Laravel 6

Notes open for creative commons use @ developer blog: https://unfoldkyle.com, github: SmilingStallman*, email: kmiskell@protonmail.com*

Learning Resources
Primary: Official Docs, https://laravel.com/docs/6.x
Secondary: https://laracasts.com/

Intro

-PHP MVC framework based on Symfony with version 1.0 in 2011, and current version (6.13)

-Built in authentication, restful routing, lightweight templating through Blade templating engine, unit testing supported out of box, wide array of OO libraries built in.

-Built in command line interface through Artisan

-Highly modularized packaging system with dedicated dependency manager

-Prepare SQL statements only (preventing sql injection), as well as many other security features

-General flow: User submits request for page. Laravel sees request and routes to proper controller based on config in *routes.php*. Controller, which holds domain knowledge and business logic, gets request. Back-end logic runs, controller passes data to view, then view loads and is displayed to user.

-Note that somePage.php and somePageController.php are separated, so controller acts as de-coupled between view and back-end logic.

-Laravel 6 requires PHP 7.2 (released Nov 2017) or higher

-Using a framework like laravel lets you not re-invent the wheel, handles security well by default, can provide performance updates, and makes extension easy via plugins. It also standardizes code when used in multi-projects, files, etc. by providing a common design and implementation.

Installation
-Install PHP, MySQL, Composer, Laravel

https://www.digitalocean.com/community/tutorials/how-to-install-linux-nginx-mariadb-php-lemp-stack-on-debian-10

https://getcomposer.org/download/

https://laravel.com/docs/6.x#installation https://laracasts.com/series/laravel-6-from-scratch/episodes/3

-If Laravel install complains about missing ext-zip missing, install via:
sudo apt-get install php-zip

-On Debian, set *$PATH* by adding below to *~/.bashrc*
export PATH="$HOME/.config/composer/vendor/bin:$PATH"

Why Use in Building API
-MVC works well for building properly designed API
-Eloquent ORM through Artisan allows for auto-creation of models based on table existing scheme and allows proper DB layer
-Automated unit test
-Fast, simple, clean routing
-Managed queuing during high load times to increase speed
-Easy and quick autoloading

Composer, Namespaces, and Autoloading
-Knowledge on all of these will be required for aptly using Laravel. See *PHP7.odt* notes for info on these under sections, "Advanced PHP," and, "Composer"

New Project
-Create new laravel project: *laravel new projectName*

-On first attempt to create new on Debian, PHP was missing dependencies. Install via:
sudo apt-get install php-mbstring
sudo apt-get install php-dom

-Creation will create folder *projectName* with all required dependencies, as well as template html page, similar to React via *create-react-app*

-Laravel comes with config for local dev server for project, typically running on *http://127.0.0.1:8000*. From project dir, start via: *php artisan serve*

-First project might take a long to build, but after first, dependencies cached, and thus *new* is almost instant.

Configuration

-All config files stored in *config* project folder. Ex. *config/app.php*, *session.php*

Env Config
-Laravel project folder contains a *.env* file which contains env config.
-Do not commit as different devs might have different dev environments

-Useful to create template *.env.example* file with commit, to serve as basis for dev env config for other devs.

-Holds config settings in style:

```
DB_HOST=127.0.0.1
APP_URL=http://localhost
SOME_PROP="value with spaces"
```

-Make sure that proper changes for prod vs dev env set in *.env* when prod pushing

-Can get *.env* values in PHP either through super-global *$_ENV['SOME_KEY']*
-Can also get via global Laravel function *env('SOME_KEY', optional-val)* where *optional-val* is returned if *env* var with *SOME_KEY* does not exist.

-Can get current app env from *APP_ENV key* through *App* facade call of *App::environment()* (ex. *local,* ex. *staging)*
-Can pass in *'environment_name'* string or *['local', 'staging', 'otherEnv']* string array to *environment(),* which then returns bool if current env matches

Hiding Env Vars
-*.env* has a bool *APP_DEBUG* var. If *true*, uncaught exception will result in debug page showing all env vars and vals

-To hide vars (ex. on prod), add a *'debug_blacklist'* assoc array containing details to hide as contents to existing returned array in *config/app.php*, in form:

```
return [

   // ...

   'debug_blacklist' => [
      '_ENV' => [
         'APP_KEY',
         'DB_PASSWORD',
      ],

      '_SERVER' => [
         'APP_KEY',
         'DB_PASSWORD',
      ],

      '_POST' => [
         'password',
      ],
   ],
];
```

-As seen above, some env vars also available to $*_SERVER* so will need to blacklist there as well, if so

Accessing Config Values
-Can access config values from files in *config* dir from global laravel function

config(‘filename.prop_name) , which takes in optional second arg to be returned if config var does not exist
-ex. *$zone = config(‘app.timezone’)* *//pulls from config/app.php*

Config Caching
-Calling *php artisan config:cache* via terminal will take all existing config files and combine into single config file

-As once done, combined config file will be called instead of *.env*, which is bad for dev env as *.env* could change on dev. Thus only cache for prod.

Maintenance Mode
-Can put app into maintenance mode, which generates custom temp view for all app requests, with default view being page displaying *503* | *Service Unavailable*
-Useful for when updating, perfoming maintenance, etc. on app

-Enable via:
php artisan down --message=”upgrading DB” --retry=60 --allow=127.0.0.1
-Last three args are optional. Can have multi *allow=… allow=…* to allow specific ip to access normal page even during maintenance mode.

-Disable via: *php artisan up*

-To create custom maintenance mode page to display instead of default, created file at *resources/views/errors/503.blade.php*

-If need zero downtime deployment and maintenance, consider using Laravel service *Envoyer*

Directory Structure

***App* Dir**
-Core code for app including controllers, middleware, exceptions, providers, etc.

***Bootstrap* Dir**
-Bootstrapping – loader first run on request, responsible for loading rest of core app code

-Contains *app.php* which handles bootstrapping the framework
-Also contains a *cache* dir for all cache files (*services.php, routes.php, etc)*

***Database* Dir**
-DB migrations, model factors, and seeds. Optionally can use to hold a SQLite DB.

***Public* Dir**
-Contains all assets (JS, CSS, images, etc) and *index.php*, which is entry point for all requests to app and configs autoloading

***Resources* Dir**

-Views, language files, and un-compiled assets from LESS, SASS, JS, etc.

Routes **Dir**
-All routes for app
-*web.php* is main route file and includes middleware for handling auth, session state, cookie encryption, etc.

-*api.php* routes for stateless routes for routes used by *RouteServiceProdvider* in *api* middleware

-*console.php* defines Closure based console commands and defines console based entry points into app

-*channels.php* holds all broadcasting channels supported by app

Storage **Dir**
-Compiled Blade templates, file based sessions, file caches, and other framework generated files

-Subdirs: *app* holds files generated by app, *framework* for framework generated files and caches, *logs* for app logs

-*storage/app/public* user-generated files (ex. avatars). Should create symbolic link at public/storage which points to this dir. Create via *php artisan storage:link*

Tests Dir
-All automated tests with example test provided by *PHPUnit*

Vendor **Dir**
-All composer dependencies

App **Dir subdirs**
-Namespace *App*

-*Broadcasting dir* – all broadcast channels for app.

-*Console dir* – Provides mechanisms to interact with app. All Artisan cmds for app. Console Kernel. Artisan files where scheduled tasks defined.

-*Events* dir – houses event classes. Events can be used to alert diff parts of app that action has occurred.

-*Exceptions* dir – app exception handler. Store exceptions here. Mod *Handler* class in this dir to change how exceptions logged and rendered.

-*Http* dir - Provides mechanisms to interact with app. Controllers, middleware, form requests. Bulk of files with logic for requests go here.

-*Job*s dir – holds queueable jobs for app. Jobs can set to be run synch or asynch. Synch jobs here.

-*Listeners* dir – classes that handle events here. Listeners receive event instance and respond with logic.

-*Mail* dir – classes that represent emails sent by app. Mail objects encapsulate logic for building email in class that can be sent using *Mail::send* method

-*Notifications* dir – All transactions notification for app. Laravel can send notifications via email, Slack, SMS, stored in DB, etc..

-*Policies* dir – auth policy classes for app. Policies used to set if user can perform a given action on a resources

-*Providers* dir – service providers for app, which bootstrap app by binding services in service container, registering events, or any other task to prep app for incoming requests.

-*Rules* dir – custom validation objects for app

Valet & Homestead

-Valet is mac only. Development environment managing nginx, allowing local access to env via tunnels, uses only 7mb, provides support for wide array of frameworks, etc.

-Alt for win/linux is Homestead. Homestead uses Vagrant, which creates and manages virtual machines through VMware, Virtualbox, etc..

-Homestead itself is a pre-packaged Vagrant “box” set up for PHP and Laravel. Quicker to set up than setting up env manually

<u>COME BACK MORE TO THIS AND SET UP ENV AFTER GO FURTHER IN LARAVEL BASICS</u>

Deployment

-Root directory for webserver should be set to *my_project/public* which holds *index.php,* which starts Laravel, includes autoloading for dependencies, requires Bootstrap, and serves as front controller for all HTTP requests entering app

-Ensure project *storage* and *bootstrap/cache* dir have write permissions

-If <u>did not</u> build project with *laravel new* (aka composer), set app key for security modules via:
php artisan key:generate

-Additional useful config (ex. timezone) in *my_project/config/app.php*

-Depending on project, also might want to configure *my_project/config/cache.php,* *my_project/config/database.php,* and *my_project/config/session.php*

-For security reasons, always host laravel projects in the root folder for the “web directory” (ex. */var/www/*). Example:

```
server {
            listen 80;
            listen [::]:80;

            root /var/www/html/quickstart/public;
            index index.php index.html index.htm index.nginx-debian.html;

            server_name example.com www.example.com;

            location / {
                  try_files $uri $uri/ /index.php?$query_string;
            }
}
```

-See full example of starting nginx *sites-available* file at:
https://laravel.com/docs/6.x/deployment

Optimization
-Do not run any of following in dev branch, only prod

-Optimize Composer’s autoloader map for quick file finding and loading:
```
composer install --optimize-autoloader –no-dev
```

-Combine all config files into single file:
```
php artisan config:cache
```

-If large app with many route, combine all route registrations into single method call:
```
php artisan route:cache
```

For more details on deployment:
https://www.digitalocean.com/community/tutorials/how-to-deploy-a-laravel-application-with-nginx-on-ubuntu-16-04

Request Lifecyle

Lifecyle Overview
-Entry point for app is *public/index.php*. All requests directed here via *nginx sites-available* config. *Index.php* loads composer generated autoloader and gets instance of app from *bootstrap/app.php*.

-Once app bootstrapped and loaded, request sent to either HTTP (*app/Http/Kernel.php*) or console kernel (depending on request origin). HTTP kernel includes bootstrappers to config error handling, logging, detect app env, etc. Also has list of middleware requests mass pass through before hitting app logic (read/write session, verify CSRF token, etc.). Also handles sending back response through similar steps.

-During kernel bootstrapping, service providers loaded. These handle all the boostraping for framework components.

-Service providers configured in *config/app.php* in the *providers* array. Providers first all registered via *register* method, then *boot* method called.

-Once app bootstrapped and all service providers registered, requent handed off to router for dispatching to route or controller, with needed running of route specific middleware as well.

-Summary: *index.php* gets request, autoloads Laravel classes, and gets instance of app. HTTP kernel then bootstraps configs and sets up middleware. Service providers then loaded, and request handed off to proper route/controller for processing. Response then passes through any needed middleware and sent back to user.

-Add custom bootstrapping in *AppServiceProvider*