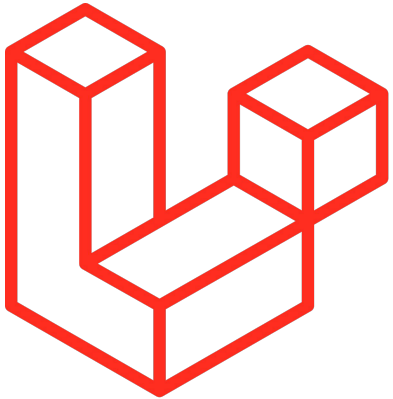

Laravel Certification Course

By Avinash Seth

Files

Uploading Files

- Laravel provides a powerful filesystem abstraction thanks to the wonderful Flysystem PHP package by Frank de Jonge. The Laravel Flysystem integration provides simple to use drivers for working with local filesystems and Amazon S3. Even better, it's amazingly simple to switch between these storage options as the API remains the same for each system.

Uploading Files Steps

- Open ***web.php*** file and add the following lines

```
Route::prefix('file')->group(function() {
    Route::view('upload', 'files.upload-file');
    Route::post('upload', 'FileController@postUploadFile')->name('post-upload-file');
});
```

Uploading Files Steps

- Open ***resources/views/files/upload-file.blade.php*** file and add the following lines

```
<form action="{{ route('post-upload-file') }}" method="post" enctype="multipart/form-data">
    @csrf
    <p>{{ Session::get('file_upload_feedback') }}</p>
    <p><input type="file" name="myfile" id="myfile" /></p>
    @error('myfile')
        <p>{{ $message }}</p>
    @enderror
    <p><button type="submit">Upload</button></p>
</form>
```

Uploading Files Steps

- Open ***FileController.php*** file and add the following lines

```
    function postUploadFile(Request $request) {
        $request->validate([
            'myfile' => 'required|mimes:pdf', // size:1024 in kilobyte
        ]);
        $fileName = time() . '.' . $request->myfile->extension();
        $request->myfile->move(public_path('uploads'), $fileName);
        $request->session()->flash('file_upload_feedback', 'File Uploaded Successfully');
        return redirect()->back();
    }
```

Uploading Files to AWS S3

- Open ***FileController.php*** file and add the following lines

```
function postUploadToAws(Request $request) {
    $request->validate([
        'myfile' => 'required|mimes:pdf', // size:1024 in kilobyte
    ]);
    $fileName = time() . '.' . $request->myfile->extension();
    $request->myfile->storeAs('files', $fileName, 's3');
    $request->session()->flash('file_upload_feedback', 'File Uploaded Successfully');
    return redirect()->back();
}
```

Install the package

- Run the following command
  - composer require league/flysystem-aws-s3-v3:"~1.0"

Edit .env file

- Update the .env file
  - AWS_ACCESS_KEY_ID=AKIAUBHRAZH2XOAZHDFX
  - AWS_SECRET_ACCESS_KEY=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
  - AWS_DEFAULT_REGION=ap-south-1
  - AWS_BUCKET=laravel-bucket-d
  - AWS_URL=s3.ap-south-1.amazonaws.com

Checking if file exists in S3

- Add the following code
  - ```
    Route::get('check', function() {
    ```
  - ```
        echo Storage::disk('s3')->exists('files/1648749001.pdf') ? 'Yes'
    : 'No';
    ```
  - ```
    });
    ```
    ```

Downloading file from S3

- Add the following code
  - ```
    Route::get('download', function() {
    ```
  - ```
        return
    ```
    ```
    Storage::disk('s3')->download('files/1648749001.pdf');
    ```
  - ```
    });
    ```
    ```

Getting file URL

- Add the following code
  - ```
    Route::get('file-url', function() {
        echo Storage::disk('s3')->url('files/1648749001.pdf');
    });
    ```
    ```

Temporary Access for a File

- Add the following code

```
Route::get('temp-access', function() {

    $url = Storage::disk('s3')->temporaryUrl(
        'files/1648749001.pdf', now()->addSeconds(20)
    );

    echo $url;

});
```

Prepending & Appending To Files

- Add the following code
  - `Storage::prepend('file.log', 'Prepended Text');`
  - `Storage::append('file.log', 'Appended Text');`

API in Laravel

Creating API

- Open ***routes/api.php*** file and add following lines

```
Route::get('user/{name}', function(Request $request) {

    return response()->json(
        [
            "greetings"=>"Hey {$request->name}, how are you doing today?"
        ]
    );

});
```

Posting data to API

- Open ***routes/api.php*** file and add following lines

```
Route::post('user/', function(Request $request) {


    return response()->json(
        [
            "greetings"=>"Hey {$request->name}, how are you doing today?"
        ]
    );

```

Ajax with API

- Open **web.php** file and add the following lines of code

```
Route::prefix('user')->group(function() {
    Route::view('/', 'user.hi');
});
```

- Open **api.php** file and add the following lines

```
Route::prefix('user')->group(function() {
    Route::post('verify-user', function() {
        return response()->json(['status'=>true,'message'=>'Unique Email']);
    });
});
```

Ajax with API

- Open **resources/views/user/hi.blade.php** file and add the following lines of code (link)

Queue

Queue in Laravel

- Laravel queues provide a unified API across a variety of different queue backends, such as Beanstalk, Amazon SQS, Redis, or even a relational database.

Steps

- Create a PaymentReminderEmail
  - php artisan make:mail PaymentDueReminderEmail
- Open PaymentDueReminderEmail.php file and add following code (**link**)
- Run following artisan command to add queue table in database
  - php artisan queue:table # creates a table for queued jobs
  - php artisan migrate # migrates the table
- Open .env file and add/edit the following line to
  - QUEUE_CONNECTION=database
- Create the job using following command
  - php artisan make:job PaymentReminderMailJob

Steps

- Add the following line in PaymentReminderMailJob.php file (link)
- Create a Controller PaymentReminderController
  - php artisan make:controller PaymentReminderController
- Add the following code in PaymentReminderController.php file (link)
- Add the following route in web.php file
  - ```
    Route::get('test-email', [PaymentReminderController::class, 'enqueue']);
    ```
- Hit the http://localhost:8000/test-email
- To execute the pending queue, run the following command
  - php artisan queue:work
    ```

Events

Events

- Laravel's events provide a simple observer pattern implementation, allowing you to subscribe and listen for various events that occur within your application.

Create Events in Laravel

- Add following code in EventServiceProvider.php file
  - ```
    protected $listen = [
            Registered::class => [
                SendEmailVerificationNotification::class,
            ],
            'App\Events\UserCommentedOnYouPhotoEvent' => [
                'App\Listeners\SaveEventDetailsToDBListener',
            ],
        ];
    ```
    ```

Create Events in Laravel

- Run the following command now
  - php artisan event:generate
- This will generate following files
  - app\Events\UserCommentedOnYouPhotoEvent.php
  - app\Listeners\SaveEventDetailsToDBListener.php
- Add following code in web.php
  - ```
    Route::get('photo/{username}/comment',
    [PhotoController::class,
    'getNotifyUserForNewComment'])->name('get-notify-user-for-comment');
    ```
- Note: Generate PhotoController via php artisan make:controller PhotoController
    ```

Create Events in Laravel

- Add the following code in PhotoController file
  - `use App\Events\UserCommentedOnYouPhotoEvent;`
  - `…`
  - `function getNotifyUserForNewComment(Request $request)`
  - `{`
  - `echo 'You commented photo of ' . $request->username;`
  - `event(new UserCommentedOnYouPhotoEvent($request));`
  - `}`
  -

Create Events in Laravel

- Update UserCommentedOnYouPhotoEvent file
  - `use Illuminate\Http\Request;`
  - `public function __construct(Request $request)`
  - `{`
  - `$this->request = $request;`
  - `}`
- Add a notificaiton table via migration
- Add a notification model

Create Events in Laravel

- Add following code in SaveEventDetailsToDBListener
  - `use App\Models\Notification;`
  - `public function handle(UserCommentedOnYouPhotoEvent $event)`
  - `{`
  - `$message = $event->request->username . ' commented on your photo';`
  - `$notification = new Notification;`
  - `$notification->user_id = rand(1000,9999);`
  - `$notification->notification_text = $message;`
  - `$notification->save();`
  - `}`

Create Events in Laravel

- Visit localhost:8000/photo/avinash/comment
- Check your database

Notifications

Notifications

- Laravel provides support for sending notifications across a variety of delivery channels, including email, SMS (via Vonage, formerly known as Nexmo), and Slack. In addition, a variety of community built notification channels have been created to send notification over dozens of different channels! Notifications may also be stored in a database so they may be displayed in your web interface.

Creating Notifications in Laravel

- Run the following commands
  - php artisan notifications:table
  - php artisan migrate
- Create your first Notification
  - php artisan make:notification RequestCallBackNotification
- Add following code in your RequestCallBackNotification.php file.

```
class RequestCallBackNotification extends Notification
{
    use Queueable;

    private $details; // this line
```

Creating Notifications in Laravel

- `public function __construct($details)`
- `    {`
- `        $this->details = $details;`
- `    }`
- Update delivery channel
  - `public function via($notifiable)`
  - `    {`
  - `        return ['mail', 'database'];`
  - `    }`

Creating Notifications in Laravel

- Update following toMail function

```
public function toMail($notifiable)
    {
        return (new MailMessage)
                    ->greeting($this->details['greeting'])
                    ->line($this->details['body'])
                    ->action($this->details['actionText'],
$this->details['actionURL'])
                    ->line($this->details['thanks']);
    }
```

Creating Notifications in Laravel

- Add following code in toDatabase function
  - ```
    public function toDatabase($notifiable)
    {
        return [
            'user_id' => $this->details['user_id'],
            'callback_date_time' =>
    $this->details['callback_date_time']
        ];
    }
    ```
    ```

Creating Notifications in Laravel

- Add following code in web.php file
  - `Route::get('request-callback', [CallBackController::class, 'requestCallBack']);`
- Run following code
  - php artisan make:controller CallBackController
- Add following code in CallBackController.php file
  - `use Carbon\Carbon;`
  - `use App\Models\User;`
  - `use App\Notifications\RequestCallBackNotification;`
  - `use Notification;`

Creating Notifications in Laravel

- Add following code in the function
  - ```
    function requestCallBack(Request $request)
    ```
  - ```
        {
        $user = User::first();
        $random = Carbon::today()->addDays(rand(1, 30));
        $details = [
    ```
  - ```
                'greeting' => 'Hi Avinash',
    ```
  - ```
                'body' => 'Rohan has requested a callback request @ ' . $random,
    ```
  - ```
                'thanks' => 'You can always check callback request from your profile page',
    ```
    ```

Creating Notifications in Laravel

- Visit
  - http://localhost:8000/request-callback

Sending Error Notification

- Add following code in RequestCallBackNotification.php file
  - ```
    public function toMail($notifiable)
    ```
  - ```
        {
    ```
  - ```
            return (new MailMessage)
    ```
  - ```
                    ->error()
    ```
  - ```
                    ->subject('Payment Failed')
    ```
  - ```
                    ->line('Hey, we couldn\'t process your payment with credit card ending **5656');
    ```
  - ```
        }
    ```
  -
    ```

Customizing Header

- Add following code in RequestCallBackNotification.php file
  - ```
    public function toMail($notifiable)
    ```
  - ```
        {
    ```
  - ```
            return (new MailMessage)
    ```
  - ```
                    ->error()
    ```
  - ```
                    ->from('payments@google.com', 'Google Payment')
    ```
  - ```
                    ->subject('Payment Failed')
    ```
  - ```
                    ->line('Hey, we couldn\'t process your payment with credit card ending **5656');
    ```
  - ```
        }
    ```

Customizing The Template

- Run following command
  - php artisan vendor:publish --tag=laravel-notifications

Checking notification from Database

- Add following code in web.php file
  - `Route::get('notifications', [CallBackController::class, 'getNotifications']);`
- Add following code in CallBackController.php file
  - `function getNotifications(Request $request) {`
  - `$user = \App\Models\User::find(1); // 1 is user id of User Model`
  - `foreach ($user->notifications as $notification) {`
  - `echo '<p>' . $notification->type . '</p>';`
  - `}`
  - `}`

Checking unread notifications

- Add following code in CallBackController.php file
  - ```
    function getNotifications(Request $request) {
    ```
  - ```
            $user = \App\Models\User::find(1); // 1 is user id of User Model
    ```
  - ```
            foreach ($user->unreadNotifications  as $notification) {
    ```
  - ```
                echo '<p>' . $notification->type . '</p>';
    ```
  - ```
            }
    ```
  - ```
        }
    ```
  -

Marking Notification as Read

- Add following code in CallBackController.php file
  - ```
    function getNotifications(Request $request) {
    ```
  - ```
            $user = \App\Models\User::find(1); // 1 is user id of User Model
    ```
  - ```
            foreach ($user->unreadNotifications as $notification) {
    ```
  - ```
                $notification->markAsRead();
    ```
  - ```
            }
    ```
  - ```
        }
    ```

Authorization

Authorization

- In addition to providing built-in authentication services, Laravel also provides a simple way to authorize user actions against a given resource. For example, even though a user is authenticated, they may not be authorized to update or delete certain Eloquent models or database records managed by your application. Laravel's authorization features provide an easy, organized way of managing these types of authorization checks.

Gates

- Gates are simply closures that determine if a user is authorized to perform a given action.

Gates

- Add following code in AuthServiceProvider
  - `use App\Models\Article;`
  - `use App\Models\User;`
- Add following code in boot function
  - ```
    Gate::define('update-article', function (User $user,
    Article $article) {
    ```
  - `return $user->id === $article->user_id;`
  - `});`
    ```

Gates

- Add following code in web.php file
  - ```
    Route::get('secret-page', function(Request $request,
    Article $article) {
    ```
  - ```

    ```
  - ```
        if (! Gate::allows('update-article', $article)) {
    ```
  - ```
            abort(403);
    ```
  - ```
        }
    ```
  - ```

    ```
  - ```
    });
    ```
    ```

Gates

- Visit localhost:8000/secret-page

Checking guard for another user

- Add the following code
  - ```
    Route::get('secret-page', function(Request $request,
    Article $article) {
    ```
  - ```

    ```
  - ```
        $user = User::where('id', 5)
    ```
  - ```
            ->first();
    ```
  - ```

    ```
  - ```
        if (Gate::forUser($user)->allows('update-article',
    $article)) {
    ```
  - ```
            // The user can update the article...
    ```
  - ```
        }
    ```

Gate Responses

- Add the following code in AuthServiceProvider
  - ```
    Gate::define('edit-settings', function (User $user) {
    ```
  - ```
                return $user->id === 1000
    ```
  - ```
                            ? Response::allow()
    ```
  - ```
                            : Response::deny('You must be an administrator.');
    ```
  - ```
            });
    ```
    ```

Gate Responses

- Add following code in web.php file
  - ```
    Route::get('secret-page', function(Request $request,
    Article $article) {
    ```
  - ```

    ```
  - ```
        $response = Gate::inspect('edit-settings');
    ```
  - ```

    ```
  - ```
        if ($response->allowed()) {
    ```
  - ```
            // The action is authorized...
    ```
  - ```
        } else {
    ```
  - ```
            echo $response->message();
    ```
  - ```
        }
    ```
    ```

Policies

- Policies are classes that organize authorization logic around a particular model or resource.

Create a policy

- Run this following command
  - php artisan make:policy StudentPolicy
- Add following code in AuthServiceProvider
  - `use App\Models\Student;`
  - `use App\Policies\StudentPolicy;`
- Add following code
  - `protected $policies = [`
  - `Student::class => StudentPolicy::class,`
  - `];`

Create a policy

- Add following code in ArticlePolicy
  - `use App\Models\User;`
  - `use App\Models\Student;`
- `public function update(User $user, Student $student)`
- `{`
- `return $user->id === $student->user_id;`
- `}`
- Add the following code in web.php file
  - `Gate::authorize('update', $student);`

Create a policy

- Add the following code in web.php file

```
Route::get('update-student', function() {
    $student = Student::where('id',4)->first();
    $user = Auth::user();
    if($user->can('update', $student))
    {
        echo 'update';
    }
    else
    {
        echo 'cannot update';
    }
});
```

Encryption

Encryption

- Laravel's encryption services provide a simple, convenient interface for encrypting and decrypting text via OpenSSL using AES-256 and AES-128 encryption. All of Laravel's encrypted values are signed using a message authentication code (MAC) so that their underlying value can not be modified or tampered with once encrypted.

Encrypting a string

- Add the following code in web.php file
  - ```
    use Illuminate\Http\Request;
    use Illuminate\Support\Facades\Crypt;

    Route::get('create-token/{token}', function(Request $request) {
        echo '<a href="/d/' . Crypt::encryptString($request->token) . '">Decrypt</a>';
    });
    ```
    ```

Encrypting a string

- Add the following code in web.php file

```
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Crypt;

Route::get('/d/{token}', function(Request $request) {
    try {
        $decrypted = Crypt::decryptString($request->token);
        echo $decrypted;
    } catch (DecryptException $e) {
        //
    }
```

Package Development

Packages in Laravel

- Packages are the primary way of adding functionality to Laravel. Packages might be anything from a great way to work with dates like Carbon or a package that allows you to associate files with Eloquent models like Spatie's Laravel Media Library.

Getting started

- Create a following folder structure
  - packages
    - avinash (vendor name)
      - seth (package name)
        - src

Getting started

- Inside **seth**(name of my package) folder, create a composer.json file and add the following code (link)
- Inside avinash/seth/src folder create a file with name Greet.php and add following code

```
namespace Avinash\Seth;
class Greet
{
    public function greet(String $name)
    {
        return 'Hello ' . $name . '! Welcome from Avinash\Seth Package';
    }
}
```

Getting started

- Inside composer.json file (root folder composer file) add the following code
  - ```
    "autoload": {
            "psr-4": {
                "App\\": "app/",
                …
                "Avinash\\Seth\\": "packages/avinash/seth/src"
            }
        },
    ```
- Run composer dump-autoload command
    ```

Getting started

- Add following code in web.php file

```
use Avinash\Seth\Greet;
Route::get('/greet/{name}', function ($name) {
    $greet = new Greet();
    return $greet->greet($name);
});
```

- Visit *http://localhost:8000/greet/avinash*

Task Scheduling

Task Scheduling

- In the past, you may have written a cron configuration entry for each task you needed to schedule on your server. However, this can quickly become a pain because your task schedule is no longer in source control and you must SSH into your server to view your existing cron entries or add additional entries.

Scheduling Artisan commands

- Run the following command
  - php artisan make:command Avinash --command=avinash:seth
- Add the following code in Avinash.php file
  - `public function handle()`
  - `{`
  - `\Log::info("Defusing nuclear bomb, one at a minute!");`
  - `}`

Scheduling Artisan commands

- Register a scheduler inside app/console/kernel.php file
  - `protected function schedule(Schedule $schedule)`
  - `    {`
  - `        $schedule->command('avinash:seth')->everyMinute();`
  - `        // check other frequencies`
  - `        // https://laravel.com/docs/8.x/scheduling#schedule-frequency-options`
  - `    }`

Scheduling Artisan commands

- Run the scheduler
  - php artisan schedule:run

Scheduling Queued Job

- Add following code in your handle function
  - ```
    protected function schedule(Schedule $schedule)
    ```
  - ```
        {
    ```
  - ```
            $schedule->job(new MyJob)->everyFiveMinutes();
    ```
  - ```
        }
    ```
  -

Scheduling Shell Commands

- Add following code in your handle function
  - ```
    protected function schedule(Schedule $schedule)
    ```
  - ```
        {
    ```
  - ```
            $schedule->exec('node /avinash/seth/purge.js')->daily();
    ```
  - ```
        }
    ```
  -
  -

Working with Timezone

- Add following code in your handle function
  - ```
    protected function schedule(Schedule $schedule)
        {
            $schedule->command('avinash:seth')
                ->timezone('America/New_York')
                ->at('13:45');
        }
    ```
    ```

Task without Overlapping

- Add following code in your handle function

```
protected function schedule(Schedule $schedule)
    {
        $schedule->command('avinash:seth')
            ->timezone('America/New_York')
            ->at('13:45')
            ->withoutOverlapping();
    }
```

Running command in maintenance mode

- Add following code in your handle function

```
protected function schedule(Schedule $schedule)
    {
        $schedule->command('avinash:seth')
            ->timezone('America/New_York')
            ->at('13:45')
            ->withoutOverlapping()
            ->evenInMaintenanceMode();
    }
```

Sending output

- Add following code in your handle function

```
protected function schedule(Schedule $schedule)
    {
        $schedule->command('avinash:seth')
            ->everyMinute()
            ->withoutOverlapping()
            ->evenInMaintenanceMode()
            ->sendOutputTo('file/to/path/output_log.txt');
    }
```

Sending email

- Add following code in your handle function

```
protected function schedule(Schedule $schedule)
    {
        $schedule->command('avinash:seth')
            ->everyMinute()
            ->withoutOverlapping()
            ->evenInMaintenanceMode()
            ->sendOutputTo('file/to/path/output_log.txt')
            ->emailOutputTo('avinash@example.com');
    }
```