EMAILX API Design & Concepts Guide

What Is an API?

An **API (Application Programming Interface)** is a contract that defines how software components communicate. In EMAILX, the **API is the core interface** between external consumers (apps, services, dashboards) and your internal logic (auth, notifications, scheduling, etc.).

Think of it as:

- A structured **doorway into your system**.
- A **standardized way** to expose functionalities like "send SMS", "authenticate user", etc.
- **Platform-agnostic** – APIs can be consumed by any device, client, or service.

📦 How APIs Are Structured in EMAILX

🔄 High-Level Flow

```
[Client Request]
    ↓
[Router (OpenAPI Hono Route)]
    ↓
[Controller or Handler] → (Validates / Forwards)
    ↓
[Service Layer] → (Business Logic / DB)
    ↓
[Response Returned]
```

🗂️ Folder Mapping

Layer	Path	Purpose
Router	`src/modules/*/routes.ts`	Defines the method, path, and OpenAPI metadata
Controller	`src/modules/*/controller.ts` or `handlers.ts`	Performs data handling and invokes services
Services	`src/modules/*/service.ts`	Executes logic like DB queries, 3rd-party calls
Schema	`src/modules/*/schema.ts` or `shared/schemas`	Zod schemas for request/response validation
Router Setup	`lib/create-app.ts`, `routes/index.route.ts`	Sets up the Hono app, loads all module routers

⚙️ Building an API in EMAILX

Example: `GET /notifications`

1. Define Schema (`notifications/schema.ts`)

```
export const NotificationResponse = z.object({
  id: z.string(),
  message: z.string(),
  sent: z.boolean(),
});
```

2. Define Route (`routes.ts`)

```
export const getNotifications = createRoute({
  method: "get",
  path: "/notifications",
  tags: ["Notifications"],
  responses: {
    200: jsonContent(
      z.array(NotificationResponse),
      "List of notifications"
    ),
```

```
  },
});
```

3. Controller (`controller.ts`)

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

export const getAll = async (c) => {
  const data = await listNotifications();
  return c.json(data);
};
```

4. Hook It Up

```
const router = createRouter();
router.openapi(routes.getNotifications, controller.getAll);
export default router;
```

🧠 Concepts Developers Must Master

1. HTTP Basics

- **Methods**: `GET`, `POST`, `PATCH`, `DELETE`—correspond to CRUD operations.
- **Status Codes**: `200`, `201`, `400`, `401`, `404`, `500`—communicate result intent.
- **Headers**: Used for auth (`Authorization`), content negotiation, etc.

2. RESTful Design Principles

- **Resources**: Model data as nouns (`/notifications`, `/users`)
- **Statelessness**: No session memory between requests.
- **Idempotency**: Ensure `GET`, `DELETE`, and `PUT` produce predictable outcomes.

3. Schema Validation (Zod)

Used to define both request and response structures—ensures:

- Clients send the correct format.
- You return consistent data.

4. OpenAPI Integration

- `@hono/zod-openapi` converts routes and schemas into machine-readable docs.
- Makes it easier to:
  - Auto-generate client SDKs.
  - Provide live API documentation.
  - Validate requests/responses.

5. Middlewares

Used to:

- Enforce auth (`auth.middleware.ts`)
- Log requests (`pino-logger.ts`)
- Handle errors (`BaseError.ts`, `NotificationError.ts`)

6. Error Handling

Structured error types ensure consistent client communication.

```
return c.json({ error: "Unauthorized" }, 401);
```

Use classes in `shared/errors` to model specific exception types.

7. Testing (Vitest)

Ensure APIs behave correctly with integration/unit tests:

- Test endpoint responses (`/tests/notifications.test.ts`)
- Validate edge cases (bad input, unauthorized access, etc.)

🗃️ Future-Proof Practices in EMAILX APIs

- 💡 **Schema-first** design: Start with Zod + OpenAPI.

- 📚 **Docs as Code**: Self-documenting APIs through metadata.
- 🛡️ **Security built-in**: JWT, middleware validation, safe defaults.
- 🔄 **Modularization**: Each feature is in its own folder, making scaling smooth.
- ♻️ **Reusable utilities** in `lib/` and `shared/` ensure DRY code.

📚 Recommended Reading

Topic	Resource
HTTP & REST	MDN Web Docs - HTTP
OpenAPI	OpenAPI Spec Overview
Zod	Zod Docs
Hono	Hono Documentation
Drizzle ORM	Drizzle ORM Docs
JWT	JWT Introduction

Would you like me to extract this into a Markdown format for the repo, or include a section on “API Versioning & Deprecation Strategy” next?