webkitpony Documentation

Release 0.1

Toni Michel

May 24, 2014

Contents

"building webapp-like desktop applications in python"

webkitpony is a micro framework to build dektop applications with web technologies on the basis of the python binding of the webkit rendering engine (http://code.google.com/p/pywebkitgtk/).

The project is hosted on github: https://github.com/tonimichel/webkitpony

Artwork by IYOMI

CHAPTER 1

Motivation

Building desktop applications with standard toolkits like GTK works fine - but tweaking the UI beyond the boundaries of window managers is exhausting. In contrast, building a web ui with HTML, Javascript and CSS is quite flexible. Acctually, the motivation behind webkitpony was to stay in a djangonaut-familiar environment when it comes to build desktop applications. As the basic technology (the webkit rendering engine) was given - it was about playing around and creating a simple-to-use django-like development process. The result was webkitpony.

At schnapptack, we use this approach for building desktop applications or solutions that explicitely need a non-browser client. Often we also combine having native logic and remote logic from a web application.

CHAPTER 2

Goal

The goal of webkitpony is to provide an alternative to standard desktop application development approaches. It especially targets on developers familiar with django. However, the framework is so simple, that even non-django programmers will get the point fastly.

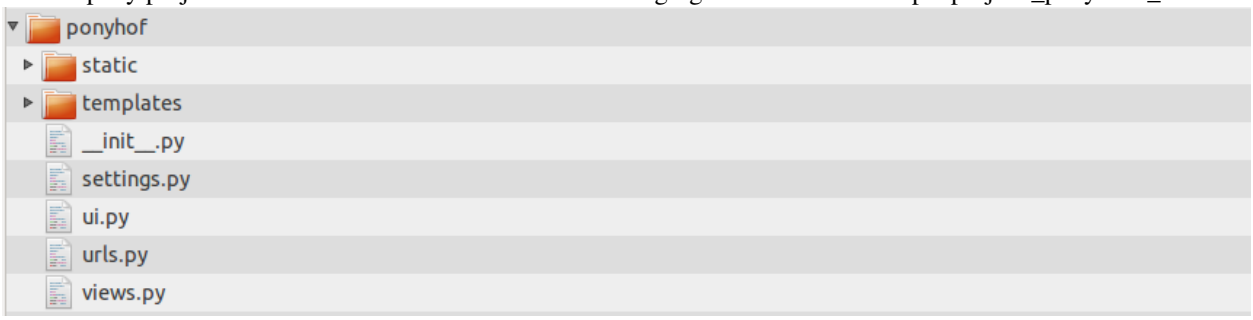
CHAPTER 3

Understanding webkitpony

3.1 Understanding webkitpony

3.1.1 Project structure

A webitpony project follows a certain structure. The following figure shows the example project _ponyfarm_:



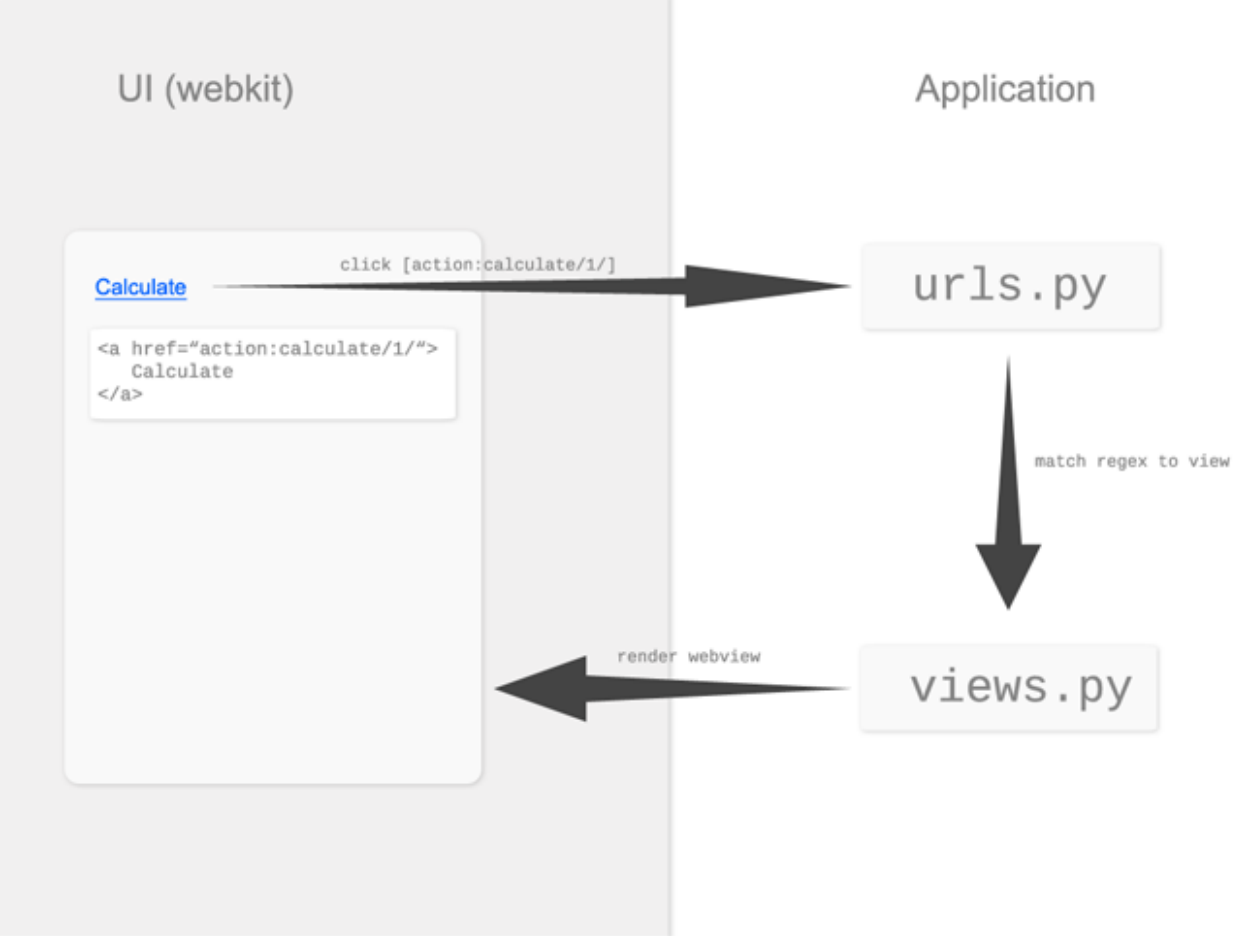
Let's start with a brief overview:

- `urls.py`: contains a list of tuples, each matching a regular expression (representing a url) to a view function.
- `views.py`: contains the view functions referenced in the `urls.py` module.
- `settings.py`: contains your project's settings like e.g. whether to enable the webkit inspector or not.
- `ride.py`: the starting point for riding your pony: `python ride.py`.
- `templates`: contains your templates with jinja2 template syntax available.
- `static`: contains the static files of your project like css and javascript.

Those who are familiar with django might already have become an idea of the webkitpony principle.

3.1.2 HTML - Python Interaction

To understand webkitpony, it is important to understand the communication between the web ui and the python code:



On the left-hand side we have the webkit (represented as a webview object), which we'll call UI. On the right-hand side we have the python application.

Whenever a link is clicked, the url of that link, namely the href attribute is sent to the webkitpony url dispatcher, which looks up the url patterns and triggers the view, passing the webview and url parameters if specified.

3.1.3 A full example

As we saw in the figure before, the html contains a button "calculate". Note that all links are prefixed with 'action:' which is used to distinguish between referencing urls and files.

```
<body>
    <a id="calculate" href="action:/calculate/1/">Calculate</a>
</body>
```

The `urls.py` module defines a variable `urlpatterns`. It consists of tuples matching urls to callback functions, which we call views. Each tuple is a regular expression (describing a url) and a view (being invoked if the regex matches). In case a regex contains a grouped expression, its value is passed as parameter to the view function.

```
urlpatterns = (
    (r'^calculate/(?P<id>[0-9]+)/$', views.calculate)
)
```

The `views.py` module defines the view function previously registered on the *calculate url*. The first paramter is always the `webview` object representing the webkit. Further parameters depend on the pattern. In this example `id` is

passed.

```
def calculate(webview, id):
    # do some stuff
    return webview.render('myapplication/myview.html', {
        'this': 'is',
        'the': 'template context'
    })
```

})

As you might notice, this principle is similar to django except, that the view takes a webview object instead of request and returns webview.render instead of a HttpResponse. Again the `webview.render` is comparable to `django.shortcuts.render`. It takes a template and a template context. The templates themselfs build upon jinja2.

3.1.4 Passing data from HTML to Python

Sometimes we want to send some form data from the UI to our python-side application. As we are not in the web, we do not have POST or GET. So, we need a way to pass data from html to the application. For this purpose *webkitpony* provides a Javascript connector enabling an Ajax-like JSON communicaton between javascript and python code. Consider the following example:

```
<body>

    <form id="myform">
        <input type="text" value="" name="first_name">
        <input type="submit" value="save">
    </form>


    <script>
        var form = $('#myform')

        form.submit(function() {
            var data = {}

            form.find('input[type="text"]').each(function() {
                var field = $(this)
                data[field.attr('name')] = field.val()
            })

            webkitpony.send('/calculate/1/', {data: data}, function(response) {
                console.log('We sent ' + data + ' and received ' + response)
            })

            return false
        })
    </script>
</body>
```

To send the form to the application we bind a submit event, construct our json-serializable data object and invoke `webkitpony.send(url, data, callback)`. Similarily to a non-javascript link click, the url is routed through our project's `urls.py` invoking the matching view function:

```
def calculate(webview, id):
    result = backend.perform_calculation(webview.DATA)
    return webview.json_response({'result': result})
```

The view function unpacks the data from the webview object (similiarily to request.POST). Instead of returning `webview.render` `webview.json_response(result)` is returned which does not re-rendered the webview. Instead json is passed back to `webkitpony.send` which finally executes the callback function.

Of course, we can also use `webkitpony.send` for links:

```
<body>

    <a id="mylink">Calculate</a>

    <script>
        $('#mylink').click(function() {
            webkitpony.send('/calculate/1/', {data: 'some data'}, function(response) {
                console.log('We sent ' + data + ' and received ' + response)
            })
            return false
        })
    </script>
</body>
```

This might be useful to build Javascript applications without "reload".

3.2 The webview object

The webview object is comparable with the request and response objects of a django request/response cycle and is always passed as first parameter to any view. The public interface provides the following methods and attributes:

`webview.render(template, context)`

Renders the webview. Parameter `template` is a string specifying the path to your template relative to your project's template dir. Parameter `context` is a dictionary applied as template context. Templates build upon jinja2.

`webview.json_response(data)`

Usefull in combination with the *client-side* `webkitpony.send(url, data, callback)` from `webkitpony.js`. Takes a single parameter `data` which must be a json-serializable data structure, e.g. a dict. Triggers the callback method of the foregoing `webkitpony.send` passing data as *response*.

`webview.data`

Comparable with `request.POST`. Provides the python deserialized dictionary of the foregoing `webkitpony.send` code.

3.3 The webkitpony.js

The `webkitpony.js` contains Javascript utils to enable the communication between the UI and the python-side of your application. At the moment webkitpony.js requires jQuery, which is inlcuded in the download package.

For now `webkitpony.js` provides a single method:

`webkitpony.send(url, data, callback)`

Parameter `url` is the url being processed by the pony's url dispatcher. `data` is a json-serializable object which is sent to the application being available in our views via `webview.DATA`. `callback` is a simple function taking a single parameter `response`. `response` contains the data sent back by the application.

3.4 The settings module

`settings.DEBUG`

Boolean indicating whether debug or development mode is enabled. If so, the webkit dom inspector and Javascript console as well as the right click context menu of webkit is availbale.

`settings.WITDH`

The default width when the pony is started.

`settings.HEIGHT`

The default height when the pony is started.

`settings.RESIZABLE`

Boolean indicating whether the window is resizable or not.

`settings.URLCONF`

Specifies the project's urlconf module.

CHAPTER 4

Getting started

4.1 Install webkitpony

4.2 Tutorial

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*