AeroAspire - SDE Intern

Gokul Krishna S

Week 4 – Day 4 (October 16)

Questions/Reflections:

1. What is a migration? How it works: generating migration file → applying it → version control.

    - A database migration is a structured way to apply changes to your database schema — such as creating new tables, modifying columns, or establishing relationships — while keeping everything under version control.

    - How migrations work:
    - **Generate migration file:** This file records the intended schema change in code form (for example, adding a new column or creating a table). Frameworks like Flask-Migrate, Django, or Sequelize CLI can auto-generate this file from model changes.
    - **Apply (run) the migration:** When you run a command like flask db upgrade or npx sequelize-cli db:migrate, the tool executes the migration on the actual database — altering tables, adding indexes, etc.
    - **Version control:** Every migration file has a unique identifier and is tracked in version control (Git). This ensures your database changes evolve alongside your codebase and can be rolled back if needed.

2. How to seed data: why and how.

    - Seeding means preloading data into your database — often useful for development or to initialize essential records.
    - Why seed data:
    - To populate test data for local or staging environments.

- To add default or reference data, like roles (Admin, User), categories, or countries.
- To speed up setup, letting new developers or automated tests start with the same baseline dataset.
- **How to seed data:**
- Write a seed script or seeder class (for example, python seed.py or a Seeder in CodeIgniter/Supabase).
- Define what data you want to insert.
- Run the script to insert records via standard database commands or the ORM.

- **Example (Python/Flask):**
  from app import db, User
  admin=User(username='admin', email='admin@example.com')
  db.session.add(admin)
  db.session.commit()

3. If you need to add a new column to tasks table after app is in use, how do you do that safely?

- Adding a column while the app is live must be handled carefully to avoid downtime or data consistency issues.
- **Safe process:**
- **Plan schema change via a migration file** — e.g., ALTER TABLE tasks ADD COLUMN priority VARCHAR(20).
- **Add column as nullable first**, without default values; this prevents table rewriting and lockups.
- **Backfill data gradually using a script** — update batches of rows instead of the entire table at once.

- **Update app code in steps:**
- Deploy code that doesn't yet use the column.
- Run migration.
- Backfill data.
- Finally, deploy code that starts using the new field.
- Test thoroughly in staging before production.

- Following this staggered approach ensures zero downtime and prevents app errors (e.g., the app trying to use a column before it exists).