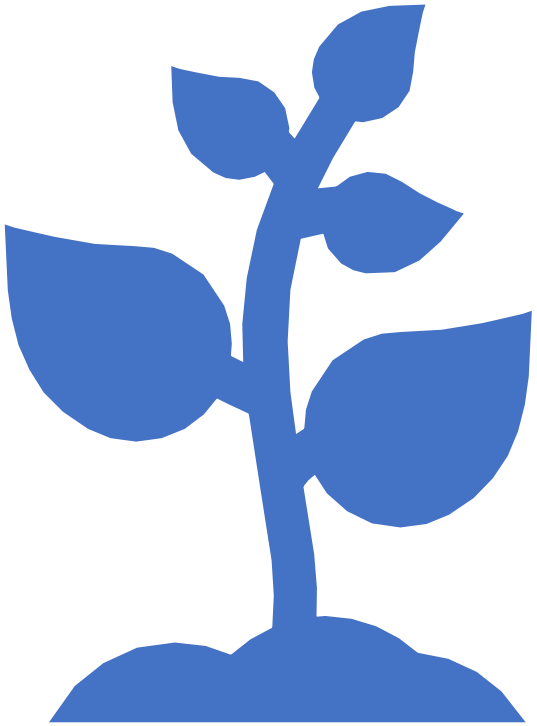
Seeding Databases and Query Building

Last time

- We looked at
  - How to define models
  - How to create databases
  - How to manually add rows
  - How to do the most basic of database queries



Seeding

Why Seeding?

- It is all about sharing and testing
- Ideally we want to give our project to another developer and only need them to set up their development environment
- Then they run a command which sets up the database correctly and enters some test data – or maybe even data which is required when the application is deployed (default accounts for example)
- Seeding is the process of automatically adding data to the database

Run tinker with

```
php artisan make:seeder AnimalTableSeeder
```

This will create the file:

database/seeds/AnimalTableSeeder.php

- This file has a single function which is run to create the seed data
- We need to implement that function

```php
<?php

use Illuminate\Database\Seeder;

class AnimalTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        //
    }
}
```

```php
<?php

use App\Animal;
use Illuminate\Database\Seeder;

class AnimalTableSeeder extends Seeder
{
...
    public function run()
    {
        $a = new Animal;
        $a->name = "Leo";
        $a->weight = 351.6;
        $a->save();
```

- For this to be run during seeding we need to add it to the DatabaseSeeder.php run function
- Like with migrations we need to think carefully about the order of things

```php
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        // $this->call(UsersTableSeeder::class);
        $this->call(AnimalTableSeeder::class);
    }
}
```

Running the Seeder

- Seeding is run with artisan simply ‘php artisan db:seed’
- With seeding enabled we can easily delete our database, run the migrations and seed test data
- It means we don’t have to worry about messing our data up

```
vagrant@homestead:~/Laravel/swanseazoo$ artisan db:seed
Seeding: AnimalTableSeeder
vagrant@homestead:~/Laravel/swanseazoo$ artisan migrate:reset
Rolling back: 2018_10_09_185751_create_animals_table
Rolled back:  2018_10_09_185751_create_animals_table
Rolling back: 2014_10_12_100000_create_password_resets_table
Rolled back:  2014_10_12_100000_create_password_resets_table
Rolling back: 2014_10_12_000000_create_users_table
Rolled back:  2014_10_12_000000_create_users_table
vagrant@homestead:~/Laravel/swanseazoo$ artisan migrate
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table
Migrating: 2018_10_09_185751_create_animals_table
Migrated:  2018_10_09_185751_create_animals_table
vagrant@homestead:~/Laravel/swanseazoo$ artisan db:seed
Seeding: AnimalTableSeeder
```

Model Factories

- Is this really any better than last week? We are still manually creating seeding data.
- Model Factories will create any number of random models for us, with realistic data!

Run tinker with

```
php artisan make:factory AnimalFactory -m Animal
```

This will create the file:

database/factories/AnimalFactory.php

setup for the model Animal.

```php
<?php


use Faker\Generator as Faker;


$factory->define(App\Animal::class, function (Faker $faker) {
    return [
        //
    ];
});
```

- The code in the file created defines a factory that produces a single Animal model on each run
- It returns an array of field name and value pairs
- Faker is a library we can use to generate fake data

The Factory File Completed

```php
<?php

use Faker\Generator as Faker;

$factory->define(App\Animal::class, function (Faker $faker) {
    return [
        'name' => $faker->firstName(),
        'weight' => $faker->randomFloat(2, 300, 500),
    ];
});
```

Faker can generate random first names.

Faker can generate floating point numbers between a minimum (300) and a maximum (500) and which has a fixed precision (2 decimal digits)

Faker

- The documentation with all the features can be found at https://github.com/fzaninotto/Faker
- It can generate fake data for:
  - People
  - Addresses
  - Phone Numbers
  - Companies
  - Random integers, floats, dates and times
  - Etc etc

```
'transferStatus' = $faker->randomElement(['in progress', 'transferred']); // or whatever you want

'projectStatus' = $faker->randomElement(['active', 'completed', 'on hold']);
```

Updating The Seeder To Use The Factory

```
public function run()
{
    $a = new Animal;
    $a->name = "Leo";
    $a->weight = 351.6;
    $a->save();

    factory(App\Animal::class, 50)->create();
}
```

Usually it is good to hard code a few models that you really understand.

Create another 50 Animal models from the registered factory.

To start over (reset, migrate and seed) you use:

```
php artisan migrate:reset
php artisan migrate
php artisan db:seed
```

This sequence of commands is so popular, it can be done with one command:

```
artisan migrate:fresh --seed
```

```
vagrant@homestead:~/Laravel/swanseazoo$ artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table
Migrating: 2018_10_09_185751_create_animals_table
Migrated:  2018_10_09_185751_create_animals_table
Seeding: AnimalTableSeeder
vagrant@homestead:~/Laravel/swanseazoo$
```

Query Building

Laravel’s Query Builder

- Instead of writing SQL statements we use Laravel’s Query Builder which is a fluent set of methods
- Each method returns a fluent query builder which allows you to chain queries
- Note: In the examples below the database table has been accessed directly using it’s name, in practice we would use the model name

```
$user = DB::table('users')->where('name', 'John')->first();
```

This will get all users named John then return the first

get() vs all()

- In my notes and tutorials I always use get… for example Animal::get()
- You might see some people use all() … Animal::all()
- The main difference between get() and all() is that get returns a fluent query builder so you can continue to build a query with it… the all() does not.
- So I recommend using get and pretending all doesn’t exist!

For next lecture – read over your database notes

We are going to be looking at relationships – one to one, one to many, many to many

Tasks

1. Using the application you created last time, create a seeder for your model which adds 5 objects when run.
2. Test your seeder using tinker.
3. Now make a factory to seed realistic data into your database, and test it using seeder.
4. Once your database is seeded try to perform queries using Laravel's query builder.