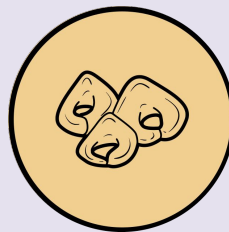


Tad&Apd



Studenti:

- Christian Angileri (330925)
- Alessio Cassieri (324396)



Progetto di sistemi paralleli e distribuiti

Il progetto verte sulla creazione di un web service che offra servizi per una ditta produttrice di tortellini attraverso un e-commerce.

La richiesta del progetto si focalizzava su dei punti generali da rispettare:

- Distinguere due tipologie di utenti del servizio: **Erogatori** e **Fruitori**;
- Utilizzo di **SOAP/REST** su piattaforma a scelta;
- L'utilizzo di un database per l'archiviazione dei dati;
- Descrizione del servizio in **WSDL** (se formato testo in XML);
- **Documentazione** divisa in: commenti su codice, elaborato testo e presentazione.

Per svolgere questo progetto abbiamo utilizzato un framework di php chiamato **Laravel** che attraverso il suo modello **MVC**, il motore di template **Blade** e l'**ORM** Eloquent permette di gestire in modo efficace le view attraverso i controller, richiamare gli oggetti attraverso i tag Blade su web e di gestire le query del database. Come database è stato utilizzato **mysql** sulla piattaforma **XAMPP** costituita da Apache HTTP server. Successivamente è stato utilizzato **Jetstream** di Laravel che è un pacchetto completo e personalizzabile per l'autenticazione degli utenti e la gestione delle sessioni nelle applicazioni web Laravel, che offre una solida base per lo sviluppo di applicazioni sicure ed efficienti. Ultima componente chiave è stato quello di fare in modo che lo **sviluppatore** e l'**admin** del web service fossero 2 figure separate ponendo quindi come obiettivo principale quello di rendere all'admin semplici determinate operazioni di modifica e aggiunta.




Table of contents



01

Struttura MVC

Costruzione del web service usando i controller

02

Gestione database

Migrations, Seeder, .env e mysql

03

Routing

Chiamate HTTP: GET e POST

04

Demo

Demo in tempo reale + riferimenti





01

Struttura MVC



Sezione Controller

In Laravel i controller si trovano in *app/Http/Controller* in cui nel nostro caso abbiamo principalmente:

1. AdminController
2. ProductsController
3. HomeController

I controller permettono di gestire i **modelli**, le **route** e utilizzare **dati** del database usando Eloquent richiamando funzioni. Ecco un esempio di controller che seleziona tutti gli utenti presenti sul sito e li ritorna nella view *admin/userslist*:

```
function list(){  
    $users = User::all();  
    return view('admin/userslist',compact('users'));  
}
```



1. AdminController

```
class AdminController extends Controller
{
    //stampa di tutti gli utenti
    function list(){
        $users = User::all();
        return view('admin/userslist',compact('users'));
    }

    //canellazione utenti
    function deleteUser(Request $request){
        DB::table('orders')->where('user_id',$request->deleteuser->delete();
        DB::table('users')->where('id',$request->deleteuser->delete();
        return redirect()->back()->with('alert', 'Eliminato con successo!'); //t
    }

    //modifica dei prodotti
    function modProdView(){
        $prods = Product::all();
        return view('admin/modprod',compact('prods'));
    }

    // eliminazione del prodotto
    function deleteProd(Request $request){
        DB::table('orders')->where('product_id',$request->deleteprod->delete();
        DB::table('products')->where('id',$request->deleteprod->delete();
        return redirect()->back()->with('alert', 'Eliminato con successo!'); //t
    }
}
```

2. ProductsController

```
class ProductsController extends Controller
{
    function showCatalogo(){ //permette di inviare i dati dal database alla view
        $products=Product::all();
        return view('catalogo',compact('products'));
    }
}
```

3. Homecontroller

```
class HomeController extends Controller //classe home controller
{
    //la funzione ritornerà il file home.index.php che verrà usata come home page
    public function index(){ ...
    }
    //sezione dev che permette di controllare l'output dei dati trasmessi in formato json
    public function dev(Request $request){ ...
    }
    // permette l'upload di nuovi prodotti
    public function upload(Request $request){ ...
    }
    //permette di effettuare un nuovo ordine
    function new_order(Request $request){ ...
    }
    //gestisce la scheda contatti
    public function contatti(){ ...
    }
    //gestisce la scheda menu
    function menu(){ ...
    }
    //permette di inviare i dati dal database alla view
    function showCarrello(){ ...
    }
    //permette cancellazione elementi carrello
    function deleteCarrello(Request $request){ ...
    }
    function shopCarrello(){ ...
    }
}
```

- La sezione **HomeController** è la più importante dato che gestisce la maggior parte delle componenti
- Abbiamo deciso di aggiungere un controller che viene richiamata al percorso `/dev` nel quale andiamo a stampare i dati del database in formato json in modo tale che lo sviluppatore, in caso di richiesta dell'admin, può controllare l'output dei dati.

Middleware

Oltre al controller è presente il middleware che permette di ispezionare e filtrare le richieste HTTP che entrano nell'applicazione.

Nel nostro caso abbiamo creato **AdminMid**, che prende tramite il modello Auth, l'user che al momento è loggato nella sessione, e controlla se il suo campo admin true.

Richiamando questo controllo riusciamo a fare accedere solo gli utenti admin ad una certa area privata.

```
public function handle(Request $request, Closure $next)
{
    $user = Auth::user();
    if($user->admin == '1'){
        return $next($request);
    } else
        return new Response(view('errors/403'));
}
```

```
//controllo se utente è admin richiamando middleware
'admin' => \App\Http\Middleware\AdminMid::class,
```

Definisco dentro Kernel.php 'admin' che richiama il middleware sopra

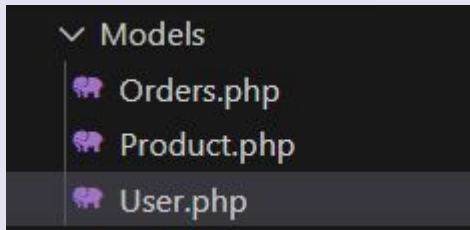


Sezione Models

In Laravel i modelli si trovano in *app/Models* e nel nostro caso sono presenti 3 modelli:

1. Orders
2. Product
3. User

I modelli sono classi che estendono la classe Model e vengono utilizzati sia per definire le relazioni tra i database (simile alle join) e sono fondamentali nell'architettura MVC. Nel caso di user è un modello preinstallato con il pacchetto Jetstream



1. Orders

```
class Orders extends Model
{
    use HasFactory;

    protected $table = 'orders';

    public function utente(){
        return $this->belongsTo(User::class); //ogni ordine appartiene ad un utente
    }

    public function prodotto(){
        return $this->belongsTo(Product::class); //un prodotto può essere associato a molti ordini
    }
}
```

2. Product

```
class Product extends Model
{
    use HasFactory;
    protected $table = 'products'; //riferimento alla tabella nel database

    public function ordini() //un ordine ha più prodotti
    {
        return $this->hasMany(Orders::class);
    }
}
```

3. User

```
//relazione 1 a molti user ordini
public function ordini()
{
    return $this->hasMany(Orders::class);
}
```



Sezione View

Le views sono divise in varie sottocartelle per specificare l'ambito di visualizzazione. Nello screen a destra è possibile vedere nelle cartelle aperte, i file che sono stati costruiti da noi, e il resto delle directory chiuse che sono aggiunte della creazione di un file Laravel + pacchetto Jetstream + varie modifiche in base alle nostre esigenze.

```
views
├── admin
│   ├── modprod.blade.php
│   ├── newprod.blade.php
│   ├── orders.blade.php
│   └── userslist.blade.php
├── api
├── auth
├── errors
│   └── 403.blade.php
├── home
│   └── index.blade.php
├── layouts
│   ├── app.blade.php
│   └── guest.blade.php
├── profile
├── vendor
├── carrello.blade.php
├── catalogo.blade.php
├── contatti.blade.php
├── dashboard.blade.php
├── dev.blade.php
├── menu.blade.php
├── navigation-menu.blade.php
├── policy.blade.php
├── terms.blade.php
└── welcome.blade.php
```

Nelle prossime slide verrà mostrato, in ordine di lettura, il contenuto e la funzione delle nostre view.

Admin

Nella sezione admin abbiamo le aree con accesso riservato. Admin può entrare in una pagina tramite la sezione dashboard.

Tra le azioni possibili, vediamo in particolare userslist.php.

Viene visualizzata la lista di utenti, con la differenza tra admin o utente normale.

Tramite dei bottoni è possibile cancellare utenti e promuovere di ruolo.

```
@foreach ($users as $user)
<tr>
  <th scope="row">{{$user['id']}}</th>
  <td>{{$user['name']}}</td>
  <td>{{$user['email']}}</td>
  <td>{{$user['password']}}</td>
  <td>{{$user['created_at']}}</td>
  <td>@if ($user['admin'])
    Si
  @else
    No
  @endif
</td>
<td>
  @if (!$user['admin'])
    <form action="{{url('deleteuser')}}" method="POST" enctype="multipart/form-data">
      @csrf
      <button type="submit" class="btn btn-outline-danger" name="deleteuser" value="{{$user['id']}}"
        onclick="return confirm('Sicuro di eliminare utente, eliminerai pure tutti i suoi ordini nel carrello');">
        Cancella</button>
    </form>
  @endif
</td>
<td>
  @if (!$user['admin'])
    <form action="{{url('promoteuser')}}" method="POST" enctype="multipart/form-data">
      @csrf
      <button type="submit" class="btn btn-outline-warning" name="promoteuser" value="{{$user['id']}}"
        onclick="return confirm('Sicuro di promuovere utente ad amministratore?');">
        Promuovi</button>
    </form>
  @endif
</td>
</tr>
@endforeach
```

Errors

Questa pagina viene visualizzata nel caso in cui si voglia accedere dall'url ad una route per cui non si hanno i permessi.

Come è possibile vedere dal codice html verrà visualizzata una pagina di errore 403 dove viene spiegato che non è possibile accedere a quell'area.

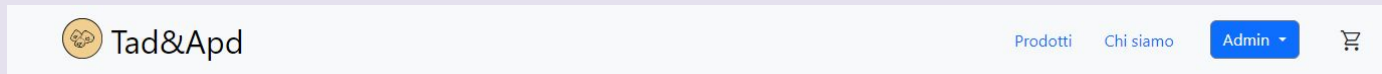
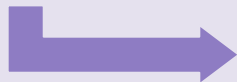
```
<body>
  <div class="wrapper">
    <div class="error-spacer"></div>
    <div role="main" class="main">
      <?php $messages = array('We need a map.', 'I think we\'re lost.', 'We took a wrong turn.');
```

Layout

In layouts sono presenti
app.blade.php e **guest.blade.php**.

App produce la view che viene visualizzata dagli admin e che consente di andare nella dashboard attraverso la navbar.

Guest gestisce la navbar degli utenti dell'applicazione che attraverso il tag `<x-guest-layout>` permette di riportarlo nelle varie sezioni della pagina



Home

Questa è l'**homepage** cioè la prima view visibile all'avvio dell'applicazione.

Sono stati utilizzati: la **navbar** precedentemente mostrata + **slideshow**, **bottoni** e vari stili utilizzando css e bootstrap per rendere il sito più simile ai vari siti web che si possono trovare in rete.

Da lì è possibile registrarsi/loggarsi e entrare nelle varie sezioni per poter acquistare prodotti.



Catalogo

Questa sezione permette di vedere i prodotti disponibili nella pagina Prodotti. Usando **@foreach** e i tag Blade **'{{{ }}** verranno richiamati gli elementi contenuti all'interno del database **Products** permettendo così di vedere i prodotti e di poter decidere quantità e se aggiungerli al carrello

```
@foreach($products as $product)
<!-- form che passa i dati al controller HomeController tramite post -->
<div class="col-4 ">
  <div class="coloreDiv">
    <form action="{{url('new_order')}}" method="POST" enctype="multipart/form-data">
      @csrf
      <!-- serve per passare id del prodotto alla query nel controller -->
      <input type="hidden" name="product_id" value="{{ $product['id'] }}">
      <br>
      
      <label for="" class="boldi fs-4" style="margin-left: 10px;">
        {{ $product['nome_prodotto'] }}
      </label>

      <br>
      <label for="" style="margin-left: 10px;">{{ $product['descrizione'] }}</label>

      <br>
      <label for="" class="fs-3" style="margin-left: 10px;">
        € {{ $product['prezzo'] }}
      </label>
      <div class="row center">
        <div class="col-md-3">
          <select name="quantity" class="form-select ">
            <option value="1" selected>1</option>
            <!-- for che va fino a 10 per il numero prodotti -->
            @for ($i=2 ; $i<=10 ; $i++) <option value="{{ $i }}">{{ $i }}</option>
          @endfor
        </div>
      </div>
    </form>
  </div>
</div>
</foreach>
```


Carrello

Nel carrello sarà possibile rivedere il riepilogo dei prodotti selezionati, il totale e ci sarà la possibilità di decidere se cancellare alcuni prodotti o di acquistare e quindi di aggiungere un ordine nel database. Ovviamente è presente un controllo che non permette ai non loggati di non poter controllare il carrello.

```
<!-- //tabella contenente la lista di prodotti aggiunti al carrello -->
<table class="table">...
</table>
<label class="input-group-text rounded">

    <!-- Stampa il totale aggiornato -->
    @php
    echo "Totale: $totale €";
    @endphp
</label>
<form action="{{url('shop')}}" method="POST" enctype="multipart/form-data">
    @csrf
    <button type="submit" class="btn btn-outline-success">Acquista</button>
</form>
@else
<p class="fs-2 centerr">Il tuo carrello è vuoto!</p>
@endif
@else
<p class="fs-2 centerr">Non hai effettuato il log-in per accedere al tuo carrello</p>
```

Dashboard

In questa sezione ha accesso soltanto l'admin e può svolgere diverse operazioni di modifica sia dei prodotti che degli utenti registrati (non vedendo le informazioni riservate). La pagina contiene soltanto una tabella dove ogni elemento è cliccabile e porta alla pagina corrispondente.

```

    <a href="admin/userslist">
      <p class="fs-4" style="text-align: center;">Lista utenti</p>
      
    </a>
  </td>
  <td>
    <a href="admin/modprod">
      <p class="fs-4" style="text-align: center;">Modifica prodotto:</p>
      
    </a>
  </td>
</tr>
<tr>
  <td>
    <a href="admin/newprod">
      <p class="fs-4" style="text-align: center;">Nuovo prodotto</p>
      
    </a>
  </td>

```

Dev

La sezione dev è stata costruita unicamente per lo **sviluppatore** che deve controllare, nel caso di una manutenzione o problemi, le **chiamate API** e i **dati** che vengono inviati e ricevute. È possibile accedere all'indirizzo *localhost:8000/dev* e il formato dei dati viene mostrato in json come possibile vedere nelle immagini sottostanti:

```
{
  "prodotti": [
    {
      "id": 1,
      "created_at": "2023-06-18 08:35:38",
      "updated_at": "2023-06-18 08:35:38",
      "nome_prodotto": "Tortellini alla Dybala",
      "descrizione": "Pacco da 500g, leggendari tortellini di Paolo Dybala",
      "prezzo": 20,
      "image": "tort_dybala.jpg"
    },
    {
      "id": 2,
      "created_at": "2023-06-18 08:35:38",
      "updated_at": "2023-06-18 08:35:38",
      "nome_prodotto": "Tortellini Barilla",
      "descrizione": "Pacco da 250g, collaborazione con barilla ad un prezzo wow",
      "prezzo": 12,
      "image": "tort_barilla.jpg"
    },
    {
      "id": 3,
      "created_at": "2023-06-18 08:35:38",
      "updated_at": "2023-06-18 08:35:38",
      "nome_prodotto": "Tortellini Bing Chilling",
      "descrizione": "Dalla ricetta del maestro John Xina, grande chef cinese",
      "prezzo": 15,
      "image": "cina.jpg"
    },
    {
      "id": 4,
      "created_at": "2023-06-18 08:35:38",
      "updated_at": "2023-06-18 08:35:38",
      "nome_prodotto": "Tortellini Parma",
      "descrizione": "Con prosciutto di Parma DOP",
      "prezzo": 17,
      "image": "parma.jpg"
    },
    {
      "id": 5,
      "created_at": "2023-06-18 08:35:38",
      "updated_at": "2023-06-18 08:35:38",
      "nome_prodotto": "Tortellini Rana ricotta",
      "descrizione": "Collaborazione con mio amico Rana, speciali alla ricotta",
      "prezzo": 14,
      "image": "ricotta.jpg"
    }
  ],
}
```

```
"ordini": [],
"utenti": [
  {
    "id": 1,
    "name": "Admin",
    "email": "admin@tad.apd.com",
    "email_verified_at": "2023-06-18 08:35:38",
    "password": "$2y$10$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi",
    "two_factor_secret": null,
    "two_factor_recovery_codes": null,
    "two_factor_confirmed_at": null,
    "remember_token": "zuXl9itfdw6OyqzmKXY5U3qvcwYpS6D93NWXUJ3VZmGyC39r7rAx5bTeHv1r5",
    "current_team_id": null,
    "profile_photo_path": null,
    "created_at": "2023-06-18 08:35:38",
    "updated_at": "2023-06-18 08:35:38",
    "admin": 1
  },
  {
    "id": 2,
    "name": "massimo",
    "email": "massimo@visconti.com",
    "email_verified_at": "2023-06-18 08:35:38",
    "password": "$2y$10$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi",
    "two_factor_secret": null,
    "two_factor_recovery_codes": null,
    "two_factor_confirmed_at": null,
    "remember_token": "rfpjdSs4X",
    "current_team_id": null,
    "profile_photo_path": null,
    "created_at": "2023-06-18 08:35:38",
    "updated_at": "2023-06-18 08:35:38",
    "admin": 0
  }
]
```



02

Gestione database



Sezione Database → Migrations


In Laravel la struttura del database viene definita su **migration** che si trova in **database\migrations**.

Nel nostro caso molte migrations sono state aggiunte con Jetstream ma noi ci siamo focalizzati su 3 principali

1. create_user_table
2. create_products
3. create_orders

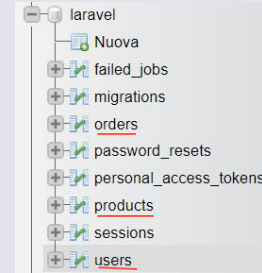
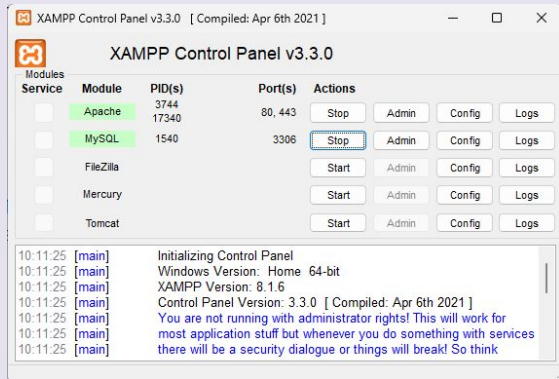
Andando a modificare il campo **DB_DATABASE** del `.env` possiamo ricollegarci ad un database che nel nostro caso risiedeva su xampp. Eseguendo il comando `php artisan migrate` il database verrà creato automaticamente seguendo le leggi definite su migration:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```



Sezione Database → Mysql

Come già detto in precedenza è stato utilizzato Xampp nel quale usiamo mysql attraverso la porta 80 del localhost:



User (con password criptata):

id	name	email	email_verified_at	password
1	Admin	admin@tad.apd.com	2023-06-18 08:35:38	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro9IIc/.c
2	massimo	massimo@visconti.com	2023-06-18 08:35:38	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro9IIc/.c

Product:

id	created_at	updated_at	nome_prodotto	descrizione	prezzo	image
1	2023-06-18 08:35:38	2023-06-18 08:35:38	Tortellini alla Dybala	Pacco da 500g, leggendari tortellini di Paolino Dybala	20	tort_dybala.jpg
2	2023-06-18 08:35:38	2023-06-18 08:35:38	Tortellini Barilla	Pacco da 250g, collaborazione con barilla ad un prezzo wow	12	tort_barilla.jpg
3	2023-06-18 08:35:38	2023-06-18 08:35:38	Tortellini Bing Chilling	Dalla ricetta del maestro John Xina, grande chef cinese	15	cina.jpg
4	2023-06-18 08:35:38	2023-06-18 08:35:38	Tortellini Parma	Con prosciutto di Parma DOP	17	parma.jpg
5	2023-06-18 08:35:38	2023-06-18 08:35:38	Tortellini Rana ricotta	Collaborazione con mio amico Rana, speciali alla ricotta	14	ricotta.jpg
6	2023-06-18 08:35:38	2023-06-18 08:35:38	Tortellini Gluten free	Pensiamo proprio a tutto, per i nostri amici celiaci	11	gluten.jpg

Sezione Database → Seeders

All'interno di Laravel è possibile immettere dati all'interno del database, popolandolo a proprio piacimento prima dell'avvio del web server.

Nel nostro caso abbiamo inserito sia dei prodotti, sia alcuni utenti come un admin e un user predefiniti.

```
public function run()
{
    Product::create([
        'nome_prodotto' => 'Tortellini alla Dybala',
        'descrizione' => 'Pacco da 500g, leggendari tortellini di Pauolo Dybala',
        'prezzo' => 20,
        'image' => 'tort_dybala.jpg'
    ]);
    Product::create([
        'nome_prodotto' => 'Tortellini Barilla',
        'descrizione' => 'Pacco da 250g, collaborazione con barilla ad un prezzo wow',
        'prezzo' => 12.10,
        'image' => 'tort_barilla.jpg'
    ]);
    Product::create([
        'nome_prodotto' => 'Tortellini Bing Chilling',
        'descrizione' => 'Dalla ricetta del maestro John Xina, grande chef cinese',
        'prezzo' => 15,
        'image' => 'cina.jpg'
    ]);
}
```

TortDyb.php

Tramite DatabaseSeeder.php richiamo le classi

```
public function run()
{
    // per fare il seed devo fare php artisan migrate:fresh --seed
    $this->call(Admin::class);
    $this->call(TortDyb::class); //aggiunge il tortellino alla dybala
}
```

Il comando per eseguire il seeder è il seguente:

php artisan migrate:fresh --seed





03

Routing




Sezione routing

In Laravel il routing ricopre un ruolo fondamentale specialmente per quanto riguarda uno degli obiettivi principali di questo progetto.

Utilizzando **web.php**, infatti, abbiamo registrato le varie **web routes** e le **API routes** per la nostra applicazione. Queste routes vengono caricate dal **RouteServiceProvider**. In questo modo sono state possibili, attraverso delle richieste **HTTP**, le varie richieste web e le richieste API in formato JSON. per effettuare queste operazioni sono state necessarie delle direttive di importazione, nel nostro caso:

```
use App\Http\Controllers\AdminController;  
use Illuminate\Support\Facades\Route;  
use App\Http\Controllers\ProductsController;  
use App\Http\Controllers\HomeController; //route Home controller  
use Laravel\Fortify\RoutePath;
```



Il ruolo fondamentale delle **routes** è quello di definire il path in cui i dati o le views verranno visualizzate. Nel nostro caso è necessario utilizzare i controller per seguire la struttura di laravel e il secondo parametro sarà appunto la funzione che andrà a definire ciò che sarà mostrato nella route.

```
//in questo modo nell' / verrà richiamato nel controller home controller la funzione index
route::get('/', [HomeController::class, 'index']);

route::post('/upload_post', [HomeController::class, 'upload']);

route::post('/new_order', [HomeController::class, 'new_order']);

route::get('contatti', [HomeController::class, 'contatti']);

Route::get('/carrello', [HomeController::class, 'showCarrello']);
//sezione che permette di visualizzare in formato JSON come vengono visualizzati i dati passati
Route::get('/dev', [HomeController::class, 'dev']);

route::post('/delete', [HomeController::class, 'deleteCarrello']);

route::post('/deleteuser', [AdminController::class, 'deleteUser']);

route::post('/deleteprod', [AdminController::class, 'deleteProd']);

route::post('shop', [HomeController::class, 'shopCarrello']);

Route::get('/catalogo', [ProductsController::class, 'showCatalogo']);
```



Private Area

Tramite la funzione `group` posso andare a definire più chiamate di routing, in questo caso prima eseguo il middleware per identificare se l'utente in sessione è admin, se ci viene restituito il Closure **next** le seguenti aree sono accessibili all'utente, sennò viene visualizzata una pagina di errore

```
//controlli per instradare solo gli admin in certe aree
Route::group(['middleware' => ['auth','admin']], function(){
    Route::get('/dashboard',function(){
        return view('dashboard');
    }->name('dashboard');
    Route::get('/admin/newprod',function(){
        return view('admin/newprod');
    }->name('admin/newprod');
    Route::get('/admin/userslist',[AdminController::class,'list']->name('admin/userslist');
    Route::get('/admin/modprod',[AdminController::class,'modProdView']->name('admin/modprod');
    Route::get('/admin/orders',function(){
        return view('admin/orders');
    }->name('admin/orders');
    Route::get('/dev',[HomeController::class,'dev']->name('dev'));
});
```



04

Demo e riferimenti



Demo

