The 20 Minute Wiki

TurboGears concepts and tutorial

Bologna, 9/5/2007

Stefano Zacchiroli
zack@cs.unibo.it

Disclaimer

- Some stuff (shamelessly) took from
  - the 20 minute wiki tutorial http://docs.turbogears.org/1.0/Wiki20/Page1
  - Crhistopher Arndt's talk at RuPy conference 2007 http://chrisarndt.de/talks/rupy/

Part I

TurboGears Concepts

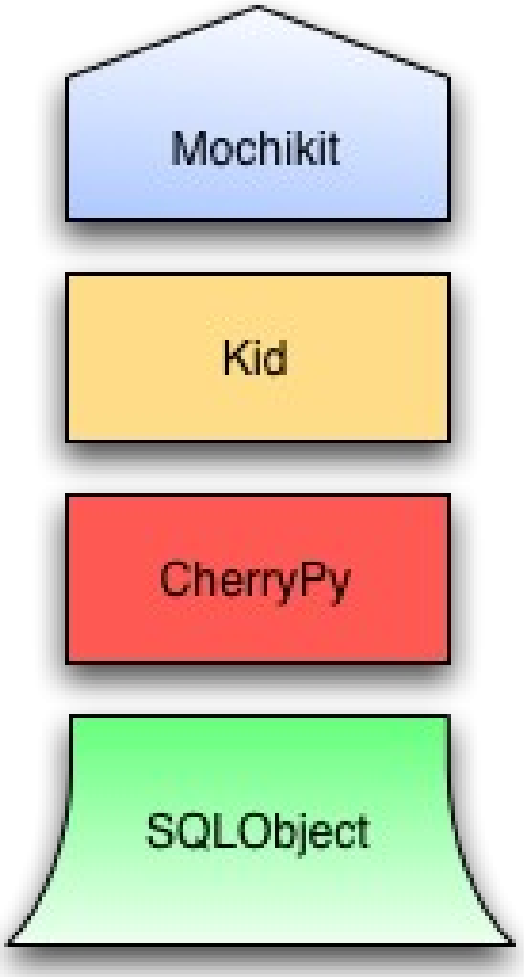
What is TurboGears?

- a python web framework
  - comparable to Django and Ruby on Rails (the latter Ruby-based)
  - Open Source (MIT license)
  - still young (1st public version autumn 2005)
  - buzzword compliant: MVC, REST, AJAX

What can it be used for?

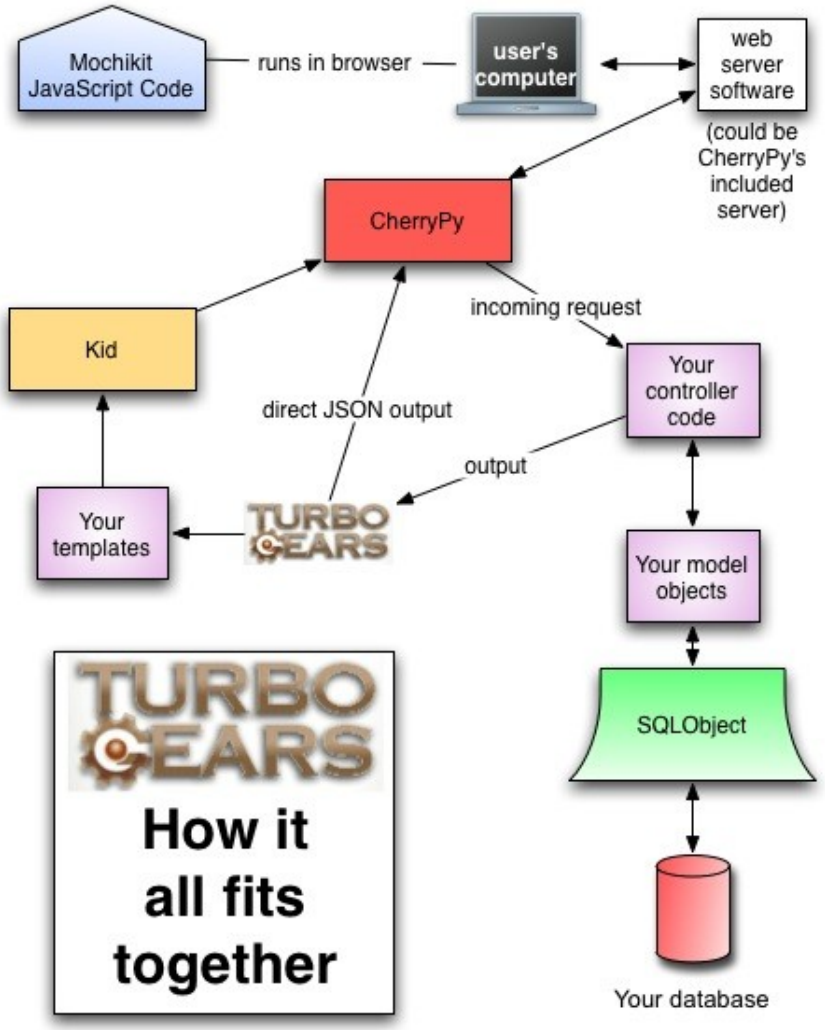
- “classic” web (1.0/2.0/whatever) apps ...
- http://docs.turbogears.org/1.0/SitesUsingTurboGears
  - aggregators
  - blogs
  - social neworking
  - ...
- ... i.e. database-frontends on the web!

Which components?



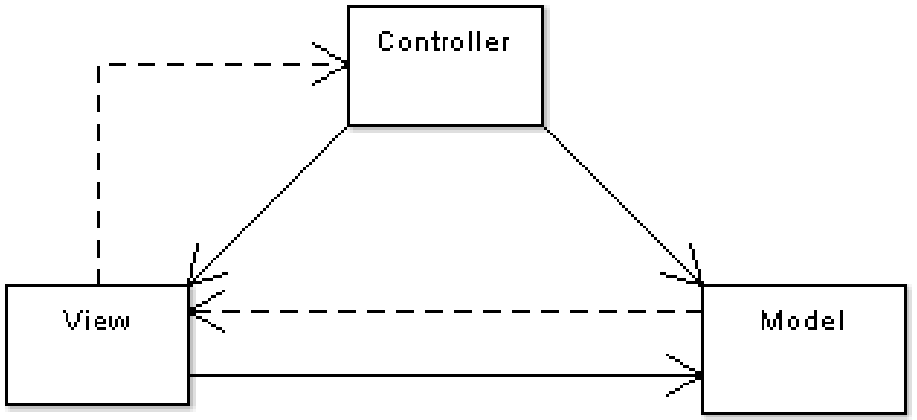
- philosophy: reuse existing stuff for
  - db abstraction
  - application server
  - template engine
  - javascript
- other bits:
  - formencode, nose, simplejson

Putting it all together



- the big picture of component interactions
- da http://docs.turbogears.org

MVC: Model / View / Controller



- buzzword ... but
  - helps separating concerns
- foster reusable components
- on the web:
  - db / template / data manipulation

Recipe

1. scaffold (i.e. tg quickstart)
2. code the model
3. create the db
4. fill db with sample data
5. design your URLs
6. write controller methods
7. write templates
8. bells and whistles: CSS / JavaScript
9. deploy
10. (shuffle and iterate over and over again)

Part II

The 20 Minute Wiki Tutorial

Scaffold

- create a skeleton project from scratch

```
tg-admin quickstart
Enter project name: Wiki 20
Enter package name [wiki20]: wiki20
Do you need Identity (usernames/passwords) in
this project? [no] no

...output...

cd Wiki-20
```

Now serving ...

- let's test what we have

```
python start-wiki20.py

# or ./start-wiki20.py on *nix
```

- now just visit http://localhost:8080/
- easy?
- a working (yet useless) web app

Code the model

- first of all we need a wiki page model
- edit wiki20/model.py

```
class Page(SQLObject):
    pagename = UnicodeCol(alternateID=True,
        length=30)
    data = UnicodeCol()
```

- note: automatic reloading upon save
  - beware of the “5 second rules”

Create the DB

- database configuration in dev.cfg
  - per default relies on SQLite
- create the table (from the model) without a bit of SQL

```
tg-admin sql create
```

Create a template

- create a template

```
cp wiki20/templates/welcome.kid
wiki20/templates/page.kid
```

- edit wiki20/templates/page.kid, bits:

```
<title> ${page.pagename} - 20 Minute Wiki </title>
...
<div class="main_content">
<div style="float:right; width: 10em">
 Viewing <span py:replace="page.pagename">Page Name Goes
Here</span>
 <br/>
 You can return to the <a href="/">FrontPage</a>.
</div>
...
<div py:replace="XML(data)">Page text goes here.</div>
```

Working on templates

- page.kid and other templates can be opened directly in a browser ...
  - “py:” is a separate namespace
- ... and hence can be edited by web designers with their extra-cool apps
- just be careful to provide “sample” data in the template

Write a controller

- in wiki20/controllers.py

```
import turbogears
from turbogears import controllers, expose
from wiki20.model import Page
from docutils.core import publish_parts

class Root(controllers.RootController):
    @expose(template="wiki20.templates.page")   #1
    def index(self , pagename="FrontPage"):     #2
        page = Page.byPagename(pagename)        #3
        content = publish_parts(page.data,
            writer_name="html")['html_body']    #4
        return dict(data=content, page=page)    #5
```

Add sample data

- sample data can be easily provided using

```
tg-admin toolbox
```

- visit http://localhost:7654
  - choose “CatWalk”
  - and add a “FrontPage” page

Add page editing

- wiki20/templates/edit.kid, bits:

```
...
<form action="save" method="post">
 <input type="hidden" name="pagename"
        value="${page.pagename}"/>
 <textarea name="data" py:content="page.data"
           rows="10" cols="60"/>
 <input type="submit" name="submit" value="Save"/>
</form>
...
```

- ... and its controllers.py method:

```
@expose("wiki20.templates.edit")
def edit(self, pagename):
    page = Page.byPagename(pagename)
    return dict(page=page)
```

Add page editing (cont.)

- the link in wiki20/templates/page.kid:

```
...
<div py:replace="XML(data)">Page text goes here.</div>
<p><a href="${tg.url('/edit',
      pagename=page.pagename)}">Edit this page</a>
</p>
...
```

- save action backend (controller method)

```
@expose()
def save(self, pagename, data, submit):
    page = Page.byPagename(pagename)
    page.data = data
    turbogears.flash("Changes saved!")
    raise turbogears.redirect("/", pagename=pagename)
```

Add page listing

- wiki20/templates/pagelist.kid

```
<h1>All Of Your Pages</h1>
<ul>
 <li py:for="pagename in pages">
  <a href="${tg.url('/' + pagename)}"
     py:content="pagename">Page Name Here.</a>
 </li>
</ul>
```

- controller method:

```
@expose("wiki20.templates.pagelist")
def pagelist(self):
    pages = [page.pagename for page in
             Page.select(orderBy=Page.q.pagename)]
    return dict(pages=pages)
```

You want AJAX? We got AJAX!

- let's AJAX-ify page listing
- step 1: expose the data in an JavaScript friendly method: JSON
- @expose pagelist via JSON too

```
@expose("wiki20.templates.pagelist")
@expose("json")
def pagelist(self):
  ...
```

- visit: http://localhost:8080/pagelist?tg_format=json

We got AJAX! (cont.)

- MochiKit: client-side JavaScript framework
- add to wiki20/config/app.cfg:

```
tg.include_widgets = ['turbogears.mochikit']
```

  - (then restart the app)
- add id-s to what we want to change client-side later on, in master.kid:

```
<p>View the <a id="pagelist"
                href="${tg.url('/pagelist')}">
    complete list of pages.</a>
</p>
<div id="pagelist_results"></div>
```

We got AJAX! (cont.)

- JavaScript show time, still in master.kid:

```
<script type="text/javascript">
addLoadEvent(function(){
connect($('pagelist'),'onclick', function (e) {
  e.preventDefault();
  var d = loadJSONDoc("${std.url('/pagelist',
                          tg_format='json')}");
  d.addCallback(showPageList); });
});
</script>
```

- i.e. connect an onclick callback “showPageList”

We got AJAX! (cont.)

- JavaScript show time, here is the callback:

```
<script type="text/javascript">
function showPageList(result) {
    var currentpagelist = UL(null,
       map(row_display, result["pages"]));
    replaceChildNodes("pagelist_results",
       currentpagelist);
}
function row_display(pagename) {
    return LI(null, A({"href" :
      "${std.url('/')}" + pagename}, pagename))
}
</script>
```

That's it!