LARAVEL 10

Part 3: Blades

WHAT IS BLADE FILE

Blade is the simple, yet powerful templating engine that is included with Laravel. Unlike some PHP templating engines, Blade does not restrict you from using plain PHP code in your templates. In fact, all Blade templates are compiled into plain PHP code and cached until they are modified, meaning Blade adds essentially zero overhead to your application. Blade template files use the .blade.php file extension and are typically stored in the resources/views directory.

https://laravel.com/docs/10.x/blade

ADD LAYOUT BLADE

Resources → views

Add new folder “layouts”

Add the blade files like the top menu

INCLUDE

You can call the new layout blade at any blade file like this

```
@include('layouts.header')
```

EXTEND AND YIELD

Create your template blade file and use yield for dynamic content

```
@yield('content')
```

If you want to use this template just you need to extend the file name

```
@extends('index')
```

```
@section('content')
```

Html code

```
@endsection
```

STACK AND PUSH

Inside the blade layout we can add

```
@stack('css')
```

Inside the blade pages

```
@push('css')
<style>
  h1{
    color: red;
  }
</style>
@endpush
```

PREPEND AND PREPENDONCE

Inside the blade pages you can add

```
@prepend('css')
<h2>Hi</h2>
@endprepend
```

@INCLUDEIF , @INCLUDEWHEN @INCLUDEUNLESS , @INCLUDEFIRST ASSIGN VARIABLES

More details

https://laravel.com/docs/10.x/blade

@EACH DIRECTIVE TO FETCH LOOP IN ONE LINE

Instead using @foreach you can cut this code into a separate file amd replace all @foreach code with the below line

```
@each('clients.data', $client, 'data’)
```

And you can file if no data found like below

```
@each('clients.data', $client, 'data, 'clients.nodata')
```

ADD ASSETS (CSS, JS, IMAGES)

Goto public folder and create a folder for the assets and be sure to create sub folder for each asset type

You can call the asset from the blade view file as below

```
<img src="{{ asset('assets/images/blue_jay_bird_nature.jpg') }}"
alt="image bird">
```

DIFFERENCE BETWEEN PUBLIC AND RESOURCES FOLDERS

https://stackoverflow.com/questions/28496301/laravel-5-0-folder-structure-public-vs-resources

CREATE DIRECTIVE

https://arjunamrutiya.medium.com/mastering-laravel-blade-directives-a-step-by-step-guide-8fb36dbe59d1

ADD NEW COLUMN TO DB DURING PRODUCTION

https://medium.com/@laraveltuts/laravel-add-a-new-column-to-existing-table-in-a-migration-fe4b78a3eb92

Then be sure to add in the model fillable

SEEDERS, FACTORIES AND FAKERS

https://laravel.com/docs/10.x/seeding

https://laravel.com/docs/10.x/helpers#method-fake

ELOQUENT RELATIONSHIPS

ONE TO ONE RELATIONSHIP

```php
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class User extends Model
{
    /**
     * Get the phone record associated with the user.
     */
    public function phone()
    {
        return $this->hasOne(Phone::class);
    }
}
```

ONE TO ONE RELATIONSHIP

- Once the relationship is defined, we may retrieve the related record using Eloquent's dynamic properties. Dynamic properties allow you to access relationship methods as if they were properties defined on the model:

```
$phone = User::find(1)->phone;
echo $phone->phone_no;
```

ONE TO ONE RELATIONSHIP

- **Defining The Inverse Of The Relationship**
  - So, we can access the Phone model from our User. Now, let's define a relationship on the Phone model that will let us access the User that owns the phone. We can define the inverse of a hasOne relationship using the belongsTo method:

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class Phone extends Model
{
    /**
     * Get the user that owns the phone.
     */
    public function user()
    {
        return $this->belongsTo('App\Models\User');
    }
}
```

ONE TO ONE RELATIONSHIP

- **Defining The Inverse Of The Relationship**
- Once the relationship is defined, we may retrieve the related record using Eloquent's dynamic properties.

```
$user = Phone::find(1)->user;
    echo $user->name;
```

ELOQUENT ONE TO MANY RELATIONSHIP

A one-to-many relationship is used to define relationships where a single model owns any amount of other models. For example, a customer may have an infinite number of orders. Like all other Eloquent relationships, one-to-many relationships are defined by placing a function on your Eloquent model:

```
class User extends Model
{

    public function orders()
    {
        return $this->hasMany(Order::class);
    }

}
```

ELOQUENT ONE TO MANY RELATIONSHIP

- For this example, Eloquent will assume the foreign key on the Order model is customer_id.
- Once the relationship has been defined, we can access the collection of orders by accessing the order property. Remember, since Eloquent provides "dynamic properties", we can access relationship methods as if they were defined as properties on the model:

```
public function index() {
   $orders = Customer::find(1)->orders;
        foreach ($orders as $order) {
      echo $order->total_amount;
      echo ":";
      echo $order->order_date;
      echo "<br/>";
        }  }
```

ELOQUENT ONE TO MANY RELATIONSHIP

- One To Many (Inverse)
- Let's define a relationship to allow an order to access its parent customer. To define the inverse of a hasMany relationship, define a relationship function on the child model which calls the belongsTo method:

```
class Order extends Model
{

    public function customer()
    {
        return $this->belongsTo(Customer::class);
    }

}
```

ELOQUENT ONE TO MANY RELATIONSHIP

- Once the relationship has been defined, we can retrieve the Customer model for an order by accessing the customer "dynamic property":

```
$order = Order::find(1);
echo $order->customer->name;
```

MANY TO MANY RELATIONSHIP

- Many-to-many relations are slightly more complicated than hasOne and hasMany relationships. An example of such a relationship is a user with many roles, where the roles are also shared by other users. For example, many users may have the role of "Admin".
- To define this relationship, three database tables are needed: users, roles, and role_user. The role_user table is derived from the alphabetical order of the related model names, and contains the user_id and role_id columns:

MANY TO MANY RELATIONSHIP

- First of all, you must create the 3rd table migration which represents many-to-many relationship.
- for our example here, it will be as follow:

```
Schema::create('role_user', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('role_id');
        $table->unsignedBigInteger('user_id');
        $table->foreign('user_id')->references('id')->on('users')
          ->onDelete('cascade')->onUpdate('cascade');
        $table->foreign('role_id')->references('id')->on('roles')
          ->onDelete('cascade')->onUpdate('cascade');

        $table->timestamps();
    });
```

MANY TO MANY RELATIONSHIP

- Many-to-many relationships are defined by writing a method that returns the result of the belongsToMany method. For example, let's define the roles method on our User model:

```
class User extends Model
{
   /**
    * The roles that belong to the user.
    */
   public function roles()
   {
      return $this->belongsToMany(Role::class);
   }}
```

MANY TO MANY RELATIONSHIP

- Once the relationship is defined, you may access the user's roles using the roles dynamic property:

```
$user = User::find(11);
foreach ($user->roles as $role) {
   echo $role->role_name;
    echo "<br/>";}
```

MANY TO MANY RELATIONSHIP

- **Defining The Inverse Of The Relationship**
- To define the inverse of a many-to-many relationship, you place another call to belongsToMany on your related model. To continue our user roles example, let's define the users method on the Role model:

```
class Role extends Model
{
    /**
     * The users that belong to the role.
     */
    public function users()
    {
        return $this->belongsToMany(User::class);
    }
}
```

MANY TO MANY RELATIONSHIP

- **Retrieving Intermediate Table Columns**

- working with many-to-many relations requires the presence of an intermediate table. For example, let's assume our User object has many Role objects that it is related to. After accessing this relationship, we may access the intermediate table using the pivot attribute on the models:

```
$user = App\User::find(1);
foreach ($user->roles as $role) {
    echo $role->pivot->role_id;
}
```

MANY TO MANY RELATIONSHIP

- **Retrieving Intermediate Table Columns**
- Notice that each Role model we retrieve is automatically assigned a pivot attribute. This attribute contains a model representing the intermediate table, and may be used like any other Eloquent model.
- By default, only the model keys will be present on the pivot object. If your pivot table contains extra attributes, you must specify them when defining the relationship:

```
return $this->belongsToMany('App\Role')->withPivot('column1', 'column2');
```

HASONE - HASMANY

The only difference between hasOne and belongsTo is where the foreign key column is located.

Let's say you have two entities: User and an Account.

If the users table has the account_id column then a User belongsTo Account. (And the Account either hasOne or hasMany Users)

But if the users table does not have the account_id column, and instead the accounts table has the user_id column, then User hasOne or hasMany Accounts

In short hasOne and belongsTo are inverses of one another - if one record belongTo the other, the other hasOne of the first. Or, more accurately, eiterh hasOne or hasMany - depending on how many times its id appears.

SHORT FOREIGN KEY RELATIONSHIP IN MIGRATION

Example

```
$table->foreignId('user_id')->constrained('users');
```

RELATIONSHIP HASONETHROUGH , HASMANYTHROUGH

```
return $this->hasOneThrough(
        Owner::class,
        Car::class,
        'mechanic_id', // Foreign key on the cars table...
        'car_id', // Foreign key on the owners table...
);
```

Thank you

KOICA
Korea International Cooperation Agency

السويد
Sverige

ECTD
Egyptian Council
For Training & Development

العمل اللائق للمرأة

منظمة العمل الدولية

هيئة الأمم المتحدة للمرأة

المجلس القومي للمرأة
THE NATIONAL COUNCIL FOR WOMEN