Alembic revisions + SQLAlchemy model stubs

Target repo path: `/Users/sheriftito/Downloads/psychsync/backend/app`

This document contains ready-to-drop files (Alembic `env.py`, two first revisions, and SQLAlchemy model stubs + DB session/base) tailored to the project layout under `backend/app`.

> **Important:** Do not copy-paste everything into a single file — create the files in the paths shown below.

File: `backend/app/db/base.py`

```
# backend/app/db/base.py
import sqlalchemy as sa
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    """Central declarative base for all models. Alembic will import
Base.metadata."""
    pass
```

File: `backend/app/db/session.py`

```
# backend/app/db/session.py
import os
from sqlalchemy.ext.asyncio import create_async_engine, async_sessionmaker
from sqlalchemy.orm import sessionmaker
from .base import Base

DATABASE_URL = os.getenv(
    "DATABASE_URL",
    "postgresql+asyncpg://psychsync:psychsync@localhost:5432/psychsync",
)

# Async engine for FastAPI
engine = create_async_engine(DATABASE_URL, future=True, pool_pre_ping=True)
AsyncSessionLocal = async_sessionmaker(engine, expire_on_commit=False)

# Convenience: sync engine (for Alembic autogenerate if you prefer run_sync)
# from sqlalchemy import create_engine
# sync_engine = create_engine(DATABASE_URL.replace("+asyncpg", ""))
```

File: `backend/app/db/models/__init__.py`

```
# backend/app/db/models/__init__.py
# Import model modules so Base.metadata collects them for Alembic autogenerate
from .organization import Organization
from .user import User
from .role import Role
from .org_member import OrgMember
from .team import Team
from .team_member import TeamMember
from .framework import Framework
from .question import Question
from .question_option import QuestionOption
from .assessment import Assessment
from .response import Response
from .score import Score
from .invitation import Invitation
from .audit_log import AuditLog
```

File: `backend/app/db/models/organization.py`

```
# backend/app/db/models/organization.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID
from ..base import Base

class Organization(Base):
    __tablename__ = "organizations"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    name = sa.Column(sa.Text, nullable=False)
    slug = sa.Column(sa.Text, nullable=False, unique=True)
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
    updated_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
    deleted_at = sa.Column(sa.TIMESTAMP(timezone=True), nullable=True)
```

File: `backend/app/db/models/user.py`

```
# backend/app/db/models/user.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID, CITEXT
from ..base import Base

class User(Base):
    __tablename__ = "users"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    email = sa.Column(CITEXT, nullable=False, unique=True)
    password_hash = sa.Column(sa.Text, nullable=False)
    full_name = sa.Column(sa.Text, nullable=True)
    avatar_url = sa.Column(sa.Text, nullable=True)
    is_active = sa.Column(sa.Boolean, nullable=False,
server_default=sa.text("true"))
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
    updated_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
    deleted_at = sa.Column(sa.TIMESTAMP(timezone=True), nullable=True)
```

File: `backend/app/db/models/role.py`

```
# backend/app/db/models/role.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID
from ..base import Base

class Role(Base):
    __tablename__ = "roles"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    scope = sa.Column(sa.Text, nullable=False)
    code = sa.Column(sa.Text, nullable=False)
    __table_args__ = (sa.UniqueConstraint('scope', 'code',
name='uq_roles_scope_code'),)
```

File: `backend/app/db/models/org_member.py`

```
# backend/app/db/models/org_member.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID
from ..base import Base

class OrgMember(Base):
    __tablename__ = "org_members"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    org_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('organizations.id',
ondelete='CASCADE'), nullable=False)
    user_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('users.id',
ondelete='CASCADE'), nullable=False)
    role_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('roles.id'),
nullable=False)
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
    __table_args__ = (sa.UniqueConstraint('org_id', 'user_id',
name='uq_org_user'),)
```

File: `backend/app/db/models/team.py`

```
# backend/app/db/models/team.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID
from ..base import Base

class Team(Base):
    __tablename__ = "teams"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    org_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('organizations.id',
ondelete='CASCADE'), nullable=False)
    name = sa.Column(sa.Text, nullable=False)
    description = sa.Column(sa.Text, nullable=True)
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
    updated_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
    deleted_at = sa.Column(sa.TIMESTAMP(timezone=True), nullable=True)
    __table_args__ = (sa.UniqueConstraint('org_id', 'name',
name='uq_team_org_name'),)
```

File: `backend/app/db/models/team_member.py`

```
# backend/app/db/models/team_member.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID
from ..base import Base

class TeamMember(Base):
    __tablename__ = "team_members"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    team_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('teams.id',
ondelete='CASCADE'), nullable=False)
    user_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('users.id',
ondelete='CASCADE'), nullable=False)
    role_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('roles.id'),
nullable=False)
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
    __table_args__ = (sa.UniqueConstraint('team_id', 'user_id',
name='uq_team_user'),)
```

File: `backend/app/db/models/framework.py`

```
# backend/app/db/models/framework.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID
from ..base import Base

class Framework(Base):
    __tablename__ = "frameworks"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    code = sa.Column(sa.Text, nullable=False, unique=True)
    name = sa.Column(sa.Text, nullable=False)
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
```

```
    updated_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
```

File: `backend/app/db/models/question.py`

```
# backend/app/db/models/question.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID
from ..base import Base

class Question(Base):
    __tablename__ = "questions"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    framework_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('frameworks.id',
ondelete='CASCADE'), nullable=False)
    code = sa.Column(sa.Text, nullable=True)
    body = sa.Column(sa.Text, nullable=False)
    kind = sa.Column(sa.Text, nullable=False)
    position = sa.Column(sa.Integer, nullable=False,
server_default=sa.text('0'))
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
```

File: `backend/app/db/models/question_option.py`

```
# backend/app/db/models/question_option.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID
from ..base import Base

class QuestionOption(Base):
    __tablename__ = "question_options"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    question_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('questions.id',
ondelete='CASCADE'), nullable=False)
    label = sa.Column(sa.Text, nullable=False)
    value = sa.Column(sa.Text, nullable=False)
    weight = sa.Column(sa.Numeric, nullable=True)
    position = sa.Column(sa.Integer, nullable=False,
server_default=sa.text('0'))
```

File: `backend/app/db/models/assessment.py`

```
# backend/app/db/models/assessment.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID
from ..base import Base

class Assessment(Base):
    __tablename__ = "assessments"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    org_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('organizations.id',
ondelete='CASCADE'), nullable=False)
    team_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('teams.id',
ondelete='SET NULL'), nullable=True)
    user_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('users.id',
ondelete='CASCADE'), nullable=False)
    framework_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('frameworks.id',
ondelete='RESTRICT'), nullable=False)
    status = sa.Column(sa.Text, nullable=False)
    started_at = sa.Column(sa.TIMESTAMP(timezone=True), nullable=True)
    completed_at = sa.Column(sa.TIMESTAMP(timezone=True), nullable=True)
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
```

File: `backend/app/db/models/response.py`

```
# backend/app/db/models/response.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID, JSONB
from ..base import Base

class Response(Base):
    __tablename__ = "responses"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    assessment_id = sa.Column(UUID(as_uuid=True),
sa.ForeignKey('assessments.id', ondelete='CASCADE'), nullable=False)
    question_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('questions.id',
ondelete='CASCADE'), nullable=False)
    answer_json = sa.Column(JSONB, nullable=False)
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
    __table_args__ = (sa.UniqueConstraint('assessment_id', 'question_id',
name='uq_response_assessment_question'),)
```

File: `backend/app/db/models/score.py`

```
# backend/app/db/models/score.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID
from ..base import Base

class Score(Base):
    __tablename__ = "scores"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    assessment_id = sa.Column(UUID(as_uuid=True),
sa.ForeignKey('assessments.id', ondelete='CASCADE'), nullable=False)
    dimension = sa.Column(sa.Text, nullable=False)
    value = sa.Column(sa.Numeric, nullable=False)
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
    __table_args__ = (sa.UniqueConstraint('assessment_id', 'dimension',
name='uq_score_assessment_dimension'),)
```

File: `backend/app/db/models/invitation.py`

```
# backend/app/db/models/invitation.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID, CITEXT
from ..base import Base

class Invitation(Base):
    __tablename__ = "invitations"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
```

```
    org_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('organizations.id',
ondelete='CASCADE'), nullable=False)
    email = sa.Column(CITEXT, nullable=False)
    token = sa.Column(sa.Text, nullable=False, unique=True)
    expires_at = sa.Column(sa.TIMESTAMP(timezone=True), nullable=False)
    accepted_at = sa.Column(sa.TIMESTAMP(timezone=True), nullable=True)
```

File: `backend/app/db/models/audit_log.py`

```
# backend/app/db/models/audit_log.py
import sqlalchemy as sa
from sqlalchemy.dialects.postgresql import UUID, JSONB
from ..base import Base

class AuditLog(Base):
    __tablename__ = "audit_logs"

    id = sa.Column(UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()"))
    org_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('organizations.id',
ondelete='CASCADE'), nullable=False)
    actor_user_id = sa.Column(UUID(as_uuid=True), sa.ForeignKey('users.id',
ondelete='SET NULL'), nullable=True)
    action = sa.Column(sa.Text, nullable=False)
    entity = sa.Column(sa.Text, nullable=False)
    entity_id = sa.Column(UUID(as_uuid=True), nullable=True)
    meta = sa.Column(JSONB, nullable=True)
    created_at = sa.Column(sa.TIMESTAMP(timezone=True),
server_default=sa.text("NOW()"), nullable=False)
```

Alembic: `env.py` (async)

File: `backend/app/alembic/env.py`

```
# backend/app/alembic/env.py
import os
from logging.config import fileConfig
from sqlalchemy import pool
from alembic import context

# bring in Base metadata
from app.db.base import Base
```

```
# import models package so migrations can autogenerate
import app.db.models  # noqa

config = context.config
fileConfig(config.config_file_name)

target_metadata = Base.metadata


def get_url():
    return os.getenv("DATABASE_URL")


def run_migrations_offline():
    url = get_url()
    context.configure(url=url, target_metadata=target_metadata,
literal_binds=True)
    with context.begin_transaction():
        context.run_migrations()


def run_migrations_online():
    from sqlalchemy.ext.asyncio import async_engine_from_config
    connectable = async_engine_from_config(
        {"sqlalchemy.url": get_url()}, prefix="sqlalchemy.",
poolclass=pool.NullPool
    )

    async def do_run_migrations():
        async with connectable.connect() as connection:
            await connection.run_sync(lambda conn:
context.configure(connection=conn, target_metadata=target_metadata))
            await connection.run_sync(lambda conn: context.begin_transaction())
            await connection.run_sync(lambda conn: context.run_migrations())

    import asyncio
    asyncio.run(do_run_migrations())


if context.is_offline_mode():
    run_migrations_offline()
else:
    run_migrations_online()
```

Alembic revision 1: extensions + triggers

File: `backend/app/alembic/versions/0001_extensions_triggers.py`

```
# backend/app/alembic/versions/0001_extensions_triggers.py
from alembic import op
import sqlalchemy as sa

revision = '0001_extensions_triggers'
down_revision = None
branch_labels = None
depends_on = None


def upgrade():
    # Enable extensions
    op.execute('CREATE EXTENSION IF NOT EXISTS pgcrypto;')
    op.execute('CREATE EXTENSION IF NOT EXISTS citext;')

    # set_updated_at trigger
    op.execute("""
    CREATE OR REPLACE FUNCTION set_updated_at()
    RETURNS TRIGGER AS $$
    BEGIN
      NEW.updated_at = NOW();
      RETURN NEW;
    END;$$ LANGUAGE plpgsql;
    """)


def downgrade():
    op.execute('DROP FUNCTION IF EXISTS set_updated_at() CASCADE;')
    # NOTE: we intentionally do not DROP extensions here to avoid removing them
if other DB objects need them.
```

Alembic revision 2: core tables

File: `backend/app/alembic/versions/0002_core_tables.py`

```
# backend/app/alembic/versions/0002_core_tables.py
from alembic import op
import sqlalchemy as sa
from sqlalchemy.dialects import postgresql
```

```
revision = '0002_core_tables'
down_revision = '0001_extensions_triggers'
branch_labels = None
depends_on = None


def upgrade():
    # organizations
    op.create_table(
        'organizations',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('name', sa.Text(), nullable=False),
        sa.Column('slug', sa.Text(), nullable=False, unique=True),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.Column('updated_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.Column('deleted_at', sa.TIMESTAMP(timezone=True), nullable=True),
    )

    # users
    op.create_table(
        'users',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('email', postgresql.CITEXT(), nullable=False, unique=True),
        sa.Column('password_hash', sa.Text(), nullable=False),
        sa.Column('full_name', sa.Text(), nullable=True),
        sa.Column('avatar_url', sa.Text(), nullable=True),
        sa.Column('is_active', sa.Boolean(), nullable=False,
server_default=sa.text('true')),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.Column('updated_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.Column('deleted_at', sa.TIMESTAMP(timezone=True), nullable=True),
    )

    # roles
    op.create_table(
        'roles',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('scope', sa.Text(), nullable=False),
        sa.Column('code', sa.Text(), nullable=False),
        sa.UniqueConstraint('scope', 'code', name='uq_roles_scope_code'),
    )

    # org_members
    op.create_table(
        'org_members',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('org_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('organizations.id', ondelete='CASCADE'), nullable=False),
        sa.Column('user_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('users.id', ondelete='CASCADE'), nullable=False),
        sa.Column('role_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('roles.id'), nullable=False),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.UniqueConstraint('org_id', 'user_id', name='uq_org_user'),
    )

    # teams
    op.create_table(
        'teams',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('org_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('organizations.id', ondelete='CASCADE'), nullable=False),
        sa.Column('name', sa.Text(), nullable=False),
        sa.Column('description', sa.Text(), nullable=True),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.Column('updated_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.Column('deleted_at', sa.TIMESTAMP(timezone=True), nullable=True),
        sa.UniqueConstraint('org_id', 'name', name='uq_team_org_name'),
    )

    # team_members
    op.create_table(
        'team_members',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('team_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('teams.id', ondelete='CASCADE'), nullable=False),
        sa.Column('user_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('users.id', ondelete='CASCADE'), nullable=False),
        sa.Column('role_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('roles.id'), nullable=False),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.UniqueConstraint('team_id', 'user_id', name='uq_team_user'),
    )

    # frameworks
    op.create_table(
        'frameworks',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('code', sa.Text(), nullable=False, unique=True),
        sa.Column('name', sa.Text(), nullable=False),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.Column('updated_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
    )

    # questions
    op.create_table(
        'questions',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('framework_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('frameworks.id', ondelete='CASCADE'), nullable=False),
        sa.Column('code', sa.Text(), nullable=True),
        sa.Column('body', sa.Text(), nullable=False),
        sa.Column('kind', sa.Text(), nullable=False),
        sa.Column('position', sa.Integer(), nullable=False,
server_default=sa.text('0')),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
    )

    # question_options
    op.create_table(
        'question_options',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('question_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('questions.id', ondelete='CASCADE'), nullable=False),
        sa.Column('label', sa.Text(), nullable=False),
        sa.Column('value', sa.Text(), nullable=False),
        sa.Column('weight', sa.Numeric(), nullable=True),
        sa.Column('position', sa.Integer(), nullable=False,
server_default=sa.text('0')),
    )

    # assessments
    op.create_table(
        'assessments',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('org_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('organizations.id', ondelete='CASCADE'), nullable=False),
        sa.Column('team_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('teams.id', ondelete='SET NULL'), nullable=True),
        sa.Column('user_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('users.id', ondelete='CASCADE'), nullable=False),
        sa.Column('framework_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('frameworks.id', ondelete='RESTRICT'), nullable=False),
        sa.Column('status', sa.Text(), nullable=False),
        sa.Column('started_at', sa.TIMESTAMP(timezone=True), nullable=True),
        sa.Column('completed_at', sa.TIMESTAMP(timezone=True), nullable=True),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
    )

    # responses
    op.create_table(
        'responses',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('assessment_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('assessments.id', ondelete='CASCADE'), nullable=False),
        sa.Column('question_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('questions.id', ondelete='CASCADE'), nullable=False),
        sa.Column('answer_json', postgresql.JSONB(), nullable=False),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.UniqueConstraint('assessment_id', 'question_id',
name='uq_response_assessment_question'),
    )

    # scores
    op.create_table(
        'scores',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('assessment_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('assessments.id', ondelete='CASCADE'), nullable=False),
        sa.Column('dimension', sa.Text(), nullable=False),
        sa.Column('value', sa.Numeric(), nullable=False),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
        sa.UniqueConstraint('assessment_id', 'dimension',
name='uq_score_assessment_dimension'),
    )
    # invitations
    op.create_table(
        'invitations',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('org_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('organizations.id', ondelete='CASCADE'), nullable=False),
        sa.Column('email', postgresql.CITEXT(), nullable=False),
        sa.Column('token', sa.Text(), nullable=False, unique=True),
        sa.Column('expires_at', sa.TIMESTAMP(timezone=True), nullable=False),
        sa.Column('accepted_at', sa.TIMESTAMP(timezone=True), nullable=True),
    )

    # audit_logs
    op.create_table(
        'audit_logs',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True,
server_default=sa.text("gen_random_uuid()")),
        sa.Column('org_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('organizations.id', ondelete='CASCADE'), nullable=False),
        sa.Column('actor_user_id', postgresql.UUID(as_uuid=True),
sa.ForeignKey('users.id', ondelete='SET NULL'), nullable=True),
        sa.Column('action', sa.Text(), nullable=False),
        sa.Column('entity', sa.Text(), nullable=False),
        sa.Column('entity_id', postgresql.UUID(as_uuid=True), nullable=True),
        sa.Column('meta', postgresql.JSONB(), nullable=True),
        sa.Column('created_at', sa.TIMESTAMP(timezone=True),
server_default=sa.text('NOW()'), nullable=False),
    )

    # Indexes (examples)
    op.create_index('ix_users_email', 'users', ['email'])
    op.create_index('ix_teams_org', 'teams', ['org_id'])
    op.create_index('ix_assessments_lookup', 'assessments', ['org_id',
'team_id', 'user_id', 'framework_id'])


def downgrade():
    # drop in reverse order
    op.drop_index('ix_assessments_lookup', table_name='assessments')
    op.drop_index('ix_teams_org', table_name='teams')
    op.drop_index('ix_users_email', table_name='users')

    op.drop_table('audit_logs')
    op.drop_table('invitations')
    op.drop_table('scores')
    op.drop_table('responses')
    op.drop_table('assessments')
    op.drop_table('question_options')
    op.drop_table('questions')
    op.drop_table('frameworks')
    op.drop_table('team_members')
    op.drop_table('teams')
    op.drop_table('org_members')
    op.drop_table('roles')
    op.drop_table('users')
    op.drop_table('organizations')
```

Quick README / Next steps

1. Place these files under your project: `backend/app/db/...` and `backend/app/alembic/...` (and `backend/app/alembic/versions/`).
2. Ensure `alembic.ini` points `script_location = backend/app/alembic` (or run Alembic from `backend/app` and set `script_location=alembic`).
3. Install dependencies (example):

```
pip install "sqlalchemy[asyncio]" asyncpg alembic python-dotenv
```

1. Export DB URL and run migrations:

```
export DATABASE_URL=postgresql+asyncpg://psychsync:psychsync@localhost:5432/
psychsync
cd /Users/sheriftito/Downloads/psychsync/backend/app
alembic upgrade head
```

1. If you use Docker compose dev DB, point `DATABASE_URL` to the container and run `alembic upgrade head` from the same working directory.

If you want, I can also: - create a seed migration (roles seed), - convert these stubs to Pydantic schemas, - or produce a `Makefile` and `docker-compose.yml` snippet for the dev DB.

Tell me what you'd like next and I'll add it to the canvas.