Bast

Release 1.0

Apr 23, 2019

Contents

CHAPTER 1

About Bast

Bast is a Simple and Elegant Framework. The main aim of Bast is to create an enjoyable and creative Experience for Developers. Bast attempts to take the pain out of development by making common tasks used in the majority of web projects easy. Bast is aimed to be platform Independent and it's core Language is Python. Uses Python 3.*

CHAPTER 2

Usage

To install Bast, you can download it easily from Pypi using

```
$ pip install Bast
```

Bast comes bundled with a very powerful CLI tool called `panther`. To show the available commands, use

```
$ panther --help
```

To create a setup a new project, use

```
$ panther new project_name
$ cd project
$ panther run
```

To visit the website and see if it's setup successfully, visit `127.0.0.1:2000` in your browser

CHAPTER 3

Update

Bast Routing is now relatively simple and much more easy to use. It embodies the way and manner Laravel defines it's URL's but instead of the `@` symbol, Bast makes use of the `.` You do not need to import your controller again

```
from bast import Route

route = Route()
route.get(url='/', controller='ExampleController.index')
```

Bast Controllers are Python Classes which inherit from the Bast Controller Class. Using `panther create:controller ControllerName` creates a controller file in the controller package. To render template in controller, use `self.view('template.html', args=None)` where the args is a Dictionary object and optional

```
from bast import Controller


class TestController(Controller):
    def index(self):
        self.view('index.html')
```

To run your app use

```
$ panther run
```

CHAPTER 4

Documentation

4.1 Introduction

Bast is a Simple and Elegant Framework. The main aim of Bast is to create an enjoyable and creative Experience for Developers. Bast attempts to take the pain out of development by making common tasks used in the majority of web projects easy. Bast is aimed to be platform Independent and it's core Language is Python. Uses Python 3.*

Bast is heavily based on the Tornado Web Framework as it makes use of the Tornado HTTP Server so most of the methods from Tornado are overriden in Bast. They all perform the same function more or less. Taking the complete advantage of Tornado's asynchronous and non blocking network IO, Bast has eliminated most of the complexities inherent in Tornado making it so much more easier and simpler to make use of

Orator ORM is used to handle the Object Relational Mapping aspect of Bast. Visit Orator ORM's website for it's full documentation

Jinja Templating is used to handle the View Aspect of the Framework

4.2 Installation

To install the framework, it is highly recommended you make use of the available Python Package on Pypi

```
$ pip3 install bast
```

This installs the latest version of Bast from Pypi. If you would prefer to make use of setup tools, you can clone the source from the Github Repository then using setuptools, you can install it

```
$ python setup.py build
$ python setup.py install
```

NOTE: It is advisable to create a virtual environment for the installation of Bast Framework in order to avoid conflict among installed packages

Once Bast has been installed, you now have access to the `panther` CLI tool. To make a new project,

```
$ panther new my_project
```

This scaffolds the boilerplate code from Github which has the base folders and files. To run the project

Visit `localhost:2000` to view whether it has been installed successfully

4.3 Controllers

Controllers in Bast handle the logic of our app as stated in the MVC paradigm `M - Model | V - View | C - Controller`. To create controllers in Bast

```
$ panther create:controller MyController
```

This creates a `MyController.py` file in the controller directory. The skeleton for the MyController file is

```
from bast import Controller


class MyController(Controller):
    pass
```

You can then create any method and use the methods inherent in the controller class

4.3.1 Controller Methods and Functions

class `bast.controller.`**`Controller`**(*application*, *request*, ***kwargs*)

`Controller.`**`write_error`**(*status_code*, ***kwargs*)
Handle Exceptions from the server. Formats the HTML into readable form

`Controller.`**`view`**(*template_name*, *kwargs=None*)
Used to render template to view

```
from bast import Controller

class MyController(Controller):
    def index(self):
        self.view('index.html')
```

`Controller.`**`only`**(*arguments*)
returns the key, value pair of the arguments passed as a dict object

```
from bast import Controller

class MyController(Controller):
    def index(self):
        data = self.only(['username'])
```

Returns only the argument username and assigns it to the data variable.

`Controller.`**`except_`**(*arguments*)
returns the arguments passed to the route except that set by user

```python
from bast import Controller

class MyController(Controller):
    def index(self):
        data = self.except_(['arg_name'])
```

Returns a dictionary of all arguments except for that provided by as `arg_name`

`Controller.`**`json`**`(`*data*`)`
Encodes the dictionary being passed to JSON and sets the Header to application/json

`Controller.`**`get`**`(`***kwargs*`)`

`Controller.`**`post`**`(`***kwargs*`)`

`Controller.`**`put`**`(`***kwargs*`)`

`Controller.`**`delete`**`(`***kwargs*`)`

`Controller.`**`get_argument`**`(`*name*, *default=None*, *strip=True*`)`
Returns the value of the argument with the given name.

If default is not provided, returns `None`

If the argument appears in the url more than once, we return the last value.

The returned value is always unicode

`Controller.`**`get_arguments`**`(`*name*, *strip=True*`)`
Returns a list of the arguments with the given name.

If the argument is not present, returns an empty list.

The returned values are always unicode.

`Controller.`**`headers`**`()`
Returns all headers associated with the request

`Controller.`**`header`**`(`*param*`)`
Returns the header specified by the key provided

4.4 Routing

Routing in Bast is very simple and efficient. It follows the routing convention of Laravel and Adonis. A simple example is shown below

```python
from bast import Route

route = Route()
route.get('/', 'ExampleController.index')
```

Bast Routes support `GET`, `POST`, `PUT` and `DELETE` requests.

4.4.1 Routing Methods

`class` `bast.route.`**`Route`**
Route Class. Appends the URL, Controller Instance and Method to a list instance to be passed on to the server instance

`Route.`**`get`**`(`*url*, *controller*`)`
Gets the Controller and adds the route, controller and method to the url list for GET request

`Route.`**`post`**`(`*url*, *controller*`)`
Gets the Controller and adds the route, controller and method to the url list for the POST request

`Route.`**`put`**`(`*url*, *controller*`)`
Gets the Controller and adds the route, controller and method to the url list for PUT request

`Route.`**`delete`**`(`*url*, *controller*`)`
Gets the Controller and adds the route, controller and method to the url list for the DELETE request

`Route.`**`middleware`**`(`*args*`)`
Appends a Middleware to the route which is to be executed before the route runs

`Route.`**`__return_controller__`**`(`*controller*`)`

`Route.`**`all`**`()`
Returns the list of URL. Used by Server to get the list of URLS. This is passed to the Bast HTTP Server

4.5 Templating

Bast makes use of Jinja as it's template engine. To render view from Controller use `self.view('template.html')` to render the view.

The `view` function takes in the template name as an argument and an optional parameter `args` which is of type `dict`. This is used to pass data from the controller to the view

To access the data passed as args, use the key in the template to reference it. In controller, we pass `{ 'foo': 'bar'}` to the view together with the template. To access it in the template, we use the key to access it `{{ foo }}`

All HTML files goes in `public/templates` directory as the server loads the templates directly from there.

To create view, you can use the `panther` CLI tool to create the view

```
$ panther create:view mytemplate
```

This creates a `mytemplate.html` in the `public/templates` folder.

Another option is to directly create the `mytemplate.html` in the `public/templates` folder

4.5.1 Static Files

Static Files are to be stored in the `public/static` folders. Cascading Style Sheet (CSS) files are to be stored in `public/static/css` folder, Javascript/JS files are to be in the `public/static/js` folder and Images are to be in the `public/static/images` folder.

Bast has built in functions to access files in each of the folders and renders it to the view with it's appropriate HTML code.

To render scripts , `{{ script('myscript.js') }}` which in view would translate to `<script type="text/javascript" src="/script/myscript.js"></script>`

To render CSS `{{ css('mystyle.css') }}` which in view would translate to `<link rel="stylesheet" href="/css/mystyle.css">`

To render images `{{ image('myimage.png', 'alt_name') }}` which in view would translate to `<img src="/images/myimage.png" alt="alt_name">`. **P.S.: The "alt_name" is optional**

For further options on how to use Jinja Templates, visit the Jinja Documentation

4.6 Models

The power of Models are harnessed by Bast using Orator. To create Models, the `panther` CLI tool is used

```
$ panther create:model User
```

The above code creates a model and migration file for the model.

```
# Model File
from bast import Models

class User(Models):
    __table__ = 'user'
```

You can change the table the Model maps to by changing the `__table__` variable to the table name

To create only a model file without a migration file, use

```
$ panther create:model User --migration=False
```

The above command creates a model file without a migration file

Bast makes use of Orator. Check out the docs at Orator ORM for more ways on how to use the Orator ORM

4.7 Migrations

Bast Migrations are relatively easy to create and run. To use the `migration` command, the `panther` commandline tool is used to run the migrations. When you create models, migration files are generated automatically with it unless you don't want it.

To create Migration file, use

```
$ panther create:migration Test
```

This creates a `2018_08_01_154353_test.py` inside `database/migrations` folder.

To run migrations,

```
$ panther migration:run
```

This runs every migration file present in the `database/migrations` folder

To rollback the last migration

```
$ panther migration:rollback
```

To rollback/reset all migration

```
$ panther migration:reset
```

CHAPTER 5

Maintainer

```
$  Majiyagbe Oluwole
```

CHAPTER 6

Contributors

```
$ Majiyagbe Oluwole
$ Azeez Abiodun Solomon
```

CHAPTER 7

License

This Framework is Licensed under MIT License

CHAPTER 8

Credits

Bast runs on the Tornado HTTP Server.

For templating, Bast makes use of the Jinja Templating Engine.

Eloquent Object Relation Mapping is achieved using Orator ORM

Index