pyxley Documentation

Release 0.0.7

Nicholas Kridler

June 20, 2016

Contents

1 Why Pyxley? 3

1.1 React 3
1.2 Flask 3
1.3 PyxleyJS 3
1.4 Insane Templates 3

2 Core Components 5

2.1 Charts 6
2.2 Filters 6

3 A Basic Pyxley App 7

3.1 JavaScript! 7
3.2 HTML & CSS 8
3.3 Flask 8
3.4 Adding Some Pyxley 8

4 Pyxley In Production 11

4.1 Coming Soon! 11

5 About 13

6 Features 15

7 Installation 17

8 Indices and tables 19

The Pyxley python package makes it easier to deploy Flask-powered dashboards using a collection of common JavaScript charting libraries. UI components are powered by PyxleyJS.

CHAPTER 1

Why Pyxley?

Pyxley was born out of the desire to combine the ease of data manipulation in pandas with the beautiful visualizations available in JavaScript. It was largely inspired by Real Python. The goal was to create a library of reusable visualization components that allow the quick development of web-based data products. For data scientists with limited JavaScript exposure, this package is meant to provide generic visualizations along with the ability to customize as needed.

1.1 React

Pyxley utilizes Facebook's React. React is only concerned with the UI, so we can use it for only the front-end portion of our web-applications.

1.2 Flask

Flask is a great micro web-framework. It allows us to very easily stand up web applications and services.

1.3 PyxleyJS

Pyxley relies on a JavaScript library that heavily leverages React and existing visualization libraries. Wrappers for common libraries have been created so that the user only needs to specify the type of chart they want, the filters they wish to include, and the parameters specific to that visualization.

To download or contribute, visit PyxleyJS.

1.4 Insane Templates

A lot of other projects rely on a set of complicated templates that attempt to cover as many use cases as possible. The wonderful thing about React is the ability to create compositions of components. This means that we only need a single template: a parent component that manages all of the child components. With this layout, we can use factories within JavaScript to create the components using the supplied parameters. Organizing the code in this way allows us to create a common framework through which we can create a variety of different components.

For example, the underlying interface for a Dropdown Button and a Line Chart are the same. The only difference is the options supplied by the user. This provides a really easy way to integrate with Python. We can use Python for organizing the types and options. PyReact can then be used to transform to JavaScript.

CHAPTER 2

Core Components

In Pyxley, the core component is the `UILayout`. This component is composed of a list of `charts` and `filters`, a single React component from a JavaScript file, and the Flask app.

```
# Make a UI
from pyxley import UILayout
ui = UILayout(
    "FilterChart",
    "./static/bower_components/pyxley/build/pyxley.js",
    "component_id")
```

This will create a UI object that's based on the `FilterChart` React component in `pyxley.js`. It will be bound to an html `div` element called `component_id`.

If we wanted to add a filter and a chart we could do so with the following

```
# Make a Button
cols = [c for c in df.columns if c != "Date"]
btn = SelectButton("Data", cols, "Data", "Steps")

# Make a FilterFrame and add the button to the UI
ui.add_filter(btn)

# Make a Figure, add some settings, make a line plot
fig = Figure("/mgchart/", "mychart")
fig.graphics.transition_on_update(True)
fig.graphics.animate_on_load()
fig.layout.set_size(width=450, height=200)
fig.layout.set_margin(left=40, right=40)
lc = LineChart(sf, fig, "Date", ["value"], init_params={"Data": "Steps"}, timeseries=True)
ui.add_chart(lc)
```

Calling the `ui.add_chart` and `ui.add_filter` methods simply adds the components we've created to the layout.

```
app = Flask(__name__)
sb = ui.render_layout(app, "./static/layout.js")
```

Calling `ui.render_layout` builds the JavaScript file containing everything we've created.

2.1 Charts

Charts are meant to span any visualization of data we wish to construct. This includes line plots, histograms, tables, etc. Several wrappers have been introduced and more will be added over time.

2.1.1 Implementation

All `charts` are `UIComponents` that have the following attributes and methods

- An endpoint route method. The user may specify one to override the default.
- A `url` attribute that the route function is assigned to by the flask app.
- A `chart_id` attribute that specifies the element id.
- A `to_json` method that formats the json response.

2.2 Filters

Filters are implemented in nearly the same way that `charts` are implemented. The only difference is the lack of the `to_json` method.

CHAPTER 3

A Basic Pyxley App

Here we will go through building a basic web-application using Pyxley and Flask.

I recommend visiting the Real Python blog for a great intro to a basic app.

```
App
|   package.json
|   .bowerrc
|   bower.json
|
---project
    |   app.py
    |   templates
    |
    ---static
        |   css
        |   js
```

Some notes about the above structure

- This assumes that you are running the app from the `project` folder.
- Any JavaScript created by the app should go in the `js` folder.

3.1 JavaScript!

3.1.1 Node & NPM

At the highest level, Node is our biggest JavaScript dependency. PyReact needs a JavaScript runtime and Node fills that role. In addition, we can use NPM to install Bower. This document won't show you how to get Node or NPM, but for Mac OS X users, you can get it through homebrew.

Once you have NPM, simply type

```
npm install -g bower
```

This will give bower global access so that you can execute it.

3.1.2 Bower

Bower is a great package manager. In the examples directory, each of the examples has a `bower.json` file. This file contains all the necessary packages for the app to run. Install the packages by typing `bower install` in the directory of the `bower.json` file. There is a file called `.bowerrc` that specifies the location that bower will store the libraries.

If the `.bowerrc` file looks like

```
{
    "directory": "./project/static/bower_components"
}
```

Then the folder structure in static should look like the figure below after installation.

```
---static
    |   css
    |   js
    |   bower_components
```

3.2 HTML & CSS

HTML templates used by flask are stored in the `templates` folder. For our purposes, we only need a really basic template that has a single `div` element.

```
<div id="component_id"></div>
```

Everything in our app will be tied to this single component.

3.2.1 CSS

We store any additional CSS we need in the `static\CSS` folder.

3.3 Flask

Courtesy of the Flask website, "Hello, World!" in Flask looks like the code below.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

We simply need to build upon this.

3.4 Adding Some Pyxley

Let's start by importing some simple things and building upon the example above. We will import some `pyxley` components, create a UI, and load a data frame.

```
# Import flask and pandas
from flask import Flask, render_template
import pandas as pd

# Import pyxley stuff
from pyxley import UILayout
from pyxley.filters import SelectButton
from pyxley.charts.mg import LineChart, Figure

# Read in the data and stack it, so that we can filter on columns
df = pd.read_csv("fitbit_data.csv")
sf = df.set_index("Date").stack().reset_index()
sf = sf.rename(columns={"level_1": "Data", 0: "value"})

# Make a UI
ui = UILayout(
    "FilterChart",
    "./static/bower_components/pyxley/build/pyxley.js",
    "component_id")

# Create the flask app
app = Flask(__name__)
```

At this point we now have some data and a layout to build upon. Adding the code below will add a dropdown select button and a line plot.

```
# Make a Button
cols = [c for c in df.columns if c != "Date"]
btn = SelectButton("Data", cols, "Data", "Steps")

# Make a FilterFrame and add the button to the UI
ui.add_filter(btn)

# Make a Figure, add some settings, make a line plot
fig = Figure("/mgchart/", "mychart")
fig.graphics.transition_on_update(True)
fig.graphics.animate_on_load()
fig.layout.set_size(width=450, height=200)
fig.layout.set_margin(left=40, right=40)
lc = LineChart(sf, fig, "Date", ["value"], init_params={"Data": "Steps"}, timeseries=True)
ui.add_chart(lc)
```

Finally, we need to transform all of the inputs into javascript and render the HTML template. This assumes that the `index.html` template has all of the javascript and css files specified in the HTML.

```
sb = ui.render_layout(app, "./static/layout.js")

@app.route('/', methods=["GET"])
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run()
```

Now when you run `app.py` from the `project` folder, accessing your localhost on port 5000 will lead to a simple plot. This example was adapted from the metricsgraphics example in the Pyxley repository.

CHAPTER 4

Pyxley In Production

4.1 Coming Soon!

CHAPTER 5

About

Read more about Pyxley at the Multithreaded blog.

CHAPTER 6

Features

- Simple web-applications using Flask, PyReact, and Pandas
- Integration with d3.js based libraries such as MetricsGraphics
- Ability to integrate custom React UIs

CHAPTER 7

Installation

Install Pyxley by running:

```
pip install pyxley
```

CHAPTER 8

Indices and tables

- genindex
- modindex
- search