Django and Pyjamas

Django and Pyjamas mariage as an alternative way to create Web Applications

Pawel Prokop

pawel.prokop@adfinem.net

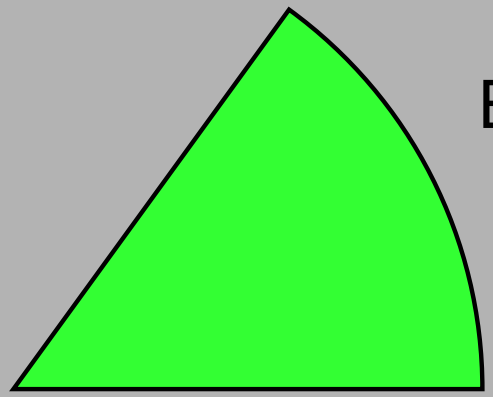
November 24, 2011

Introduction

I'll tell a story about some mariage... ...Django ...Pyjamas

Part I: Introduction

but first...



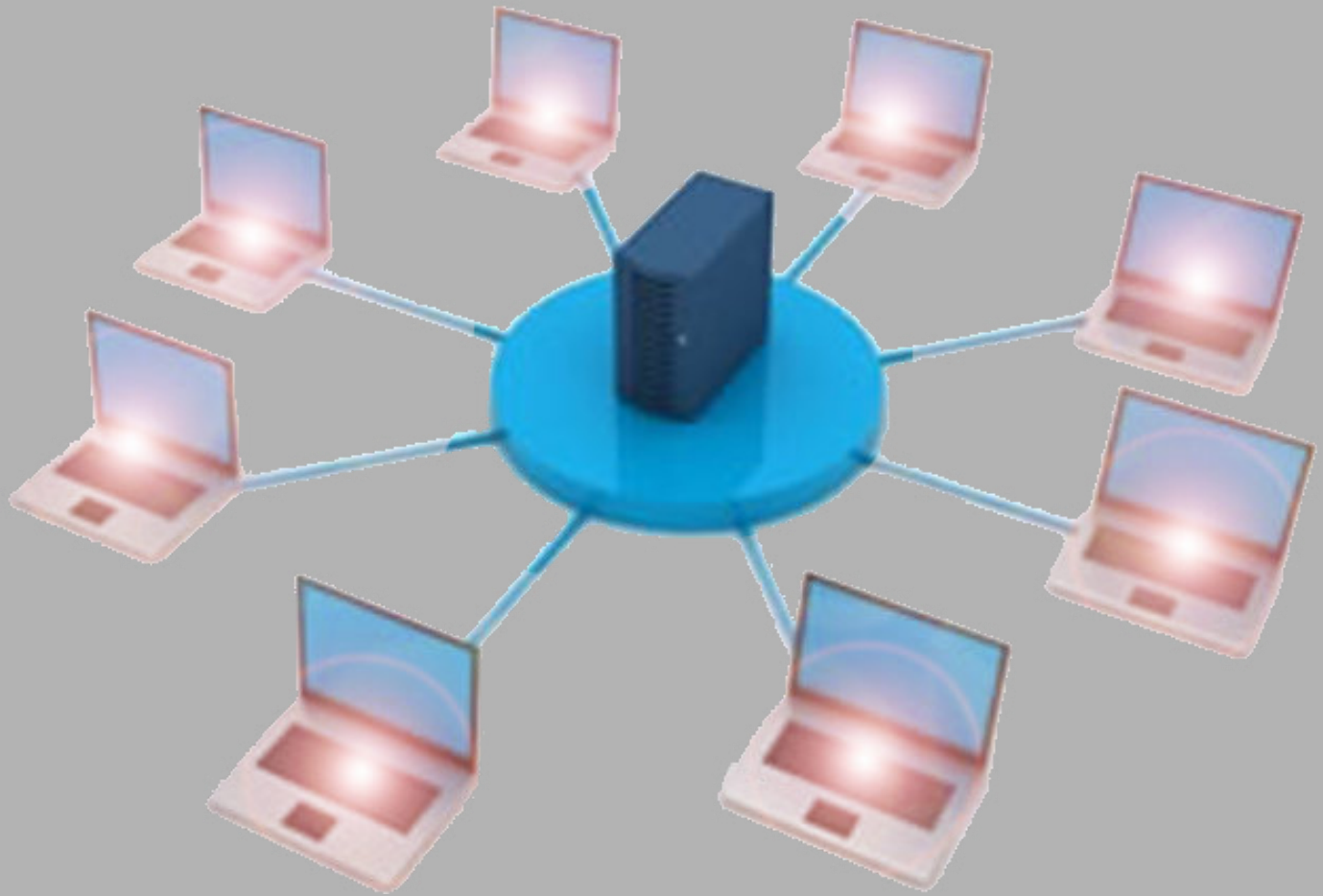
Ecosystem

Web application architecture

Web application architecture

Hosted on server...

Web application architecture



...and accessed from clients

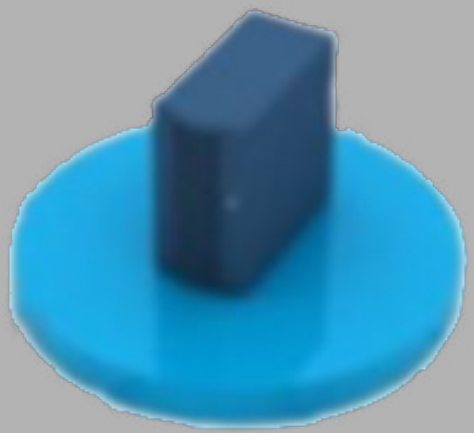
Web application architecture

...from many browsers

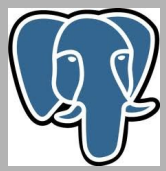
Web application architecture

...and many operating systems

Hosting applications (servers & frameworks)



Database systems

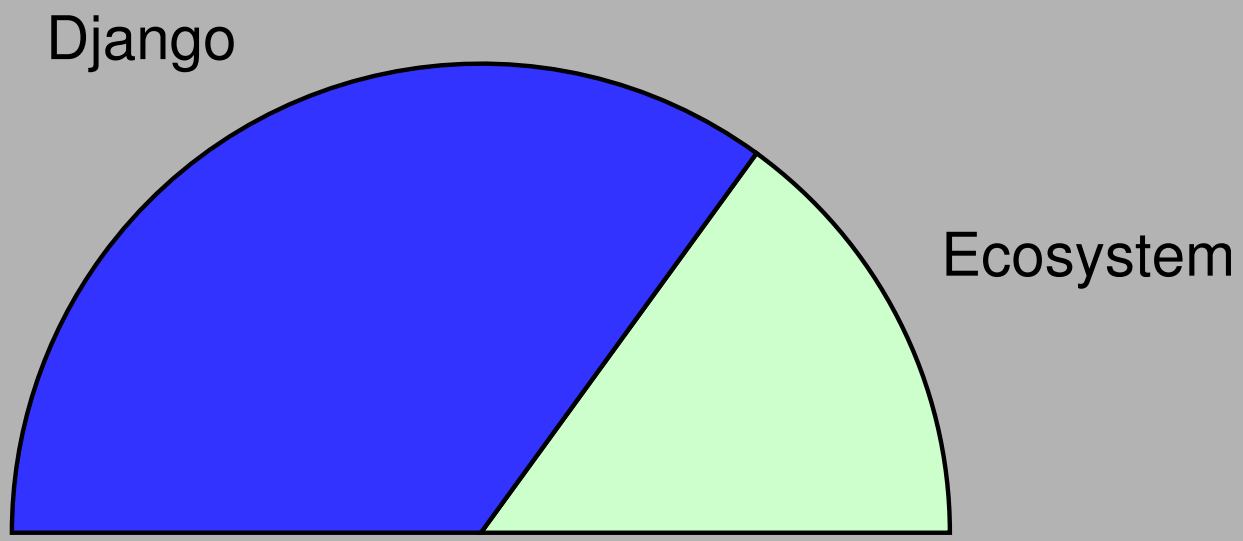


Upgrading...

...the server

...client not involved

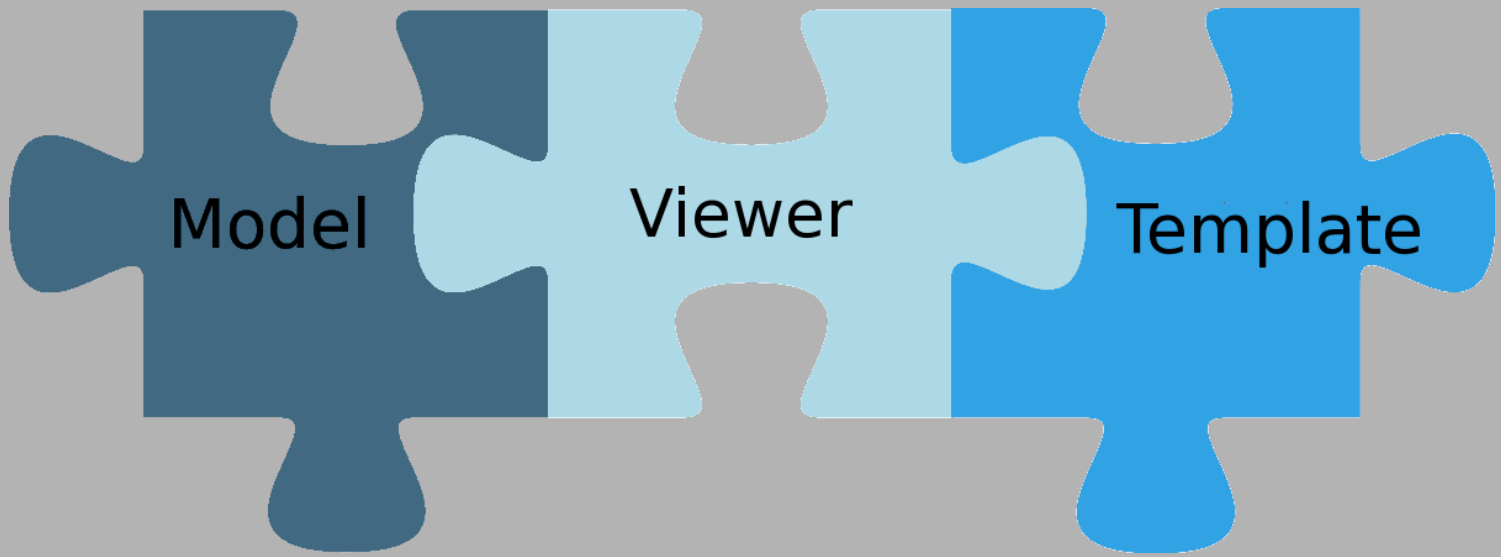
Part II: Django



Who the hell is...

- framework
- written in python
- talks:
  - python
  - HTTP
  - JSONRPC

Philosophy



- **Model** - database objects
- **Template** - html rendering mechanism
- **Viewer** - bussiness rules

Django installation

- Get latest official - stable version

```
1 wget http://www.djangoproject.com/download/1.3.1/tarball/
2 tar xzvf Django-1.3.1.tar.gz
3 cd Django-1.3.1
4 sudo python setup.py install
```

- Get latest from repository

```
1 svn co http://code.djangoproject.com/svn/django/trunk/ django_trunk
```

Verification

- Verify installation (from python console)

```
1 import django
2 django.get_version()
```

Project creation

- Create project filesystem

```
1 django-admin.py startproject myportal
```

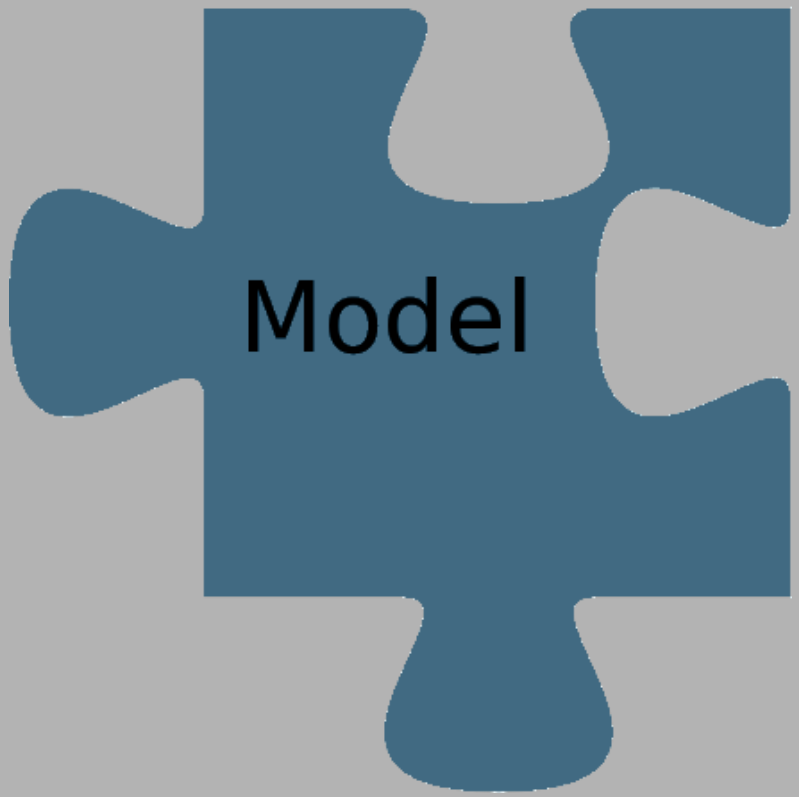
- Files created:
  - `./manage.py`
  - `./myportal`
  - `./myportal/urls.py`
  - `./myportal/settings.py`

Create application

- Create application structure

```
1 python manage.py startapp myapp
```

- Files created:
  - ./models.py
  - ./views.py
  - ./tests.py


Model

Database

- create database manualy
- let django to generate database from django model

Django model

- models.py

```
1  from django.db import models
2
3  class Users (models.Model):
4      login = models.CharField(max_length=32)
5      name = models.CharField(max_length=256)
6      activity = models.BooleanField(default=False)
7      expire = models.DateField(null=True, blank=True)
8      passwd = models.CharField(max_length=256)
```

- Built-in field classes:

  `AutoField, BigIntegerField, BooleanField, CharField, DateField, DateTimeField, DecimalField, EmailField, FileField, FloatField, ImageField, IntegerField, IPAddressField, SmallIntegerField, TextField, TimeField, URLField, XMLField`

  ...and more...

Field arguments

- `null` - field can be null
- `blank` - field can be blank
- `choices` - choices will be displayed for field on admin panel
- `db_column` - customize column name
- `db_index` - create index for this field
- `db_tablespace` - defines name of tablespace
- `default` - default value for this field
- `editable` - specifies if field is editable via admin panel
- `error_messages` - allows to override default error message

...field arguments, cont.

- `help_text` - specifies text that will be displayed next to field on admin panel
- `primary_key` - specifies if the field is primary key
- `unique` - specifies if the field is unique for the table
- `unique_for_date` - specifies DateField within this field should be unique
- `unique_for_month` - specifies month field within this field should be unique
- `unique_for_year` - specifies year field within this field should be unique
- `verbose_name` - defines verbose field name
- `validators` - allows to append custom validation method (this method should raise ValidationError)

Populate model to database

- Populate model to database

```
1 python manage.py syncdb
```

Customize model

- validate model

```
1 python manage.py validate
```

- display customized sql

```
1 python manage.py sqlcustom myapp
```

- display all drops

```
1 python manage.py sqlclear myapp
```

- display indexes

```
1 python manage.py sqlindexes myapp
```

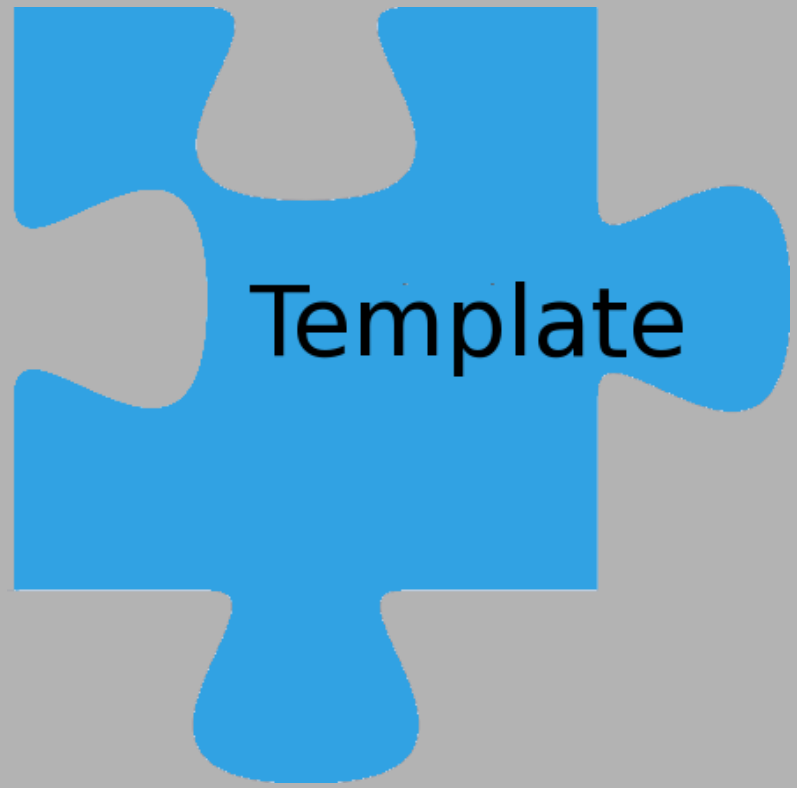
- display all

```
1 python manage.py sqlall myapp
```

Adding custom sql

- Create `./myapp/sql/users.sql` file that contains:

```
1 insert into myapp_users (login, name, activity, passwd)
2     values ('pablo', 'Pawel Prokop', 't',
3         'ec222c5f42458e0b3e7505a689e223ba624aeb84');
```


Template

Forms

- In application directory create file:
  `credential_forms.py`

```
1  from django import forms
2
3  class LoginForm(forms.Form):
4
5      login = forms.CharField()
6      password = forms.CharField(widget=forms.PasswordInput)
```

- Built-in field classes: `BooleanField`, `CharField`, `ChoiceField`, `DateField`, `DecimalField`, `EmailField`, `FileField`, `IPAddressField`, `FloatField`, `IntegerField`, `MultipleChoiceField`, `ComboField`, and more...

Field arguments

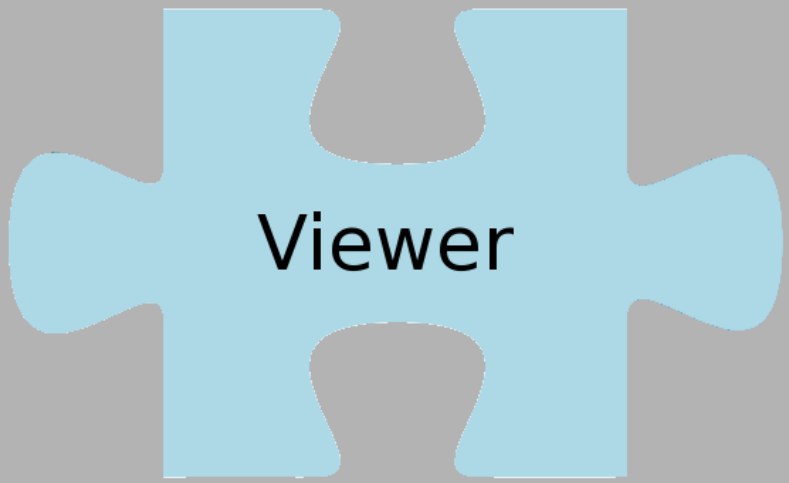
- `required` - field is required and validation will fail
- `label` - defines display for the field
- `initial` - defines initial value
- `widget` - specifies a class to render this field
- `help_text` - text that will be displayed next to the field
- `error_messages` - allows to override default error messages
- `validators` - allows to append custom validation method (this method should raise ValidationError)
- `localize` - enables localization of form rendering

Templates

Create directory `templates` and put there all template files
Let `credential_template.html` be like follow:

```
1  <html>
2      <form action="/myportal/login" method="post">
3
4  {% csrf_token %}
5  {{ loginForm }}
6
7      <input type="submit" value="Login"/>
8      </form>
9  </html>
```

Update `TEMPLATE_DIRS` in `settings.py`


Viewer

Spin Model and Template with Viewer

Create file `views.py` as follows:

```
1  from django.template import RequestContext, Context, loader
2  from django.http import HttpResponse
3  from django.core.context_processors import csrf
4  from django.shortcuts import render_to_response
5
6  import credential_forms
7
8
9  def index(request):
10
11     loginForm = credential_forms.LoginForm()
12
13     params = {'loginForm': loginForm}
14
15     return render_to_response('credential_template.html',
16         params, context_instance=RequestContext(request))
```

Append bussiness rule for login feature

To `views.py` append implementation of login logic:

```
1  def login(request):
2
3      params = {}
4
5      if request.method == 'POST':
6
7          loginForm = credential_forms.LoginForm(request.POST)
8
9          username = 'Unknown'
10
11         import models
12         user = models.Users.objects \
13             .filter(login=loginForm['login'].value()) \
14             .filter(passwd=encrypt(loginForm['passwd'].value()))
15
16         if len(user) == 1:
17             username = user[0].name
18
19         params = {'name' : username}
20
21     return render_to_response('user_info.html', params,
22         context_instance=RequestContext(request))
```

...and one convenience method

```
1  def encrypt(string):
2      import hashlib
3      s = hashlib.sha1()
4      s.update(str(string))
5      return s.hexdigest()
```

...and one more template

Create template file: `user_info.html` in `templates` directory

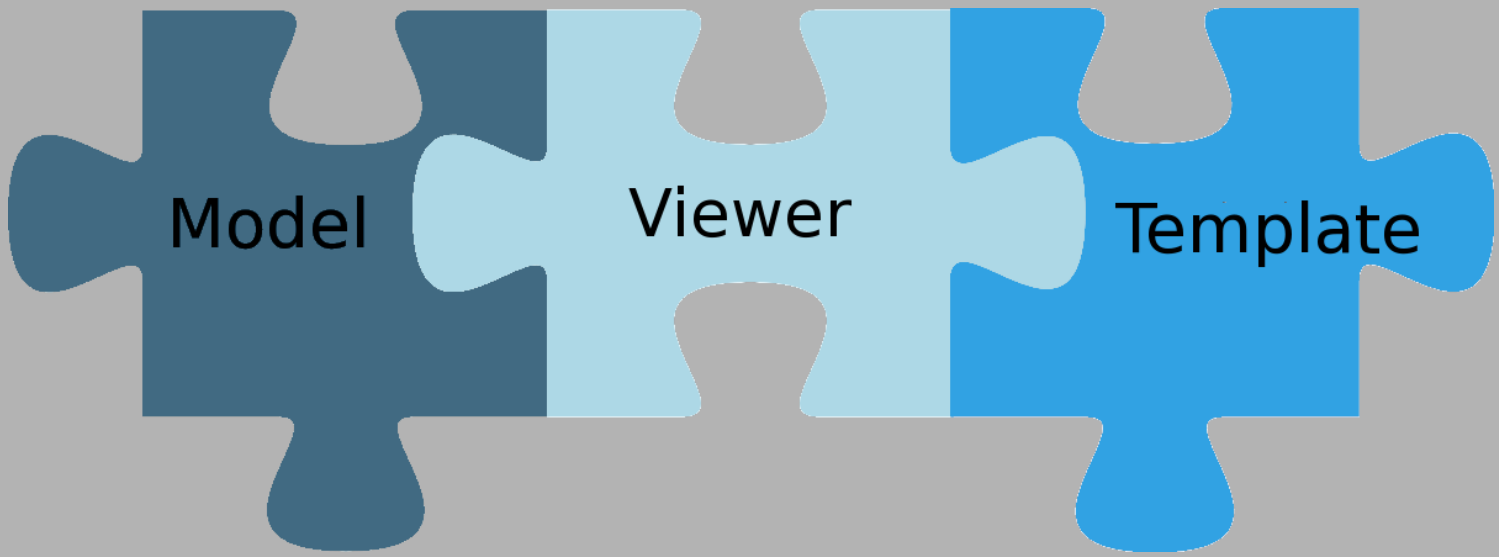
```
1  <html>
2  {% csrf_token %}
3
4  <p>Hello: {{ name }}
5
6  <p>
7  <a href="/myportal">Go Back</a>
8
9  </html>
```

...and create interface

Add the following to `myportal/urls.py`

```
1     (r'^myportal/$', 'myapp.views.index'),
2     (r'^myportal/login$', 'myapp.views.login'),
```

Model-Template-Viewer



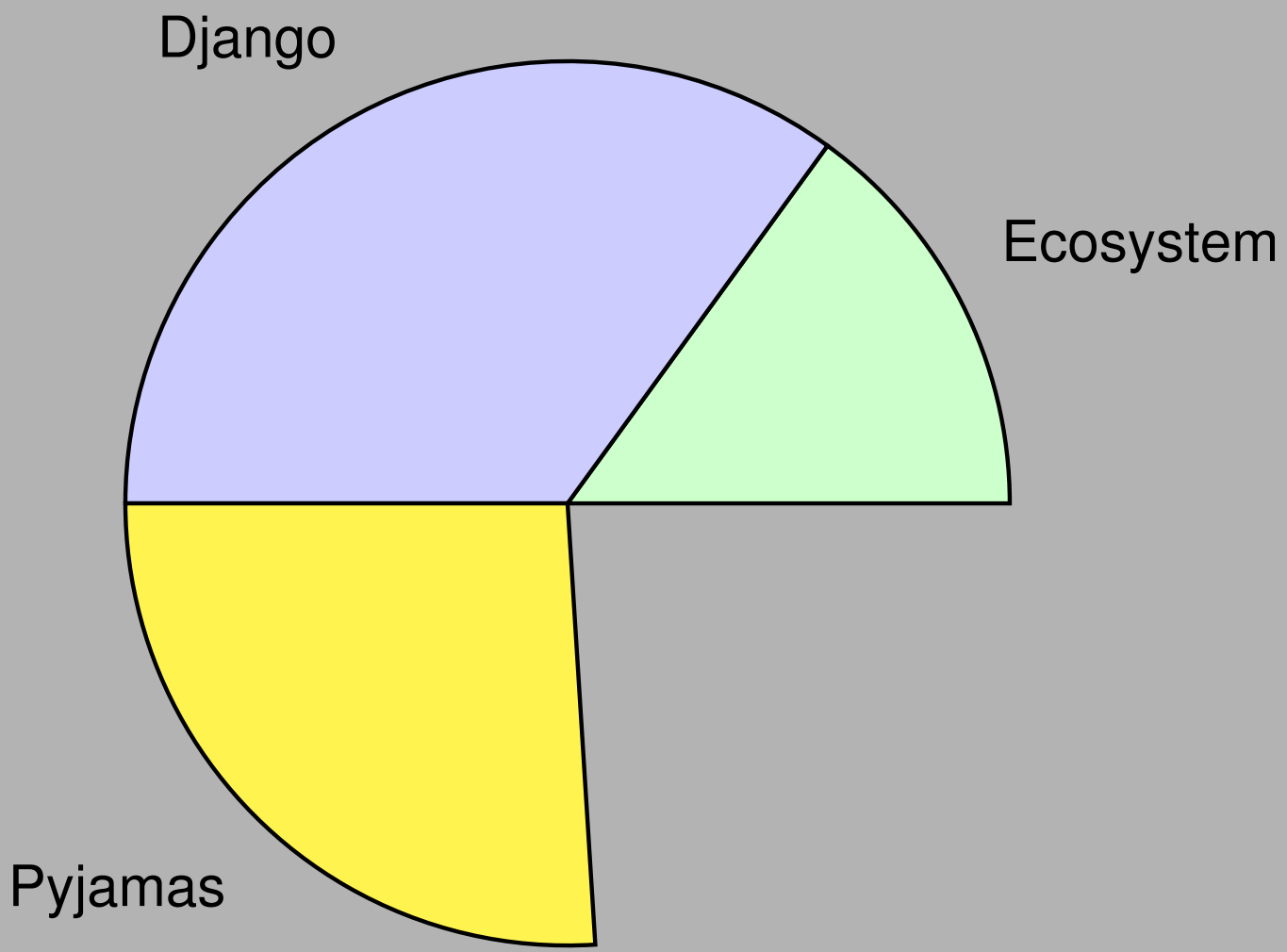
Let's Rock

- Launch django server

```
1 python manage.py runserver
```

- Launch the browser and type:
  `http://127.0.0.1:8000/myportal`

Part III: Pyjamas



Who the hell is Pyjamas?

- API compatibile with GWT
- python implementation of java script
- compiler
- java script generator
- talks
  - python
  - javascript
  - JSONRPC

What Pyjamas gives?

- dynamic and reusable UI components
- python types simulated in JavaScript
- RPC mechanism
- handles cross-browser issues
- allows mix .js code with .py code
- design of application in python object oriented fashion
- itself for free

Supported modules from python

- base64.py
- binascii.py
- csv.py
- datetime.py
- math.py
- md5.py
- os.py
- random.py
- re.py
- sets.py
- struct.py
- sys.py
- time.py
- urllib.py

Supported built-in types

- dict
- frozenset
- int
- list
- long
- object
- set
- tuple

Requirements

- pyjamas

```
1 git clone git://pyjs.org/git/pyjamas.git
2 cd pyjamas
3 python bootstrap.py
```

- libcommondjango (*for* `JSONRPCService` *and* `jsonremote`)

```
1 svn checkout http://svn.pimentech.org/pimentech/libcommonDjango
2 cd libcommonDjango
3 make install
```

Create pyjamas application

Create file: `MyApp.py` in `myapp` directory:

```
1  from pyjamas.ui.RootPanel import RootPanel
2
3  import LoginPanel
4
5  class MyApp:
6
7      def __init__(self):
8          self.mainPanel = LoginPanel.LoginPanel()
9
10     def onModuleLoad(self):
11         RootPanel().add(self.mainPanel)
12
13
14 if __name__ == '__main__':
15     app = MyApp()
16     app.onModuleLoad()
```

Create a panel

Create file: `LoginPanel.py` in `myapp` directory:

```
1  from pyjamas.ui.TextBox import TextBox
2  from pyjamas.ui.Label import Label
3  from pyjamas.ui.HorizontalPanel import HorizontalPanel
4
5  class LoginPanel(HorizontalPanel):
6
7      def __init__(self):
8          HorizontalPanel.__init__(self)
9          self.setBorderWidth(0)
10         self.layout()
11
12     def layout(self):
13         self.loginLabel = Label('Login')
14         self.loginLabel.setHeight("20px")
15
16         self.loginText = TextBox()
17         self.loginText.setHeight("20px")
18         self.loginText.setWidth("60px")
19
20         self.add(self.loginLabel)
21         self.add(self.loginText)
```

Widgets

Pyjamas offers a lot of widgets in package:
`pyjamas.ui.widgets`

- Button
- Calendar
- CheckBox
- DialogBox
- DockPanel
- FlowPanel
- FormPanel
- Frame
- HTML
- HorizontalPanel
- HorizontalSlider
- Hyperlink
- Image
- Label
- ListBox
- MenuBar
- MenuItem
- Panel
- PasswordTextBox
- PushButton
- RadioButton
- ScrollPanel
- TextArea
- TextBox
- Tooltip
- Tree
- TreeItem

Add more controls

- Add password control to `LoginPanel` class

```
1         self.loginText.addKeyboardListener(self)
2
3         self.passwdLabel = Label('Password')
4         self.passwdLabel.setHeight("20px")
5
6         self.passwdText = PasswordTextBox()
7         self.passwdText.setHeight("20px")
8         self.passwdText.setWidth("60px")
9         self.passwdText.addKeyboardListener(self)
10
11        self.add(self.passwdLabel)
12        self.add(self.passwdText)
```

- Do not forget imports

```
1 from pyjamas.ui import KeyboardListener
2 from pyjamas.ui.PasswordTextBox import PasswordTextBox
```

Listeners

- ClickListener
- FocusListener
- KeyboardListener
- MouseListener
- MultiListener

Compile to javascript

Use `pyjsbuild`:

```
1 pyjsbuild MyApp.py
2 pyjsbuild LoginPanel.py
```

to generate `output/` directory containing html files with javascript

Update `urls.py` to see Pyjamas application

- Add the following line to `urls.py`

```
1
2       (r'^myportal/myapp/(?P<path>.*)$', 'django.views.static.serve',
3           {'document_root':
4               str(os.path.join(os.path.dirname(__file__),
5                   '../myapp/output').replace('\\','/'))}),
```

- Do not forget to:

```
1   import os
```

Implement KeyboarListener methods

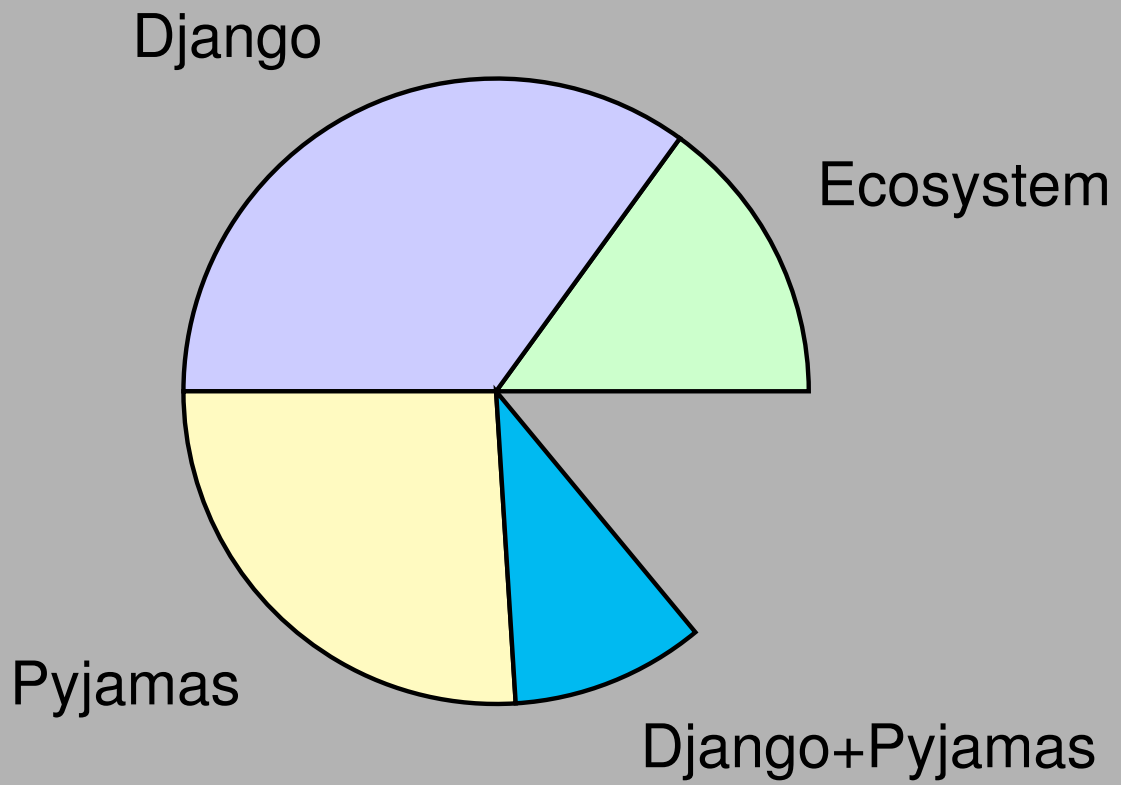
- add the following code to `LoginPanel`

```
1     def onKeyPress(self, sender, keyCode, modifiers):
2
3         if keyCode == KeyboardListener.KEY_ENTER:
4             if sender == self.loginText:
5                 if self.passwdText.getText() == "":
6                     self.passwdText.setFocus()
7                 else:
8                     self.doLogin(self.loginText.getText(),
9                         self.passwdText.getText())
10
11            elif sender == self.passwdText:
12                self.doLogin(self.loginText.getText(),
13                    self.passwdText.getText())
```

- add method to send login request

```
1     def doLogin(self, login, passwd):
2         ret = self.service.Login(login, passwd, self)
```

Part IV: Django+Pyjamas



Connect `LoginPanel` to django application

- create connection point in `LoginPanel.py`

```
1  class DataService(JSONProxy):
2      def __init__(self):
3          JSONProxy.__init__(self, "/login_service/", ["Login"],
4              headers= { 'X-CSRFToken': Cookies.getCookie('csrftoken') } )
```

- do not forget imports

```
1  from pyjamas.JSONService import JSONProxy
2  from pyjamas import Cookies
```

- add member to `LoginPanel` constructor

```
1          self.service = DataService()
```

- update `urls.py`

```
1      (r'^login_service/$', 'myapp.views.service'),
```

Handle response from server

- add response method to `LoginPanel` class

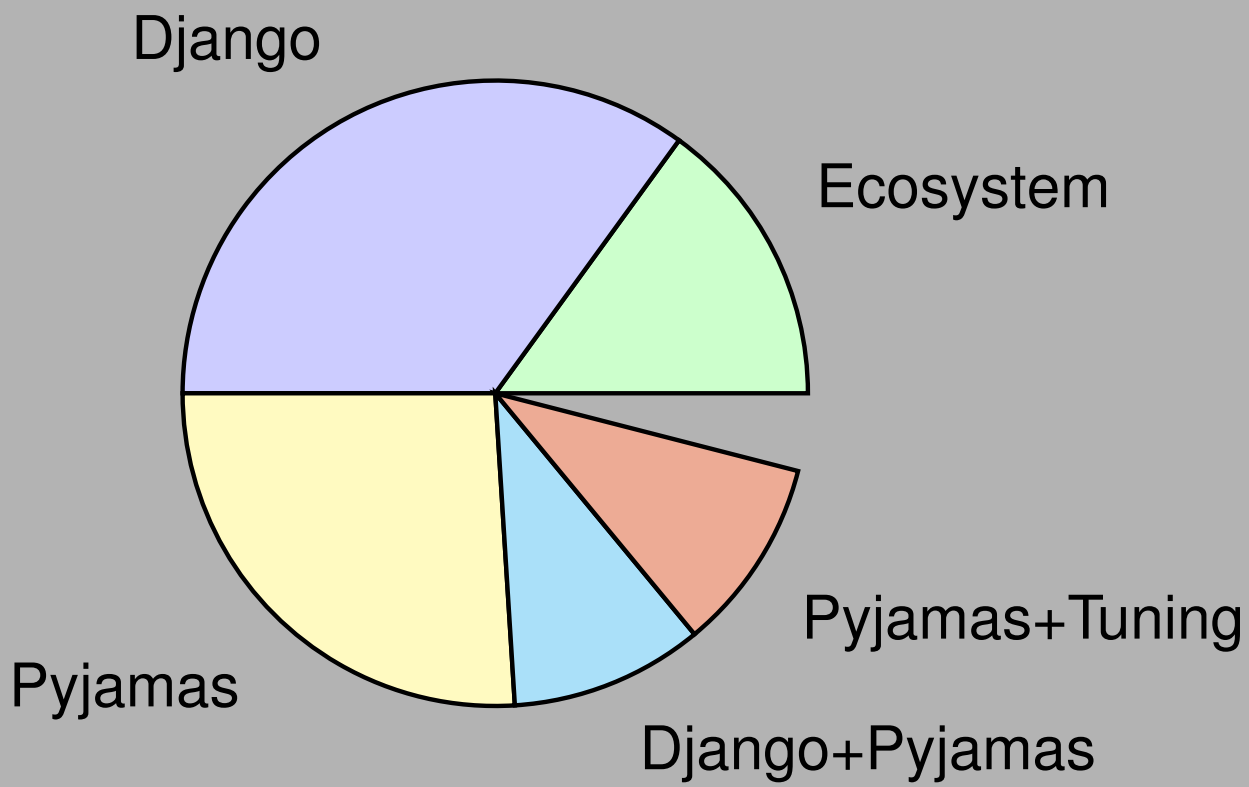
```
1     def onRemoteResponse(self, response, request_info):
2         self.layout()
3
4         if response['name']:
5             self.add(Label(response['name']))
```

Implement Login method on server-side

```
1  from django.pimentech.network import JSONRPCService, jsonremote
2
3  service = JSONRPCService()
4
5  @jsonremote(service)
6  def Login(request, login, passwd):
7
8      import models
9
10     username = 'Unknown'
11
12     user = models.Users.objects \
13         .filter(login=login).filter(passwd=encrypt(passwd))
14
15     if len(user) == 1:
16         username = user[0].name
17
18     response = { 'name' : username }
19
20     return response
```

Let's Rock!

Part IV: Pyjamas+Tuning



Tune up with CSS (Cascade Style Sheet)

- in `myapp` create directory `public` for CSS and overriden htmls
- use generated `MyApp.html`

```
1  <html>
2  <!-- auto-generated html - you should consider editing and
3  adapting this to suit your requirements
4  -->
5  <head>
6  <meta name="pygwt:module" content="Login">
7  <link href="MyApp.css" rel="stylesheet">
8  <title>Login module</title>
9  </head>
10 <body bgcolor="white">
11 <script language="javascript" src="bootstrap.js"></script>
12 <iframe id='__pygwt_historyFrame' style='width:0;height:0;border:0'></iframe>
13 </body>
14 </html>
```

Create `MyApp.css`

```
1  .graybutton
2  {
3      font-size:11px;
4      font-family:Verdana,sans-serif;
5      font-weight:bold;
6      color:#888888;
7      width:100px;
8      background-color:#eeeeee;
9      border-style:solid;
10     border-color:#BBBBBB;
11     border-width:1px;
12 }
13
14 .graybuttonlight
15 {
16     font-size:11px;
17     font-family:Verdana,sans-serif;
18     font-weight:bold;
19     color:#66aaaa;
20     width:100px;
21     background-color:#eeeef4;
22     border-style:solid;
23     border-color:#9999dd;
24     border-width:1px;
25 }
```

Create custom highlight button

```
1  from pyjamas.ui.Button import Button
2
3  class HighlightButton(Button):
4
5      def __init__(self, html=None, listener=None, **kwargs):
6          Button.__init__(self, html, listener, **kwargs)
7          self.normalStyle = self.getStyleName()
8          self.highlightStyle = self.getStyleName()
9          self.addMouseListener(self)
10
11     def setNormalStyle(self, style):
12         self.normalStyle = style
13
14     def setHighlightStyle(self, style):
15         self.highlightStyle = style
16
17     def onMouseEnter(self, sender, event):
18         Button.onMouseEnter(self, sender, event)
19         self.setStyleName('graybuttonlight')
20
21     def onMouseLeave(self, sender, event):
22         Button.onMouseLeave(self, sender, event)
23
24         self.setStyleName('graybutton')
```

Logout functionality

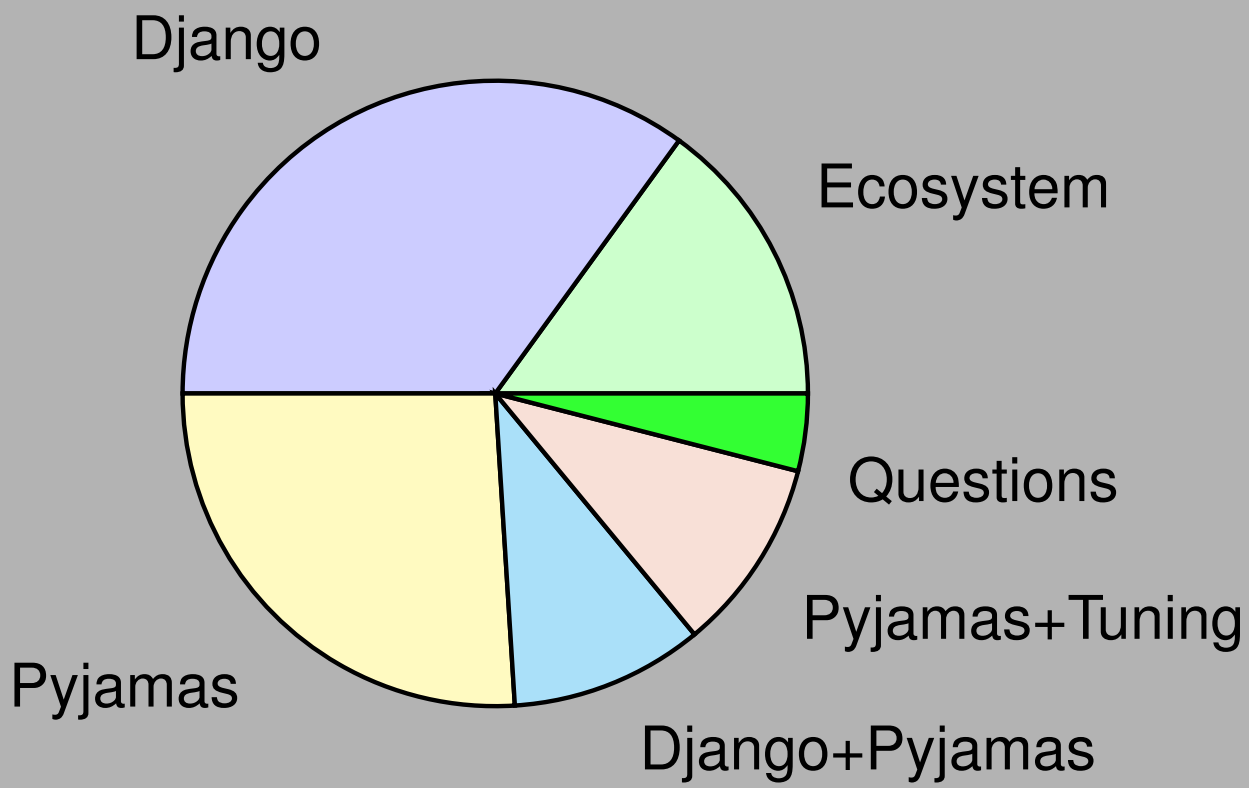
- JSONProxy (same as Login)
- button on GUI
- method on service

Let's Rock!

Summary

- complex technology
- advanced
- object oriented
- all with python
- open and free

Part VI: Questions



Usefull links

- http://djangoproject.com
- http://pyjs.org
- http://pyjs.org/api/

Credits

1. flickr/python/raym5
2. flickr/server upgrade/msarturlr
3. flickr/drinks/kdinuraj