INTRO

Basis of web dev https://youtu.be/ysEN5RaKOlA

For Backend, first week to-dos:

1. Go over laracast.com videos and it will give you a good basis of how Laravel works https://laracasts.com/series/laravel-6-from-scratch/episodes/1
2. Simultaneously consult https://laravel.com/docs/7.x to read up more on documentation of the framework
3. Do a follow along project for practice i.e. https://www.youtube.com/watch?v=ImtZ5yENzgE

This document contains information regarding setting up your machine, git info, MVC architecture, concepts/examples/documentation of the Laravel framework and some educational/interesting links at the end. Feel free to add to this document and pass it on to future interns as well!

SETUP & TECH & SOME DEFINITIONS

Summary of setup

1. get vm, provision, get homestead <tell Vagrant to use specific laravel stuff>, cloned repo, installed dependencies, migrated db, and imported dev db, pointed the fmls and links to vm IP).
2. Mirror everything in VM essentially.

Tap credentials (changed, ask for new ones)

1. Cp2 admin@cloudhorizon.com/Evolution2017!
2. For demo: admin@tapmedical.ca Password: tapmedical
3. Cp2 for the development site: username: administrator@tapmedical.ca pwd: Evolution2017!

What's Vagrant

1. Remote development environments force users to give up favorite editors/programs. Vagrant works on your local system with the tools you're already familiar with.
2. Makes it easy to work with VM (virtual machine).
3. Manage's VM environments in a single workflow.
4. It builds a top, for instance, VirtualBox.
5. https://www.taniarascia.com/what-are-vagrant-and-virtualbox-and-how-do-i-use-them/
6. https://stackoverflow.com/questions/34745295/a-vagrant-environment-or-target-machine-is-required

Vagrant

1. If your machine is down
2. Re-open VM
3. Start homestead
4. In local terminal: vagrant up, vagrant reload --provision, vagrant ssh, cd into proper file and enter **php artisan serve** in cmd

Laravel Homestead box

1. https://laravel.com/docs/5.8/homestead#installation-and-setup
2. Pre-packaged Vagrant box that provides you a dev environment without requiring you to install PHP, a web server and any other server software on your local machine. Vagrant boxes are completely disposable.
3. No worries about messing up your OS.

Difference between Docker vs VM

1. https://www.nuvation.com/resources/article/4-reasons-you-should-be-using-vms-fpga-and-software-development
2. VM's are safer, on your own os, won't corrupt the environment.
3. Use VM so everyone is on the same track, it is a box that can't access your room.

4. Dockers: so let's say you need this program, and this dependency, and this other thing to run your application, you can package everything into a docker container so that other people can simply download the container and run the application without going through the hassle of downloading all the separate stuff needed.

How to tell mac to look for certain domains

1. Need to go to sudo nano /etc/hosts
2. To tell your mac where to point at i.e. 192.168.10.10
3. This is where you register tm.tapmedical.test etc.
4. Now https://tm.tapmedical.test/monitor/routeWorks works
5. In fact, everything doing in test works, and will need to be merged to dev to be deployed

HTTP (Hypertext Transfer Protocol)

1. Communication between client computers and web servers is done by sending HTTP requests and receiving HTTP responses
2. https://www.w3schools.com/whatis/whatis_http.asp

Api call

1. In simple words an **API call** is the process of adding an endpoint to a URL and sending a **request** to a server. This is known as making an **API call**. Let's take an example. When you use an app or ask a question via a browser, you are actually making an **API call**.

API routes/Endpoints

1. https://smartbear.com/learn/performance-monitoring/api-endpoints/
2. https://stackoverflow.com/questions/2122604/what-is-an-endpoint
3. **API: Application Programming Interface**
4. APIs allow two systems to communicate with one another.
5. An API provides the language and contract for how two systems interact.
6. APIs can use HTTP requests to get info from a web app or server.
7. REST is more lightweight, using URLs to receive or send info.
8. REST uses 4 HTTP verbs (GET, POST, PUT, DELETE) to perform tasks.
9. Endpoints = functions available through the API, can include an url.
10. Route: “name” you use to access endpoints used in the URL.
11. Each endpoint is the location from which APIs can access the resources they need to carry out their function.

Routes convention and RESTful Controllers (example)

1. When you show an article, load a view called **show** (and the controller method is also show)
2. When you list a collection of articles, we load a view called **index** (and the controller method is also index)

3. This follows CRUD (so much of the websites we create) -> create, read, update, delete … or 7 RESTful controller actions…
    a. **REST** is a robust API architecture and **CRUD** is a cycle for keeping records current and permanent. ... Mapping **CRUD** principles to **REST** means understanding that GET, PUT, POST and CREATE, READ, UPDATE, DELETE have striking similarities because the former grouping applies the principles of the latter.
    b. See difference here: https://softwareengineering.stackexchange.com/questions/120716/difference-between-rest-and-crud
    c. **Essentially, REST is high level API, CRUD is primitive operations.**

Bootstrap

1. CSS framework.

What is provisioning

1. All tasks related to deployment and configurations of applications making them ready to use.

SequelPro

1. A fast, easy-to-use Mac DB management application for working with MySQL and MariaDB databases.
2. Many other GUI for mysql etc… (such as Querious2, TablePlus…)

Swagger

1. Generating API document → downside is that we need to keep it maintained manually, and it is not machine-readable.
2. Swagger is a set of rules for a format describing REST APIs. It is both machine and human readable. It can be used to share documentation among product managers, testers and devs.
3. An open-source framework for designing and describing APIs.
4. **Essentially, it is an interactive API documentation platform to let i.e. front end/users know what your API does and what it returns. That is, trying API calls in the browser**
5. https://medium.com/@garrettvorce/getting-started-with-laravel-and-swagger-b14c66f35576
6. http://fmls.tapmedical.test/api/docs#/

Dependencies

1. Basically one software package depends on another one to run.
2. Usually package managers resolve this for you. It is a complex process.
3. I.e. Program X uses library Y.
4. Program modules relying on one another

Nano, Vim etc.

1. They are terminal text editors.
2. Vim is more powerful and tough to master.

Google Analytics

1. Track usage of app/web
2. https://blog.hashvel.com/posts/laravel-google-analytics/
3. https://www.youtube.com/watch?v=Mh66DpKelyo
4. https://www.youtube.com/watch?v=3ll-TV7TcMs&t=629s

GIT

We use bitbucket. You will be given access/account.
Docs: https://git-scm.com/
Downloads: https://git-scm.com/downloads

1. Versioning tool for projects
2. Allows you to switch 'versions' of your project, so if you want to rollback a change, git makes the process easy
3. Enables collaboration between team members, multiple members can work on their own 'branches'
4. Branches enable developers to work on new changes safely without breaking production code

Quick commands:

For committing changes:

1. Add / Delete code to the project
2. git add . **or** git add <filename> if you want to add a specific file in your working / local directory that will be staged to change
3. git commit -m '<your message>'
4. git push

For pulling from remote repo:

1. git pull **or** git fetch

For stashing your new unpushed code whenever your branch is behind from master

1. git stash

Pull request

1. A pull request is a way for you to contribute to a project by making changes to the code of a project, and then asking the repository owner to merge your changes with the current main project.
    a. For pushing to dev, just do a PR and add Nemanja as a reviewer!
2. https://www.youtube.com/watch?v=qjUkCwmdaDo (**REALLY GOOD** link to know how to create a branch, set it upstream and doing pull requests!!!)

Git branch

1. If something is not merged to dev, tell person to **checkout** in a branch if needed
2. For instance, if new changes to /dashboard isn't pushed to dev, one does not have access to https://tm.tapmedical.test/monitor/dashboard, but can use the dev version https://tm.dev.tapmedical.ca/monitor/dashboard

3. To check if there's anything to push to dev. Simply create PR (new checkin kiosk branch to dev) and bitbucket will tell you: no commits on feat/new-checkin-kiosk that aren't on development.
4. All the same login as cp2 superadmin
5. https://www.youtube.com/watch?v=qjUkCwmdaDo (link for creating branches etc.)

ARCHITECTURE

Model-View-Controller:

Model-view controller (MVC) for short, is a software design pattern that Laravel and other popular frameworks use (ASP.NET core, DJango, CakePHP are some). The MVC software design pattern tries to accomplish splitting the logic of the application into three different components, Model VIew and controllers.

Model: Is essentially the structure and the logic of data the application manipulates. For example Tap medical backend utilizes several model classes to represent various data, for instance Users, Appointments and doctors.

Views: This is usually pages, or the GUI for desktop applications. This components is the representation of data. THis can be graphs, a web page, tables json etc. Tap medicals instance it would be the JSON resources that are returned from API calls.

Controllers: Responsible for getting input and producing output. This can be through form submissions in a web page and API calls to an URL. Controllers can also be responsible for validating data,

LARAVEL SPECIFICS

Here are some notes, but refer to Laracasts.com/documentation when needed

Composer

1. A tool for **dependency management in PHP.** Allows you to declare the libraries your project depends on and it will manage (install/update) them for you.
    a. Example: we need an easy way to work with dates ⇒ pull in **Carbon tool (for dates**!)
    b. Other examples: Faker (to create test/dummy data for your application)
        i. Suppose:
            1. You have a project that depends on a number of libraries.
            2. Some of those libraries depend on other libraries.
        ii. Composer:
            1. Enables you to declare the libraries you depend on.
            2. Finds out which versions of which packages can and need to be installed, and installs them (meaning it downloads them into your project).
            3. You can update all your dependencies in one command.

Migrations

1. migration:rollback will undo the previous step in **down()** of whatever was added.
2. migration:fresh will drop all tables and remake them
3. Use php artisan make:migration add_new_whatever, to make changes to existing tables. Once changes are made, you need to migrate the table again using **php artisan migrate**.

Tinker

1. Use php artisan tinker in order to test things (it is a playground)
2. Once you ssh into vagrant and into your code/tapmedical folder, use **php artisan tinker** to launch the tinkering environment.
3. Now you can do things like,
    a. $appointment = new App\Models\Appointment;
    b. $appointment->date = '2020-08-27';
    c. Etc…
    d. $appointment->save();
    e. Now this row is added to the database and can test functions with it and such

Views

1. Essentially just a page displayed to the client.
2. Have a main view, i.e. app.blade.php, where all the HTML, styles, scripts reside in. Otherwise, we need to add these to every SINGLE view, which is not practical. So leverage **layout files (**app.blade.php**)**.

3. In subsequent views, we want only HTML that is unique to our views.

Laravel: Dingo

1. "The Dingo API package is meant to provide you with a set of tools to help you easily and quickly build your own API – content negotiation, multiple auth adapters, API versioning, rate limiting and more!"
2. https://github.com/dingo/api/wiki/Creating-API-Endpoints
3. https://madewithlaravel.com/dingo#:~:text=%22The%20Dingo%20API%20package%20is,%2C%20rate%20limiting%20and%20more!%22
4. https://github.com/dingo/api/wiki/Installation

Difference between web.php and api.php and other routes files

1. To organize your code and routes!
2. For apps use api.php, for websites use web.php
3. https://www.reddit.com/r/laravel/comments/852lls/web_routes_vs_api_routes/
4. Web routes use **web middleware** whereas api routes use **api middleware**
5. Api routes disable some middleware such as csrf_token validation on post. The routes also become /api/{route_name}
6. https://laracasts.com/discuss/channels/laravel/laravel-purpose-and-different-between-webphp-and-apiphp?page=1
7. For instance, in tapmedical we have many routes files, one for new-checkin-kiosk, one for dashboard, api, api2, monitor, marketplace etc.

Laravel: RouteServiceProvider

1. To avoid bunching all of our routes in the same file in a large project where you would need to group() every few lines of code into groups for specific middleware (i.e. only those routes that are accessible when authenticated), make a separate route file (or folders with files). Now, if for instance you want to know which groups of routes need authentication to access, you can go to the auth.php. https://www.youtube.com/watch?v=lUXnIEa7eiA
2. To do so, make your new routes file in the routes folder, then **map** them inside RouteServiceProvider. Notice that by default, we have mapWebRoutes() and mapApiRoutes(). You can create new ones as you please (see pics). If you make a new routes file, you need to map it, otherwise it doesn't exist to the program.
3. Now all of your routes that need, for instance, a certain middleware, are in the same place.
4. You need to create the middleware in **Kernel**, see https://laravel.com/docs/7.x/middleware for how to register middleware in kernel
5. Ex: We use /api/ in tapmedical.test/api/v3/virtual/test because of config/api.php where our API_PREFIX is set to 'prefix' => env('API_PREFIX', 'api'),

```php
<?php

$api = app('Dingo\Api\Routing\Router');

$api->version('v1', ['middleware' => ['api3']], function ($api) {
    $api->group(['prefix' => 'v3/virtual'], function () use ($api) {
        $api->get('/test', 'App\Http\Api\V3\Controllers\AppointmentController@show');
    });
});
```

```php
//TEMPORARY FOR API3
protected function mapApi3Routes()
{
    Route::group([
        'middleware' => 'api3',
        'namespace'  => $this->namespace."\\Api",
        'prefix'     => 'api',
        'as'         => 'api',
          'guard' => 'api',
    ], function ($router) {
        require base_path('routes/api3.php');
    });
}
```

```php
public function map(Request $request)
{
    $this->mapWebRoutes($request);

    $this->mapApiRoutes();

    $this->mapApi2Routes();

    //TEMPORARY FOR API3
    $this->mapApi3Routes();

    $this->mapAdminRoutes();
```

Laravel: Observers

1. https://www.youtube.com/watch?v=7GUaH6BI_V0&t=56s

Laravel: middleware

1. They are a series of “layers” HTTP requests must pass through before they hit your application!!
2. Must be registered in kernel
3. Can have groups
4. Can have default for all http requests to your app(global)
5. Can run them in order if needed etc.
6. See https://laravel.com/docs/7.x/middleware

Laravel: Task scheduling

1. https://www.youtube.com/watch?v=x7yQkQMp7Lw
2. Essentially, you could create commands where that command handles something, and you could register this in **kernel.php** of **console** in order to execute the command every X time (i.e. every 5 minutes, everyday, under certain conditions etc.).
3. https://laravel.com/docs/7.x/scheduling

Laravel: Command

1. Ex: In Tap, every 5 minutes, a command will force us to grab all the upcoming appointments of the day, and for each and every one of them, fire off the event which is broadcasted. This event calculates both time and position of the user who checked in using the virtual line. So now every five minute, the front end will know about the changes and can retrieve this information to display it to the user.

Laravel: Logging

1. https://laravel.com/docs/7.x/logging
2. Basically, log info regarding what is happening in the application on different channels.

Laravel: Mail

1. You may want to send emails at different stages in your application. For instance, when the user wants to report an error.
2. Create a mailable
3. Make what you want to send in build()
4. Create a view that receivers will see
5. Make the endpoint to send it
6. https://laravel.com/docs/7.x/mail

Laravel: Throttle/model binding

1. To limit functions/api calls to a certain IP within a certain time limit
    a. I.e. 'throttle: 600,1' means 600 access within 1 minute
2. https://www.cloudways.com/blog/laravel-and-api-rate-limiting/
3. Binding (route-model binding): https://stackoverflow.com/questions/51612727/what-does-bindings-middleware-do-in-laravel-5-6

Tap Medical: one API call

1. Specifically for the virtual line, we will use **one** api call to return all the appointment information, all this information will be stored in the front end. FE will hold an array of fetched appointments, and will display one set of data for the appointments in the list, and another set of data for the details tab. This way, we do not lose extra seconds to make another api call when looking at appointment details! Previously, we had an @index to display all appointments and a @show which is fired when you click on view appointment.

Listener, events, sockets, broadcasting

1. https://www.youtube.com/watch?v=1qIsWjajVts
2. https://www.youtube.com/watch?v=2PTgJwxf6UI
3. Flow
    a. I.e. everyone who watches the channel on their TV will see the "broadcast" (ex: a commercial) right away.
    b. The issue is, the server won't send any pages to the client after the client has loaded their page from their request.
    c. So we need a Socket server, it keeps a tab on the client side.
    d. So the Socket server will push down new info to anyone who is tuned in to these channels.

e. Now let's say something happens on the server (i.e. a transaction), the Web server will trigger the event and it's going to publish/broadcast over to Socket server (Web tells Socket server hey, I just received a new comment on winter sport channel, all the people who are listening will receive the news).
f. Now the socket server cannot update the client side. We do it by using a client side service (Echo!).
g. So basically Laravel Echo manages updating the page at this point.
h. Laravel Echo is what receives info from the socket server and echo is running on every single client.
i. Once Echo receives it (in JSON form), passes it to JS and will use those techs to display.

Laravel ::

1. (the :: operator) https://stackoverflow.com/questions/39198357/what-does-double-colon-in-laravel-means
    a. To access static, constant and overridden properties of method of a class

Laravel: foreign keys

1. It is to link tables, save space! No need to repeat/have redundant info
2. https://laracasts.com/series/laravel-6-from-scratch/episodes/30?autoplay=true (BEST ONE)
    a. REMEMBER, FK won't work if your relationship method names are wrong (i.e. not convention). Since laravel assumes user_id, and not, for instance, author_id.
3. https://stackoverflow.com/questions/36964427/foreign-key-or-no-foreign-key-defining-laravel-relationships
4. https://www.linkedin.com/pulse/importance-foreign-key-constraint-tim-miles/
5. Ex: If the user attempts to enter a record in the Orders table using a CustomerID that is not found in the Customers table, the database will reject that entry and display a warning message.

Laravel: not using primary key

1. If you want to use something else as key, i.e. not id, that is inside of /articles/{articles} and passing id to a controller that takes for instances myMethod($id) or myMethod(Article $article), you can override public function **getRouteKeyName()** to return another column. This will make Laravel do a query like Article::where('slug', $article)->first();
    a. https://laracasts.com/series/laravel-6-from-scratch/episodes/26?autoplay=true ~4:00

Laravel Eloquent (really good)

1. https://laracasts.com/series/laravel-6-from-scratch/episodes/29?autoplay=true
    a. I.e. $user->projects from a one to many relationship will return a projects collection that you can iterate over.

2. Q: how do they all link, like how do I create many Articles as one user?
    a. https://laracasts.com/series/laravel-6-from-scratch/episodes/30?autoplay=true
    b. A: using foreign keys https://laracasts.com/series/laravel-6-from-scratch/episodes/30?autoplay=true
3. Remember, although articles() is a method inside of user model, when you call it, you call it as a **property** i.e. $user->articles;
4. **REMEMBER, FK won't work if your relationship method names are wrong (i.e. not convention). Since laravel assumes user_id, and not, for instance, author_id. Unless you set the second argument as the id key… like**
    a. Return $this->belongsTo(User::class, 'user_id'); inside of public function author() https://laracasts.com/series/laravel-6-from-scratch/episodes/30?autoplay=true

Laravel: Many-to-many relationship

1. I.e. an article has many tags, a tag has many articles…
2. We need 3 tables for this, one on each side of the equation and an intermediate/pivot/linking table
    a. Convention for this linking table is singular words, underscore, and in alphabetical order (i.e. article_tag).
    b. In this table, we need a connection for the two tables (articles and tags)
    c. In this table, we need a unique key (combination of article_id and tag_id is unique), so you don't end up in a situation where you have duplicates.
    d. Remember, inside for instance, Article Model, you have
    e.
```php
public function tags()
{
    return $this->belongsToMany(Tag::class);
}
```
    f. https://laracasts.com/series/laravel-6-from-scratch/episodes/31
    g.
```php
public function up()
{
    Schema::create('tags', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('name');
        $table->timestamps();
    });

    Schema::create('article_tag', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->unsignedBigInteger('article_id');
        $table->unsignedBigInteger('tag_id');
        $table->timestamps();

        $table->unique(['article_id', 'tag_id']);

        $table->foreign('article_id')->references('id')->on('articles')->onDelete('cascade');
        $table->foreign('tag_id')->references('id')->on('tags')->onDelete('cascade');
    });
}
```
    h. Can use $article->tags->pluck('name'), this will return only the names of the tags (without id, created at etc.)

i. https://laracasts.com/series/laravel-6-from-scratch/episodes/32?autoplay=true (this is when we click on tags, to show all articles with the tag) → very good!!!
j. **PIVOT:** https://youtu.be/akKWC_vP7sE this video shows withPivot, basically meaning that you need to add withPivot([]); array of column names in the pivot table inside of relationship function to access those columns using keyword pivot (i.e $role->pivot->name)

Laravel time formatting

1. time/dateTime type in SQL db are already in 24 hr format
2. https://dev.mysql.com/doc/refman/8.0/en/datetime.html
3. To reformat it → https://ideone.com/XdX9oL

Laravel responses

1. https://www.youtube.com/watch?v=3xfIbE4KjDY
2. Or see official documentation

Laravel wildcard routes

1. https://laracasts.com/series/laravel-6-from-scratch/episodes/19?autoplay=true (very good resource)

Testing the kiosk

1. Create appointment on -> https://fmls.telus.dev.tapmedical.ca/cp2/signin
    a. Try on -> https://fmls.telus.dev.tapmedical.ca/new-checkin/
2. Use http://fmls.tapmedical.test/api/checkin/upcoming to see upcoming database for appointments of the day

Laravel: mass assignment

1. Laravel protects you from mass assignment…
2. This is why, you need to declare fillables in your model
    a. Protected $fillable = [‘title’, ‘excerpt’, ‘body’];
3. This is important because for instance, you don’t want User::create(request()->all())
    a. Meaning that i.e. if the user has limited access or is a paid subscriber etc., you don’t want to allow the user to change that. I.e. guest can pass [‘subscriber’ => true]
4. Or use protected $guarded = [];, this basically is the reverse of $fillable, i.e. what do you want to protect?

Laravel: Validation

1. See corresponding section in laracast.com

Laravel command: php artisan route:list

1. Shows all listed routes, methods, uri, middleware etc…

Laravel: auth()

1. To print logged in user name do {{ auth()->user()->name }} or {{ Auth::user()->name }}
2. https://laracasts.com/series/laravel-6-from-scratch/episodes/34?autoplay=true

Laravel: pw reset flow

1. https://laracasts.com/series/laravel-6-from-scratch/episodes/35?autoplay=true

Laravel: attach()/detach() methods

1. https://laravel.com/docs/7.x/eloquent-relationships
2. Basically, to join many to many relationship models

Laravel: $this

1. $this refers to the current file

Laravel: tip

1. When a lot of methods have the same code, just make a method of it instead for reusability

Laravel: named routes

1. Using named routes prevents you from needing to change all hard coded uri when one is changed.
2. Use proper convention tho → name of something.methodName
    a. I.e. Route::get('/articles/{articles}', ArticlesController@show) -> name('articles.show;);
    b. Now in front end/blade, you can do {{ route('articles.show', $articles) }} //notice second argument is wildcard
    c. Or can do {{ route('articles.index', ['tag' => $tag->name]}}, here, the second argument is the query string

Laravel: Collections

1. https://laracasts.com/series/laravel-6-from-scratch/episodes/36?autoplay=true
2. I.e. App\Article::all(); or App\Article::first()->tags; → you get a collection of articles.
3. Now you can play with it like $tags->where('name', 'laravel'), or $tags->first() etc.
4. See collection methods! (a lot of them)
    a. I.e. Can collect array using **collect(['one', 'two', 'three']);**

Laravel: with()//Eagerloading

1. $articles = App\Article::with('tags')->get();, this returns all the articles and their associated tags!! Wowzers, this basically eagerloads the relationship with "tags" (so basically tags() inside of Article model)

Laravel: CSRF

1. https://laracasts.com/series/laravel-6-from-scratch/episodes/37?autoplay=true

2. Wherever there is a form that submits a POST, PUT, DELETE request, make it a habit to include @csrf...

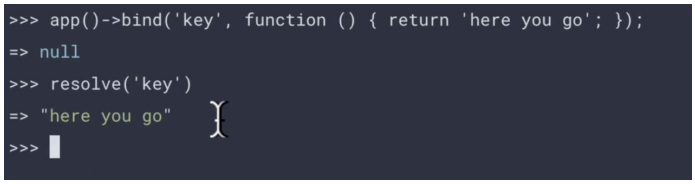
Laravel: Response.php

1. You can find all the error codes here in your project

Laravel: Facades

1. https://laracasts.com/series/laravel-6-from-scratch/episodes/40?autoplay=true

Laravel: service container / providers

1. Bedrock of the framework, a container of services (stores services), a place to retrieve and store services…
2. **Containers** basics: https://laracasts.com/series/laravel-6-from-scratch/episodes/38?autoplay=true
3. **Providers:** https://laracasts.com/series/laravel-6-from-scratch/episodes/39?autoplay=true
4. 

    a. Here you bind a key into the container, and resolve it

Singleton:

1. One and one only object. Even if you create it multiple times, it's still the same. It is the instantiation of a class to one "single" instance.

Laravel: Faker & factory & onDelete('cascade')

1. Dummy rows in DB, see link to know how to use in tinker (factory)
2. To make dummy factory → Php artisan make:factory
3. https://laracasts.com/series/laravel-6-from-scratch/episodes/30?autoplay=true
4. **Real data**
    a. For our products such as kiosks, we should always use the production version and with fake data to mimic function/flows. Don't use real data as may leak/mess up info etc. (i.e. info from actual clinic).

Laravel: Mail/mailtrap.io/Notifications

1. https://laracasts.com/series/laravel-6-from-scratch/episodes/42
2. Mailtrap.io https://laracasts.com/series/laravel-6-from-scratch/episodes/43?autoplay=true
3. Notif https://laracasts.com/series/laravel-6-from-scratch/episodes/46?autoplay=true

Laravel: Database notifications

1. https://laracasts.com/series/laravel-6-from-scratch/episodes/47?autoplay=true

Laravel: RouteServiceProvider

1. https://medium.com/@thesourav/organize-your-laravel-routes-for-better-and-maintainable-code-4ad9b76aed0f

Blade: select multiple tags for a post

1. https://laracasts.com/series/laravel-6-from-scratch/episodes/33?autoplay=true

Blade: forelse loop

1. Basically foreach loop with an else statement (i.e. if exists, do the loop, else do something else).

MSCL. LINKS

- **Backend:** https://www.youtube.com/watch?v=cbSrsYiRamo involves backend lang + sql (plugs in the wall)
- **Api:** https://youtu.be/s7wmiS2mSXY
- **PHP FIX:** https://www.youtube.com/watch?v=vT5luUaAcQ8
- **Apache down fix:** https://www.youtube.com/watch?v=GxSlL3kjSac
- **PHP summary:** https://www.youtube.com/watch?v=ubfxi21M1vQ
- **Websocket review:** https://www.youtube.com/watch?v=vQjiN8Qgs3c
- **Microsoft intern:** https://www.youtube.com/watch?v=1NKGJI_r1e8
- **Broadcasting:** https://www.youtube.com/watch?v=1qlsWjajVts
- **Vagrant + Laravel:** https://stackoverflow.com/questions/34745295/a-vagrant-environment-or-target-machine-is-required
- **Laravel Event firing:** https://www.youtube.com/watch?v=7GUaH6BI_V0
- **Difference between** (c/c++/c#): https://youtu.be/zGrTT4k1-yc