Auth Development — Full Chronological Code Changelog

This document collects the code snippets and decisions you and I iterated on from Day 1 through the current session. It includes the key files, explanations for changes, and the code exactly as pasted during the conversation where available.

1) Initial database snippet (SQLAlchemy engine & session)

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from dotenv import load_dotenv
import os

load_dotenv()

DATABASE_URL = os.getenv("DATABASE_URL")

engine = create_engine(DATABASE_URL, pool_pre_ping=True)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

2) Debugged database.py with Base and get_db (final working db file posted by you)

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base
from dotenv import load_dotenv, find_dotenv
import os

# Load .env
load_dotenv(find_dotenv())

# Get DB URL
DATABASE_URL = os.getenv("DATABASE_URL")
print("DATABASE_URL =", DATABASE_URL)

# Create engine
engine = create_engine(DATABASE_URL, pool_pre_ping=True)

# Base (IMPORTANT!)
Base = declarative_base()

# Session
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

# Dependency for routes
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

3) main.py (app startup and router includes) — version you shared later

```
from fastapi import FastAPI
from .core.database import engine
from .models.base import Base
```

```
# Import routers
from .routes import auth, user, organization, product

app = FastAPI()

# Create all tables
Base.metadata.create_all(bind=engine)

@app.get("/health")
def health_check():
    return {"status": "ok"}

# Include API routes
app.include_router(auth.router, prefix="/auth", tags=["Auth"])
app.include_router(user.router, prefix="/users", tags=["Users"])
app.include_router(organization.router, prefix="/organizations", tags=["Organizations"])
app.include_router(product.router, prefix="/products", tags=["Products"])
```

4) Simple product test route (product router)

```
from fastapi import APIRouter

router = APIRouter()

@router.get("/test")
def test_product():
    return {"msg": "product ok"}
```

5) Simple user router test route (you posted)

```
from fastapi import APIRouter

router = APIRouter()

@router.get("/test")
def test_user():
    return {"msg": "user ok"}
```

6) Auth routes (register/login) you provided

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from app.models.user import User
from app.core.database import get_db
from app.core.security import hash_password, verify_password, create_access_token
from pydantic import BaseModel

router = APIRouter()

class RegisterSchema(BaseModel):
    email: str
    password: str

class LoginSchema(BaseModel):
    email: str
    password: str


@router.post("/register")
def register_user(data: RegisterSchema, db: Session = Depends(get_db)):
    existing = db.query(User).filter(User.email == data.email).first()
    if existing:
        raise HTTPException(status_code=400, detail="Email already registered")
    new_user = User(
        email=data.email,
        password_hash=hash_password(data.password)
    )
    db.add(new_user)
    db.commit()
    db.refresh(new_user)

    return {"message": "User registered", "user_id": new_user.id}


@router.post("/login")
def login_user(data: LoginSchema, db: Session = Depends(get_db)):
    user = db.query(User).filter(User.email == data.email).first()
    if not user:
        raise HTTPException(status_code=400, detail="Invalid credentials")

    if not verify_password(data.password, user.password_hash):
        raise HTTPException(status_code=400, detail="Invalid credentials")

    token = create_access_token({"sub": str(user.id)})

    return {"access_token": token, "token_type": "bearer"}
```

7) Protected user endpoint (get current user) you posted

```
from fastapi import APIRouter, Depends
from app.core.security import get_current_user

router = APIRouter()

@router.get("/me")
def get_me(current_user = Depends(get_current_user)):
    return {"id": current_user.id, "email": current_user.email}
```

8) A main auth/login example payload you used in Try it out (JSON body)

```
{
  "email": "test@example.com",
  "password": "pass123"
}
```

9) Debug logs and errors you encountered (selected excerpts) — helpful to reproduce issues

```
- NameError: name 'DATABASE_URL' is not defined  (fixed by moving load_dotenv/find_dotenv earlier
)
- psycopg2.OperationalError: password authentication failed for user "erp" (fixed by ensuring doc
ker Postgres credentials and .env matched)
- sqlalchemy.exc.ProgrammingError: column users.password_hash does not exist  (schema mismatch —
ran create_all; previously DB had different columns)
- passlib bcrypt ValueError: password cannot be longer than 72 bytes (observed when passing very
long password strings)
- ImportError: cannot import name 'get_current_user' from app.core.security (fix: implement and e
xport the helper in security.py)
- NameError: name 'Depends' is not defined in main.py (fix: import Depends where used)
- OpenAPI / Swagger: OAuth2PasswordBearer shows password flow; to authorize use the token returne
d from /auth/login as Bearer token in the Authorize dialog.
```

Appendix — Suggested Next Steps

- Add Alembic migrations and generate a migration that matches your models to avoid schema vs DB drift.
- Move secret keys into environment (.env) and never commit them.
- Add integration tests for auth endpoints (register/login/me).
- Harden password policy and add rate-limiting for auth endpoints.
- Export ERD and security diagrams as separate PNGs if you want to embed them into other docs.