Passing multiple data to view

In both cases every key will be transformed in a variable in the view

1)
```
return view(viewname)->with([
    ‘var1’ => value,
    ‘var2’ => value
    ]);
```

2)
```
return view(viewname, compact(‘var1’, ‘var2’))
```

External CSS style sheet

Must be stored in Public/CSS and included in the view blade file as

```
<link href="{{ asset('/css/main.css') }}" media="all" rel="stylesheet" type="text/css" />
```

Protect DB data info in config/database.php

Specify values in the env.php file and then in database.php

```
…
‘host’ => env(‘DB_HOST’, ‘localhost’);
…
```

Edit tables through migration

Create table: `php artisan make:migration create_whatever_table --create=”whatever”`

Editing migration file, rolling back migration then re-make migration is not recommended when there is already data in the DB or in production environment. It is therefore better to do:

```
php artisan make:migration updating_tablename --table=”tablename”
```

new migration file will have main table data already in up method, add to the down method for rollback:

`$table→dropColumn(‘columnName’);` [*might require a driver package → doctrine/dbal?*]

Eloquent

Laravel Active Records Implementation

Model is representation of database: there is one class that represents an associated DB table: table users will have a user model, table products a Product model...

```
php artisan make:model Product
```

Since it extends Model, it inheritss all the methods just like (example using php artisan thinker)

App\Article::all()->toArray(); → *retrieve all from DB and casts into an array*

Create (simple)

- $article = new app\Article;
- $article->title = ‘*title*’;
- $article->body= ‘*body*’;
- $article->save(); → *persist (saves it in the DB)*

Mass assignment →creates and persist

- $article = app\Article::create([‘title’=>’new title’, ‘body’=>’new body’, ‘published_at’=> Carbon\Carbon::now]); *carbon is library for time/date*

  waring! requires in Article model:
  ```
  protected $fillable= [
      ‘title’,
      ‘body’,
      ‘published_at’
      ];
  ```

Select

- $article = App\Article::findOrFail($*id*); *saves work, !need if statement if id !exist*
- $article = App\Article::where(‘*body*’, ‘*Lorem ipsum*’)->get(); → *returns Collection Object!*
- $article = App\Article::where(‘*body*’, ‘*Lorem ipsum*’)->first(); → *returns Article class*

Update record on database:

- $article->title=’*new title*’;
- $article→save();

  OR
- *$article = app\Article::*findOrFail*($id);*
- *$article→update(‘title’=>’new title’);*

  OR BETTER
- *public function update($id, Request $request)*
- *{*
- *$article = app\Article::*findOrFail*($id);*
- *$article→update(‘$request’->all());*
- *}*

Create Model with Artisan → with migration and controller

php artisan make:model Product -m (*migration*) -c (*controller*) -r (*resource*)

creates a Model, a migration and a controller: framework assumes that the table will be called with name of model but small letter, and in plural. Here class Product will be tied to products table and ProductsController controller.

It is possible to specify otherwise, in the model add: adding a protected property $table :

```
class Product extends Model
{
protected $table = ’produtos’;
```

Using form facade → deprecated

1) Require package: composer require illuminate/html

2) *Register Service Provider:*

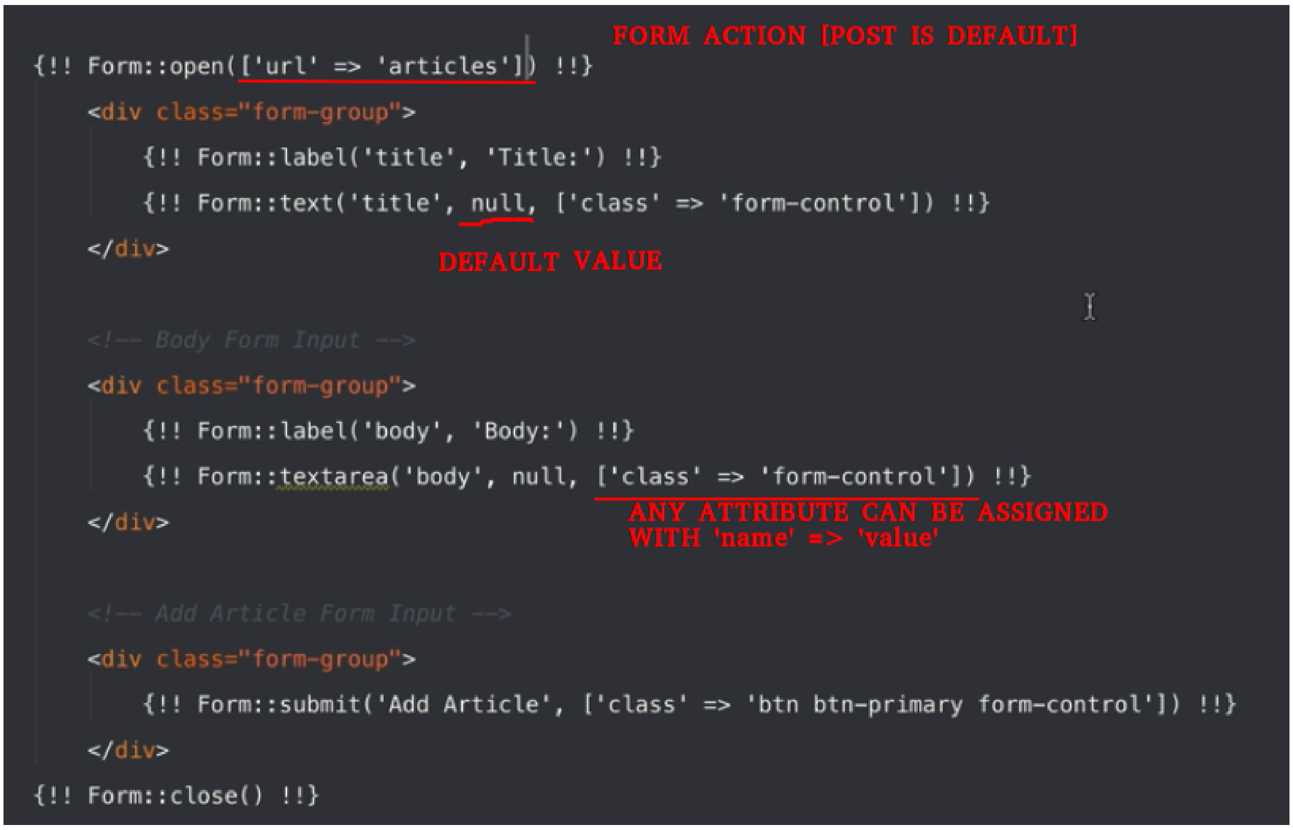
in config/app.php add:

```
...
'providers' => [
…
'Illuminate\Html\HtmlServiceProvider'
```

3) *reference html facade (config/app.php)*

```
…
'aliases' => [
…
'Form' => 'Illuminate\Htlm\FormFacade',
'Html' => 'Illuminate\Htlm\HtmlFacade'
```

Build form with form facade



```
{!! Form::open(['url' => 'articles']) !!}
    <div class="form-group">
        {!! Form::label('title', 'Title:') !!}
        {!! Form::text('title', null, ['class' => 'form-control']) !!}
    </div>

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

    <!-- Add Article Form Input -->
    <div class="form-group">
        {!! Form::submit('Add Article', ['class' => 'btn btn-primary form-control']) !!}
    </div>
{!! Form::close() !!}
```

FORM ACTION [POST IS DEFAULT]
DEFAULT VALUE
ANY ATTRIBUTE CAN BE ASSIGNED WITH 'name' => 'value'

!! for specifying other types [ex.date] of input use following

{!! Form::input('type', 'name', default_value, ['attrName' => 'attrValue']) !!}}

Query scopes

Limit to articles that have been published / what we want to achieve in long code:

```
$articles = Article::latest(‘published_at’)→where(‘published_at’ ‘<=’
Carbon::now())→get()
```

alternative is to create scope:

```
$articles = Article::latest(‘published_at’)→published()→get();
```

in Model create method:

```
public function scopePublished($query){

query->where(‘published_at’ ‘<=’ Carbon::now()

}
```

another possibility

```
public function scopeUnpublished($query){

query->where(‘published_at’ ‘>=’ Carbon::now()

}
```

Using user-friendly time with Carbon

1) tell laravel to treat dates as Carbon instance

in Model create attribute:

`protected $dates = [‘published_at];`

2) it is now possible to access Carbon attributes, for example:

`$article→published_at→diffForHumans();` → *will return ex. “5 hours ago”*

Validation

1) **using FormRequests**

`php artisan make:request CreateArticleRequest` → *will add class to http\Requests*

```
public function authorize();
{
return true; → anyone can make this request
}
```

```
public function rules();
return [
‘title’ => ‘required|min:6’,
…
```

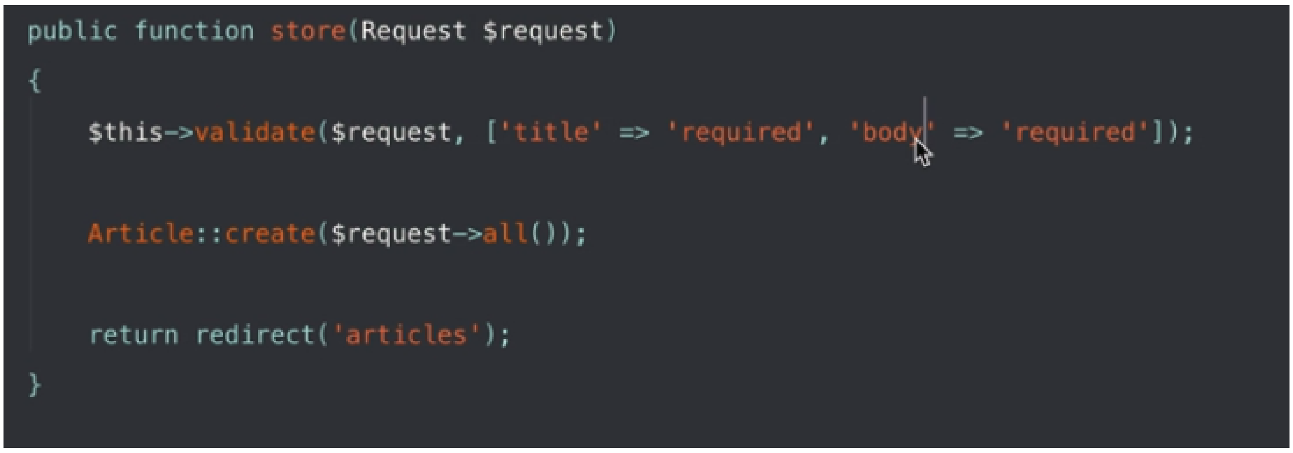
Validation Errors

all views have access to a variable called $errors

example to show errors in a view:

```
@if ($errors→any() )
        @foreach($errors->all() as $error)
        <li>{{$error }}</li>
        @endforeach
@endif
```

2) calling validation method directly in the controller



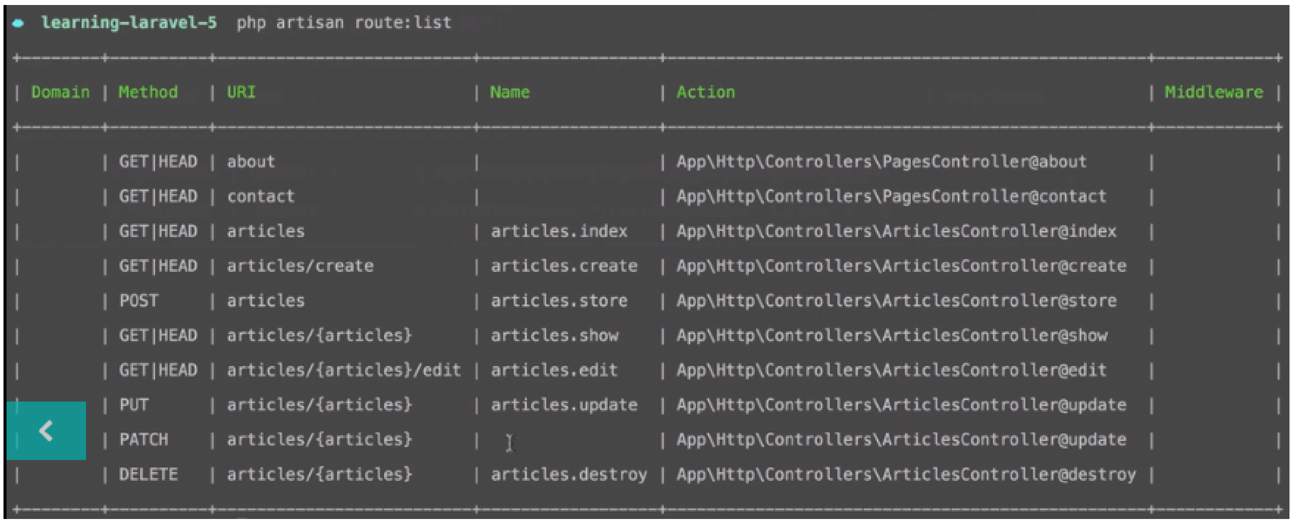
```
public function store(Request $request)
{
    $this->validate($request, ['title' => 'required', 'body' => 'required']);

    Article::create($request->all());

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

Resourceful routing

instead of specifying all the routes individually, it is possible to use resource:

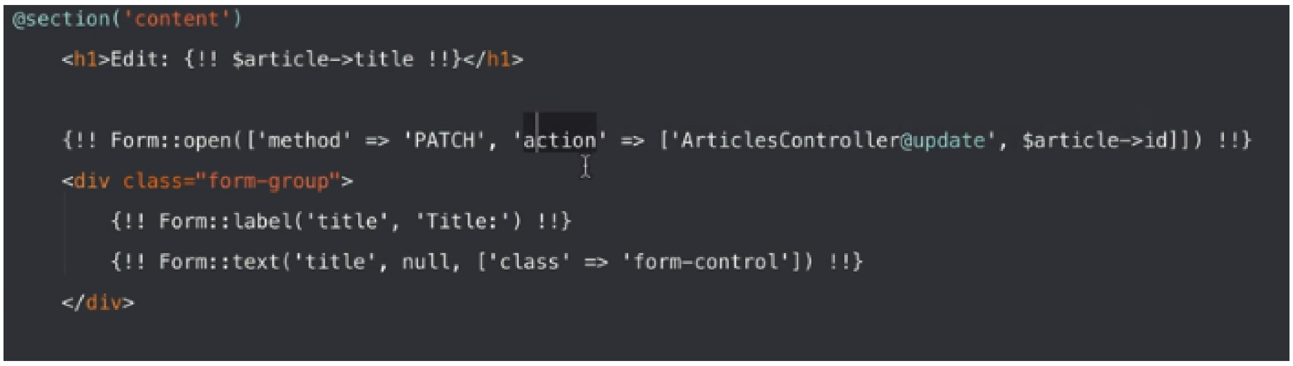
Route::resource(‘articles’, ‘ArticlesController’);

Laravel will automatically generate all the routes following basic REST conventions:



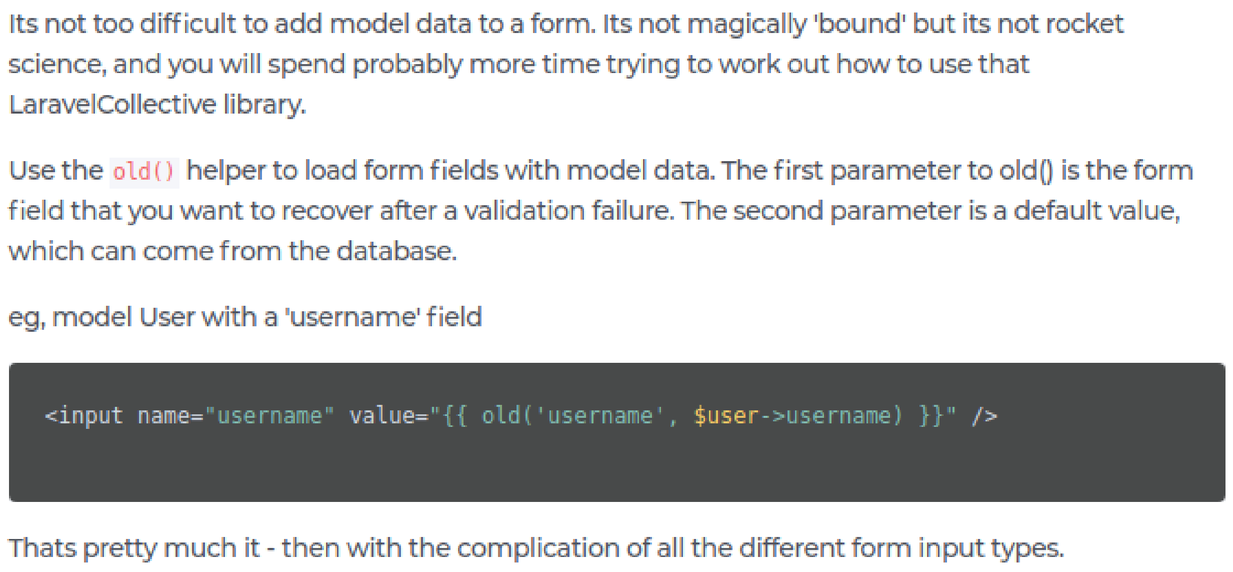
```
• learning-laravel-5  php artisan route:list
+--------+----------+--------------------------+------------------+---------------------------------------------------+------------+
| Domain | Method   | URI                      | Name             | Action                                            | Middleware |
+--------+----------+--------------------------+------------------+---------------------------------------------------+------------+
|        | GET|HEAD | about                    |                  | App\Http\Controllers\PagesController@about        |            |
|        | GET|HEAD | contact                  |                  | App\Http\Controllers\PagesController@contact      |            |
|        | GET|HEAD | articles                 | articles.index   | App\Http\Controllers\ArticlesController@index     |            |
|        | GET|HEAD | articles/create          | articles.create  | App\Http\Controllers\ArticlesController@create    |            |
|        | POST     | articles                 | articles.store   | App\Http\Controllers\ArticlesController@store     |            |
|        | GET|HEAD | articles/{articles}      | articles.show    | App\Http\Controllers\ArticlesController@show      |            |
|        | GET|HEAD | articles/{articles}/edit | articles.edit    | App\Http\Controllers\ArticlesController@edit      |            |
|        | PUT      | articles/{articles}      | articles.update  | App\Http\Controllers\ArticlesController@update    |            |
|        | PATCH    | articles/{articles}      |                  | App\Http\Controllers\ArticlesController@update    |            |
|        | DELETE   | articles/{articles}      | articles.destroy | App\Http\Controllers\ArticlesController@destroy   |            |
+--------+----------+--------------------------+------------------+---------------------------------------------------+------------+
```

Using patch method to update trough form



```
@section('content')
    <h1>Edit: {!! $article->title !!}</h1>

    {!! Form::open(['method' => 'PATCH', 'action' => ['ArticlesController@update', $article->id]]) !!}
    <div class="form-group">
        {!! Form::label('title', 'Title:') !!}
        {!! Form::text('title', null, ['class' => 'form-control']) !!}
    </div>
```

Form-model binding



Its not too difficult to add model data to a form. Its not magically 'bound' but its not rocket science, and you will spend probably more time trying to work out how to use that LaravelCollective library.

Use the `old()` helper to load form fields with model data. The first parameter to old() is the form field that you want to recover after a validation failure. The second parameter is a default value, which can come from the database.

eg, model User with a 'username' field

```
<input name="username" value="{{ old('username', $user->username) }}" />
```

Thats pretty much it - then with the complication of all the different form input types.

Reusing views partials (used in project as components)

@include('folder.viewName')

does not need @stop tag, can be used for forms or errors

to pass variables → @include('folder.viewName', ['VariableName' => 'variableValue'])

Eloquent Relationships