Authentication System Documentation

Overview

The SSVproff API implements a complete JWT-based authentication system with the following features:

- User registration and login
- JWT access and refresh tokens
- Password hashing with bcrypt
- Protected endpoints
- Example CRUD operations (Tasks)

Architecture

Components

1. **Models** (`app/models/`)
   - `User` : User account information
   - `Task` : Example resource for authenticated users

2. **Schemas** (`app/schemas/`)
   - `UserCreate` , `UserLogin` , `UserResponse` : User operations
   - `Token` , `TokenPayload` : JWT token handling
   - `TaskCreate` , `TaskUpdate` , `TaskResponse` : Task operations

3. **Services** (`app/services/`)
   - `auth.py` : Authentication business logic
   - `task.py` : Task management business logic

4. **Security** (`app/core/security.py`)
   - Password hashing (bcrypt)
   - JWT token creation and validation
   - Token types: access (30 min) and refresh (7 days)

5. **Dependencies** (`app/api/deps.py`)
   - `get_current_user` : Dependency for protected endpoints
   - `get_current_active_superuser` : Dependency for admin endpoints

API Endpoints

Authentication

Register User

```
POST /api/v1/auth/register
Content-Type: application/json

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

Response:


```
{
  "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"
}
```


Login

```
POST /api/v1/auth/login
Content-Type: application/json

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

Response:


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


Get Current User

```
GET /api/v1/auth/me
Authorization: Bearer <access_token>
```

Response:

```
{
  "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"
}
```

Refresh Token

```
POST /api/v1/auth/refresh
Content-Type: application/json

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

Response:

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

Tasks (Protected Endpoints)

All task endpoints require authentication via the `Authorization: Bearer <access_token>` header.

List Tasks

```
GET /api/v1/tasks/?skip=0&limit=10&completed=false
Authorization: Bearer <access_token>
```

Create Task

```
POST /api/v1/tasks/
Authorization: Bearer <access_token>
Content-Type: application/json

{
  "title": "Complete project",
  "description": "Finish the API implementation",
  "is_completed": false
}
```

Get Task

```
GET /api/v1/tasks/{task_id}
Authorization: Bearer <access_token>
```

Update Task

```
PUT /api/v1/tasks/{task_id}
Authorization: Bearer <access_token>
Content-Type: application/json

{
  "is_completed": true
}
```

Delete Task

```
DELETE /api/v1/tasks/{task_id}
Authorization: Bearer <access_token>
```

Database Schema

Users Table

```sql
CREATE TABLE users (
    id UUID PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    hashed_password VARCHAR(255) NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    is_superuser BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Indexes
CREATE INDEX ix_users_email ON users(email);
CREATE INDEX ix_users_username ON users(username);
CREATE INDEX ix_users_email_active ON users(email, is_active);
CREATE INDEX ix_users_username_active ON users(username, is_active);
```

Tasks Table

```sql
CREATE TABLE tasks (
    id UUID PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    is_completed BOOLEAN DEFAULT FALSE,
    owner_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Indexes
CREATE INDEX ix_tasks_owner_id ON tasks(owner_id);
CREATE INDEX ix_tasks_owner_completed ON tasks(owner_id, is_completed);
CREATE INDEX ix_tasks_owner_created ON tasks(owner_id, created_at);
```

Setup Instructions

1. Install Dependencies

```
cd api
pip install -r requirements.txt
```

2. Configure Environment

```
cp .env.example .env
# Edit .env with your configuration
```

Important environment variables:
- `SECRET_KEY`: Generate with `openssl rand -hex 32`
- `DATABASE_URL`: PostgreSQL or SQLite connection string
- `ACCESS_TOKEN_EXPIRE_MINUTES`: Token expiration (default: 30)

3. Initialize Database

Option A: Using Alembic Migrations (Recommended)

```
# Run migrations
alembic upgrade head

# Or use the helper script
./scripts/run_migrations.sh
```

Option B: Using Initialization Script (Development)

```
# Creates tables and test users
python scripts/init_db.py
```

This creates:
- Test user: `test@example.com` / `testpassword123`
- Admin user: `admin@example.com` / `admin123` (superuser)

4. Run the API

```
# Development
uvicorn app.main:app --reload --port 8000

# Production
uvicorn app.main:app --host 0.0.0.0 --port 8000
```

5. Access API Documentation

- Swagger UI: http://localhost:8000/docs
- ReDoc: http://localhost:8000/redoc

Testing

Run Tests

```
# All tests
pytest

# With coverage
pytest --cov=app tests/

# Specific test file
pytest tests/test_auth_endpoints.py

# Verbose output
pytest -v
```

Test Coverage

Current test coverage includes:
- User registration (success, duplicate email/username, validation)
- User login (success, wrong password, nonexistent user)
- Current user endpoint (success, no token, invalid token)
- Token refresh (success, invalid token, wrong token type)
- Task CRUD operations (create, read, update, delete)
- User isolation (users can only access their own tasks)

Security Best Practices

1. **Environment Variables**
   - Never commit `.env` files
   - Use strong, random `SECRET_KEY` in production
   - Generate with: `openssl rand -hex 32`

2. **Passwords**
   - Minimum 8 characters
   - Hashed with bcrypt
   - Never stored in plain text

3. **Tokens**
   - Access tokens: short-lived (30 minutes)
   - Refresh tokens: longer-lived (7 days)
   - Store securely on client side (httpOnly cookies recommended)

4. **CORS**
   - Configure `BACKEND_CORS_ORIGINS` in .env
   - Only allow trusted origins

5. **HTTPS**
   - Always use HTTPS in production
   - Tokens are vulnerable over HTTP

Error Handling

The API returns appropriate HTTP status codes:

- `200 OK` : Successful request
- `201 Created` : Resource created successfully
- `204 No Content` : Successful deletion
- `400 Bad Request` : Invalid input (e.g., duplicate email)
- `401 Unauthorized` : Missing or invalid authentication
- `403 Forbidden` : Insufficient permissions
- `404 Not Found` : Resource not found
- `422 Unprocessable Entity` : Validation error

Error response format:

```
{
  "detail": "Error message here"
}
```

Extending the Authentication System

Adding a New Protected Resource

1. Create the model in `app/models/` :

```
from app.models.user import GUID

class MyResource(Base):
    __tablename__ = "my_resources"
    id = Column(GUID(), primary_key=True, default=uuid.uuid4)
    owner_id = Column(GUID(), ForeignKey("users.id"))
    # ... other fields
```

1. Create schemas in `app/schemas/` :

```
class MyResourceCreate(BaseModel):
    # fields

class MyResourceResponse(BaseModel):
    # fields
    model_config = ConfigDict(from_attributes=True)
```

1. Create service in `app/services/` :

```
def get_user_resources(db: Session, user: User):
    return db.query(MyResource).filter(
        MyResource.owner_id == user.id
    ).all()
```

1. Create endpoints in `app/api/v1/endpoints/` :

```
@router.get("/")
def list_resources(
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    return get_user_resources(db, current_user)
```

1. Register router in `app/api/v1/api.py`:

```
api_router.include_router(
    my_resources.router,
    prefix="/my-resources",
    tags=["my-resources"]
)
```

1. Create migration:

```
alembic revision --autogenerate -m "add_my_resource_table"
alembic upgrade head
```

Troubleshooting

Database Connection Issues

```
sqlalchemy.exc.OperationalError: connection refused
```

Solution: Check if PostgreSQL is running or use SQLite for development:

```
# In .env
DATABASE_URL=sqlite:///./ssvproff_dev.db
```

Token Validation Errors

```
Could not validate credentials
```

Solution: Ensure `SECRET_KEY` matches between token creation and validation.

Import Errors

```
ModuleNotFoundError: No module named 'app'
```

Solution: Run commands from the `api/` directory or ensure PYTHONPATH is set.

Additional Resources

- FastAPI Documentation (https://fastapi.tiangolo.com/)
- SQLAlchemy Documentation (https://docs.sqlalchemy.org/)
- Pydantic Documentation (https://docs.pydantic.dev/)

- JWT.io (https://jwt.io/) - Debug JWT tokens
- Alembic Documentation (https://alembic.sqlalchemy.org/)