LARAVEL 10

Part 2: DB & Models

CONNECT TO DB

Configure DB from .env file

```
DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=dbName

DB_USERNAME=root

DB_PASSWORD=
```

MYSQL DB CONECTION

Config → database.php

Create your DB

From .env add

```
DB_SOCKET=/Applications/MAMP/tmp/mysql/mysql.sock
```

From cmd

php artisan migrate

MIGRATION ARTISAN COMMANDS

- php artisan migrate:status
- php artisan migrate:rollback
- php artisan migrate:reset
- php artisan migrate:refresh
- php artisan migrate:fresh

https://laravel.com/docs/10.x/migrations

CREATE YOUR FIRST MIGRATION TABLE

From cmd type

php artisan make:migration create_clients_table

And add the columns and constraints

https://laravel.com/docs/10.x/migrations#creating-tables

https://www.heinsoe.com/blog/85

CREATE YOUR FIRST MODEL

From cmd type

php artisan make:model Client

Note: First letter is capital and is a single to word clients which used in the previous slide

USE MODEL IN A CONTROLLER

Inside your controller use the model

```
use App\Models\Client;
```

INSERT DATA TO DB

Inside your controller use the method

```
public function store(Request $request)
    {
        $client = new Client();
        $client->name = $request->name;
        $client->email = $request->email;
        $client->save();
        return 'Added Successfully';
    }
```

GET DATA USING THE MODEL

Create blade file for your data

Inside the controller use method

```
public function index()
    {
        $client = Client::get();
        return view("clientsList", compact("client"));
    }
```

GET DATA USING THE MODEL

Inside you blade file you can get the method data

```
@foreach($client as $data)
        <tr>
          <td>{{ $data->name }}</td>
          <td>{{ $data->email }}</td>
        </tr>
@endforeach
```

INSERT DATA TO DB (ANOTHER METHOD)

Inside the model

```
class Client extends Model
{
    use HasFactory;
    protected $fillable = [
        'name',
        'email',
    ];
}
```

INSERT DATA TO DB (ANOTHER METHOD)

Inside the controller

Define array as a property to the controller class

```
private $columns = ['name', 'email'];


public function store(Request $request): RedirectResponse
    {
        Client::create($request->only($this->columns));
        return redirect('clients');
    }
```

UPDATE DB

Update blade file table

```
@foreach($client as $data)
  <tr>
    <td>{{ $data->name }}</td>
    <td>{{ $data->email }}</td>
    <td><a href="editClient/{{ $data->id }}">Edit</a></td>
  </tr>
@endforeach
```

UPDATE DB

Create blade file for the update form

```
<form action="{{ route('updateClient',[$client->id]) }}" method="post">
    @csrf
    @method('put')
    <input type="text" name="name" placeholder="Name" value="{{ $client->name }}">
    <hr>
    <input type="text" name="email" placeholder="Email" value="{{ $client->email }}">
    <input type="submit" value="Submit">
</form>
```

UPDATE DB

Create Routes

```
Route::get("editClient/{id}", [exampleController::class,"edit"])-
>name('editClient’);
```

```
Route::put("updateClient/{id}",
[exampleController::class,"update"])->name('updateClient');
```

UPDATE DB

Inside the controller

```
public function edit(string $id)
{
    $client = Client::findOrFail($id);
    return view('updateClient', compact('client'));
}

public function update(Request $request, string $id): RedirectResponse
{
    Client::where('id', $id)->update($request->only($this->columns));
    return redirect('clients');
}
```

SHOW 1 ROW FROM DB

Update your blade file to have a show link

```
@foreach($client as $data)
  <tr>
    <td>{{ $data->name }}</td>
    <td>{{ $data->email }}</td>
    <td><a href="editClient/{{ $data->id }}">Edit</a></td>
    <td><a href="showClient/{{ $data->id }}">Show</a></td>
  </tr>
@endforeach
```

SHOW 1 ROW FROM DB

- Create a blade file to show the data
- Add new route

```
Route::get("show/{id}", [exampleController::class,"show"])-
>name('showClient');
```

SHOW 1 ROW FROM DB

Add method to your controller

```
public function show(string $id)
{
    $client = Client::findOrFail($id);
    return view('show', compact('client'));
}
```

DELETE FROM DB

Inside the clients list blade file add in a new td tag

```
<td>
  <form action="{{ route('deleteClient') }}" method="post">
    @csrf
    @method('DELETE')
    <input type="hidden" name="id" value="{{ $data->id }}">
    <input type="submit" value="delete">
  </form>
</td>
```

DELETE FROM DB

Add new route

```
Route::delete("delete", [exampleController::class,"destroy"])-
>name('deleteClient');
```

DELETE FROM DB

Add new method to your controller

```
public function destroy(Request $request): RedirectResponse
{
    $id = $request->id;
    Client::where('id', $id)->delete();
    return redirect('clients');
}
```

SOFTDELETE

Database → migrations → Your table file

Add

```
$table->softDeletes();
```

Then be sure to migrate from cmd (fresh to add only the new column)

```
php artisan migrate:fresh
```

SOFTDELETE

App → Models → your model name

Add below

```
use Illuminate\Database\Eloquent\SoftDeletes;
```

Inside the class be sure to use the softDeletes like this

```
use HasFactory, SoftDeletes;
```

FORCE DELETE

Example in your controller

```
public function destroy(Request $request): RedirectResponse
    {
        $id = $request->id;
        //Client::where('id', $id)->forceDelete(); // to permanent delete
       Client::where('id', $id)->delete(); // softdelete
        return redirect('clients');
    }
```

RESTORE DELETED RECORDS

Add a blade file for the trashed records and be sure to add a button to restore

```
<td>
  <form action="{{ route('restore') }}" method="post">
    @csrf
    <input type="hidden" name="id" value="{{ $data->id }}">
    <input type="submit" value="Restore">
  </form>
</td>
```

RESTORE DELETED RECORDS

Add method to your controller to show deleted records page

```
public function showDeleted()
    {
        $client = Client::onlyTrashed()->get();
        return view('trashedClients', compact('client'));
    }
```

RESTORE DELETED RECORDS

Add method to your controller for restore

```
public function restore(Request $request): RedirectResponse
{
    $id = $request->id;
    Client::where('id', $id)->restore();
    return redirect('clients');
}
```

RESTORE DELETED RECORDS

Add route

```
Route::get("deleted", [exampleController::class,"showDeleted"])-
>name('showDeleted');

Route::post("restore", [exampleController::class,"restore"])-
>name('restore');
```

FORM VALIDATION

Inside your controller you can edit the store method

```
    public function store(Request $request): RedirectResponse
    {
        $request->validate([
            'name'=>'required|string',
            'email'=> 'requierd|string'
        ]);
        Client::create($request->only($this->columns));
        return redirect('clients');
    }
```

FORM VALIDATION

Inside the form you can add

```
@error('name')
    {{ $message }}
@enderror
```

FORM VALIDATION

To restore old values in the form use value old as below

```
<input type="text" name="name" placeholder="Name" value="{{
old('name') }}">
```

CHECKBOX RETRIEVE OLD VALUE

```
<input type="checkbox" name="checkboxName" value="1" {{
old('checkboxName') ? 'checked' : '' }}>
```

Or

@checked(old('checkBoxName'))

FORM VALIDATION

To restore old values in the form use value old as below

For dropdown

```
<option value="option1" {{ old('selectField') == 'option1' ?
'selected' : '' }}>Option 1</option>
```

Or

```
<option value="option1" {{ @selected(old('selectField') ==
"option1") }}>Option 1</option>
```

USE VALIDATION FOR STORE (BEST PRACTICE)

Update the store method as below

```
    public function store(Request $request): RedirectResponse
    {
        $data = $request->validate([
            'name' => 'required|string',
            'email' => 'required|string'
        ]);
        Client::create($data);
        return redirect('clients');
    }
```

CUSTOM ERROR MESSAGES

```php
$messages=[
            'title.required'=>'Title is required',
            'title.string'=>'Should be string',
            'description.required'=> 'should be text',
        ];
$validation = $request->validate([
            'title'=>'required|string',
            'description'=> 'required|string',
            'user_id'=> 'required'
            ], $messages);

        Post::create($validation);
        return 'Added Successfully';
```

MORE ABOUT VALIDATION

https://laravel.com/docs/10.x/validation

Validation rules

https://laravel.com/docs/10.x/validation#available-validation-rules

UPLOAD FILE

Be sure to add the images folder to the public folder → assets

Goto config → filesystems.php and add below code

```
'imgs' => [
            'driver' => 'local',
            'root' => base_path() . 'public/assets/images/',
            'url' => env('APP_URL').'/public',
            'visibility' => 'public',
            'throw' => false,
        ],
```

UPLOAD FILE

More details about filesystems.php

https://laravel.com/docs/10.x/filesystem

UPLOAD FILE - CREATE VIEW FORM

Example

```
<form action="{{ route('upload') }}" method="post"
enctype="multipart/form-data">
        @csrf
        <input type="file" name="photo">
        <br>
        <input type="submit" name="upload" value="Upload">
    </form>
```

UPLOAD FILE - CONTROLLER

Add the method

```
public function upload(Request $request){
        $file_extension = $request->photo->getClientOriginalExtension();
        $file_name = time() . '.' . $file_extension;
        $path = 'images';
        $request->photo->move($path, $file_name);
        return 'Uploaded';
    }
```

ANOTHER METHOD
UPLOAD FILE – CREATE TRAIT FOR YOUR UPLOADER

Add Traits folder to App and add file Common.php, then add below code

```php
namespace App\Traits;

Trait Common {
    public function uploadFile($file, $path){
        $file_extension = $file->getClientOriginalExtension();
        $file_name = time() . '.' . $file_extension;
        $file->move($path, $file_name);
        return $file;
    }
}
```

ANOTHER METHOD
UPLOAD FILE – CREATE TRAIT FOR YOUR UPLOADER

You can call the Trait from the controller like below

```
use App\Traits\Common;
```

Inside the class

```
use Common;
```

And inside the method

```
$fileName = $this->uploadFile($request->photo, 'images');
```

ANOTHER METHOD
UPLOAD FILE – CREATE TRAIT FOR YOUR UPLOADER

Inside your store controller method you can use

```
$data = $request->validate([
            'title' => 'required',
            'description' => 'required',
            'image' => 'required|mimes:png,jpg,jpeg|max:2048',
        ]);
        $fileName = $this->uploadFile($request->image, 'assets/images');
        $data['image']= $fileName;
        Car::create($data);
        return 'Created successfully';
```

STORE METHOD INCLUDING CHECKBOX VALUE

```php
$data = $request->validate([
            'title' => 'required',
            'description' => 'required',
            'image' => 'required|mimes:png,jpg,jpeg|max:2048',
        ]);
        $fileName = $this->uploadFile($request->image, 'assets/images');
        $data['image']= $fileName;
        $data['published'] = isset($request->published);
        Car::create($data);
        return 'Created successfully';
```

Thank you

KOICA
Korea International Cooperation Agency

السويد
Sverige

ECTD
Egyptian Council
For Training & Development

العمل اللائق للمرأة

منظمة العمل الدولية

هيئة الأمم المتحدة للمرأة

المجلس القومي للمرأة
THE NATIONAL COUNCIL FOR WOMEN