
* *Manual vendor chunking stabilizes CDN caching.*
* *A /portal/ base path ensures compatibility with subdirectory deployments.*

*Resulting bundles colocate hashed JavaScript, fonts, and locales under the /dist directory—ensuring consistent asset resolution across environments.*

*【F:package.json】【F:vite.config.ts】*

## 

## *RUNTIME INTERACTION DIAGRAM*

**

***SUMMARY***

*The Weddi Frontend architecture fuses provider layering, deterministic state management, and build-time optimization to form a stable, scalable SPA foundation.*

*By unifying these layers—providers, Redux configuration, and Vite-based build tooling—the system achieves predictable behavior across personas, devices, and environments.*

*This understanding sets the stage for the next section, where the filesystem layout illustrates how folder organization reinforces the architectural model and isolates feature domains without breaking shared infrastructure.*

# FOLDER & FILE STRUCTURE (DEEP INSPECTION)

*Purpose of this section: Explain how the repository’s physical layout represents the architectural principles introduced earlier. Each folder and file contributes a distinct role in enforcing modularity, scalability, and maintainability across the application.*

## OVERVIEW

*The source tree is intentionally structured to reflect functional domains rather than technical layers. Each feature module is self-contained—bundling its UI, hooks, and API logic—while shared directories provide reusable primitives and infrastructure support. This design ensures that contributors can navigate from high-level concepts to concrete implementations with minimal cognitive overhead.*

*System configuration files (such as vite.config.ts, tailwind.config.js, and tsconfig.json) complement the application code by codifying build, style, and typing rules—aligning the local developer environment with the production runtime.*

| **Path** | **Purpose** | **Notes** |
| --- | --- | --- |
| src/app/App.tsx | Defines the browser router, lazy module boundaries, error fallbacks, token expiration envelope, and toast container. | Centralizes the /portal/ basename, ensures each feature is wrapped in Suspense, and mounts shared loaders so asynchronous activity remains visible across screens.【F:src/app/App.tsx†L37-L136】 |
| src/app/store/index.ts | Bootstraps the Redux store, integrates the RTK Query API reducer, and wires middleware for loading counters. | Provides unified reducer composition, typed exports, and DevTools labeling for transparent state debugging.【F:src/app/store/index.ts†L1-L39】 |
| src/assets/styles/index.css & fonts.scss | Deliver global styling, scrollbar overrides, skeleton animations, and IBM Plex Arabic font imports. | Loaded at the entry point to enforce consistent typography, spacing, and motion behavior across browsers.【F:src/index.tsx†L3-L13】【F:src/assets/styles/index.css†L1-L140】 |
| src/config/app.config.ts & formConfig.tsx | Define application constants and reusable form builders for the hearing wizard. | Provide centralized metadata (APP\_TITLE, masks, validators) to ensure UI consistency and reduce redundancy across steps.【F:src/config/app.config.ts†L1-L31】【F:src/config/formConfig.tsx†L1-L125】 |
| src/features/auth | Implements authentication workflows, persona selection, token validation, and cookie synchronization. | AuthProvider manages token decoding and persona hydration, while LogOut.tsx clears session data to guarantee secure transitions.【F:src/features/auth/components/AuthProvider.tsx†L1-L253】【F:src/features/auth/LogOut.tsx†L1-L24】 |
| src/features/dashboard | Provides authenticated landing views, role-based widgets, and quick navigation to hearing modules. | Suspense wrappers and hooks ensure fast rendering while adapting data per user persona.【F:src/features/dashboard/index.tsx†L1-L75】 |
| src/features/hearings/initiate | Manages the multi-step hearing initiation wizard and associated hooks. | Includes useCookieState for persistence and typed payloads for case creation integrated with persona metadata.【F:src/features/hearings/initiate/hooks/useCookieState.ts†L1-L124】【F:src/features/hearings/initiate/modules/case-creation/index.tsx†L1-L55】 |
| src/features/hearings/manage | Supports search, pagination, and post-submission topic updates for hearings. | Lazy-loaded to optimize bundle size while preserving consistent layout and toast behavior.【F:src/app/App.tsx†L24-L98】 |
| src/i18n | Controls runtime localization and document direction. | LanguageDirectionProvider toggles direction and provides hooks, while i18n.tsx dynamically loads translation resources.【F:src/i18n/LanguageDirectionProvider.tsx†L1-L60】 |
| src/providers | Central hub for all global providers (Redux, cookies, date contexts, tabs, form controllers, etc.). | AppProvider composes them hierarchically, while TokenExpirationProvider monitors session expiry and notifies users via toasts.【F:src/providers/AppProvider.tsx†L1-L41】【F:src/providers/TokenExpirationProvider.tsx†L1-L75】 |
| src/services/apiClient.ts | Core RTK Query configuration handling authentication, retries, and normalization. | Consolidates OAuth refresh, token propagation, persona injection, and error normalization, ensuring consistent API behavior.【F:src/services/apiClient.ts†L20-L399】 |
| src/shared/components | Houses UI primitives (buttons, loaders, modals, tables, tabs). | Shared components follow Tailwind conventions and provide consistent fallback/error interfaces.【F:src/shared/components/loader/index.tsx†L1-L28】【F:src/shared/components/errors/ErrorFallback.tsx†L1-L38】 |
| src/shared/hooks | Contains reusable logic for cookie handling, navigation, and keyboard guards. | Examples include useClearCaseData, which resets wizard state across cookies and Redux contexts.【F:src/shared/hooks/useClearCaseData.ts†L1-L55】 |
| src/shared/layouts | Defines structural shells for authenticated content and offline messaging. | MainLayout orchestrates modals, persona context, and nested routes.【F:src/shared/layouts/MainLayout.tsx†L1-L145】 |
| src/utils | Offers helper functions for API normalization, date conversion, and environment validation. | Ensures strict variable checks and type-safe transformations to maintain runtime stability.【F:src/utils/helpers.ts†L1-L104】【F:src/utils/api/errorHandler.ts†L1-L120】 |
| tailwind.config.js | Customizes Tailwind theme, breakpoints, safelist patterns, and font settings. | Supports high-resolution layouts and enforces IBM Plex Arabic as the primary font.【F:tailwind.config.js†L1-L190】 |
| vite.config.ts | Defines Vite’s build pipeline, plugins, environment variables, and alias mapping. | Enables faster builds and consistent import resolution (@ → src).【F:vite.config.ts†L1-L63】 |
| tsconfig.json & tsconfig.paths.json | TypeScript compiler and alias configurations. | Enforces strict mode and JSX settings aligned with project-wide typing strategy.【F:tsconfig.json†L1-L25】【F:tsconfig.paths.json†L1-L8】 |
| eslint.config.js | Linting rules for JS/TS/React with flat config syntax. | Extends recommended presets and integrates plugins for unused imports and Node-aware overrides.【F:eslint.config.js†L1-L44】 |

***SUMMARY***

*The repository’s structure is a living representation of its conceptual model. By mapping domain boundaries directly to folder organization, the project enforces clear ownership, facilitates parallel development, and simplifies debugging. This coherence between physical layout and architectural intent sets the stage for the next section: Technology Stack & Dependency Mapping.*

# TECHNOLOGY STACK & DEPENDENCIES

*Purpose of this section: Identify the foundational technologies, libraries, and build tools that enable the Weddi Frontend to function reliably and scale efficiently.*

## OVERVIEW

*The Weddi Frontend leverages a modern, modular ecosystem optimized for maintainability, performance, and multilingual support. Each dependency is intentionally selected to balance developer productivity, runtime efficiency, and long-term extensibility.*

*The core runtime is built on React 18 (react, react-dom) combined with React Router v6 for navigation, Redux Toolkit for predictable state management, and RTK Query for declarative data fetching. React Hook Form governs form state handling, enabling components to stay declarative while asynchronous data flows remain centrally orchestrated. This integration ensures that persona-driven workflows reuse logic consistently across features.*

*【F:package.json†L12-L48】*

*At the build layer, Vite 6.2.6 serves as the primary toolchain—augmented by @vitejs/plugin-react and its SWC variant to achieve near-instant Hot Module Replacement and optimized production bundling. Styling is managed via Tailwind CSS 3.4.x, assisted by tailwind-merge and tailwindcss-animate, while PostCSS and Autoprefixer enforce cross-browser consistency. The styling system fully supports RTL layouts and responsive scaling, ensuring that visual fidelity is maintained across device classes.*

*【F:package.json†L12-L76】【F:*[*tailwind.config.js*](http://tailwind.config.js)*†L1-L190】*

*For internationalization (i18n), the project employs i18next with react-i18next bindings, dynamically loading resources using i18next-resources-to-backend. Date logic combines react-date-object, react-multi-date-picker, and custom Hijri/Gregorian conversion utilities, ensuring culturally accurate date handling. UI accessibility and design consistency are reinforced using Headless UI, Radix UI, Heroicons, and Lucide, minimizing bespoke widget code while maintaining semantic structure.*

*【F:src/utils/helpers.ts†L1-L104】*

*Supporting dependencies further enhance functionality:*

* *Axios for specialized HTTP calls (legacy or token-specific cases).*
* *.Jose and jwt-decode for secure JWT handling during authentication.*
* *react-toastify for uniform notification delivery.*
* *react-error-boundary for fault isolation and graceful recovery.*

*Development utilities—ESLint 9, TypeScript 5.7.3, Tailwind CLI, and Vite—form a static analysis and build layer ensuring consistent linting, strict type safety, and reproducible environments.*

*【F: package.json†L12-L76】*

***SUMMARY***

*The chosen stack is not arbitrary—it directly mirrors the architectural goals of the system. React and RTK Query define predictable data flows; Vite and Tailwind provide rapid iteration and visual coherence; i18next and date libraries localize the experience. Together, these dependencies unify the development and runtime experience, preparing the ground for the next section: Environment Configuration & Setup.*

# ENVIRONMENT & SYSTEM REQUIREMENTS

*Purpose of this section: Define the runtime prerequisites, environment variables, and configuration nuances that must be satisfied before the Weddi Frontend can run reliably across development, staging, and production environments..*

## NODE.JS & PACKAGE MANAGER

* *Recommended: Node.js 18 LTS or later to support Vite 6.x, TypeScript 5.7, and modern ES module features.*
* *Package Manager: The project is optimized for npm (the default with Node). Although alternatives like pnpm or yarn can technically work, the lockfile and build scripts are designed for npm workflows.【F:package.json†L1-L14】*

## BROWSER SUPPORT

* *Target evergreen browsers that support ES2020 modules, CSS variables, and flexbox layouts.*
* *QA teams should focus on Chrome, Edge, and Firefox (current LTS versions) to validate:*
  + *Proper RTL direction handling*
  + *Responsive design on large displays per Tailwind’s extended breakpoints.【F:tailwind.config.js†L1-L190】】*

## CRITICAL ENVIRONMENT VARIABLES

*Environment variables are read via process.env during build/runtime and import.meta.env in the browser.*

*Core Variables Include:*

* *VITE\_API\_URL*
  + *Base URL for Weddi Services APIs used by RTK Query’s fetchBaseQuery.*
  + *Defined in apiClient.ts for request routing.【F:src/services/apiClient.ts†L30-L165】*
* *VITE\_OAUTH\_CLIENT\_ID, VITE\_OAUTH\_CLIENT\_SECRET, VITE\_OAUTH\_GRANT\_TYPE*
  + *OAuth credentials used to mint bearer tokens.*
  + *Must be stored securely and never committed to the repository.【F:src/services/apiClient.ts†L20-L83】*
* *VITE\_REDIRECT\_URL, VITE\_REDIRECT\_URL\_LOCAL, VITE\_LOGIN\_SWITCH*
  + *Control redirect flows and session handling within the TokenExpirationProvider.【F:src/providers/TokenExpirationProvider.tsx†L14-L55】*
* *Additional Secrets:*
  + *Any extra configuration values referenced in helpers must be provided via .env files or deployment environment settings. The getEnvVar utility enforces presence checks at runtime to avoid silent misconfigurations.【F:src/utils/helpers.ts†L92-L104】*

## MANAGING SECRETS & .ENV FILES

* *Local Development: use .env.local (excluded from version control).*
* *CI/CD or Production: inject secrets securely using the deployment platform (e.g., GitHub Actions Secrets, Vercel/Netlify environment variables).*
* *Never commit sensitive data or client secrets into the repository*

## ADDITIONAL SYSTEM REQUIREMENTS & PRACTICAL NOTES

* *Memory/CPU: Standard developer hardware is sufficient, but CI builds benefit from more RAM/cores.*
* *Network Access: Required to reach VITE\_API\_URL and the external identity provider endpoints during local and integration testing.*
* *Time Zone & Locale: Some modules rely on Hijri/Gregorian conversions — ensure consistency across environments when testing date behavior.*
* *Caching & CDN: The vite.config.ts file defines the base path (/portal/) and chunking strategy for CDN compatibility; verify configurations before deployment.【F:vite.config.ts†L1-L63】*

***SUMMARY***

*A compliant runtime environment—Node 18+, correctly injected environment variables, and support for modern browsers—is essential for stable development and deployment. Once these prerequisites are satisfied, contributors can run the app locally, validate integration with Weddi services, and produce consistent, reproducible builds..*

# CODE STRUCTURE & COMPONENT HIERARCHY

*Purpose of this section: Explain how components compose, how props and context flow through the tree, and why this structure supports modular feature development.*

## 

## INDEX.TSX

*The application’s rendering sequence begins in index.tsx, which mounts AppProvider and App inside React.StrictMode.*

*AppProvider layers all critical contexts — including Redux, error boundaries, cookies, tabs, date providers, form controllers, and language direction — before any UI component mounts.*

*This guarantees that every descendant component can safely access typed hooks (useAppDispatch, useLanguageDirection, useFormContext) rather than relying on prop drilling or duplicate state containers.*

*【F:src/index.tsx†L1-L15】【F:src/providers/AppProvider.tsx†L1-L41】.*

## APP.TSX

*App.tsx defines the primary routing graph using React Router 6, with lazy-loaded modules via React.lazy and Suspense.*

*Each route is wrapped by LazyLoader and paired with a shared Loader overlay to maintain consistent async feedback.*

*Feature-specific skeletons (like StepperSkeleton) enhance perceived responsiveness during data loads.*

*The RouterProvider is wrapped within OfflineLayout, ensuring that offline banners, toasts, and connectivity-aware UX remain globally available.*

*【F:src/app/App.tsx†L19-L139】*

## MAINLAYOUT.TSX

*MainLayout forms the bridge between providers and routed content.*

*It hydrates persona state from cookies, exposes user type context setters, and renders structural UI elements (headers, modals, etc.) around a React Router Outlet.*

*Child components (e.g., dashboard banners, hearing modules) consume persona flags and tokens through context hooks, ensuring consistent behavior even when the user’s active role changes mid-session.*

*【F:src/shared/layouts/MainLayout.tsx†L1-L145】*

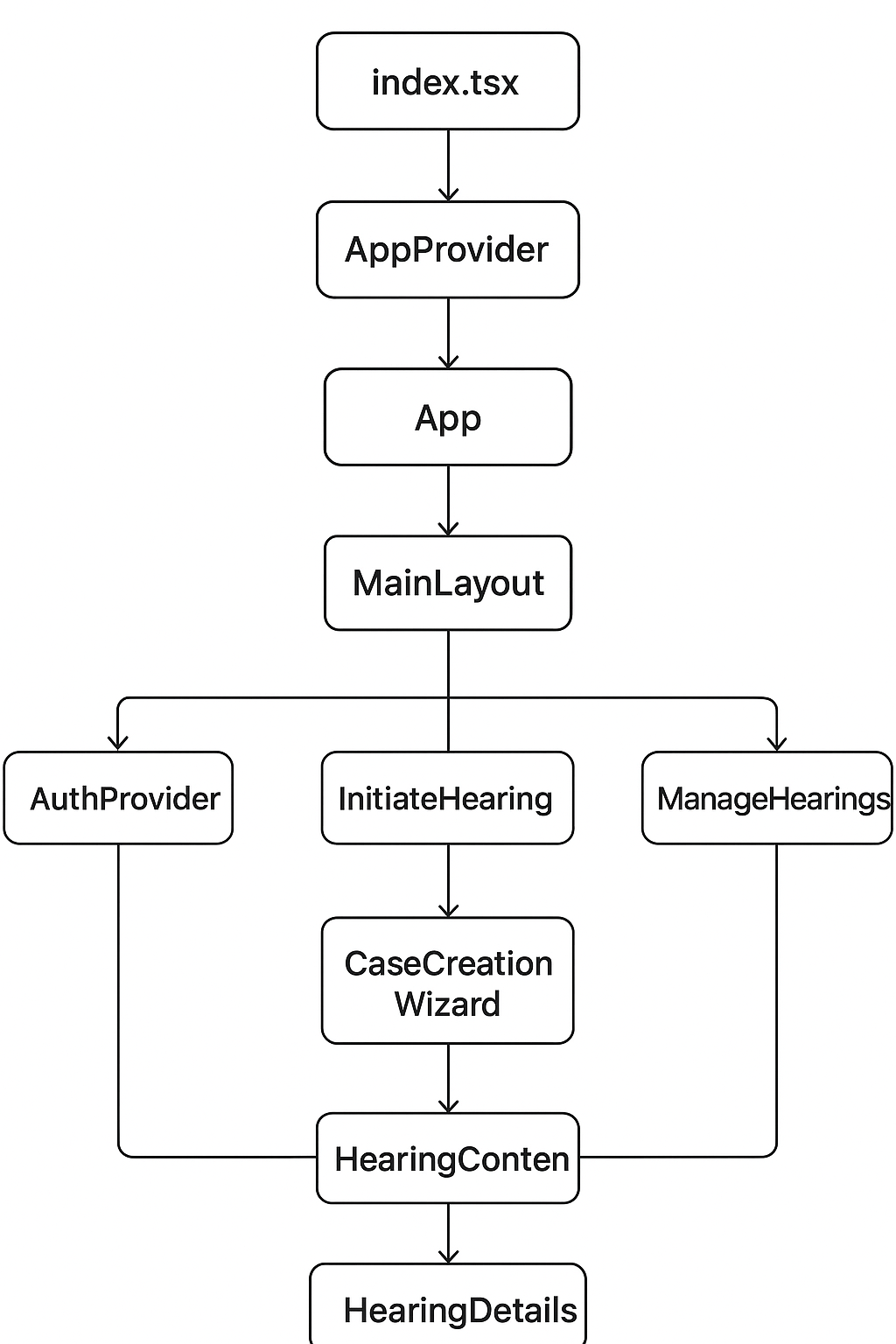
*【F:src/providers/context/userTypeContext.tsx†L37-L82】*

## OVERVIEW DIAGRAM

*Props generally flow top-down for presentation logic, while domain state propagates through contexts and Redux slices.*

*For instance:*

* *The dashboard reads persona context to conditionally render widgets.*
* *The AuthProvider writes updates when persona lookups or NIC validations resolve.*
* *The hearing initiation wizard integrates React Hook Form, cookie synchronization hooks, and Redux slices to maintain form state across steps—even after reloads.*
  + *【F:src/features/dashboard/index.tsx†L1-L75】*
  + *【F:src/features/auth/components/AuthProvider.tsx†L121-L253】*
  + *【F:src/features/hearings/initiate/hooks/useCookieState.ts†L1-L124】*

**

## *NAMING CONVENTIONS & MODULARITY*

* *Components: PascalCase (MainLayout.tsx)*
* *Hooks: camelCase (useClearCaseData.ts)*
* *Features: domain-based folders (src/features/<domain>)*
* *Shared logic: src/shared and src/utils*

*This organization supports modular expansion: teams can introduce new features under src/features with minimal cross-impact, provided routes and context providers are properly registered.*

***SUMMARY***

*The component hierarchy reveals a clean flow of data and control—contexts manage global state, routes isolate feature boundaries, and presentation layers remain declarative.*

*This cohesion enables scalable, maintainable development and sets the stage for the next section on Styling & Theming, where visual consistency and accessibility are addressed..*

# STYLING AND THEMING

*Purpose of this section: Outline the styling framework, theming architecture, and design tokens that define the Weddi Frontend’s visual system, accessibility standards, and RTL readiness.*

*The application’s styling foundation is built on Tailwind CSS, enhanced with custom breakpoints extending up to 8K displays and fluid container widths fixed at 100% for large screens. This ensures full responsiveness across HRSD’s diverse hardware ecosystem—from typical desktop workstations to wall-mounted service displays. Safelist patterns are defined to preserve dynamically generated grid utilities during production builds, preventing layout regressions where column classes are computed at runtime.*

*【F:tailwind.config.js†L8-L63】*

*Typography is unified through the IBM Plex Arabic font family, imported globally via fonts.scss and mapped in Tailwind’s extended fontFamily definitions. The theme further extends Tailwind’s defaults to include precise spacing, padding, and border-radius scales—allowing expressive layouts without inline overrides. Color tokens and animation utilities are configured in tailwind.config.js and complemented by index.css, which standardizes UI primitives such as scrollbars, tables, and skeleton loaders to promote accessibility and visual consistency.*

*【F:tailwind.config.js†L24-L190】*

*【F:src/assets/styles/index.css†L1-L140】*

*The theming system prioritizes bilingual and RTL (Right-to-Left) support. The LanguageDirectionProvider dynamically toggles the dir attribute of the document root whenever the user switches languages, ensuring grids and flex layouts invert seamlessly. UI feedback elements—like loaders and progress bars—inherit brand accent colors aligned with HRSD’s corporate palette, maintaining a cohesive visual identity across dashboards, wizards, and administrative interfaces.*

*【F:src/i18n/LanguageDirectionProvider.tsx†L20-L49】*

*【F:src/shared/components/loader/index.tsx†L1-L28】*

***SUMMARY***

*Weddi’s styling framework—anchored in Tailwind’s scalability and enhanced by RTL-aware providers—ensures both visual continuity and accessibility compliance. With the design language established, the next section explores routing and navigation, mapping how users traverse and interact with these themed experiences.*

# ROUTING AND NAVIGATION

*Purpose of this section: Define the routing topology, describe navigation safeguards, and demonstrate how route composition aligns with feature boundaries and user flows.*

*The routing system is configured in App.tsx using React Router’s createBrowserRouter, with a /portal/ basename to support deployments where the SPA resides within a subdirectory. At the root level, MainLayout anchors the primary view hierarchy, hosting:*

* *An index route for the dashboard,*
* *An initiate-hearing route with nested stepper-based case creation, and*
* *manage-hearings routes for listing, detail viewing, and topic updates.*

*Modules are lazily imported, ensuring that hearing-related bundles are fetched only when accessed—minimizing the initial load and improving runtime performance.*

*【F:src/app/App.tsx†L41-L99】【F:src/app/App.tsx†L117-L139】*

*Dedicated /portal/login and /portal/logout routes isolate authentication transitions. The Logout route explicitly clears cookies and local storage before redirecting to environment-defined endpoints, while LoginLazy provides persona selection and token bootstrap workflows. Each route tree is wrapped with error boundaries to ensure that unexpected exceptions display controlled fallback UIs rather than blank screens.*

*【F:src/app/App.tsx†L101-L114】【F:src/features/auth/LogOut.tsx†L1-L24】.*

## OVERVIEW

*Navigation safeguards are enforced through context providers instead of route wrappers:*

* *TokenExpirationProvider monitors session validity and triggers re-authentication on token expiry.*
* *OfflineLayout exposes network connectivity status via global banners.*
* *MainLayout centralizes user feedback through NIC error modals and persona-change prompts triggered by backend responses.*

*This provider-driven architecture preserves a declarative routing configuration while maintaining global control over session state and error handling logic.*

*【F:src/providers/TokenExpirationProvider.tsx†L14-L75】  
【F:src/shared/layouts/MainLayout.tsx†L1-L145】*

***SUMMARY***

*The routing strategy reflects feature-based modularity and leverages providers as navigation guardians rather than imperative guards. This design ensures resilience, clear separation of concerns, and consistent user flow—laying the groundwork for the next section on data management and API integration, which powers these routes with live system data.*

# DATA MANAGEMENT AND API INTEGRATION

*Purpose of this section: Describe how the application fetches, transforms, and manages data from backend services to maintain consistency, fault tolerance, and real-time synchronization across all features.*

*The data layer is built around Redux Toolkit Query (RTK Query), configured in apiClient.ts using a custom createApi setup. The base query attaches both legacy (accesstoken) and modern OAuth (Authorization) headers to each request. Before dispatch, the transformRequest function enriches outgoing payloads with persona-aware metadata—such as SourceSystem, IDNumber, PlaintiffId, FileNumber, and government selections—ensuring backend endpoints receive complete contextual information. This design abstracts repetitive logic away from feature modules, preserving parity between the active persona state and API payloads even during role changes or wizard restarts.*

*【F:src/services/apiClient.ts†L115-L316】*

*When backend responses indicate invalid\_token, 401, or 403 errors, the base query automatically invokes the token refresh mechanism. The refreshToken function exchanges client credentials against /WeddiOauth2/v1/token, updates cookies with a new OAuth token and explicit expiry, and retries the original request. Unified error normalization through handleApiResponseLegacy and handleApiError ensures consistent handling across the app: backend codes are surfaced as toast notifications for users, while typed exceptions propagate to global error boundaries for developers. This structured flow keeps error states observable yet non-disruptive, supporting both user experience and maintainability.*

*【F:src/services/apiClient.ts†L20-L194】*

*【F:src/services/apiClient.ts†L321-L399】*

*Each feature module extends the shared API via api.injectEndpoints, registering domain-specific operations such as persona lookups (GetUserType), NIC queries, dashboard statistics, hearing lists, and wizard mutations. The auto-generated RTK Query hooks expose typed responses, loading indicators, and error flags, simplifying UI logic by removing the need for manual promise chains. Complementary utilities like useCookieState maintain wizard progress through cookies, allowing users to resume seamlessly across reloads or multi-tab workflows.*

*【F:src/features/auth/components/AuthProvider.tsx†L121-L253】【F:src/features/hearings/initiate/hooks/useCookieState.ts†L1-L124】*

***SUMMARY***

*RTK Query serves as the unified interface for all API communication, strengthened by cookie-aware persistence and automatic token lifecycle management. This foundation guarantees reliable, context-rich data exchange across modules—preparing the groundwork for the next section on TypeScript enforcement and static validation, which ensures that these interactions remain type-safe and predictable.*

# TYPESCRIPT USAGE AND TYPE SAFETY

## PURPOSE: *Explain how TypeScript configuration, interfaces, and runtime guards maintain correctness and code quality.*

## CONFIGURATION

* *tsconfig.json uses strict mode, ESNext module resolution, JSX react-jsx transform, and JSON module support.*
* *tsconfig.paths.json aliases @/\* → src, enforcing clean import patterns.*
* *Running tsc --noEmit during checks prevents code with type errors from building.*
  + *Refs: 【F:tsconfig.json†L1-L25】【F:tsconfig.paths.json†L1-L8】*

## RUNTIME GUARDS

* *Context hooks throw descriptive errors when used outside their providers (LanguageDirectionProvider, TokenExpirationProvider, UserTypeContext).*
* *Redux slices define typed state shapes; RTK Query endpoints return typed response interfaces.*
* *These constraints prevent runtime type drift when backend payloads evolve.*
  + *Refs: 【F:src/i18n/LanguageDirectionProvider.tsx†L20-L60】【F:src/providers/TokenExpirationProvider.tsx†L14-L75】【F:src/app/store/index.ts†L1-L39】*

## UTILITY ENFORCEMENT

* *getEnvVar retrieves environment variables with type conversions and throws early for missing keys.*
* *Date helpers validate Hijri data using regex and conversion rules to stay consistent with backend logic.*
  + *Ref: 【F:src/utils/helpers.ts†L92-L155】*

## SUMMARY / KEY INSIGHT: *Strict TypeScript settings and runtime guards create a dual safety net—compile-time validation and runtime protection. The next section extends this rigor to linting and formatting.*

# LINTING, FORMATTING & QUALITY GATES

**PURPOSE*:***

*Describe the static-analysis and formatting systems that keep code uniform and error-free.*

## LINTING

* *eslint.config.js extends JavaScript, TypeScript, and React rule sets.*
* *Integrates eslint-plugin-unused-imports to remove dead code.*
* *Treats config files as CommonJS to recognize Node globals safely.*
* *npm run check runs both ESLint and tsc --noEmit, surfacing syntax and type issues in one command.*
  + *Refs: 【F:eslint.config.js†L1-L44】【F:package.json†L6-L14】*

## FORMATTING

* *Prettier is applied implicitly through IDE settings.*
* *Import order and whitespace are enforced via ESLint rules.*
* *Teams may add .prettierrc if explicit configuration is preferred.*
  + *Ref: 【F:eslint.config.js†L1-L44】*

## CONTINUOUS INTEGRATION

* *No CI or pre-commit hooks exist yet.*
* *Integrating npm run check into CI ensures linting and type checks on every pull request.*
  + *Ref: 【F:package.json†L6-L14】*

## SUMMARY / KEY INSIGHT:

*Static analysis complements TypeScript safety, maintaining readability and preventing regressions. The next section examines testing coverage to close remaining quality gaps.*

# TESTING FRAMEWORK & COVERAGE

## PURPOSE

*Assess current testing status and outline future improvements.*

## CURRENT STATE

* *No automated tests exist; package.json lacks a test script.*
* *QA depends on manual regression testing for login, case initiation, and dashboard flows.*
  + *Ref: 【F:package.json†L6-L14】*

## FUTURE DIRECTION

* *Adopt Vitest with React Testing Library for unit and integration coverage.*
* *Focus on:*
  + *AuthProvider, MainLayout, and hearing wizard modules.*
  + *Token refresh and cookie synchronization logic.*
  + *Persona switching and route smoke tests.*
* *Include loader and routing transition tests to validate UX continuity.*

## SUMMARY / KEY INSIGHT:

*Testing is currently the weakest layer of assurance. Establishing automated suites will strengthen deployment confidence and reduce reliance on manual QA.*

# BUILD AND DEPLOYMENT PIPELINE

## PURPOSE: *Explain how builds are produced, what artifacts they generate, and how deployment workflows should be orchestrated for consistency across environments.*

## BUILD PROCESS

* *npm run build first executes TypeScript compilation (tsc) to ensure type safety before invoking vite build.*
* *If type errors occur, the build halts, preventing defective artifacts.*
* *vite.config.ts defines:*
  + *Base path /portal/*
  + *React plugin integration*
  + *Copying of static assets (fonts, locales)*
  + *Manual vendor chunking for stable cache keys*
* *The resulting dist/ directory includes hashed JS/CSS bundles and copied resources, ready for CDN hosting.*
  + *Refs: 【F:package.json†L6-L14】【F:vite.config.ts†L1-L63】*

## DEPLOYMENT WORKFLOW

* *Follow a deterministic sequence:*
  + *Install dependencies using lockfile-based caching.*
  + *Run npm run check to validate linting and typing.*
  + *Execute npm run build.*
  + *Archive the dist/ directory.*
  + *Deploy to the hosting environment.*
* *Reverse proxies or static hosts must rewrite requests to /portal/ to prevent 404 errors on deep links.*
* *Run npm run preview for smoke testing post-build to validate environment configurations.*
  + *Ref: 【F:vite.config.ts†L1-L63】*

## SUMMARY / KEY INSIGHT:

*A deterministic build pipeline guarantees reproducibility and stable deployments. Since failures can still occur in runtime environments, the following section details how to troubleshoot and resolve common issues efficiently.*

# TROUBLESHOOTING / FAQ

## PURPOSE: *Compile common failure modes, diagnose root causes, and prescribe actionable resolutions to support rapid incident response.*

| ***Issue*** | ***Cause*** | ***Resolution*** |
| --- | --- | --- |
| *Redirect loop between portal and identity provider* | *Token cookie missing or expired; TokenExpirationProvider cleared session and redirected.* | *Inspect cookies for token; reauthenticate via HRSD identity provider; verify VITE\_REDIRECT\_URL(\_LOCAL) configuration.【F:src/providers/TokenExpirationProvider.tsx†L14-L55】* |
| *Dashboard modules stuck in loading state* | *AuthProvider waiting for NIC data or persona lookup; MyClientsToken missing.* | *Ensure redirect includes token parameter; clear cookies via /logout; confirm backend GetNICDetails availability.【F:src/features/auth/components/AuthProvider.tsx†L121-L205】* |
| *Legal representative persona modal cannot proceed* | *Missing persistence of Main/Sub government selections.* | *Select both dropdown values in LoginAccountSelect; verify cookie sync so selections reach transformRequest.【F:src/features/auth/components/LoginAccountSelect.tsx†L108-L197】【F:src/services/apiClient.ts†L240-L309】* |
| *OAuth requests return 401* | *Refresh token flow missing credentials or using expired secrets.* | *Set VITE\_OAUTH\_CLIENT\_ID/SECRET/GRANT\_TYPE to match backend; monitor refreshToken logs.【F:src/services/apiClient.ts†L20-L87】* |
| *Case wizard loses progress after reload* | *useCookieState blocked updates due to caseDataCleared flag.* | *Run useClearCaseData reset helper or delete caseDataCleared cookie to restore persistence.【F:src/features/hearings/initiate/hooks/useCookieState.ts†L52-L75】【F:src/shared/hooks/useClearCaseData.ts†L43-L55】* |
| *Persona switches leave stale data* | *Persona cookies persisted across sessions.* | *Trigger /logout or manually clear persona cookies before re-login.【F:src/features/auth/LogOut.tsx†L1-L24】* |
| *Tailwind classes missing in production* | *Purge removed dynamic class names.* | *Extend safelist in tailwind.config.js or adjust dynamic class generation patterns.【F:tailwind.config.js†L8-L63】* |
| *Layout renders LTR for Arabic locale* | *LanguageDirectionProvider not initialized or missing translation namespace.* | *Verify inclusion in AppProvider; ensure i18n loads the meta namespace; clear cache and reload.【F:src/providers/AppProvider.tsx†L17-L40】【F:src/i18n/LanguageDirectionProvider.tsx†L20-L49】* |
| *Toasts show raw backend error codes* | *Missing message mapping in error handler.* | *Extend customErrorMessages or adjust SUPPRESSED\_ERROR\_CODES for readable feedback.【F:src/utils/api/errorHandler.ts†L1-L120】* |
| *Loader overlay never dismisses after API failure* | *pendingCount not decremented due to unhandled exception.* | *Wrap async calls in try/finally to ensure stopLoading runs; confirm correct Redux flow.【F:src/app/store/slices/loadingSlice.ts†L1-L21】【F:src/services/apiClient.ts†L115-L194】* |
| *Local preview serves blank page* | *Host missing /portal/ rewrite or missing build artifacts.* | *Run npm run build before npm run preview; configure host rewrites for SPA routing.【F:vite.config.ts†L1-L63】* |

**Summary / Key Insight*:*** *Troubleshooting documentation accelerates incident response. Each fix ties back to security and performance considerations, laying the foundation for the next section on integrated safeguards and optimization strategies.*

# SECURITY AND PERFORMANCE NOTES

**PURPOSE*:***

*Highlight measures that protect user data, maintain session integrity, and optimize performance under real-world load.*

## SECURITY

* *Token lifecycle is managed through TokenExpirationProvider, which:*
  + *Decodes JWT payloads.*
  + *Warns users five minutes before expiry.*
  + *Removes expired tokens and redirects according to configured URLs.*
* *OAuth refresh logic in apiClient.ts ensures continuous authentication without unnecessary token exposure.*
* *Cookies use scoped paths and the secure flag (enforced via useCookieService outside development), minimizing attack vectors for sensitive identifiers.*
  + *Refs: 【F:src/providers/TokenExpirationProvider.tsx†L14-L55】【F:src/services/apiClient.ts†L20-L194】【F:src/shared/hooks/useCookies.ts†L1-L38】*

## PERFORMANCE

* *Lazy route loading and manual vendor chunking in Vite stabilize bundle names for long-term caching.*
* *Loader overlays are synchronized through a Redux pendingCount, preventing flickering during simultaneous API calls.*
* *Cookie-synchronized hooks (useCookieState, useClearCaseData) persist metadata across sessions and tabs, reducing redundant network requests.*
  + *Refs: 【F:vite.config.ts†L26-L63】【F:src/app/store/slices/loadingSlice.ts†L1-L21】【F:src/features/hearings/initiate/hooks/useCookieState.ts†L1-L124】*

## ACCESSIBILITY AND RESILIENCE

* *OfflineLayout proactively surfaces connectivity issues.*
* *i18n providers maintain text direction and translations without blocking rendering, improving responsiveness during network degradation or mid-session language changes.*
  + *Refs: 【F:src/app/App.tsx†L10-L139】【F:src/i18n/LanguageDirectionProvider.tsx†L20-L49】*

## SUMMARY / KEY INSIGHT*:*

*Security and performance optimizations are embedded at every architectural layer—providers, hooks, and build tooling. Sustaining these standards demands disciplined, consistent collaboration across contributors.*

# CONTRIBUTION GUIDELINES

## PURPOSE*: Define conventions for branching, commits, code style, and review workflows to maintain the clean edition’s consistency.*

*Branching and Workflow*

* *Use the format feature/<scope> or bugfix/<scope>.*
* *Rebase frequently with main to minimize merge conflicts.*
* *Pull requests must:*
  + *Reference related tickets (e.g., Jira).*
  + *Include screenshots or GIFs for UI changes.*
  + *Pass npm run check before push to satisfy linting and typing rules.*
  + *Ref: 【F:package.json†L6-L14】*

## COMMIT AND CODE STYLE

* *Follow Conventional Commits (e.g., feat(dashboard): ..., fix(api): ...) for automated changelog generation and clarity.*
* *Favor functional components, hooks, and Tailwind utilities. Avoid class components or inline styles except for rare dynamic theming.*
* *Integrate new stateful logic into Redux slices or context providers for maintainability.*
  + *Refs: 【F:src/shared/components/loader/index.tsx†L1-L28】【F:src/app/store/index.ts†L1-L39】*

## REVIEW STANDARDS

* *Confirm new API calls use api.injectEndpoints.*
* *Ensure translations are updated for all user-facing changes.*
* *Verify persona cookies remain consistent.*
* *Complex flows must be documented in tech-design.md or related documents to centralize project knowledge.*
  + *Ref: 【F:tech-design.md†L19-L122】*

## SUMMARY / KEY INSIGHT:

*Consistent contribution discipline ensures maintainability and transparency. Clear changelogs and version tracking underpin the upcoming Versioning Strategy section.*

# CHANGE LOG & VERSION HISTORY

## PURPOSE:

*Advocate for disciplined versioning and provide a template for documenting changes over time.*

*The repository currently declares version `0.0.0`, signifying a pre-release state. Adopting Semantic Versioning (`MAJOR.MINOR.PATCH`) empowers teams to communicate breaking changes, feature additions, and bug fixes succinctly. Each release should capture highlights under Added/Changed/Fixed headings, enabling stakeholders to assess upgrade impact rapidly.*

*【F:package.json】*

**Example entry**

* *Added*
  + *Legal representative dashboard banner enhancements drawing on new schedule endpoints.*
* *Changed*
* *Upgraded Vite to 6.2.6 to leverage improved legacy plugin support.*
* *Fixed*
* *Resolved OAuth token refresh race condition impacting idle sessions.*

*Maintaining such records in `CHANGELOG.md` (or similar) streamlines communication across engineering, QA, and operations.*

## SUMMARY / KEY INSIGHT:

*Transparent version history underpins coordinated releases. Looking forward, the roadmap of future improvements clarifies where the clean edition can evolve next.*

# FUTURE IMPROVEMENTS AND NEXT STEPS

## PURPOSE:

*Outline strategic enhancements that would elevate reliability, developer experience, and feature velocity.*

*High-priority initiatives include introducing automated tests (Vitest + React Testing Library) for authentication, routing guards, and wizard flows; establishing Storybook or a similar component workbench to document shared UI primitives; and expanding performance monitoring via Lighthouse CI or Web Vitals dashboards. These actions would catch regressions earlier and document component behavior for design alignment.*

*【F:src/features/auth/components/AuthProvider.tsx】*

*【F:src/shared/components/loader/index.tsx】*

*Type safety can be further hardened by refining Redux slices that currently rely on loosely typed defaults (`any` usage in form reducers). Additionally, automating locale synchronization, introducing environment variable templates, and exploring performance budgeting (e.g., limiting bundle size growth per release) would keep the clean edition scalable as new features land.*

*【F:src/app/store/slices/*[*formSlice.ts*](http://formslice.ts)*】*

*【F:src/utils/*[*helpers.ts*](http://helpers.ts)*】*

## SUMMARY / KEY INSIGHT:

*A proactive improvement roadmap ensures the clean edition remains sustainable. Before implementing these ideas, stakeholders should revisit foundational terminology and references, presented in the appendices below.*

# APPENDICES

## PURPOSE:

*Provide reference material—glossary, related documentation, and external links—to support contextual understanding*.

## GLOSSARY

* *Wedd: HRSD’s Amicable Settlement program for employment dispute resolution, serviced through this portal.*
* *NIC: National Identity Centre service that validates user identity during login workflows.【F: src/features/auth/components/AuthProvider.tsx】*
* *Persona: Role-specific identity (Worker, Establishment, Legal representative, Embassy delegate) dictating UI access and payload metadata. 【F: src/features/auth/components/LoginAccountSelect.tsx】*

## EXTERNAL REFERENCES

*HRSD identity provider documentation (external): governs authentication flows consumed by this front-end. Weddi-Services REST API specifications (internal): describe request/response payloads referenced by RTK Query endpoints.*

## SUMMARY / KEY INSIGHT

*Armed with terminology and references, contributors can bridge code to institutional knowledge. The subsequent onboarding notes translate these insights into actionable guidance for new developers*.

# FINAL NOTES AND RECOMMENDATIONS FOR NEW DEVELOPERS

## PURPOSE:

Offer onboarding guidance tailored to engineers who are newly responsible for extending or maintaining the Weddi Frontend.

## START WITH PROVIDER COMPOSITION:

Read `AppProvider`, `MainLayout`, and `apiClient` before touching feature code. Understanding how providers wrap the router, how persona state is injected, and how API requests are transformed will prevent missteps when adding new routes or endpoints.

【F: src/providers/AppProvider.tsx】

【F: src/shared/layouts/MainLayout.tsx】

【F:src/services/apiClient.ts】

## TRACE PERSONA DATA END-TO-END:

Follow the flow from `AuthProvider` through cookie persistence (`useCookieState`, `useClearCaseData`) and into RTK Query transforms. This reveals why certain cookies exist and how they influence headers or payloads, which is critical when debugging role-based behaviour.

【F: src/features/auth/components/AuthProvider.tsx】

【F: src/features/hearings/initiate/hooks/[useCookieState.ts](http://usecookiestate.ts)】

## REVIEW SHARED COMPONENTS AND HOOKS:

Explore `src/shared/components` and `src/shared/hooks` to avoid reinventing primitives or utilities. Tabs, loaders, and cookie services are already battle-tested; reuse them to maintain consistency.

【F: src/shared/components/loader/index.tsx】

【F: src/shared/hooks/[useClearCaseData.ts](http://useclearcasedata.ts)】

## BE MINDFUL OF ENVIRONMENT DEPENDENCIES:

Before working on features that touch authentication or data fetching, ensure your `. env.local` mirrors staging configurations. Missing OAuth credentials or API URLs manifest as runtime failures that can be mistaken for code regressions.

【F:src/services/apiClient.ts】

【F: src/utils/[helpers.ts](http://helpers.ts)】

**ANTICIPATE RTL AND LOCALIZATION IMPACT:**

When adding layouts or copy, test both Arabic and English views. Confirm that direction changes via `LanguageDirectionProvider` do not break flex/grid structures and that translations exist for new strings. 【F: src/i18n/LanguageDirectionProvider.tsx】

## COORDINATE WITH QA EARLY:

Because automated tests are not yet implemented, collaborate with QA analysts to script manual regression steps for features you touch. Provide them with route paths and persona scenarios to streamline verification.

## SUMMARY / KEY INSIGHT

Following this path helps new developers acclimate quickly while preserving the clean edition’s architectural rigor.