Onboarding Guide

📌 Project Overview

SMSX is an SMS-centric API tool designed with future scalability in mind. It uses:

- **Hono** for lightweight web server routing.
- **Drizzle ORM** for type-safe SQL operations.
- **Zod + OpenAPI** for schema validation and documentation.
- **PostgreSQL** as the primary database.
- **Vitest** for testing.

⚙️ Prerequisites

Ensure the following are installed:

- Node.js ≥ 18
- `pnpm` package manager
- PostgreSQL database
- Optional: Docker (for DB container)

🏗️ Project Setup

1. Clone the repo

```
git clone <repo-url>
cd africoda-smsx
```

2. Install dependencies

```
pnpm install
```

3. Set up environment variables

```
cp .env.example .env
# Then edit `.env` with your credentials
```

4. Initialize the database

Ensure PostgreSQL is running, then run:

```
pnpm drizzle-kit push
```

5. Run in development

```
pnpm dev
```

The app will start on `http://localhost:9999` (or your configured `PORT`).

👨🏽‍💻 Development Tutorial

📁 Project Structure

```
src/
├── modules/        # Feature-based modules (auth, notifications, etc.)
│   └── notifications/
│       ├── controller.ts
│       ├── service.ts
│       ├── routes.ts
│       └── schema.ts
├── lib/            # App-wide helper setup (e.g., router config, OpenAPI setup)
├── shared/         # Reusable constants, errors, types, and middleware
├── routes/         # Aggregated route entry points
├── db/             # Database schema and setup
└── utils/          # Utility functions (e.g., error handling)
```

🚀 Creating a New Feature Module

Let’s walk through adding a new `tasks` feature as documented in `doc.md`.

1. Create Service Logic (`src/modules/tasks/service.ts`)

```
export const getTasks = async () => {
  // Your DB logic here
  return [{ id: 1, title: "Test Task" }];
};
```

2. Controller (`controller.ts`)

```
import { getTasks } from "./service";

export const listTasks = (c) => {
  const tasks = await getTasks();
  return c.json({ tasks });
};
```

3. Define Route Schema (`routes.ts`)

```
import { createRoute } from "@hono/zod-openapi";
import { z } from "zod";

export const list = createRoute({
  method: "get",
  path: "/tasks",
  tags: ["Tasks"],
  responses: {
    200: {
      content: {
        "application/json": {
          schema: z.object({
            tasks: z.array(z.object({ id: z.number(), title: z.string() })),
          }),
        },
```

```
      },
    },
  },
});
```

4. Combine Route + Controller (`index.ts`)

```
import { createRouter } from "@/lib/create-app";
import * as routes from "./routes";
import * as controller from "./controller";

const router = createRouter()
  .openapi(routes.list, controller.listTasks);

export default router;
```

5. Register Route in Main Router (`src/routes/index.route.ts`)

```
import taskRoutes from "@/modules/tasks";

const router = createRouter();
router.route("/api", taskRoutes);
export default router;
```

🧪 Running Tests

```
pnpm test
```

Test files are in `src/tests/`. You can write new tests using Vitest.

📚 API Docs

When you use `@hono/zod-openapi`, your routes are automatically spec-compliant. You can integrate `@scalar/hono-api-reference` to expose live docs easily.

📐 Code Quality

- Lint with: `pnpm lint`
- Format code with: `pnpm lint:fix`
- Type check with: `pnpm typecheck`

🛠️ Tech Summary

Tool/Lib	Purpose
Hono	Web server framework
Zod + OpenAPI	Schema validation and docs
Drizzle ORM	DB schema and migrations
PostgreSQL	Persistent storage
Vitest	Unit testing
Pino + hono-pino	Logging middleware
dotenv	Environment variable management

🌍 Contributing Guidelines

- Follow feature-module pattern (`modules/<feature>`).
- Keep controller-service separation.
- Write tests for every route and service logic.
- Update OpenAPI schema when endpoints are added.
- Commit using conventional commits (`feat:`, `fix:`, etc.).
