🔐 STEP 2: Authentication & RBAC - API Documentation

📋 Available Endpoints

Base URL

```
http://localhost:3000/api
```

🔓 Public Endpoints (No Authentication Required)

1. Login

POST `/auth/login`

Login to get JWT tokens.

Request Body:

```json
{
  "email": "admin@demo.ngosaas.com",
  "password": "Demo@123"
}
```

Response (200 OK):

```json
```

```json
{
 "user": {
  "id": "uuid",
  "email": "admin@demo.ngosaas.com",
  "firstName": "Demo",
  "lastName": "Admin",
  "role": "NGO_ADMIN",
  "isSuperAdmin": false,
  "tenant": {
   "id": "uuid",
   "name": "Demo NGO Foundation",
   "subdomain": "demo"
  }
 },
 "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
 "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
 "expiresIn": "15m"
}
```

Error Responses:

- `401 Unauthorized` - Invalid credentials
- `401 Unauthorized` - Account inactive/suspended/expired

---

2. Register New Tenant

POST `/auth/register`

Register a new NGO organization with admin user.

Request Body:

json

```json
{
  "email": "admin@neworg.com",
  "password": "SecurePass123!",
  "firstName": "John",
  "lastName": "Doe",
  "phone": "+91-9876543210",
  "organizationName": "New NGO Foundation",
  "subdomain": "new-ngo"
}
```

Response (201 Created):

json

```json
{
 "user": {
  "id": "uuid",
  "email": "admin@neworg.com",
  "firstName": "John",
  "lastName": "Doe",
  "role": "NGO_ADMIN",
  "isSuperAdmin": false,
  "tenant": {
   "id": "uuid",
   "name": "New NGO Foundation",
   "subdomain": "new-ngo"
  }
 },
 "accessToken": "...",
 "refreshToken": "...",
 "expiresIn": "15m"
}
```

Error Responses:

- `409 Conflict` - Email already registered
- `409 Conflict` - Subdomain already taken
- `400 Bad Request` - Invalid subdomain format

Business Logic:

- Creates new tenant with TRIAL status
- Assigns STARTER plan by default
- Creates admin user with NGO_ADMIN role

- Trial period: 14 days (configurable)
- Auto-login after registration

---

3. Refresh Token

POST `/auth/refresh`

Get new access token using refresh token.

Request Body:

```json
{
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Response (200 OK):

```json
{
  "accessToken": "new_access_token...",
  "refreshToken": "new_refresh_token...",
  "expiresIn": "15m"
}
```

Error Responses:

- `401 Unauthorized` - Invalid or expired refresh token

🔒 Protected Endpoints (Authentication Required)

4. Get Current User Profile

GET `/auth/me`

Get authenticated user's profile information.

Headers:

```
Authorization: Bearer {accessToken}
```

Response (200 OK):

```json
{
  "userId": "uuid",
  "email": "admin@demo.ngosaas.com",
  "tenantId": "uuid",
  "role": "NGO_ADMIN",
  "isSuperAdmin": false
}
```

---

5. Logout

POST `/auth/logout`

Invalidate refresh token and logout.

Headers:

```
Authorization: Bearer {accessToken}
```

Response (200 OK):

```json
{
  "message": "Logged out successfully"
}
```

---

6. Change Password

POST `/auth/change-password`

Change user's password.

Headers:

```
Authorization: Bearer {accessToken}
```

Request Body:

```json
{
  "currentPassword": "OldPassword123!",
  "newPassword": "NewPassword456!"
}
```

Response (200 OK):

```json
{
  "message": "Password changed successfully. Please login again."
}
```

Error Responses:

- `401 Unauthorized` - Current password is incorrect
- `400 Bad Request` - New password doesn't meet requirements

🧪 Testing with cURL

Login

```bash
curl -X POST http://localhost:3000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "admin@demo.ngosaas.com",
    "password": "Demo@123"
  }'
```

Register New Tenant

```bash
```

```bash
curl -X POST http://localhost:3000/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "email": "admin@myngo.com",
    "password": "MySecure123!",
    "firstName": "Jane",
    "lastName": "Smith",
    "phone": "+91-9876543210",
    "organizationName": "My NGO",
    "subdomain": "myngo"
  }'
```

Get Profile (Protected)

bash

```bash
curl -X GET http://localhost:3000/api/auth/me \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN"
```

Refresh Token

bash

```bash
curl -X POST http://localhost:3000/api/auth/refresh \
  -H "Content-Type: application/json" \
  -d '{
    "refreshToken": "YOUR_REFRESH_TOKEN"
  }'
```

Logout

```bash
curl -X POST http://localhost:3000/api/auth/logout \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN"
```

🔐 JWT Token Structure

Access Token Payload

```json
{
  "sub": "user-uuid",
  "email": "user@example.com",
  "tenantId": "tenant-uuid",
  "roleId": "role-uuid",
  "roleName": "NGO_ADMIN",
  "isSuperAdmin": false,
  "iat": 1234567890,
  "exp": 1234567890
}
```

Token Lifecycle

- **Access Token**: Expires in 15 minutes (configurable)
- **Refresh Token**: Expires in 7 days (configurable)
- Refresh tokens are stored in database

- Logout invalidates the refresh token

🛡️ Security Features

Password Requirements

- Minimum 8 characters
- Must contain uppercase letter (configurable)
- Must contain lowercase letter (configurable)
- Must contain number (configurable)
- Must contain special character (configurable)

Protection Mechanisms

1. **JWT Authentication**: All routes protected by default
2. **Tenant Isolation**: Users can only access their tenant's data
3. **Role-Based Access**: Permissions checked based on roles
4. **Super Admin Bypass**: Platform admins can access all tenants
5. **Account Status Checks**: Inactive/suspended accounts blocked
6. **Refresh Token Validation**: Stored tokens must match
7. **Password Hashing**: bcrypt with salt rounds

🎭 User Roles

Available Roles

1. **NGO_ADMIN** - Full access to organization
2. **PROJECT_MANAGER** - Manage projects & beneficiaries
3. **FINANCE_MANAGER** - Manage finances & donors
4. **FIELD_STAFF** - Read-only access
5. **DONOR** - View reports only

Super Admin

- Email: `admin@ngosaas.com`
- Password: `SuperAdmin@123`
- Can access all tenants
- Bypasses tenant isolation
- Platform-level access

🚨 Error Handling

Standard Error Response Format

```json
{
  "statusCode": 401,
  "message": "Invalid email or password",
  "error": "Unauthorized"
}
```

Common HTTP Status Codes

- `200 OK` - Success
- `201 Created` - Resource created
- `400 Bad Request` - Invalid input
- `401 Unauthorized` - Authentication failed
- `403 Forbidden` - Access denied
- `404 Not Found` - Resource not found
- `409 Conflict` - Resource already exists
- `500 Internal Server Error` - Server error

🧩 Integration Example (Frontend)

React + Axios Example

```
typescript
```

```
import axios from 'axios';

const API_BASE_URL = 'http://localhost:3000/api';

// Create axios instance
const api = axios.create({
  baseURL: API_BASE_URL,
});

// Add request interceptor to attach token
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('accessToken');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Add response interceptor to handle token refresh
api.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;

    if (error.response?.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;

      try {
        const refreshToken = localStorage.getItem('refreshToken');
        const response = await axios.post(`${API_BASE_URL}/auth/refresh`, {
          refreshToken,
        });
        const { accessToken } = response.data;
        localStorage.setItem('accessToken', accessToken);

        originalRequest.headers.Authorization = `Bearer ${accessToken}`;
        return api(originalRequest);
      } catch (refreshError) {
        // Redirect to login
        localStorage.clear();
        window.location.href = '/login';
        return Promise.reject(refreshError);
      }
    }

    return Promise.reject(error);
  }
);

// Auth functions
export const authService = {
  async login(email: string, password: string) {
    const response = await api.post('/auth/login', { email, password });
    const { accessToken, refreshToken } = response.data;
    localStorage.setItem('accessToken', accessToken);
    localStorage.setItem('refreshToken', refreshToken);
    return response.data;
  },

  async register(data: RegisterDto) {
    const response = await api.post('/auth/register', data);
    const { accessToken, refreshToken } = response.data;
    localStorage.setItem('accessToken', accessToken);
    localStorage.setItem('refreshToken', refreshToken);
    return response.data;
  },

  async logout() {
    await api.post('/auth/logout');
    localStorage.clear();
  },

  async getProfile() {
    const response = await api.get('/auth/me');
    return response.data;
  },
};

export default api;
```

✅ Testing Checklist

Before moving to STEP 3, verify:

- [ ] Login with demo credentials works
- [ ] Login with super admin works
- [ ] Register new tenant succeeds
- [ ] Duplicate email registration fails
- [ ] Duplicate subdomain registration fails
- [ ] Access token expires after 15 minutes
- [ ] Refresh token works
- [ ] Logout invalidates refresh token

- [ ] Protected routes require authentication
- [ ] Invalid token returns 401
- [ ] Tenant isolation is enforced
- [ ] Super admin can bypass tenant isolation
- [ ] Role-based access works
- [ ] Password change works
- [ ] Password validation works

🐛 Common Issues & Solutions

Issue: "Cannot connect to database"

Solution: Ensure PostgreSQL is running and DATABASE_URL is correct

Issue: "JWT malformed"

Solution: Check JWT_SECRET and JWT_REFRESH_SECRET are set in .env

Issue: "Token has expired"

Solution: Use refresh token endpoint to get new access token

Issue: "Tenant account is suspended"

Solution: Update tenant status in database or contact super admin

Issue: "Role not found during seeding"

Solution: Run `npm run prisma:seed` to create default roles

🚀 Next Steps

STEP 2 is complete! ✅

You now have:

- ✅ Complete authentication system
- ✅ JWT token management
- ✅ Multi-tenant isolation
- ✅ Role-based access control
- ✅ Super admin functionality
- ✅ Secure password handling

Ready for STEP 3?

Next we'll implement:

- Subscription & billing engine with Razorpay
- Payment webhooks
- Invoice generation
- Feature gating
- Usage tracking

Reply with **"Proceed to STEP 3"** when ready!