Laravel: un framework MVC per PHP

Prime nozioni, installazione, tool

Cos'è un framework

- https://en.wikipedia.org/wiki/Software_framework
- https://en.wikipedia.org/wiki/Software_framework#cite_note-1
- Da studiare: https://github.com/TendTo/Rally-Championship-Manager
- Magari si capisce l'autenticazione…

Prerequisiti

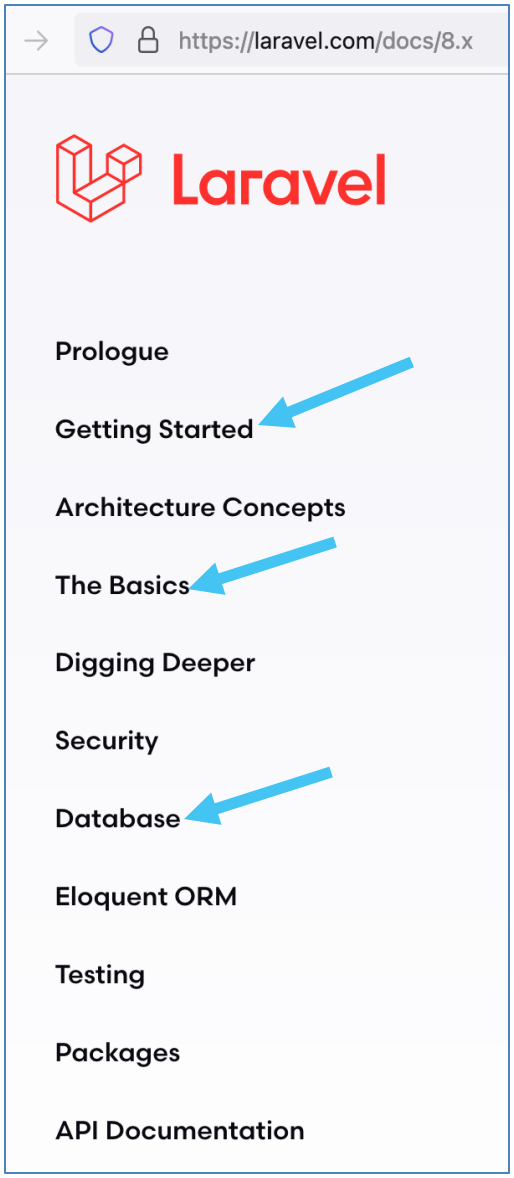
Prerequisiti: buona conoscenza di PHP, in particolare a oggetti.

Alcune risorse per un rapido ripasso di PHP:

- https://www.ntu.edu.sg/home/ehchua/programming/#php
- https://developer.hyvor.com/tutorials/php/
- https://www.w3schools.com/php/
- https://www.w3schools.com/php/php_oop_what_is.asp
- https://www.php.net/manual/language.oop5.php
- https://www.tutorialspoint.com/php/php_object_oriented.htm
- https://www.tutorialspoint.com/php

Queste note introduttive sono piuttosto complete, anzi si noti che: eventuali slide (o parti di esse) con questo sfondo azzurro non servono per lo studio, ma solo per approfondimenti.

Laravel: risorse "ufficiali"



- https://laravel.com/docs/9.x (2022) è il riferimento di base per Laravel 9.x
- Si tengano presenti la dashboard laterale del sito, riprodotta qui a sinistra e, in particolare, per iniziare, i *topic* qui sotto:

Getting Started
- Installation
- Configuration
- Directory Structure
- Starter Kits
- Deployment

The Basics
- Routing
- Middleware
- CSRF Protection
- Controllers
- Requests
- Responses
- Views
- Blade Templates
- URL Generation

Database
- Getting Started
- Query Builder
- Pagination
- Migrations

Quickstart

- Obsoleto, ma efficace, dal portale ufficiale: https://laravel.com/docs/4.2/quick

Laravel: installazione manuale

Espandendo https://laravel.com/docs/installation, descriveremo l'installazione manuale di un ambiente di sviluppo/esecuzione per Laravel

Iniziamo classificando i **componenti** necessari:

- sistema operativo: Windows o Unix (negli esempi vedremo per lo più in azione Unix/Linux)
- PHP Engine (la CLI php)
- alcuni moduli (librerie binarie) per PHP
- alcuni pacchetti (package) PHP
- *composer*, gestore delle dipendenze tra pacchetti PHP

Ma cosa denota il termine *"Laravel"?*

- **Laravel** è un **framework**, basato su un insieme di package PHP, all'interno del quale sviluppare/eseguire applicazioni e API Web
- **Laravel** è anche (il nome di) un **tool** di sviluppo, scritto in PHP, capace di generare applicazioni per il framework Laravel

Laravel: requisiti per server / PC di sviluppo

- Per il server: PHP >= 7.3 (per Laravel 8.x) e le *PHP extensions* (moduli):

BCMath PHP Extension	JSON PHP Extension	PDO PHP Extension (es. per *mysql*)
Ctype PHP Extension	Mbstring PHP Extension	Tokenizer PHP Extension
Fileinfo PHP extension	OpenSSL PHP Extension	XML PHP Extension

(*v.* [*https://laravel.com/docs/deployment#server-requirements*](https://laravel.com/docs/deployment#server-requirements))

- Per il PC di sviluppo: **è indispensabile** anche il modulo *php-curl*
  Ciò è poco documentato, ma essenziale; infatti il tool *composer* (v. oltre) ci avverte:

```
$ composer diagnose | grep -i curl
cURL version: missing, using php streams fallback, which reduces performance
```

- *php -m* elenca le *estensioni* (o *moduli* – librerie binarie) attivate nella vs. installazione PHP; ecco la situazione "base" con Ubuntu 18.04 LTS:

```
$ php -m | grep -Ei '(bcmath|ctype|curl|fileinfo|json|mbstring|openssl|pdo_|tokenizer|^xml$)' \
 | tr '\n' ' '
ctype fileinfo json openssl tokenizer
```

- Come si vede, ne mancano alcuni, ma Ubuntu **non** ha un pacchetto *.deb* "Laravel", che "tiri giù" anche i pacchetti *deb* `php-bcmath` `php-mbstring` `php-mysql` `php-xml` `php-curl`, cioè quelli con i moduli PHP richiesti

Laravel: requisiti per l'engine PHP / 2

Peer una distro Ubuntu, occorre quindi installare a mano i pacchetti *deb* con i predetti moduli PHP mancanti:

```
$ sudo apt install php-bcmath php-mbstring php-mysql php-xml php-curl
```

Ora possiamo verificare di nuovo la disponibilità di tutti i moduli PHP necessari:

```
$ php -m | grep -Ei '(bcmath|ctype|fileinfo|json|mbstring|openssl|pdo|tokenizer|^xml$|curl)'
bcmath     # installata a parte, in quanto NON dipendenza di PHP su Ubuntu 18.04
ctype
curl
fileinfo
json
mbstring   # installata a parte, in quanto NON dipendenza di PHP su Ubuntu 18.04
openssl
pdo_mysql  # installata a parte, in quanto NON dipendenza di PHP su Ubuntu 18.04
tokenizer
xml        # installata a parte, in quanto NON dipendenza di PHP su Ubuntu 18.04
```

Qui è presente il modulo *pdo_mysql* per un backend *mysql*/*mariadb*

Ultima versione PHP (Ubuntu)

Ubuntu non aggiorna sempre con prontezza PHP, sicché il Laravel installato (v. oltre) potrebbe non trovare un PHP abbastanza nuovo, p.es.:

```
$ grep DISTRIB_DESCRIPTION /etc/lsb-release
DISTRIB_DESCRIPTION="Ubuntu 18.04.6 LTS"
$ php -v
PHP 7.2.24-0ubuntu0.18.04.10 (cli) (built: Oct 25 2021 17:47:59)...
```

Per Ubuntu LTS 18.04, PHP è alla v. 7.2.24, ma Laravel (nuovo) richiede v. >= 7.3:

```
$ laravel new -q my_app          # si immagini di avere installato (v. oltre) il tool laravel
In NewCommand.php line 73:
The Laravel installer requires PHP 7.3.0 or greater...                                  .
```

Per risolvere il problema e avere sempre versioni PHP fresche:

```
$ sudo add-apt-repository ppa:ondrej/php              # ondrej/php è il repository deb semi-ufficiale per PHP
$ sudo apt update
$ sudo apt upgrade php php-bcmath php-mbstring php-mysql php-xml php-curl -y # si aggiornano esplicitamente anche
...                                                                          # i pacchetti necessari, per sicurezza
```

Infine, bisogna cambiare l'engine PHP di default nella più recente disponibile:

```
$ sudo update-alternatives --config php
...                                    # dal menu proposto, si scelga PHP 8.x come nuovo default
$ php -v
PHP 8.1.2 (cli) (built: Jan 24 2022 10:42:15) (NTS) ...
```

(v. https://computingforgeeks.com/how-to-install-php-on-ubuntu/)

PHP: i progetti

- In PHP "moderno" un **package** è una *gerarchia* (albero) di directory contenenti (altre directory e) file *.php*, oltre a file di supporto (assets)
- Il package offre *funzionalità* definite da una API, cioè un gruppo di classi/metodi/... richiamabili (vedremo attraverso quali meccanismi) da altro codice PHP, tipicamente organizzato in forma di *progetto*
- Un *progetto* è un package concepito per essere eseguito come *applicazione* (Web, come *phpmyadmin*, o CLI, come "`laravel new`")
- In pratica, questo vuol dire che, se *my_proj* è la directory base del progetto *my_proj* e contiene un file *index.php*, eseguendo i comandi:

```
$ cd my_proj; php -S localhost:8080
```

  la relativa webapp *my_proj* risponderà alla URL *http://localhost:8080*
  - NB: a volte (è il caso delle app Laravel) *index.php* è in *my_proj/public*, quindi:

```
$ cd my_proj/public; php -S localhost:8080
```

- Se invece l'app è CLI, sarà un file *.php* da eseguire da riga di comando:

```
$ COMPOSER_HOME=$(composer config --global home)    # determiniamo la home di composer
$ php $COMPOSER_HOME/vendor/bin/laravel             # eseguiamo l'app (CLI) laravel (è un file PHP)
```

PHP: i package usati da un progetto

- Come fa un progetto a usare dei package? Nel caso più tipico, contiene, nella sua directory *vendor/*, le directory dei package
- Immaginiamo, p.es., che il *vendor* v1 fornisca i package *p11* e *p12* e il *vendor* v2 i package *p21* e *p22* (per ora si supponga siano stati scaricati "a mano" dal sito dei vendor)

  Se il progetto *my_proj* utilizza i package predetti, la sua directory dovrebbe presentarsi con la struttura qui a destra:

- Ovviamente, *index.php* e gli altri file del progetto dovranno avere *facilities* per fare riferimento alla API dei package (v. *namespace* e *use*) e per usarne il codice (*include/require* o, più comodo ed elegante, *autoloading*)
- I package usati da (necessari a) un progetto sono detti le sue *dipendenze*

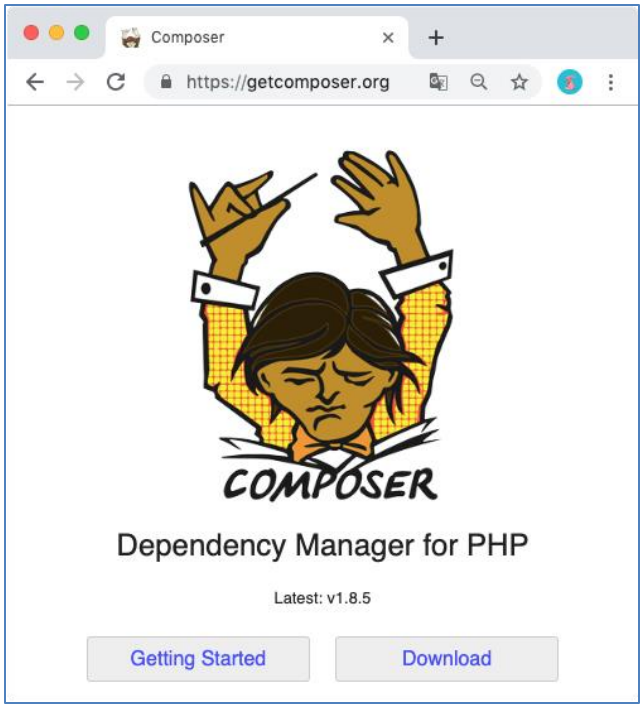
  Può essere utile, come mostrato qui a fianco, introdurre un file *json* di specifica del progetto e delle sue dipendenze, elencate nella key `require` (^*x.y* indica la versione minima)

```
{
  "name": "my_proj",
  "type": "project",
  "require": {
    "v1/p11": "^4.2",
    "v1/p12": "^1.0",
    "v2/p21": "^6.3",
    "v2/p22": "^7.0"
} }
```

- Si vedrà nel seguito come ciò consenta di automatizzare la gestione (download e update) delle dipendenze

Dipendenze PHP: il tool *composer*

- Tool (testuale) per gestire in PHP dipendenze dei progetti da pacchetti
- *composer* legge da un file *composer.json* le dipendenze del progetto PHP da certi pacchetti (anche a livello di numero di versione),
  - determina (tutti) i pacchetti necessari, li scarica e installa automaticamente



- Per default, *composer* opera a livello di singolo progetto PHP, gestisce cioè le dipendenze di quel progetto
- Installa le dipendenze (package PHP necessari al progetto) nella directory del progetto
- In alternativa, può installare i pacchetti necessari globalmente
- NB: *globalmente* significa che i pacchetti sono installati in `$HOME/.config/composer` o altra directory **dell'utente** e, cioè, "globalmente" *per un utente*, non a livello di sistema

- La "home" di *composer* si può vedere con: `composer config --global home`

Composer: installazione (via rete)

```
$ php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
# (v. https://getcomposer.org/download) scarica lo script di boot di composer, omettiamo di controllare l'hash
$ php composer-setup.php      # installerà composer.phar nella directory corrente (/Users/gp in questo caso)
All settings correct for using Composer    Downloading...
Composer (version 1.8.5) installed to: /Users/gp/composer.phar
Use it: php composer.phar
$ ls composer*  # come si vede (dal colore verde), composer.phar è un eseguibile! Cioè, è il tool composer
composer.phar composer-setup.php    # ora si può anche eliminare il file di setup composer-setup.php
```

(per altri approcci all'installazione: https://getcomposer.org/doc/00-intro.md)

- In Unix, il file *PHAR* (PHP archive) `composer.phar` si può lanciare direttamente, oppure lo si può passare a *PHP* come file da eseguire:

```
$ ./composer.phar -V       # composer.phar lanciato direttamente (ma comunque eseguito da interprete php)
Composer version 1.9.0 2019-08-02 20:55:32
$ php ./composer.phar -V   # composer.phar lanciato come argomento dell'interprete php, che lo esegue
```

- *Composer* si può anche installare dandogli un nome diverso dal *composer.phar*, come qui sotto, dove lo si chiama... *composer*:

```
$ php composer-setup.php --filename=composer
Composer (version 1.8.5) successfully installed to: /Users/gp/composer
Use it: php composer    # ovviamente questo comando va invocato dentro la cartella del file composer
```

Composer: installazione / 2

- Oltre al nome, si può anche scegliere la directory di installazione:

```
$ php composer-setup.php --install-dir=$HOME/bin --filename=composer
Composer (version 1.8.5) successfully installed to: /Users/gp/bin/composer
Use it: php /Users/gp/bin/composer
```

e, aggiungendola al *$PATH*, si può invocare *composer* direttamente:

```
$ export PATH=$HOME/bin:$PATH   # assicura che il composer appena generato sia nel PATH eseguibile
$ which -a composer
/Users/gp/bin/composer   # questo è il composer appena installato "a mano"
/usr/local/bin/composer  # questo composer era stato installato a livello di sistema (con apt / pacman / brew ...)
$ composer
Composer (version 1.8.5) ...
```

- *Composer* (versione >1.6) è anche in grado di aggiornarsi da sé:

```
$ composer self-update
Updating to version 1.9.0 (stable channel)...   Downloading (100%)
Use composer self-update --rollback to return to version 1.8.5
$ composer -V
Composer version 1.9.0 2019-08-02 20:55:32
```

- *Composer* ha molti altri comandi (alcuni mostrati nella prossima slide, una sorta di *cheat sheet* da riprendere più avanti, al bisogno)

```
$ composer
  Composer version 1.9.0 2019-08-02 20:55:32
Usage:  command [options] [arguments]
Options:
  -h, --help                     Display this help message
  -V, --version                  Display this application version
  -d, --working-dir=WORKING-DIR  If specified, use the given directory as working directory.
      --no-cache                 Prevent use of the cache
  -v|vv|vvv, --verbose           Increase the verbosity of messages:
Available commands:
  browse         Opens the package's repository URL or homepage in your browser.
  clear-cache    Clears composer's internal package cache.
  create-project Creates new project from a package into given directory.
  depends        Shows which packages cause the given package to be installed.
  diagnose       Diagnoses the system to identify common errors.
  dump-autoload  Dumps the autoloader.
  exec           Executes a vendored binary/script.
  global         Allows running commands in the global composer dir ($COMPOSER_HOME).
  help           Displays help for a command
  info           Shows information about packages.
  init           Creates a basic composer.json file in current directory.
  install        Installs project dependencies from the composer.lock file if present, or composer.json.
  list           Lists commands
  outdated       Shows installed packages that have updates available, including their latest version.
  prohibits      Shows which packages prevent the given package from being installed.
  remove         Removes a package from the require or require-dev.
  require        Adds required packages to your composer.json and installs them.
  run-script     Runs the scripts defined in composer.json.
  search         Searches for packages.
  self-update    Updates composer.phar to the latest version.
  suggests       Shows package suggestions.
  update         Upgrades dependencies to latest version according to composer.json, and updates composer.lock
  validate       Validates a composer.json and composer.lock.
```

Composer diagnose

La verifica non pare indispensabile subito dopo l'installazione, ma…

```
$ composer diagnose
Checking platform settings: OK
Checking git settings: OK git version 2.25.1
Checking http connectivity to packagist: OK
...
Checking disk free space: OK
Checking pubkeys: ...
OK
Checking composer version: OK
Composer version: 2.5.1
PHP version: 8.2.1
PHP binary path: /usr/bin/php8.2
OpenSSL version: OpenSSL 1.1.1f 31 Mar 2020
cURL version: missing, using php streams fallback, which reduces performance
zip: extension not loaded, unzip present, 7-Zip not available
```

Si attivino i moduli *php-curl* (o Laravel sarebbe lentissimo) e *php-zip*
Con Ubuntu vanno installati i relativi pacchetti

```
$ sudo apt install php-curl php-zip
```

Con Windows e XAMPP i moduli *curl* e *zip* sono pre-installati (check!), ma occorre attivare la direttiva `extension=zip` in `C:\Xampp\php\php.ini`

composer e bash-completion

Per *composer* è utile installare funzioni di auto-completamento, p.es. https://github.com/bramus/composer-autocomplete come segue:

- Una volta scaricato da GitHub il file *composer-autocomplete*, lo si installa in */etc/bash_completion.d* (o altrove, p.es. */usr/local/etc/bash_completion.d* o *$BASH_COMPLETION_COMPAT_DIR*) e si avvia una nuova bash:

```
$ sudo cp composer-autocomplete /etc/bash_completion.d/
```

```
$  # nuova sessione Bash, d'ora in poi è attivo l'autocompletamento (tasto [TAB]) per il comando composer
$ composer [TAB]
about                depends              info                 run                  update
...
create-project       i                    require              u
$ composer run-script [TAB]     # notevole il fatto che, se in composer.json (v. altrove), vi sono
il-mio-script                   # script dell'utente, il completamento automatico li trova e propone!
```

- NB: se si vuole disattivare il completamento (per le successive sessioni *bash*), basta cancellare il file dei comandi di autocompletamento

```
$ sudo rm /etc/bash_completion.d/composer-autocomplete
```

Ottima alternativa (funziona anche per il tool Laravel *artisan*, v. oltre): https://packagist.org/packages/bamarni/symfony-console-autocomplete

composer e bash-completion / 2

Da qualche versione in qua, *composer* stesso dà supporto (forse meno ricco delle soluzioni viste) all'auto-completamento:

```
$ composer completion bash > completion.bash
$ source completion.bash                                    # abilita subito l'auto-completamento
$ sudo cp completion.bash /etc/bash_completion.d/composer   # installa l'auto-completamento
```

Altri modi di installare l'autocompletamento li mostra il comando:

```
$ composer completion --help
```

Una volta installato l'autocompletamento per *composer*:

```
$   # nuova sessione Bash, d'ora in poi è attivo l'autocompletamento (tasto [TAB]) per il comando composer
$ composer [TAB]
about               depends             info                run                 update
...
create-project      i                   require             u
$ composer run-script [TAB]     # notevole il fatto che, se in composer.json (v. altrove), vi sono
il-mio-script                   # script dell'utente, il completamento automatico li trova e propone!
```

Installare package PHP con *composer*

Essenzialmente, il tool *composer* serve a installare un **package** PHP, scaricando dalla rete il package stesso (i suoi file, cioè) e le sue **dipendenze**, cioè altri package PHP forniti dai rispettivi *vendor*

- l'installazione delle dipendenze è **ricorsiva**, cioè *composer* installa anche le dipendenze delle dipendenze e così via
- l'installazione avviene lanciando *composer* dalla directory, sia *d*, in cui andrà il package, mentre le dipendenze andranno in una subdirectory *d/vendor*

Le dipendenze sono definite in un file *composer.json*, sotto la chiave *require*, con un elenco di elementi della forma:

- *vendor-name/package-name:version-number* (es. 1.0 o ^1.0 o 1.*)

(In realtà, nella determinazione delle dipendenze entra anche un file *composer.lock*, di cui non parleremo)

Installare package PHP con *composer*

Ecco i principali comandi di *composer* legati all'installazione di package PHP con dipendenze e quindi al file delle dipendenze *composer.json*:

```
$ composer
Usage:  command [options] [arguments]
  ...
Available commands:
  browse         Opens the package's repository URL or homepage in your browser
  create-project Creates new project from a package into given directory
  init           Creates a basic composer.json file in current directory
  install        Installs project dependencies from composer.json or the composer.lock file if present
  require        Adds required packages to your composer.json and installs them
  search         Searches for packages
  global         Allows running commands in the global composer dir ($COMPOSER_HOME).
  ...
```

composer global è seguito, tipicamente, da *install/require/create-project*

- Come già spiegato, *composer*, senza *global*, installa il software nella directory corrente, quella da cui viene lanciato, invece...
- *composer global* installa in una directory fissa sotto la home ~ dell'utente, su OSX *~/.composer*, su Linux *~/.config/composer* o *$COMPOSER_HOME*)
  - per rendere gli screenshot da Mac nel seguito validi anche con Linux:

```
$ ln -s ~/.config/composer ~/.composer   # compatibilità con prossime slide, se usi Linux e composer è installato
```

Installare... l'installer di Laravel

Useremo *composer* per installare il package/app (PHP) *laravel/installer* con le sue dipendenze

Lo si installa `global` cioè in *~/.config/composer* (o *$COMPOSER_HOME*)

- senza `global` sarebbe installato nella directory corrente (tipicamente di una specifica app), il che non è però particolarmente utile per l'installer di *laravel* (è un tool eseguibile, conviene averlo nel $PATH)

```
$ composer global require laravel/installer       # global significa: installa nella directory
Changed current directory to /Users/gp/.composer  # $COMPOSER_HOME (default Mac ~/.composer)
Using version ^2.1 for laravel/installer
./composer.json has been created   # NB: nella directory $COMPOSER_HOME
Loading composer repositories with package information... Updating dependencies
Package operations: 12 installs, 0 updates, 0 removals
  - Installing symfony/process (v4.2.8): Loading from cache
  ...
  - Installing laravel/installer (v2.1.0): Loading from cache
symfony/contracts suggests installing psr/cache (When using the Cache contracts)
...
guzzlehttp/guzzle suggests installing psr/log (Required for using the Log middleware)
Writing lock file
Generating autoload files
```

Lanciare l'installer di Laravel

- Nella dir `.composer`, dopo l'installazione appena compiuta, saranno comparsi l'eseguibile ***laravel*** e un ***link*** ad esso:

```
$ ls -ld ~/.composer/vendor/laravel/installer/bin/laravel ~/.composer/vendor/bin/laravel
-rwxr-xr-x  ...      .composer/vendor/laravel/installer/bin/laravel
lrwxr-xr-x  ...      .composer/vendor/bin/laravel -> ../laravel/installer/laravel
```

- Ora si aggiunge al PATH la directory `.composer/vendor/bin/` con (il link al file) *laravel*, per semplificarne l'invocazione:

```
$ ls ~/.composer/vendor/bin/laravel
.composer/vendor/bin/laravel
$ export PATH=~/.composer/vendor/bin:$PATH   # questo comando su PATH si può inserire in .bash_profile
$ laravel
Laravel Installer 3.0.1
Usage:  command [options] [arguments]
Available commands:
  completion    Dump the shell completion script
  new           Create a new Laravel application ...
```

- NB: il tool *laravel* è chiamato *installer* di Laravel nel senso che genera applicazioni Laravel, *installando* per esse il framework Laravel, cioè i pacchetti/librerie che costituiscono Laravel

Aggiornare l'installer di Laravel / 1

Periodicamente il vendor *laravel* aggiorna l'installer (sul repo GitHub); per aggiornare il tool sul vostro sistema:

```
$ composer global update laravel/installer
```

Spesso però questo non avrà effetto, vediamo il perché sperimentalmente:

- per cominciare, rimuoviamo l'installer di laravel e installiamone volutamente una versione vecchia:

```
$ rm -rf .config/composer/vendor/laravel/
$ composer global require laravel/installer=4.1.0
Changed current directory to /home/gp1/.config/composer
./composer.json has been updated
Running composer update laravel/installer
Loading composer repositories with package information
Updating dependencies
Your requirements could not be resolved to an installable set of packages. Problem 1
 - Root composer.json requires laravel/installer 4.1.0 ...
 - laravel/installer v4.1.0 requires symfony/console ^4.0|^5.0 ... but package is
fixed to v7.0.2 ... and that version does not match.
Use the option --with-all-dependencies (-W) to allow upgrades, downgrades and
removals for packages currently locked to specific versions.
Installation failed, reverting ./composer.json to their original content.
```

Aggiornare l'installer di Laravel / 2

```
$ composer global require laravel/installer=4.1.0
...
Your requirements could not be resolved to an installable set of packages. Problem 1
 - laravel/installer v4.1.0 requires symfony/console ^4.0|^5.0 ... but package is
fixed to v7.0.2 ... and that version does not match.
Use the option --with-all-dependencies (-W)
```

Spesso però questo non avrà effetto, vediamo il perché sperimentalmente:

- per cominciare, rimuoviamo l'installer di laravel e installiamone volutamente una versione vecchia:

Installer di Laravel in Windows: configurazioni

L'installazione avverrà in *c:\Users\gp\AppData\Roaming\Composer*

```
C:> composer global require laravel/installer
Changed current directory to C:/Users/gp/AppData/Roaming/Composer
...
```

Per lanciare l'installer *laravel* conviene configurare PATH, da riga di comando o in permanenza dal pannello di controllo:

```
C:> set PATH=c:\Users\gp\AppData\Roaming\Composer\vendor\bin;%PATH%
C:> laravel
Laravel Installer 4.2.17
...
```

Per funzionare, il comando *laravel* deve "vedere" il comando *composer*

Il modo più ovvio è definire un file *composer.bat* come qui a destra, ponendolo p.es. nella directory in cui si trova l' archivio PHP *composer*

```
@ECHO OFF
@REM Versione naive
set COMPOSER=c:\Users\gp\tsdw
php %COMPOSER% %*
```

Conviene poi aggiungere a PATH la directory in cui è *composer.bat*:

```
C:> SET PATH=c:\Users\gp\tsdw;%PATH%
```

Altre indicazioni

- https://victorlava.com/install-php-and-composer-on-windows-10-for-use-in-cmd-or-powershell/#Install-PHP-on-Windows

Alla fine bisogna capire come installare php e composer su Windows, forse la URL dà qualche indicazione utile

Generare applicazioni Laravel col tool *laravel*

- Con il tool *laravel*, si genera una applicazione Laravel *my_app*, insieme con le molte dipendenze richieste (v. sotto `Installing dependencies...`)

```
$ laravel new -n my_app # serve rete per scaricare i pacchetti php (se non in cache)
Crafting application... Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
Package operations: 76 installs, 0 updates, 0 removals
  - Installing doctrine/inflector (v1.3.0): Downloading (100%)
...
  - Installing phpunit/phpunit (8.5.2): Downloading (100%)
... suggests ...   ... suggests ... # pacchetti di cui composer suggerisce l'installazione

Generating optimized autoload files

> @php -r "file_exists('.env') || copy('.env.example', '.env');"
> @php artisan key:generate -ansi      # artisan è il tool fondamentale di laravel, qui viene
Application key set successfully.      # invocato (automaticamente) per generare la chiave nel file .env

> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: beyondcode/laravel-dump-server
...

Package manifest generated successfully.
Application ready! Build something amazing.
```

NB: ***-n*** evita che *laravel new* (versione 5) ponga all'utente delle domande interattive durante il processo di generazione dell'app Laravel

La app generata da Laravel

- L'applicazione generata dal *laravel new my_app* viene installata in una sua directory chiamata *my_app*, in cui troviamo il suo codice e il framework Laravel (una copia specifica per questa app)

```
$ ls my_app/
README.md  composer.json  package-lock.json  resources  tests
app        composer.lock  package.json       routes     vendor
artisan    config         phpunit.xml        server.php webpack.mix.js
bootstrap  database       public             storage
```

- in *my_app/vendor* troviamo la copia del framework Laravel, le sue dipendenze ed eventuali altri pacchetti PHP installati (per esempio sulla base dei `...suggests...` di `laravel new`)

```
$ ls my_app/vendor/
autoload.php  facade        league     opis          psr         theseer
bin           fideloper     mockery    paragonie     psy         tijsverkoyen
composer      filp          monolog    phar-io       ramsey      vlucas
dnoegel       fzaninotto    myclabs    phpdocumentor scrivo      webmozart
doctrine      hamcrest      nesbot     phpoption     sebastian
dragonmantank jakub-onderka nikic      phpspec       swiftmailer
egulias       laravel       nunomaduro phpunit       symfony
```

Generare app laravel con *composer*

```
composer create-project laravel/laravel your-project-name --prefer-dist
composer create-project laravel/laravel=8.1.2 your-project-name –prefer-dist
composer create-project laravel/laravel=8.1.* your-project-name –prefer-dist
```

- Consente di installare versioni specifiche e più vecchie di Laravel
- https://stackoverflow.com/questions/23754260/installing-specific-laravel-version-with-composer-create-project
- Equivalente: *git clone* del repo *Laravel*
- Aggiornamento attraverso *composer update* nella dir dell'app

Uso di Laravel: i due ruoli di *composer*

- Come visto, con *composer* si è installato *"global"* il pacchetto PHP *laravel/installer* (e altri pacchetti da cui questo dipende)
  - nella directory *~/.composer/...* è quindi comparso il tool *laravel*:

```
~ $ .composer/vendor/bin/laravel
Laravel Installer 2.1.0
Usage:  command [options] [arguments]
Options:
  -h, --help            Display this help message
  -V, --version         Display this application version
  -v|vv|vvv, --verbose  Increase the verbosity of messages
Available commands:
  help  Displays help for a command
  list  Lists commands
  new   Create a new Laravel application
```

- In relazione a Laravel, e precisamente ad app Laravel come *my_app*, *composer* ha anche il ruolo di gestire (installare/aggiornare) le dipendenze dell'app da vari pacchetti/librerie PHP (poste in *my_app/vendor*):

```
my_app $ ls vendor/
autoload.php  doctrine      filp          league    nikic       phpdocumentor  psy          theseer
beyondcode    dragonmantank fzaninotto    mockery   nunomaduro  phpoption      ramsey       tijsverkoyen
bin           egulias       hamcrest      monolog   opis        phpspec        sebastian    vlucas
composer      erusev        jakub-onderka myclabs   paragonie   phpunit        swiftmailer  webmozart
dnoegel       fideloper     laravel       nesbot    phar-io     psr            symfony
```

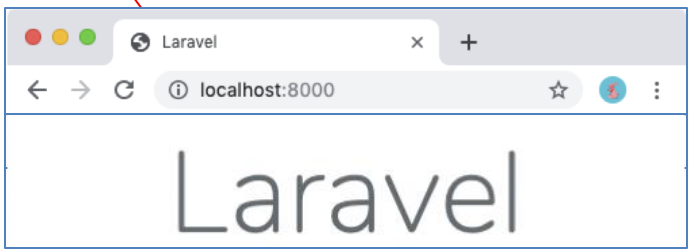
Servire app Laravel

```
$ cd my_app              # si entra nella directory della app Laravel generata
my_app $ file artisan    # in essa c'è il file artisan, script PHP di gestione della app
Artisan: a /usr/bin/env php script text executable, ASCII text
my_app $ php artisan     # ... help con numerosissimi comand, tra cui serve
...
```

- Ora è già possibile servire l'app (template) generata a clienti locali (si presume si usi una macchina di sviluppo), con *php artisan serve* :

```
my_app $ php artisan serve  # avvia un server, ora si può puntare il browser a localhost:8000
Laravel development server started: <http://127.0.0.1:8000>
[Thu May 16 03:08:18 2019] 127.0.0.1:56160 [200]: /favicon.ico
```

- NB: questo "web server" *artisan serve*, che usa la CLI PHP, è adatto alla fase di sviluppo, ma non "in produzione"



- *artisan*, con tantissimi comandi, è il tool fondamentale per Laravel

L'altra possibilità è servire le app Laravel con Apache, correttamente configurato e con il suo modulo PHP (v. altre slide)

Servire altre app Laravel

- Per passare a generare un'altra app, basta cambiare directory (presumibilmente la *home*)
- E usare di nuovo il wizard *laravel new*
  - stavolta i pacchetti che compongono il framework sono già in *cache*, quindi non verranno scaricati

```
$ laravel new my_app_1
Crafting application...
Package operations: 76 installs, 0 updates, 0 removals
  - Installing doctrine/inflector (v1.3.0): Loading from cache
...
Generating optimized autoload files
...
Application key set successfully.
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
...
Package manifest generated successfully.
Application ready! Build something amazing
```

- Per servire questa app *insieme* alla precedente, va impiegato un altro port

```
$ cd my_app_1
my_app $ php artisan serve -port=8001
Laravel server started:
<http://127.0.0.1:8001> ...
```

Servire direttamente da PHP

- Il comando *artisan serve*, oltre a effettuare qualche verifica e mostrare dei diagnostici, non fa altro che invocare l'interprete PHP CLI sul file *server.php*, generato automaticamente dal tool *laravel new*, nella directory base (p.es. *my_app*) della applicazione
- Quindi, un altro modo di "servire" un'app Laravel ai clienti, è invocare dalla shell, nella directory della app:

```
$ cd my_app
my_app $ php -S localhost:8001 server.php
Listening on http://localhost:8001
Document root is /Users/gp/laracode/my_app
Press Ctrl-C to quit.
```

  (così però il cliente (browser) ha una certa visibilità della presenza di altri file presenti nelle (sotto)directory della app)

- NB: si potrebbe anche lanciare *php -S localhost:8001* senza l'arg *server.php* e l'app risponderebbe su *http://localhost:8001/server.php*, ma i link dell'applicazione probabilmente non funzionerebbero
  - Inoltre per il server, la "Root dir" sarebbe quella dell'app (*my_app* nell'esempio): altri file in essa e in sottodirectory restano visibili

Il tool *artisan*

Si è visto che nella directory di una app Laravel è presente *artisan,* script PHP, di cui si sono usati i comandi *serve*, *key:generate*, *package:discover*

Si tratta di un tool per lo sviluppatore Laravel, ricchissimo di funzionalità:

```
app8 $ php artisan
Laravel Framework 5.8.33
Usage:  command [options] [arguments]
Options:
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
      --ansi            Force ANSI output
      --no-ansi         Disable ANSI output
  -n, --no-interaction  Do not ask any interactive question
      --env[=ENV]       The environment the command should run under
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
Available commands:
  clear-compiled       Remove the compiled class file
  down                 Put the application into maintenance mode
  dump-server          Start the dump server to collect dump information.
  env                  Display the current framework environment
  help                 Displays help for a command
  inspire              Display an inspiring quote
  list                 Lists commands
  migrate              Run the database migrations
  optimize             Cache the framework bootstrap files
  preset               Swap the front-end scaffolding for the application
  serve                Serve the application on the PHP development server
  tinker               Interact with your application
  up                   Bring the application out of maintenance mode
 app
  app:name             Set the application namespace
```

```
 auth
  auth:clear-resets    Flush expired password reset tokens
 cache
  cache:clear          Flush the application cache
  cache:forget         Remove an item from the cache
  cache:table          Create a migration for the cache database
table
 config
  config:cache         Create a cache file for faster ...
  config:clear         Remove the configuration cache file
 db
  db:seed              Seed the database with records
 event
  event:cache          Discover and cache application's events ...
  event:clear          Clear all cached events and listeners
  event:generate       Generate missing events and listeners ...
  event:list           List the application's events and ...
 key
  key:generate         Set the application key
 make
  make:auth            Scaffold basic login and registration ...
  make:channel         Create a new channel class
  make:command         Create a new Artisan command
  make:controller      Create a new controller class
  make:event           Create a new event class
  make:exception       Create a new custom exception class
  make:factory         Create a new model factory
  make:job             Create a new job class
  make:listener        Create a new event listener class
  make:mail            Create a new email class
  make:middleware      Create a new middleware class
  make:migration       Create a new migration file
  make:model           Create a new Eloquent model class
  make:notification    Create a new notification class
  make:observer        Create a new observer class
  make:policy          Create a new policy class
  make:provider        Create a new service provider class
  make:request         Create a new form request class
  make:resource        Create a new resource
  make:rule            Create a new validation rule
  make:seeder          Create a new seeder class
  make:test            Create a new test class
 migrate
  migrate:fresh        Drop all tables and re-run all
migrations
  migrate:install      Create the migration repository
  migrate:refresh      Reset and re-run all migrations
  migrate:reset        Rollback all database migrations
  migrate:rollback     Rollback the last database migration
  migrate:status       Show the status of each migration
 notifications
  notifications:table  Create migration for notifications table
 optimize
  optimize:clear       Remove the cached bootstrap files
 package
  package:discover     Rebuild the cached package manifest
 queue
  queue:failed         List all of the failed queue jobs
  queue:failed-table   Create a migration for the failed ...
  queue:flush          Flush all of the failed queue jobs
  queue:forget         Delete a failed queue job
  queue:listen         Listen to a given queue
  queue:restart        Restart queue worker daemons after job
  queue:retry          Retry a failed queue job
  queue:table          Create a migration for jpbs queue
  queue:work           Start processing jobs on the queue
 route
  route:cache          Create a route cache file for faster ...
  route:clear          Remove the route cache file
  route:list           List all registered routes
 schedule
  schedule:run         Run the scheduled commands
 session
  session:table        Create a migration for session DB table
 storage
  storage:link         Create symbolic link for public/storage
 vendor
  vendor:publish       Publish publishable assets from packages
 view
  view:cache           Compile application's Blade templates
  view:clear           Clear all compiled view files
```

Artisan: autocompletamento nativo

```
my_app $ ./artisan completion > ~/artisan.sh   # genera codice bash per autocompletamento e lo salva
my_app $ source artisan ~/artisan.sh           # attiva autocompletamento
```

artisan e bash-completion: script

- Per *artisan* sarebbe assai utile il completamento automatico (col tasto [TAB]) sulla *bash*, data la miriade di comandi e opzioni
- Questa funzionalità è (quasi) nativa per *zsh*, mentre per *bash* si può usare lo script da https://gist.github.com/tuanpht/2c92f39c74f404ffc712c9078a384f39

  NB: dopo `complete -F _artisan php` aggiungere `complete -F _artisan artisan`
- Il file scaricato e modificato, chiamato p.es. *artisan.auto*, va posto in */etc/bash_completion.d/* o in */usr/local/etc/bash_completion.d/* o *$BASH_COMPLETION_COMPAT_DIR* (che in genere è nella *home*):

```
$ sudo cp artisan.auto /etc/bash_completion.d/
$ Ctrl-D
```

- Aperta una nuova shell, si verifica che il completamento sia attivo:

```
$   # nuova sessione Bash, con autocompletamento per artisan attivo
$ complete -p artisan
complete -F _artisan artisan
```

- Ora però occorre avere un comando *artisan* da autocompletare...

artisan e bash-completion: script e alias

- Occorre definire un comando di bash '*artisan*' da autocompletare; lo si introduce con *alias* ed ecco attivo il completamento con tab:

```
my_app $ alias artisan='php artisan'   # si potrebbe inserire in .bashrc etc.
my_app $ artisan [TAB]                 # premendo il tasto TAB si attiva il completamento automatico
app:name             help               make:migration       migrate:rollback     route:cache
auth:clear-resets    inspire            make:model           migrate:status       route:clear
cache:clear          key:generate       make:notification    notifications:table  route:list
...
down                 make:event         make:seeder          queue:flush          up
dump-server          make:exception     make:test            queue:forget         vendor:publish
env                  make:factory       migrate              queue:listen         view:cache
event:cache          make:job           migrate:fresh        queue:restart        view:clear
...
my_app $ artisan migrat [TAB]    # il completamento funziona anche con sottocomandi e opzioni
artisan migrat
migrate           migrate:install   migrate:reset     migrate:status
migrate:fresh     migrate:refresh   migrate:rollback
```

- Per disattivare l'autocompletion (mantenendo l'alias) nella sessione corrente:

```
my_app1 $ complete -r artisan
my_app1 $ complete -p artisan
-bash: complete: artisan: nessun completamento specificato
```

- Per disattivarlo in permanenza, in successive sessioni *bash*, e per ogni app:

```
my_app1 $ sudo rm /etc/bash_completion.d/artisan
```

artisan e bash-completion, con PHP / 1

Un altro modo di dotare *artisan* della *bash completion* è installare dell' apposito codice PHP, attraverso *composer*, localmente a ciascuna app

- ciò perché anche *artisan* è installato localmente alla app

```
my_app $ composer require balping/artisan-bash-completion
...
  - Installing balping/artisan-bash-completion (v1.0.0): Extracting archive
```

- Il pacchetto installato contiene anche lo script per la *bash completion*; lo si installa in */etc* (o */usr/local/etc* o `$BASH_COMPLETION_COMPAT_DIR`) (NB: ciò va fatto una volta sola, per una qualsiasi app, qui sotto *my_app*):

```
my_app $ sudo cp vendor/balping/artisan-bash-completion/artisan /etc/bash_completion.d/
my_app $ Ctrl-D
```

- Aperta una nuova shell, si verifica che *artisan* abbia l'autocompletamento:

```
my_app $  # questa è una nuova sessione Bash, con autocompletamento per artisan attivo, verifichiamolo:
my_app $ complete -p artisan
complete -F _artisan artisan
```

- Anche in questo caso, va definito il comando/alias *artisan* da autocompletare

artisan e bash-completion, con PHP / 2

Ottima alternativa (serve anche il gestore di pacchetti *composer*, v. prima):
https://packagist.org/packages/bamarni/symfony-console-autocomplete

```
$ composer global require bamarni/symfony-console-autocomplete
Changed current directory to /home/gp/.config/composer
...
  - Installing  bamarni/symfony-console-autocomplete (v1.5.5): Extracting archive
...
```

- Per questa soluzione, come si vede, l'installazione è globale (il che è certamente un vantaggio)
- I collegamenti con *bash-completion* sono gestiti più semplicemente eseguendo (meglio se una volta per tutte da *.bash_profile*):

```
$ eval "$(symfony-autocomplete)"
```

- Aperta una nuova shell, si verifica che *artisan* abbia l'auto-completamento:

```
my_app $   # questa è una nuova sessione Bash, con autocompletamento per artisan attivo, verifichiamolo:
my_app $ complete -p artisan
complete -o default -F _symfony artisan
```

- Anche in questo caso, va definito il comando/alias *artisan* da autocompletare