Admin System & User Management Guide

🔐 Overview

The platform implements a secure admin-based user management system:

- **First user** automatically becomes admin
- **Subsequent users** can only be created by admins
- **Admin privileges** required for API key management
- **Token-based authentication** for all operations

🎯 Key Concepts

User Roles

Role	Can Do
Admin	Everything: create users, manage API keys, all regular features
Regular User	Use platform: create projects, chats, share with others

Admin Privileges

Admins have exclusive access to:

- ✅ Create new user accounts
- ✅ Add/remove AI provider API keys
- ✅ View all API keys and usage
- ✅ Enable/disable API keys
- ✅ All regular user features

🚀 Initial Setup Flow

Step 1: Create First Admin User

The first user registration requires NO authentication:

bash

```bash
# Register first user (becomes admin automatically)
curl -X POST http://localhost/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "admin",
    "password": "SecurePassword123!",
    "email": "admin@example.com"
  }'
```

Response:

json

```json
{
  "user_id": 1,
  "username": "admin",
  "email": "admin@example.com",
  "is_admin": true,
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Important: Save the token! You'll need it for all admin operations.

Step 2: Add AI Provider Keys (Admin Only)

```bash
# Use the admin token from registration
ADMIN_TOKEN="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

# Add OpenAI key
curl -X POST http://localhost/api/providers/keys \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "provider_id": 1,
    "key_value": "sk-your-openai-key-here"
  }'

# Add Anthropic key
curl -X POST http://localhost/api/providers/keys \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "provider_id": 2,
    "key_value": "sk-ant-your-anthropic-key-here"
  }'
```

Step 3: Create Additional Users (Admin Only)

```bash
# Create a regular user (requires admin token)
curl -X POST http://localhost/api/auth/register \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "username": "john",
    "password": "UserPassword123!",
    "email": "john@example.com"
  }'
```

Response:

```json
{
  "user_id": 2,
  "username": "john",
  "email": "john@example.com",
  "is_admin": false,
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

🔑 Authentication Flow

User Login

```bash
# Login with username and password
curl -X POST http://localhost/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "admin",
    "password": "SecurePassword123!"
  }'
```

Response includes:

- `user_id` - User's unique ID
- `username` - Username
- `is_admin` - Admin status (true/false)
- `token` - JWT token for API requests

Get Current User Info

```bash
# Check your current user details
curl http://localhost/api/auth/me \
  -H "Authorization: Bearer $TOKEN"
```

Response:

```json
{
  "id": 1,
  "username": "admin",
  "email": "admin@example.com",
  "is_admin": true,
  "created_at": "2024-12-05T10:30:00.000Z"
}
```

👥 User Management

Creating Users

As Admin:

```bash
curl -X POST http://localhost/api/auth/register \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "username": "newuser",
    "password": "Password123!",
    "email": "user@example.com"
  }'
```

Without Admin Token:

```bash
curl -X POST http://localhost/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "hacker",
    "password": "Password123!"
  }'
```

Response (Error):

```json
{
  "error": "unauthorized",
  "message": "Authentication required to create new users"
}
```

Checking Admin Status

```bash
# Login returns is_admin flag
curl -X POST http://localhost/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"SecurePassword123!"}' \
  | jq '.is_admin'

# Output: true
```

🔧 API Key Management (Admin Only)

List All Providers

```bash
# Available to all authenticated users
curl http://localhost/api/providers \
  -H "Authorization: Bearer $TOKEN"
```

Response:

```json
{
  "providers": [
    {
      "id": 1,
      "name": "chatgpt",
      "display_name": "ChatGPT",
      "is_active": true,
      "available_keys": 2
    },
    {
      "id": 2,
      "name": "claude",
      "display_name": "Claude",
      "is_active": true,
      "available_keys": 1
    }
  ]
}
```

Add API Key (Admin Only)

bash

```
curl -X POST http://localhost/api/providers/keys \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "provider_id": 1,
    "key_value": "sk-proj-abc123..."
  }'
```

View All API Keys (Admin Only)

bash

```
curl http://localhost/api/providers/keys \
  -H "Authorization: Bearer $ADMIN_TOKEN"
```

Response:

json

```
{
  "keys": [
    {
      "id": 1,
      "provider_id": 1,
      "provider_name": "ChatGPT",
      "key_preview": "sk-proj-ab...",
      "usage_count": 42,
      "last_used_at": "2024-12-05T14:30:00.000Z",
      "status": "active",
      "rate_limit_reset_at": null,
      "created_at": "2024-12-01T10:00:00.000Z"
    }
  ]
}
```

Update API Key Status (Admin Only)

```bash
# Disable a key
curl -X PUT http://localhost/api/providers/keys/1 \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"status": "disabled"}'

# Re-enable a key
curl -X PUT http://localhost/api/providers/keys/1 \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"status": "active"}'
```

Delete API Key (Admin Only)

```bash
curl -X DELETE http://localhost/api/providers/keys/1 \
  -H "Authorization: Bearer $ADMIN_TOKEN"
```

🚫 Access Control Examples

Scenario 1: Regular User Tries to Add User

```bash
# Login as regular user
REGULAR_TOKEN=$(curl -s -X POST http://localhost/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"john","password":"UserPassword123!"}' \
  | jq -r '.token')

# Try to create another user
curl -X POST http://localhost/api/auth/register \
  -H "Authorization: Bearer $REGULAR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"username":"hacker","password":"Pass123!"}'
```

Response (Error):

```json
{
  "error": "forbidden",
  "message": "Only administrators can create new users"
}
```

Scenario 2: Regular User Tries to Add API Key

```bash
curl -X POST http://localhost/api/providers/keys \
  -H "Authorization: Bearer $REGULAR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"provider_id":1,"key_value":"sk-stolen-key"}'
```

Response (Error):

```json
{
  "error": "forbidden",
  "message": "Administrator access required"
}
```

Scenario 3: No Token Provided

```bash
curl -X POST http://localhost/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"hacker","password":"Pass123!"}'
```

Response (Error):

```json
{
  "error": "unauthorized",
  "message": "Authentication required to create new users"
}
```

🗄️ Database Schema

Users Table

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    email VARCHAR(100),
    is_admin BOOLEAN DEFAULT FALSE,  -- New field
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Checking Admin Users

```sql
-- List all admin users
SELECT id, username, email, created_at
FROM users
WHERE is_admin = true;

-- Count total users and admins
SELECT
  COUNT(*) as total_users,
  COUNT(*) FILTER (WHERE is_admin = true) as admin_users,
  COUNT(*) FILTER (WHERE is_admin = false) as regular_users
FROM users;
```

🔐 Security Best Practices

Password Requirements

The system enforces:

- ✅ Minimum 6 characters (increase in production)
- ✅ Bcrypt hashing with salt
- ✅ No plain text storage

Recommended for production:

```javascript
// Add to auth.js validation
if (!/(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/.test(password)) {
  throw new AppError(400, 'validation_error',
    'Password must contain uppercase, lowercase, and numbers');
}
```

Token Security

- ✅ JWT tokens expire after 7 days
- ✅ Tokens include user ID, username, and admin flag
- ✅ Tokens signed with secret key
- ✅ HTTPS required in production

Admin Account Protection

Recommendations:

1. **Strong password** for admin account
2. **Unique email** for admin
3. **Regular token rotation** (re-login periodically)
4. **Monitor admin actions** (add logging)
5. **Backup admin account** (create second admin)

Creating Backup Admin

```bash
# As existing admin, create another admin
# (Requires manual database update after creation)

# 1. Create user normally
curl -X POST http://localhost/api/auth/register \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"username":"backup_admin","password":"SecurePass123!"}'

# 2. Promote to admin (database command)
docker-compose exec postgres psql -U aiplatform -d aiplatform \
  -c "UPDATE users SET is_admin = true WHERE username = 'backup_admin';"
```

🧪 Testing Admin System

Test Script

bash

```bash
#!/bin/bash

echo "Testing Admin System..."

# 1. Register first user (should become admin)
echo "1. Creating first admin user..."
ADMIN_RESPONSE=$(curl -s -X POST http://localhost/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"testadmin","password":"Admin123!"}')

ADMIN_TOKEN=$(echo $ADMIN_RESPONSE | jq -r '.token')
IS_ADMIN=$(echo $ADMIN_RESPONSE | jq -r '.is_admin')

if [ "$IS_ADMIN" = "true" ]; then
  echo "✓ First user is admin"
else
  echo "✗ First user is NOT admin - FAIL"
  exit 1
fi

# 2. Try to create user without token (should fail)
echo "2. Trying to create user without admin token..."
RESPONSE=$(curl -s -X POST http://localhost/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"hacker","password":"Pass123!"}')

if echo $RESPONSE | grep -q "unauthorized"; then
  echo "✓ Registration blocked without token"
else
  echo "✗ Registration allowed without token - FAIL"
  exit 1
fi

# 3. Create user with admin token (should succeed)
echo "3. Creating user with admin token..."
RESPONSE=$(curl -s -X POST http://localhost/api/auth/register \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"username":"testuser","password":"User123!"}')

USER_TOKEN=$(echo $RESPONSE | jq -r '.token')

if [ "$USER_TOKEN" != "null" ]; then
  echo "✓ User created successfully"
```

```
else
  echo "✗ User creation failed - FAIL"
  exit 1
fi

# 4. Try to create another user with regular user token (should fail)
echo "4. Trying to create user with regular user token..."
RESPONSE=$(curl -s -X POST http://localhost/api/auth/register \
  -H "Authorization: Bearer $USER_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"username":"hacker2","password":"Pass123!"}')

if echo $RESPONSE | grep -q "forbidden"; then
  echo "✓ Regular user cannot create users"
else
  echo "✗ Regular user can create users - FAIL"
  exit 1
fi

# 5. Try to add API key with regular user (should fail)
echo "5. Trying to add API key with regular user..."
RESPONSE=$(curl -s -X POST http://localhost/api/providers/keys \
  -H "Authorization: Bearer $USER_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"provider_id":1,"key_value":"fake-key"}')

if echo $RESPONSE | grep -q "forbidden"; then
  echo "✓ Regular user cannot add API keys"
else
  echo "✗ Regular user can add API keys - FAIL"
  exit 1
fi

# 6. Add API key with admin (should succeed)
echo "6. Adding API key with admin token..."
RESPONSE=$(curl -s -X POST http://localhost/api/providers/keys \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"provider_id":1,"key_value":"test-key-12345"}')

if echo $RESPONSE | grep -q "API key added successfully"; then
  echo "✓ Admin can add API keys"
else
  echo "✗ Admin cannot add API keys - FAIL"
  exit 1
fi
```

```
echo ""
echo "✅ All admin system tests passed!"
```

📊 Monitoring Admin Actions

View User Activity

sql

```sql
-- Recent user registrations
SELECT
  id,
  username,
  is_admin,
  created_at
FROM users
ORDER BY created_at DESC
LIMIT 10;

-- API key usage by provider
SELECT
  p.display_name,
  COUNT(k.id) as total_keys,
  COUNT(k.id) FILTER (WHERE k.status = 'active') as active_keys,
  SUM(k.usage_count) as total_usage
FROM ai_providers p
LEFT JOIN api_keys k ON k.provider_id = p.id
GROUP BY p.id, p.display_name;
```

🔄 Migration from Existing System

If you already have users without the `is_admin` field:

```sql
-- Add is_admin column
ALTER TABLE users ADD COLUMN IF NOT EXISTS is_admin BOOLEAN DEFAULT FALSE;

-- Make first user admin
UPDATE users
SET is_admin = true
WHERE id = (SELECT MIN(id) FROM users);

-- Or make specific user admin
UPDATE users
SET is_admin = true
WHERE username = 'admin';
```

🎓 Summary

✅ **First user** = automatic admin
✅ **Admin required** for user creation
✅ **Admin required** for API key management
✅ **Token-based** authentication
✅ **Secure by default**

This ensures your platform is secure and only authorized admins can manage users and API keys!