FastAPI JWT Auth Starter (Clean Architecture)

A neat, clean, and standard scaffold you can drop into any project. It gives you:

- **JWT auth with access & refresh tokens** (rotation-ready)
- **Scoped (role-based) authorization** using OAuth2 scopes
- **Organized Swagger UI** with tag groups & descriptions
- **Modular routers** so all APIs appear in a **single Swagger**
- **12-factor config** via environment variables
- **Hashing** with `passlib[bcrypt]`

  Works even if you don't wire a real DB yet (in-memory demo repo included). Swap with SQLAlchemy when ready.

---

1) Project Structure

```
fastapi-jwt-starter/
├─ pyproject.toml
├─ .env.example
├─ app/
│  ├─ main.py
│  ├─ deps/__init__.py
│  ├─ core/
│  │  ├─ config.py
│  │  ├─ security.py
│  │  └─ jwt.py
│  ├─ models/
│  │  └─ user.py
│  ├─ schemas/
│  │  ├─ auth.py
│  │  └─ user.py
│  ├─ repositories/
│  │  └─ users.py
│  ├─ routers/
│  │  ├─ auth.py
│  │  ├─ users.py
│  │  └─ health.py
│  └─ utils/
│     └─ clock.py
└─ tests/
   └─ test_auth.py
```

---

2) Dependencies (pyproject.toml)

```
[project]
name = "fastapi-jwt-starter"
version = "0.1.0"
dependencies = [
  "fastapi>=0.115",
  "uvicorn[standard]>=0.30",
  "pydantic>=2.7",
  "pydantic-settings>=2.3",
  "python-jose[cryptography]>=3.3.0",
  "passlib[bcrypt]>=1.7.4",
]

[tool.uvicorn]
factory = true
```

---

3) Environment (.env.example)

```
APP_NAME=My API Suite
APP_ENV=dev
SECRET_KEY=change-this-in-prod
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30
REFRESH_TOKEN_EXPIRE_DAYS=7
CORS_ORIGINS=*
```

---

4) Core — Config & Clock

app/core/config.py

```
from pydantic_settings import BaseSettings
from pydantic import Field
from typing import List

class Settings(BaseSettings):
    app_name: str = Field(default="API Suite")
    app_env: str = Field(default="dev")

    secret_key: str = Field(default="change-me")
```

```
    algorithm: str = Field(default="HS256")
    access_token_expire_minutes: int = Field(default=30)
    refresh_token_expire_days: int = Field(default=7)

    cors_origins: str = Field(default="*")  # comma separated or "*"

    class Config:
        env_file = ".env"
        env_file_encoding = "utf-8"

settings = Settings()
```

app/utils/clock.py

```
from datetime import datetime, timezone

def now_utc() -> datetime:
    return datetime.now(timezone.utc)
```

---

5) Core — Security & JWT helpers

app/core/security.py

```
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")


def verify_password(plain: str, hashed: str) -> bool:
    return pwd_context.verify(plain, hashed)


def hash_password(plain: str) -> str:
    return pwd_context.hash(plain)
```

app/core/jwt.py

```
from datetime import timedelta
from typing import Any, Dict, Optional
from jose import jwt
from app.core.config import settings
```

```python
from app.utils.clock import now_utc


def _encode(payload: Dict[str, Any], expires_delta: timedelta) -> str:
    to_encode = payload.copy()
    expire = now_utc() + expires_delta
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, settings.secret_key,
algorithm=settings.algorithm)


def create_access_token(subject: str, scopes: list[str] | None = None) -> str:
    scopes = scopes or []
    return _encode(
        {"sub": subject, "type": "access", "scopes": scopes},
        expires_delta=timedelta(minutes=settings.access_token_expire_minutes),
    )


def create_refresh_token(subject: str) -> str:
    return _encode(
        {"sub": subject, "type": "refresh"},
        expires_delta=timedelta(days=settings.refresh_token_expire_days),
    )


def decode_token(token: str) -> Dict[str, Any]:
    return jwt.decode(token, settings.secret_key,
algorithms=[settings.algorithm])
```

---

6) Schemas

app/schemas/user.py

```python
from pydantic import BaseModel, EmailStr
from typing import Optional, List

class UserBase(BaseModel):
    id: str
    email: EmailStr
    full_name: Optional[str] = None
    is_active: bool = True
    scopes: List[str] = []
```

```python
class UserCreate(BaseModel):
    email: EmailStr
    full_name: Optional[str] = None
    password: str

class UserPublic(UserBase):
    pass
```

app/schemas/auth.py

```python
from pydantic import BaseModel
from typing import List

class Token(BaseModel):
    access_token: str
    refresh_token: str
    token_type: str = "bearer"

class TokenPayload(BaseModel):
    sub: str
    type: str
    scopes: List[str] = []
```

---

7) Models & Repository (demo in-memory)

app/models/user.py

```python
from dataclasses import dataclass
from typing import List

@dataclass
class User:
    id: str
    email: str
    full_name: str | None
    hashed_password: str
    is_active: bool
    scopes: List[str]
```

app/repositories/users.py

```python
from typing import Dict, Optional
from app.models.user import User
from app.core.security import hash_password

# Demo in-memory store (replace with DB/SQLAlchemy)
_USERS: Dict[str, User] = {}


def seed_admin() -> None:
    if "1" not in _USERS:
        _USERS["1"] = User(
            id="1",
            email="admin@example.com",
            full_name="Admin",
            hashed_password=hash_password("admin"),
            is_active=True,
            scopes=["admin", "users:read", "users:write"],
        )


def get_by_email(email: str) -> Optional[User]:
    return next((u for u in _USERS.values() if u.email.lower() ==
email.lower()), None)


def get_by_id(user_id: str) -> Optional[User]:
    return _USERS.get(user_id)


def create_user(user_id: str, email: str, full_name: str | None, password: str)
-> User:
    user = User(
        id=user_id,
        email=email,
        full_name=full_name,
        hashed_password=hash_password(password),
        is_active=True,
        scopes=["users:read"],
    )
    _USERS[user.id] = user
    return user
```

---

8) Auth Dependencies

app/deps/init.py

```
from fastapi import Depends, HTTPException, status, Security
from fastapi.security import OAuth2PasswordBearer, SecurityScopes
from app.core.jwt import decode_token
from app.repositories.users import get_by_id

oauth2_scheme = OAuth2PasswordBearer(
    tokenUrl="/auth/login",
    scopes={
        "admin": "Administrative actions",
        "users:read": "Read users",
        "users:write": "Create/update users",
    },
)


def get_current_user(token: str = Depends(oauth2_scheme)):
    try:
        payload = decode_token(token)
        if payload.get("type") != "access":
            raise ValueError("Not an access token")
        user_id: str = payload.get("sub")
    except Exception:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Invalid token")

    user = get_by_id(user_id)
    if not user or not user.is_active:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Inactive user")
    return user


def require_scopes(sec_scopes: SecurityScopes, user =
Security(get_current_user)):
    # If no scopes required, allow
    if not sec_scopes.scopes:
        return user
    missing = [s for s in sec_scopes.scopes if s not in user.scopes]
    if missing:
        raise HTTPException(status_code=403, detail=f"Missing scopes: {',
'.join(missing)}")
    return user
```

9) Routers — Auth, Users, Health

app/routers/auth.py

```
from fastapi import APIRouter, Depends, HTTPException
from fastapi.security import OAuth2PasswordRequestForm
from app.core.security import verify_password
from app.core.jwt import create_access_token, create_refresh_token, decode_token
from app.schemas.auth import Token
from app.repositories import users as repo

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

@router.post("/login", response_model=Token, summary="Login with username/
password")
def login(form: OAuth2PasswordRequestForm = Depends()):
    # OAuth2 form uses username field; we treat it as email
    user = repo.get_by_email(form.username)
    if not user or not verify_password(form.password, user.hashed_password):
        raise HTTPException(status_code=400, detail="Incorrect email or
password")

    access = create_access_token(subject=user.id, scopes=user.scopes)
    refresh = create_refresh_token(subject=user.id)
    return Token(access_token=access, refresh_token=refresh)

@router.post("/refresh", response_model=Token, summary="Refresh access token")
def refresh_token(refresh_token: str):
    payload = decode_token(refresh_token)
    if payload.get("type") != "refresh":
        raise HTTPException(status_code=400, detail="Invalid refresh token")
    user_id = payload.get("sub")
    user = repo.get_by_id(user_id)
    if not user:
        raise HTTPException(status_code=400, detail="User not found")
    access = create_access_token(subject=user.id, scopes=user.scopes)
    # rotation-friendly: issue a new refresh token
    new_refresh = create_refresh_token(subject=user.id)
    return Token(access_token=access, refresh_token=new_refresh)
```

app/routers/users.py

```
from fastapi import APIRouter, Security
from app.schemas.user import UserPublic, UserCreate
```

```
from app.repositories import users as repo
from app.deps import require_scopes

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

@router.get("/me", response_model=UserPublic, summary="Current user profile")
def read_me(user = Security(require_scopes, scopes=[])):
    return {
        "id": user.id,
        "email": user.email,
        "full_name": user.full_name,
        "is_active": user.is_active,
        "scopes": user.scopes,
    }

@router.post("/", response_model=UserPublic, summary="Create a user (admin)")
def create_user(payload: UserCreate, user = Security(require_scopes,
scopes=["users:write"])):
    created = repo.create_user(user_id=str(len(repo._USERS) + 1),
email=payload.email, full_name=payload.full_name, password=payload.password)
    return {
        "id": created.id,
        "email": created.email,
        "full_name": created.full_name,
        "is_active": created.is_active,
        "scopes": created.scopes,
    }
```

app/routers/health.py

```
from fastapi import APIRouter

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

@router.get("/live", summary="Liveness probe")
def live():
    return {"status": "ok"}

@router.get("/ready", summary="Readiness probe")
def ready():
    return {"status": "ready"}
```

---

10) Single Swagger UI with Tag Groups

app/main.py

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.core.config import settings
from app.routers import auth, users, health
from app.repositories.users import seed_admin

# Tag metadata (controls Swagger grouping & descriptions)
openapi_tags = [
    {"name": "Auth", "description": "Login, refresh tokens, and auth flows."},
    {"name": "Users", "description": "User-facing endpoints and admin
operations."},
    {"name": "Health", "description": "Service health checks."},
]

app = FastAPI(
    title=settings.app_name,
    version="1.0.0",
    description="All APIs in one place with JWT-based auth & scopes.",
    openapi_tags=openapi_tags,
)

# CORS
origins = [o.strip() for o in settings.cors_origins.split(",")] if
settings.cors_origins else ["*"]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include routers -> single Swagger UI
app.include_router(auth.router)
app.include_router(users.router)
app.include_router(health.router)

# Seed demo admin
@app.on_event("startup")
def startup_seed():
    seed_admin()
```

Run with:

```
uvicorn app.main:app --reload
```

Your Swagger UI is at `http://127.0.0.1:8000/docs` with all APIs neatly grouped.

---

11) How to Use (Quickstart)

1) Start the server and open **/docs**. 2) Use **Authorize** (top-right) with `OAuth2` scheme: - Click **Authorize**, add `bearer` token after logging in. 3) First, call **POST /auth/login** with: - username: `admin@example.com`, password: `admin` (seeded) 4) Copy the `access_token` from response; click **Authorize**, paste `Bearer <token>`. 5) Try **GET /users/me**. 6) Create a user with **POST /users/** (requires `users:write`).

---

12) Swap In a Real Database (optional)

- Replace the in-memory repo with SQLAlchemy models & a `UserRepository` interface.
- Keep the same router signatures; nothing else changes.

---

13) Production Notes

- **Rotate the** `SECRET_KEY` and store it securely.
- Prefer **short access tokens** (15–30m) + **refresh rotation**.
- Consider storing refresh token hashes (allow revoke/blacklist on logout).
- Enforce HTTPS, secure cookies (if using cookies), and strict CORS.
- Add rate limiting and audit logging on auth endpoints.
- Add `/auth/logout` to revoke refresh tokens when you implement persistence.

---

14) Testing Stub (pytest)

tests/test_auth.py

```
from fastapi.testclient import TestClient
from app.main import app

client = TestClient(app)

def test_login_success():
    r = client.post("/auth/login", data={"username": "admin@example.com",
```

```
"password": "admin"})
    assert r.status_code == 200
    assert "access_token" in r.json()
```

---

15) Adding More Apps under One Swagger

Create new routers (e.g., `claims.py`, `billing.py`) and include them in `main.py` with `prefix` and `tags`. They'll appear as new groups automatically:

```
from fastapi import APIRouter
claims = APIRouter(prefix="/claims", tags=["Claims"])  # new group
```

This keeps **all APIs in one Swagger GUI**, cleanly categorized by tag groups.