WEB 306 – PHP: Databases and Framework

Day 11 – Eloquent ORM

1. Relational Databases

1.1 About Relational Databases

> "A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables."
>
> — From relational database, TechTarget

Basically, relational databases are databases which use tables with rows and columns to store data which can relate to data in other tables in complex ways.

Database Resources

1. database (TechTarget)
2. relational database (TechTarget)
3. non-relational database (MongoDB)
4. operational database (odb) (Techopedia)
5. NoSQL Meets Bitcoin and Brings Down Two Exchanges: The Story of Flexcoin and Poloniex

1.2 Tables

If a Facebook user leaves a comment on a post then that comment is related to both the user's account and the post which the comment is on.

This relationship would be created by using a table for users, a table for posts and a table for comments. Each of these tables would have a column for an ID number, known as a "primary key" and the other tables would be able to reference this ID number as a "foreign key" to form a relationship.

id 🔑	user	post
1	5	6
2	7	5
3	4	8

2. Types of Relationships

2.1 One-to-One Relationships

> A one-to-one relationship in a relational database occurs when one parent record or field has either zero or one child record only. These relationships are the easiest to represent in databases because both the parent and child records may be in the same table.
>
> — From One-to-One Relationship, Techopedia

2.2 One-to-Many/Many-to-One Relationships

> In relational databases, a one-to-many relationship occurs when a parent record in one table can potentially reference several child records in another table. In a one-to-many relationship, the parent is not required to have child records; therefore, the one-to-many relationship allows zero child records, a single child record or multiple child records. The important thing is that the child cannot have more than one parent record.
>
> — From One-to-Many Relationship, Techopedia

2.3 Many-to-Many Relationships

> A many-to-many relationship refers to a relationship between tables in a database when a parent row in one table contains several child rows in the second table, and vice versa. Many-to-many relationships are often tricky to represent.
>
> — From Many-to-Many Relationship, Techopedia

3. Object Relational Mapping

> Object-relational mapping (ORM) is a mechanism that makes it possible to address, access and manipulate objects without having to consider how those objects relate to their data sources. ORM lets programmers maintain a consistent view of objects over time, even as the sources that deliver them, the sinks that receive them and the applications that access them change.
>
> — From object-relational mapping (ORM), TechTarget

Laravel Resources

1. Laravel Website
2. Laravel Documentation
3. Laracasts
4. Laravel News
5. Scotch.io
6. Stack Overflow

4. Eloquent ORM

4.1 About the Eloquent ORM

One of Laravel's most compelling features is its Eloquent ORM (Object-Relational Mapper). Laravel's Eloquent ORM simplifies the process of creating relational database tables. It does so using models. With the exception of many-to-many relationships, each model generally maps to a single database table. Then in model files, one-to-one, one-to-many or many-to-many relationships can be formed between models using methods.

PHP Resources

1. PHP Website
2. XAMPP
3. Documentation
4. W3Schools
5. CSS Tricks
6. Scotch.io
7. Stack Overflow

4.2 One-to-One Relationships

The simplest type of relationship is a one-to-one relationship. We will create a one-to-one relationship between an `Artist` and their `Bio` because each artist would only need one biography. To do so, we will use the `App\Models\Bio` namespace, create a method inside of our `Artist` class and name it `bio()`. This method will return the `hasOne()` method which will take the `Bio` class as a parameter.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use App\Models\Bio;

class Artist extends Model {
   protected $table = 'artists';

   protected $fillable = ['name', 'image', 'styles'];

   /**
    * Get the bio associated with the artist.
    */
   public function bio() {
        return $this->hasOne(Bio::class);
   }
}
```

We will now need to actually create the `Bio` model. To do so, enter the following artisan command.

```
php artisan make:model Bio
```

In this model, we will need to use the `App\Models\Artist` namespace to create an inverse relationship with the `Artist` model by creating a method called `artist()` which returns the

belongsTo() method which will take the Artist class as a parameter.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use App\Models\Artist;

class Bio extends Model {
   protected $table = 'bios';

   protected $fillable = ['title', 'text'];

   /**
    * Get the artist that owns the bio.
    */
   public function artist() {
        return $this->belongsTo(Artist::class);
   }
}
```

In order to have this relationship reflected in the database we also need to create a database migration for bios using the following artisan command.

```
php artisan make:migration create_bios_table
```

We will then create a table for bios which must include a bigInteger() field with a name of 'artist_id' and which uses the unsigned() method, which disallows negative integers and the nullable() method which allows the field to be null. This will act as a foreign key which references the primary key of the artist table.

```
class CreateBiosTable extends Migration {
   public function up() {
        Schema::create('bios', function (Blueprint $table) {
             $table->id();
             $table->string('title', 255);
             $table->text('text');
             $table->bigInteger('artist_id')->unsigned()->nullable();
             $table->timestamps();
        });
   }

   public function down() {
        Schema::dropIfExists('bios');
   }
}
```

4.2 One-to-Many Relationships

One of the most common types of relationships is a one-to-many relationship. Each Artist will have many Artworks so we will use a one-to-many relationship. To do so, we will use the App\Models\

Artwork namespace, create a method inside of our Artist class and name it artworks(). This method will return the hasMany() method which will take the Artwork class as a parameter.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use App\Models\Bio;
use App\Models\Artwork;

class Artist extends Model {
   // Existing Artist class code...

   /**
     * Get the artworks for the artist.
     */
   public function artworks() {
        return $this->hasMany(Artwork::class);
   }
}
```

We will now need to actually create the Artwork model. To do so, enter the following artisan command.

```
php artisan make:model Artwork
```

In this model, we will need to use the App\Models\Artist namespace to create an inverse relationship with the Artist model by creating a method called artist() which returns the belongsTo() method which will take the Artist class as a parameter.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use App\Models\Artist;

class Artwork extends Model {
   protected $table = 'artworks';

   protected $fillable = ['title', 'text'];

   /**
     * Get the artist that owns the bio.
     */
   public function artist() {
        return $this->belongsTo(Artist::class);
   }
}
```

In order to have this relationship reflected in the database we also need to create a database migration for artworks using the following artisan command.

```
php artisan make:migration create_artworks_table
```

We will then create a table for bios which must include a `bigInteger()` field with a name of `'artist_id'` and which uses the `unsigned()` method, which disallows negative integers and the `nullable()` method which allows the field to be null. This will act as a foreign key which references the primary key of the artist table.

```
class CreateArtworksTable extends Migration {
    public function up() {
        Schema::create('artworks', function(Blueprint $table) {
            $table->id();
            $table->string('title', 255);
            $table->string('image', 255);
            $table->text('statement')->nullable();
            $table->bigInteger('artist_id')->unsigned()->nullable();
            $table->timestamps();
        });
    }

    public function down() {
        Schema::dropIfExists('artworks');
    }
}
```

4.2 Many-to-Many Relationships

The most complex type of relationship is a many-to-many relationship. Unlike one-to-one and one-to-many relationships, many-to-many relationships require three database tables: one for each model and one to store the foreign keys which reference the primary keys of each model. We will use a many-to-many relationship to create galleries which will display the artworks. Each gallery could have many artworks in it and each artwork could be displayed in many galleries.

To do so, we will use the `App\Models\Gallery` namespace, create a method inside of our `Artwork` class and name it `galleries()`. This method will return the `belongsToMany()` method which will take the `Gallery` class as a parameter. For a many-to-many relationship it is also safest to include parameters which reperesent the name of the table which uses the foreign keys and the names of the fields which hold the foreign key for each model. By default Laravel will assume that this table will be named using the names of both models in alphabetical order, separated by an underscore.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use App\Models\Artist;
use App\Models\Gallery;

class Artwork extends Model {
    // Existing Artwork class code...

    /**
```

```
     * The galleries that belong to the artwork.
     */
    public function galleries() {
        return $this->belongsToMany(Gallery::class, 'artwork_gallery',
            'artwork_id', 'gallery_id'););
    }
}
```

We will now need to actually create the `Gallery` model. To do so, enter the following artisan command.

```
php artisan make:model Gallery
```

In this model, we will need to use the `App\Models\Artwork` namespace to create an inverse relationship with the `Artwork` model by creating a method called `artworks()` which returns the `belongsToMany()` method which will take the `Artwork` class as a parameter.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use App\Models\Artwork;

class Gallery extends Model {
    protected $table = 'galleries';

    protected $fillable = ['title'];

    /**
     * Get the artworks for the artist.
     */
    public function artworks() {
        return $this->belongsToMany(Artwork::class, 'artwork_gallery',
            'gallery_id', 'artwork_id');
    }
}
```

In order to have this relationship reflected in the database we also need to create a database migration for artworks using the following artisan command.

```
php artisan make:migration create_galleries_table
```

This migration won't actually need a foreign key. It will jsut store the data for a gallery.

```
class CreateGalleriesTable extends Migration {
    public function up() {
        Schema::create('galleries', function(Blueprint $table) {
            $table->id();
            $table->string('title', 255);
            $table->timestamps();
        });
    }
```

```
    public function down() {
        Schema::dropIfExists('galleries');
    }
}
```

Instead, because this is a many-to-many relationship, we need to create a separate migration to generate the foreign keys so that they are in a separate table. To create this table, use the following artisan command.

```
php artisan make:migration create_artwork_gallery_table
```

This migration will have a field for a foreign key called `'artwork_id'` and a field for a foreign key called `'gallery_id'` which will represent the primary keys of each table. Because these keys have a dedicated table there can be multiple rows representing different pairings of keys which allows the many-to-many relationship to work.

```
class CreateArtworkGalleryTable extends Migration {
    public function up() {
        Schema::create('artwork_gallery', function(Blueprint $table) {
            $table->id();
            $table->bigInteger('artwork_id')->unsigned()->nullable();
            $table->bigInteger('gallery_id')->unsigned()->nullable();
            $table->timestamps();
        });
    }

    public function down() {
        Schema::dropIfExists('artwork_gallery');
    }
}
```

To run all of our migrations, use the following artisan command.

```
php artisan migrate
```

5. Resource Routes

We have a lot of models now so we will need a lot of corresponding routes and controllers. When using Laravel's "resource controllers" which already have named functions for performing CRUD operations, you can also use a `resources()` route method which allows you to quickly define many routes at once by automatically creating routes which map to the CRUD operations used in Laravel's resource controllers.

```
Route::resources([
    '/' => ' App\Http\Controllers\ArtistController',
    'artists' => 'App\Http\Controllers\ArtistController',
    'bios' => 'App\Http\Controllers\BioController',
    'artworks' => 'App\Http\Controllers\ArtworkController',
    'galleries' => 'App\Http\Controllers\GalleryController',
]);
```

6. Relational Resource Controllers

6.1 Generating Controllers

To have artisan generate resource controllers for our new models, run the following commands.

```
php artisan make:controller BioController --resource
php artisan make:controller ArtworkController --resource
php artisan make:controller GalleryController --resource
```

6.2 Artist Controller

6.2.1 Artist Controller Namespaces

It is necessary to include the following namespaces in the controller file, including the class of the model.

```
namespace App\Http\Controllers;

use Illuminate\Http\ArtistRequest;

use App\Models\Artist;
use App\Models\Artwork;

use URL;

class ArtistController extends Controller {
```

6.2.2 Index Method

It is necessary to use the `with()` method and to specify the name of a table before the `get()` method to pass a related model to the view as a property of the main model.

```
public function index() {
   $artists = Artists::with('artworks')->get();

   return view('view-artists', [
      'title' => 'Artists',
      'artists' => $artists
    ]);
}
```

6.2.5 Show Method

We will add a `show()` method to our `ArtistController` so that we can display a single artist.

```
public function show($id) {
   $artists = Artists::with('artworks')->find($id);

   return view('show-artist', [
```

```
            'artist' => $artist,
            'title' => $artist->name,
        ]);
    }
```

6.2.6 Edit Method

```
    public function edit($id) {
        $artists = Artists::with('artworks')->find($id);

        return view('edit-artist', [
            'artist' => $artist,
            'title' => 'Edit ' . $artist->name,
        ]);
    }
```

6.3 Bio Controller

6.3.1 Bio Controller Namespaces

It is necessary to include the following namespaces in the controller file, including the class of the model.

```
    namespace App\Http\Controllers;

    use Illuminate\Http\Request;

    use App\Models\Bio;
    use App\Models\Artist;

    use URL;

    class BioController extends Controller {
```

6.3.2 Create Method

We need to pass the artists to the view so that we can loop through them in the Add Bio form and allow the user to select an artist to associate the bio with.

```
    public function create() {
        $artists = Artists::get();

        return view('add-bio', [
            'title' => 'Add Bio',
            'artists' => $artists
        ]);
    }
```

6.3.3 Store Method

When creating a new bio we need to set the `artist_id` property so that the bio is associated with a

specific artist.

```
public function store(Request $request) {
   $bio = new Bio;
   $bio->title = $request->title;
   $bio->text = $request->text;
   $bio->artist_id = $request->artist_id;

   $bio->save();

   return redirect('/artists/' . $bio->artist->id)->with(
      'success',
      'New bio added for ' . $bio->artist->name . '!'
   ]);
}
```

6.3.4 Edit Method

```
public function edit($id) {
   $bio = Bio::find($id);
   $artists = Artists::get();

   return view('edit-artist', [
        'bio' => $bio,
        'title' => 'Edit ' . $bio->title,
        'artists' => $artists
   ]);
}
```

6.3.5 Update Method

```
public function update(ArtistRequest $request, $id) {
   $bio = Bio::find($id);
   $bio->title = $request->title;
   $bio->text = $request->text;
   $bio->artist_id = $request->artist_id;

   $bio->save();

   return redirect('/artists/' . $bio->artist->id)->with(
      'success',
      'Bio updated for ' . $bio->artist->name . '!'
   ]);
}
```

6.4 Artwork Controller

6.4.1 Artist Controller Namespaces

It is necessary to include the following namespaces in the controller file, including the class of the

model.

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\Artwork;
use App\Models\Artist;
use App\Models\Gallery;

use URL;

class ArtistController extends Controller {
```

6.4.2 Index Method

We can use multiple parameters in the `with()` method if our model has multiple relationships.

```
public function index() {
   $artworks = Artworks::with('artist', 'galleries')->get();

   return view('view-artworks', [
      'title' => 'Artworks',
      'artworks' => $artworks
   ]);
}
```

6.4.3 Create Method

```
public function create() {
   $artists = Artists::get()
   $galleries = Galleries::get()

   return view('add-artworks', [
      'title' => 'Add Artwork',
      'artists' => $artists,
      'galleries' => $galleries
   ]);
}
```

6.4.4 Store Method

```
public function store(Request $request) {
   $artwork = new Artwork;
   $artwork->title = $request->title;
   $artwork->statement = $request->statement;

   if ($request->hasFile('image')) {
        $image = $request->file('image');

        $image_name = strtolower(str_replace(' ', '_',
```

```
                $request->title));
            $image_time = time();
            $image_extension = '.' . $image->getClientOriginalExtension();

            $full_image_name = $image_title . '_' . $image_time .
                $image_extension;
            $image_path = date('Y/m/d');
            $destination_path = public_path('/images/' . $image_path);
            $image->move($destination_path, $full_image_name);

            $artwork->image = $image_path . '/' . $full_image_name;
        }

        $artwork->save();
```

With a many-to-many relationship, it is necessary to use thte following code to sync all of the chosen related galleries with the artwork in the `artwork_gallery` table.

```
        $galleries = $request->get('galleries');
        $artwork->galleries()->sync($galleries);

        return redirect('/artworks')->with(
            'success',
            'New artwork, "' . $artwork->title . '" added successfully!'
        ]);
    }
```

6.4.5 Show Method

```
    public function show($id) {
        $artists = Artists::with('artworks')->find($id);

        return view('show-artist', [
            'artist' => $artist,
            'title' => $artist->name,
        ]);
    }
```

6.4.6 Edit Method

```
    public function edit($id) {
        $artists = Artists::with('artworks')->find($id);

        return view('edit-artist', [
            'artist' => $artist,
            'title' => 'Edit ' . $artist->name,
        ]);
    }
```

6.4.7 Update Method

```
public function update(Request $request, $id) {
   $artwork = Artwork::find($id);
   $artwork->title = $request->title;
   $artwork->statement = $request->statement;

   if ($request->hasFile('image')) {
        $image = $request->file('image');

        $image_name = strtolower(str_replace(' ', '_',
             $request->title));
        $image_time = time();
        $image_extension = '.' . $image->getClientOriginalExtension();

        $full_image_name = $image_title . '_' . $image_time .
             $image_extension;
        $image_path = date('Y/m/d');
        $destination_path = public_path('/images/' . $image_path);
        $image->move($destination_path, $full_image_name);

        $artwork->image = $image_path . '/' . $full_image_name;
   } else {
        $artwork->image = $request->old_image;
   }

   $artwork->save();

   $galleries = $request->get('galleries');
   $artwork->galleries()->sync($galleries);

   return redirect('/artworks')->with(
       'success',
       'The artwork, "' . $artwork->title . '" updated successfully!'
    ]);
}
```

6.5 Gallery Controller

6.5.1 Gallery Controller Namespaces

It is necessary to include the following namespaces in the controller file, including the class of the model.

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Artwork;
use App\Gallery;
```

```
use URL;

class ArtistController extends Controller {
```

6.4.2 Index Method

```
public function index() {
    $galleries = Gallery::with('artworks')->get();

    return view('view-galleries', [
        'title' => 'Galleries',
        'galleries' => $galleries
    ]);
}
```

6.4.3 Create Method

```
public function create() {
    $artworks = Artwork::get()

    return view('add-gallery', [
        'title' => 'Add Galleryr',
        'artworks' => $artworks
    ]);
}
```

6.4.4 Store Method

```
public function store(Request $request) {
    $gallery = new Gallery;
    $gallery->title = $request->title;

    $gallery->save();
```

With a many-to-many relationship, it is necessary to use the following code to sync all of the chosen related artworks with the gallery in the `artwork_gallery` table.

```
    $artworks = $request->get('artworks');
    $gallery->artworks()->sync($artworks);

    return redirect('/galleries')->with(
        'success',
        'New gallery, "' . $gallery->title . '" added successfully!'
    ]);
}
```

6.4.5 Show Method

```
public function show($id) {
    $artists = Artists::with('artworks')->find($id);

    return view('show-artist', [
```

```
            'artist' => $artist,
            'title' => $artist->name,
        ]);
    }
```

6.4.6 Edit Method

```
    public function edit($id) {
        $artists = Artists::with('artworks')->find($id);

        return view('edit-artist', [
            'artist' => $artist,
            'title' => 'Edit ' . $artist->name,
        ]);
    }
```

6.4.7 Update Method

```
    public function update(Request $request, $id) {
        $gallery = Gallery::find($id);
        $gallery->title = $request->title;

        $gallery->save();

        $artworks = $request->get('artworks');
        $gallery->artworks()->sync($artworks);

        return redirect('/galleries')->with(
            'success',
            'New gallery, "' . $gallery->title . '" updated successfully!'
        ]);
    }
```

7. Relational Blade Templates

7.1 Show Artist

```
        <div class="col-3">
            <h2>{{ $artist->name }}</h2>
            <p><img src="{{ url('/img/' . $artist->image) }}"
                alt="{{ $artist->name }}"></p>
            <p>Styles: {{ $artist->styles }}</p>

            @if ($artist->bio)
            <h3>{{ $artist->bio->title }}</h3>
            <p>{{ $artist->bio->text }}</p>

            <a href="{{ url('/' . $artist->bio->id . '/edit') }}"
                class="btn btn-dark"><span class="fa fa-edit">
```

```
                </span>Edit Bio</a>
            @else
            <a href="{{ url('/bios/create') }}" class="btn btn-dark"><span
                class="fa fa-plus"></span>Add Bio</a>
            @endif
        </div>
        <div class="col-7">
            <h3>Artworks by {{ $artist->name }}</h3>
            @foreach ($artist->artworks as $artwork)
            <p><img src="{{ url('/img/' . $artwork->image) }}"
                alt="{{ $artwork->title }}"></p>
            <p>{{ $artwork->statement }}</p>
            @endforeach
        </div>
        <div class="col-2">
            <h3>Manage {{ $artist->name }}</h3>
            {!! Form::open(['action' =>
                ['App\Http\Controllers\ArtistController@destroy',
                $artist->id], 'method' => 'delete']) !!}

                <a href="{{ url('/' . $artist->id . '/edit') }}"
                    class="btn btn-dark"><span class="fa fa-edit">
                    </span>Edit</a>

                {!! Form::button(['<span class="fa fa-trash"></span>
                    Delete', ['type' => 'submit', 'class' => 'btn
                    btn-dark']) !!}
            {!! Form::close() !!}
        </div>
```

7.2 Add Bio

```
    {!! Form::open(['action' =>
        ['App\Http\Controllers\BioController@store']]) !!}
        <div class="form-group">
            {!! Form::label(['title', 'Title:') !!}
            {!! Form::text(['title', null, ['class' => 'form-control'])
                !!}
        </div>
        <div class="form-group">
            {!! Form::label(['text', 'Bio text:') !!}
            {!! Form::textarea(['text', null, ['class' => 'form-control',
                'rows' => 5]) !!}
        </div>
        <div class="form-group">
            {!! Form::label(['artist_id', 'Artist:') !!}
            <select name="artist_id" id="artist_id" class="form-control">
                @foreach ($artists as $artist)
                <option value="{{ $artist->id }}">{{ $artist->name }}
```

```
                    </option>
                @endforeach
            </select>
    </div>

    {!! Form::button(['<span class="fa fa-plus"></span> Add Bio',
    ['type' => 'submit', 'class' => 'btn btn-dark']) !!}
{!! Form::close() !!}
```

7.3 Edit Bio

```
{!! Form::model($bio, ['action' =>
    ['App\Http\Controllers\BioController@update', $bio->id],
    'method' => 'put']) !!}
    <div class="form-group">
        {!! Form::label(['title', 'Title:') !!}
        {!! Form::text(['title', null, ['class' => 'form-control'])
            !!}
    </div>
    <div class="form-group">
        {!! Form::label(['text', 'Bio text:') !!}
        {!! Form::textarea(['text', null, ['class' => 'form-control',
            'rows' => 5]) !!}
    </div>
    <div class="form-group">
        {!! Form::label(['artist_id', 'Artist:') !!}
        <select name="artist_id" id="artist_id" class="form-control">
            @foreach ($artists as $artist)
            <option value="{{ $artist->id }}" @if ($artist->id ==
                $bio->artist->id) selected @endif>{{ $artist->name
                }}</option>
            @endforeach
        </select>
    </div>

    {!! Form::button(['<span class="fa fa-edit"></span> Edit Bio',
    ['type' => 'submit', 'class' => 'btn btn-dark']) !!}
{!! Form::close() !!}
```

7.4 Add Artwork

```
{!! Form::open([['action' =>
    ['App\Http\Controllers\ArtworkController@store'],
    'files' => true]) !!}
    <div class="form-group">
        {!! Form::label(['title', 'Title:') !!}
        {!! Form::text(['title', null, ['class' => 'form-control'])
            !!}
```

```
        {!! Form::label(['image', 'Upload Image:') !!}
        {!! Form::file(['image', ['class' => 'btn btn-dark']) !!}
    </div>
    <div class="form-group">
        {!! Form::label(['statement', 'Statement:') !!}
        {!! Form::textarea(['statement', null, ['class' =>
            'form-control', 'rows' => 5]) !!}
    </div>
    <div class="form-group">
        {!! Form::label(['artist_id', 'Artist:') !!}
        <select name="artist_id" id="artist_id" class="form-control">
            @foreach ($artists as $artist)
            <option value="{{ $artist->id }}">{{ $artist->name }}
                </option>
            @endforeach
        </select>
    </div>

    {!! Form::button(['<span class="fa fa-plus"></span> Add Artwork',
        ['type' => 'submit', 'class' => 'btn btn-dark']) !!}
{!! Form::close() !!}
```

7.5 Edit Artwork

```
{!! Form::model($artwork, [['action' =>
    ['App\Http\Controllers\ArtworkController@update',
    $artwork->id], 'method' => 'put', 'files' => true]) !!}
    <div class="form-group">
        {!! Form::label(['title', 'Title:') !!}
        {!! Form::text(['title', null, ['class' => 'form-control'])
            !!}

        {!! Form::label(['image', 'Upload Image:') !!}
        {!! Form::file(['image', ['class' => 'btn btn-dark']) !!}
        {!! Form::hidden(['old_image', $artwork->image) !!}
    </div>
    <div class="form-group">
        {!! Form::label(['statement', 'Statement:') !!}
        {!! Form::textarea(['statement', null, ['class' =>
            'form-control', 'rows' => 5]) !!}
    </div>
    <div class="form-group">
        {!! Form::label(['artist_id', 'Artist:') !!}
        <select name="artist_id" id="artist_id" class="form-control">
            @foreach ($artists as $artist)
            <option value="{{ $artist->id }}" @if ($artist->id ==
                $artwork->artist->id) selected @endif>{{
                $artist->name }}</option>
            @endforeach
```

```
        </select>
    </div>

    {!! Form::button(['<span class="fa fa-plus"></span> Add Artwork',
        ['type' => 'submit', 'class' => 'btn btn-dark']) !!}
{!! Form::close() !!}
```

7.6 Add Gallery

```
{!! Form::open([['action' =>
    ['App\Http\Controllers\GalleryController@store']]) !!}
    <div class="form-group">
        {!! Form::label(['title', 'Title:') !!}
        {!! Form::text(['title', null, ['class' => 'form-control'])
            !!}
    </div>
    <div class="form-group">
        {!! Form::label(['artworks[]', 'Artworks:') !!}
        <select name=”artworks[]” id=”artworks” class=”form-control”>
            @foreach ($artworks as $artwork)
            <option value=”{{ $artwork->id }}”>{{ $artwork->name }}
                </option>
            @endforeach
        </select>
    </div>

    {!! Form::button(['<span class="fa fa-plus"></span> Add Gallery',
        ['type' => 'submit', 'class' => 'btn btn-dark']) !!}
{!! Form::close() !!}
```

7.7 Edit Gallery

```
{!! Form::model($gallery,
    [['action' => ['App\Http\Controllers\GalleryController@update',
    $gallery->id], 'method' => 'put']) !!}
    <div class="form-group">
        {!! Form::label(['title', 'Title:') !!}
        {!! Form::text(['title', null, ['class' => 'form-control'])
            !!}
    </div>
    <div class="form-group">
        {!! Form::label(['artworks[]', 'Artworks:') !!}
        <select name=”'artworks[]” id=”artist_id”
            class=”form-control”>
        @foreach ($artworks as $artwork)
            <option value=”{{ $artist->id }}"
            @foreach ($gallery->artworks as $gallery_artwork)
                @if ($artist->id == $gallery_artwork->id)
                    selected
```

```
                    @endif
                @endforeach
                >{{ $artist->name }}</option>
            @endforeach
            </select>
    </div>

    {!! Form::button(['<span class="fa fa-plus"></span> Add Gallery',
            ['type' => 'submit', 'class' => 'btn btn-dark']) !!}
{!! Form::close() !!}
```

7.8 View Galleries

```
@foreach ($galleries as $gallery)
    <div class="col-3">
            <h2>{{ $gallery->title }}</h2>

            @foreach ($galleries->artworks as $artwork)
            <p><img src="{{ url('/img/' . $artwork->image) }}"
                alt="{{ $artwork->title }}"></p>
            <h3>{{ $artwork->title }}</h3>
            <h4>By {{ $artwork->artist->name }}</h4>
            <p>{{ $artwork->statement }}</p>
            @endforeach

            {!! Form::open(['action' =>
                ['App\Http\Controllers\GalleryController@destroy',
                $artist->id], 'method' => 'delete']) !!}

                <a href="{{ url('/' . $artist->id . '/edit') }}"
                    class="btn btn-dark"><span class="fa fa-edit">
                    </span>Edit</a>

                {!! Form::button(['<span class="fa fa-trash"></span>
                    Delete', ['type' => 'submit', 'class' => 'btn
                    btn-dark']) !!}
            {!! Form::close() !!}
    </div>
    @endforeach
```