PROGRAMMAZIONE WEB

PROGRAMMAZIONE AGILE CON LARAVEL

Prof. Ada Bagozi
ada.bagozi@unibs.it

I pilastri dello sviluppo agile

- ✓ Limitare il più possibile la scrittura di codice ripetitivo
  - ✓ e.g., utilizzo di ORM (Object-Relational Mapping)
  - ✓ e.g., generazione automatica di codice ripetitivo
- ✓ Sviluppare facendo uso di Design Pattern
- ✓ Creare gruppi di lavoro in cui fin dall'inizio gli sviluppatori SW lavorano fianco a fianco con gli end-user
  - ✓ Focus sulla prototipazione rapida

Cos'è un framework di sviluppo?

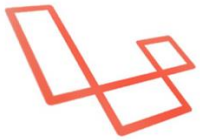
- ✓ Un ***framework*** è una struttura logica a supporto dello sviluppo software
- ✓ Quasi sempre un framework implementa un particolare ***design pattern***
- ✓ Composto da una serie di librerie di codice in uno specifico linguaggio di programmazione
- ✓ Talvolta accompagnato da strumenti di sviluppo, come IDE o debugger
- ✓ Un framework presuppone l'adozione di una specifica ***metodologia di programmazione***

Laravel

Un ***web application framework*** open source e free, creato da Taylor Otwell

Orientato allo sviluppo agile di applicazioni Web in accordo con il ***design pattern MVC***

A partire da Marzo 2015, uno dei più importanti framework PHP (altri esempi, Symfony2, Nette, CodeIgniter, Yii2)

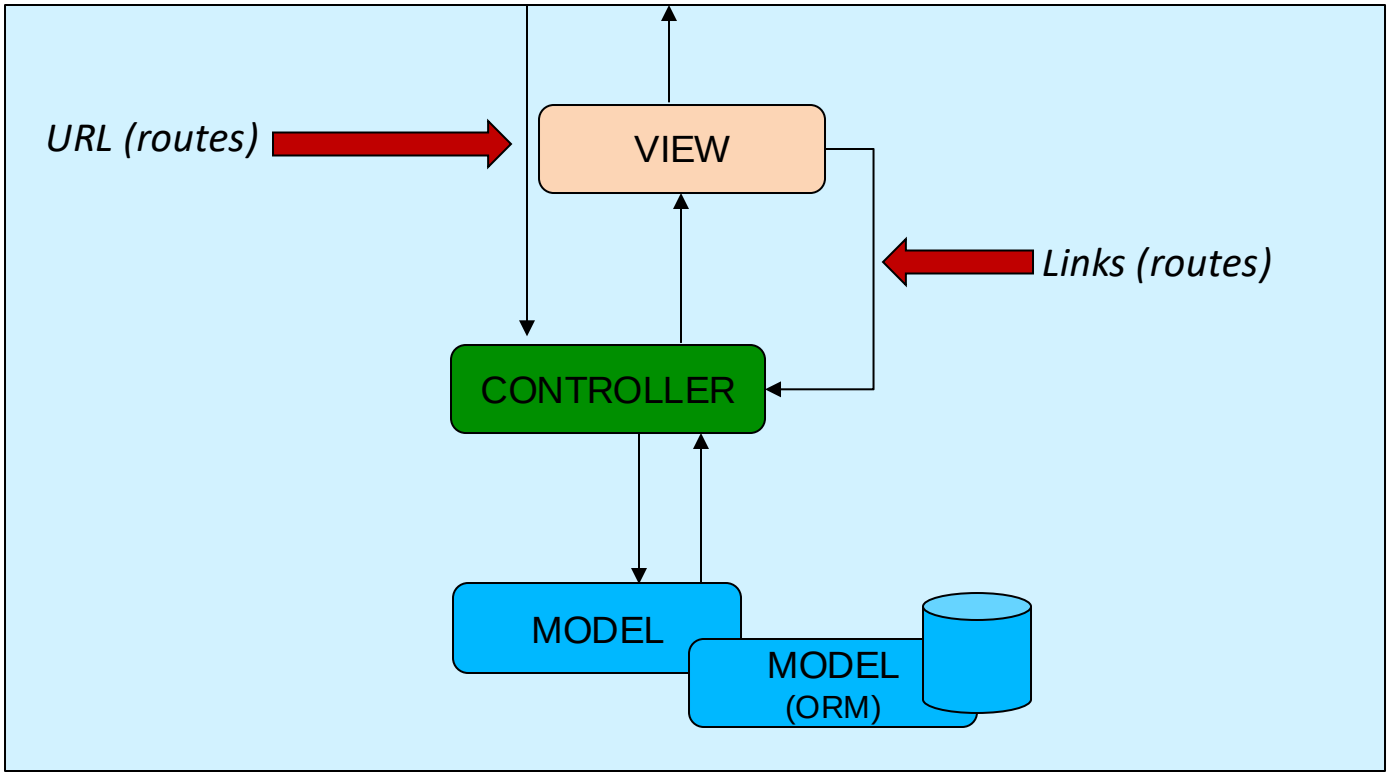


MVC design pattern (I)

Model-**V**iew-**C**ontroller (MVC) è un design pattern architetturale che semplifica lo sviluppo di applicazioni separando i componenti software in

- ✓ ***Model***: componenti che forniscono i metodi per accedere ed elaborare i dati utili all'applicazione
- ✓ ***View***: componenti che visualizzano i dati contenuti nel Model e si occupano dell'interazione con utenti e agenti
- ✓ ***Controller***: componenti che ricevono i comandi dall'utente (attraverso le View) e li attuano invocando gli altri componenti (Model)

MVC design pattern (II)



Laravel – Requisiti

Versione corrente 12 (Febbraio 2025)

Il framework **Laravel** presenta alcuni (non molti) requisiti (https://laravel.com/); per la versione 12:

- ✓ PHP ≥ 8.2 (`php -v` *oppure* `php --version`)
- ✓ Alcune estensioni PHP

Laravel utilizza **Composer** (https://getcomposer.org/) per la gestione delle dipendenze. Di conseguenza, prima di utilizzare Laravel, è necessario assicurarsi che Composer sia installato sulla macchina

Laravel – Versioni

Versione	Data di rilascio	Versione PHP	Bug fix fino al	Security fix fino al
1.0	Giugno 2011			
2.0	Settembre 2011			
3.0	22 Febbraio 2012			
3.1	27 Marzo 2012			
3.2	22 Maggio 2012			

...

7	3 Marzo 2020[13]	≥ 7.2.5	6 Ottobre 2020	3 Marzo 2021
8	8 Settembre 2020[14]	≥ 7.3	26 Luglio 2022	24 Gennaio 2023
9	8 Febbraio 2022[15]	≥ 8.0[16]	8 Agosto 2024	6 Febbraio 2024
10	7 Febbraio 2023[15]	≥ 8.1[17]	6 Agosto 2024	4 Febbraio 2025
11	12 Marzo 2024[18]	≥ 8.2[19]	5 Agosto 2025	3 Febbraio 2026
12[20]	Q1 2025	≥ 8.2	Q3 2026	Q1 2027

Legenda	Vecchia versione, nessun supporto	Vecchia versione, ancora supportata	**Versione corrente**	Versione futura

Creazione di un nuovo progetto

Tramite l'utilizzo di **Composer**:

- ✓ creando il progetto direttamente, utilizzando le librerie Laravel scaricate dalla rete

```
composer create-project laravel/laravel:^12.0 ${applicationName}
```

Per scaricare e aggiornare tutte le dipendenze:

```
composer update
```

Per far partire l'applicazione:

```
php artisan serve
```

start a local server on 8000 port

Struttura delle directory (I)

`app/` - contiene la business logic dell'applicazione, raccoglie tra gli altri i file relativi ai controller (**`app/http/Controllers/`**) e le classi del Model (**`app/Models/`**)

`routes/` - contiene i file relativi alle rotte, ovvero la gestione degli URL e i mapping tra questi e le funzionalità offerte dall'applicazione

`config/` - raccoglie i file contenenti le impostazioni di configurazione dell'intera applicazione

`public/` - contiene tutte le risorse statiche pubbliche (script JS, CSS, immagini, font, etc.) del progetto

Struttura delle directory (II)

`resources/` - raccoglie, tra gli altri, i file per la generazione delle viste (**`resources/views/`**)

`vendor/` - contiene, come da specifiche, tutte le dipendenze

`database/` - contiene tutti i file relativi alla gestione del database, tra cui gli script per la generazione delle tabelle (**`database/migrations/`**), gli script per il popolamento delle tabelle (**`database/seeders/`**), le factory per creare massivamente contenuti di test (**`database/factories/`**)

Il file **`.env`** contiene variabili di configurazione importanti per il funzionamento dell'applicazione

Il file .env


```
APP_ENV=local
APP_KEY=base64:qgfgaLl+6zHGy/c0ifodVDS/DJ6Ew43tF0uO0owh+V8=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

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

Definizione delle rotte (I)

Meccanismo del *routing* – Gestione degli URL dell'applicazione Web e mapping tra URL e i metodi dei controller dell'applicazione, tramite i quali accedere alle funzionalità dell'applicazione

Meccanismo incentivato dalle politiche SEO moderne

`http://www.website.it/script.php?id=123`

`http://www.website.it/script/123`

Definizione delle rotte (II)

Le rotte sono definite nel file **`routes/web.php`** utilizzando il *facade* **`Route`**

- ✓ metodi corrispondenti ai metodi HTTP **`get`**, **`post`**, **`put`**, **`delete`**, …
- ✓ metodi **`match`** o **`any`**
- ✓ rotte parametriche e vincoli
- ✓ gruppi di rotte

Esempi di rotte


```php
Route::get('/say-hello', function(){
    return "Hello hello!";
});

Route::match(['get', 'post'], '/say-hello-multiple', function() {
    return 'This is a GET or POST request';
});

Route::any('/any', function() {
    return 'I can respond to any http method';
});

Route::get('/post/{id}', function($id) {
    return "You requested post with ID = " . $id;
});

Route::get('/optPost/{id?}', function($id = 1) {
    return "You requested post with ID = " . $id;
});

Route::group(['prefix' => 'admin'], function () {
    Route::get('users', function ()    {
       // Matches The "/admin/users" URL
        return "Bye bye users!";
    });

    Route::get('clients', function ()    {
       // Matches The "/admin/users" URL
        return "Bye bye clients!";
    });
});
```

Creazione di un controller

- Classi PHP i cui metodi sono anche identificati con il termine di *azioni*
- Vanno posizionati nella cartella **`App/Http/Controllers`** e sono sotto-classi di **`Illuminate\Routing\Controller`**
- Sono generati tramite il comando `php artisan make:controller ${nome_controller}`

```php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class FrontController extends Controller
{
    public function getHome()
    {
        return view('index');
    }
}
```

Metodi HTTP e rotte RESTful

La corrispondenza tra metodi HTTP e azioni CRUD è stata nel corso degli anni standardizzata, facilitando ulteriormente la stesura di codice

- **`GET`** – lettura di una risorsa (se viene specificato l'ID) oppure di un insieme di risorse
- **`POST`** – scrittura di una risorsa (in combinazione con il metodo **`GET`** per la visualizzazione della form di inserimento dati)
- **`PUT/PATCH`** – modifica di una risorsa di cui viene specificato l'ID (in combinazione con il metodo **`GET`** per la visualizzazione della form di inserimento dati)
- **`DELETE`** – cancellazione di una risorsa (di cui viene specificato l'ID)

Routing RESTful (I)


```
Route::resource('photo', PhotoController::class);
```

Verb	Path	Action	Route Name
GET	/photo	index	photo.index
GET	/photo/create	create	photo.create
POST	/photo	store	photo.store
GET	/photo/{photo}	show	photo.show
GET	/photo/{photo}/edit	edit	photo.edit
PUT/PATCH	/photo/{photo}	update	photo.update
DELETE	/photo/{photo}	destroy	photo.destroy

```
php artisan make:controller PhotoController --resource
```

Rotte per l'esempio di riferimento


```php
/*
Route::get('/', function () {
    return view('index');
})->name('home');
*/

Route::get('/', [FrontController::class, 'getHome'])->name('home');

Route::resource('book', BookController::class);
// Placeholder for:
// - Route::get('/book', [BookController::class, 'index'])->name('book.index'); // Display the list of books
// - Route::get('/book/create', [BookController::class, 'create'])->name('book.create'); // Display the creation form
// - Route::post('/book', [BookController::class, 'store'])->name('book.store'); // Save the book in DB
// - Route::get('/book/{id}', [BookController::class, 'show'])->name('book.show'); // Display the a single book
// - Route::get('/book/{id}/edit', [BookController::class, 'edit'])->name('book.edit'); // Display the edit form
// - Route::put('/book/{id}', [BookController::class, 'update'])->name('book.update'); // Update the book in DB
// - Route::delete('/book/{id}', [BookController::class, 'destroy'])->name('book.destroy'); // Delete the book from ID

Route::resource('author', AuthorController::class);
```

Le View

- ✓ Il componente che è responsabile della visualizzazione in XHTML delle informazioni conservate nella nostra applicazione
- ✓ Laravel supporta la definizione di viste come semplici file PHP (estensione **`.php`**) o Blade (estensione **`blade.php`**), un template system molto diffuso
- ✓ Le viste sono invocate dall'interno dei metodi dei controller tramite l'helper **`View`**
- ✓ Le viste sono file posizionati all'interno della cartella **`resources/views`** e sue sottocartelle
- ✓ Per esempio, una vista nella cartella **`resource/views/author/index`** va referenziata con la notazione **`author.index`**
- ✓ Il passaggio di parametri ad una **`View`** viene effettuato in due modi:
  - ✓ tramite il secondo parametro (array associativo) dell'helper **`View`**
  - ✓ tramite il metodo **`with(nome_parametro, valore_parametro)`** dell'helper stesso, invocabile più volte

View e layout

- ✓ Blade, in quanto template system, permette lo sviluppo di *layout*, ovvero file con estensione **`blade.php`** per organizzare la struttura visuale delle pagine
- ✓ Un layout definisce un *template*, che può essere istanziato in una vista
- ✓ Nel layout, è possibile fare uso di *placeholder* attraverso la direttiva **`@yield('nome_placeholder')`**
- ✓ Ogni vista che istanzia un layout deve riportare all'inizio la dichiarazione **`@extends('nome_layout')`** (i nomi e l'organizzazione in cartelle dei layout seguono la stessa logica delle viste)
- ✓ Nella vista che istanzia un layout, un placeholder viene sostituito con codice HTML/PHP attraverso la direttiva **`@section('nome_placeholder')`**

```
@section('active_MyLibrary','active')

@section('breadcrumb')
<li class="breadcrumb-item" aria-current="page"><a href="{{ route('home') }}">Home</a></li>
<li class="breadcrumb-item active" aria-current="page"><a href="{{ route('book.index') }}">Library</a></li>
<li class="breadcrumb-item active" aria-current="page"><a href="{{ route('book.index') }}">Books</a></li>
<li class="breadcrumb-item active" aria-current="page">Delete book</li>
@endsection
```

Oggetto Request

L'oggetto `request` è passato come input ai metodi dei controller (azioni)

Si tratta di un'istanza della classe `Illuminate\Http\Request`

Da questo oggetto è possibile estrarre:

- Gli input dei campi di una form
- Il metodo HTTP che si sta utilizzando
- Le informazioni sugli headers del pacchetto HTTP
- I cookies scambiati tra il client e il server

Oggetto Request e form

Estrarre i valori dei campi di una form tramite l'oggetto **`request`**

```
$name = $request->input('name');
$email = $request->input('email', 'mail@domain.com');
$allParameters = $request->all();
```

Altri input dei metodi del controller sono gli eventuali parametri delle rotte parametriche

```
public function update(Request $request, $id) {
    $dl = new DataLayer();
    $dl->editAuthor($id, $request->input('firstName'), $request->input('lastName'));
    return Redirect::to(route('author.index'));
}
```

Oggetto Request e metodi HTTP

L'identificazione del metodo HTTP tramite il quale arriva la richiesta può rivelarsi utile per capire l'origine della richiesta stessa

- **`GET`** – form, link/bottoni, barra degli indirizzi del browser
- **`POST`**/**`PUT`**/**`DELETE`** – tramite form

Utile per prevenire accessi errati alle pagine e alle risorse del sito

Per estrarre il metodo HTTP tramite l'oggetto **`request`**

```
if($request->isMethod('post')) {
        ...
}
```

Header HTTP nella richiesta

- ✓ Tramite il metodo `$request->header('')` si possono visualizzare gli headers inviati nella richiesta

```
$value = $request->header('X-Header-Name');

$value = $request->header('X-Header-Name', 'default');
```

Elenca contenuto directory

[Host] = localhost
[Connection] = keep-alive
[Accept] = text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
[User-Agent] = Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.63 Safari/537.31
[Referer] = http://localhost/prova/headers/
[Accept-Encoding] = gzip,deflate,sdch
[Accept-Language] = it-IT,it;q=0.8,en-US;q=0.6,en;q=0.4
[Accept-Charset] = ISO-8859-1,utf-8;q=0.7,*;q=0.3
[Cookie] = PHPSESSID=r7vuie4l3mi4t75f45001c5m23; mail=osor@osor.it

Oggetto Response (I)

L'oggetto **`response`** è la controparte dell'oggetto **`request`** ed è configurato all'interno dei metodi dei controller (implicitamente quando si genera una view)

Si tratta di un'istanza della classe **`Illuminate\Http\Response`**

```
Route::get('/', function () {
    return 'Hello World';
});
```

```
use Illuminate\Http\Response;
Route::get(/', function () {
    return new Response('Hello World');
});
```

Oggetto Response (II)

L'oggetto **`response`** è la controparte dell'oggetto **`request`** ed è configurato all'interno dei metodi dei controller (implicitamente quando si genera una view)

Si tratta di un'istanza della classe **`Illuminate\Http\Response`**

```
Route::get('/', function () {
    return 'Hello World';
});
```

```
Route::get('/', function () {
    return response('Hello World');
});
```

Oggetto Response (III)

Tramite l'oggetto **`response`** è possibile modificare gli header del pacchetto HTTP

```
return response('Hello World)
   ->header('Content-Type', 'text/plain');
```

```
return response()->view('view.name')
   ->header('Content-Type', 'text/plain');
```

Redirect (I)

Una delle azioni che è possibile compiere grazie alla modifica dell'header nella risposta HTTP:

- verso una rotta identificata tramite un nome

```php
return Redirect::to(route('book.index'));
```

```php
return redirect()->route('book.index');
```

```php
return redirect()->route('book.edit', ['book' => $book->id]);
```

- verso la pagina precedente

```php
return back()->withInput();
```

Redirect (II)

- verso un'azione specifica di un controller

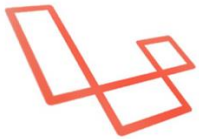
```
return redirect()->action([FrontController::class,'getHome']);
```

- verso un collegamento esterno

```
return redirect()->away('https://www.gazzetta.it/');
```

Link utili

La documentazione ufficiale di Laravel:
https://laravel.com/docs/12.x



laravel

PROGRAMMAZIONE WEB

PROGRAMMAZIONE AGILE CON LARAVEL

Prof. Ada Bagozi
ada.bagozi@unibs.it