LARAVEL API TUTORIAL

Chapter 1 (STARTER)

* Start the project
laravel new hdt_backend

* Create a model, controller and migration table for task
php artisan make:model Task -cm

* In The Migration File with the date == current date and name == task name, define the tables

```
<?php

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

return new class extends Migration
{
/**
* Run the migrations.
*/
public function up(): void
{
Schema::create('tasks', function (Blueprint $table) {
$table->id();
$table-> string("title");
$table -> boolean("is_done")->default(false);
$table->timestamps();
});
}

/**
* Reverse the migrations.
*/
public function down(): void
{
Schema::dropIfExists('tasks');
}
};
```

* Configure the Database in the .env file
DB_CONNECTION=sqlite
#DB_HOST=localhost

#DB_PORT=1433
#DB_DATABASE=HDSS_laravel
#DB_USERNAME=sa
#DB_PASSWORD=HydotTech

* Migrate the Database
php artisan migrate

* Create 2 Additional Resources
php artisan make:resource TaskCollection
php artisan make:resource TaskResource

The TaskCollection will not be edited because it will be used to group the data into keys
The TaskResource will be use to add additional data to the expected data base on a condition

* In the TaskController, add this code
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Task;
use App\Http\Resources\TaskCollection;

class TaskController extends Controller
{
public function index(Request $request){

return new TaskCollection(Task::all());

}
}

* In the api.php, define a route
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\TaskController;

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
return $request->user();
});

Route::get("getTask",[TaskController::class,'index']);

* In the Task Model, you can change the cast of some fields and also hide some of the data

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Task extends Model
{
use HasFactory;

protected $cast = [
"is_done" => "boolean",
];
protected $hidden = [
"updated_at"
];
}
```

Pattern
Database => migrations => TaskCollection => TaskController => Route => TaskModel => TaskResource => migrate => TestApi

Chapter 2 (CRUD)

* In the Task model, under the fillables modify it this way

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Task extends Model
{
use HasFactory;

protected $cast = [
"is_done" => "boolean",
```

```
];
protected $hidden = [
"updated_at"
];

protected $fillable = [
"title",
"is_done",
];

}
```

With this I can access the title and the is_done fields

* In the api.php define the route this way

```
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\TaskController;

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
return $request->user();
});



Route::apiResource("tasks",TaskController::class,);
```

The Api will just be 1 but each Http Method will link to a particular controller

* In the TaskController, add this code

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Task;
use App\Http\Resources\TaskCollection;
use App\Http\Resources\TaskResource;


class TaskController extends Controller
```

```
{

//Show all the Task
public function index(Request $request){

return new TaskCollection(Task::all());

}

//Show one task by the Id
public function show(Request $request, Task $task){
return new TaskResource($task);
}

//Save an object into the database (POST)
public function store(Request $request){
$validated = $request->validate([
"title" => "required|max:255",
]);

$task = Task::create($validated);

return new TaskResource($task);
}

//Put (Update base on their Id)
public function update(Request $request, Task $task){
$validated = $request->validate([
"title" => "sometimes|required|max:255",
"is_done" => "sometimes|required|max:255",
]);

$task -> update($validated);

return new TaskResource($task);
}

//Delete a resource
public function destroy(Request $request, Task $task){

$task -> delete();

return response()->noContent();
}
```

}

index => Get All
show => Get 1
store => Post request
Update => Put request,
Destroy => Delete request

Chapter 3(CRUD Manual Approach Part 1)

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Product;
class ProductController extends Controller
{
function addProducts(Request $req){
$product = new Product;
$product->product_name = $req->product_name;
$product->product_description = $req->product_description;
$product->quantity = $req->quantity;
$product->price = $req->price;
$product->seller_name = $req->seller_name;
$product->product_id = $this->IdGenerator(); // Assuming you have a method for generating product IDs

$existingProduct = Product::where('product_name', $product->product_name)->first();

if ($existingProduct) {
return response()->json(["error" => $product->product_name. " Already Exists"], 400);
} else {

$Counter = Product::count();

if($Counter>=5){
return response()->json(["error" => "You have more than 5 products"], 400);
}

$result = $product->save();

if($result){

```
return response()->json(["Result"=>"Success"], 200);
} else {
return response()->json(["Result"=>"Failed"], 500);
}
}
}

function getProducts(){


return Product::all();
}

function getOneProduct($PrId){
$a = Product::where('product_id', $PrId)->first();
if($a==null){
return Product::all();
}
else{
return $a;
}


}




function IdGenerator(): string {
$randomID = str_pad(mt_rand(1, 99999), 5, '0', STR_PAD_LEFT);
return $randomID;
}

}
```

Their Respective Routes

```
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\TaskController;
use App\Http\Controllers\ProductController;
Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
return $request->user();
});
```

```
Route::apiResource("tasks",TaskController::class,);
Route::post("add",[ProductController::class,'addProducts']);
Route::get("get",[ProductController::class,'getProducts']);
Route::get("getOne/{PrId}",[ProductController::class,'getOneProduct']);
```

Products Models

```
<?php

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

return new class extends Migration
{
/**
* Run the migrations.
*/
public function up(): void
{
Schema::create('products', function (Blueprint $table) {
$table->id();
$table -> string("product_name");
$table -> string("product_description");
$table -> integer("quantity");
$table -> float("price");
$table -> string("seller_name");
$table -> string("product_id");



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

/**
* Reverse the migrations.
*/
public function down(): void
{
Schema::dropIfExists('products');
}
};
```

Next [Update, Search, Delete, File Uploads, Authentication, Database Connection, Revision]

Chapter 3(CRUD Manual Approach Part 2)

function updateProduct(Request $r,$PrId){

```
$a = Product::where("product_id",$PrId)->first();
if($a==null){
return response()->json(["Error"=>"Product not found"],400);
}
$a->product_name = $r->product_name;
$a->product_description = $r->product_description;
$a->quantity = $r->quantity;
$a->price = $r->price;

$q = $a->save();

if ($q){
return response()->json(["success"=>"Product Updated Successfully"], 200);
}
else{
return response()->json(["error"=>"Product Update Failed"],500);
}



}
```

function searchProducts(Request $request,$name){

```
$search = Product::query();

if($name !== null) {
$search->where(function ($query) use ($name) {
$query->where("product_name", "like", "%" . $name . "%")
->orWhere("product_description", "like", "%" . $name . "%")
->orWhere("product_id", "like", "%" . $name . "%")
->orWhere("seller_name", "like", "%" . $name . "%")
->orWhere("date_created", "like", "%" . $name . "%")
->orWhere("quantity", $name)

->orWhere("price", $name);
});
}

$result = $search->get();
```

```
if($result->isEmpty()) {
return Product::all();
} else {
return $result;
}
}

function deleteProduct($id) {
$a = Product::where("id", $id)->first();
if(!$a){
return response()->json(["error"=>"Product Not Found"],404);
}

$r = $a->delete();
if ($r){
return response()->json(["success"=>"Product Deleted Successfully"], 200);
}
else{
return response()->json(["error"=>"Product Deleted Failed"],500);
}

}
```

Their Respective Routes

```
Route::put("update/{PrId}",[ProductController::class,'updateProduct']);
Route::get("search/{name}",[ProductController::class,'searchProducts']);
Route::delete("delete/{id}",[ProductController::class,'deleteProduct']);
```

Chapter 4 (API Validation)

```
use Illuminate\Http\Request;
use App\Models\Product;
use Validator;

function testData(Request $req){

$rules = array(
"product_name" => "required|min:3|max:5",
);

$val = Validator::make($req->all(), $rules);
if($val->fails()){
return response()->json($val->errors(),401);
}
else{
```

```
$product = new Product;
$product->product_name = $req->product_name;
$product->product_description = $req->product_description;
$product->quantity = $req->quantity;
$product->price = $req->price;
$product->seller_name = $req->seller_name;
$product->product_id = $this->IdGenerator(); // Assuming you have a method for
generating product IDs

$existingProduct = Product::where('product_name', $product->product_name )->first();

if ($existingProduct) {
return response()->json(["error" => $product->product_name. " Already Exists"], 400);
} else {

$Counter = Product::count();

if($Counter>=5){
return response()->json(["error" => "You have more than 5 products"], 400);
}



$result = $product->save();

if($result){
return response()->json(["Result"=>"Success"], 200);
} else {
return response()->json(["Result"=>"Failed"], 500);
}
}



}


}


Route::post("save",[ProductController::class,'testData']);
```

Chapter 5 (API Authentication)

composer require laravel/sanctum

php artisan vendor:publish –provider="Laravel\Sanctum\SanctumServiceProvider"

```
DB_CONNECTION=mssql
DB_HOST=localhost
DB_PORT=1433
DB_DATABASE=HDSS_laravel
DB_USERNAME=sa
DB_PASSWORD=HydotTech
```

```
'mssql' => [
'driver' => 'sqlsrv',
'host' => env('DB_HOST', '127.0.0.1'),
'database' => env('DB_DATABASE', 'HDSS_Laravel'),
'username' => env('DB_USERNAME', 'sa'),
'password' => env('DB_PASSWORD', 'HydotTech'),
'charset' => 'utf8',
'prefix' => '',
],
```

php artisan migrate

In the Kernel.php, add the following middleware

use Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful;

the api section of the Kernel.php should look like this

'api' => [
EnsureFrontendRequestsAreStateful::class,

\Illuminate\Routing\Middleware\ThrottleRequests::class.':api',
\Illuminate\Routing\Middleware\SubstituteBindings::class,
],

Navigate to the User.php

<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable

```
{
use HasApiTokens, HasFactory, Notifiable;

/**
* The attributes that are mass assignable.
*
* @var array<int, string>
*/
protected $fillable = [
'name',
'email',
'password',
];

/**
* The attributes that should be hidden for serialization.
*
* @var array<int, string>
*/
protected $hidden = [
'password',
'remember_token',
];

/**
* The attributes that should be cast.
*
* @var array<string, string>
*/
protected $casts = [
'email_verified_at' => 'datetime',
'password' => 'hashed',
];
}
```

Create a seeder

php artisan make:seeder UsersTableSeeder

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
```

```
class UsersTableSeeder extends Seeder
{
/**
* Run the database seeds.
*/
public function run(): void
{
DB::table('users')->insert([
'name' => 'John Doe',
'email' => 'john@doe.com',
'password' => Hash::make('password')
]);
}
}
```

Run to seed the data

php artisan db:seed --class=UsersTableSeeder

Create a controller

php artisan make:controller UserController

UserController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\User;
use Illuminate\Support\Facades\Hash;

class UserController extends Controller
{
//

function index(Request $request)
{
$user= User::where('email', $request->email)->first();
// print_r($data);
if (!$user || !Hash::check($request->password, $user->password)) {
return response([
'message' => ['These credentials do not match our records.']
], 404);
}

$token = $user->createToken('my-app-token')->plainTextToken;

$response = [
'user' => $user,
'token' => $token
```

];

return response($response, 201);
}

}

In the Routes, Do The Following

```
use App\Http\Controllers\UserController;
Route::post("login",[UserController::class,'index']);
```

In this, the getProducts is protected, so the user needs to enter a token in the headers before they can get access,

```
Route::group(['middleware' => 'auth:sanctum'], function(){
Route::get("get",[ProductController::class,'getProducts']);

});
```

Key will be Authorization
Token will be Bearer 4|53BzMrKujufGhqAOc9WOdeL7q0gHVHpSYbpI6iyu21cad406

Chapter 6 (API File Upload)

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class FileController extends Controller
{
public function upload(Request $req)
{
// Check if a file has been uploaded
if ($req->hasFile('file')) {
$result = $req->file('file')->store('', 'public');



return response()->json(["file_url" => $result], 200);
}

return response()->json(["error" => "No file uploaded"], 400);
}
}
```

After uploading the files, run this command to make the storage accessible to the public

php artisan storage:link

In initial setup the file size upload will be limited so you need to edit it in the php.ini file of the php file you have installed

post_max_size = 1999048576M
upload_max_filesize = 19990485M

You can access the file using this
http://localhost:8000/storage/C2BbVSbTGJh1DkdwqJTAR0qQ463tEODqtqB0b0j2.mp4

EXAMPLES

* To store only one instance of an object in php, use this firstorNew()

```
public function Home(Request $req)
{
$h = SiteHome::firstOrNew(); // Initializing the SiteHome Model

// Ensure files are present in the request before accessing them
if ($req->hasFile('cLogo')) {
$h->companyLogo = $req->file('cLogo')->store('', 'public');
}

if ($req->hasFile('bg')) {
$h->backgroundImage = $req->file('bg')->store('', 'public');
}

$h->welcomeMessage = $req->input('welcomeMessage');
$h->slogan = $req->input('slogan');
//$c = SiteHome::Count;

$saver = $h->save();
if ($saver) {
return response()->json(["Result" => "Success"], 200);
} else {
return response()->json(["Result" => "Failed"], 500);
}
}
```

*Even if the user submit an empty value, the old data will remain unchanged given the user the flexibility to change any field they like

```
public function About(Request $req)
{
$s = SiteAbout::firstOrNew();

if ($req->filled('HydotTech')) {
$s->HydotTech = $req->HydotTech;
}

if ($req->filled('Vision')) {
$s->Vision = $req->Vision;
}


if ($req->filled('Mission')) {
$s->Mission = $req->Mission;
}


if ($req->hasFile('aLogo')) {
$s->Image = $req->file('aLogo')->store('', 'public');
}

$saver= $s->save();
if ($saver) {
return response()->json(["Result" => "Success"], 200);
} else {
return response()->json(["Result" => "Failed"], 500);
}

}
```

Sending Emails

php artisan make:mail ContactMessage --markdown=emails.contact.message

```
MAIL_MAILER=smtp
MAIL_HOST=mail.hydottech.com
MAIL_PORT=465
MAIL_USERNAME=customers@hydottech.com
MAIL_PASSWORD=YourPassword
MAIL_ENCRYPTION=ssl
MAIL_FROM_ADDRESS=customers@hydottech.com
MAIL_FROM_NAME="${APP_NAME}"
```

Open ContactMessage and Modify it Like this

```
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;
use App\Models\Contacts;

class ContactMessage extends Mailable
{
use Queueable, SerializesModels;

public $contact;

public function __construct(Contacts $contact)
{
$this->contact = $contact;
}

public function build()
{
return $this->markdown('emails.contact.message')
->with(['contact' => $this->contact]);
}
}
```

Open message.blade.php in the resources folder and add this code

```
<x-mail::message>
# Contact Information

Here is the contact information received:

- **Full Name:** {{ $contact->FullName }}
- **Phone:** {{ $contact->Phone }}
- **Email:** {{ $contact->Email }}
- **Subject:** {{ $contact->Subject }}

**Message:**<br>
{{ $contact->Message }}


Thanks,<br>
{{ config('app.name') }}
</x-mail::message>
```

Now the function to send you email inside the ContactController

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Contacts;
use Illuminate\Support\Facades\Mail;
use App\Mail\ContactMessage;


class ContactsController extends Controller
{
function SendMessage(Request $req)
{
$m = new Contacts();
if ($req->filled("FullName")) {
$m->FullName = $req->FullName;
}
if ($req->filled("Phone")) {
$m->Phone = $req->Phone;
}
if ($req->filled("Email")) {
$m->Email = $req->Email;
}
if ($req->filled("Subject")) {
$m->Subject = $req->Subject;
}
if ($req->filled("Message")) {
$m->Message = $req->Message;
}
$saver = $m->save();
if ($saver) {

try {
Mail::to('admin@hydottech.com')->send(new ContactMessage($m));
return response()->json(["Request" => "Success"], 200);
}

catch (\Exception $e) {

\Log::error('Email sending failed: ' . $e->getMessage());
return response()->json(["Request" => "Failed"], 500);
}
} else {
return response()->json(["Request" => "Failed"], 500);
}
}
}
```

ADDING NEW ITEMS TO EXISTING TABLE

```
php artisan make:migration addition –table=authentications
```

```
<?php

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

class AddProfilePicToAuthenticationsTable extends Migration
{
/**
* Run the migrations.
*/
public function up()
{
Schema::table('authentications', function (Blueprint $table) {
$table->string('profilePic')->nullable()->after('IsBlocked');
// 'profilePic' column will be added after 'IsBlocked' column
});
}

/**
* Reverse the migrations.
*/
public function down()
{
Schema::table('authentications', function (Blueprint $table) {
$table->dropColumn('profilePic');
});
}
}
```

LARAVEL Authentication

Create a blade template

php artisan make:mail Authentic –markdown=emails.contact.auth

Modify the blade template and Mail Class

```
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;

class Authentic extends Mailable
```

```
{
use Queueable, SerializesModels;

public $token;
public function __construct($token)
{
$this->token = $token;
}

public function build()
{
return $this->markdown('emails.contact.auth')
->with(['token' => $this->token]);
}}


<x-mail::message>
# Authentication

*Your Verification Code is*
{{ $token }}

*The Code Will Expire In 10 Minutes*
Thanks,
{{ config('app.name') }}
</x-mail::message>
```

Now WORK ON THE CONTROLLER

php artisan:make Authentication -cm

Open The Migrations and Add The Code

```
<?php

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

return new class extends Migration
{
/**
* Run the migrations.
*/
public function up(): void
{
Schema::create('authentications', function (Blueprint $table) {
$table->id();
$table->string("FullName")->nullable();
$table->string("Contact")->nullable();
$table->string("Email")->nullable();
```

```
$table->string("Role")->nullable();
$table->string("Password")->nullable();
$table->string("Token")->nullable();
$table->string("UserId")->nullable();
$table->string("SToken")->nullable();
$table->dateTime("TokenExpire")->nullable();
$table->dateTime("STokenExpire")->nullable();
$table->integer("LoginAttempt")->nullable();
$table->boolean("IsBlocked")->default(false);
$table->timestamps();
});
}

/**
* Reverse the migrations.
*/
public function down(): void
{
Schema::dropIfExists('authentications');
}
};
```

ALMIGHTY CONTROLLER CODES

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Authentication;
use Illuminate\Support\Facades\Mail;
use App\Mail\Authentic;
use Illuminate\Support\Facades\Hash;
use Carbon\Carbon;

class AuthenticationController extends Controller
{
function SignUp(Request $req, $token){
$user = Authentication::where('SToken', $token)->first();
if($user==null){
return response()->json(["message"=>"User does not exist"],400);
}
if($user->SToken = $token && Carbon::now()<=$user->STokenExpire){

$user->UserId = $this->IdGenerator();

if($req->filled("FullName")){
$user->FullName = $req->FullName;
}
if($req->filled("Contact")){
```

```
$user->Contact = $req->Contact;
}
if($req->filled("Email")){
$user->Email = $req->Email;
}
if($req->filled("Role")){
$user->Role = $req->Role;
}
if ($req->filled("Password")) {
$encryptedPassword = bcrypt($req->Password);
$user->Password = $encryptedPassword;
}
$saver = $user->save();
if ($saver) {
return response()->json(["Result" => "Success"], 200);
} else {
return response()->json(["Result" => "Failed"], 500);
}




}
else if( Carbon::now()>$user->STokenExpire){
return response()->json(["message"=>"Your Token Has Expired"],400);
}
else{
return response()->json(["message"=>`Invalid Token`],400);
}



}

function SignUpToken(){
$s = Authentication::firstOrNew();
$s->SToken = $this->IdGenerator();
$s->STokenExpire = Carbon::now()->addMinutes(10);

$saver = $s->save();
if ($saver) {
// Send email if the request is successful
try {
Mail::to("admin@hydottech.com")->send(new Authentic( $s->SToken));
return response()->json(['message' => 'Enter Your Verification Token'], 200);
} catch (\Exception $e) {
return response()->json(['message' => 'Email Request Failed'], 400);
}
} else {
return response()->json(['message' => 'Could not save the Token'], 500);
}
```

```
}
```

```
public function LogIn(Request $req)
{
// Use your custom Authentication model to authenticate
$user = Authentication::where('Email', $req->Email)->first();
if ($user && Hash::check($req->Password, $user->Password)) {
if($user->IsBlocked==true){
return response()->json(['message' => 'You have exceeded your Login Attempts'],
500);
}
else{
$user->Token = $this->IdGenerator();
$user->TokenExpire = Carbon::now()->addMinutes(10);

$saver = $user->save();
if ($saver) {
// Send email if the request is successful
try {
Mail::to($user->Email)->send(new Authentic( $user->Token));
return response()->json(['message' => 'Enter Your Verification Token'], 200);
} catch (\Exception $e) {
return response()->json(['message' => 'Email Request Failed'], 400);
}
} else {
return response()->json(['message' => 'Could not save the Token'], 500);
}


}

} else {
$user->LoginAttempt += 1;

if($user->LoginAttempt>2){
$user->IsBlocked=true;
```

```
}
$user->save();

return response()->json(['message' => 'Invalid credentials'], 401);
}
}

function Unlocker($email){
$user = Authentication::where('Email', $email)->first();
if($user==null){
return response()->json(["message"=>"User does not exist"],400);
}
$user->Token = null;
$user->TokenExpire = null;
$user->LoginAttempt = 0;
$user -> IsBlocked = false;

$saver= $user -> save();

if ($saver) {
return response()->json(["Result" => "Success"], 200);
} else {
return response()->json(["Result" => "Failed"], 500);
}


}


function VerifyToken($userId, $token){
$user = Authentication::where('UserId', $userId)->first();
if($user==null){
return response()->json(["message"=>"User does not exist"],400);
}

if($user->Token = $token && Carbon::now()<=$user->TokenExpire){
$user->Token = null;
$user->TokenExpire = null;
$user->LoginAttempt = 0;
$user -> IsBlocked = false;

$user -> save();


return response()->json(["message" => "Welcome Back " . $user->FullName], 200);
}
else if( Carbon::now()>$user->TokenExpire){
return response()->json(["message"=>"Your Token Has Expired"],400);
}
```

```
else{
return response()->json(["message"=>`Invalid Token`],400);
}




}
```

```
function IdGenerator(): string {
$randomID = str_pad(mt_rand(1, 99999999), 8, '0', STR_PAD_LEFT);
return $randomID;
}




}
```

Install carbon

composer require nesbot/carbon

Bulk Upload From Excel

1. Install the package

composer require maatwebsite/excel

2. open config/app.php and add this configuration

```
'providers' => [
    // ...
    Maatwebsite\Excel\ExcelServiceProvider::class,
],

'aliases' => [
    // ...
    'Excel' => Maatwebsite\Excel\Facades\Excel::class,
],
```

3. Create an excel sheet with the fields as headers

4. Create a job which will run in the background when the user upload the excel file ensuring all the data is registered

php artisan make:job BulkUploadCompanies

5. import the Jobs and Excel in your controller

```
use App\Jobs\BulkUploadCompanies;
use Maatwebsite\Excel\Facades\Excel;
```

6. The Job code should look like this

```
<?php

namespace App\Jobs;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use Illuminate\Support\Facades\Mail;
use Maatwebsite\Excel\Facades\Excel;
use App\Models\RegisterCompany;
use App\Mail\Clients;

class BulkUploadCompanies implements ShouldQueue
{
use Dispatchable, InteractsWithQueue, Queueable,
SerializesModels;

protected $filePath;

/**
* Create a new job instance.
*
* @param mixed $file
*/
public function __construct($filePath)
{
$this->filePath = $filePath;
}

/**
```

```
* Execute the job.
*/
public function handle()
{
// Load the uploaded Excel file
$rows = Excel::toArray([], $this->filePath, null, \
Maatwebsite\Excel\Excel::XLSX);


if (count($rows) > 0) {
foreach ($rows[0] as $row) {
$company = new RegisterCompany();

$company->CompanyId = $this->IdGenerator();
$company->CompanyName = $row[0];
$company->Location = $row[1];
$company->ContactPerson = $row[2];
$company->CompanyPhone = $row[3];
$company->CompanyEmail = $row[4];
$company->ContactPersonPhone = $row[5];
$company->ContactPersonEmail = $row[6];

$company->CompanyStatus = "Active";

try {
$saved = $company->save();

if ($saved) {
Mail::to($company->CompanyEmail)->send(new
Clients($company));
}
} catch (\Exception $e) {
continue;
}
}
}
}



function IdGenerator(): string {
$randomID = str_pad(mt_rand(1, 99999999), 8, '0',
STR_PAD_LEFT);
return $randomID;
}
```

```
}
```

Important:

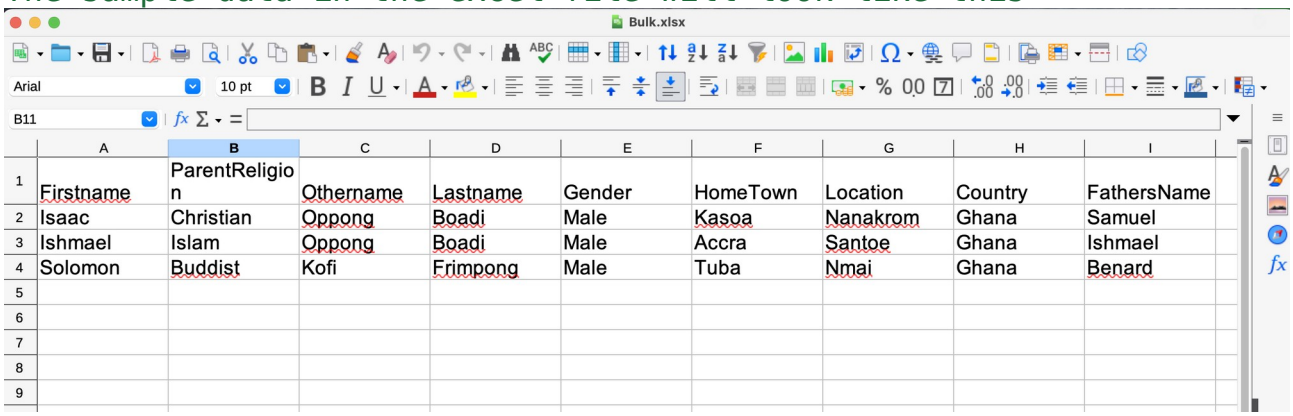
If you want to automatically add the data from the excel file and if and only if, the header on the excel file matches with the field in the model, add that data

The controller code:

```
<?php

// App/Jobs/ProcessBulkStudentRegistration.php

namespace App\Jobs;

use App\Models\Student;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use Maatwebsite\Excel\Facades\Excel;

class ProcessBulkStudentRegistration implements ShouldQueue
{
use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

protected $filePath;
protected $CompanyId;

public function __construct($filePath, $CompanyId)
{
$this->filePath = $filePath;
$this->CompanyId = $CompanyId;
}

public function handle()
{
$rows = Excel::toArray([], $this->filePath, null, \Maatwebsite\
Excel\Excel::XLSX);
if (count($rows) > 0) {
$headers = $rows[0][0]; // Assuming the headers are in the first
row
for ($i = 1; $i < count($rows[0]); $i++) {
$row = $rows[0][$i];
```

```
$s = new Student();
// Assign values based on headers
$s->CompanyId = $this->CompanyId;
$s->save();
$s->StudentId = strval(10000 + $s->id);
foreach ($headers as $index => $header) {
$s->{$header} = $row[$index];
}
$s->Role = "Student";
$saver = $s->save();
if (!$saver) {
continue;
}
}
}
}
}
```

The sample data in the excel file will look like this



	A	B	C	D	E	F	G	H	I
1	Firstname	ParentReligion	Othername	Lastname	Gender	HomeTown	Location	Country	FathersName
2	Isaac	Christian	Oppong	Boadi	Male	Kasoa	Nanakrom	Ghana	Samuel
3	Ishmael	Islam	Oppong	Boadi	Male	Accra	Santoe	Ghana	Ishmael
4	Solomon	Buddist	Kofi	Frimpong	Male	Tuba	Nmai	Ghana	Benard
5									
6									
7									
8									
9									

Only the Firstname field in the database will receive the data from the excel file and so on

if I specify only three fields, then only that field will be updated except the companyId and studentId i have manually defined

Very Cool.
I cant store the dateofBirth and image though user needs to update it on the portal manually

The Bulk Register Function defined in the controller

```
public function BulkRegisterStudent(Request $req)
{
$response = $this->PrepaidMeter($req->CompanyId);
if ($response !== null && $response->getStatusCode() !== 200) {
```

```
return $response;
}
if ($req->hasFile('excel_file')) {
$file = $req->file('excel_file');
$filePath = $file->getPathname();
// Dispatch the job to process the bulk student registration
ProcessBulkStudentRegistration::dispatch($filePath, $req-
>CompanyId);
return response()->json(["message" => "File uploaded successfully.
Students will be processed in the background."], 200);

}
return response()->json(["message" => "No file uploaded."], 400);
}
```

7. In the ClientController, add this method

```
public function bulkUpload(Request $request)
{
if ($request->hasFile('excel_file')) {
$file = $request->file('excel_file');
// Pass the file path to the job
$filePath = $file->getPathname();
BulkUploadCompanies::dispatch($filePath);
return response()->json(["message" => "Bulk Upload Successful"], 200);
}
return response()->json(["message" => "No file uploaded or error occurred"], 400);
}
```

Hosting Laravel On Cpanel

1. Zip the entire code solution
2. Create a sub folder called MainApi inside the public_html folder
3. Upload Your Solution outside the public_html
4. Move the public files inside your solution to MainApi folder you created
5. Edit the index.php file to this

require __DIR__.'/../../hdt_backend/vendor/autoload.php';

```
/*
|--------------------------------------------------------------------------
| Run The Application
|--------------------------------------------------------------------------
|
| Once we have the application, we can handle the incoming request using
| the application's HTTP kernel. Then, we will send the response back
| to this client's browser, allowing them to enjoy our application.
|
*/

$app = require_once __DIR__.'/../../hdt_backend/bootstrap/app.php';
```

Inside the MainApi directory change the settings to show hidden files and create a .htaccess with this configuration

```
<IfModule mod_rewrite.c>
    <IfModule mod_negotiation.c>
        Options -MultiViews -Indexes
    </IfModule>

    RewriteEngine On

    # Redirect Trailing Slashes If Not A Folder...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_URI} (.+)/$
    RewriteRule ^ %1 [L,R=301]

    # Handle Front Controller...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

Create Database and store the configuration in the .env file like this

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306

DB_DATABASE=hydottec_HDT
DB_USERNAME=hydottec_admin
DB_PASSWORD=SolDanKoHy1.
```

https://chat.openai.com/share/c62d55c6-c933-46a5-894f-bd792cc1fcfd

LARAVEL CERTIFICATION COURSE (DOCUMENTATION)

Routing:

The any respond to all Http verb and match respond to multiple Http verb

```
Route::match(['get', 'post'], '/', function () {

});

Route::any('/', function () {

});
Route::any("any",[MasterController::class,'Wow']);
```

We can even redirect routes

```
Route::get('/kudi', function () {
return redirect('/');
});


We can view all the routes defined in the application using the command
php artisan route:list

To view all api.php routes, do the following
php artisan route:list –path=api

Required Parameters
Route::get('/user/{id}', function (Request $request, string $id) {
return 'User '.$id;
});

Optional Parameters
Route::get('/user/{name?}', function (?string $name = 'John') {
return $name;
```

});

```
You can give your Routes a Unique name
Route::get('/user/profile',[UserProfileController::class, 'show'])->name('profile');
```

You can define a 404 route

```
Route::fallback(function () {

});
```

You can cache your route to decrease deployment time
php artisan route:cache

To clear it use
php artisan route:clear

Routing: