fantastico Documentation

Release 0.0.1-b44

Radu Viorel Cosnita

April 21, 2013

CHAPTER

ONE

INTRODUCTION

1.1 Why another python framework?

The main reason for developing a new framework is simple: I want to use it for teaching purposes. I have seen many projects which fail either because of poor coding or because they become legacy very fast. I will not get into details why and what could have been done. It defeats the purpose.

Each piece of code that is being added to fantastico will follow these simple rules:

1. *The code is written because is needed and there is no clean way to achieve the requirement with existing fantastico features*.
2. The code is developed using TDD (Test Driven Development).
3. The code quality is 9+ (reported by pylint).
4. The code coverage is 90%+ (reported by nose coverage).
5. The code is fully documented and included into documentation.

1.1.1 What do you want to teach who?

I am a big fan of Agile practices and currently I own a domain called scrum-expert.ro. This is meant to become a collection of hands on resource of how to develop good software with high quality and in a reasonable amount of time. Resources will cover topics like

1. Incremental development always ready for rollout.
2. TDD (Test Driven Development)
3. XP (eXtreme programming)
4. Scrum
5. Projects setup for Continuous Delivery

and many other topics that are required for delivering high quality software but apparently so many companies are ignoring nowadays.

1.2 Fantastico's initial ideas

- Very fast and pluggable routing engine.
- Easily creation of REST apis.
- Easily publishing of content (dynamic content).

- Easily composition of available content.
- Easily deployment on non expensive infrastructures (AWS, RackSpace).

Once the features above are developed there should be extremely easy to create the following sample applications:

1. Blog development
2. Web Forms development.
3. Personal web sites.

CHAPTER

TWO

GETTING STARTED

2.1 Installation manual

In this section you can find out how to configure fantastico framework for different purposes.

2.1.1 Developing a new fantastico project

Currently fantastico is in early stages so we did not really use it to create new projects. The desired way we want to provide this is presented below:

pip-3.2 install fantastico

Done, now you are ready to follow our tutorials about creating new projects.

2.1.2 Contributing to fantastico framework

Fantastico is an open source MIT licensed project to which any contribution is welcomed. If you like this framework idea and you want to contribute do the following (I assume you are on an ubuntu machine):

```
#. Create a github account.
#. Ask for permissions to contribute to this project (send an email to radu.cosnita@gmail.com) - I w
#. Create a folder where you want to hold fantastico framework files. (e.g worspace_fantastico)
#. cd ~/workspace_fantastico
#. git clone git@github.com:rcosnita/fantastico
#. sudo apt-get install python3-setuptools
#. sh virtual_env/setup_dev_env.sh
#. cd ~/workspace_fantastico
#. git clone git@github.com:rcosnita/fantastico fantastico-doc
#. git checkout gh-pages
```

Now you have a fully functional fantastico workspace. I personally use PyDev and spring toolsuite but you are free to use whatever editor you want. The only rule we follow is *always keep the code stable*. To check the stability of your contribution before commiting the code follow the steps below:

```
#. cd ~/workspace_fantastico/fantastico/fantastico
#. sh run_tests.sh (we expect no failure in here)
#. sh run_pylint.sh (we expect 9+ rated code otherwise the build will fail).
#. cd ~/workspace_fantastico/fantastico
#. export BUILD_NUMBER=1
#. ./build_docs.sh (this will autogenerate documentation).
#. Look into ~/workspace_fantastico/fantastico-doc
#. Here you can see the autogenerated documentation (do not commit this as Jenkins will do this for y
#. Be brave and push your newly awesome contribution.
```

2.2 Fantastico settings

Fantastico is configured using a plain settings file. This file is located in the root of fantastico framework or in the root folder of your project. Before we dig further into configuration options lets see a very simple settings file:

```
class BasicSettings(object):
    @property
    def installed_middleware(self):
        return ['fantastico.middleware.RequestResponseMiddleware',
                'fantastico.middleware.RoutingEngineMiddleware']
```

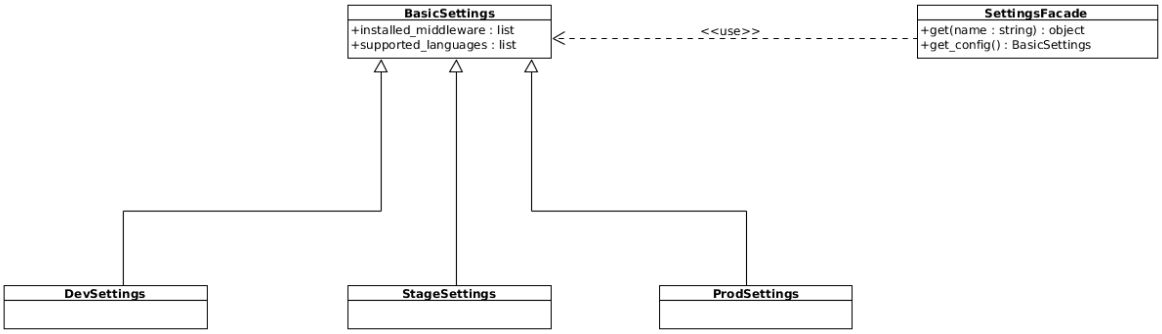
The above code sample represent the minimum required configuration for fantastico framework to run. The order in which middlewares are listed is the order in which they are executed when an http request is made.

2.2.1 Settings API

Below you can find technical information about settings.

class fantastico.settings.**BasicSettings**

This is the core class that describes all available settings of fantastico framework. For convenience all options have default values that ensure minimum functionality of the framework. Below you can find an example of three possible configuration: Dev / Stage / Production.



As you can see, if you want to overwrite basic configuration you simply have to extend the class and set new values for the attributes you want to overwrite.

installed_middleware

Property that holds all installed middlewares.

supported_languages

Property that holds all supported languages this fantastico instance.

2.2.2 Create Dev configuration

Let's imagine you want to create a custom dev configuration for your project. Below you can find the code for this:

```
class DevSettings(BasicSettings):
    @property
    def supported_languages(self):
        return ["en_us"]
```

The above configuration actually overwrites supported languages. This mean that only en_us is relevant for **Dev** environment. You can do the same for **Stage**, **Prod** or any other custom configuration.

2.2.3 Using a specifc configuration

class fantastico.settings.**SettingsFacade**(*environ=None*)

For using a specific fantastico configuration you need to do two simple steps:

- Set **FANTASTICO_ACTIVE_CONFIG** to the fully python qualified class name you want to use. E.g: fantastico.settings.BasicSettings
- In your code, you can use the following snippet to access a specific setting:

```
from fantastico.settings import SettingsFacade

print(SettingsFacade().get("installed_middleware"))
```

If no active configuration is set in the fantastico.settings.BasicSettings will be used.

get(*name*)

Method used to retrieve a setting value.

Parameters

- **name** – Setting name.
- **type** – string

Returns The setting value.

Return type object

get_config()

Method used to return the active configuration which is used by this facade.

Return type fantastico.settings.BasicSettings

Returns Active configuration currently used.

CHAPTER

THREE

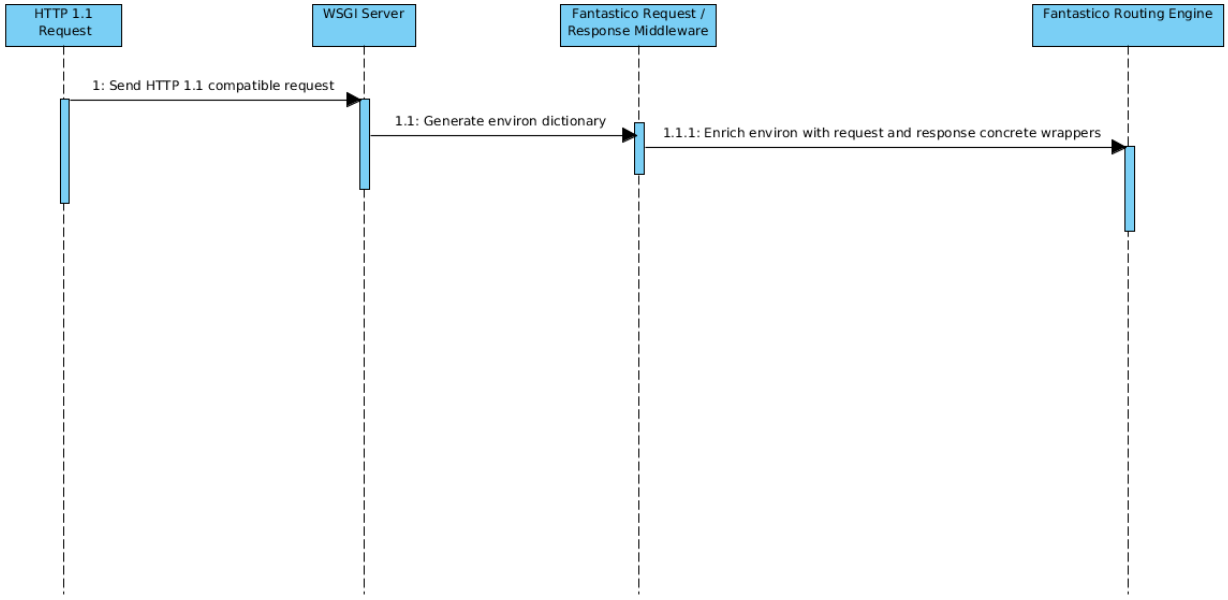
FANTASTICO FEATURES

3.1 Request lifecycle

In this document you can find how a request is processed by fantastico framework. By default WSGI applications use a dictionary that contains various useful keys:

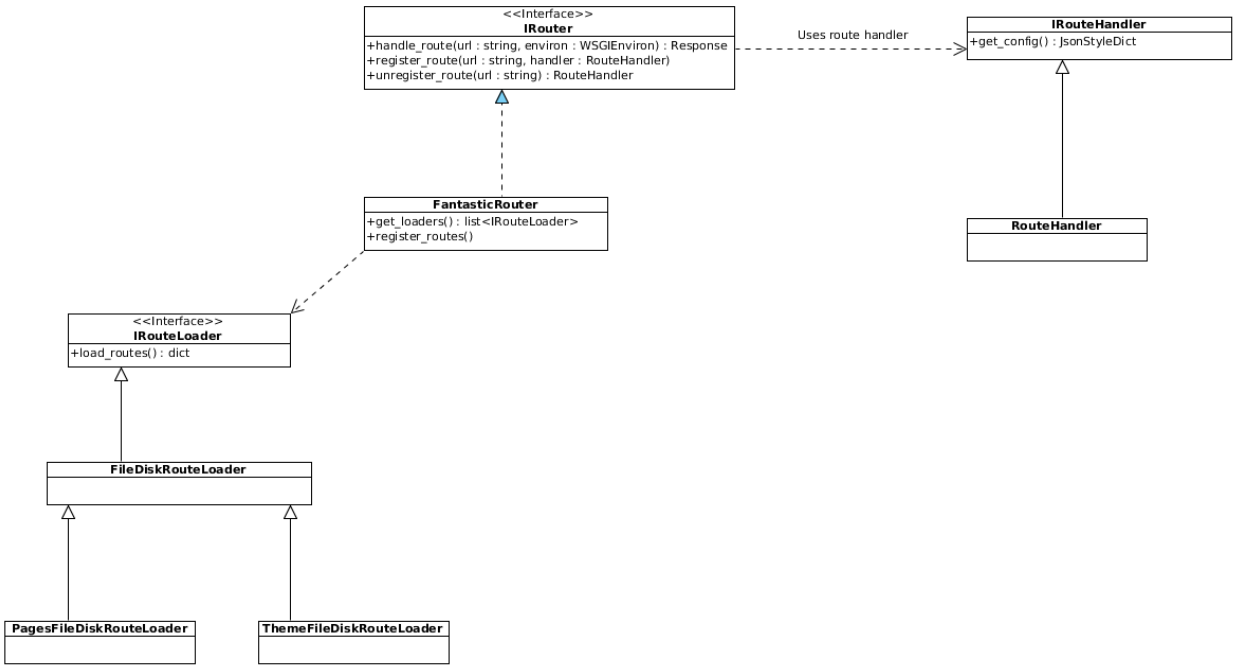
- HTTP Headers
- HTTP Cookies
- Helper keys (e.g file wrapper).

In fantastico we want to hide the complexity of this dictionary and allow developers to use some standardized objects. Fantastico framework follows a Request / Response paradigm. This mean that for every single http request only one single http response will be generated. Below, you can find a simple example of how requests are processed by fantastico framework:



In order to not reinvent the wheels fantastico relies on WebOb python framework in order to correctly generate request and response objects. For more information read WebOB Doc.

3.2 Routing engine



CHAPTER

FOUR

BUILD STATUS

If you want to see the current build status of the project visit Build status.

CHAPTER

FIVE

LICENSE

B

G

I

S