NGO Accounting System - API Security & Documentation Guide

Overview

This document provides comprehensive information about the NGO Accounting System's enhanced API security features, authentication mechanisms, and endpoint documentation.

Table of Contents

Security Features

1. Enhanced Authentication

- **JWT-based authentication** with access and refresh tokens
- **Two-Factor Authentication (2FA)** using TOTP
- **Password strength validation** with complexity requirements
- **Account lockout** after failed login attempts
- **Password reset** with secure token-based flow

2. Request Security

- **Input validation** using Marshmallow schemas
- **XSS protection** with content sanitization
- **SQL injection prevention** through ORM usage
- **CSRF protection** for state-changing operations
- **Content-Type validation** and payload size limits

3. Infrastructure Security

- **Rate limiting** per endpoint and IP address
- **CORS configuration** with whitelist origins
- **Security headers** (HSTS, CSP, X-Frame-Options, etc.)
- **TLS/SSL enforcement** in production
- **Session security** with secure cookies

4. Data Protection

- **Audit logging** for all sensitive operations
- **Data encryption** for sensitive fields
- **Secure password hashing** using bcrypt
- **Token blacklisting** for logout functionality

Authentication & Authorization

JWT Token Structure

```json
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",
  "refresh_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",
  "expires_in": 28800,
  "user": {
    "id": 1,
    "username": "admin",
    "role_name": "Administrator",
    "permissions": ["*"]
  }
}
```

Authentication Flow

1. **Login**: POST `/api/v1/auth/login`
2. **Token Refresh**: POST `/api/v1/auth/refresh`
3. **Logout**: POST `/api/v1/auth/logout`

2FA Setup Flow

1. **Enable 2FA**: POST `/api/v1/auth/enable-2fa`
2. **Verify Setup**: POST `/api/v1/auth/verify-2fa`

3. **Login with 2FA**: Include `totp_code` in login request

Permission System

The system uses role-based access control with granular permissions:

- `account_read`, `account_create`, `account_update`, `account_delete`
- `journal_read`, `journal_create`, `journal_update`, `journal_post`
- `reports_read`, `budget_read`, `grant_read`
- `system_admin` (for system-level operations)

API Endpoints

Authentication Endpoints

POST `/api/v1/auth/login`

Authenticate user and receive JWT tokens.

Request:

```json
{
  "username": "admin",
  "password": "password123",
  "totp_code": "123456"  // Optional, required if 2FA enabled
}
```

Response:

```json
{
  "access_token": "...",
  "refresh_token": "...",
  "expires_in": 28800,
  "user": { ... }
}
```

Rate Limit: 5 requests per minute per IP

POST `/api/v1/auth/change-password`

Change user password (requires authentication).

Request:

```json
{
  "current_password": "oldpassword",
  "new_password": "newpassword123"
}
```

Rate Limit: 3 requests per minute

Account Management

GET `/api/v1/accounts`

Get chart of accounts with pagination and filtering.

Query Parameters:

- `page`: Page number (default: 1)
- `per_page`: Items per page (default: 50, max: 100)
- `account_type`: Filter by account type
- `search`: Search in account name/code

Headers Required:

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

Response:

```json
{
  "accounts": [...],
  "total": 150,
  "pages": 3,
  "current_page": 1
}
```

POST `/api/v1/accounts`

Create a new account.

Required Permission: `account_create`

Request:

```json
{
  "code": "1111",
  "name": "Cash Account",
  "name_ar": "حساب النقد",
  "account_type": "asset",
  "parent_id": 10,
  "description": "Main cash account"
}
```

Journal Entries

GET `/api/v1/journal-entries`

Get journal entries with filtering options.

Query Parameters:

- `start_date`: Filter from date (YYYY-MM-DD)
- `end_date`: Filter to date (YYYY-MM-DD)
- `entry_type`: Filter by entry type
- `is_posted`: Filter by posted status

POST `/api/v1/journal-entries`

Create a new journal entry.

Required Permission: `journal_create`

Request:

```json
```

```json
{
  "entry_date": "2024-01-15",
  "description": "Cash receipt from donor",
  "currency_id": 1,
  "exchange_rate": 1.0,
  "lines": [
    {
      "account_id": 1,
      "debit_amount": 5000.00,
      "credit_amount": 0.00,
      "description": "Cash received",
      "project_id": 1
    },
    {
      "account_id": 10,
      "debit_amount": 0.00,
      "credit_amount": 5000.00,
      "description": "Grant revenue"
    }
  ]
}
```

Reports

GET `/api/v1/reports/trial-balance`

Generate trial balance report.

Query Parameters:

- `as_of_date`: Report date (YYYY-MM-DD)

GET `/api/v1/reports/income-statement`

Generate income statement.

Query Parameters:

- `start_date`: Period start date (required)
- `end_date`: Period end date (required)

Dashboard Analytics

GET `/api/v1/dashboard/overview`

Get comprehensive dashboard data.

Query Parameters:

- `start_date`: Analysis period start
- `end_date`: Analysis period end

Data Import/Export

POST `/api/v1/data-exchange/import/accounts`

Import chart of accounts from CSV/Excel file.

Request: Multipart form data with file upload

Response:

```json
{
  "message": "Accounts imported successfully",
  "imported_count": 25,
  "updated_count": 5,
  "total_processed": 30
}
```

GET `/api/v1/data-exchange/export/trial-balance`

Export trial balance data.

Query Parameters:

- `as_of_date`: Report date
- `format`: Export format (excel, csv, pdf)

Response: File download

Request Validation

Validation Rules

All API endpoints implement comprehensive validation:

1. **Field Validation:**
    - Required fields check

- Data type validation
- Length constraints
- Format validation (email, date, etc.)

2. **Business Logic Validation:**
   - Journal entries must balance (debits = credits)
   - Account codes must be unique
   - Date ranges must be logical
   - Permission checks
3. **Security Validation:**
   - XSS prevention through input sanitization
   - SQL injection prevention
   - File type validation for uploads
   - Content length limits

Example Validation Error Response

```json
{
  "message": "Validation failed",
  "errors": {
    "name": ["This field is required"],
    "email": ["Invalid email format"],
    "amount": ["Must be greater than 0"]
  }
}
```

Rate Limiting

Default Limits

- **General API**: 1000 requests per hour per IP
- **Authentication**: 5 login attempts per minute per IP
- **Password operations**: 3 requests per minute per user
- **Data export**: 10 requests per hour per user

Rate Limit Headers

```
X-RateLimit-Limit: 1000
X-RateLimit-Remaining: 999
X-RateLimit-Reset: 1640995200
```

Rate Limit Exceeded Response

```json
{
  "error": "Rate Limit Exceeded",
  "message": "Too many requests. Please try again later.",
  "status_code": 429
}
```

Error Handling

Standard Error Response Format

```json
{
  "error": "Error Type",
  "message": "Human-readable error message",
  "status_code": 400,
  "timestamp": "2024-01-15T10:30:00Z"
}
```

Common HTTP Status Codes

- `200 OK`: Success
- `201 Created`: Resource created successfully
- `400 Bad Request`: Invalid request data
- `401 Unauthorized`: Authentication required
- `403 Forbidden`: Insufficient permissions
- `404 Not Found`: Resource not found
- `409 Conflict`: Resource conflict (duplicate)
- `422 Unprocessable Entity`: Validation errors
- `429 Too Many Requests`: Rate limit exceeded
- `500 Internal Server Error`: Server error

Development Setup

Environment Variables

Create a `.env` file with the following variables:

```bash
# Flask Configuration
FLASK_ENV=development
SECRET_KEY=your-secret-key-here
JWT_SECRET_KEY=your-jwt-secret-key-here

# Database
DATABASE_URL=sqlite:///accounting_dev.db

# Email Configuration
MAIL_SERVER=smtp.gmail.com
MAIL_PORT=587
MAIL_USE_TLS=true
MAIL_USERNAME=your-email@domain.com
MAIL_PASSWORD=your-app-password

# Redis (for rate limiting and caching)
REDIS_URL=redis://localhost:6379/0

# CORS Origins
CORS_ORIGINS=http://localhost:3000,http://127.0.0.1:3000

# Security Settings
MAX_LOGIN_ATTEMPTS=5
ACCOUNT_LOCKOUT_DURATION=30
ENABLE_2FA=true
```

Installation and Setup

1. **Install dependencies:**

```bash
pip install -r requirements.txt
```

2. **Initialize database:**

```bash
python database_setup.py create
```

3. **Run the application:**

```bash
python app.py
```

Testing

Run the test suite:

```bash
pytest tests/ -v --cov=.
```

Production Deployment

Security Checklist

- [ ] Use HTTPS/TLS encryption
- [ ] Set strong `SECRET_KEY` and `JWT_SECRET_KEY`
- [ ] Configure proper CORS origins
- [ ] Enable security headers
- [ ] Set up proper database credentials
- [ ] Configure email service
- [ ] Set up Redis for caching and rate limiting
- [ ] Enable audit logging
- [ ] Configure backup strategy
- [ ] Set up monitoring and alerting

Environment Variables for Production

```bash
```

```
FLASK_ENV=production
SECRET_KEY=strong-random-secret-key
JWT_SECRET_KEY=strong-random-jwt-key
DATABASE_URL=postgresql://user:password@host:5432/dbname
REDIS_URL=redis://host:6379/0
CORS_ORIGINS=https://your-frontend-domain.com
SSL_REDIRECT=true
```

Docker Deployment

```dockerfile
FROM python:3.11-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]
```

Database Backup

The system includes automated backup functionality:

- Daily database backups
- Configurable retention period
- Backup verification
- Restore capabilities

Monitoring

Set up monitoring for:

- API response times
- Error rates
- Failed authentication attempts
- Database performance

- System resource usage

Best Practices

For Developers

1. **Always validate input** at the API layer
2. **Use proper HTTP status codes** for responses
3. **Implement proper error handling** with user-friendly messages
4. **Follow RESTful conventions** for endpoint design
5. **Document API changes** and maintain backward compatibility
6. **Write comprehensive tests** for all endpoints

For API Consumers

1. **Store JWT tokens securely** (never in localStorage for production)
2. **Implement proper error handling** for all API calls
3. **Respect rate limits** and implement retry logic
4. **Use HTTPS** for all API communications
5. **Validate data** before sending to API
6. **Handle token expiration** gracefully

Support and Contact

For technical support or questions about the API:

- Documentation: [Internal Wiki/Docs]
- Issue Tracking: [Your Issue Tracker]
- Email: [Technical Support Email]

Security Notice: This system handles sensitive financial data. Always follow security best practices and report any security concerns immediately.