WEB 306 – PHP: Databases and Framework

Day 9 – Intro to Laravel

1. About Laravel

Laravel is a free and open source MVC framework for creating large-scale CRUD applications with PHP. It was created by Taylor Otwell in 2011 in an effort to create a more robust alternative to CodeIgniter, another PHP framework, which lacked several key features of Laravel. Some of these specifically related to security such as authentication and authorization. The name has no true meaning but was inspired by the name of a castle, Cair Paravel, from the Narnia series of novels.

Laravel itself uses another PHP framework known as Symfony as a basis. Symfony is a more lightweight collection of PHP libraries which was in turn inspired by the Spring framework for Java.

The first release of Laravel included models, views and a routing system but did not include controllers which prevented it from being considered a fully MVC-compliant framework initially. In Laravel 2 which was realeased in 2011, controllers were introduced which made it a complete MVC framework. This release also introduced an improved HTML templating engine called Blade. Laravel 3 introduced a CLI (command-line interface) called Artisan as well as support for database versioning. Since Laravel 4 was released in 2013, Laravel has been distributed through a collection of separate Composer packages which allows it to be more lightweight, modular and extendable.

Since 2015 Laravel has been considered the most popular MVC framework for PHP.

Laravel Resources

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

2. Virtual Hosts

2.1 About Virtual Hosts

In order to work with many large web development frameworks such as Laravel locally, it is sometimes necessary for you to set up what is known as a "virtual host" on your computer. A virtual host lets you store a project anywhere on your computer and lets you create an alias for it which looks like a web address i.e. `project.local`, `coolsite.test`. This means that if you have a project with a long address you do not need to enter it to view your site. In my `C:/` drive, I have set

up a folder called Sandbox which contains ALL of my web development projects and uses a virtual host called `sandbox.local`.

2.2 Setting Up a Virtual Host with XAMPP

Creating a virtual host requires editing core files on your computer so you must have administrator priviledges when you open these files. There are two files which you need to edit to set up a virtual host using XAMPP. The first one is stored in `C:/xampp/apache/conf/extra` and is called `http-vhosts.conf`. You can open this file in VS Code or Atom like any other text file. Even though you already have access to the address `localhost` for viewing local files you MUST define it as one of your hosts using the following code before you define any new hosts.

```
  <VirtualHost *:80>
    DocumentRoot "C:/xampp/htdocs"
    ServerName localhost
  </VirtualHost>
```

The `DocumentRoot` setting points to the path your project is stored in on your computer. The `ServerName` setting is the address which you want to be able to access your project from in your browser. It is important that this address does not end in a real TLD (top-level domain name) like `.com` or `.net`.

Developers used to use the TLD `.dev` but this address has been registered as a real TLD by Google and they have begun to sell `.dev` domains so it is not recommended to use this as a fake TLD anymore. Instead it is recommended that you use either `.local` or `.test` which has been excluded from being eligible for being a real TLD.

The `ServerAlias` setting is the same address prefaced with `www.` like a web address. The path to your project must also be included in the opening `Directory` tag.

To add a new host, make sure that you have all of the following settings in your `httpd-vhosts.conf` file.

```
  <VirtualHost artists.local:80>
      DocumentRoot "C:/Sandbox/web-306-day-10-intro-laravel-class/
          artists/public"
      ServerName artists.local
      ServerAlias www.artists.local
      <Directory "C:/Sandbox/web-306-day-10-intro-laravel-class/
          artists/public">
        Options All
        Require all granted
        Options Indexes FollowSymLinks Includes execCGI
        AllowOverride All
        Order Allow,Deny
        Allow From All
      </Directory>
  </VirtualHost>
```

The second file which you must edit is stored in `C:\Windows\System32\drivers\etc` and is simply called `hosts`. This is a Windows system file so to edit this file you must open Notepad ***with administrator priviledges*** and then go to **File > Open...**, navigate to `C:\Windows\System32\`

drivers\etc and open the file called **hosts**. It is a hidden file so you will need to select "All files (*.*)" from the dropdown in the bottom right. This file lists IP addresses and their associated virtual hosts on your computer. To add a new host enter a line which reflects the format of the following.

```
::1             artists.local
```

After completing this process you must restart Apache using the XAMPP control panel.

2.3 Setting Up a Virtual Host with MAMP

The ideal way to set up a virtual host with MAMP is to use MAMP Pro which automates the process for you. However, if you are using the free version of MAMP then you can create virtual hosts in almost exactly the same way as you would with XAMPP.

There are three files which you need to edit to set up a virtual host using MAMP. The first one is stored in **Applications/MAMP/conf/apache** and is called **httpd.conf**. You can open this file in VS Code or Atom like any other text file. Scroll to the bottom of this file and find the following lines:

```
# Virtual Hosts
# Include /Applications/MAMP/conf/apache/extra/httpd-vhosts.conf
```

Uncomment the bottom line:

```
# Virtual Hosts
Include /Applications/MAMP/conf/apache/extra/httpd-vhosts.conf
```

The second file is stored in **Applications/MAMP/conf/apache/extra** and is called **http-vhosts.conf**. You can open this file in Atom like any other text file. Even though you already have access to the address **localhost** for viewing local files you MUST define it as one of your hosts using the following code before you define any new hosts.

```
<VirtualHost *:80>
  DocumentRoot /Applications/MAMP/htdocs
  ServerName localhost
</VirtualHost>
```

To add a new host, make sure that you have all of the following settings in your **httpd-vhosts.conf** file.

```
<VirtualHost artists.local:80>
    DocumentRoot /Users/username/Sandbox/web-306-day-10-intro-
        laravel-class/artists/public
    ServerName artists.local
    ServerAlias www.artists.local
    <Directory /Users/username/Sandbox/web-306-day-10-intro-
        laravel-class/artists/public>
      Options All
      Require all granted
      Options Indexes FollowSymLinks Includes execCGI
      AllowOverride All
      Order Allow,Deny
      Allow From All
    </Directory>
```

```
</VirtualHost>
```

The third file which you must edit is stored in `Macintosh HD/private/etc` and is simply called `hosts`. This is a Mac system file so to edit this file you must open TextWrangler ***while logged in as a Mac administrator*** and then go to **File > Open.** It is a hidden file so you will need to select **Everything** from the **Enable** dropdown menu at the bottom of the **Open** dialog box. You will also need to ensure that the **"Show hidden items"** checkbox is checked.

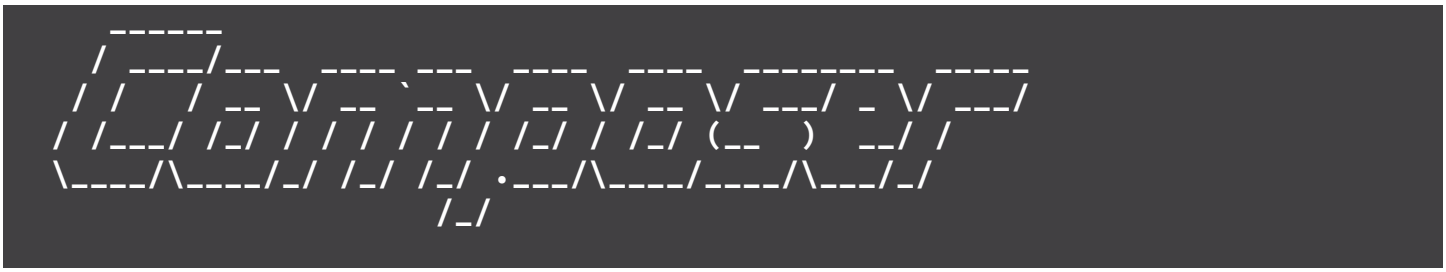
Navigate to `Macintosh HD/private/etc` and open the file called `hosts`. This file lists IP addresses and their associated virtual hosts on your computer. To add a new host enter a line which reflects the format of the following.

```
::1             artists.local
```

When you try to edit this file you will be asked if you're sure you want to unlock "hosts". Click **Unlock**. After completing this process you must restart Apache using the MAMP control panel.

3. Installing Composer

Laravel requires Composer. To download and install Composer on Windows, go to the Composer website, download the installation wizard, go through the steps and be sure to add Composer to the `$PATH` variable of your computer. To check if Composer installed successfully, open Git Bash, or Terminal on a Mac, and simply enter the command `composer`. If Composer is installed then you should see this:



Whenever you enter the command `composer` with the command `require` to install a package all of the files will be downloaded into a folder in the root directory of you project called `vendor`. A JSON file called `composer.json` will also be created in your project's root directory which will list all of the required packages for your project.

To download and install Composer on a Mac you will need to use the Terminal and reference the instructions on the website.

4. Installing Node.js and NPM

4.1 About Node.js

Node.js is an open-source, cross-platform environment which allows us to use JavaScript, which is a ***front-end/client-side*** language, as a ***back-end/server-side*** language, like PHP. It also allows developers to use JavaScript to write native computer programs and even to program robots.

4.2 The History of Node.js

Node.js was originally created by Ryan Dahl in 2009. At the time it only supported Linux and MacOS operating systems but it has since been adapted to work on Windows.

While Node.js was created over a decade after the first server-side environment for JavaScript it grew to become the most popular and successful server-side environment for JavaScript.

In 2015 the non-profit Node.js Foundation was formed and since then it has been responsible for managing the development and maintenance of Node.js. Node.js is now used by companies such as GoDaddy, IBM, LinkedIn, Paypal, Walmart and Yahoo!.

Node.js Resources

1. Node.js Website
2. MDN
3. W3Schools
4. CSS Tricks
5. Scotch.io
6. Stack Overflow

4.3 Installing Node.js

To install Node.js go to the Node.js website and install the version which is recommended for most users.
On Windows, we can use the `start` command in the terminal to open a web address.

```
$ start https://nodejs.org
```

Go through the installation wizard and once you have finished, open Git Bash and enter the following command to make sure it installed correctly and to check the version number.

```
$ node -v
```

4.4 About NPM

NPM stands for Node Package Manager. It is used to download packages for JavaScript, Node.js and TypeScript as well as CSS and Sass libraries.

There are two primary parts to NPM: The CLI and the package registry. The CLI is comprised of the built-in NPM terminal commands which are used to perform operations such as installing, updating and removing NPM packages. The package registry is the hosting service where NPM packages are stored.

NPM is the default package manager which is installed with Node.js. There are alternatives to the NPM CLI such as the Yarn, but even when you are using the Yarn CLI to download Node.js packages, it is still downloading them using the NPM package registry. Until recently there were no viable competing package registries for JavaScript and Node.js packages but this is rapidly changing.

4.5 The History of NPM

NPM was originally developed by Isaac Z. Schlueter in 2010 who was dissatisfied with the available options for package management at the time. He set out to create a package manager specifically for Node.js which offered a superior experience to package manages such as Pear, a PHP package manager.

Unlike Ryan Dahl who gave up ownership of Node.js, Isaac Schlueter initially maintained ownership of NPM and served as its CEO until January 2019. Schlueter still serves as the Chief Product Officer. Contrary to popular belief, NPM is not managed by a non-profit like Node.js is and is a for-profit corporation. NPM collects data about the packages you download and sells that data to third-parties. NPM was recently purchased by Microsoft.

4.6 Using NPM

NPM should be installed when you install Node.js.

```
$ start https://www.npmjs.com/
```

To check if you have NPM installed and the version number enter the following command into Git Bash.

```
$ npm -v
```

When using NPM with a project, it creates a JSON file called **package.json** in the project's root directory, which lists all of the required packages for the project. At a minimum, this JSON file would include the following properties:

```
{
  “name”: “intro-npm”,
  “version”: “1.0.0”,
  “main”: “index.js”
}
```

You can create a **package.json** file manually or have NPM guide you through the process using this command:

```
$ npm init
```

This command will ask you to answer a series of questions which will help it to create the JSON file. For the most part, if you don't know the answers to any of the questions that NPM init asks then it's okay to just leave them blank to use the default. This is what the JSON file that this command creates would look like.

```
{
  “name”: “intro-npm”,
  “version”: “1.0.0”,
  “description”: “”,
  “main”: “index.js”,
  “scripts”: {
       “test”: “echo \”Error: no test specified\” && exit 1”
  },
  “author”: “”,
  “license”: “ISC”
}
```

5. Installing Laravel

5.1 Server Requirements

In order to install the latest version of Laravel (5.7), your server must meet the following requirements:

1. **PHP >= 7.3**
2. **BCMath PHP Extension**
3. **Ctype PHP Extension**
4. **Fileinfo PHP Extension**
5. **JSON PHP Extension**
6. **Mbstring PHP Extension**
7. **OpenSSL PHP Extension**
8. **PDO PHP Extension**
9. **Tokenizer PHP Extension**
10. **XML PHP Extension**

The latest version of XAMPP already meets these requirements but if you needed to check to see if your server meets these requirements then you could use the `phpinfo()` function in a PHP file to check your server and PHP settings:

PHP Resources

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

```
<?php
phpinfo();
```

If you need to enable one of these PHP extensions then you could most likely do so in your server's `php.ini` file.

5.2 Installing with the Laravel Installer

The easiest way to install Laravel is to install the Laravel installer which will allow you to run a simpler command every time you want to download and install a new Laravel project. To install the Laravel installer, use the following Composer command in Git Bash:

```
$ composer global require laravel/installer
```

In order for this command to work it is necessary for Composer to be added to your computer's `$PATH` variable. If Composer is not added to your `$PATH` variable then you can add it by reinstalling Composer.

If you are using Linux or MacOS you may need to use the `sudo` command to grant administrator priviledges and you will need to enter your password.

```
$ sudo composer global require laravel/installer
```

Once you have the Laravel installer installed you can simply run the command `laravel new`, followed by the name of your project.

```
$ laravel new helloworld
```

5.3 Installing with Composer

Alternatively, if you are unable to install the Laravel installer then you can use Composer to install Laravel for each project using the following command.

```
$ composer create-project --prefer-dist laravel/laravel helloworld
```

5.4 Artisan Local Development Server

If you were not able to set up a virtual host using XAMPP, Laravel also provides a built-in development server which can be started using Artisan, their CLI with the following command.

```
$ php artisan serve
```

If you are running your server using the CLI I would recommend having a dedicated terminal window open for running the server and a seperate terminal for running other commands.

6. Hello World App

6.1 Folders and Files

We will start by creating a simple "Hello World" application which doesn't rely on a database connection. Once you create a new Laravel project you will have the following folders and files in the root directory.

```
/helloworld
   /app
   /bootstrap
   /config
   /database
   /public
   /resources
   /routes
   /storage
   /tests
   /vendor
   .env
   composer.json
   package.json
   server.php
```

These folders serve the following purposes:

1. `/app` **- This folder holds files which are used to build the back-end of our app such as both our Models and our Controllers.**
2. `/bootstrap` **- This folder does not hold the front-end CSS/SASS/JS library known as**

Bootstrap. The term "bootstrap" simply refers to files which are used to get an app started. The files in this folder are used to create the foundation of the application. You shouldn't need to edit them regularly.

3. `/database` - This folder contains files which are used to manage databases by creating and updating tables and columns but this is not where you would connect to the database.
4. `/public` - This folder holds the files which would be publicly viewable on your web host. The folder contains an `index.php` file and a `.htaccess` file which are used to direct traffic to all other pages in the site. When creating a virtual host you would point the host to this folder.
5. `/resources` - This folder contains our view files such as our Blade HTML templates and pre-compiled front-end assets such as SCSS files, JS files and images.
6. `/routes` - This folder contains files which specify which URLs are associated with your Laravel site and either which Blade templates should be returned or which methods should be called when they are visited.
7. `/storage` - This folder contains compiled regular PHP files which have been genereted from Blade templates.
8. `/tests` - This folder contains files which are used for tests.
9. `/vendor` - The folder contains Composer packages which Laravel requires such as Symfony, Dotenv and the majority of the Laravel framework files.
10. `.env` - This is where you define environment variables such as database connection credentials. They are then imported into the app using Dotenv.
11. `composer.json` - This JSON file lists all of the required Composer packages which must be installed for Laravel to run properly.
12. `package.json` - This JSON file lists all of the required NPM packages which are required for Laravel's front-end assets such as SASS, JavaScript, the Bootstrap CSS/SASS/JS framework and the VueJS JavaScript framework to work.
13. `server.php` - This file is used to run the Artisan local development server.

PHPMyAdmin Resources

1. PHPMyAdmin Website
2. XAMPP
3. PHPMyAdmin Documentation
4. PHPMyAdmin Support
5. PHPMyAdmin Tutorials
6. Stack Overflow

6.2 Installing and Updating Required Packages

Generally when starting a project you would want to ensure that all of the dependencies are installed and up to date. To update all of the Composer dependencies run the following command.

```
$ cd helloworld
$ composer update
```

To install the required Node modules run the following command.

```
$ npm install
```

6.3 Routes

The first thing we need to do to create a simple app is to create a route which will define an address the user can go to to load a page. Average web URLs are defined in the **web.php** file which is inside the **/routes** folder. Routes are created using the Route class and accessing static methods from it. The **get()** method uses the GET HTTP request method. The first parameter it takes is the URL it is associated with and the second parameter is either a function which returns a Blade template using the **view()** function or the name of a controller and associated method to call. The **view()** function takes two parameters. The first is the name of a Blade template. The second is an associative array of variables to pass to the template.

```
Route::get('/helloworld', function () {
    return view('helloworld', ['title' => 'Hello World!']);
});
```

6.4 Blade Template

Now that we have route which loads a Blade template, the next step is to create a very simple Blade template. Blade templates are saved the the **/resources/views** folder. We will save this as **helloworld.blade.php**.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
            initial-scale=1.0">
        <title>{{ $title }}</title>
    </head>
    <body>
        <h1>{{ $title }}</h1>
    </body>
</html>
```

To verify test the application, either view the site using your virtual host or run the following command to use the Artisan local development server.

```
$ php artisan serve
```

7. Artists App

Laravel would normally be used to create larger scale CRUD applications so we will create a more complex app now. It will be a database of artists. To do so, we need to repeat the process we just followed to create a virtual host, create a new Laravel project using the Laravel installer and updating and installing the required Composer and NPM packages.

```
$ laravel new artists
```

```
$ cd artists
$ composer update
$ npm install
```

We will also need to create a new database called `artists` using PHPMyAdmin. It is not necessary for us to create a table as Laravel will be used to create database tables for us automatically.

MySQL Resources

1. MySQL Website
2. MySQL Documentation
3. XAMPP
4. W3Schools
5. Stack Overflow

8. Dotenv

In order to connect our Laravel app to our database we need to alter the database credentials in the `.env` file. This is an environment file where we can store sensitive data like usernames and passwords without putting them in the code so that we can work on database projects collaboratively and share our code publicly in places like GitHub without displaying our credentials. When using XAMPP, our database host is localhost and unless we create a new database account in PHPMyAdmin, then our username is "root" and our password is blank by default. Your `.env` file should reflect the following.

```
APP_NAME=Artists
APP_ENV=local
APP_KEY=base64:YH2cy+JOyOZ+kUHQJlbWj68NoKtuSlOwe/xmErT9Zeg=
APP_DEBUG=true
APP_URL=http://artists.test

LOG_CHANNEL=stack
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=artists
DB_USERNAME=root
DB_PASSWORD=
```

9. Migrations

9.1 Generating Migrations

Now that we are connected to a database we can create our database migration. A "migration" is a file which is generated to create and update database tables based on changes which have been made to the model's class. To have Artisan generate a migration file enter the following command.

```
php artisan make:migration create_artists_table
```

9.2 Editing Migrations

Once you run this command a new migration file is generated and placed in the `database/migrations` folder. You would then open this file and edit it to specify which columns you need in the database table. The migration is defined using a class. This class contains two methods: the `up()` method and the `down()` method. The `up()` method is used to add new tables and columns to the database while the `down()` method is used to reverse any actions performed by the `up()` method.

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

class CreateArtistsTable extends Migration {
    public function up() {
        Schema::create('artists', function (Blueprint $table) {
            $table->id(); // Alias of $table->bigIncrements('id');
            $table->string('name', 255);
            $table->string('image', 255);
            $table->text('styles');
            $table->integer('series')->nullable();
            $table->integer('works')->nullable();
            $table->timestamps();
        });
    }

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

The `bigIncrements()` method should be used to define the primary key. This is because it is an auto incrementing `BIGINT` type which does not have a limit which means that our app will not break if many entries are added to the database. You can also use the `id()` method as a shorthand for this.

The `string()` method is used to define `VARCHAR` type columns and should have a limit set for it. The `text()` method is used to define `TEXT` type columns and does not need a limit. The `integer()` method is obviously used for integers. The `nullable()` method is used to indicate that a column can be `NULL` and should be used for any fields which are not required. Laravel also requires the inclusion of the `timestamps()` method which tracks when a table was created and when it was last updated.

MariaDB Resources

1. MariaDB Website
2. MariaDB Documentation
3. XAMPP
4. Stack Overflow

SQL Resources

1. XAMPP
2. W3Schools
3. Scotch.io
4. Stack Overflow

9.3 Migrating Migrations

Once you have finished editing a migration you would then activate it to create a database table ot column. You would do this by entering the following Artisan command.

```
php artisan migrate
```

If you decide that you want to undo a migration then you can use the rollback command to run the `down()` method.

```
php artisan migrate:rollback
```

To rollback all of your migrations use the following command.

```
php artisan migrate:reset
```

To rollback and migrate again in one command use the following command.

```
php artisan migrate:refresh
```

To drop all tables and run all migrations for a fresh start use the following command.

```
php artisan migrate:fresh
```

10. Models

Now that we have created a database table using a migration we can progress to creating a model. To have Artisan generate a model file enter the following command.

```
php artisan make:model Artist
```

You can also have Artisan generate both a model and a database migration file based on that model at the same time using the following command.

```
php artisan make:model Artist --migration
```

Unless you are working with relational data, models in Laravel are generally fairly simple. They would have a protected property called `$table` to define the database table which is associated with the model and an array called `$fillable` which would define which columns of the database table can be filled out by the user.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

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

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

11. Routes

Next we need to create some routes. We won't use our routes to return a view this time though. Instead they will call a method inside of our controller. Routes that store data in the database use the `post()` method to indicate that they are using the **POST** HTTP request method. Routes that update data use the `put()` method to replicate the **PUT** HTTP request method. Routes that delete data use the `delete()` method to replicate the **DELETE** HTTP request method. Routes that are used for editing, updating or deleting data need to pass the ID to the controller in the URL.

```
use Illuminate\Support\Facades\Route;

Route::get('/', 'App\Http\Controllers\ArtistsController@index');

Route::get('/add', 'App\Http\Controllers\ArtistsController@create');
Route::post('/store', 'App\Http\Controllers\ArtistsController@store');

Route::get('/{id}/edit/', 'App\Http\Controllers\ArtistsController@
edit');
Route::put('/{id}/update/', 'App\Http\Controllers\ArtistsController@
update');

Route::delete('/{id}/destroy/', 'App\Http\Controllers\
ArtistsController@destroy');
```

12. Controllers

12.1 Generating Controllers

To have Artisan generate a controller file run the following command.

```
php artisan make:controller ArtistsController
```

To have Artisan generate a "resource controller" which already has named functions for performing CRUD operations run the following command.

```
php artisan make:controller ArtistsController --resource
```

12.2 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\Artist;

use URL;

class ArtistsController extends Controller {
```

12.3 Index Method

The index method uses a static get method of the model to select all artist from the database. Then it returns a view and passes variables for the title and the artists to it.

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

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

12.4 Create Method

The create method lists an array of image files, returns a view file and passes variables for the title and the image files to it.

```
public function create() {
   $images = [
        'Da Vinci' => 'da-vinci.jpg',
        'Walt Disney' => 'disney.jpg',
        'Frida Kahlo' => 'kahlo.jpg',
        'Picasso' => 'picasso.jpg'
   ];

   return view('add-artist', [
      'title' => 'Add Artist',
      'images' => $images
   ]);
}
```

12.5 Store Method

The store method validates the form using the `validate()` method of the request object, creates a new instance of the `Artist` class and then sets each property of the artist object to be equal to the corresponding property of the request object which was collected from the form. The artist object is then saved and stored in the database using the model's `save()` method. Finally, the user is redirected back to the home page with a success message.

```
public function store(Request $request) {
   $request->validate([
        'name' => 'required|max:255',
        'image' => 'required|max:255',
        'styles' => 'required'
   ]);

   $artist = new Artist;
   $artist->name = $request->name;
```

```
    $artist->image = $request->image;
    $artist->styles = $request->styles;

    $artist->save();

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

12.6 Edit Method

The edit method is very similar to the create method except that it takes in an ID from the route, selects artists from the database if their ID matches the ID and then selects the first one. The artist is then passed to the view so that its properties can be used in the form to display their currently existing data.

```
  public function edit($id) {
    $artist = Artists::get('id', $id)->first();

    $images = [
         'Da Vinci' => 'da-vinci.jpg',
         'Walt Disney' => 'disney.jpg',
         'Frida Kahlo' => 'kahlo.jpg',
         'Picasso' => 'picasso.jpg'
    ];

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

12.7 Update Method

The update method is very similar to the store method exceot that it selects a specific artist and reassigns its properties insttead of creating a new instance of the `Artist` class.

```
  public function update(Request $request, $id) {
    $request->validate([
         'name' => 'required|max:255',
         'image' => 'required|max:255',
         'styles' => 'required'
    ]);

    $artist = Artists::get('id', $id)->first();
    $artist->name = $request->name;
```

```
    $artist->image = $request->image;
    $artist->styles = $request->styles;

    $artist->save();

    return redirect(URL::previous())->with(
        'success',
        'The artist, "' . $artist->name . '" was updated successfully!'
    ]);
  }
```

12.8 Destroy Method

The destroy method is very simple. It just selects an artist with an ID that matches the one which was passed to the controller in the route. Then it uses the `delete()` method to delete it and redirects back to the previous page using the `URL` class and the `previous()` method.

```
  public function destroy($id) {
    $artist = Artists::get('id', $id)->first();
    $artist->delete();

    return redirect(URL::previous())->with(
        'success',
        'The artist, "' . $artist->name . '" was deleted successfully!'
    ]);
  }
```