Authentication API Functions

Technical Manual - API Reference

SONARWORKS WORKFLOW SYSTEM

Version 1.1

Table of Contents

1. Introduction

This manual documents the Authentication API functions for the Sonar Workflow System. These APIs enable client applications to authenticate users, manage sessions, and handle password operations.

API Characteristics:

- RESTful architecture
- JSON request/response format
- JWT (JSON Web Token) authentication
- HTTPS required for all endpoints

1.1 Base URL

[URL]
```
https://your-server.com/api
```

1.2 Content Type

[HTTP Headers]
```
Content-Type: application/json
Accept: application/json
```

2. Authentication Overview

2.1 Authentication Flow

1. Client sends credentials to /api/auth/login
2. Server validates credentials and returns JWT token
3. Client stores token securely
4. Client includes token in Authorization header for subsequent requests
5. Token is refreshed before expiration or upon 401 response
6. Client calls /api/auth/logout to invalidate token

2.2 JWT Token Structure

The JWT token consists of three parts: Header, Payload, and Signature.

[JSON]
```
{
  "header": {
    "alg": "HS256",
    "typ": "JWT"
  },
  "payload": {
    "sub": "user-uuid",
    "username": "john.doe",
    "roles": ["STAFF", "INITIATOR"],
    "iat": 1705320000,
    "exp": 1705348800
  }
}
```

2.3 Using the Token

[HTTP Header]

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

3. API Endpoints Summary

Method	Endpoint	Description	Auth Required
POST	/api/auth/login	Authenticate user	No
POST	/api/auth/logout	Invalidate session	Yes
POST	/api/auth/refresh	Refresh JWT token	Yes
POST	/api/password/change	Change password	Yes
POST	/api/password/reset-request	Request password reset	No
POST	/api/password/reset-confirm	Confirm password reset	No

4. Login Function

4.1 Endpoint

[HTTP]

```
POST /api/auth/login
```

4.2 Request

Request Headers:

[HTTP Headers]

```
Content-Type: application/json
```

Request Body:

[JSON]

```
{
  "username": "john.doe",
  "password": "SecureP@ssw0rd"
}
```

Request Parameters:

Parameter	Type	Required	Description

username	string	Yes	User's login username
password	string	Yes	User's password

4.3 Response

Success Response (200 OK):

```
[JSON]
{
  "success": true,
  "message": "Login successful",
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "tokenType": "Bearer",
    "expiresIn": 28800,
    "user": {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "username": "john.doe",
      "email": "john.doe@company.com",
      "firstName": "John",
      "lastName": "Doe",
      "roles": ["STAFF", "INITIATOR"],
      "mustChangePassword": false
    }
  }
}
```

Error Response (401 Unauthorized):

```
[JSON]
{
  "success": false,
  "message": "Invalid credentials",
  "error": "INVALID_CREDENTIALS"
}
```

4.4 Error Codes

Error Code	HTTP Status	Description
INVALID_CREDENTIALS	401	Wrong username or password
ACCOUNT_LOCKED	403	Account is locked
ACCOUNT_DISABLED	403	Account is disabled
PASSWORD_EXPIRED	403	Password has expired
MUST_CHANGE_PASSWORD	403	Password change required

5. Logout Function

5.1 Endpoint

[HTTP]

```
POST /api/auth/logout
```

5.2 Request

[HTTP Headers]

```
Authorization: Bearer <token>
Content-Type: application/json
```

Request Body: Empty or not required

5.3 Response

[JSON]

```
{
  "success": true,
  "message": "Logout successful"
}
```

5.4 Behavior

- Token is invalidated on server
- Session is terminated
- Subsequent requests with same token will fail
- Audit log entry created

6. Token Refresh Function

6.1 Endpoint

[HTTP]

```
POST /api/auth/refresh
```

6.2 Request

[HTTP Headers]

```
Authorization: Bearer <current-token>
Content-Type: application/json
```

6.3 Response

[JSON]

```
{
  "success": true,
  "message": "Token refreshed",
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...(new token)",
    "tokenType": "Bearer",
    "expiresIn": 28800
  }
}
```

6.4 Usage Notes

- Call before token expires to maintain session
- Old token is invalidated upon refresh
- Use the new token for subsequent requests
- Refresh within token validity window

7. Password Functions

7.1 Change Password

Endpoint:

```
[HTTP]
POST /api/password/change
```

Request Headers:

```
[HTTP Headers]
Authorization: Bearer <token>
Content-Type: application/json
```

Request Body:

```
[JSON]
{
  "currentPassword": "OldP@ssw0rd",
  "newPassword": "NewSecureP@ss!",
  "confirmPassword": "NewSecureP@ss!"
}
```

Response:

```
[JSON]
{
  "success": true,
  "message": "Password changed successfully"
}
```

7.2 Request Password Reset

Endpoint:

[HTTP]
```
POST /api/password/reset-request
```

Request Body:

[JSON]
```
{
  "email": "john.doe@company.com"
}
```

Response:

[JSON]
```
{
  "success": true,
  "message": "If the email exists, a reset link has been sent"
}
```

NOTE: Response is always success to prevent email enumeration attacks.

7.3 Confirm Password Reset

Endpoint:

[HTTP]
```
POST /api/password/reset-confirm
```

Request Body:

[JSON]
```
{
  "token": "reset-token-from-email",
  "newPassword": "NewSecureP@ss!",
  "confirmPassword": "NewSecureP@ss!"
}
```

Response:

[JSON]
```
{
  "success": true,
  "message": "Password has been reset successfully"
}
```

8. Error Handling

8.1 Error Response Format

[JSON]
```
{
  "success": false,
  "message": "Human-readable error message",
  "error": "ERROR_CODE",
  "details": {
    "field": "Specific field error"
  },
  "timestamp": "2024-01-15T14:30:00Z"
}
```

8.2 Common HTTP Status Codes

Status	Meaning	Action
200	Success	Process response data
400	Bad Request	Check request format/parameters
401	Unauthorized	Login or refresh token
403	Forbidden	Insufficient permissions
404	Not Found	Resource doesn't exist
422	Validation Error	Check field validation errors
500	Server Error	Retry or contact support

8.3 Handling Token Expiration

[JavaScript]

```javascript
// Pseudo-code for handling 401 responses
async function apiCall(endpoint, options) {
  const response = await fetch(endpoint, options);

  if (response.status === 401) {
    // Try to refresh the token
    const refreshed = await refreshToken();
    if (refreshed) {
      // Retry the original request with new token
      return await fetch(endpoint, {
        ...options,
        headers: {
          ...options.headers,
          'Authorization': `Bearer ${newToken}`
        }
      });
    } else {
      // Refresh failed, redirect to login
      redirectToLogin();
    }
  }
  return response;
}
```

9. Security Best Practices

9.1 Token Storage

- Store tokens in secure, HTTP-only cookies when possible
- If using localStorage, be aware of XSS risks
- Never store tokens in URL parameters
- Clear tokens on logout

9.2 HTTPS Requirements

- All API calls must use HTTPS
- Reject HTTP connections
- Validate SSL certificates
- Use TLS 1.2 or higher

9.3 Token Handling

- Refresh tokens before expiration
- Handle token expiration gracefully
- Don't log tokens in plain text
- Invalidate tokens on password change

9.4 Rate Limiting

The API implements rate limiting to prevent abuse:

Endpoint	Limit	Window
/api/auth/login	5 attempts	15 minutes
/api/password/reset-request	3 attempts	1 hour
/api/password/change	5 attempts	1 hour

10. Code Examples

10.1 JavaScript/TypeScript Example

[TypeScript]

```
// Authentication Service
class AuthService {
  private baseUrl = 'https://api.yourserver.com/api';
  private token: string | null = null;

  async login(username: string, password: string):
Promise<LoginResponse> {
    const response = await fetch(`${this.baseUrl}/auth/login`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ username, password })
    });

    const data = await response.json();
    if (data.success) {
      this.token = data.data.token;
      localStorage.setItem('token', this.token);
    }
    return data;
  }

  async logout(): Promise<void> {
    await fetch(`${this.baseUrl}/auth/logout`, {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${this.token}`,
        'Content-Type': 'application/json'
      }
    });
    this.token = null;
    localStorage.removeItem('token');
  }

  getAuthHeader(): object {
    return { 'Authorization': `Bearer ${this.token}` };
  }
}
```

10.2 Python Example

[Python]

```python
import requests

class AuthClient:
    def __init__(self, base_url):
        self.base_url = base_url
        self.token = None

    def login(self, username, password):
        response = requests.post(
            f"{self.base_url}/api/auth/login",
            json={"username": username, "password": password}
        )
        data = response.json()
        if data.get("success"):
            self.token = data["data"]["token"]
        return data
    def logout(self):
        response = requests.post(
            f"{self.base_url}/api/auth/logout",
            headers=self.get_auth_header()
        )
        self.token = None
        return response.json()

    def get_auth_header(self):
        return {"Authorization": f"Bearer {self.token}"}

# Usage
client = AuthClient("https://api.yourserver.com")
result = client.login("john.doe", "password123")
print(result)
```

10.3 cURL Examples

[Bash]

```
# Login
curl -X POST https://api.yourserver.com/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"john.doe","password":"password123"}'

# Use token for authenticated request
curl -X GET https://api.yourserver.com/api/workflows \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..."

# Logout
curl -X POST https://api.yourserver.com/api/auth/logout \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..."
```