

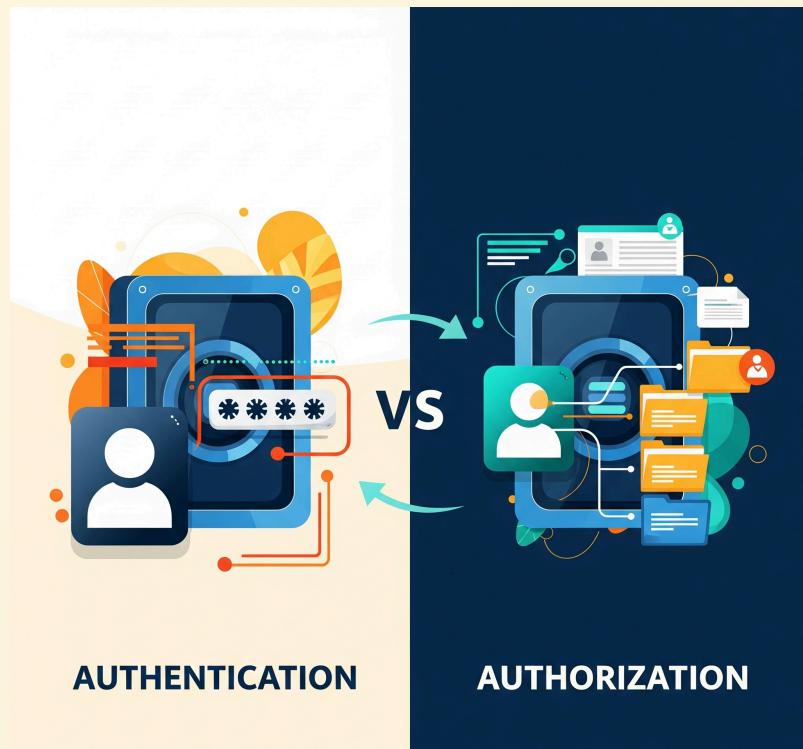
Week 6: Security

Overview

- Authentication and authorization
- Register and login flow using a local database
- Password hashing - `passlib` library
- JWT (JSON Web Tokens) - `jose` library
- Securing routes with JWT
- Testing with `curl`, `Swagger UI` and `Postman`

Authentication vs Authorization

| Authentication | Authorization |
|------------------------|-----------------------------|
| Verifies user identity | Determines user permissions |



Database vs Pydantic models

| Feature | Database Model (User) | Request Model (UserCreate) |
|--------------------|--------------------------|-------------------------------|
| Backed by DB table | ✓ Yes | ✗ No |
| ORM usage | ✓ (SQLAlchemy) | ✗ (Pydantic) |
| Input validation | ✗ | ✓ |
| JSON schema | ✗ | ✓ |
| Base class | Base | BaseModel |

Register user flow

1. User submits registration form
2. Validate input with Pydantic model
3. Hash password using `passlib`
4. Create User object
5. Save to database
6. Return success response

Database model for user

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import declarative_base
from database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    username = Column(String, unique=True, index=True)
    fullname = Column(String)
    email = Column(String, unique=True, index=True)
    hashed_password = Column(String)
```

Pydantic models for user registration

```
from pydantic import BaseModel

class UserRequest(BaseModel):
    username: str
    fullname: str
    email: str
    password: str

class UserResponse(BaseModel):
    username: str
    email: str
```

Code Example: Register User

```
@router.post("/register", response_model=UserResponse)
def create_user(user_req: UserRequest, db: Session = Depends(get_db)):
    existing_user = db.query(User).filter(User.username == user_req.username).first()
    if existing_user:
        raise HTTPException(status_code=400, detail="Username already exists")
    new_user = User(
        username=user_req.username,
        fullname=user_req.fullname,
        email=user_req.email,
        hashed_password=hash_password(user_req.password)
    )

    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    response = UserResponse(username=new_user.username, email=new_user.email)
    return response
```

Testing User Registration with `curl`

```
curl -X POST "http://localhost:8000/users/register" \  
-H "Content-Type: application/json" \  
-d '{"username": "testuser3", "fullname": "Test User",  
"email": "foo@dom.com", "password": "password"}'
```


Testing User Registration with Swagger UI

- Open browser and go to `http://localhost:8000/docs`
- Click on the `/users/register` endpoint
- Click "Try it out"
- Fill in the form with user details
- Click "Execute"
- Check the response for success or error messages

Login user flow

1. User submits login form
2. Validate input with Pydantic model
3. Check credentials against database
4. If valid, generate JWT token
5. Return token in response
6. Use token for subsequent requests

Pydantic models for user login

```
from pydantic import BaseModel

class UserLoginRequest(BaseModel):
    username: str
    password: str

class UserLoginResponse(BaseModel):
    message: str
    username: str
    access_token: str
    access_token_type: str = "bearer"
```

JWT (JSON Web Tokens)

- JSON Web Tokens (JWT) are encoded access tokens
- Contain claims (user ID, role, expiration, ...)
- Signed using a secret key or RSA

Create JWT access token

```
from jose import jwt

SECRET_KEY = "your_secret_key"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

def create_access_token(data: dict, expires_delta: timedelta = None):
    to_encode = data.copy()
    expire = datetime.utcnow() + (expires_delta or timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES))
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt
```

Validate user credentials

```
@router.post("/login", response_model=UserLoginResponse)
def login(user_req: UserLoginRequest, db: Session = Depends(get_db)):
    user = db.query(User).filter(User.username == user_req.username).first()

    if not user or not verify_password(user_req.password, user.hashed_password):
        raise HTTPException(status_code=401, detail="Invalid username or password")

    access_token = create_access_token(data={"sub": user.username})
    return UserLoginResponse(
        message="Login successful",
        username=user.username,
        access_token=access_token
    )
```

Testing User Login with `curl`

```
curl -X POST "http://localhost:8000/users/login" \  
-H "Content-Type: application/json" \  
-d '{"username": "testuser3", "password": "password"}'
```

Response

```
{  
  "message": "Login successful",  
  "username": "testuser3",  
  "access_token": "YOUR ACCESS TOKEN",  
  "access_token_type": "bearer"  
}
```

Securing Routes with JWT

- Only logged in users can access the `GET /users` endpoint
- Use `Depends` to inject the token dependency
- `get_current_user` decodes token and returns user

```
@router.get("/", dependencies=[Depends(get_current_user)])
def get_users(db: Session = Depends(get_db), current_user: User = Depends(get_current_user)):
    users = db.query(User).all()
    return users
```


Get current user from JWT

```
def get_current_user(token: str = Security(oauth2_scheme), db: Session = Depends(get_db)):
    credentials_exception = HTTPException(
        status_code=401,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = decode_access_token(token)
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
    except Exception:
        raise credentials_exception
    user = db.query(User).filter(User.username == username).first()
    if user is None:
        raise credentials_exception
    return user
```

Decoding JWT

```
SECRET_KEY = "your_secret_key"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

def decode_access_token(token: str):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        return payload
    except jwt.JWTError:
        return None
```

Testing Secured Route with Postman

1. Open Postman
2. Create a new request
3. Set method to **GET** and URL to **http://localhost:8000/users**
4. Go to the "Authorization" tab
5. Select "Bearer Token" from the dropdown
6. Paste the JWT token from the login response
7. Click "Send"
8. You should see the list of users if the token is valid

Create a React app to test the API

- The React app will have a simple UI for
 - user registration
 - user login
 - displaying all users data
 - deleting a user by ID

Remember

- Database models vs Pydantic models
- Password `hashing` and `JWT` concepts
- Creating secure `register` / `login` flows
- `Securing routes` with dependencies