DieKoffieBlik

Coding Standards

1. Introduction..... 2
2. General Coding Guidelines..... 3
3. Frontend Standards..... 3
3.1 React Native App (Expo + TS)..... 3
3.2 Next.js Web Application..... 4
4. Backend & Database Standards..... 4
4.1 Supabase Database..... 4
4.2 API & Data Access..... 5
5. Testing Standards..... 5
6. File and Folder Structure..... 6
6.1 React Native App..... 6
6.2 Next.js Web App..... 6
7. Git & Version Control Guidelines..... 7

1. Introduction

This document defines the coding standards, conventions, and repository structure used in the development of the **Coffee Shop Platform**. The platform consists of:

- **Mobile App** (React Native + TypeScript)
- **Web Application** (Next.js + TypeScript)
- **Database** (Supabase)

The goal of these standards is to ensure:

- **Consistency** across frontend and backend code.
- **Clarity** so new developers can easily understand the codebase.
- **Reliability** through structured testing and validation.
- **Maintainability** for long-term project sustainability.

---

2. General Coding Guidelines

- **Language**: TypeScript for both frontend and backend.
- **Linting**: ESLint with project-specific rules is enforced.
- **Formatting**: Prettier (integrated with ESLint) ensures consistent formatting.
- **Naming Conventions**:
    - Variables & functions → `camelCase`
    - Components, classes, types, interfaces → `PascalCase`

    - Constants → `UPPER_SNAKE_CASE`

- **Code Style:**
    - Use functional components and hooks in React/Next.js.
    - Avoid long functions (>50 lines). Break into smaller reusable functions.
    - Prefer composition over inheritance.
    - Always handle errors explicitly (try/catch, error boundaries).

---

3. Frontend Standards

3.1 React Native App (Expo + TS)

- Each screen/component is in its own folder.
- Common UI elements live in `/components`.
- Global styles/themes live in `/styles`.
- API communication is abstracted into `/services` or `/backend`.
- State management is handled via React hooks or context where necessary.
- All props and state typed with TypeScript.

Example:

```
type CoffeeCardProps = {
  name: string;
  price: number;
};

export function CoffeeCard({ name, price }: CoffeeCardProps) {
  return (
    <View>
      <Text>{name}</Text>
      <Text>R{price}</Text>
    </View>
  );
}
```

3.2 Next.js Web Application

- Use the **App Router** (`/app`) structure.
- Pages live under `/app`, grouped by route.
- Reusable components go under `/components`.
- Server actions and API routes live under `/app/api`.
- Supabase client wrapped in a central utility file for reuse.
- Type safety is enforced with TypeScript validation where applicable.

---

4. Backend & Database Standards

4.1 Supabase Database

- Tables named in **snake_case**.
- Columns named in **snake_case** (e.g., `user_id`, `order_status`).

- Primary keys always use `id` (UUID by default in Supabase).
- Foreign keys named as `<table>_id` (e.g., `user_id` in `orders`).
- All queries go through the Supabase client SDK.

4.2 API & Data Access

- Use a **service layer** for all database interactions.
- Follow RESTful patterns when exposing API endpoints (Next.js API routes).
- Never expose secrets in frontend code (use environment variables).

---

5. Testing Standards

- **Unit Tests**: For isolated functions/components.
- **Integration Tests**: For combined features (e.g., order flow using mock Supabase).
- **End-to-End (E2E) Tests**: For full user flows (e.g., login → order → payment).
- **Frameworks**:
    - Jest for unit and integration tests.
    - Playwright or Cypress for e2e tests.
- All tests live in `__tests__` directories alongside code.
- Test files follow `the *.test.ts` naming convention.

---

6. File and Folder Structure

6.1 React Native App

```
coffee-shop-app/
 ├── app/                 # Expo Router screens
 │    ├── index.tsx       # Home
 │    ├── login.tsx       # Login
 │    └── order/          # Order flow
 ├── components/          # Reusable components
 ├── services/            # API calls (Supabase, payments)
 ├── styles/              # Shared styles & themes
 ├── utils/               # Helpers
 ├── __tests__/           # Unit & integration tests
 ├── assets/              # Images, fonts
 ├── package.json
 └── tsconfig.json
```

6.2 Next.js Web App

```
coffee-shop-web/
 ├── app/                 # App router
 │    ├── page.tsx        # Landing page
 │    ├── login/          # Login routes
 │    └── dashboard/      # Staff/customer dashboards
 ├── components/          # UI components
 ├── lib/                 # Supabase client, utils
 ├── __tests__/           # Tests
 ├── public/              # Static assets
 ├── package.json
 └── tsconfig.json
```

---

7. Git & Version Control Guidelines

- Use **feature branches** (`feature/login`, `feature/payment`).
- Commit messages follow **Conventional Commits**:
    - `feat:` → new feature
    - `mobile:` → new mobile feature
    - `fix:` → bug fix
    - `test:` → adding tests
    - `docs:` → documentation only
- Pull Requests required for merging into `main`.
- All PRs must pass linting + test checks before merge.