web frameworks design comparison

draft - please help me improve it

focus on Model-View-Controller frameworks

Controllers

In Rails

```
class MyTestController < ApplicationController
   def index
      render_text “Hello World”
   end
end
```

The name of the class has to match the name of the controller file.

Controllers

In Django

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello World”)
```

Django is explicit, you need to import all functions you use.

Controllers

In Cherrypy and TurboGears 1.0

```
import cherrypy

class MyRoot:

@cherrypy.expose()
def index(self):
     return "Hello World"
```

Cherrypy, Turbogears, and Pylons are also explicit. You need to import all functions you want to use.

Controllers

In web2py

```
def index():
     return "Hello World"
```

web2py is similar to Rails. It imports for you all the web2py keyword. Often, like in this case, you do not need any.

Get/Post requests

In Rails

```
class MyTestController < ApplicationController
   def index
      render_text "Hello "+params[:who]
   end
end
```

GET and POST variables are passed via params but other request parameters (client ip for example) are passed via a different mechanism.

Get/Post requests

In Django

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello World %s” % \
          request.REQUEST[‘who’])
```

Nice, simple. The request contains all the info. You can use .GET or .POST instead of .REQUEST to be more specific.

Get/Post requests

In Cherrypy and TurboGears 1.0

```
import cherrypy

class MyRoot:

@cherrypy.expose()
def index(self,who):
     return "Hello %s" % who
```

GET and POST variables are passed via arguments of the action, but other request parameters (client ip for example) are passed via a different mechanism.

Get/Post requests

In web2py

```
def index():
    return "Hello %s" % request.vars.who
```

Similar to Django. All request data is in one place. You can use .get_vars and .post_vars instead of .vars to be more specific.

Dispatching

In Rails URL http://hostname/MyTest/index gets mapped into

```
class MyTestController < ApplicationController
   def index
      render_text “Hello World”
   end
end
```

By default Rails does not allow running multiple apps without running multiple copies of Rails, since the name of the app is not in the URL, only the controller name (MyTest) and the action name (index) appear.

This can be changed by configuring routes.

Dispatching

In Django you need to edit url.py to map URLs into actions

```
from django.conf.urls.defaults import *

urlpatterns = patterns('',
      (r'^index$', myapp.mycontroller.index),
)
```

This is the equivalent of Rails' routes and it requires using regular expressions.

There is no default. You need one entry in url.py for every action.

Dispatching

In Cherrypy and TurboGears 1.0

```
import cherrypy

class MyRoot:

@cherrypy.expose()
def index(self,who):
     return "Hello %s" % who
```

Works very much like Rails and default mapping between URL and action can be overwritten.

Dispatching

In web2py a URL like http://hostname/myapp/mycontroller/index calls

```
def index():
     return "Hello %s" % request.vars.who
```

Similar to Rails and Charrypy but, by default the URL requires that you specify the name of the app. This allows web2py to run multiple apps without using routes.

Web2py has its own version of routes that supports two different syntaxes (with and without regular expression) to overwrite the mapping and reverse mapping as well.

Calling Views

In Rails

```
class MyTestController < ApplicationController
   def index
      @message="Hello World"
   end
end
```

It calls the default view (MyTest/index) which renders the page. The variables marked by @ are global vars and are passed to the view.

Notice that if the view is not defined, this results in an error message.

Calling Views

In Django

```
from django.shortcuts import render_to_response

def index(request):
    return render_to_response("index.html”,
          {‘message’:’Hello World’})
```

This is the short way of doing it in Django. You have to specify the view name “index.html” since there is no default. Parameters are passed via a dictionary.

You get an error message if the view is not defined.

Notice that in Django a view is called a template and a controller is called a view.

Calling Views

In TurboGears 1.0 with Cherrypy

```
import turbogears
from turbogears import controllers, expose

class MyRoot(controllers.RootController):

@expose(template="MyApp.MyRoot.index")
def index(self):
     return dict(message=”Hello World”)
```

The view is specified in the expose decorator.

Calling Views

In web2py

```
def index():
     return dict(message="Hello World")
```

The last line works like Cherrypy but by default it looks for a view called "mycontroller/index.html" in "myapp". If this view does not exist it uses a generic view to show all variables returned in the dictionary.

The default can be overwritten with response.view='filename.html'

Views

In Rails

```
<table>
 <% @recipes.each do |recipe| %>
  <tr>
   <td><%= recipe.name %></td>
  </tr>
 <% end %>
</table>
```

It allows full Ruby in views but:

- it does not escape strings by default (unsafe)

- <% %> requires a special editor since < > are special in HTML

Views

In Django

```
<table>
{% for recipe in recipes %}
   <tr>
     <td>{{recipe.name}}</td>
   </tr>
{% endfor %}
</table>
```

The choice of {% %} and {{ }} tags is good because any HTML editor can deal with them.

The code looks like Python code but it is not (notice the “endfor” which is not a python keyword. This limits what you can do in views.

Views

Kid or Genshi in TurboGears 1.0 or Cherrypy

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:py="http://purl.org/kid/ns#">
...
<table>
   <tr py:for="recipe in recipes" >
      <td py:content=”recipe.name”>
      </td>
   </tr>
</table>
```

This allows full Python quotes py:for=”...” but it can only be used to generate HTML/XML views, not dynamical JavaScript for example.

Views

In web2py

```
<table>
 {{for recipe in recipes:}}>
  <tr>
   <td>{{=recipe.name}}</td>
  </tr>
 {{pass}}
</table>
```

Similar to Django but full Python in the code (notice “pass” is a Python keyword) without indentation requirements (web2py indents the code for you at runtime). Only one type of escape sequence {{ }} which is transparent to all HTML editors. All string are escaped (unless otherwise specified, like in Django and Kid). It can be used to generate JavaScript (like Django and Rails).

Escaping in Views

In Rails

```
<%%= message>
```

The double %% indicate the text has to be escaped. This is off by default, hence unsafe. Should be the opposite.

Escaping in Views

In Django

```
{% filter safe %}
   {{ message }}
{% endfilter %}
```

Since Django 1.0 all text is escaped by default. You mark it as safe if you do not want it to be escaped.

Escaping in Views

Kid or Genshi in TurboGears 1.0 or Cherrypy

```
<div py:content=”XML(recipe.name)”></div>
```

Text is escaped by default. If text should not be escaped it has to be marked with XML

Escaping in Views

In web2py

```
{{=XML(recipe.name,sanitize=False)}}
```

Text is escaped by default. If text should not be escaped it has to be marked with XML. The optional sanitize option perform XML sanitization by selective escaping some tags and not others. Which tags have to be escaped and which tag attributes can be specified via arguments of XML.

Views Hierarchy

In Rails

```
<title>Layout Example</title>
  <body>
    <%= yield %>
  </body>
</html>

and in controller:
render :layout='filename.html.erb'
```

The rendered page is inserted in the <%= yield %> tag in the layout.

One can include other views with <%= render ... %>

Notice that also :layout follow a naming convention and there is a default.

Views Hierarchy

In Django

```
<title>Layout Example</title>
  </head>
  <body>
    {% block main %} {% endblock %}
  </body>
</html>

and in view:
{%block main%}body{%endblock%}
```

Views can be extended and included using blocks that have names.

Views Hierarchy

Kid or Genshi in TurboGears 1.0 or Cherrypy

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:py="http://purl.org/kid/ns#"
            py:extends="'master.kid'">
...
```

Views Hierarchy

In web2py

```
<title>Layout Example</title>
  <body>
    {{include}}
  </body>
</html>

and in view:
{{extend 'layout.html'}}
body
```

Notation similar to Rails but called like in Kid. The body replaces {{include}} in the layout. layouts can extend other layouts. Views can include other views.

Forms

In Rails

```
<%= form_tag :action => "update" dp %>
 Name: <%= text_field "item", "name" %><br />
 Value: <%= text_field "item", "value" %><br />
 <%= submit_tag %>
<%= end %>
```

Rails has helpers to create forms but that’s it. As far as I know there is no standard mechanism to automatically create forms from models (database tables). Perhaps there are Rails add-on to do this.

There is a mechanism to validate submitted forms.

Forms

In Django

# in model
```
class ArticleForm(ModelForm):
     class Meta:
         model = Article
```
# in controller
```
def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            return HttpResponseRedirect('/thanks/')
    else:
        form = ContactForm() # An unbound form
    return render_to_response('contact.html', {
        'form': form,})
```

In Django, you can create a Form (ArticleForm) from a model (Article).

The Form knows how to serialize itself and validate the input on self-submission, but the errors are not automatically inserted in the form.

Forms

Kid or Genshi in TurboGears 1.0 or Cherrypy

```
?
```

I believe you need to use a library like ToscaWidgets. Sorry, I am not familiar with them. If you know how to fill this page please let me know.

Forms

In web2py

```
def contact(request):
    form = SQLFORM(Article)
    if form.accepts(request.vars):
        redirect('thanks')
    return dict(form=form)
```

This is the same as the previous Django form (generated from the Article) model, except that when the form is serialized, if there are errors, they are displayed in the form itself (unless specified otherwise).

Web2py forms can also prevent double submission.

Validation

In Rails

```
ActiveForm::Definition::create :article do |f|
 f.section :details do |s|
 s.text_element :email,:class => 'required' do |e|
   e.validates_as_email :msg => 'not email'
 end
end
```

Rails defines validators like requires, email, etc. You can associate validators to a form. In your controllers you need to check if a form is valid, and, on error, alter the page to include the errors generated by validation.

Validation

In Django

```
from django.core.validators import *

class Article(models.Model):
    email = models.EmailField(
      validator_list=[isValidEmail])
```

Very much like Rails but more validators. Validators are specified in models and/or forms.

Validation

In web2py

```
db.define_table('Article',SQLField('email'))

db.Article.email.requires=IS_EMAIL()
```

Similar to Django and Rails because validators are attached to table fields and form fields but validators are classes not objects. This means they must be instantiated with (). You can pass arguments to the validators to change their behavior (for example the error message).

The presence of validators affects the way a field is rendered in a form. For example IS_IN_SET() renders the field with a dropbox.

Models and Migrations

In Rails

```
class Article < ActiveRecord::Migration
  def self.up
     create_table :articles do |t|
       t.column :name, :string
       t.column :description, :text
     end
  end
end
```

In Rails there is a place to define tables that need to be created/deleted/altered (migrations) and a different place to establish relations between tables (one to many, many to many, etc)

Models and Migrations

In Django

```
class Article(models.Model):
    name = models.StringField()
    description = models.TextField()
```

In Django there is one place where models are defined. If the tables do not exist they are created. Django does not do migrations (i.e. it does not alter or drop tables if the models change). For many to many relations, it creates the intermediate link table for you.

Models and Migrations

In SQLAlchemy (used in TG and Pylons)

```
from turbogears.database import metadata, mapper
  sqlalchemy
import Table, Column, Integer

mytable = Table('mytable', metadata,
    Column('id', Integer, primary_key=True))

class MyTable(object): pass

mapper(MyTable, mytable)
```

SQLAlchemy makes a distinction between what tables are in the database and how they are mapped into Python objects. This is because SQLAchemy can deal with legacy databases. Rails, Django and web2py can but with limitations (in the case of web2py for example, tables must have an integer auto-increment key field called “id”).

Models and Migrations

In web2py

```
Article=db.define_table('Article',
                 SQLField('email','string'),
                 SQLField('description','text')
```

The syntax is a little different but the functionality is similar to Rails. If the the table in the model above does not exist it is created. If the model changes, web2py alters the table accordingly. One to many ad many to many relations are implied by reference fields.

Select Query

In Rails

```
Article.find(:first,:conditions => [
   "id > :id AND name = :name",
        {:id => 3,      :name => "test" }])
```

This is the most common notation for a select. The conditions argument is basically a SQL statement but the parameters are passed as additional arguments.

Select Query

In Django

```
Article.objects.filter(id__gt=3,name='test')
```

“id__gt=3” reads “id greater than 3”.

Django queries are lazy-evaluated.

Select Query

In SQLAlchemy (used in TG and Pylons)

```
query(Article).filter_by(id>3, name='test')
```

Select Query

In web2py

```
db(Article.id>3 and Article.name=='test').select()
```

In web2py you always need to specify which db you are acting on because you can have multiple db connections in the same code.

Transcations

In Rails

```
class MyTestController < ApplicationController
   def index
      Record.transaction do
         ...
      end
   end
end
```

Transactions

In Django

```
from django.http import HttpResponse
from django.db import transaction

@transaction.commit_on_success
def index(request):
    ...
    return HttpResponse("Hello World”)
```

There are multiple decorators: autocommit (commits always), commit_on_success (commit only if not Exception), commit_manually (requires calling) transaction.commit() or transaction.rollback()

Transactions

In TurboGears

```
import turbogears
from turbogears import controllers, expose

class MyRoot(controllers.RootController):

@expose(template="MyApp.MyRoot.index")
def index(self):
     ...
     return dict(message=”Hello World”)
```

By default all actions are enclosed in a transaction that commits on success and rollsback on exception.

Transactions

In web2py

```
def index()
    ...
    return dict(message=”Hello World”)
```

By default all actions are enclosed in a transaction that commits on success and rollsback on exception.

Internationalization

In Rails

Rails currently doesn’t offer any explicit support for internationalization. Perhaps it should, perhaps it’s too app specific to generalize.

http://wiki.rubyonrails.org/rails/pages/Internationalization

Thinks will change in Rails 2.2 but here we talk about present, not future.

Internationalization

In Django

```
from django.http import HttpResponse
from django.utils.translation import ugettext as _

def my_view(request):
    message = _("Hello World")
    return HttpResponse(message)
```

Using ‘_’ is the common convention.

Requires a few shell commands to build and edit the dictionaries.

Internationalization

In TurboGears

```
import turbogears
from turbogears import controllers, expose

class MyRoot(controllers.RootController):

@expose(template="MyApp.MyRoot.index")
def index(self):
     message=_(“Hello World”)
     return dict(message=message)
```

As in the case of Django, this requires some shell commands to build and edit the dictionaries.

web2py

vs

Other Web Frameworks

Other Web Frameworks?

- j2ee
- PHP
- CakePHP
- Django
- Pylons
- Ruby on Rails (RoR)

Who's not in the list?

- TurboGears (because TG2 is not that different from Pylons+SQLAlchemy +Genshi)
- web.py, werkzeug, karrigell, psp, etc. (all excellent frameworks with their functionalities are too limited for a fair comparison)

Who’s not in the list?

- Cherrypy (the cherrypy wsgiserver is included in web2py)
- j2ee (there are too many to choose)
- Zope (sorry, I do not understand Zope)

Underlying Language

web2py		python
j2ee		java
PHP		syntax draws upon C, Java, and Perl
CakePHP		php
Django		python
Pylons		python
RoR		ruby

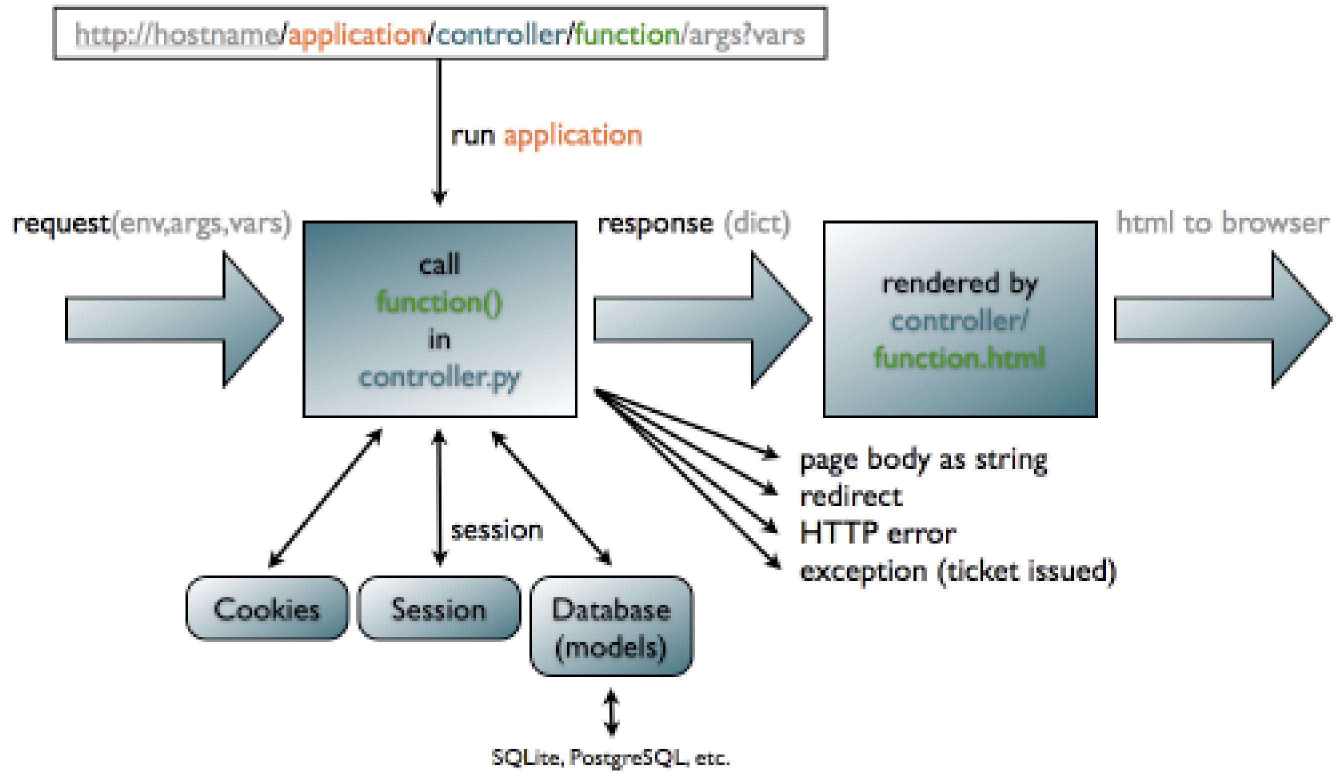
Model View Controller

web2py	yes	
j2ee	yes	
PHP	no	
CakePHP	yes	
Django	yes	
Pylons	yes	
RoR	yes	

Model View Controller

- in web2py, given a model, the default controller appadmin.py provides a database administrative interface (each app has its own)
- in web2py, every controller function, by default, has a generic view

Model View Controller



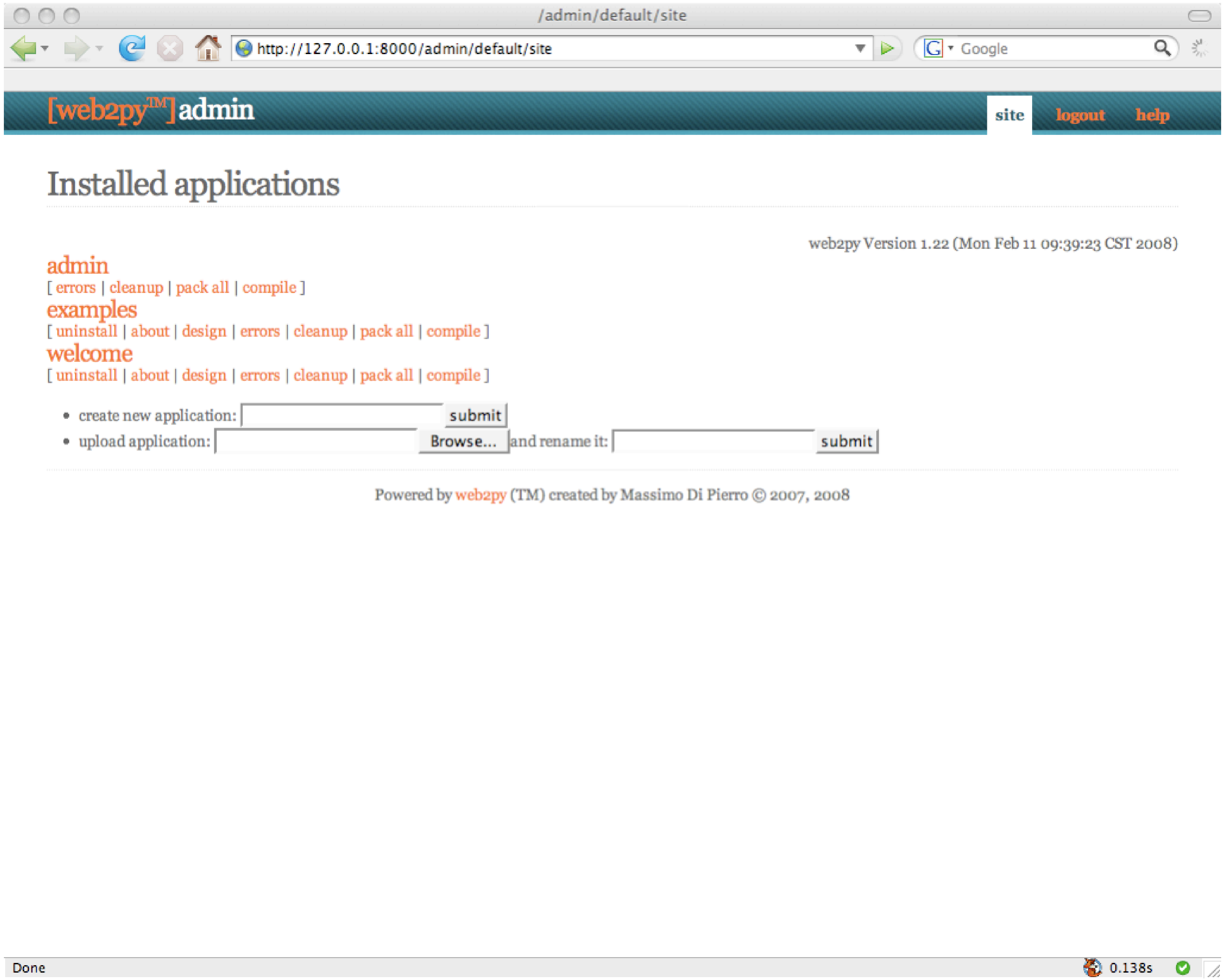
Web Based Interface

web2py	yes	
j2ee	no	
PHP	no	
CakePHP	no	
Django	no	Django has a database administrative interface only not not an app development/management administrative interface like web2py.
Pylons	no	
RoR	no	only at Heroku.com which, anyway, is very limited compared to the web2py one.

Web Based Interface

- The web2py web based administrative interface allows you to do development, debugging, testing, deployment, maintenance, and database administration.
- The use of the web based interface is “optional” and not required. The same functionality can be accessed via the Python shell.

Web Based Interface



/admin/default/site

http://127.0.0.1:8000/admin/default/site

[web2py™] admin

site logout help

Installed applications

web2py Version 1.22 (Mon Feb 11 09:39:23 CST 2008)

admin
[errors | cleanup | pack all | compile]

examples
[uninstall | about | design | errors | cleanup | pack all | compile]

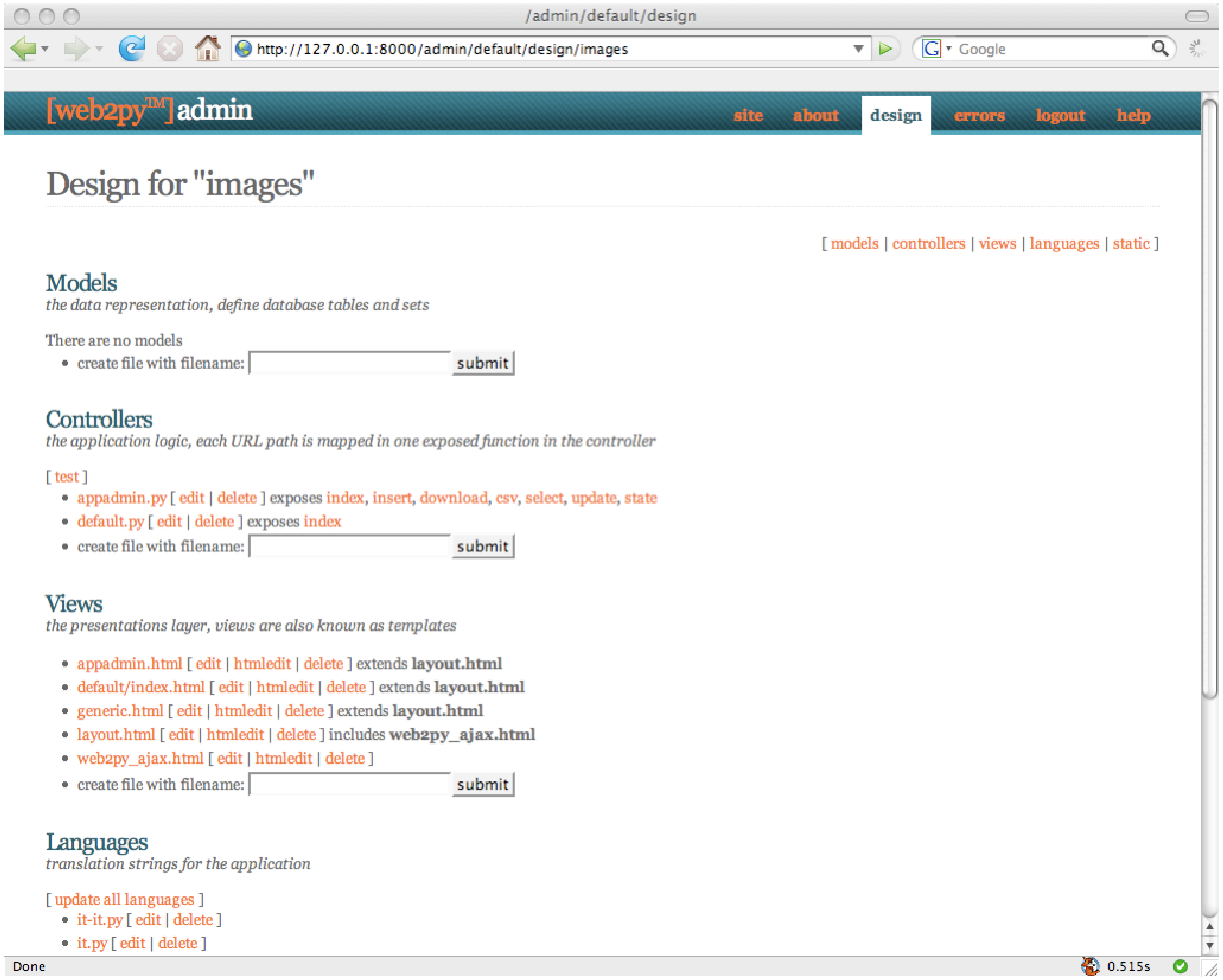
welcome
[uninstall | about | design | errors | cleanup | pack all | compile]

- create new application: submit
- upload application: Browse... and rename it: submit

Powered by web2py (TM) created by Massimo Di Pierro © 2007, 2008

Done 0.138s

Web Based Interface



/admin/default/design

http://127.0.0.1:8000/admin/default/design/images

[web2py™] admin

site about design errors logout help

Design for "images"

[models | controllers | views | languages | static]

Models
the data representation, define database tables and sets

There are no models
- create file with filename: submit

Controllers
the application logic, each URL path is mapped in one exposed function in the controller

[test]
- appadmin.py [edit | delete] exposes index, insert, download, csv, select, update, state
- default.py [edit | delete] exposes index
- create file with filename: submit

Views
the presentations layer, views are also known as templates

- appadmin.html [edit | htmledit | delete] extends **layout.html**
- default/index.html [edit | htmledit | delete] extends **layout.html**
- generic.html [edit | htmledit | delete] extends **layout.html**
- layout.html [edit | htmledit | delete] includes **web2py_ajax.html**
- web2py_ajax.html [edit | htmledit | delete]
- create file with filename: submit

Languages
translation strings for the application

[update all languages]
- it-it.py [edit | delete]
- it.py [edit | delete]

Done 0.515s

Web Based Database Administrative Interface

web2py	yes	one for every app
j2ee	no	via third party application
PHP	no	via third party application
CakePHP	no	via third party application
Django	yes	
Pylons	no	via third party application
RoR	no	via third party application

Generic CRUD helpers/controllers

web2py	yes	
j2ee	no	
PHP	no	
CakePHP	yes	
Django	yes	
Pylons	no	
RoR	yes	

upload forms

- Only web2py and Django have a standard mechanism to handle file upload and secure storage.
- In case of the web2py the uploaded file is securely renamed, stored on disk and the name is store in the database. Upload is always done via streaming in order to handle large files.

Byte Code Compilation

web2py	yes	there is a button [compile all] it compiles models, controllers and views
j2ee	yes	
PHP	no	
CakePHP	no	
Django	yes	it is always possible to bytecode compile python code, usually not the views/templates, but this is not as trivial as clicking on one button
Pylons	yes	
RoR	no	

Byte Code Compilation

- web2py and j2ee are the only frameworks that allow to byte code compile applications and distribute them in closed source.

Ticketing System

web2py	yes	
j2ee	no	
PHP	no	
CakePHP	no	
Django	no	can be configured to send you an email in case of error
Pylons	no	can be configured to send you an email in case of error
RoR	no	

Ticketing System

- web2py has not distinction between debugging and production modes. All uncaught exceptions are logged and a ticket is issued to the visitor in order to recover the associated log.
- Administrator can browse and read logs via the administrative interface

Zero Installation

web2py	yes	
j2ee	no	
PHP	no	
CakePHP	no	
Django	no	
Pylons	no	
RoR	no	

Zero Installation

- The binary distributions of web2py (for Windows and Mac) package the interpreter, the SQLite database and the administrative interface.
- They require no installation and can run off a USB drive.

Zero Configuration

web2py	yes	
j2ee	no	
PHP	no	
CakePHP	no	
Django	no	
Pylons	no	
RoR	no	

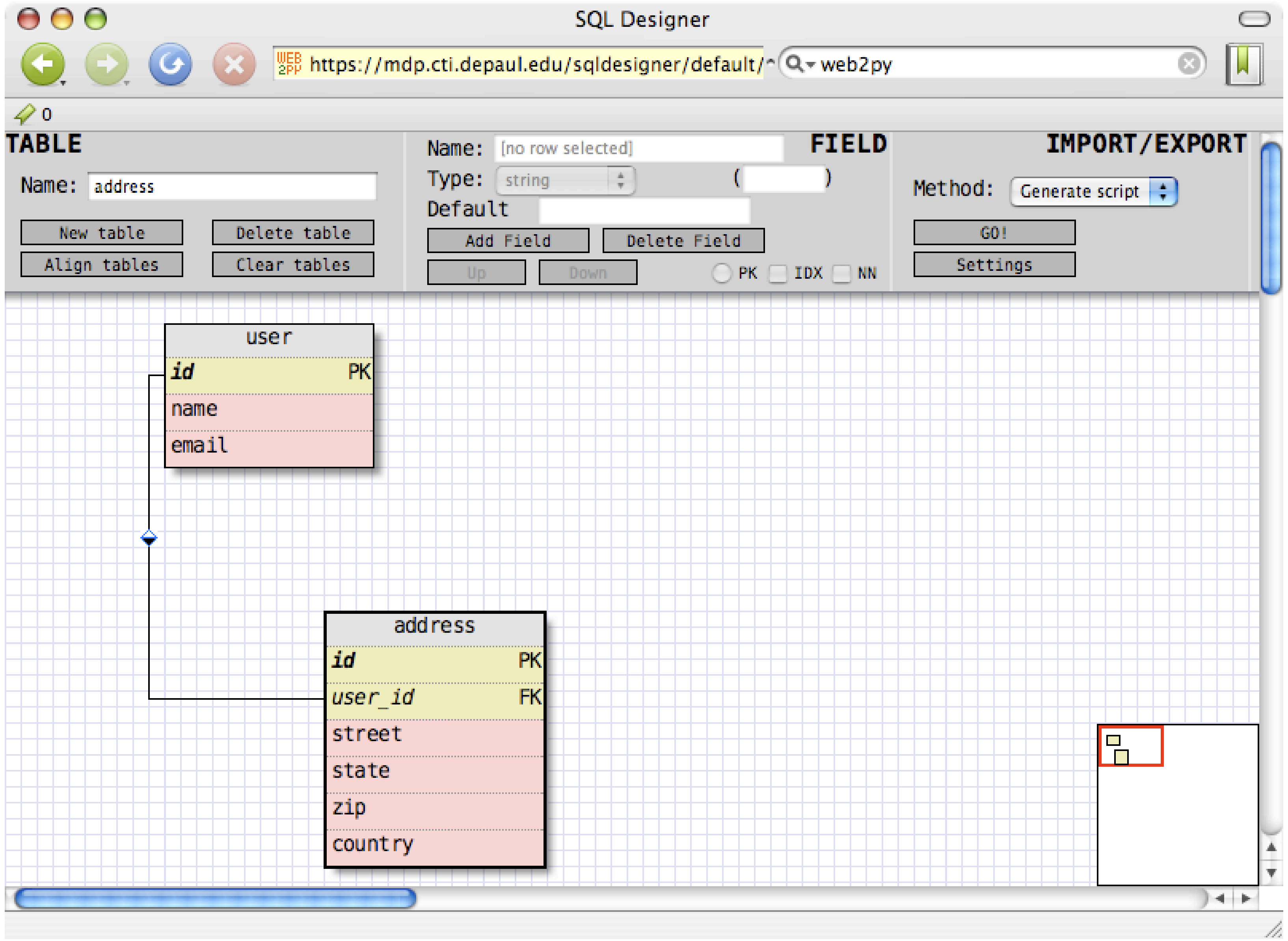
Zero Configuration

- web2py has no configuration file at the framework level. This ensures an easy setup and portability of applications. All other frameworks require some type of configuration.
- web2py applications can have configuration files.

Web Based Model Designer

web2py	yes	on web page
j2ee	no	
PHP	no	
CakePHP	no	
Django	no	
Pylons	yes	via CatWalk (SQLObjects only?)
RoR	no	

Web Based Model Designer



user

id	PK
name	
email	

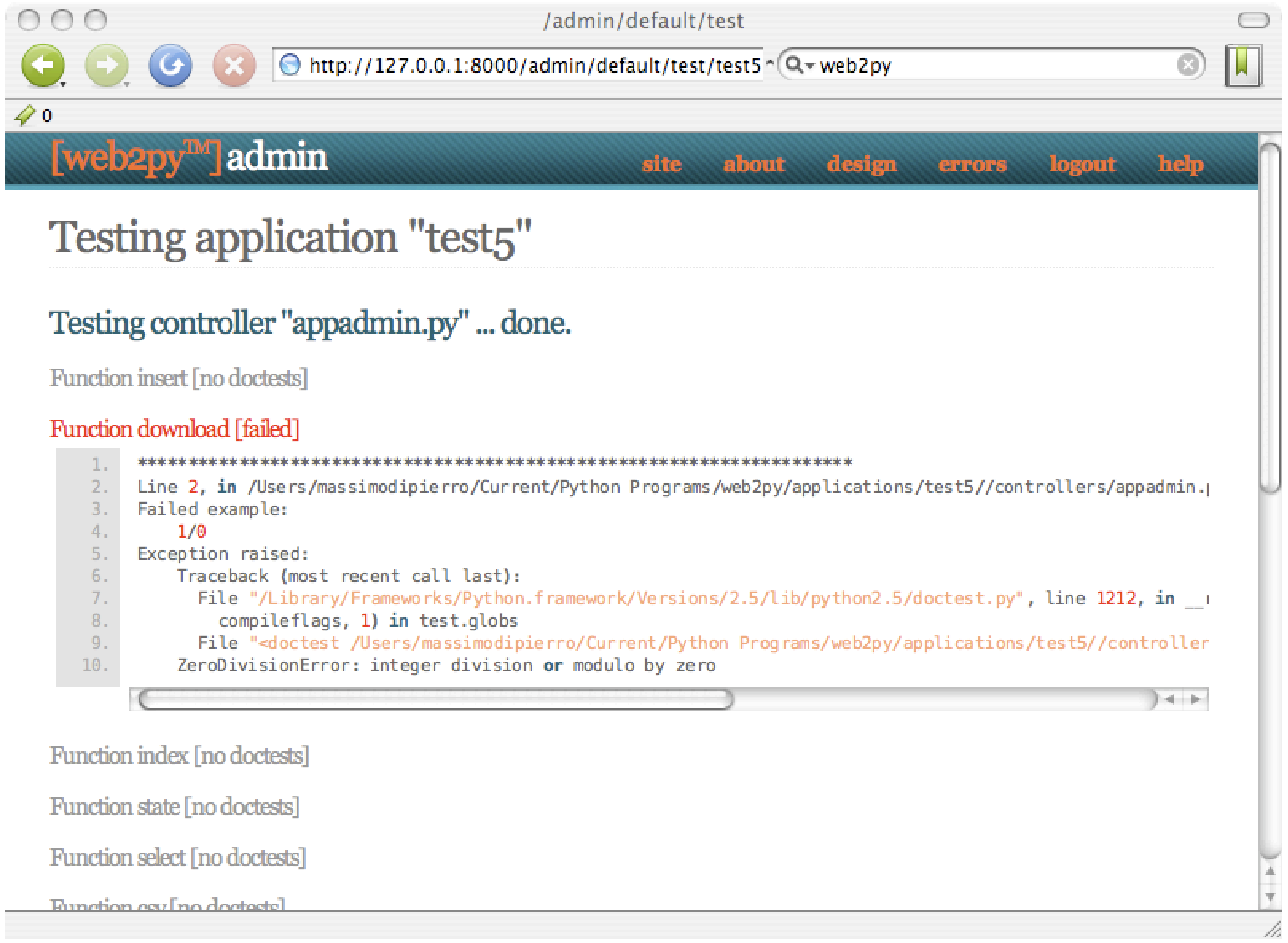
address

id	PK
user_id	FK
street	
state	
zip	
country	

Web Based Testing

web2py	yes	as web interface to DocTests
j2ee	no	
PHP	no	
CakePHP	no	
Django	no	
Pylons	no	
RoR	no	

Web Based Testing



Testing application "test5"

Testing controller "appadmin.py" ... done.

Function insert [no doctests]

Function download [failed]

```
**********************************************************************
Line 2, in /Users/massimodipierro/Current/Python Programs/web2py/applications/test5//controllers/appadmin.
Failed example:
    1/0
Exception raised:
    Traceback (most recent call last):
      File "/Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/doctest.py", line 1212, in __
        compileflags, 1) in test.globs
      File "<doctest /Users/massimodipierro/Current/Python Programs/web2py/applications/test5//controller
    ZeroDivisionError: integer division or modulo by zero
```

Function index [no doctests]

Function state [no doctests]

Function select [no doctests]

Function csv [no doctests]

Runs on Google App Engine

web2py	yes	with some limitations
j2ee	no	
PHP	no	
CakePHP	no	
Django	yes	but not the ORM
Pylons	yes	but not all its components
RoR	no	

Runs on Google App Engine

- web2py is the only framework that allows to develop on your own platform and then run the app, unmodified on the Google App Engine (with the limitations imposed by the App Engine).
- No need to rewrite the model since the web2py database abstraction layer supports the Google Query Language.

Caching

web2py	yes	for any function, you can specify whether to cache in ram, on disk, with memcache, or combinations.
j2ee	yes	with third party components
PHP	yes	memcache
CakePHP	yes	memcache
Django	yes	ram, disk, db, memcache
Pylons	yes	ram, disk, db, memcache
RoR	yes	memcache

Native Template Language

web2py	yes	100% Python with no indentation need
j2ee	yes	most common are XML or JSP
PHP	yes	PHP is itself a template language
CakePHP	yes	PHP
Django	yes	Django Template Language
Pylons	yes	Kid, Genshi, Mako, Cheetah, etc.
RoR	yes	Ruby

Native Template Language

- Any Python Framework can use any Python-based Template Language (for example web2py can use Genshi, Pylons can use web2py's).
- The native web2py template language consists of pure code embedded in {{ }} tags inside HTML. Blocks end with "pass", so no indentation requirements.

Template Language

- web2py View Example:

```
<html><body>

{{for i in range(10):}}

<b>Hello number {{=i}}</b><br/>

{{pass}}

</body></html>
```

Template Extension

web2py	yes	
j2ee	yes	
PHP	yes	
CakePHP	yes	
Django	yes	
Pylons	yes	
RoR	yes	

Template Extension

- web2py Example:

```
{{extend ‘layout.html’}}

<h1>Hello world</h1>

{{include ‘sidebar.html’}}
```

HTML Helpers

web2py	yes	
j2ee	no	
PHP	no	
CakePHP	yes	
Django	yes	
Pylons	yes	
RoR	yes	

Internationalization

web2py	yes	
j2ee	no	
PHP	no	
CakePHP	yes	
Django	yes	
Pylons	yes	
RoR	no	

Internationalization

- In web2py, text is marked for translation using T("hello world").
- Translations are edited via the provided administrative interface.
- It is possible to have variables in translations like in

```
T(“Hello %(name)s”,dict(name=”Massimo”))
```

Database Abstraction

web2py	yes	
j2ee	no	limited to JavaBeans
PHP	no	PearDB does not count because because it requires the developer to write SQL queries and has no Object Relational Mapper
CakePHP	no	
Django	yes	
Pylons	yes	via SQLAlchemy or SQLObjects
RoR	yes	via ActiveRecords

Database Abstraction

- The web2py ORM works seamlessly with SQLite, MySQL, PostgreSQL, Oracle and on the Google App Engine (with the limitations imposed by the Google system)

Database Abstraction

- web2py example

```
rows=db(db.user.birthday.year()>1950).select()
```

- equivalent Django example

```
rows=User.objects.filter(birthday__year__gt=1950)
```

Left Outer Joins

web2py	yes	
j2ee		n/a because no ORM
PHP		n/a because no ORM
CakePHP		n/a because no ORM
Django	yes	requires a custom Q object
Pylons	yes	with SQLAlchemy, no with SQLObjects
RoR	yes	

Left Outer Joins

- All the Python ORMs have the ability to execute raw SQL therefore they allow allow LEFT OUTER JOIN although not in a SQL-dialect independent way

Automatic Migrations

web2py	yes	
j2ee	no	
PHP	no	
CakePHP	no	
Django	no	
Pylons	yes	
RoR	yes	

Automatic Migrations

- In web2py if one changes the data model, it automatically and transparently generates and executes SQL to ALTER TABLEs. There is no special command to type like in Rails.

Multiple Databases

web2py	yes	
j2ee	yes	
PHP	yes	
CakePHP	yes	
Django	no	but there is a branch that allows it
Pylons	yes	
RoR	?	

Multiple Databases

- In web2py table objects are attributes of a database connection therefore there is no conflict if one establishes multiple connections.
- In other framework tables are represented by classes and there may be conflicts if two databases have tables with same name.

Distributed Transactions

web2py	yes	with PostgreSQL only
j2ee	yes	
PHP	no	
CakePHP	no	
Django	no	
Pylons	no	
RoR	no	

CRUD methods

web2py	yes	
j2ee	no	via third party plugin
PHP	no	via third party plugin
CakePHP	yes	
Django	yes	
Pylons	no	
RoR	yes	

Blocks SQL Injections

web2py	yes	
j2ee	no	up to the programmer to write secure code
PHP	no	
CakePHP	no	
Django	yes	
Pylons	yes	
RoR	yes	

Blocks Double Submit

web2py	yes	
j2ee	no	
PHP	no	
CakePHP	no	
Django	no	
Pylons	no	
RoR	no	

Blocks Double Submit

- web2py provides methods to generate forms form database tables and automatically validates and processes forms on submission. It also injects a one-time token in each form to prevent double form submission and some reply attacks.

xmlrpc services

web2py	yes	
j2ee	yes	
PHP	no	
CakePHP	no	
Django	no	
Pylons	yes	
RoR	no	

Included Ajax Library

web2py	yes	jQuery
j2ee	no	
PHP	no	
CakePHP	yes	jQuery
Django	no	
Pylons	no	
RoR	yes	Scriptaculous

Included Ajax Library

- Any of the frameworks can use any third party Ajax libraries, here we are concerned with server-side programming only.

JSON support

web2py	yes	simplejson is included
j2ee	yes	
PHP	yes	
CakePHP	yes	
Django	yes	simplejson is included
Pylons	yes	simplejson is included
RoR	yes	

File Streaming

web2py	yes	by default for all static large files
j2ee	yes	not by default
PHP	yes	
CakePHP	yes	
Django	yes	
Pylons	yes	
RoR	yes	

IF_MODIFIED_SINCE

web2py	yes	by default
j2ee	no	not out of the box rely on web server for static content, requires programming otherwise
PHP	no	
CakePHP	no	
Django	no	
Pylons	yes	
RoR	no	

206 PARTIAL CONTENT

web2py	yes	by default
j2ee	no	not out of the box rely on web server for static content, requires programming otherwise
PHP	no	
CakePHP	no	
Django	no	
Pylons	yes	
RoR	no	

Handlers for Web Servers

- web2py is wsgi compliant.
- comes with the cherrypy wsgi fast and ssl-enbled web server.
- runs with apache and mod_proxy or mod_rewrite or mod_wsgi.
- runs with lightpd with FastCGI.
- runs as CGI script.

Routes

web2py	yes	including reversed, with or without regex allows IP filtering
j2ee	no	delegated to web server
PHP	no	delegated to web server
CakePHP	yes	no reversed, no IP filter
Django	yes	uses regex, no reversed routes, no IP filter
Pylons	yes	similar to rails
RoR	yes	reversed routes? no IP filter

Routes

- In web2py routes are "optional" and there is a default mapping between URLs and controllers (similar to Rails).
- IP filter is a web2py feature that allows to map URLs into controllers in a way that depends on the visitor IP pattern. In this way different visitors see different pages but the same URL.

Documentation

web2py		120 pages draft manual online, one book
j2ee		too many published books
PHP		many published books
CakePHP		online examples only
Django		three books
Pylons		online examples only
RoR		many published books

Documentation

- Most of the frameworks, including web2py, have extensive online documentation
- Web2py also has a repository of free plug-in applications including wikis, blogs, a chat line, an online store, log analyzer, and more.

Links to web2py

- http://mdp.cti.depaul.edu
- FAQ: http://mdp.cti.depaul.edu/AlterEgo
- Free Apps: http://mdp.cti.depaul.edu/appliances