Multi-Tenant SaaS API Endpoint Structure Guide

This document provides recommended endpoint structures for a multi-tenant SaaS application, enabling secure and organized access to tenant-specific resources.

1. Key Principles

- Isolate tenants by namespace in API endpoints.
- Support both path and subdomain tenant identification patterns.
- Follow RESTful conventions where appropriate.

2. Endpoint Naming Patterns

2.1. Path-based Tenant Identification

```
POST   /api/{tenantId}/users
GET    /api/{tenantId}/projects/{projectId}
PUT    /api/{tenantId}/settings
```

2.2. Subdomain-based Tenant Identification

```
POST   https://{tenant}.api.example.com/users
GET    https://{tenant}.api.example.com/projects/{projectId}
PUT    https://{tenant}.api.example.com/settings
```

Choose the approach best suited for your application and infrastructure.

3. Sample Resource Definitions

Resource	Endpoint Example	Description
User List	`GET /api/{tenantId}/users`	Retrieve all users for a tenant
Create Project	`POST /api/{tenantId}/projects`	Create a new project for a tenant
Project Detail	`GET /api/{tenantId}/projects/{projectId}`	Retrieve a specific project
Organization Settings	`GET /api/{tenantId}/settings`	Get or update tenant settings

4. Authentication & Authorization

- All endpoints require authentication (e.g., bearer token in the `Authorization` header).
- Ensure that tokens are scoped to specific tenants to prevent cross-tenant access.

5. Error Responses

- Use standard HTTP response codes: `401 Unauthorized`, `403 Forbidden`, `404 Not Found`, etc.
- Return tenant-scoped error messages when applicable.

6. Example: Get Users for Tenant

```
GET /api/acme-corp/users
Authorization: Bearer <token>
```

```
{
  "users": [
    {"id": "u1", "name": "John Doe"},
    {"id": "u2", "name": "Jane Smith"}
  ]
}
```

7. Versioning (Optional)

```
GET /api/v1/{tenantId}/users
```

Version endpoints to ensure backward compatibility.