Spyre Documentation

Release 0.2.0

adam hajari

Apr 27, 2018

Contents

Spyre is a Web Application Framework for providing a simple user interface for Python data projects.

CHAPTER 1

Content

1.1 Requirements

Spyre runs on the minimalist python web framework, cherrypy , with jinja2 templating. At it's heart, spyre is about data and data visualization, so you'll also need pandas and matplotlib.

1.2 Installation

```
$ pip install dataspyre
```

1.3 Getting started

A basic web app is just a set of inputs that a user can change in order to update the state of the app's outputs. In a spyre app we define the attributes of these inputs and outpus as lists of dictionaries and define a set of methods that generate the actual content of the app's outputs. Let's start by looking at at a couple of examples.

1.3.1 Example 1: simple example

```
from spyre import server

class SimpleApp(server.App):
    title = "Simple App"
    inputs = [dict( type='text',
                    key='words',
                    label='write words here',
                    value='hello world',
                    action_id='simple_html_output')]
```

```
    outputs = [dict( type='html',
                     id='simple_html_output')]

    def getHTML(self, params):
        words = params["words"]
        return "Here's what you wrote in the textbox: <b>%s</b>" % words

app = SimpleApp()
app.launch()
```

The SimpleApp class inherits server.App which includes a few methods that you can override to generate outputs. In this case we want our app to display HTML (just text for now) so we'll overide the getHTML method. This method should return a string.

We also need to specify the attributes of our inputs and outputs which we can do by defining the App's inputs and outputs variables.

inputs

This is a list of dictionaries, one dictionary for each input element. In our simple example above, there's only one input, of type "text". We give it a label and initial value with the keys "label" and "value". The value from this input will be used as an input parameter when generating the outputs (html in this case), so we need to also give it a key that we can reference in the getHTML method. "action_id" is an optional variable that equals id from either an output or a control element (we'll get to controls in the next example). When action_id is defined, a change in the input will result in either an update to the referenced output or a call to the functions connected to the referenced control.

outputs

An output's *type* can be "plot", "image", "table", "html", or "download". In addition to the type, we also need to provide a unique id (must start with a letter). If this output is suppose to get updated on execution of one of the controls specified in the list of controls, we need to also specify the control_id of that controller (which we'll see in the next example). All outputs get generated on page load by default. If we want an output *not* to load on the page load, we can also specify an "on_page_load" attribute and set it to False.

controls

Controls are one mechanism by which a spyre app can update its outputs. A control's *id* can be referenced by either an input or an output. When an output references the control's id, executing the control updates that output. When an input references the controld's id (via the "action_id"), updating the input executes the control. The two control *type* options are "button" and "hidden". "button" will add a button to the left panel. No control is added to the left-panel for control_types "hidden" (this is useful for linking a single input action to multiple outputs).

generating an output

Let's get back to our getHTML method. Notice that it takes a single argument: params. params is a dictionary containing:

1. all of the input values (where the key is specified in the input dictionary)
2. the output_id for the output that needs to get created. You usually don't need to do anything with this.

With the exception of the input type "checkboxgroup", the value of each of the params elements is a string. The string returned by getHTML will be displayed in the right panel of our Spyre app.

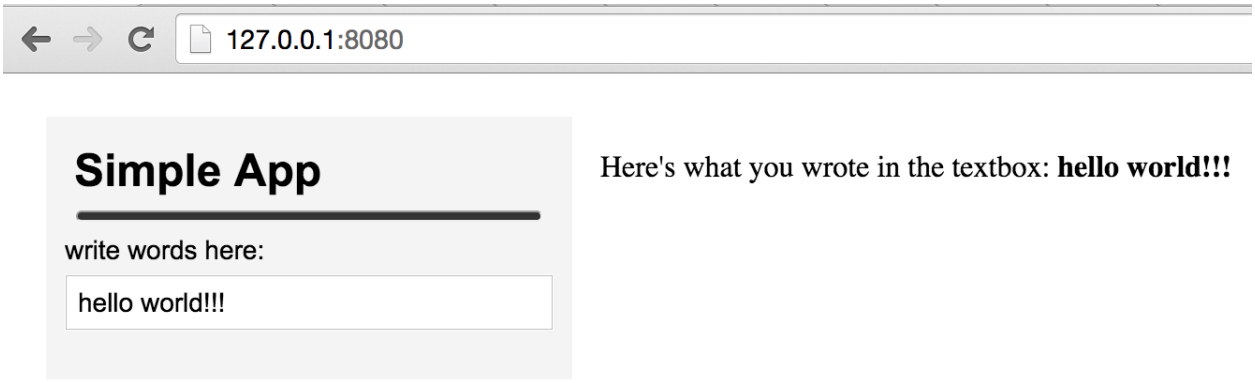
launching the app

To launch our app we just need to create an instance of our SimpleApp class and call the launch method. Assuming you name this file "simple_sine_example.py" you can launch this app from the command line with:

```
$ python simple_sine_example.py
```

The output will indicate where the app is being served (http://127.0.0.1:8080 by default)

If all goes smoothly your spyre app should look like this:



1.3.2 Example 2: tabs and tables

Let's look at another example to introduce controls, tabs, and a second output type, tables. Many apps will require multiple outputs. In these cases, it's often cleaner to put each output in a separte tab.

In the example below we'll show historical stock data in a line graph and a table, each in it's own tab. Since inputs can only have a single action_id (and we have two outputs), we'll need to introduce a button control in order to update both outputs:

```
from spyre import server

import pandas as pd
import urllib2
import json

class StockExample(server.App):
    title = "Historical Stock Prices"

    inputs = [{        "type":'dropdown',
                    "label": 'Company',
                    "options" : [ {"label": "Google", "value":"GOOG"},
                                  {"label": "Yahoo", "value":"YHOO"},
                                  {"label": "Apple", "value":"AAPL"}],
                    "key": 'ticker',
                    "action_id": "update_data"}]

    controls = [{    "type" : "hidden",
                    "id" : "update_data"}]

    tabs = ["Plot", "Table"]

    outputs = [{ "type" : "plot",
```

```
                    "id" : "plot",
                    "control_id" : "update_data",
                    "tab" : "Plot"},
                { "type" : "table",
                    "id" : "table_id",
                    "control_id" : "update_data",
                    "tab" : "Table",
                    "on_page_load" : True }]

    def getData(self, params):
        ticker = params['ticker']
        # make call to yahoo finance api to get historical stock data
        api_url = 'https://chartapi.finance.yahoo.com/instrument/1.0/{}/chartdata;
↪type=quote;range=3m/json'.format(ticker)
        result = urllib2.urlopen(api_url).read()
        data = json.loads(result.replace('finance_charts_json_callback( ','')[:-1])
↪# strip away the javascript and load json
        self.company_name = data['meta']['Company-Name']
        df = pd.DataFrame.from_records(data['series'])
        df['Date'] = pd.to_datetime(df['Date'],format='%Y%m%d')
        return df

    def getPlot(self, params):
        df = self.getData(params).set_index('Date').drop(['volume'],axis=1)
        plt_obj = df.plot()
        plt_obj.set_ylabel("Price")
        plt_obj.set_title(self.company_name)
        fig = plt_obj.get_figure()
        return fig

app = StockExample()
app.launch(port=9093)
```

There's a few things to point out here:

1. This app uses a dropdown input type. It still has a label and variable_name (that's common to all input types), but you now also need to enumerate all of the options for the dropdown menu. For each of the options, "label" is displayed in the menu and "value" is value of that input variable when that option is selected.
2. The tabs variable is a list of tab names. These names are used as labels for the tabs as well as html ids so they can't contain any spaces.
3. There's a "table" output type that requires all of the same attribute types as the plot output type.
4. Additionally, we need to specify a "tabs" attribute for each output. This should match the name of one of the items listed in the tabs list.
5. The control variable has control_type, label, and control_id attributes. Each output has an optional control_id attribute which can be used to reference a control. When a control action is taken (such as clicking a button), every output that references that control will be updated.

We're also overriding getData, a method which should fetch or generate the data that will go into the table. Just like getPlot, it takes a params argument which is a dictionary containing all of our input variables. getData should return a pandas DataFrame.

Launch the app just as you did in the previous example. The app now has two tabs.

Plot tab

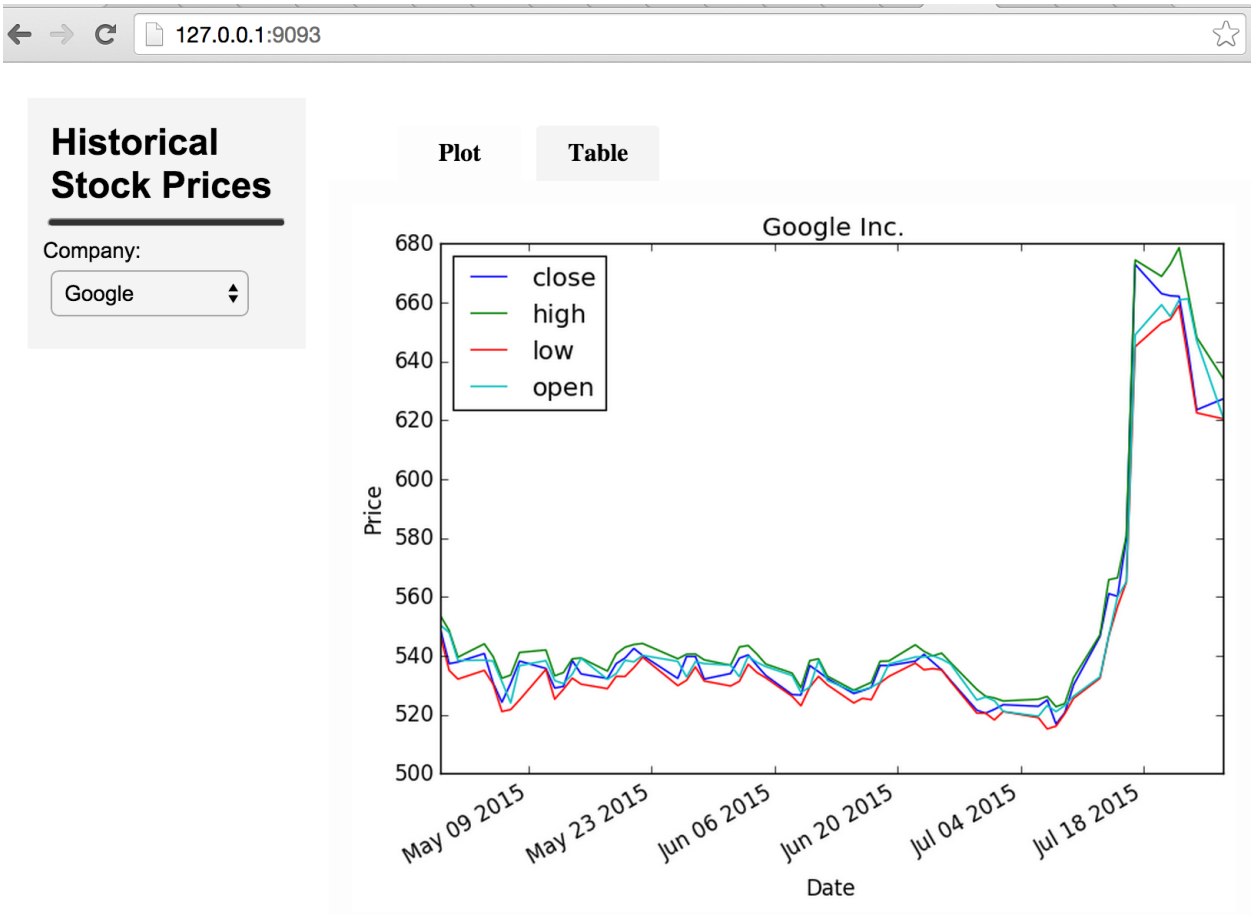
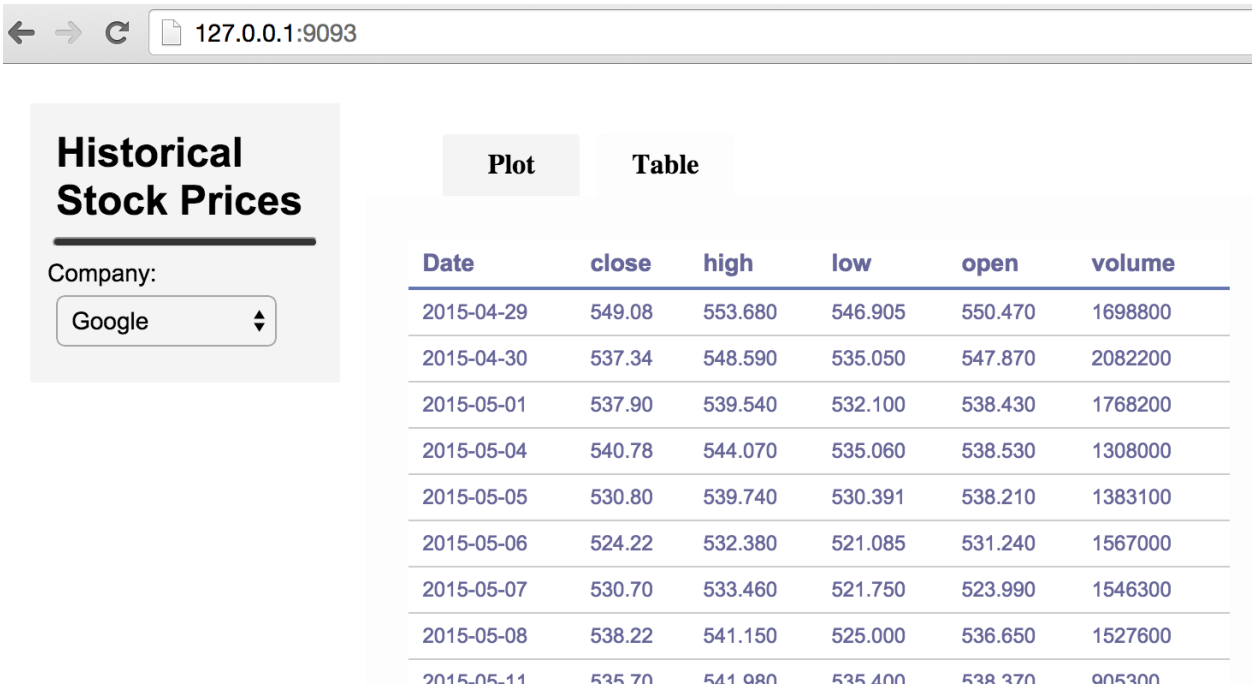

127.0.0.1:9093
Historical Stock Prices
Company:
Google
Plot
Table
Google Inc.
close
high
low
open
Price
680
660
640
620
600
580
560
540
520
500
May 09 2015
May 23 2015
Jun 06 2015
Jun 20 2015
Jul 04 2015
Jul 18 2015
Date

Table tab



1.4 Inputs

Inputs for a spyre app are defined as a list of dictionaries. Each element in the list represents a different input and the attributes for each input are defined in these dictionaries. Each input type has some mandatory attributes: (i.e. type, key) and some optional attributes (i.e. label, value, action_id)

1.4.1 Types

TextBox

This is an example of the text input element

```
{
    "type":'text',
    "label": 'Title',
    "value" : 'Simple Sine Wave',
    "key": 'title',
    "action_id" : "refresh",
}
```

RadioButton

This is an example of the RadioButton input element

```
{
    "type":'radiobuttons',
```

```
    "label": 'Function',
    "options" : [
        {"label": "Sine", "value":"sin", "checked":True},
        {"label":"Cosine", "value":"cos"}
    ],
    "key": 'func_type',
    "action_id" : "refresh",
},
```

Checkbox Group

This is an example of the Checkbox Group input element

```
{
    "type":'checkboxgroup',
    "label": 'Axis Labels',
    "options" : [
        {"label": "x-axis", "value":"x", "checked":True},
        {"label":"y-axis", "value":"y"}
    ],
    "key": 'axis_label',
    "action_id" : "refresh",
}
```

Dropdown

This is an example of the Dropdown input element

```
{
    "type":'dropdown',
    "label": 'Line Color',
    "options" : colors,
    "key": 'color',
    "action_id" : "refresh",
    "linked_key": 'title',
    "linked_type": 'text',
    "linked_value":"hey"
}
```

Slider Input

This is an example of the Slider input element

```
{
    "type":'slider',
    "label": 'frequency',
    "key": 'freq',
    "value" : 2,
    "min" : 1,
    "max" : 30,
    "action_id" : "refresh",
    "linked_key": 'title',
    "linked_type": 'text',
}
```

1.5 Outputs

The attributes for the outputs for a spyre app are defined as a list of dictionaries. Each element in the outputs list should have a corresponding method which generates that output. The output dictionary should specify the type and provide and id. The id must be alphanumeric and cannot start with a number. Outputs can also have a control_id key, which references an id from the controls list.

By default the outputs load on page load using the default values for each of the inputs. If an output should not load on page load, set the 'on_page_load' attribute to False (see example for Download output type)

1.5.1 Generating outputs

There are a few options for generating an output's content.

Overriding the method from the server.App class

Each output type has a corresponding method in the server.App class that gets called whenever an instance of that output get displayed in the app. For instance, if an html output is included in your list of outputs, server.App's getHTML() method gets called everytime that block of html gets loaded.

You can override the methods for each of the output types. For instance, if an app had an html output that was suppose to display the string *Be bold*, you could include this method in your app's class:

```
getHTML(self, params):
    return "be <b>bold</b>"
```

Matching the method name to the output id

If server.App's built-in output method isn't overridden for an output in the app's outputs list, the built-in method will look for a method with a name matching the output id. Suppose, for instance, our app only has one output:

```
outputs = [{ 'type':'html',
             'id':'aphorism1'}]
```

If we do not overide the getHTML method, server.App's getHTML method will look for a method named "aphorism1". We can generate output then by creating an output that matches that name:

```
def aphorism1(self, params):
    return "if it ain't broke, don't fix it."
```

If we have more than one output of the same type, we can use the method naming convention to generate outputs for both of them:

```
outputs = [{ 'type':'html',
             'id':'aphorism1'},
             { 'type':'html',
             'id':'aphorism2'},
             { 'type':'html',
             'id':'aphorism3'},]
```

```
def aphorism1(self, params):
    return "if it ain't broke, don't fix it."

def aphorism2(self, params):
    return "The art of prophecy is very difficult - especially with respect to the␣
↪future."

def aphorism3(self, params):
    return "All you need in this life is ignorance and confidence, and then success␣
↪is sure. "
```

Including a getData method

The getData method can be used to generate the output for tables, plots, or downloads. The getData method should return a pandas dataframe and will be converted into the approriate output. Using getData, an app can generate up to three outputs with a single method.

1.5.2 Output types

Table

```
{
    'type':'table',
    'id':'average_rainfall_table'
}
```

Plot

```
{
    'type':'plot',
    'id':'average_rainfall_linegraph'
}
```

HTML

```
{
    'type':'html',
    'id':'readme'
}
```

Image

```
{
    'type':'image',
    'id':'cat_photo'
}
```

Download

```
{
    'type':'download',
    'id':'results_csv',
    'on_page_load':False
}
```

CHAPTER 2

License

MIT

CHAPTER 3

Indices and tables

- genindex
- modindex
- search