**API Documentation for Multi-Tenant SaaS Platform**

**1. API Introduction**

This document provides detailed API documentation for the Multi-Tenant SaaS Platform developed using .NET Core Web API. The platform supports multiple tenants, where each tenant can have its own set of users and can perform CRUD operations. Authentication and authorization are handled through JWT tokens and roles such as 'Admin' and 'User'.

Additionally, the system includes:

* **MasterTenantDb**: A central database storing the information about registered tenants, including their connection strings.
* **Isolated Tenant Databases**: A separate database for each tenant, created dynamically upon tenant registration.
* **Serilog Framework**: Implemented for centralized logging.

**2. Application Workflow**

1. **Tenant Management**:
   * The **Master Admin** uses the Tenant APIs to register, update, deactivate, and manage tenants.
   * Upon registration, a new isolated database is created for the tenant using the naming convention: Tenant\_<NameOfDatabase>, where NameOfDatabase is replaced by the name provided during registration.
   * The tenant's connection string is stored in the MasterTenantDb.
2. **Tenant Authentication**:
   * Tenants log in using their email and password via the /api/tenant/login API.
   * Upon successful login, a bearer token is generated. This token includes the TenantId, enabling access to the specific tenant’s isolated database.
3. **User Management**:
   * User CRUD operations are performed on the isolated database corresponding to the logged-in tenant.
   * The connection string for the specific tenant database is retrieved using the TenantId from the token.
   * Role-based access control ensures proper permissions within each tenant.
4. **Logging**:
   * Logging is implemented using the **Serilog framework** for tracking tenant-specific activities and ensuring centralized log management.

**3. Tenant Controller**

**3.1 Overview**

The Tenant Controller manages tenant-related operations such as registration, update, deactivation, login, and deletion.

**3.2 API Endpoints**

**POST /api/tenant/register**  
Registers a new tenant in the system. If the tenant name is valid and doesn't already exist, the tenant is registered successfully, and an isolated database is created.

**Request Body:**

**{**

**"tenantName": "string",**

**"adminEmail": "string",**

**"adminPasswordHash": "string",**

**"adminPhoneNumber": 0,**

**"adminFullName": "string"**

**}**

**Responses:**

* 200 OK: Tenant registered successfully.
* 400 Bad Request: Tenant name is required.
* 409 Conflict: Tenant with the same name already exists.

**Example:**

POST /api/tenant/register

{

"tenantName": "AGCE",

"adminEmail": "agce@gmail.com",

"adminPasswordHash": "agce@123",

"adminPhoneNumber": 1234567899,

"adminFullName": "agce"

}

**PUT /api/tenant/update/{tenantId}**  
Updates the details of a tenant based on the provided tenant ID.

**Request Body:**

**{**

**"tenantName": "string",**

**"isActive": true,**

**"adminEmail": "string",**

**"adminPasswordHash": "string",**

**"adminPhoneNumber": 0,**

**"adminFullName": "string"**

**}**

**Responses:**

* 200 OK: Tenant updated successfully.
* 400 Bad Request: Tenant name is required.

**Example:**

PUT /api/tenant/update/1

{

"tenantName": "agce",

"isActive": true,

"adminEmail": "agce@gmail.com",

"adminPasswordHash": "agce@123",

"adminPhoneNumber": 9876543121,

"adminFullName": "agce2"

}

**PUT /api/tenant/deactivate/{tenantId}**  
Deactivates a tenant based on the provided tenant ID.

**Responses:**

* 200 OK: Tenant deactivated successfully.

**Example:**

PUT /api/tenant/deactivate/1

**POST /api/tenant/login**  
Allows the tenant admin to log in using admin credentials. A JWT token is returned upon successful login.

**Request Body:**

{

"AdminEmail": "string",

"AdminPassword": "string"

}

**Responses:**

* 200 OK: JWT token returned.
* 401 Unauthorized: Invalid credentials.

**Example:**

POST /api/tenant/login

{

"AdminEmail": "admin@example.com",

"AdminPassword": "password123"

}

**DELETE /api/tenant/delete/{id}**  
Deletes a tenant based on the provided tenant ID.

**Responses:**

* 200 OK: Tenant deleted successfully.
* 500 Internal Server Error: Invalid tenant ID.

**Example:**

DELETE /api/tenant/delete/1

**4. User Controller**

**4.1 Overview**

The User Controller manages user-related operations such as user registration, update, deletion, and retrieval of user details within a tenant's isolated database.

**4.2 API Endpoints**

**POST /api/user/register**  
Registers a new user in the tenant’s isolated database. The tenant ID is extracted from the JWT token.

**Request Body:**

**{**

**"userName": "string",**

**"email": "string",**

**"password": "string",**

**"role": "string"**

**}**

**Responses:**

* 200 OK: User created successfully.
* 401 Unauthorized: Tenant ID missing in token.

**Example:**

POST /api/user/register

{

"UserName": "JohnDoe",

"Email": "johndoe@example.com",

"Password": "password123",

“Role”: “Admin”

}

**PUT /api/user/update/{userId}**  
Updates user details within the tenant’s isolated database.

**Request Body:**

**{**

**"userName": "string",**

**"email": "string",**

**"role": "string",**

**"isActive": true**

**}**

**Responses:**

* 200 OK: User updated successfully.
* 401 Unauthorized: Tenant ID missing in token.

**Example:**

PUT /api/user/update/1

{

"userName": "UpdatedUserName",

"email": [updatedemail@example.com](mailto:updatedemail@example.com),

"role": "updatedrole"

"isActive": "true"

}

**DELETE /api/user/delete/{userId}**  
Deletes a user from the tenant’s isolated database.

**Responses:**

* 200 OK: User deleted successfully.
* 401 Unauthorized: Tenant ID missing in token.

**Example:**

DELETE /api/user/delete/1

**GET /api/user/getall**  
Retrieves a list of all users within the tenant’s isolated database.

**Responses:**

* 200 OK: List of all users.
* 401 Unauthorized: Tenant ID missing in token.

**Example:**

GET /api/user/getall

**GET /api/user/byid/{userId}**  
Fetches the details of a user by user ID.

**Responses:**

* 200 OK: User details.
* 401 Unauthorized: Tenant ID missing in token.
* 404 Not Found: User not found.

**Example:**

GET /api/user/byid/1