Low-Level Design (LLD) — Rules & Codes App (React + TypeScript + SCSS modules)

> **Status notes:** You marked several items as `TODO` or `Revisit`. I preserved these flags throughout the document where applicable. Treat them as action items to resolve during API / backend discussions or during early implementation planning.

---

1. High-Level Overview

- **Project name:** Rules & Codes Management (placeholder)
- **Purpose:** Administer and manage business rules and codes with draft/live workflows, advanced filtering, and role-based access.
- **Tech stack:** React + TypeScript + SCSS modules, React Router (v7), Zustand (based on requirements), Axios, date-fns, MSAL (Auth - Revisit), ESLint + Prettier

---

2. Actors & Roles

- **Admin** — full access: create/update/delete/publish/close rules & codes, manage drafts
- **Normal User** — restricted actions (view, use filters, save drafts maybe) — exact permissions **TODO: define**

---

3. Screen / Feature Inventory (flat list)

1. Login (root: `/#/login`) — center login over background image
2. Rules List (main=1, sub=1) — list with advanced filter, table, bulk/select actions
3. Create Rule (main=1, sub=2) — form (2-column)
4. View Draft Rules (main=1, sub=3) — draft list with advanced filter
5. Update Rule (`/rule/edit/:id`) — edit form (2-column)
6. Codes List (main=2, sub=1) — list with advanced filter, similar to rules
7. Create Code (main=2, sub=2) — form (2-column)
8. View Draft Codes (main=2, sub=3) — draft list
9. Update Code (`/code/edit/:id`) — edit form (2-column)

> All list screens: advanced filter, sorting (TODO), pagination, selection, bulk actions (close/delete/save-to-live)

---

4. Screen-level LLD

Each screen section contains: layout, components, navigation (route), interactions, state, data & API notes, validation/business rules (TODO where provided).

4.1 Login

- **Route:** `/#/login`
- **Layout:** Full-bleed background image; centered login button/card
- **Components:** `LoginCard` (Button, Loader), `AuthRedirector` (handles MSAL redirect) — `Button`, `Navigation`
- **Interactions:** Microsoft authentication via MSAL. On success, store user details in global state and redirect to home. (Auth flow: **Revisit** for MSAL config)
- **State:** local UI state (loading, error). Global: `auth.user` after success.
- **Data/API:** Auth endpoints handled by MSAL; user profile retrieval **TODO**
- **Validation/Rules:** NA

4.2 Rules List (Main=1 | Sub=1)

- **Route:** `/#/home?main=1&sub=1` (Revisit)
- **Layout:** Tabbed layout (2 main tabs × 3 sub-tabs). This is main tab 1, sub tab 1. Primary area: advanced filter panel + table.
- **Components:** `TabLayout`, `AdvancedFilter`, `RuleTable`, `Pagination`, `FilterChipList`, `BulkActionsBar`, `ConfirmationModal` (for close), `Indicator`
- **Interactions:**
- Apply/clear advanced filters
- Sort table (fields to be defined — TODO)
- Pagination
- Select single/multiple rows
- Navigate to Update Rule screen
- Trigger Close Rule → show justification modal → call API
- **State:**
- Local: `rulesList`, `filterState`, `ui.loading`, `ui.selectedRows`
- Global: none required (unless filters or selection must persist across pages)
- **Data/API:**
- Endpoint(s): `GET /api/rules` with filter & paging parameters (TODO: exact contract)
- `POST /api/rules/close` — close selected rules (requires justification)
- Response model: List of `RuleSummary` objects (TODO: define fields)
- **Validation/Rules:** Sorting fields and business rules around close action — **TODO**

4.3 Create Rule (Main=1 | Sub=2)

- **Route:** `/#/home?main=1&sub=2` (Revisit)
- **Layout:** 2-column form layout inside tabbed container

- **Components:** `RuleForm` (composed from atoms: `TextInput`, `DateTimePicker`, `Dropdown`, `OnOffSwitch`, `MultiSelect`, `RichText` [if needed])
- **Interactions:** Fill inputs, validate, Save Draft, Save Live, Cancel
- **State:** Local form state (controlled fields). On successful save, clear or redirect.
- **Data/API:**
- `POST /api/rules` (create)
- Request/response shape: **TODO**
- **Validation/Rules:** Form field validations and conditional rules — **TODO**

4.4 View Draft Rules (Main=1 | Sub=3)

- **Route:** `/#/home?main=1&sub=3` (Revisit)
- **Layout & Components:** Similar to Rules List — `AdvancedFilter`, `DraftRuleTable`, `Pagination`, `ConfirmationModal`
- **Interactions:** Filter, select rows, Save to Live (publish), Delete draft (with justification), Navigate to Update
- **State:** Local: `draftRules`, `filterState`
- **Data/API:**
- `GET /api/rules/drafts`
- `POST /api/rules/:id/publish` (save to live)
- `DELETE /api/rules/:id` or `POST /api/rules/:id/delete` (with justification)
- **Validation/Rules:** TODO

4.5 Update Rule

- **Route:** `/rule/edit/:id`
- **Layout:** 2-column form (same as Create Rule) prefilled with API data
- **Components:** `RuleForm` (reused)
- **Interactions:** Load rule by id, edit, save/update, cancel
- **State:** Local: `formState` prefilled from `GET /api/rules/:id`
- **Data/API:** `GET /api/rules/:id`, `PUT /api/rules/:id`
- **Validation/Rules:** TODO

4.6 Codes List (Main=2 | Sub=1)

- **Route:** `/#/home?main=2&sub=1` (Revisit)
- **Layout/Components/Interactions/State/Data:** Mirror Rules List screen but for `Code` entities. Sorting fields & business rules — **TODO**

4.7 Create Code (Main=2 | Sub=2)

- **Route:** `/#/home?main=2&sub=2` (Revisit)

- **Layout:** 2-column form
- **Components:** `CodeForm` (reused patterns from RuleForm)
- **Interactions/State/Data/Validation:** Similar to Create Rule. **TODO** for exact fields and validation

---

4.8 View Draft Codes (Main=2 | Sub=3)

- Same pattern as View Draft Rules. Actions: filter, publish (save-to-live), delete draft, edit.
- **Data/API:** `GET /api/codes/drafts`, `POST /api/codes/:id/publish`, `DELETE /api/codes/:id` — **TODO**

4.9 Update Code

- **Route:** `/code/edit/:id`
- **Layout/Components/Interactions/State/Data:** Mirror Update Rule. Prefill from `GET /api/codes/:id` and `PUT /api/codes/:id`. **TODO** validations

---

5. Data Models (Suggested interfaces — draft)

Fill exact fields after backend contract is available.

```ts
// src/types/index.ts
export interface User {
  id: string;
  name: string;
  email: string;
  roles: string[]; // ['ADMIN','USER']
}

export interface RuleSummary {
  id: string;
  title: string;
  status: 'DRAFT' | 'LIVE' | 'CLOSED' | string;
  createdAt: string; // ISO
  updatedAt?: string;
  // TODO: add business-specific fields
}

export interface RuleDetail extends RuleSummary {
  description?: string;
  effectiveFrom?: string;
  effectiveTo?: string;
  owner?: string;
```

```
  // TODO: full payload fields
}

export interface CodeSummary {
  id: string;
  code: string;
  description?: string;
  status: 'DRAFT'|'LIVE'|'CLOSED'|string;
}
```

---

6. State Management (Zustand)

- **Stores:**
- `useAuthStore` — user, tokens, isAuthenticated
- `useUiStore` — global UI flags (global loader, global error)
- `useFilterStore` (optional) — persisted filters across navigation (Revisit: decide if filters should persist)
- `useEntitiesStore` (optional) — caches for rules/codes lists if required
- **Local state:** component-controlled forms, transient UI state, pagination state unless global persistence required.

---

7. Routing

- **Library:** React Router v7
- **Route map (suggested):**
- `/login`
- `/home?main=1&sub=1` — rules list
- `/home?main=1&sub=2` — create rule
- `/home?main=1&sub=3` — draft rules
- `/rule/edit/:id`
- `/home?main=2&sub=1` — codes list
- `/home?main=2&sub=2` — create code
- `/home?main=2&sub=3` — draft codes
- `/code/edit/:id`

Revisit: canonical query param approach vs nested routes — decide during routing design.

---

8. Styling & SCSS Modules

- **Per-component SCSS modules:** each component folder contains `Component.module.scss`
- **Global tokens:** `/styles/_variables.scss` (colors, spacing, typography). SCSS variables naming: underscore_separated (as requested)

- **Theming:** prepare for light/dark tokens (if needed)
- **Conventions:** prefer BEM-ish local class names inside modules to keep readability.

---

9. Reusable Components (folder & responsibilities)

`src/components/<ComponentName>/` → `{Component}.tsx`, `index.ts`, `{Component}.module.scss`

Core atoms & molecules - `Button`, `Icon`, `TextInput`, `TextArea`, `Checkbox`, `Dropdown`, `MultiSelect`, `DatePicker`, `DateTimePicker`, `OnOffSwitch`, `Loader`, `Alert`

Organisms / patterns - `Table` (with selectable rows, sorting hooks) - `TabLayout` (supports recursive tabs as required) - `Modal` / `ConfirmationModal` - `FilterLayout` / `AdvancedFilter` (composed from atomic inputs) - `Pagination` - `Header`, `Footer`, `SecuredLayout` / `PrivateRoute` wrapper

---

10. Folder & Code Organization (suggested)

```
src/
├─ assets/
├─ components/
│  ├─ Button/
│  ├─ Table/
│  └─ ...
├─ constants/
├─ core/      # bootstrapping, App.tsx, router
├─ contexts/  # if any React Context used
├─ helpers/
├─ hooks/
├─ layouts/
├─ mocks/
├─ pages/
│  ├─ Login/
│  ├─ Home/
│  └─ RuleEdit/
├─ services/  # axios instances, API clients
├─ store/     # zustand stores
├─ styles/    # global scss tokens and utility classes
├─ types/
└─ utils/
```

Naming rules enforced: - TypeScript variables & identifiers: `camelCase` - SCSS variables: `underscore_separated`

11. API & Integration (high-level)

- Use `src/services/api.ts` to create configured Axios instance (interceptors for auth token, error handling)
- Services per entity: `services/rules.ts`, `services/codes.ts` exposing typed methods
- Error handling: central error parsing utility + global error boundary and toasts/alerts

Example service signature (pseudocode):

```
export const fetchRules = (filters: RuleFilterParams, page: number, size: number) => axios.get('/api/rules', { params: { ...filters, page, size } });
```

> **TODO:** finalize exact endpoints and payload contracts with backend team.

---

12. Validation, Business Rules & Edge Cases

- Many business rules are marked `TODO` — items to clarify:
- Sorting fields for lists
- Exact publish/close/delete flow and required justification fields
- Permissions: what Admin vs Normal User can perform
- Form field-level validations and conditional rules

---

13. Non-Functional Requirements

- **Performance:** server-side pagination, lazy-loading heavy components (React.lazy + Suspense), memoize large lists (`React.memo`, `useMemo`), virtualized tables if row counts are large
- **Security:** store only non-sensitive user metadata in client state. Tokens via MSAL/secure storage (Revisit)
- **Testing:** unit tests for components, integration tests for key flows, e2e for publish/delete flows — set up Testing Library + Vitest/Jest + Playwright/Cypress later

---

14. Developer Tooling & CI

- ESLint + Prettier config (shared), TypeScript strictness (`strict` true recommended)
- Husky pre-commit hooks (lint, type-check)
- CI pipeline: build, lint, test, bundle size check

---

15. Implementation Roadmap & Priorities (suggested)

1. Authentication flow & protected routes (MSAL — Revisit)
2. Basic routing, Secured layout, global stores (auth, ui)
3. Skeleton of Rules List with table, filters (static mock data)
4. Create Rule form + API contract for create
5. Edit Rule flow + Drafts
6. Mirror above for Codes feature
7. Finalize sorting & business rules, polish UX

---

16. TODO & Revisit Checklist (actionable)

- [ ] Finalize routing approach: query params vs nested routes. **Revisit**
- [ ] MSAL authentication configuration and token handling. **Revisit**
- [ ] Backend API contract: endpoints, request & response schema for rules/codes. **TODO**
- [ ] Sorting fields and business rules for lists. **TODO**
- [ ] Decide which filters persist across navigation (useFilterStore?). **Revisit**
- [ ] Confirm admin vs normal user permission matrix. **TODO**
- [ ] Define exact Rule/Code payload fields for `RuleDetail` / `CodeDetail`. **TODO**
- [ ] Accessibility requirements (a11y) — if required, add checklist. **Revisit**

---

17. Deliverables (what AI or dev can produce next)

- Component hierarchy per screen (tree)
- Detailed TypeScript interfaces for all API payloads (once backend contract provided)
- Sample `RuleForm` and `CodeForm` component implementations
- Example `Table` component with selection & server-side pagination hooks
- Folder skeleton generator (file templates)

Notes

- This LLD is intentionally mid-fidelity and marked where you requested `TODO` / `Revisit`. Once the backend API contract and exact business rules are available, the LLD can be expanded into full component-level code artifacts and typed services.

Prepared by: AI-assisted LLD generator — provide more backend details or indicate which deliverable you want next (e.g., component tree, code templates, interfaces) and I will generate it.