Description of the issue — what happened, expected vs. actual behavior.
Databases include no data after auto deployment. Requires manually connect to the database and insert the data every time. This cause inconvenience and hard to achieve simple deployment objective. Yes I can sign up for a new user, however; new user have no permission doing anything – There's no predefined admin account, admin role, etc.

Expected:
Databases include predefined data after running the deployment commands.

Actual:
Manual data insertion is required after deployment.

Environment & setup details at the time of the issue.
Backend/app/main.py:
The following action: backend startup event not included in earlier version

```
@app.on_event("startup")
Line54Line57


async def startup_event():
```

Steps to reproduce the issue.
Remove the following section.

```
@app.on_event("startup")
async def startup_event():
    await wait_for_db()

    async with async_session_maker() as session:
        await wait_for_migrations_to_complete(session)
        await seed_roles(session)
        await seed_categories(session)
        await seed_users(session, await
anext(get_user_manager(SQLAlchemyUserDatabase(session, User))))
```

Diagnosis — your interpretation of what caused the problem.
Some kind of auto data insertion during the backup startup required.

Research process — resources you consulted (documentation, blogs, forums, AI tools, videos).
Using AI for hint, and noticed a "seeding" file is required:

Related resources:

https://learn.microsoft.com/en-us/ef/core/modeling/data-seeding

Resolution steps — what finally worked. Outcome verification — how you confirmed the issue was resolved.
Add the following event after startup in main.py

```
@app.on_event("startup")
async def startup_event():
    async with async_session_maker() as session:
        await wait_for_migrations_to_complete(session)
        await seed_roles(session)
        await seed_categories(session)
        await seed_users(session, await
anext(get_user_manager(SQLAlchemyUserDatabase(session, User))))
```

Take user seeding, as example:

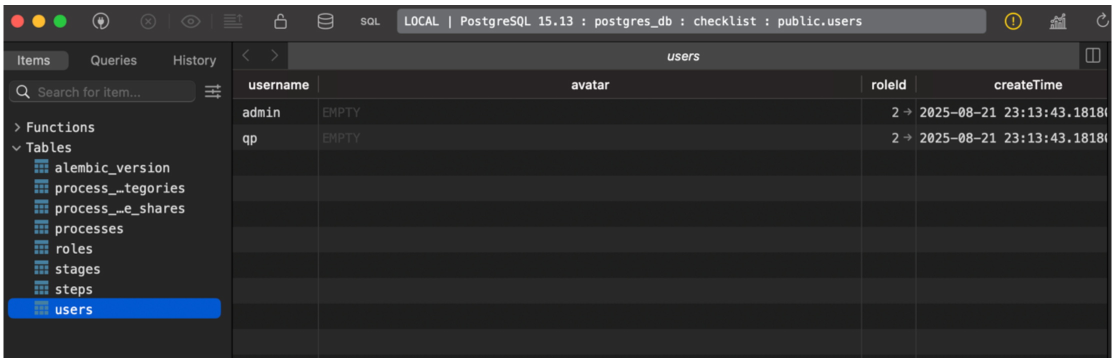
```
mock_roles = [
    {
        "id": 1,
        "name": "user",
        "description": "Regular User",
        "addPeople": False,
        "deletePeople": False,
        "editPeople": False,
        "editPassword": False,
        "addRole": False,
        "editRole": False,
        "shareChecklist": True,
        "shareAnyChecklist": False,
        "editChecklist": True,
        "editAnyChecklist": False,
        "deleteChecklist": True,
        "deleteAnyChecklist": False,
        "addChecklist": True,
    },
```

```
mock_users = [
```

```
    {
        "id": uuid4(),
        "username": "admin",
        "avatar": "",
        "roleId": 2,
        "email": "admin@example.com",
        "password": "123456",
    }
```
….

```
async def seed_users(session: AsyncSession, manager: UserManager):
    existing_users_result = await session.execute(select(User.email))
    existing_emails = set(existing_users_result.scalars().all())

    for user_data in mock_users:
        if user_data["email"] in existing_emails:
            continue

        hashed_password = await
manager.hash_password(user_data.pop("password"))

        user = User(
            **user_data,
            hashed_password=hashed_password,
            is_active=True,
            is_verified=True,
            is_superuser=False,
        )
        session.add(user)

    await session.commit()
```

Outcome verification:
Admin user and testing user added after backend startup.



LOCAL | PostgreSQL 15.13 : postgres_db : checklist : public.users

Tables: alembic_version, process_…tegories, process_…e_shares, processes, roles, stages, steps, users

username	avatar	roleId	createTime
admin	EMPTY	2 →	2025-08-21 23:13:43.1818
qp	EMPTY	2 →	2025-08-21 23:13:43.1818

Description

When executing Docker compose up build for the first time, it is expected that all services should start sequentially and complete database migration. But the backend container crashed immediately during the startup phase, and the error log showed that it could not connect to PostgreSQL, resulting in the interruption of ALEMBIC migration and seed data insertion.

Environment & Setup
Docker Compose runs four services: backend, frontend, db, and migration
The startup event of the initial code did not wait for the database to be fully ready, but directly created the Asynchronous Session and checked the migration

Steps to Reproduce
Execute Docker
    compose up -- build
in the root directory of the project
When the db container is still initializing, the backend container starts and runs startup_ event
The backend immediately attempts to access the unprepared database and throws an OperalError: could not connect to the server

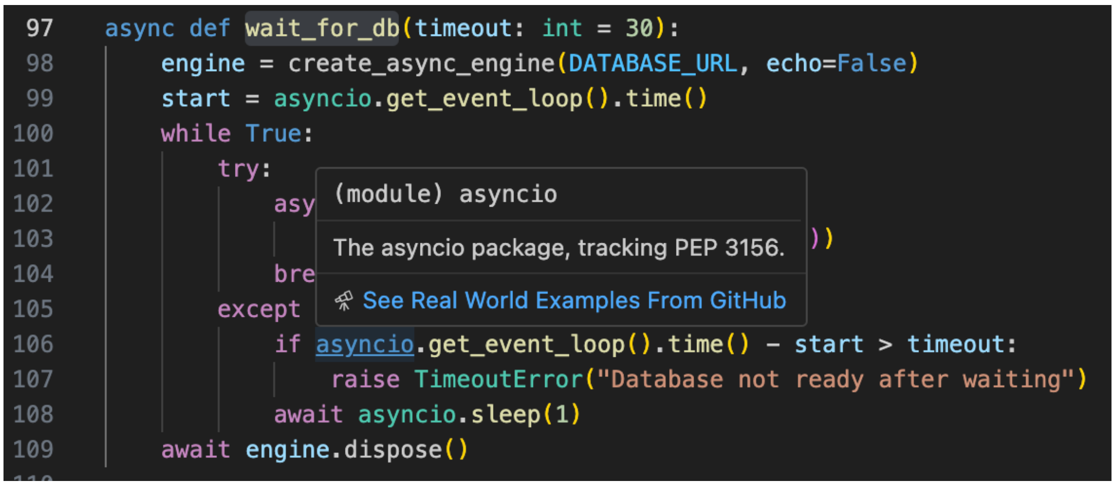
Diagnosis
Depends_on only guarantees the order of container startup and does not wait for the database port to open. Startup_ event executed async_dession_maker () before the database could be connected, causing the first interaction with the database (checking migration, inserting seeds) to crash due to connection failure.

Research Process
Refer to the FastAPI documentation and SQLAlchemy asynchronous session initialization method.
https://fastapi.tiangolo.com/#check-it
Search for solutions to Docker Compose and PostgreSQL initialization issues (Stack Overflow, official docs).
https://stackoverflow.com/questions/31746182/docker-compose-wait-for-container-x-before-starting-y
https://stackoverflow.com/questions/31746182/docker-compose-wait-for-container-x-before-starting-y/41854997#41854997

Use AI tools to confirm the habitual pattern of waiting for the database to be available before starting the application.

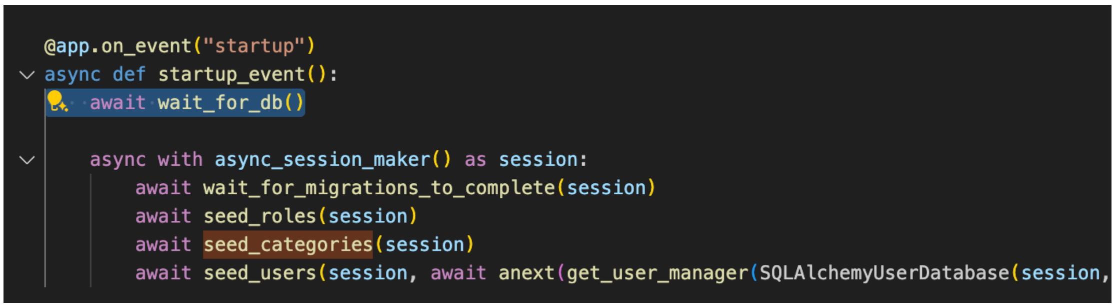
Resolution Steps
1，Implement wait_for.db in backend/app/seed/seed. py and ensure PostgreSQL accepts connections through loop connection testing
2，Call await wait_for_db() at the beginning of startup_ event, first verify that the database port is open, then create a session and check the migration
3，Build and execute Docker compose up – build

Wait for db added:



```
async def wait_for_db(timeout: int = 30):
    engine = create_async_engine(DATABASE_URL, echo=False)
    start = asyncio.get_event_loop().time()
    while True:
        try:
            asy
                                                  ))
            bre
        except
            if asyncio.get_event_loop().time() - start > timeout:
                raise TimeoutError("Database not ready after waiting")
            await asyncio.sleep(1)
    await engine.dispose()
```

(module) asyncio
The asyncio package, tracking PEP 3156.
See Real World Examples From GitHub



```
@app.on_event("startup")
async def startup_event():
    await wait_for_db()

    async with async_session_maker() as session:
        await wait_for_migrations_to_complete(session)
        await seed_roles(session)
        await seed_categories(session)
        await seed_users(session, await anext(get_user_manager(SQLAlchemyUserDatabase(session,
```

Outcome Verification
The log shows that the polling information waiting for the database was printed first, and then the migration and seed data insertion were successfully executed.
Docker PS visible backend container keeps running, access http://localhost:8000/
Return to health response.
This fix ensures that the backend service continues to execute subsequent logic only after the database is fully ready, thereby avoiding startup contention issues during initial deployment.

Reference:

FASTAPI. FastAPI. (n.d.). https://fastapi.tiangolo.com/#check-it

AndriySvyryd. (n.d.). *Data seeding - EF core*. EF Core | Microsoft Learn.
https://learn.microsoft.com/en-us/ef/core/modeling/data-seeding

Docker compose wait for container x before starting Y. Stack Overflow. (2015, July 1). https://stackoverflow.com/questions/31746182/docker-compose-wait-for-container-x-before-starting-y