Simple for me to understand

A) Blade::

- go to views and create a new file (master) : app.blade.php (here we will have html5 full --)

<html>
<head>
<link rel = “stylesheet” href = “https://www.getbootstrap.com/... “>
</head>
<title> :: </title>

<body>

@yield('content') //i want my content to be yeilded here

</body>

Inside views eg. Home.blade.php files:

@extends(‘app’) //meaning we will extend to the app master page

So we want the content to be used:

@section(‘content’) or for any divisions we can give name like this

@stop

@section(‘footer’)

<script> alert(‘hello world’); </script>

@stop

B) Include css:

- inside master blade page:

//css file needs to be inside public/assests/css .. if we want the different location then we can give

<link rel = “stylesheet” href = {{ asset(‘assets/css/bootstrap.css’) }} />

Or

<href = {{ asset(‘/css/your_name.css’) }}

c) doing simple conditional blade:

```
$people =  [ ‘ram’ , ‘krishna’ , ‘shyam’]; // then return view

return view(‘pages.contact’ , compact(‘people’);
```

//inside our pages.contact:

```
@if (count($people))

<ul>  @foreach ($people as $person)

<li> {{  $person }} </li>

@endforeach

</ul>


@endif
```

C) Environment:

- have a look at .env and config folder once
- we need not give the host name and db password in config
- we can easily get it throught .env file and remember .env is ignored inside .gitignore file
- inside config/database.php u can easily choose the driver for your database either mysql , postgresql , or sqlite
- Choose the default one
- Inside the .env file give the name of the database , username and the password

D) Migration:

- it’s like the version control for db
- We need not extract the db and then send it to others ..
- There are two methods up and down

- Run existing migration: php artisan migrate (default given migration: users)
- Eg. I want to change column user to username:
    - First roll back : php artisan migrate:rollback

```
Change the value to username : see below
$table->string('username'); //which was previously
string('name');
```

    - Then: php artisan migrate

Creating New Mirgration:

- U can see the help as: php artisan help make:migration (can check help for any things like this)
- Make table:
    - Syntax: php artisan make:migration (name _of_migration) –create=”(table_name)”
    - php artisan make:migration create_articles_table --create="articles" // table = “name” for existing table
- u can see this migration just made inside database/migrations
- Let’s do one table for articles:: (it’s similar how we used in PHPMyAdmin)

```
Schema::create('articles', function(Blueprint $table)
{
  $table->increments('id');
  $table->string('title');
  $table->text('body');
  $table->timestamps();
  $table->timestamp('published_at');

});
```

- For Down method use this:

```
public function down()
{
   Schema::drop(name_table);
}
```

    -

- Then: migrate it using: php artisan migrate

- U want to add new column to that table:
    - php artisan make:migration add_excerpt_to_articles_table --table=”articles”

E) Eloquent:

- active record implementation of laravel
- If I have Articles table then eloquent model is called “Article” , if (table Users , model is user)
- So , let’s make model article:
    - Php artisan make:model Article // since Article is model name , the migration directly takes place , coz it easily gets the table Articles from our DB
    - You will see this model as the child of the App directory ie. App/Article.php (you will get it)
- Tinker: it gives good command line interface , for working with Laravel codebase , we can do general php and more…
- How to assign values to the column of table:
- First create a new object:
    - $article = new App\Article;
    - $article->title = ‘My First Article’;
    - $article->body = ‘Lorem ipsum …. ‘;
    - //don’t worry for timestamp
    - $article->published_at = Carbon\Carbon::now(); //Carbon is library used by laravel
- You can view that using $article->toarray();
- Saving the data to database:
    - $article->save();
    - //returns true if correct
- Fetching all the data:
    - App\Article::all->toArray(); //fetch everything of Article model i.e Articles table and give it to array;
- If you want to update it: simple overwrite in the given object:
    - $article->title = ‘new title’;
    - $article->save();
    - $article->toArray();

- **Selecting from table using eloquent (SELECT * FROM with id)**
    - $article = App\Article::find(1); //find the data with id 1 from the Model
    - $article->toArray(); //see the result

- **Select from table with variable value (SELECT FROM table with value = ‘kkjk’)**
    - $article = App\Article::where(‘body’ ,’this is my body’)->get(); // I want to get the result
    - I get the collection with this

- **Select from table with variable the very first result:**
    - $article = App\Article::where(‘body’ ,’jkjkjk’)->first();

Faster Eloquent Way:

- **Saving the data :**
- $article = App\Article::create([‘title’ => ‘my title’ , ‘body’ => ‘my body’ , ‘published_at’ => Carbon\Carbon::now()]);   //we send the data as a associative array
- //Mass Assignment this is very very important thing Laravel is protecting us so just see it :P
- So go to App\Article.php , we will make some code hereby:
    - Eg. Protected $fillable = [ ]; //array is created , which defines which attributes can be mass assigned //fillable field for the article or which attribute I am ok being mass assigned , id cant be included similarly user_id

```
protected  $fillable = [
    'title',
    'body',
    'published_at'
];
```

Update in Faster Way:

- Get the data:
    - $article = App\Article::find(2); //get the data with id = 2;
    - $article->body = ‘updated one’;
- Next method (similar to create): //first u need to get the id
    - $article->update([‘body’ => ‘updated body’]);

F) Model View Controller Workflow:

- We are working with the articles:

- **Route:**

```
//For our Articles
Route::get('articles' , 'ArticlesController@index');
We need to have articlescontroller
```

- **Controller:**

php artisan make: controller ArticlesController –plain

- Inside the controller:
  - Import model Article on top by ::> **use App\Article;**

```
public function index()
{
    $articles = Article::all();

    return $articles;
}
```

- in the above case JSON returns all the parsed data … so we want to make it clean …. Check this we will updating the same index function inside the ArticlesController

- **So we will pass this data to a view ok ?**

```
public function index()
{
    $articles = Article::all();

    return view('articles.index' ,
compact('articles'); //we will send $articles array
to view: Views/Articles/index.blade.php
}
```

- **View:**
- Inside Articles/index.blade.php
  - First extend to master page
  - Use the sections
  -

```
@section('content')
    <h1> Articles </h1>
```

```
    <hr>
   @foreach($articles as $article)

        <article>
            <h2> {{ $article->title }} </h2>
           <div class = "body"> {{ $article->body }}
</div>
        </article>
    @endforeach
    @stop
```

- We need to give link to the titles:

```
<h2><a href="articles/{{ $article->id }}">
    article->title }}</a></h2>
```

- **Passing id through articles :: public/articles/2**
- Go to routes
- ```
  Route::get('articles/{id}' , 'ArticlesController@show');
  ```
- Inside the pages controller new function is to be made:

```
public function show($id)
{
    $article = Article::findorfail($id);
    return
view('articles.show',compact('article'));
}

}
```

- We need to check for the show.blade.php also:

```
@extends('app')

@section('content')


    <h1>{{ $article->title }} </h1>
```
```

```
    <article>
        {{ $article->body }}
    </article>

@stop
```

G) Creating new Article (use of form and save data)

- Route:

```
Route::get('articles/create' ,
'ArticlesController@create');
```

- Controller:

```
public  function create()
{
    return view('articles.create');
}
```

Giving Link to any pages:

```
<a href={{ action('ArticlesController@create') }}>Create</a>
<a href={{ action('PagesController@contact') }}>Contact Me! </a>
```

- **Form Package: html**
- composer require illuminate/html (illumate is used by laravel itself)
- you can check these components at: github.com/illuminate/html

- **Service Providers:**
    - Go to config/app.php … there is big list of service providers
    - Add to list

```
'Illuminate\Html\HtmlServiceProvider',
```

- Aliases:
  - Scroll down to Aliases list :
  - Aliases gives the sort of global way to access the namespaces
  - Add new item for Form and HTML to last:

```
'Form'      => 'Illuminate\Html\FormFacade',
'Html'     => 'Illuminate\Html\HtmlFacade',
```

- **Form:**
  - **Opening and closing a form:**

After using these things now we can work on with form so lets just try:

- In the beginning the after opening the form we see that it sends request to the same page:

```
{!! Form::open() !!}

{!! Form::close() !!}
```

  - **Adding a nice form:**

```
{!! Form::open() !!}
  <div class="form-group">
  {!! Form::label('title' ,'Title:') !!}
  {!! Form::text('title' ,null , ['class' => 'form-control'
, 'placeholder' => 'Give the title' ,]) !!} //null means
value , name is first one
  </div>

  <div class="form-group">
      {!! Form::label('body' ,'Body:') !!}
      {!! Form::textarea('body' , null , ['class' => 'form-
control']) !!}
  </div>

  <div class="form-group"> //no nid 2 give name for btn
      {!! Form::submit('Write Article' , ['class' => 'btn
btn-primary form-control']) !!}
  </div>


{!! Form::close() !!}
```

Posting data:

Since we are posting the data we need to now do something let's check it very easy

- **Routes:**

```
Route::post('articles' , 'ArticlesController@store'); //send post req to articles page
```

But our form by default sends the req to current page, so we will have to change it also:

Inside the ArticlesController we need a method to show the data passed by the form:

Firstly add: @ top of Controller (ArticlesController)

```
use Request;
```

View all the data:

```
public function store()
{
    $input = Request::all(); //this gives all
    $title = Request::get('title');
    return $input;
}
```

Changing the form Action:

```
{!! Form::open(['url' => 'articles']) !!}
```

Creating a new Post:

```
public  function store()
{
    $input = Request::all();
    $input['published_at'] = Carbon::now();

    Article::create($input); //we had other ways also
```

```
    return redirect('articles');
}
```

View the latest one: inside index function

```
$articles = Article::latest()->get();
```

H) Dates , Mutators and Scopes:

Delete this line from the controller because we need to give date and time differently:

```
$input['published_at'] = Carbon::now();
```

a)Add Date to the form:

Inside create.blade.php (within our form)

```
<div class = "form-group">
    {!! Form::label('published_at' , 'Published On:' ) !!}
    {!! Form::input('date' ,'published_at', date('Y-m-d') ,
['class' => 'form-control']) !!}
</div>
```

This gives the date of which the article was published but we need time also. We don't know this now. Accessors and mutators allows us to manipulate the data before they are inserted into the database or after retrived.

Mutators:

Let's add a new method:

```
    // set followed by field name  eg. setNameAttribute
or setAddressAttribute
    public function setPublishedAtAttribute($date)
    {
```

```
       $this->atttributes['published_at'] =
Carbon::createFromFormat('Y-m-d' , $date);
Or
Carbon::parse('Y-m-d' , $date); //which can b used for future
one , it wont show the time

    }

}
```

So after doing this what we can see is that the future posts also will be shown, we assume that we keep future dates:

Limiting the Future posts:

We can do that using our controller:

We need to get the articles where **published_ at** is less than or equal to current now time.

```
$articles = Article::latest('published_at')->where('published_at' ,
'<=' , Carbon::now())->get();
```

Scope:

But still there is problem. Using this same long code for time and again is a problem so we can use scope by which we can reduce our code. We take the query add clause to it and use it in eloquent model.

Since I want the latest articles that have been published so I can do this as:

Change the code at Controller:

```
$articles = Article::latest('published_at')->published()->get();
```

We need a scope called published() , let’s open our Model add new query scope.

```php
public function scopepublished($query)
{
    $query->where('published_at' , '<=' , Carbon::now());
}
```

Similarly the scope for the unpublished Articles can be fetched easily:

```php
public function scopeUnpublished($query)
{
    $query->where('published_at' , '>' , Carbon::now());
}
```

Dates:

Using carbon very easily we can play with the date and time. Inside the method show. Since Carbon was used for created at this works.

```php
dd($article->created_at);
```

or get year month day

```php
dd($article->created_at->year);
```

Manipulate it: you can also add the days to it:

```php
dd($article->created_at->addDays(55));
```

also can format it as:

($article->created_at->format('Y-m'); //it's all your choice

Human Readable form:

```php
dd($article->created_at->addDays(-4)->diffForHumans());
```

For the things that I saved manually we need to change it using Carbon.

For this just open the Model Articles.php and then add the method:

I) Validation:

Requests:

Since we save the data from the store() function inside the ArticlesController so we need to make some changes in the same function. Currently even if we press the save button so error message is shown on being empty.

We need to create a new Form request since we are working with Form. So run this.

php artisan make:request CreateArticleRequest // location is App\Http\Requests\... .php

- There are two different methods at this class: Authorize and rules
- Authorize means: do the users have permission to make such kinds of requests. Ram cannot edit the comment made by Shyam.
- Currently we don't have membership system so **authorize()** function returns true for now.
- Next we have rules:

```
public function rules()
{
    return [
        'title' => 'required|min:3',
        'body' => 'required',
        'published_at' => 'required|date'
    ];
}
```

- I guess you have easily understood the things
- Since we are working from ArticlesController\Store()

```
public  function
store(Requests\CreateArticleRequest $request)
```

- Inside store just type request name: Create…. It will automatically show options. And name it as $request … selecting Requests\CreateArticleRequest and press alt+enter import the class which now look like:

```
public  function store(CreateArticleRequest
$request)
```

- After doing this what we can see is that before saving the data into database firstly validation will take place
- Understand the working cycle, from create.blade.php requests is passed for creating form which comes to store function which again triggers to our

created request and checks if we can validate or not and then it checks the parameters.

```
public function store(CreateArticleRequest $request)
{
    //validation
    Article::create(Request::all());
    return redirect('articles');
}
```

- If validation fails he will be sent back to the same page
- Since we are using request we no longer need façade , you can remove use Request also:

```
public function store(CreateArticleRequest $request)
{    //validation
    Article::create($request->all());
    return redirect('articles');
}
```

- Currently nothing happens if there is some error in validation so in the view we can display some message:
- Below submit button:

```
@if ($errors->any())
     <ul class="alert alert-danger">

         @foreach($errors->all() as $error)
                 <li> {{ $error }}</li>
             @endforeach

     </ul>
    @endif
```

J) Form Reuse and View Partials:

Resource Route:

- We want to edit the articles , its url needs to be like this: articles/{article}/edit
- Go to routes.php
- Make a new resource route for articles like this:

```
Route::resource('articles' , 'ArticlesController');
```

- This is helpful for all the CRUD based applications
- You can see the list of all the routes by : php artisan route:list
- You can see the list of routes for viewing to articles , editing it , deleting (run command and see right now)
- You need not worry to provide route for viewing , editing for eg.

```
//Route::get('articles' , 'ArticlesController@index');
//Route::get('articles/create' , 'ArticlesController@create');
//Route::get('articles/{id}' , 'ArticlesController@show');
```

Edit Article:

- Create a new method (since route is already made by above list) in controller

```
public function edit($id)
{
    $article = Article::findOrFail($id);

    return view('articles.edit', compact('article'));
}
```

- Article is model
- Make a new view inside articles folder:

```
@extends('app')

@section('content')

    <h1>Edit  <i>{{ $article->title }} </i></h1>

    @include('errors.list')

    {!! Form::model($article , ['method' => 'PATCH' , 'action' =>
['ArticlesController@update' , $article->id ]]) !!}

    @include('articles.form' , ['submitButtonText' => 'Update Article'])

    {!! Form::close() !!}


@stop
```

- You can see above , we use eloquent model for form , method is Patch , we can easily use $article model which is passed from the controller to the view.
- We need to check the update method at our controller:

```
public function update($id , ArticleRequest $request)
{
   $article = Article::findOrFail($id);

   $article->update($request->all());

   return redirect('articles');
}
```

- What we have done is that we have simplified our Request to name ArticleRequest , which can be used in creating and updating
- Update the Request name for method store also and update in top

Form Reuse:

- In the above view u can see we have used method @include( ) which includes the view ,
- Copy the content of the form elements , apart from Form::openI() and close , and store it inside articles/form.blade.php
- Change the submit button like this:
- ```

     {!! Form::submit($submitButtonText , ['class' => 'btn btn-primary form-control']) !!}

  ```
- And in our edit.blade.php
    - ```
      @include('articles.form' , ['submitButtonText' => 'Update Article'])
      ```
    - Meaning we update submitButtonText to Update Article
- Same thing is to be applied for creating new article , only the form method will be different
    - ```
      @include('articles.form' , ['submitButtonText' => 'Add Article'])
      ```
- Errors is saved to list.blade.php inside Errors folder

K) Eloquent Relationship:

- One user can create many articles , and the articles is owned by the user. There is a relationship between these two.
- We have user model , add a new method there:
      ```

```php
/**
 * A user can have many articles
 *
 * @return \Illuminate\Database\Eloquent\Relations\HasMany
 */
public function articles()
{
    return $this->hasMany('App\Article');
}
```

- Similarly we need to represent this relation on Article model also:

```php
/**
 * A user can have many articles
 *
 * @return \Illuminate\Database\Eloquent\Relations\HasMany
 */
public function user()
{
    return $this->belongsto('App\User');
}
```

- But the problem with our create_article migration , we have not linked the article with user , so we need to do this.

```php
$table->integer('user_id')->unsigned();
```

- Meaning unsigned() means the user_id cant be negative.
- We also need to setup foreign constraint , if the user deletes their account , we want to delete all the things that the user own. (we want articles to be deleted).
- So we add a foreign key user_id , which references the id on users table and on Deleting the account we cascade down the articles created by them. (on the same above table)

```php
$table->foreign('user_id')
    ->references('id')
    ->on('users')
    ->onDelete('cascade');
```

- Since we added new things , we need to refresh the migration:
    - php artisan migrate:refresh
- This may give some problems ….. (if your migrations are not managed properly … check the table in PHPMyAdmin , delete unwanted migrations and again run the migration)
- We need some authentication system , for now we used hidden user_id , we don’t want to write it here because we will discuss things in coming time.

K) Authentication System: