web development & Framework
Unit -3
Migrations

Reference link

- https://laravel.com/docs/9.x/migrations#introduction

Introduction to Migrations

- Migrations are like version control for your database, allowing your team to define and share the application's database schema definition.
- If you have ever had to tell a teammate to manually add a column to their local database schema after pulling in your changes from source control, you've faced the problem that database migrations solve.

- The Laravel Schema facade provides database agnostic support for creating and manipulating tables across all of Laravel's supported database systems. Typically, migrations will use this facade to create and modify database tables and columns.

Artisan migration command

- You may use the make:migration Artisan command to generate a database migration. The new migration will be placed in your database/migrations directory. Each migration filename contains a timestamp that allows Laravel to determine the order of the migrations:

- php artisan make:migration create_flights_table
- Laravel will use the name of the migration to attempt to guess the name of the table and whether or not the migration will be creating a new table. If Laravel is able to determine the table name from the migration name, Laravel will pre-fill the generated migration file with the specified table. Otherwise, you may simply specify the table in the migration file manually.

Migration structure

- A migration class contains two methods: up and down. The up method is used to add new tables, columns, or indexes to your database, while the down method should reverse the operations performed by the up method.

- To run all of your outstanding migrations, execute the migrate Artisan command:

- php artisan migrate
- If you would like to see which migrations have run thus far, you may use the migrate:status Artisan command:

- php artisan migrate:status

Migration structure

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('flights', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('flights');
    }
```

How to create a table using a migration

- To create a new database table, use the create method on the Schema facade. The create method accepts two arguments: the first is the name of the table, while the second is a closure which receives a Blueprint object that may be used to define the new table:

- When creating the table, you may use any of the schema builder's column methods to define the table's columns.

- refer link

byteXL

How to create a table using a migration

```php
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email');
    $table->timestamps();
});
```

Rolling Back Migrations

- To roll back the latest migration operation, you may use the rollback Artisan command. This command rolls back the last "batch" of migrations, which may include multiple migration files:

- php artisan migrate:rollback
- You may roll back a limited number of migrations by providing the step option to the rollback command. For example, the following command will roll back the last five migrations:

- php artisan migrate:rollback --step=5
- The migrate:reset command will roll back all of your application's migrations:

- php artisan migrate:reset

Rolling Back Migrations

- Roll Back & Migrate Using A Single Command
- The migrate:refresh command will roll back all of your migrations and then execute the migrate command. This command effectively re-creates your entire database:

- php artisan migrate:refresh
-
- # Refresh the database and run all database seeds...
- php artisan migrate:refresh --seed
- You may roll back and re-migrate a limited number of migrations by providing the step option to the refresh command. For example, the following command will roll back and re-migrate the last five migrations:

- php artisan migrate:refresh --step=5

Drop All Tables & Migrate

- The migrate:fresh command will drop all tables from the database and then execute the migrate command:

- php artisan migrate:fresh
-
- php artisan migrate:fresh --seed

Seeding

- Laravel also includes a simple way to seed your database with test data using seed classes
- . All seed classes are stored in database/seeds. Seed classes may have any name you wish, but probably should follow some sensible convention, such as UserTableSeeder, etc.
- By default, a DatabaseSeeder class is defined for you. From this class, you may use the call method to run other seed classes, allowing you to control the seeding order.

Seeding

- To seed your database, you may use the db:seed command on the Artisan CLI:

- php artisan db:seed
- By default, the db:seed command runs the DatabaseSeeder class, which may be used to call other seed classes. However, you may use the --class option to specify a specific seeder class to run individually:

- php artisan db:seed --class=UserTableSeeder
- You may also seed your database using the migrate:refresh command, which will also rollback and re-run all of your migrations:

- php artisan migrate:refresh --seed