Radiant Framework

Yeison Cardona

Apr 07, 2024

CONTENTS

1 Overview **3**

2 Key Features **5**
- 2.1 Python-Centric Development . . . 5
- 2.2 Server and Browser Compatibility . . . 5
- 2.3 Resource Management . . . 5
- 2.4 Runtime Configuration . . . 5

3 Benefits **7**

4 Installation **9**

5 Bare minimum **11**
- 5.1 Prerequisites . . . 11
- 5.2 Creating a Simple Web Page . . . 11

6 Documentation Overview **13**
- 6.1 Radiant Framework . . . 13
- 6.2 WebSockets . . . 15
- 6.3 Python API . . . 16
- 6.4 Brython Enhancement . . . 17
- 6.5 Templates . . . 19
- 6.6 Editing HTML Meta Tags in Radiant . . . 20

A Brython Framework for Web Apps development.

CHAPTER

ONE

OVERVIEW

Radiant Framework is a novel web framework designed to leverage the capabilities of Brython, a browser-based Python implementation. This innovative approach allows developers to write web applications entirely in Python, bypassing the conventional requirements of HTML, CSS, or JavaScript for frontend development.

CHAPTER

TWO

KEY FEATURES

2.1 Python-Centric Development

- **Unified Language Usage**: Write your entire web application using Python, ensuring a consistent and streamlined coding experience.
- **Brython Integration**: Utilizes Brython for executing Python code in the browser, enabling a seamless transition of server-side code to client-side execution.

2.2 Server and Browser Compatibility

- **Dual Environment Execution**: Radiant enables the same application code to run both on the server and in the browser, maximizing code reusability and efficiency.
- **Tornado Web Server**: On the server-side, Radiant harnesses the Tornado web server to deploy applications, known for its scalability and non-blocking network I/O capabilities.

2.3 Resource Management

- **Static File Handling**: Simplifies the management of static files (images, stylesheets, etc.), by setting up a local path for their serving, facilitating their inclusion in the application.

2.4 Runtime Configuration

- **Dynamic HTML Templates**: Radiant offers a custom HTML template system, configurable at runtime, to dynamically import server-side scripts into the browser environment.

CHAPTER

THREE

BENEFITS

- **Streamlined Development Process**: By unifying the development language and environment, Radiant significantly reduces the complexity and learning curve associated with traditional web development.
- **Code Efficiency**: Eliminates the need for writing separate frontend and backend code, leading to more maintainable and concise codebases.
- **Focus on Quality**: Developers can concentrate on crafting high-quality Python code, without the distractions of dealing with various web technologies.

CHAPTER

FOUR

INSTALLATION

To install Radiant, you can use `pip`, the Python package manager. Simply run the following command in your terminal:

```
pip install radiant-framework
```

CHAPTER

FIVE

BARE MINIMUM

To help you get started with Radiant, let's walk through a bare minimum example. This example will demonstrate how to create a simple web page that displays some text. We'll utilize the Radiant framework to craft the page and run it on a local server. This is an excellent way to familiarize yourself with how Radiant functions and to begin exploring its capabilities.

5.1 Prerequisites

Before proceeding, ensure that you have Radiant installed on your system. If you haven't installed Radiant yet, please refer to the *Installation* section for guidance.

5.2 Creating a Simple Web Page

The following script illustrates a basic application using Radiant. This script will set up a simple web page displaying a heading.

```
from radiant.framework.server import RadiantAPI
from browser import document, html

class BareMinimum(RadiantAPI):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        document.select_one('body') <= html.H1('Radiant-Framework')

if __name__ == '__main__':
    BareMinimum()
```

CHAPTER

SIX

DOCUMENTATION OVERVIEW

6.1 Radiant Framework

6.1.1 Brython

Brython (Browser Python) represents a pioneering approach, aimed at supplanting JavaScript with Python as the primary scripting language for web development. Uniquely crafted as a Python 3 implementation, Brython is meticulously tailored to harmonize with the HTML5 framework. This adaptation ensures seamless integration with the Document Object Model (DOM), enabling interaction with web page elements and responsiveness to user-triggered events, thus offering a robust and intuitive environment for modern web applications.

Radiant, a versatile web framework, extends its functionality by integrating Brython. This integration allows developers to write Python code that runs directly in the browser, bypassing the traditional need for JavaScript. This feature simplifies the process of creating interactive web applications by leveraging the familiarity and power of Python.

6.1.2 Radiant

Radiant emerges as an innovative framework uniquely designed for Brython, distinguished by its remarkable capability to be executed directly from Python. This exceptional feature empowers you to write Python code, save it as a file, and then seamlessly execute and interact with it in a web browser. At its core, Radiant embraces an object-oriented paradigm, necessitating the use of classes. Crucially, it mandates the inheritance from `RadiantAPI`, a cornerstone class that provides the foundational structure and functionalities essential for leveraging the full potential of the Radiant framework in Brython-based web applications.

[ ]:
```
from radiant.framework.server import RadiantAPI
from browser import document, html

class BareMinimum(RadiantAPI):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        document.select_one('body') <= html.H1('Radiant-Framework')

if __name__ == '__main__':
    BareMinimum()
```

Radiant server running on port 5000

6.1.3 RadiantServer Options

The `RadiantServer` class plays a pivotal role in the Radiant framework, functioning as the engine for processing additional configuration options. This class allows for a higher level of customization and control over the behavior of Brython applications. By extending `RadiantServer`, developers can fine-tune aspects such as server settings, resource management, and response handling, thus enhancing the overall functionality and efficiency of their web applications.

```
[ ]: from radiant.framework.server import RadiantAPI, RadiantServer
     from browser import document, html


     class BareMinimum(RadiantAPI):

         # --------------------------------------------------------------------
         def __init__(self, *args, **kwargs):
             """"""
             super().__init__(*args, **kwargs)
             document.select_one('body') <= html.H1('Radiant Framework')
             document.select_one('body') <= html.H2('Options')


     if __name__ == '__main__':
         RadiantServer(
             'BareMinimum',
             host='localhost',
             port=5000,
             brython_version='3.11.2',
             debug_level=0,
         )
```

The configuration options for Brython in Radiant include fundamental parameters like: - `host='localhost'`: Specifies the server address, with 'localhost' indicating that the server runs on the local machine. - `port=5000`: Determines the port number for the server, where 5000 is a common default for web applications. - `brython_version='3.11.2'`: Sets the version of Brython to be used, ensuring compatibility and feature support. - `debug_level=0`: Adjusts the level of debugging information displayed, with 0 typically representing minimal output.

6.1.4 Multipage Support

One of the most useful features of the Radiant framework is the ability to create multi-page web applications. This feature allows developers to organize content and functionality across different pages, enhancing user experience and navigation. Implementing multipage support in Radiant is straightforward and offers a scalable way to structure complex web applications.

```
[ ]: from radiant.framework.server import RadiantAPI, RadiantServer
     from browser import document, html


     class BareMinimum(RadiantAPI):

         # --------------------------------------------------------------------
         def __init__(self, *args, **kwargs):
             """"""
```

(continues on next page)

(continued from previous page)

```
        super().__init__(*args, **kwargs)
        document.select_one('body') <= html.H1('Hello World')
        document.select_one('body') <= html.H2('Multipage support')
        document.select_one('body') <= html.A('second page', href='/multipage')


if __name__ == '__main__':
    RadiantServer(
        'BareMinimum',
        pages=([r'^/multipage$', '_second_page.Second'],),
    )
```

6.2 WebSockets

6.2.1 Server

This section delves into the intricacies of utilizing WebSockets within the Radiant and Tornado ecosystem. It's crucial to understand that WebSockets functionality is integrated within Tornado, and not directly within Radiant (Brython). To effectively incorporate WebSockets in a Radiant-based application, one must establish a `WebSocketHandler` within the Tornado framework. For a comprehensive understanding of the `WebSocketHandler` class and its implementation, refer to the Tornado documentation.

[ ]:
```
#ws_handler.py

from tornado.websocket import WebSocketHandler

class WSHandler(WebSocketHandler):

    def open(self):
        ...

    def on_close(self):
        ...

    def on_message(self, message):
        ...
```

To integrate a `WebSocketHandler` in your RadiantServer, it can be passed as a parameter to the `websockethandler` during server creation. This process effectively bridges Radiant and Tornado, allowing for WebSocket functionalities within your Radiant application.

[ ]:
```
RadiantServer('MainApp', websockethandler=('ws_handler.py', 'WSHandler'))
```

Creating a WebSocket server in this context involves setting up a listener on a specified URL, such as `/ws`. To define the WebSocketHandler class, pass a tuple that includes the path to the module and the class name. For example, if your module is `ws_handler.py` and the class is `WSHandler`, this information should be specified accordingly.

6.2.2 Client

For constructing a WebSocket client in a Brython environment, the `browser.websocket` module is utilized. The following example demonstrates the procedure to create a WebSocket client, leveraging Brython's capabilities.

[ ]:
```
from browser import websocket

def on_open(evt):
    print('Connection opened.')

def on_message(evt):
    print('Message received:', evt.data)

def on_close(evt):
    print('Connection closed.')

ws = websocket.WebSocket('ws://localhost:8888/ws')
ws.bind('open', on_open)
ws.bind('message', on_message)
ws.bind('close', on_close)
```

In the demonstrated example, a new WebSocket object is instantiated and connected to the server using `ws://localhost:8888/ws` as the URL. Event handlers for `open`, `message`, and `close` events are then bound to this WebSocket. Upon opening the connection, the `on_open` function is triggered, and similarly for other events. The URL in the `WebSocket` constructor can be modified to align with your server's specifics, and the event handlers can be tailored to manage the data received from the server.

6.3 Python API

A distinctive feature of Radiant Framework is its ability to execute pure Python code directly from the interface through the Tornado server. This integration facilitates seamless Python scripting within a web-based environment, enhancing the interactivity and functionality of web applications developed using Brython.

To implement this feature, the Python code must be hosted in a separate script, within a class that inherits from `PythonHandler`. This approach organizes the code efficiently and leverages the capabilities of the Tornado server for Python execution.

[ ]:
```
from radiant.framework.server import PythonHandler
import math


class MyClass(PythonHandler):

    def local_python(self):
        """"""
        return "This file are running from Local Python environment"

    def pitagoras(self, a, b):
        """"""
        return math.sqrt(a**2 + b**2)
```

(continues on next page)

(continued from previous page)

```
    def test(self):
        """"""
        return True
```

Next, it's necessary to configure the main script to load this module. In this case, you should include `python=('python_foo.py', 'MyClass')` in the arguments of `RadiantServer`. This setup ensures that the specified Python class is correctly loaded and integrated into the server environment.

This configuration defines how we use the module and the name we assign to it. In this instance, we are using `MyClass` as the name for our module.

```
[ ]: class BareMinimum(RadiantAPI):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        document.select_one('body') <= html.H1('Radiant-Framework')

        c = self.MyClass.pitagoras(3, 5)
```

The complete code for this implementation is presented below:

```
[ ]: from radiant.framework.server import RadiantAPI, RadiantServer
from browser import document, html

class BareMinimum(RadiantAPI):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        document.select_one('body') <= html.H1('Radiant-Framework')

        c = self.MyClass.pitagoras(3, 5)

if __name__ == '__main__':
    RadiantServer(
        'BareMinimum',
        python=('python_foo.py', 'MyClass'),
    )
```

6.4 Brython Enhancement

While Brython stands out for its excellence, it's noted that its interaction with HTML could be further enhanced. This observation particularly pertains to the dynamic integration and manipulation of HTML elements within Brython scripts.

```
[1]: from radiant.framework import html, select
```

The Radiant framework introduces some modules and functions that serve as replacements for Brython's standard modules. These modules are designed to extend and refine the functionalities typically offered by Brython, tailoring them more specifically for the Radiant framework's context.

6.4.1 select

The select function in Radiant framework offers a unique capability: it allows functions to be applied directly to the list returned, as if it were a single element. This feature significantly simplifies the manipulation of HTML elements, streamlining the process of applying changes or retrieving information.

```
[ ]: selection = select('.my-class')

selection.bind('mouseover', lambda evt: print(evt))
setattr(title.style, 'color', 'cyan')
selection.style.color = 'cyan'
selection.style = {'background-color': 'red', }
```

6.4.2 html Extensions

An innovative addition to the Radiant framework's html module is the classes method. This method is designed to simplify the dynamic management of CSS classes in HTML elements, enhancing the ease and flexibility of styling and theming.

```
[ ]: title = html.H1('Radiant-Framework', Class='my-class')

title.classes.append('new-class')
title.classes.extend(['new-new-class', 'new-new-new-class'])
```

6.4.3 html as Context Manager

A notable functionality of the Radiant framework's html module is its use as a Context Manager. This approach greatly enhances the flexibility in nesting components, a common practice in HTML, by simplifying the structure and syntax required for creating complex HTML hierarchies.

```
[ ]: with html.DIV(style={'background-color': 'blue'}).context(parent):
    with html.DIV().context:
        with html.SPAN().context as span:
            span.html = "Texto de ejemplo"
```

6.4.4 styles object

Radiant introduces the styles object, an innovative way to interact with CSS styles of DOM elements. This object simplifies style manipulation, especially when working with multiple elements, and provides a more intuitive, Pythonic interface.

The styles object can be accessed on a selection of elements, allowing you to get or set styles in a more readable and Python-friendly way.

```
[ ]: selection = select('.my-class')
selection.styles.background_color = 'pink'
```

In this example, background_color is a property of the styles object. It corresponds to the background-color CSS property. Setting this property updates the style of all elements in the selection.

```
[ ]: selection = select('.my-class')
     selection.styles.background_color = 'pink'
```

The styles object allows you to replace multiple styles at once in a more Pythonic manner, avoiding the traditional dictionary syntax.

```
[ ]: selection = select('.my-class')
     selection.styles.background_color = 'red'
     selection.styles.color = 'white'
```

This code replaces the traditional dictionary-based approach:

```
[ ]: selection.style = {'background-color': 'red', 'color': 'white'}
```

6.5 Templates

This feature in the Radian framework enables the use of Brython Templates functionalities through HTML files.

For instance, the following is an example of the `main.html` code:

```
<div class="">
    <h1>Brython Templates</h1>
    <h2>{title}</h2>
</div>

<div class="">
    <h1>Brython Templates b-code</h1>
    <h2 b-code="for item in items:">{item}</h2>
</div>
```

The usage is as follows:

```
[ ]: from radiant.framework.server import RadiantAPI, RadiantServer, render
     from radiant.framework import html


     class StaticApp(RadiantAPI):

         def __init__(self, *args, **kwargs):
             """"""
             super().__init__(*args, **kwargs)
             self.body <= html.H1('Radiant-Framework')
             self.body <= self.main()


         def main(self):
             """"""
             context = {
                 'title': 'Title',
                 'items': [f'item-{i}' for i in range(10)],
```

(continues on next page)

(continued from previous page)

```
        }
        return render('main.html', context)


if __name__ == '__main__':
    StaticApp()
```

This code demonstrates the use of the `render` function within the Radiant framework. It outlines the creation of a `StaticApp` class, which inherits from `RadiantAPI`. The class defines a constructor and a main method. The main method constructs a context containing a title and a list of items, and then it returns the output of the `render` function. This function is used to integrate dynamic data from the context with an HTML template (`main.html`), a common technique in web applications for generating dynamic web pages.

6.6 Editing HTML Meta Tags in Radiant

Radiant introduces a streamlined way to dynamically update the meta tags of an HTML page. This feature allows for easy modification of essential SEO and social media attributes, enhancing the flexibility of web page metadata management.

Meta tags in an HTML document provide important information about the page, which is used by search engines and social media platforms. These tags include details like the page title, description, image, and more. With Radiant, you can dynamically update these tags using the following arguments:

```
page_title
page_favicon
page_description
page_image
page_url
page_summary_large_image
page_site
page_author
page_copyright
```

[ ]:
```
if __name__ == '__main__':
    RadiantServer(
        'BareMinimum',
        host='localhost',
        port=5000,
        brython_version='3.11.2',
        debug_level=0,

        page_title="Example Title",
        page_description="Description of the page",
        # Other arguments...
```

(continues on next page)

(continued from previous page)

```
)
    )
```

- `page_title`: Updates the content of the `<title>` tag and other relevant meta tags.
- `page_favicon`: URL of the favicon for the page.
- `page_description`: Sets the description meta tag, used by search engines and social sharing.
- `page_image`: URL of the image to be used in Open Graph and Twitter Cards.
- `page_url`: Canonical URL of the page.
- `page_summary_large_image`: Specific to Twitter Cards, denotes a large summary image.
- `page_site`: The Twitter username of the website or site creator.
- `page_author`: Author of the page's content.
- `page_copyright`: Copyright information of the page.

Citing Radiant Framework

If you utilize Radiant in your research or project, we kindly ask that you acknowledge it by citing it in your work. Please use the following BibTeX entry:

```
@software{radiant_framework_2023,
  author = {Dunderlab},
  title = {Radiant Framework},
  url = {https://github.com/dunderlab/python-radiant_framework},
  version = {0.1a21},
  year = {2023}
}
```

Dunderlab. (2023). Radiant Framework (Version 0.1a21) [Software]. Available at https://github.com/dunderlab/python-radiant_framework