LINE DOCUMENTATION

Line Framework 2.0

ABSTRACT

This Document is created to guide the user of line framework throughout his/her project.

Abnet Kebede

Website Development with PHP

Line Documentation

Contents

Chapter 1

The Framework

1.1. From the Developer

This simple framework is developed for the developer who wants to develop their project using php simply other than using more complex frameworks. this means to use this framework you don't need to know another programming language like other php frameworks. I my-self choose not to learn another programming language other than php to develop software using php framework.

The framework can be used to develop back end and the front end. It also can be used to develop rest API. line can make the API for you if you want to use the default API of the framework. The default API is not fully finished yet but it will be discussed on another chapter on this document. You can use the default API for only read purpose if you want.

Through time there are many things that should be updated in line. By using this version of line framework, you will not have that kind of applicability because there is no updater for this version (version 1.0) but we are working updater for the next versions.

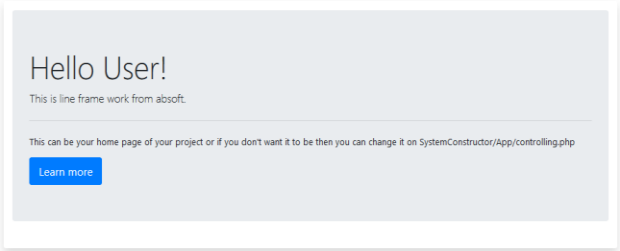
1.2. Preparing the Framework

When you start project the first thing you need to do is to download the folder of line framework which will be the project folder. So, you can rename the project folder to your project name. after doing that you should also rename the project from the system description file. The file name is sys_description.json. it is json file inside there is place for the project name then rename the project from there.

```
{
    "project_name": "framework",
    "version": "version1.0",
    "framework_type": "full",

    "trusted_ip": [
        "localhost",
        "10.240.72.49",
        "::1"
    ]
}
```

1.3. Changing the first page of the project

When you navigate to the http://localhost this will open default page of the framework. You can change this page into your page first page in two ways. The first is you can redirect to the page you created inside the Templates folder.



And the second is you can redirect to the controller you wish to be computed. To do that first you need to find ../SystemConstructor/App/Engines/ViewerEngine.php file under prepare view. Inside this file you should delete the go to address and then write the function circled in blue on the image below. And write the name of the controller.

```
49          if(isset($this->headers->page_name) && isset($this->headers->sub_page)){...}
72          else if(isset($this->headers->home_page)){
73
74              //$this->GO_TO_ADDRESS = $this->_main_address."app/Templates/public.php";
75
76              Route::goRoute( name: "Posts.show");
77              die("");
78
79          }
80
81          if($this->GO_TO_ADDRESS == $this->_main_address."SystemConstructor/App/Templates/error/index.php")
82
83              $this->error_page = true;
84
85          }else{
86              $this->error_page = false;
87          }
88
```

1.4. The front end

We tried to make the development of the front end simpler and easy. To understand the development of the front end you should you should understand the term template. Template is the a php file which can be written using php, html, java script and Cascade Style Sheet. A template can be web page or header or footer for collection of other templates. A template can only be created inside Templates folder. You can create a folder inside the Templates folder and create your template inside the newly created folder. The navigation of this kind of template will be discussed later on this document.

Chapter 2

Line Folder Structure

2. Line Folder Structure

2.1. API Folder

The folder API is on the top of the line framework folder structure in alphabetical order. It consists of two php files named Starter.php and test.php.

Starter.php contains all the inclusion of php files which are used by the API. Which can be user models user controllers built in classes. Test.php include starter.php and contains the logic which used to decompose the request from API user and send it to proxy class inside the SystemConstructor folder. Test.php also the page which the result data will be shown as json file.

2.2. App Folder

Contains app folder contains every thing you will edit to develop you web site. The app folder contains the following folders and files.

2.2.1. Bootstrap

This folder contains two folders js and css. The folder js contains all the java script files used by bootstrap front end framework and other two java script files build by line framework developers. The js folder contains _main.js.bootstrap.mini.js, home.js, jquery3.3.1.min.js and popper.min.js.

The css folder contains bootstrap.min.css file which used by bootstrap frame work and main.css build by line developer.

2.2.2. Controller

This folder contains nothing at first but when you develop your project all the user made controller class will be here. Controller cannot be created at another folder only in controller folder.

2.2.3. DatabaseBuilder

This folder also contains nothing at first but when you develop your project all the users made databasebuilder classes will be here. Database builder is an entity which serve as blueprint for the database table. Database builder class can only be created inside this folder.

2.2.4. Resource folder

The users of line frameworks suggested to put all resources used by the project inside this folder. As the name indicates all the images, videos, audios and other resources used by the line framework should be put in here.

2.2.5. Models

This folder contains nothing at first but when you develop your project all the user made controller class will be here. All models can only be created inside this folder.

2.2.6. Conf JSON file

This file contains all the route address of you project and their parameters as key value pair respectively.

2.2.7. Index php file

This file is your application without this file you cannot run your application. This file is all you see on your application every competition ever made on you application is computed on this file. Don't make no change on this file!

2.3. System Constructor

This folder is the main folder in the line framework. It contains all the built in class which are classified in to two folders which are called App and Line.

2.3.1. App

In this folder there are seven folders called Engines, ErrorHandling, Loaders, Pager, Routing, Security, Templates. In Engines folder there is three php classes Engine, ViewerEngine and ControllingEngine. Engine class contains the logic to receive requests and interpret requests and transfer the request to the appropriate class. If the request is

sent for viewing then the request will be sent to ViewerEngine and if the request is sent to controllers then the request will be sent to the ControllingEngine. then the ControllingEngine will compute the request and include the required user defined controller and model and break down the request and pass the parameters to the appropriate controller.

ErrorHandling folder contains one php file and one php named error_handler.php and ErrorHandler respectively. This file and class are responsible for all error catching and displaying them to the developer.

Loaders folder contains two php file and one php class which are constant_loader.php, app_loader.php and Loader.php. constatnt_loader.php contains all the files inclusion which are necessary in order the framework to work properly. And app_loader.php file includes all the class needed which are in the systemConstractor/app folder. Loader class contains static functions which are needed to load class and resources.

Pager folder contains two classes Alert and Pager. Alert class contains static methods which are responsible for sending alert messages to view and displaying them when ever they called. Pager class contains method which are responsible to limit the data that are shown on the view and dividing them in to many pages and instead of showing them all at once in long page.

Routing contains three php files Route.php and route_map.php. Route.php is php class which contains all the function used for routing purpose. Route_map.php is php file used to map entity with its corresponding controller.

Security folder contains one php file and one php class named ip_check.php and IpCheck respectively. Ip_checker.php contains the logic to check IP of the user of the application whether the IP is trusted or not. Loader class contains all the function which will help the developer to load (to include) the files he wants. IpCheck.php is php class consists of all function used to check client IP whether it is trusted or not.

Templates folder contains all the view of the system including error pages warning page command page of the system which developers see when they create controller, entity and models. And also this folder contains the documentations view of the framework.

2.3.2. Line

This folder contains built in classes in general this folder is the back bone of the frame work. This folder contains 6 folders those are api, Attributes, Creation, Database, Modeling and QureyConstruction. Api folder contains all the classes that are used by the default built in API.

Attributes folder contains all kinds of SQL database attributes classes. Creation folder contains only one php class called Creator which extends controller it controls all requests from the user to create models, controllers and database builders.

Database folder contains all classes used for database connection and also contains database configuration file. Modeling folder contains main class of Controller, Entity, Model, and Schema. Query Construction folder contains all the class which are used for constructing SQL query.

2.4. Templates

This folder contains all the templates. At first this folder contains one folder and one file. System_templates is folder contains all the system templates which are used by the framework. And public.php is template which will be shown as the default page of line framework.

2.5. Other files

Index.php is php file which used to redirect to the developer page. Sys_description.json is json file which describe the project and it allows developers to set trusted IP address so that any other IP address will not be allowed.

Chapter 3

Creating Entity Model and Controller

3. Creating Database Builder, Model, and Controller

There are two ways to create database builder, controller and model. Those are manually and using built in user interface. By using built in user interface there is another two ways you can create model, controller and database builders. Those are using simple interface and by using CLI (Command Line Interface).

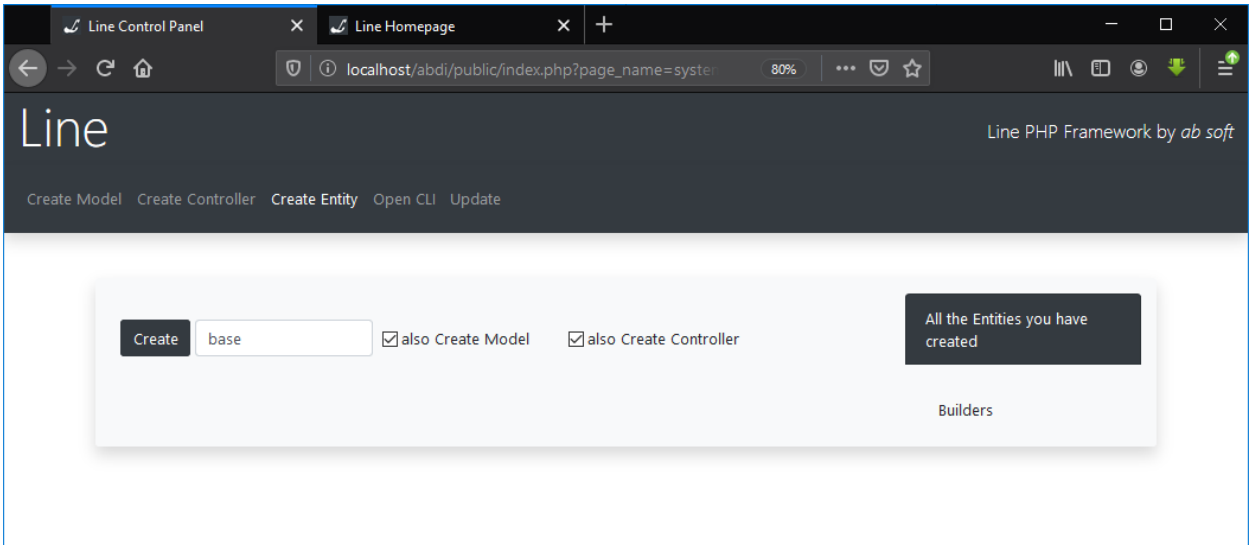
3.1. Creating Database Builder

You can create database builder manually inside the DatabaseBuilder folder. Database builder name should be plural noun and it should start with Capital letter. And the name should not have space. For spacing you can use under score. And all database builders should contain the following.

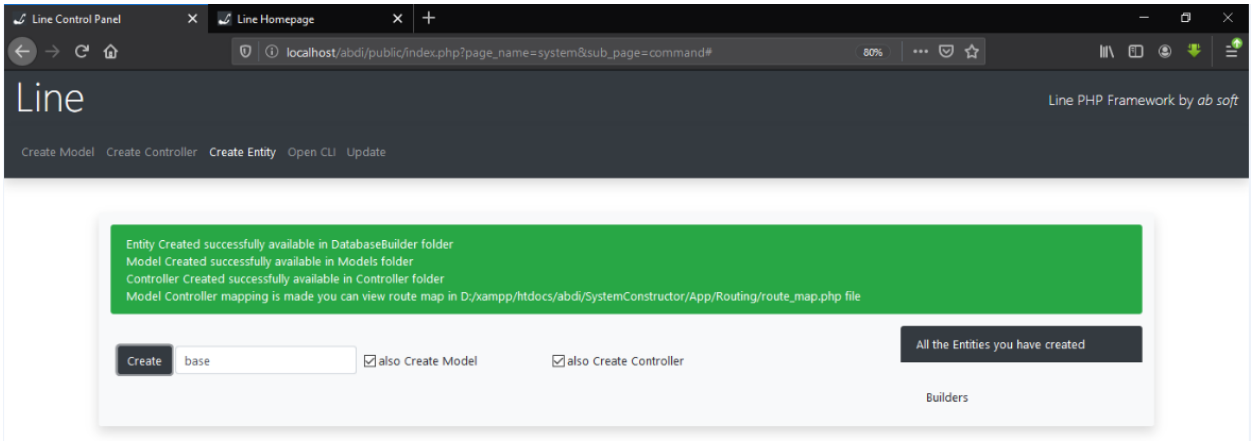
```php

<?php

class Builders extends Entity{

    function construct(Schema $table, $table_name = "Builders"){

        $this->TABLE_NAME = $table_name;

        $this->ATTRIBUTES = [
            /*
             $table->string("name")->length(30)->nullable(boolean)->reference("table_name")->on("id"),
            */
            $table->autoincrement( name: "id"),

        ];

    }

}

?>

```

The other way to create database builder is to use graphical user interface. As the following.

Step 1: navigate the line control panel of your project folder using your favorite browser. For example, in my case my project is running on localhost so the address will look like http://localhost/pages/system/command then the GUI will be shown to you as follows.

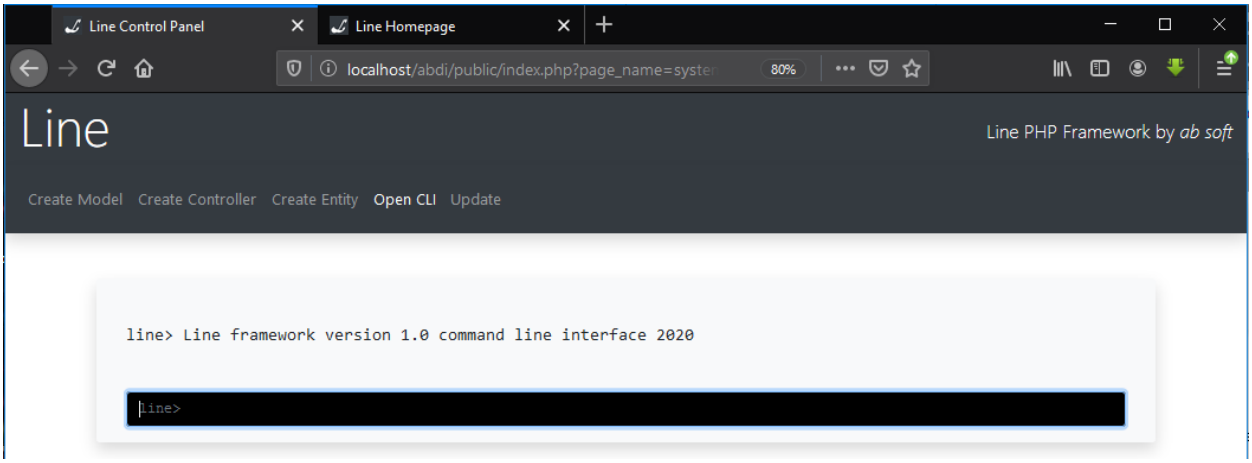


Step 2: select Create Entity from the top bar then set the name of the entity and if you want to create model and controller for the database builder you can select the check box or unselect the check box otherwise. If you select 'also create controller' then entity controller mapping will be done for you automatically.



There is another way to create database builders. It using CLI (Command Line Interface).

Step 1: navigate to the line control panel of your project folder using your favorite browser. For example, in my case my project running on localhost the address will look like this http://localhost/pages/system/command then the GUI will be shown to you as follows.



Step 2:then select Open CLI from the top nav bar. after that write the following command.

`absoft createEntity entity_name`

after creating the base things automatically or manually you should set attributes. In line framework all attributes will extend the class Attribute and class Attribute has many function. Those are:-

```
function getName();
function Reference($table_name);
function getReference();
function on($attribute_name);
function getAttribute();
function getLength();
function length($length);
function nullable($boolean);
function getNullable();
function sign($boolean);
function auto_increment($boolean);
function setPrimaryKey();
```

Some attributes extend all properties and some will not. In line framework there are 9 attributes those are string, autoincrement, text, int, date, time, double, float, timestamp.

Strings extends all of attributes properties and implement all but sing and autoincrement properties. By default, attribute string set to nullable and length 30.

Text extends all of the attribute properties and implement all but sing, autoincrement and set Primary key properties. By default, attribute text is set to nullable and the length is 100.

Int extends all of the attribute properties and implement all. By default, attribute int set to nullable, signed, autoincremented and length 11.

Date extends all of the attribute properties and implements all but length, autoincrement and sign. By default, attribute date set to nullable.

Time extends all of the attribute properties and implements all but length, autoincrement and sing. By default, attribute date is set to nullable.

Double extends all of the attribute properties and implement all but length, autoincrement and primary key. By default, attribute double set to nullable and signed.

Float extends all of the attribute properties and implement all but autoincrement. By default, attribute float set to nullable, signed and length 11.

Timestamp extends all of the attribute properties and implement all but autoincrement, sign and length. By default, attribute timestamp is set to nullable.

To change the name of the primary key you can set the PRIMARY_KEY attribute to whatever your attribute name is. Also, you can change the database attribute DATABASE to the database server you want to set in this version Microsoft SQL server and My SQL server are supported. For My SQL you can set the database attribute to 'mysql' for MS SQL server you can set the Database attribute to 'mssql'.

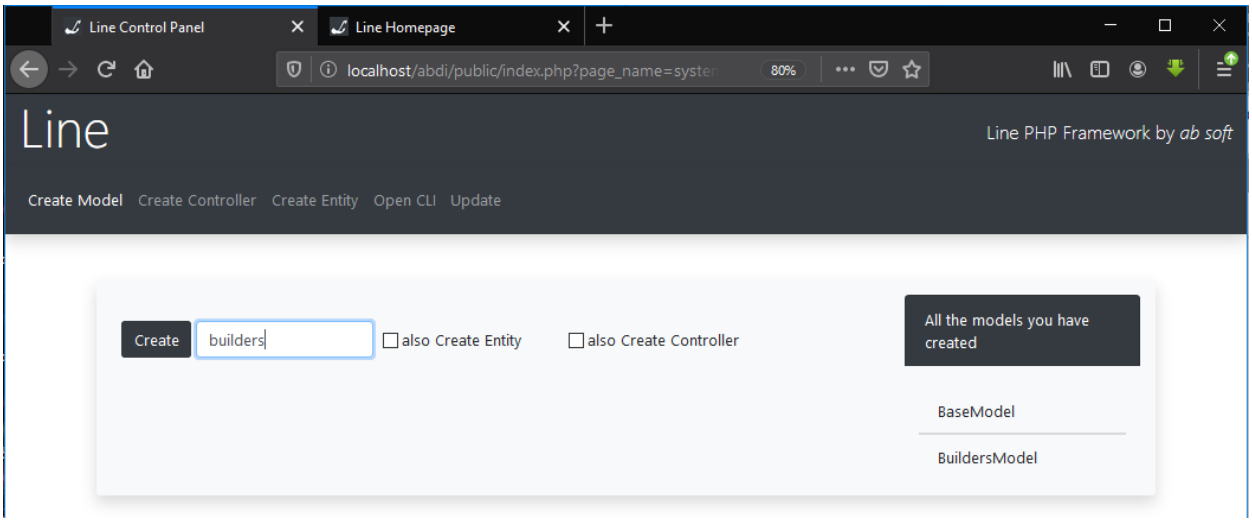
3.2. Creating Models

Like creating Database builders there are three ways you can create models. Those are manually, using GUI (Graphical User Interface) and using CLI (Command Line Interface).

```php
<?php

class BuildersModel extends Model{

    /*
    public $MAINS = [
        "id" => "",
        "username" => "",
        "f_name" => ""
    ];

    */

    public $TABLE_NAME = "Builders";

    public $MAINS = [
        "id" => "",
        "name" => "",
        "age" => ""
    ];

}
?>
```

All models should extend the super class Model. You can create models by using GUI in two steps.

Step 1: navigate to Line Control Panel then select create Model.

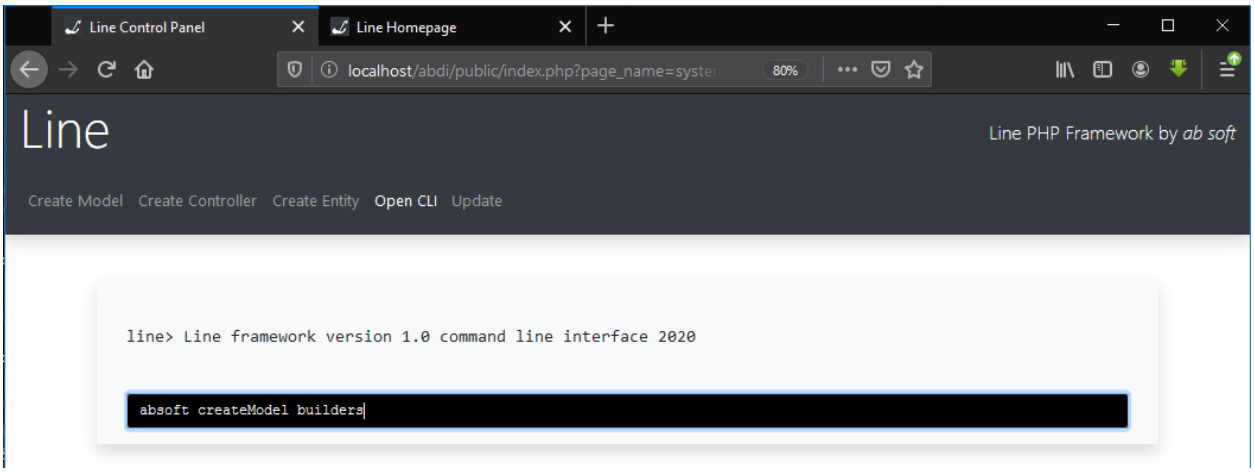


Step 2: set the model name on the space provided and you can also select the check boxes if you want to create the corresponding entity and controller. Then click create automatically it will create the model. You don't need to set the name of the model start with the capital letter because line will do it for you.

You can create models by using CLI in one command.

```
absoft createModel model_name
```

the above command will create the model with the model name. once again you don't need to set the name of the model to start with capital letter line will do that for you.



After creating the model basic things, you should set the attributes as an array manually. Attributes should be the same as the corresponding database builder.

```
15    public $TABLE_NAME = "Builders";
16
17    public $MAINS = [
18        "id" => "",
19        "name" => "",
20        "age" => ""
21    ];
```

3.3. Creating Controllers

The same as the database builders and models we can create the controllers too. Manually, using GUI and using CLI. The new thing here is when you create controllers you should also create mapping between the corresponding Database Builder and the controller.

Line Documentation

```php
<?php

class ReportsController extends Controller{


    public function route($name, $parameter)
    {

        if($name == "show"){

            $return = $this->show();

        }else if($name == "save"){

            $return = $this->save($parameter);

        }
        else {

            $return = Route::display( page_name: "error", sub_page: "index", [
                "title" => "Route Not Found!",
                "description" => "There is no Route named MaterialsController.".$name,
                "error_page" => __FILE__." on Line ".__LINE__
            ]);

        }

        return $return;

    }

    public function show(){

    }

    public function save($request){

    }


}
?>
```

If you want to use CLI you can create with one command. But this time you should create the mapping manually inside ../SystemConstructor/App/Routing/route_map.php.

`absoft createController controller_name`

```php
<?php
/** Created by PhpStorm. ...*/


/*
Route::set("ControllerName", "EntityName");
*/
Route::set("BuildersController", "Builders");
Route::set("BaseController", "Base");
```

Chapter 4

Routing in Line

4. Routing in line

4.1. What is Route Map

Route map is mapping of controller with its corresponding Database Builder so that the framework can understand which model and entity to load automatically for the controller when the application is running. If you ask how it is done since the name of the controller, the model and the Database builder has common beginning name so using php reflection it will be included on the control page where all the controlling request goes to. Routing is done by using the class Route and the route map file found in ../SystemConstructor/App/Routing/route_map.php.

4.2. Class Route

Class Route is concrete class with route map array and many functions helps with the routing process.

```
9   class Route{
10
11      private static $route_map = [];
12
13
14      function __construct(){...}
18
19      public static function route($name, $req_array = []){...}
28
29      public static function routeAddress($name){...}
66
67
68      public static function getController($name){...}
96
97      public static function getModel($name){...}
125
126     public static function get($name){...}
148
149     public static function getModelName($cont_name){...}
162
163
164     public static function display($page_name, $sub_page, $request = array()){...}
184
185     public static function goRoute($name, $req_array = array()){...}
190
191     public static function goRouteAddress($name, $req_array = array()){...}
211
212     public static function viewAddress($page_name, $sub_page, $parameter_array = array()){...}
230
231     public static function view($page_name, $sub_page, $parameter_array = array()){...}
238
239     public static function set($controller_name, $model){...}
244
```

Line Documentation

The first method route returns json encoded string for the engine by computing the given name and request array parameters. The parameter name indicates the name of controller and the method inside the controller. It should be written as 'Controller_name.method_name'. the method route uses the method routeAddress. You should use it to return for the engine inside a controllers method.

The method routeAddress returns the address of the routing by computing the given name and then it will return associative array of header for the route method. Don't use the method routeAddress it is defined for only the method route.

The method getController uses controller name as parameter then returns the object of the controller.

The method getModel uses the controller name as input and returns the object of the corresponding model.

The method get uses the controller name as input the returns the name of the corresponding entity name.

The method getModelName uses the controller name as input then as the name indicate it will return the name of the corresponding model.

The method display is probably it will be the most used method in your project from the method in the class Route. It will take the page name, sub page name and request array for the page as an input then it will redirect to the page.

The method goRoute uses the controller route address and request array as input and then it will redirect to the controlling. Uses the method goRouteAddress.

The method display takes page name, sub page name and request array as parameter and then returns json encoded string. The return string should be returned to the engine inside a controller because that is where the encoded string will be decoded and then the required view will be shown. You should only use this method inside the controller to view page.

The method viewAddress take page name, sub page name and request array and then returns address string of the template page required.

The method view takes page name, sub page name and request array and then returns nothing but it will redirect to the required page.

The method set uses controller name and entity name as input and then it will add that to the route map array.

For example, for display method usage in template code segment.

```
115
116                 $return["for"] = $request->for;
117                 $return["error_message"] = $result["returned"];
118                 Route::display( page_name: "previews",  sub_page: "show", $return);
119                 die("go");
```

Example for route method usage on some line template code segment as an action on the html form.

```
59          <form action="<?php Route::route( name: "UsersController.login"); ?>" method="post">
60              <div class="form-group">
61                  <input type="text" class="form-control bg-light" id="InputUsername" name="username" placehol
62              </div>
63              <br>
64              <div class="form-group">
65                  <input type="password" class="form-control bg-light" id="InputPassword" name="password" plac
66              </div>
67
68              <button type="submit" class="btn btn-dark">Login</button>
69
70              <a class="card-link" href="#">Forget Password</a>
71          </form>
```

4.3. Set Route name in Controller

To print route address on template we need to set controller route address and if it is necessary, we can give request array as parameter to route method. Now the question is how do we write the controller route address? The route address contains the controller name and the method name connected by dot. The controller name can be found because it is mapped in route_map.php file but how do we map the name of the method and the real method?

All user controller class should extend the super abstract controller class. And there is a method which should be implemented by the child class. The method is named route and it takes two parameter 'name' as a method name and 'parameter' as request array which is send to that specific method. You can think route method as main method in java or C++. Route method called from the controlling engine file and also the name and the request array will be set by that file.

A developer can create any number of method and give any kind of route name for that specific method inside the controller. For example, I created BuildersController inside that controller I will create new method called 'paly' and the next images shows you how to map that method.

If the method will not receive any request then you can create the method like this inside the controller class

```
44      public function play(){
45
46          //codes to be executed
47
48      }
```

Then you can map the method inside route method by giving any name in my case I gave it play

```
18          }else if($name == "play"){
19
20              $this->play();
21
22          }
```

If the method will receive request then we will create the method like this inside the controller class

```
44      public function play($request){
45
46          //codes to be executed
47
48      }
```

Then we will map it like this inside the route method

```
18          }else if($name == "play"){
19
20              $this->play($parameter);
21
22          }
```

If you want to route to this method then you should write like this

If you want to print the route address then

`Route::route("controller_name.method_name", $request_array);`

In my case

`Route::route("BuildersController.play", $request_array);`

`if the method don't accept requests then you can skip the request_array.`

Chapter 5

Templates

5. What is Template

Template is the a php file which can be written using php, html, java script and Cascade Style Sheet. A template can be web page or header or footer for collection of other templates. A template can only be created inside Templates folder. You can create a folder inside the Templates folder and create your template inside the newly created folder.

5.1. Creating Templates

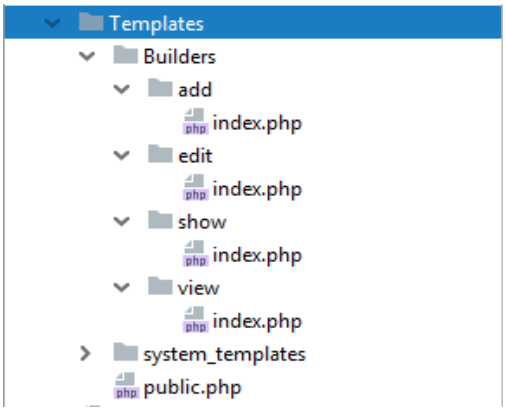
We said that templates can only be created inside the Templates folder but how? There is no way that templates can automatically be created, but we tried to make it as simple as we can. You can create template by following the next steps.

Step 1: you should create page folder for the template to be easily route. There can be many page folders inside the Templates folder.

Step 2:you should create subpage folder inside the page folder. You can create as many subpage folders as you like with different name inside the page folder.

Step 3:you should create your template file inside the subpage folder and it should be named index.php. the index file should be the only file in that subpage folder.

To explain this, I created Page folder named “Builders” and subpage folders add, edit, show and view as shown on the image below. Then I created four templates named index.



If you want to create header or footer which will not be navigate directly the you can create inside the Templates folder.

5.2. Built in methods and variables for templates and their use

There are 3 useful variables built for line users those are _app_url, _main_address and request. All of them are access by using this key word of php. Like `$this->_app_url`

_app_url is a string URL address for the main project folder so that you can access any file from that point on. _main_address is the same as the _app_url but it is disk address.

But you might not use them because for inclusion of template you have loadTemplate method. This method takes the name or the address of the template (address after template folder address) as parameter then it will include the view (template). There is also one static method which is in loader class named resource. It helps to load resource files as base 64 encoding.

5.3. Routing to the templates

When you navigate to the public folder the you will see the default line welcome page. You can navigate to other templates you made by setting page name and subpage name after pages to the address you entered to get to the welcome page. In my case I use the address http://localhost to view the welcome page and I will add the page name and subpage name of my new template after pages to view it. which means I use http://localhost/pages/builders/add to view the template placed at the address ../Templates/builders/add/index.php.

Chapter 6

Database Connectivity

6. Database in Line

The database in line framework is Structured Query Language (SQL). You can use different kinds of servers and only one database from each. You can use MySQL and MSSQL database servers.

6.1. Single/Multiple Database Configuration

In line framework there is a way that you can configure your database connection manually. To configure your database connection first you should decide which database server you want to use. Because line only works for SQL servers. From SQL server MySQL and MSSQL server. To configure your MySQL server there are some steps to follow.

Step 1: go to your line project folder and go to /SystemConstructor/Line/Database/Database.json.

Step 2: If you want to use MySQL server then under 'mysql' you can set all variables. Or if you want to use the MSSQL server then you can set the variables under 'mssql' server. And if you want to use both servers then fill all the variables.

```
{
  "MySql": {
    "DB_NAME": "order_us",
    "DB_USERNAME": "root",
    "DB_PASSWORD": "",
    "HOST_ADDRESS": "localhost"
  },
  "MsSql": {
    "DB_NAME": "order_us",
    "DB_USERNAME": "abnet",
    "DB_PASSWORD": "abnetk",
    "HOST_ADDRESS": "localhost"
  }
}
```

Step 3:if you fill MySQL then delete the MSSQL. if you fill MSSQL then delete the MySQL if you fill both then don't delete any. After that save the file.

6.2. Creating table on Database

After creating all models and database builders you have to create their table in order to fetch and store data to the database and now, we will see how to create the tables. You can create the tables manually on the database server but line provide a way to do that automatically but to do that as always there are some steps you need to follow.

Step 1: open the line control panel and select CLI tab from the top navigation bar.

Step 2: write the following command on the CLI if you want to create all the tables with one command.

```
absoft buildDatabase all
```

then all the tables will be created on database. if you want to create only one table the you can replace 'all' with the database builder name on the above command then it will create the database table on the database.

```
absoft buildDatabase database_builder_name
```

6.3. Fetching data From Database

After you create data from database you might want to retrieve data from database. to do that the first thing is you need to have database builder and model for that specific table. If you already have that you can proceed to the next step if you don't view chapter 3.

Step 1: if you are in a controller with different name from the model and controller then you have to include the model file to that controller otherwise you don't need to do that.

```
1
2   <?php
3
4   Loader::loadModel( model_name: "TeachesModel");
5   Loader::loadModel( model_name: "UsersModel");
6   Loader::loadModel( model_name: "CoursesModel");
7   Loader::loadModel( model_name: "SectionsModel");
8   Loader::loadModel( model_name: "AssessmentsModel");
9   Loader::loadModel( model_name: "SemestersModel");
10
11  class AssessmentResultsController extends Controller{
12
```

Step 2:create the object of that model then use the search method in the model object.

```
33      public function show(){
34
35          $object = new BaseModel();
36
37          $result = $object->search(
38              [
39                  "id" => [
40                      "value" => "1",
41                      "equ" => "=",
42                      "det" => "and"
43                  ]
44              ],
45              [
46                  "id",
47                  "name",
48                  "age"
49              ]
50          );
51
52          $data = $result["returned"];
53
54      }
```

The first parameter of the search method is the condition to search. In the above image the search has only one condition it is if the id is equals to 1. The 'det' part is only used if there is another condition to follow.

The second parameter of the search method is the filter of the column to fetch. In the above image the search result table will have three column those are id, name and age.

There is the third parameter called join. it will join the table/database builders name that you set on the join parameter.

There is also the fourth variable called order by. This parameter accepts associative array like on the image below. The 'att' stands for attribute you will set the attribute you wanted to order by and 'det' stands for determiner you will set 0 or 1. 0 is descending order and 1 is ascending order.

```
188             [
189                 "att" => "id",
190                 "det" => "0"
191             ]
```

The search result comes in associative array form.

6.4. Saving Data into Database

To save data in to database you can use 'addRecord' method which built in function for all models. The method takes only one variable which must be associative array of attribute with its value to be set.

```php
public function save($request){

    $model = new BaseModel();

    $model->addRecord(
        [
            "id" => "1",
            "name" => "Abnet Kebede",
            "age" => "13"
        ]
    );

}
```

Here is example for saving data. The first image shows how the data will be send to the controller from form and the second show how to receive and save to database.

```php
<form class="container" action="<?php Route::route( name: "UsersController.save"); ?>" method="post">

    <div>
        <div class="form-group">
            <input type="text" class="form-control" name="username" aria-describedby="usernameHelp" placeholder="Username" require
        </div>

        <div class="form-group">
            <input type="email" class="form-control" name="email" aria-describedby="emailHelp" placeholder="Email Address" require
        </div>

        <div class="row">
            <div class="col-md-6 col-sm-12">
                <input type="text" name="f_name" class="form-control" placeholder="First name">
                <br>
            </div>
            <br>
            <div class="col-md-6 col-sm-12">
                <input type="text" name="l_name" class="form-control" placeholder="Last name">
            </div>
        </div>
        <br>

        <div class="form-group">
            <input type="number" class="form-control" name="phone" aria-describedby="phoneHelp" placeholder="Phone Number" require
        </div>

        <div class="input-group mb-3">
            <div class="input-group-prepend">
                <label class="input-group-text" for="user_role">Employee Type</label>
            </div>
```

```
                <select class="custom-select" id="user_role" name="role" required>
                    <option value="designer">Designer</option>
                    <option value="cashier">cashier</option>
                </select>
            </div>

            <div class="form-group">
                <label for="n_password_input">Create Password</label>
                <input type="password" class="form-control" id="n_password_input" name="password" placeholder="Create Password" requir
            </div>
            <div class="form-group">
                <label for="c_password_input">Confirm Password</label>
                <input type="password" class="form-control" id="c_password_input" name="confirmation" placeholder="Confirm Password" r
            </div>
            <button class="btn btn-primary" type="submit">Request Registration</button>
        </div>

    </form>
```

At the controller.

```
public function save($request){

    if(isset($request->username) && isset($request->password) && isset($request->confirmation) && isset($request->email) &&
        isset($request->f_name) && isset($request->l_name) && isset($request->role) && isset($request->phone)){

        $username = $request->username;
        $password = $request->password;
        $c_password = $request->confirmation;
        $role = $request->role;
        $f_name = $request->f_name;
        $l_name = $request->l_name;
        $email = $request->email;
        $phone = $request->phone;

        if($password === $c_password){

            $data = [
                "username" => $username,
                "password" => password_hash($password, algo: PASSWORD_DEFAULT),
                "email" => $email,
                "phone_number" => $phone,
                "f_name" => $f_name,
                "l_name" => $l_name,
                "role" => $role,
                "status" => "request"
            ];

            $model = new UsersModel();

            $result = $model->addRecord($data);

            if($result['message'] == "1"){
```

```
                Route::display( page_name: "users", sub_page: "add", [
                    "success_message" => "your Requests Sent successfully please wait for our approval. Thank you"
                ]);

            }else{

                Route::display( page_name: "users", sub_page: "add", [
                    "error_message" => $result["returned"].". your Request were not sent."
                ]);

            }

        }else{

            Route::display( page_name: "users", sub_page: "add", [
                "error_message" => "The password you enter does not match. your Request were not sent."
            ]);

        }

    }

}
```

6.5. Updating data on Database

To update data on database you can use the built-in method called ‘update’. The method update takes 2 parameters which are array.

The first parameter called set. It must be associative array of the attribute with its new value.

The second parameter is called condition. It must also be associative array.

```
70      public function update($request){
71
72          $model = new BaseModel();
73
74          $model->update(
75              [
76                  "name" => "babbi",
77                  "age" => 13,
78              ],
79              [
80                  "id" => [
81                      "value" => "1",
82                      "equ" => "=",
83                      "det" => "and"
84                  ],
85                  "name" => [
86                      "value" => "abnet",
87                      "equ" => "=",
88                      "det" => "and"
89                  ]
90              ]
91          );
92
93      }
```

6.6. Deleting data from Database

You can delete data from database by using deleteRecord built in method. The method deleteRecord takes one parameter called condition and it must be array. The parameter is optional if you don't set condition all the data on the table will be deleted.

```
95      public function delete($request){
96
97          $model = new BaseModel();
98
99          $model->deleteRecord(
100             [
101                 "id" => [
102                     "value" => "2",
103                     "equ" => "=",
104                     "det" => "and"
105                 ]
106             ]
107         );
108
109     }
```

Chapter 7

Other things in Line

7. Chapter content

In this chapter we will try to see how some things are done in php like how to use paging, how to use IP checker and so on.

7.1. Ip checker

Developer might want to keep their system from untrusted users. And there is a way to secure your system by using line framework. What it does is that it detects their IP address and then it will check the IP if the IP is trusted or not. If the IP is not on the list of untrusted IP addresses then it will pass otherwise it will be blocked.

To do that the developer needs to put the untrusted IP addresses. There is a file in the project folder which is called system description. It is a json file and developers can put their untrusted client IP addresses as an array under the variable called untrusted_ip.

In line the control panel can only be accessed from the server machine you cannot access the control panel from any other device connected with the server.

You can protect your system by loading the IP checker file to your page by writing the following code on the template you want to protect.

```
Loader::checkIp();
//this method will return the client IP address
IpCheck::clientIp();
```

7.2. Paging

In your system you might fetch files from your database that has more than 100 item to be displayed on the template to the user. the data that the user want to see might be on the first page so the user will see that and it will leave the page and the rest loaded data will be a west. In this case the user pays for data which he did not see. To avoid that line framework provides away to

partition the data fetched and the user will see the data as a page. This functionality is done by the pager class.

to make partition you need to create pager.

```
$pager = new Pager();
//header should be set to tell where it should redirect after getting the
//data
$headers = new stdClass();
$headers->page_name = “Page Name”;
$headers->sub_page = “Sub Page Name”;
//here you should set new pager name. you should not put the same kind of
//pager name that you used before for another pager in the system. it is
//better to use the name of the table as the name of the pager.
$pager_name = “my_pager”;
//you should also set return route name(address)
$route_name = “NameOfController.method”
//this will set how much data a single page should view. In my case 7.
$page_size = 7;
//and also you should set requests to be send back to the controller
//but if you have nothing to send then you can leave it.
$request = []
$pager->create($headers, $route_name, $name, $page_size, $request);
//to get a page you have to provide the fetched data to the method.
Pager::getPage($fetched_data, $page_number);
```

After creating and getting the data to view then you might want to make pager on the template so that the user can select any page he likes to view. You don’t need to make that because we do make that for you. You only need to call a method on from the template you want to make the pager. To the method you need to set the pager name which you named when you create the pager.

```
//this method will print a series of html anchor tags which has no class name
Pager::pagerView($name);
```

The html anchor tags that the above method print, does not has good looking so you can change the style by using other methods provided by line.

```
//you can change the class name of non-active pager links
//the method takes to parameters to be strings.
//the first parameter is name of the pager. And the second is the class name
Pager::linkPageClass($name, $class_name);
//you can change the class name active pager links by using the next method
//the method takes to parameters to be strings.
//the first parameter is name of the pager. And the second is the class name
Pager::currentPageClass($name, $class_name);
//the next method will return the current page number
Pager::getPageNumber($name);
//the next method will return the last page number
Pager::getLastPageNumber($name);
//you can also get only the cut data which will be send to the view, as an
//array
//this method take three parameters and it will return associative array
//the first is the data which will be cut and
//the second is the pager name
//the third is the page number which you want to get the data of.
Pager::pageData($data, $pager_name, $page_number);
```

7.3. Alert

In the web site you are going to build there might need to alert the user. this can be done by displaying the alert on the top of the page. Not only showing alert but also dividing them by their message content. Line provide this in simple way. If you want to send alert to the page or view or template you only need to write on line of code. It looks like the following.

```
//sending error alert
Alert::sendErrorAlert("the is an error");
//sending information alert
```

```
Alert::sendInfoAlert("there is information you need to know");
//sending success message
Alert::sendSuccessAlert("opration successful");
```

Before calling the alert, you have to set styling class name other wise it will mess up the page. The view wouldn't be attractive. Line provide way to set the class name for each alert types. Looks like the next.

```
//setting class name for success message
Alert::setSuccessClassName("class-name");
//setting class name for error message
Alert::setErrorClassName("class-name");
//setting class name for information message
Alert::setInfoClassName("class-name");
```

Next

We set class name and the Alert (message) itself and now how do we display the message. Looks like the next.

```
//write this code where you want to display your message but after setting
//class name and the message itself
Alert::display();
//this will display any alert send to the view but after it is displayed once
//it will be deleted. It means that when you reload you will not see it.
```

Chapter 8

Updating Line