FairChoices Web Development Guidelines

Next.js + React

1. Code Style and Quality

Linting & Formatting

- **ESLint:** For consistent code with React/Next/Typescript plugins.
- **Prettier:** Enforce consistent formatting (e.g. trailing commas, semicolons, tabs)
- **Husky & lint-staged:** Run linting and formatting on pre-commit.
- **Typescript:** Mandatory for type safety. Use strict mode in tsconfig.
- **Recommended VSCode extensions:** ESLint, Prettier, Tailwind CSS IntelliSense.

2. Version Control

Branching Strategy

- **Branch naming:** `[issue-id]/[type]-[short-description]` (e.g., `FAIR-123/feat-user-auth`)
- Delete merged branch.

Pull Requests

- **Template:** Include Jira ticket link, screenshots, screen recording, testing steps, and migration notes.
- **Review:** At least 1 approval for the PR.
- **Squash Merge:** Clean commit history.

Commit Messages

- Follow Conventional Commits for clarity: `feat: allow provided config object to extend other configs`

3. Component Architecture

Design Principles

- **Atomic Design:** Organize components as Atoms (buttons), Molecules (search bars), Organisms (header), Templates (layouts).
- **Storybook:** Every component requires a `.stories.tsx` file colocated with its source or separate folder structure.
- **Server Components:** Use Next.js 13+ App Router for server-rendered components where possible.
- **State Management:**
    - Local state: useState/useReducer/useContext
    - Global state: Redux Toolkit (RTK) for centralized state management.
    - Avoid prop drilling: Use React Context sparingly and utilize Redux's slice-based architecture.
    - API calls: Use RTK Query for centralized data fetching.
- **Styling & Theming**
    - **Tailwind CSS:** Primary styling solution. Use `clsx` for dynamic class names.
    - **CSS Modules:** Apply for complex, component-specific styles.
    - **Design Tokens:** Centralize colors, spacing, and typography in a dedicated file.

4. Testing Strategy

Unit & Integration Tests

- **Tools:** Jest and React Testing Library
- **Coverage Goal:** Maintain 80%+ test coverage on core features.
- **Best Practices:**
    - Test user flows over implementation details.
    - Use MSW (Mock Service Worker) to simulate API calls.

Visual Testing

- **Storybook:**
    - Hosted on Chromatic for visual regression testing.
    - Addons: Controls, Actions, Accessibility, and Figma plugin.

5. Performance & Optimization

Core Web Vitals

- **Monitoring:** Use Lighthouse CI to track performance

Turborepo Caching

- Cache node_modules, builds, and tests using `turbo.json` remote caching.
- Use `-filter` to target specific apps/packages.

Next.js Specific Optimizations

- **Dynamic imports:** Lazy-load non-critical components.
- **Image Optimization:** Use `next/image` with AVIF/WEBP support.
- **ISR/SSG:** Pre-render static pages.
- **Bundle Analysis:** Leverage `@next/bundle-analyzer` to identify and reduce bloat.
- **Partial Prerendering (PPR):** Hybrid static/dynamic rendering.

6. Security

Frontend Security

- **CSP Headers:** Configure in `next.config.js` to mitigate XSS risks.
- **Sanitization:** Use libraries like DOMPurify to clean user-generated HTML.
- **Authentication:** Implement NextAuth.js with JWT sessions and RBAC.

Backend (API Routes)

- **Rate Limiting:** Apply rate limit for API endpoints.
- **Input Validation:** Use Zod together with TypeScript for schema validation.
- **Dependency Management:** Regularly run `npm audit` and utilize Dependabot for vulnerability patches.

7. Turborepo Monorepo Structure

```
/
├─ .storybook/
├── apps/
│   ├── dcp-tool/            # Next.js 15 App Router
│   ├── storybook/       # Component documentation
│
├── packages/
│   |── ui/          # Component library (Button, Input)
│   |-- utils/       # Shared utilities
│   |
│   ├── eslint-config/   # Shared ESLint rules
│   └── tsconfig/        # Base TypeScript configs
│
├── turbo.json           # Pipeline & caching config
└── package.json         # Workspace setup
```

Turborepo Guidelines

- **Adding a Package:**
  - Use `turbo gen` to scaffold with standardized templates.
  - Update `tsconfig.json` references and package.json dependencies.
- **Dependency Management:**
  - Hoist common dependencies to root `package.json`
- **Caching:**
  - Use dependsOn in turbo.json to optimize task execution order.

8. Documentation

- **Component Library:** Storybook + Chromatic for visual testing.
- **API:** Swagger/OpenAPI for endpoints.

9. Naming Conventions

Files & Folders

- **PascalCase:** React component files (e.g., `UserProfile.tsx`)
- **Descriptive Names:** Avoid generic terms like `utils` → `date-utils`, `auth-utils`
- **camelCase:** For filenames and folders.

Components

- **Naming:**
  - Prefix with domain (e.g., FormSubmitButton, ProfileAvatar)

    - Use `index.ts` for barrel exports (never `Button.tsx` and `Button/index.tsx` together).
- **Props:**
    - Use type for public props (e.g., `type ButtonProps = { id: string }`)
    - Forwarded Refs: suffix with `Ref` (e.g., `InputRefProps`).

Variables & Functions

- **camelCase:** For variables, functions, and hooks (e.g., `taxonomyData`, `fetchTaxonomyData`).
- **Booleans:** Prefix with `is`, `has`, or `should` (e.g., `isLoading`, `hasPermission`).
- **Events:** Prefix with on (e.g., `onSubmit`, `onClick`).
- **Redux Slices:** Suffix with Slice (e.g., `authSlice`, `taxonomySlice`)

CSS/Tailwind

- **BEM-like Classes:** Only if using CSS Modules.
- **Tailwind:** Use semantic utility classes (e.g., `bg-primary` instead of `bg-[#000000]`)

Typescript

- **PascalCase:** Types and interfaces (e.g., `UserProfile`, `ApiResponse`).
- **Generic Types:** Use `T` prefix (e.g., `type PaginatedResponse<T>`).

Tests

- **Filename:** [component].test.tsx (e.g., Button.test.tsx).
- **Test IDs:** Use data-testid with kebab-case (e.g., `data-testid="user-profile-avatar`”).

10. Component Development: Dos & Don’ts

✅ **Do:**

- **Isolate Logic:** Use custom hooks for logic.
- **Type Everything:** No any - use unknown and type guards if needed.
- **Storybook First:** Build components in Storybook before integrating into apps.
- **Re-Export:** Use barrel files (index.ts) for clean imports.

❌ **Don’t:**

- **Prop Drilling:** Use Redux or Context for cross-component state.

- **Inline Styles:** Tailwind > CSS Modules > styled-components (in that order).
- **Large Components:** Split into subcomponents if exceeding 150 lines.
- **Export Default:** Always use named exports for better tree-shaking.