FastAPI JWT Authentication Explanation and Code

Explanation of FastAPI JWT Authentication Code:

1. Imports:

- from fastapi import APIRouter, Depends, HTTPException, status: Imports FastAPI classes/functions for routing, dependencies, exceptions, and HTTP status codes.
- from sqlalchemy.orm import Session: Allows database session handling using SQLAlchemy ORM.
- from fastapi.security import OAuth2PasswordRequestForm, OAuth2PasswordBearer: Imports OAuth2 form handling and bearer token mechanism.
- from app.models.UserCreationModel import User: Imports the User model from the apps user model.
- from app.core.security import verify_password, create_access_token: Custom functions to verify hashed passwords and create JWT tokens.
- from app.function.login import *: Imports login-related helper functions.
- from jose import jwt, JWTError: Used to encode and decode JWTs and handle errors.
- from app.core.config import settings: Loads app configurations including the secret key and algorithm.
- from app.core.database import get_db: Dependency to get a database session.
- from datetime import timedelta, datetime: Used for handling time-related tasks.
- from typing import List: Allows use of generic types like List.

2. OAuth2 Scheme Definition:

- oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/login"): Sets up the OAuth2 scheme and says tokens will be retrieved from the `/login` route.

3. Router & Token Blacklist Initialization:

- router = APIRouter(): Initializes the FastAPI router.
- blacklisted_tokens = set(): Used to store tokens that are "logged out".

4. /login Endpoint:

- Takes form data (username and password) and a DB session.
- Checks the users email and verifies the password.
- If valid, sets token expiry based on remember_me flag (30 days vs. 1 hour).
- Updates last activity and commits to DB.

- Returns a JWT token with user details encoded.

5. get_current_user Function:

- Decodes the token, verifies it.
- Gets user from DB by email in token.
- Returns user or raises an error if invalid.

6. /me Endpoint:

- Depends on get_current_user to get the current logged-in user.
- Updates their last activity.
- Returns a summary of the user's details.

7. decode_jwt Function:

- Decodes the JWT and returns its payload or raises an error.

8. /logout Endpoint:

- Uses token from the header.
- Decodes token and verifies user exists.
- Checks last activity ensures theyve been active within 1 day.
- Adds token to blacklist and confirms logout.

The JWT mechanism ensures secure login, token-based session handling, and user data access using proper authorization.

Code:

```
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from fastapi.security import OAuth2PasswordRequestForm, OAuth2PasswordBearer
from app.models.UserCreationModel import User
from app.core.security import verify_password, create_access_token
from app.function.login import *
from jose import jwt, JWTError
from app.core.config import settings
from app.core.database import get_db
from datetime import timedelta, datetime
from typing import List
from app.core.config import settings

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/login")
router = APIRouter()
blacklisted_tokens = set()

@router.post("/login")
def login(
    form_data: OAuth2PasswordRequestForm = Depends(),
    db: Session = Depends(get_db),
    remember_me: bool = False
):
    db_user = db.query(User).filter(User.email == form_data.username).first()
    if not db_user or not verify_password(form_data.password, db_user.hashed_password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(days=30) if remember_me else timedelta(hours=1)
    db_user.last_activity = datetime.now()
    db.commit()
    access_token = create_access_token(
                                        data={"sub":     db_user.email,     "role":     db_user.role,     "id":
db_user.id,"client_id":db_user.client_id,"username":db_user.name},
        expires_delta=access_token_expires
    )
    return {
        "access_token": access_token,
        "token_type": "bearer",
    }

def get_current_user(token: str = Depends(oauth2_scheme), db: Session = Depends(get_db)):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Invalid authentication credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, settings.SECRET_KEY, algorithms=["HS256"])
        email: str = payload.get("sub")
        if email is None:
            raise credentials_exception
    except JWTError:
        raise credentials_exception
    user = db.query(User).filter(User.email == email).first()
    if user is None:
        raise credentials_exception
    return user

@router.get("/me")
def read_users_me(current_user: User = Depends(get_current_user), db: Session = Depends(get_db)):
    current_user.last_activity = datetime.utcnow()
    db.commit()
    return {
        "user_id": current_user.id,
        "client_id": current_user.client_id,
        "email": current_user.email,
        "role": current_user.role,
        "status": current_user.status
    }

def decode_jwt(token: str):
    try:
        payload = jwt.decode(token,settings. SECRET_KEY, algorithms=[settings.ALGORITHM])
        return payload
    except JWTError:
        raise HTTPException(status_code=400, detail="Invalid token")

@router.post("/logout")
def logout(token: str = Depends(oauth2_scheme), db: Session = Depends(get_db)):
    if token in blacklisted_tokens:
        raise HTTPException(status_code=400, detail="Token already invalidated")
    try:
        token_data = decode_jwt(token)
        db_user = db.query(User).filter(User.email == token_data["sub"]).first()
        if db_user is None:
            raise HTTPException(status_code=400, detail="Invalid token")
        if db_user.last_activity and (datetime.now() - db_user.last_activity) > timedelta(days=1):
            raise HTTPException(status_code=400, detail="User has been inactive for too long")
    except HTTPException as e:
        raise e
    except Exception:
        raise HTTPException(status_code=400, detail="Error decoding token or finding user")
    blacklisted_tokens.add(token)
    return {"message": "Successfully logged out"}
```