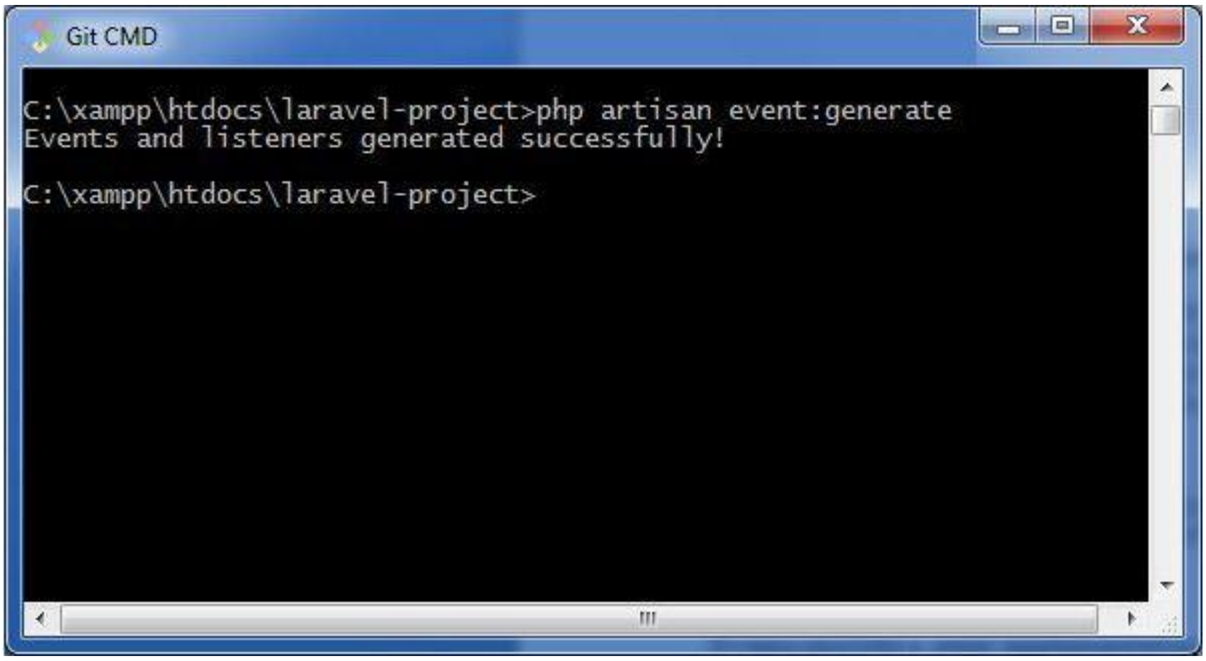
Laravel - Event Handling

Events provide a simple observer implementation which allows a user to subscribe and listen to various events triggered in the web application. All the event classes in Laravel are stored in the **app/Events** folder and the listeners are stored in the **app/Listeners** folder.

The artisan command for generating events and listeners in your web application is shown below –

php artisan event:generate

This command generates the events and listeners to the respective folders as discussed above.



```
C:\xampp\htdocs\laravel-project>php artisan event:generate
Events and listeners generated successfully!

C:\xampp\htdocs\laravel-project>
```

Events and Listeners serve a great way to decouple a web application, since one event can have multiple listeners which are independent of each other. The events folder created by the artisan command includes the following two files: event.php and SomeEvent.php. They are shown here –

Event.php

```
<?php

namespace App\Events;

abstract class Event{

  //

}
```

As mentioned above, **event.php** includes the basic definition of class **Event** and calls for namespace **App\Events**. Please note that the user defined or custom events are created in this file.

SomeEvent.php

```
<?php


namespace App\Events;


use App\Events\Event;

use Illuminate\Queue\SerializesModels;

use Illuminate\Contracts\Broadcasting\ShouldBroadcast;


class SomeEvent extends Event{

  use SerializesModels;

  /**

    * Create a new event instance.

    *

    * @return void

  */


  public function __construct() {

    //

  }


  /**

    * Get the channels the event should be broadcast on.

    *

    * @return array

  */
  public function broadcastOn() {

    return [];

  }

}
```

Observe that this file uses serialization for broadcasting events in a web application and that the necessary parameters are also initialized in this file.

For example, if we need to initialize order variable in the constructor for registering an event, we can do it in the following way –

```
public function __construct(Order $order) {

  $this->order = $order;

}
```

Listeners

Listeners handle all the activities mentioned in an event that is being registered. The artisan command **event:generate** creates all the **listeners** in the **app/listeners** directory. The Listeners folder includes a file **EventListener.php** which has all the methods required for handling listeners.

EventListener.php

```
<?php



namespace App\Listeners;



use App\Events\SomeEvent;

use Illuminate\Queue\InteractsWithQueue;

use Illuminate\Contracts\Queue\ShouldQueue;



class EventListener{

  /**
  * Create the event listener.

  *

  * @return void

 */


 public function __construct() {

  //

 }


 /**

  * Handle the event.

  *

  * @param SomeEvent $event

  * @return void

 */


 public function handle(SomeEvent $event) {

  //

 }

}
```

Laravel - Authentication

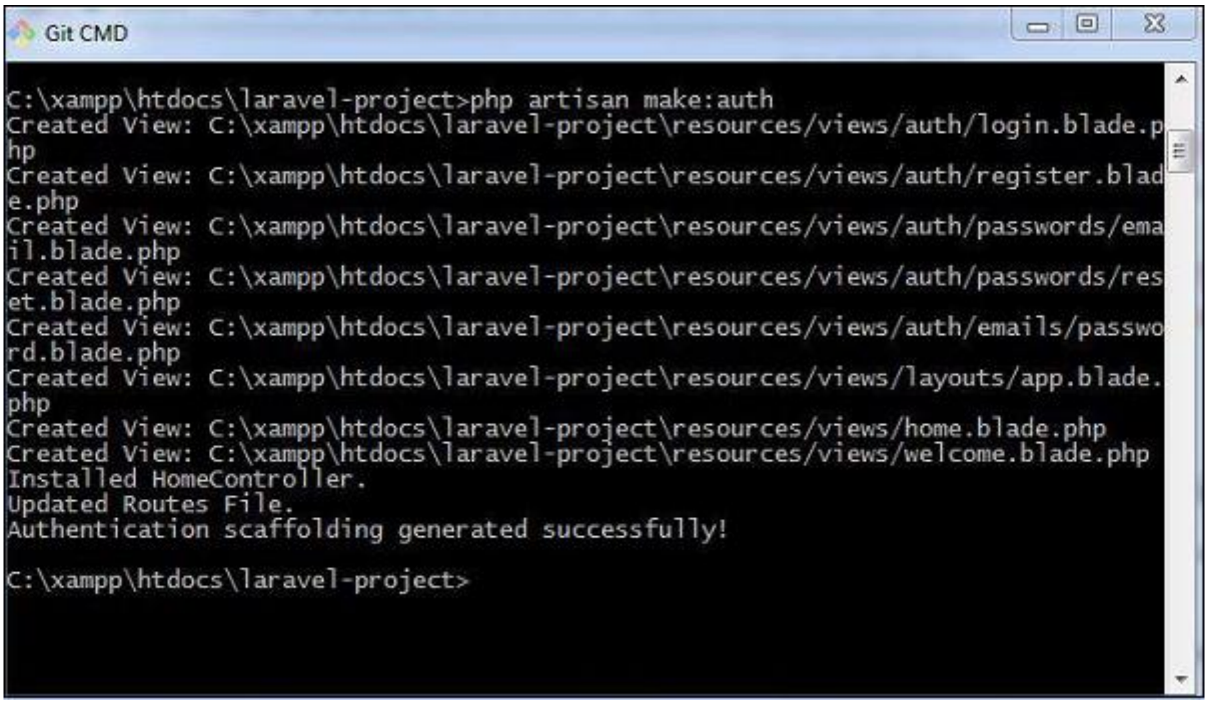
Authentication is the process of identifying the user credentials. In web applications, authentication is managed by sessions which take the input parameters such as email or username and password, for user identification. If these parameters match, the user is said to be authenticated.

Command

Laravel uses the following command to create forms and the associated controllers to perform authentication –

php artisan make:auth

This command helps in creating authentication scaffolding successfully, as shown in the following screenshot –



```
C:\xampp\htdocs\laravel-project>php artisan make:auth
Created View: C:\xampp\htdocs\laravel-project\resources/views/auth/login.blade.p
hp
Created View: C:\xampp\htdocs\laravel-project\resources/views/auth/register.blad
e.php
Created View: C:\xampp\htdocs\laravel-project\resources/views/auth/passwords/ema
il.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources/views/auth/passwords/res
et.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources/views/auth/emails/passwo
rd.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources/views/layouts/app.blade.
php
Created View: C:\xampp\htdocs\laravel-project\resources/views/home.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources/views/welcome.blade.php
Installed HomeController.
Updated Routes File.
Authentication scaffolding generated successfully!

C:\xampp\htdocs\laravel-project>
```

Controller

The controller which is used for the authentication process is **HomeController**.

```
<?php



namespace App\Http\Controllers;



use App\Http\Requests;

use Illuminate\Http\Request;



class HomeController extends Controller{

   /**

     * Create a new controller instance.
```

```
    *

    * @return void

  */


  public function __construct() {

    $this->middleware('auth');

  }


  /**

    * Show the application dashboard.

    *

    * @return \Illuminate\Http\Response

  */


  public function index() {

    return view('home');

  }

}
```

Laravel - Artisan Console

Laravel framework provides three primary tools for interaction through command-line namely: **Artisan, Ticker** and **REPL**. This chapter explains about Artisan in detail.

Introduction to Artisan

Artisan is the command line interface frequently used in Laravel and it includes a set of helpful commands for developing a web application.

Advertisement

Example

Here is a list of few commands in Artisan along with their respective functionalities –

To start Laravel project

php artisan serve

To enable caching mechanism

php artisan route:cache

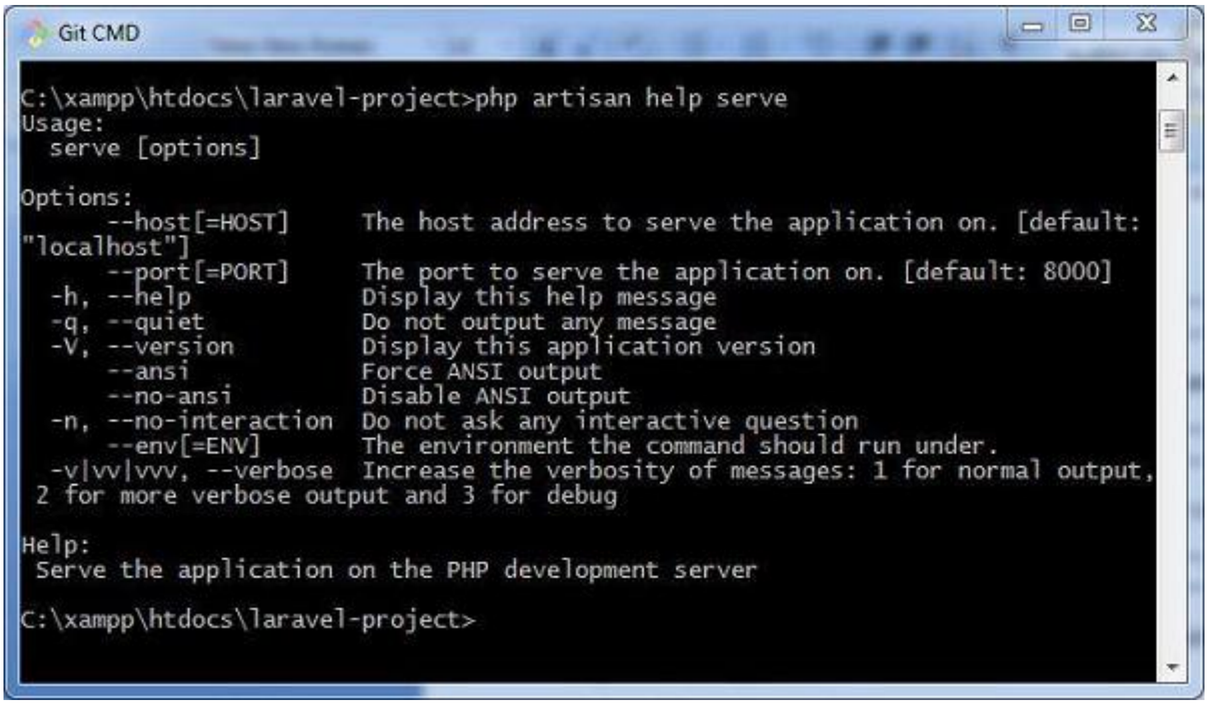
To view the list of available commands supported by Artisan

php artisan list

To view help about any command and view the available options and arguments

php artisan help serve

The following screenshot shows the output of the commands given above –



```
C:\xampp\htdocs\laravel-project>php artisan help serve
Usage:
  serve [options]

Options:
      --host[=HOST]     The host address to serve the application on. [default:
"localhost"]
      --port[=PORT]     The port to serve the application on. [default: 8000]
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
      --ansi            Force ANSI output
      --no-ansi         Disable ANSI output
  -n, --no-interaction  Do not ask any interactive question
      --env[=ENV]       The environment the command should run under.
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output,
 2 for more verbose output and 3 for debug

Help:
 Serve the application on the PHP development server

C:\xampp\htdocs\laravel-project>
```

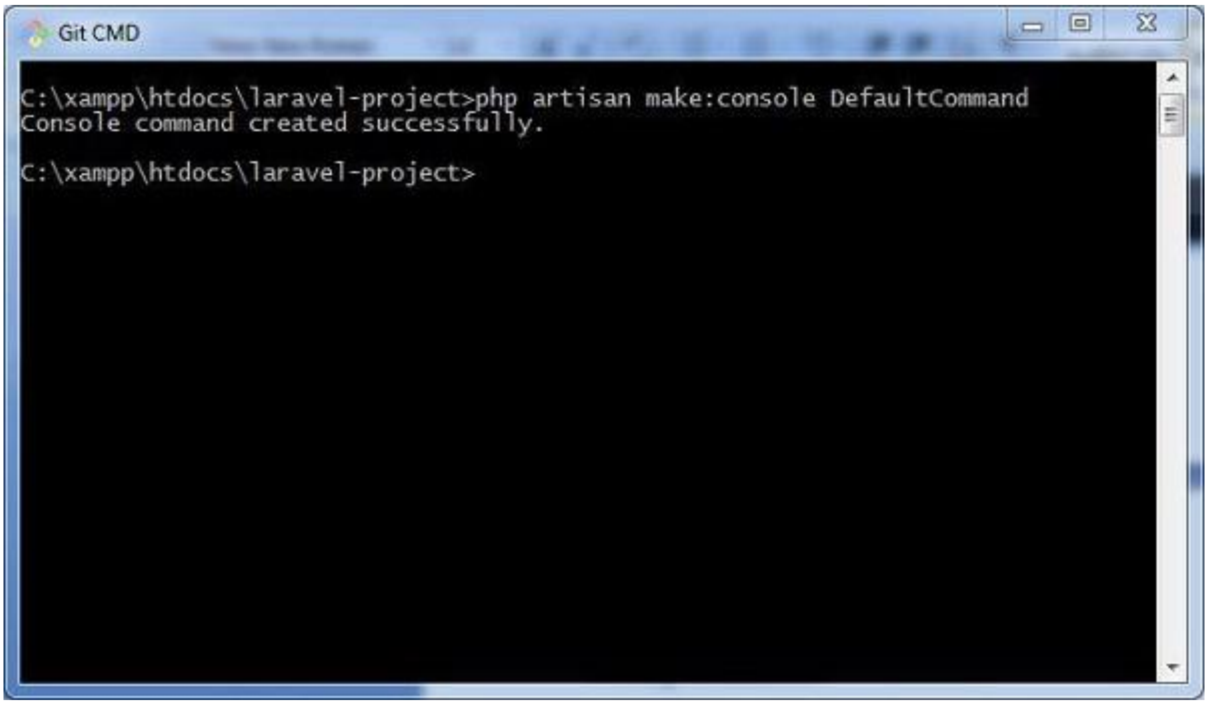
Writing Commands

In addition to the commands listed in Artisan, a user can also create a custom command which can be used in the web application. Please note that commands are stored in **app/console/commands directory**.

The default command for creating user defined command is shown below –

php artisan make:console <name-of-command>

Once you type the above given command, you can see the output as shown in the screenshot given below –



```
C:\xampp\htdocs\laravel-project>php artisan make:console DefaultCommand
Console command created successfully.

C:\xampp\htdocs\laravel-project>
```

The file created for **DefaultCommand** is named as **DefaultCommand.php** and is shown below –

```
<?php


namespace App\Console\Commands;

use Illuminate\Console\Command;


class DefaultCommand extends Command{

  /**

    * The name and signature of the console command.

    *

    * @var string

  */


  protected $signature = 'command:name';
```

```
/**

  * The console command description.

  *

  * @var string

*/


protected $description = 'Command description';


/**

  * Create a new command instance.

  *

  * @return void

*/


public function __construct() {

  parent::__construct();

}


/**

  * Execute the console command.

  *

  * @return mixed

*/


public function handle() {
```

//

}

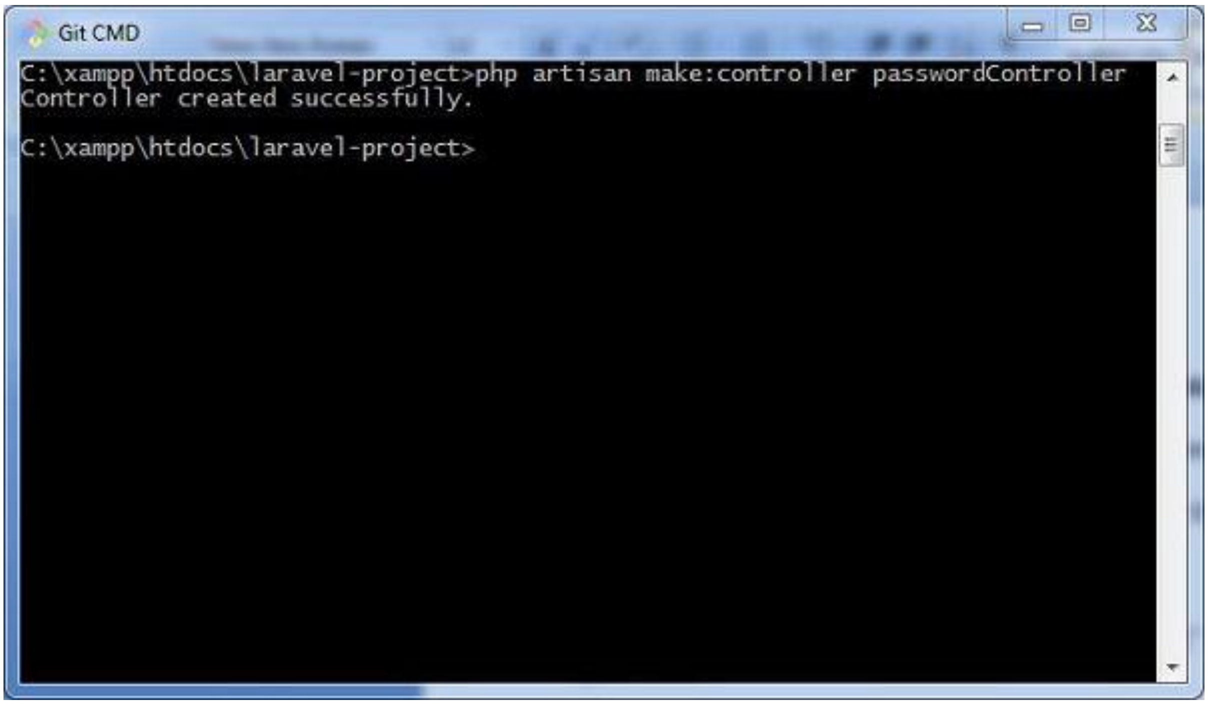
}

Laravel - Hashing

Hashing is the process of transforming a string of characters into a shorter fixed value or a key that represents the original string. Laravel uses the **Hash** facade which provides a secure way for storing passwords in a hashed manner.

Basic Usage

The following screenshot shows how to create a controller named **passwordController** which is used for storing and updating passwords –



```
C:\xampp\htdocs\laravel-project>php artisan make:controller passwordController
Controller created successfully.

C:\xampp\htdocs\laravel-project>
```

The following lines of code explain the functionality and usage of the **passwordController** –

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Hash;

use App\Http\Controllers\Controller

```
class passwordController extends Controller{
  /**
    * Updating the password for the user.
    *
    * @param Request $request
    * @return Response
  */

  public function update(Request $request) {
    // Validate the new password length...
    $request->user()->fill([
      'password' => Hash::make($request->newPassword) // Hashing passwords
    ])->save();
  }
}
```

The hashed passwords are stored using **make** method. This method allows managing the work factor of the **bcrypt** hashing algorithm, which is popularly used in Laravel.

Advertisement

Verification of Password against Hash

You should verify the password against hash to check the string which was used for conversion. For this you can use the **check** method. This is shown in the code given below −

```
if (Hash::check('plain-text', $hashedPassword)) {
  // The passwords match...
}
```