# Laravel installation

1. Make sure you have PHP and composer installed. Then run the following command to install the Laravel installer:  
     
   composer global require Laravel/installer
2. Make sure you have Node and NPM installed. If they are not installed, download [nvm-windows](https://github.com/coreybutler/nvm-windows/releases).

Run the command ***nvm arch*** to see on what bit mode node is running.   
Run the command ***nvm install lts*** to install the latest node version.

Restart the terminal and check if Node and NPM installed successfully.

# Project creation

1. Create an application:

Laravel new example-app

1. Access the directory of the project and run the command below:

cd example-app

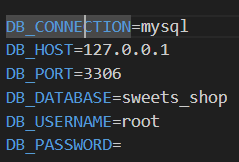
npm install && npm run build

# Project configuration

1. Start Apache and MySQL server using xampp. Access MySQL server and create the database you wish to use in your Laravel project:

Create database db\_name

1. Open the .env file from Laravel project and configure the database server and name:



1. Run the default migrations:

Php artisan migrate

1. Start the Laravel server:

php artisan serve

# Implementing CRUD functionality

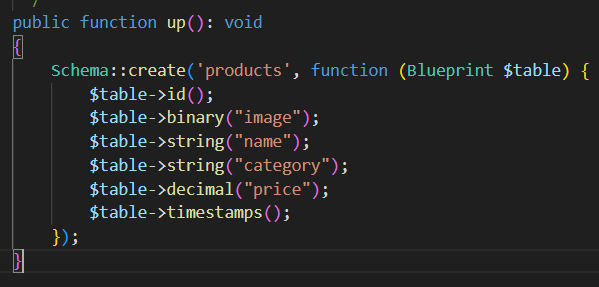


Figure 1: MVC architecture in Laravel

1. Create the migration table:

php artisan make:migration create\_products\_table

1. Specify the fields of the table:



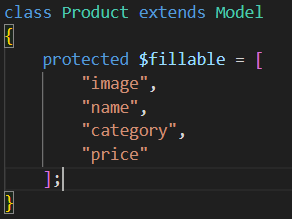
1. Run the migrations:

php artisan migrate

1. Create the model:

php artisan make:model Product

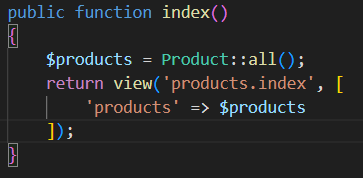
1. Specify the fillable fields of the table:



1. Create the controller:

php artisan make:controller ProductController

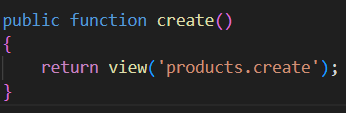
1. Define the index function in the created controller:



1. Create the index view
2. Create the route for index view:

Route::get('/products', [ProductController::class, 'index'])->name('product.index');

1. Define the create function in the controller:



1. Make the create view

Add the method="POST" and enctype="multipart/form-data" attributes to the form if you are going to upload files to the server. Also, specify the @csrf and @method("POST") tags after the form.

You can add the link to the create view in the index view pasting the code below:

<a href="{{route('product.create')}}">Create a product</a>

1. Define the create route:

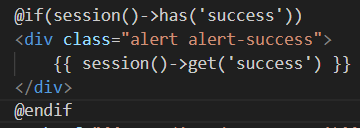
Route::get('/products/create', [ProductController::class, 'create'])->name('product.create');

1. Create the store function:



If you are not working with files, you can skip the if else declaration.

To get the success messages, go to the index view and add the following code:



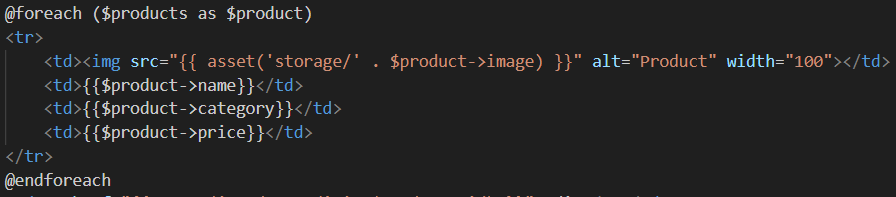
1. Create the route for the store method:

Route::post('/products', [ProductController::class, 'store'])->name('product.store');

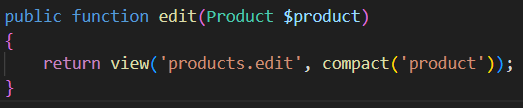
1. Go to the create view and add the store route to the form action attribute:

action="{{route('product.store')}}"

You can access the stored attributes in the index page using a foreach loop:



1. Create the edit function:



1. Create the edit view, which is similar to the create view.

You can access the curent stored attribute in the same way you accessed them in the index view, using the value="{{$product->name}}" attribute in the input element.

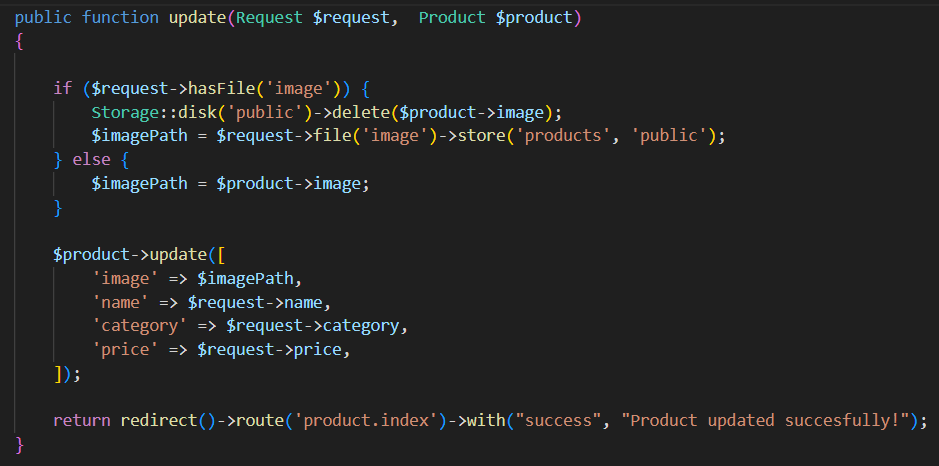
1. Create the edit route:

Route::get('/products/{product}/edit', [ProductController::class, 'edit'])->name('product.edit');

1. Navigate to index route and add the following code:

<td><a href="{{ route('product.edit', $product->id) }}">Edit</a></td>

1. Create the update function, which should be similar to the store function, with few differences:



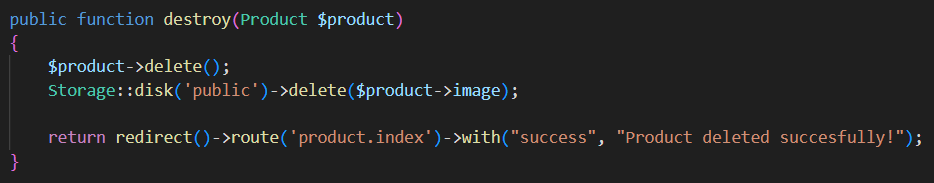
1. Go to the edit view and add the action="{{route('product.update', ['product' => $product->id])}}" attribute to the form.

Don’t forget to specify @csrf and @method("PUT") tags after the form.

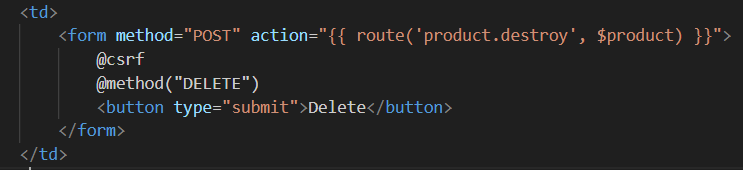
1. Create the update route:

Route::put('/products/{product}', [ProductController::class, 'update'])->name('product.update');

1. Create the destroy function:



1. Access the index view and add the following code:



1. Create the destroy route:

Route::delete('/products/{product}', [ProductController::class, 'destroy'])->name('product.destroy');