Authentication System Documentation

Overview

The Music U Lesson Scheduler implements a comprehensive JWT-based authentication system with role-based authorization for teachers and students. The system provides secure user registration, login, and access control for API endpoints.

Features

- **User Registration**: New users can register with email, username, and password
- **User Authentication**: Login with username/email and password to receive JWT tokens
- **Password Security**: Bcrypt hashing for secure password storage
- **JWT Tokens**: Stateless authentication with 30-minute token expiration
- **Role-Based Authorization**: Separate permissions for teachers and students
- **Password Management**: Change password functionality for authenticated users

Architecture

Authentication Module Structure

```
app/auth/
├── __init__.py          # Module exports
├── utils.py             # JWT and password utilities
├── dependencies.py      # FastAPI auth dependencies
└── routers.py           # Authentication endpoints
```

Key Components

1. **JWT Utilities** (`auth/utils.py`):
   - Token creation and verification
   - Password hashing and verification
   - Configurable token expiration

2. **Auth Dependencies** (`auth/dependencies.py`):
   - `get_current_user` : Extract user from JWT token
   - `get_current_active_user` : Ensure user is active
   - `require_teacher_role` : Require teacher permissions
   - `require_student_role` : Require student permissions

3. **Auth Routes** (`auth/routers.py`):
   - User registration
   - User login
   - Current user info
   - Password change

API Endpoints

Authentication Endpoints

Register User

```
POST /auth/register
```

Request Body:

```
{
  "email": "user@example.com",
  "username": "username",
  "full_name": "Full Name",
  "password": "password123",
  "is_teacher": false,
  "phone": "555-0123"
}
```

Response:

```
{
  "id": 1,
  "email": "user@example.com",
  "username": "username",
  "full_name": "Full Name",
  "is_teacher": false,
  "phone": "555-0123",
  "is_active": true,
  "created_at": "2024-08-14T10:00:00Z",
  "updated_at": null
}
```

Login User

```
POST /auth/login
```

Request Body (Form Data):

```
username: username_or_email
password: password123
```

Response:

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

Get Current User

```
GET /auth/me
Authorization: Bearer <token>
```

Response:

```
{
  "id": 1,
  "email": "user@example.com",
  "username": "username",
  "full_name": "Full Name",
  "is_teacher": false,
  "is_active": true,
  "created_at": "2024-08-14T10:00:00Z"
}
```

Change Password

```
POST /auth/change-password
Authorization: Bearer <token>
```

Request Body:

```
{
  "old_password": "currentpassword",
  "new_password": "newpassword123"
}
```

Role-Based Authorization

Teacher Permissions

- Create and manage lessons
- View all users and lessons
- Update any user profile
- Delete users (except themselves)

Student Permissions

- View their own profile and lessons
- Update their own profile (limited fields)
- Add notes to their lessons
- View lessons from their teachers

Protected Routes

Routes are protected using FastAPI dependencies:

```python
# Require any authenticated user
@router.get("/endpoint")
async def endpoint(user: User = Depends(get_current_active_user)):
    pass

# Require teacher role
@router.get("/teacher-only")
async def teacher_endpoint(user: User = Depends(require_teacher_role)):
    pass

# Require student role
@router.get("/student-only")
async def student_endpoint(user: User = Depends(require_student_role)):
    pass
```

JWT Token Structure

JWT tokens contain the following claims:

```
{
  "sub": "username",           # Subject (username)
  "user_id": 123,              # User database ID
  "role": "teacher",           # Role string ("teacher" or "student")
  "is_teacher": true,          # Boolean teacher flag
  "exp": 1692014400,           # Expiration timestamp
  "iat": 1692012600,           # Issued at timestamp
  "type": "access_token"       # Token type
}
```

Security Features

Password Security

- **Bcrypt Hashing**: All passwords are hashed using bcrypt with automatic salt generation
- **Password Validation**: Minimum 8 character password requirement
- **No Plain Text Storage**: Passwords are never stored in plain text

Token Security

- **Short Expiration**: 30-minute token expiration by default
- **Secure Signing**: Tokens signed with configurable secret key
- **Tamper Protection**: JWT signature prevents token modification

Access Control

- **Authentication Required**: Most endpoints require valid JWT token
- **Role Validation**: Role-based access control enforced at endpoint level
- **User Context**: Current user context available in all protected routes

Environment Configuration

Required environment variables:

```
SECRET_KEY=your-secret-key-here
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30
```

Usage Examples

1. Register a New Teacher

```
curl -X POST "http://localhost:8000/auth/register" \
  -H "Content-Type: application/json" \
  -d '{
    "email": "teacher@school.com",
    "username": "musicteacher",
    "full_name": "Music Teacher",
    "password": "securepass123",
    "is_teacher": true,
    "phone": "555-0100"
  }'
```

2. Login and Get Token

```
curl -X POST "http://localhost:8000/auth/login" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "username=musicteacher&password=securepass123"
```

3. Access Protected Route

```
curl -X GET "http://localhost:8000/auth/me" \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
```

Error Handling

The authentication system provides clear error responses:

- **400 Bad Request**: Invalid input data, duplicate email/username
- **401 Unauthorized**: Invalid credentials, expired/invalid token
- **403 Forbidden**: Insufficient permissions for requested action
- **422 Unprocessable Entity**: Validation errors (e.g., weak password)

Testing

The authentication system includes comprehensive tests covering:

- User registration (success and failure cases)
- User login (with username and email)
- Token validation and expiration
- Role-based authorization
- Password change functionality
- Protected route access

Run tests with:

```
pytest tests/test_auth.py -v
```

Best Practices

1. **Use HTTPS**: Always use HTTPS in production for token transmission
2. **Secure Secret Key**: Use a strong, randomly generated secret key
3. **Token Expiration**: Keep token expiration times short for security
4. **Password Requirements**: Enforce strong password policies
5. **Role Verification**: Always verify user roles at the endpoint level
6. **Error Handling**: Provide clear but secure error messages

Integration with Existing Code

The authentication system is designed to integrate seamlessly with existing user and lesson management:

- User models already include `is_teacher` and `is_active` fields
- Password hashing functions are available in `crud.py`
- JWT utilities are reusable across the application
- Dependencies can be easily added to any route for protection

This authentication system provides a solid foundation for secure access control in the Music U Lesson Scheduler application.