Coverage for **app/routers/auth.py**: 90%

69 statements 62 run 7 missing 0 excluded

« prev ^ index » next coverage.py v7.11.3, created at 2025-11-21 23:38 -0800

```python
from fastapi import APIRouter, Depends, HTTPException, status
from fastapi.security import OAuth2PasswordRequestForm
from pydantic import BaseModel, EmailStr
from typing import List

from app.services.user_service import CSVUserService
from app.deps import get_user_service, pwd_context, create_access_token, get_current_user
from app.schemas.user import UserCreate, UserOut, Token, UserUpdate


router = APIRouter(prefix="/auth", tags=["auth"])


class UserCreateRequest(BaseModel):
    username: str
    email: EmailStr
    password: str


class UserOut(BaseModel):
    id: int
    username: str
    email: EmailStr
    is_admin: bool


@router.post(
    "/register",
    response_model=UserOut,
    status_code=status.HTTP_201_CREATED,
)
```

```python
def register(payload: UserCreateRequest, svc: CSVUserService = Depends(get_user_service)):
    try:
        user = svc.create_user(
            username=payload.username,
            email=payload.email,
            password_hash=pwd_context.hash(payload.password),
            is_admin=False,
        )
        return UserOut(
            id=user["id"],
            username=user["username"],
            email=user["email"],
            is_admin=user["is_admin"],
        )
    except ValueError as e:
        msg = str(e)
        if msg in {"username_taken", "email_taken"}:
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail=msg,
            )
        raise


@router.post("/token")
def login(form: OAuth2PasswordRequestForm = Depends(), svc: CSVUserService = Depends(get_user_service)):
    user = svc.get_by_username(form.username)
    if not user or not pwd_context.verify(form.password, user["password_hash"]):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="invalid_credentials",
        )

    if user.get("is_suspended", False):
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail=f"Your account is suspended until {user.get('suspended_until', 'N/A')}.",
        )

    token = create_access_token(
        username=user["username"],
        user_id=user["id"],
        is_admin=user["is_admin"],
        minutes=60,
    )
    return Token(access_token=token, token_type="bearer")



@router.get("/me", response_model=UserOut)
def me(curr=Depends(get_current_user)):
    return UserOut(
        id=curr["id"],
        username=curr["username"],
        email=curr["email"],
        is_admin=curr["is_admin"],
    )


@router.get("/users", response_model=List[UserOut])
def list_users(
    curr=Depends(get_current_user),
    svc: CSVUserService = Depends(get_user_service),
):
    if not curr.get("is_admin"):
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Admin privileges required",
        )

    rows = svc.repo.read_all(svc.path)
    users_out = []

    for row in rows:
        user_id = int(row["id"])
        raw_flag = str(row.get("is_admin", "")).strip().lower()
        is_admin = raw_flag in {"true", "1", "yes"}

        users_out.append(
            UserOut(
                id=user_id,
                username=row["username"],
                email=row["email"],
                is_admin=is_admin,
            )
        )

    return users_out

@router.post("/suspend/{user_id}")
def suspend_user_route(
    user_id: int,
    duration_minutes: int,
    curr=Depends(get_current_user),
    svc: CSVUserService = Depends(get_user_service),
):
    if not curr.get("is_admin"):
        raise HTTPException(403, "Admin privileges required")

    try:
        suspended_user = svc.suspend_user(
            admin_id=curr["id"],
            target_id=user_id,
            duration_minutes=duration_minutes,
        )
        return {
            "message": f"User {user_id} suspended for {duration_minutes} minutes.",
            "suspended_until": suspended_user["suspended_until"],
        }
    except ValueError:
        raise HTTPException(404, "User not found")
```

```python

@router.post("/unsuspend/{user_id}")
def unsuspend_user_route(
    user_id: int,
    curr=Depends(get_current_user),
    svc: CSVUserService = Depends(get_user_service),
):
    """Allows admin to unsuspend a user."""
    if not curr.get("is_admin"):
        raise HTTPException(403, "Admin privileges required")

    try:
        svc.unsuspend_user(curr["id"], user_id)
        return {"message": f"User {user_id} is no longer suspended."}

    except ValueError:
        raise HTTPException(404, "User not found")
```