Authentication API Documentation

This document describes the authentication system implemented in the SSVproff API.

Overview

The API uses JWT (JSON Web Tokens) for authentication. Users can register, log in, and access protected endpoints using access tokens. Refresh tokens allow users to obtain new access tokens without re-authenticating.

Authentication Flow

1. **Registration**: New users register with email, username, and password
2. **Login**: Users authenticate with email and password to receive JWT tokens
3. **Access Protected Routes**: Use access token in Authorization header
4. **Token Refresh**: Use refresh token to get new access token without re-login

Security Features

- **Password Hashing**: Passwords are hashed using bcrypt with automatic salt generation
- **JWT Tokens**: Secure token-based authentication with expiration
- **Token Types**: Separate access and refresh tokens with different expiration times
- **Field Validation**: Strict validation on email, username, and password fields
- **Database Constraints**: Unique constraints on email and username

API Endpoints

POST /api/v1/auth/register

Register a new user account.

Request Body:

```json
{
  "email": "user@example.com",
  "username": "johndoe",
  "password": "securepassword123"
}
```

Response (201 Created):

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "email": "user@example.com",
  "username": "johndoe",
  "is_active": true,
  "is_superuser": false,
  "created_at": "2024-01-01T00:00:00",
  "updated_at": "2024-01-01T00:00:00"
}
```

Validation Rules:
- Email: Valid email format
- Username: 3-50 characters, alphanumeric, underscore, hyphen only
- Password: Minimum 8 characters

Error Responses:
- `400 Bad Request` : Email or username already exists
- `422 Unprocessable Entity` : Validation error

POST /api/v1/auth/login

Authenticate and receive JWT tokens.

Request Body:

```
{
  "email": "user@example.com",
  "password": "securepassword123"
}
```

Response (200 OK):

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "bearer"
}
```

Error Responses:
- `401 Unauthorized` : Incorrect email or password
- `400 Bad Request` : User account is inactive

GET /api/v1/auth/me

Get current authenticated user information. **Requires authentication.**

Headers:

```
Authorization: Bearer <access_token>
```

Response (200 OK):

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "email": "user@example.com",
  "username": "johndoe",
  "is_active": true,
  "is_superuser": false,
  "created_at": "2024-01-01T00:00:00",
  "updated_at": "2024-01-01T00:00:00"
}
```

Error Responses:
- `401 Unauthorized`: Missing, invalid, or expired token

POST /api/v1/auth/refresh

Get a new access token using a refresh token.

Request Body:

```
{
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Response (200 OK):

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "bearer"
}
```

Error Responses:
- `401 Unauthorized`: Invalid or expired refresh token

Token Details

Access Token

- **Purpose**: Access protected API endpoints
- **Expiration**: 30 minutes (configurable via `ACCESS_TOKEN_EXPIRE_MINUTES`)
- **Type**: JWT with “access” type claim

Refresh Token

- **Purpose**: Obtain new access tokens without re-login
- **Expiration**: 7 days
- **Type**: JWT with “refresh” type claim

Token Payload Structure

```json
{
  "sub": "user-id-uuid",
  "exp": 1234567890,
  "iat": 1234567890,
  "type": "access"
}
```

Using Protected Endpoints

To access protected endpoints, include the access token in the Authorization header:

```bash
curl -H "Authorization: Bearer <access_token>" \
  http://localhost:8000/api/v1/auth/me
```

Error Handling

All authentication endpoints follow consistent error response format:

```json
{
  "detail": "Error message describing the issue"
}
```

Common HTTP status codes:
- `200 OK` : Success
- `201 Created` : Resource created successfully
- `400 Bad Request` : Invalid input or business logic error
- `401 Unauthorized` : Authentication failure
- `403 Forbidden` : Insufficient permissions
- `422 Unprocessable Entity` : Validation error

Database Schema

Users Table

Column	Type	Constraints	Description
id	UUID	PRIMARY KEY, NOT NULL	Unique identifier
email	VARCHAR(255)	UNIQUE, NOT NULL, INDEXED	User email
username	VARCHAR(50)	UNIQUE, NOT NULL, INDEXED	Username
hashed_password	VARCHAR(255)	NOT NULL	Bcrypt hashed password
is_active	BOOLEAN	NOT NULL, DEFAULT TRUE	Account status
is_superuser	BOOLEAN	NOT NULL, DEFAULT FALSE	Admin privileges
created_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Creation timestamp
updated_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Update timestamp

Indexes

- `ix_users_id`: Primary key index
- `ix_users_email`: Email index for fast lookups
- `ix_users_username`: Username index for fast lookups
- `ix_users_email_active`: Composite index for active user queries
- `ix_users_username_active`: Composite index for active user queries

Security Best Practices

1. **Environment Variables**: Store `SECRET_KEY` securely
2. **Secret Key**: Use strong random key (minimum 32 characters)
3. **HTTPS**: Always use HTTPS in production
4. **Token Storage**: Store tokens securely on client side
5. **Token Expiration**: Implement token refresh flow
6. **Password Policy**: Enforce strong password requirements
7. **Rate Limiting**: Implement rate limiting on auth endpoints (recommended)

Configuration

Key environment variables for authentication:

```
# Security
SECRET_KEY=your-secret-key-change-in-production
ACCESS_TOKEN_EXPIRE_MINUTES=30

# Database
DATABASE_URL=postgresql://user:password@localhost:5432/db_name
```

Generate a secure secret key:

```
openssl rand -hex 32
```

Example Integration

Python Client Example

```
import requests

# Register
response = requests.post(
    "http://localhost:8000/api/v1/auth/register",
    json={
        "email": "user@example.com",
        "username": "johndoe",
        "password": "securepassword123"
    }
)
user = response.json()

# Login
response = requests.post(
    "http://localhost:8000/api/v1/auth/login",
    json={
        "email": "user@example.com",
        "password": "securepassword123"
    }
)
tokens = response.json()
access_token = tokens["access_token"]

# Access protected endpoint
response = requests.get(
    "http://localhost:8000/api/v1/auth/me",
    headers={"Authorization": f"Bearer {access_token}"}
)
current_user = response.json()
```

JavaScript/TypeScript Example

```javascript
// Register
const registerResponse = await fetch('http://localhost:8000/api/v1/auth/register', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    email: 'user@example.com',
    username: 'johndoe',
    password: 'securepassword123'
  })
});
const user = await registerResponse.json();

// Login
const loginResponse = await fetch('http://localhost:8000/api/v1/auth/login', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    email: 'user@example.com',
    password: 'securepassword123'
  })
});
const tokens = await loginResponse.json();

// Access protected endpoint
const meResponse = await fetch('http://localhost:8000/api/v1/auth/me', {
  headers: { 'Authorization': `Bearer ${tokens.access_token}` }
});
const currentUser = await meResponse.json();
```