



TurboGears2

Building
Full Featured Web Applications
with TurboGears2
in a bunch of minutes

Alessandro Molina - @__amol__ - amol@turbogears.org

TURBOGEARS 



TurboGears2

- Framework for rapid development encouraging customization
- Object Dispatch based. Regular expressions can get messy, never write a regex anymore
- By default an XML template engine with error detection
- Declarative Models with transactional unit of work
- Built in Validation, Authentication, Authorization, Caching, Sessions, Migrations, MongoDB Support and many more.



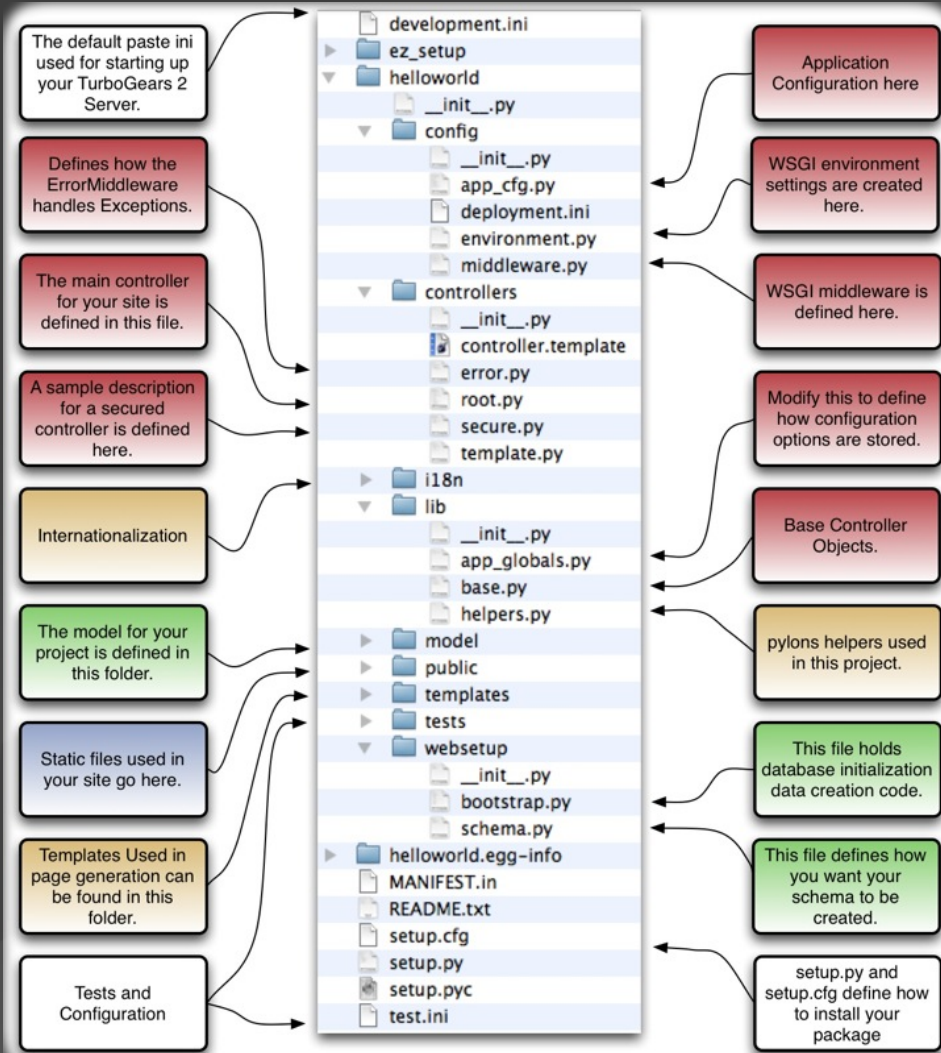
Looking at the code

Serving /movie/3 as a webpage and /movie/3.json as a json encoded response

```
class RootController(BaseController):  
  
    @expose('myproj.templates.movie')  
    @expose('json')  
    @validate({'movie':SQLAEntityConverter(model.Movie)})  
    def movie(self, movie, **kw):  
        return dict(movie=movie, user=request.identity and request.identity['user'])
```



What it looks like





TurboGears for RAD

- 2.0 had sprox and tgext.crud: Flexible, but hard to use!
- 2.1 had many sprox improvements and added the EasyCrudRestController
- 2.1.4 had many hooks improvements that made tgext.pluggable possible!

With EasyCrudRestController and pluggable applications rapid prototyping can be rapid for real



EasyCrudRestController

Aims at making possible to create full administrative interfaces in a bunch of seconds

The screenshot shows a web browser window with the URL `localhost:8080/admin/users/`. The page header includes the TurboGears2 logo and navigation links: `Welcome`, `About`, `Serving Data`, `WSGI Environment`, `Logout`, and `Admin`. The main content area is titled `User Listing` and includes a sidebar with `Group`, `Permission`, and `User` links. A `New User` button is present. A search bar contains `user_name` and `equals`. Below is a table of users:

actions	user_name	email_address	display_name	created	groups
edit delete	manager	manager@somedomain.com	Example manager	06/08/2012 02:41AM	managers
edit delete	editor	editor@somedomain.com	Example editor	06/08/2012 02:41AM	

Copyright © TurboGears2 2012

TURBOGEARS under the hood



Easy CRUD

Minimal setup is minimal for real!

```
from tgext.crud import EasyCrudRestController  
  
class GalleriesController(EasyCrudRestController):  
    model = model.Gallery
```

This provides CRUD interface with Search, ordering and autogenerated JSON Rest API for that model.



Custom CRUD

Customizing the Crud Controller can be done from the `__form_options__` and `__table_options__` variables of the class.

```
class PhotosController(EasyCrudRestController):
    model = model.Photo
    allow_only = predicates.in_group('photos')
    title = "Manage Photos"
    keep_params = ['gallery']

    __form_options__ = {
        '__hide_fields__': ['uid', 'author', 'gallery'],
        '__field_widget_types__': {'image': FileField},
        '__field_validator_types__': {'image': FieldStorageUploadConverter},
        '__field_widget_args__': {'author': {'default': lambda: request.identity['user'].user_id}}
    }

    __table_options__ = {
        '__omit_fields__': ['uid', 'author_id', 'gallery_id', 'gallery'],
        '__xml_fields__': ['image'],
        'image': lambda filler, row: html.literal('' % row.image.thumb_url)
    }
```






Custom CRUD Result

Result is a web page to upload photos to a gallery with uploaded image preview

← → ↻ 🏠 🌐 localhost:8080/photos/manage_photos/?gallery=2

Photo Listing

[+ New Photo](#)

Actions	Name	Description	Image	Author
edit	Mountain Valley	Mountain Valley		manager
edit	Winter Reflections	Winter Reflections		manager



Let's plug them all

CRUD tools provide a quick and easy way to prototype new functions.

But... there are things which are not a plain CRUD, how can you speed up their development?

Fastest solution is to have things already done by someone else!

That's what pluggable applications are for



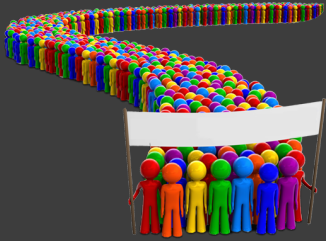
Pluggables

Pluggable applications provide ready made features that can be plugged into your applications.

- Implemented as *tgext.pluggable*, if you don't use them they won't bother you
- As easy as *plug(base_config, 'appname')*
- They look a lot like a plain application and provide models, controllers, templates, helpers, partials, database bootstrap and so on.
- Creating one as easy as *paster quickstart-pluggable appname*
- Sadly supported only for SQLAlchemy storage backed, mongodb planned



Available Pluggables



tgapp-registration

Provides a registration process with activation email. It's heavily customizable using hooks.

tgapp-smallpress

Provides multblog with WYSIWYG editor, tagcloud, attachments, drafts, future publications and Whoosh based search.



tgapp-fbauth

facebook authentication, registration and connection to existing accounts.

tgapp-photos

Provides partials to display photos and photo galleries with automatic thumbnails.





Available Pluggables



tgapp-userprofile

Provides a basic profile page and badge for users with profile picture took from facebook, gravatar or custom source.

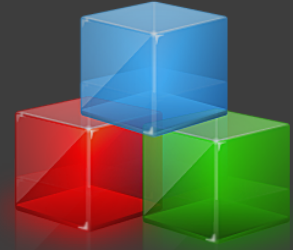


stroller

Provides eCommerce application with categories, multiple pictures for each product and orders management.

libacr

Provides a powerful CMS where pages are splitted into slices each editable and of its own type.



Custom slices and new type of contents can be easily created directly from the CMS itself without editing code.



DebugBar

To improve developers life the first pluggable application created has been the debugbar.

Query	Params	Time	Actions
<pre>SELECT count(*) AS count_1 FROM (SELECT tg_user.password AS tg_user_password, tg_user.user_id AS tg_user_user_id, tg_user.user_name AS tg_user_user_name, tg_user.email_address AS tg_user_email_address, tg_user.display_name AS tg_user_display_name, tg_user.created AS tg_user_created FROM tg_user) AS anon_1</pre>	[]	5.0361 ms	RESULTS EXPLAIN
<pre>SELECT tg_user.password AS tg_user_password, tg_user.user_id AS tg_user_user_id, tg_user.user_name AS tg_user_user_name, tg_user.email_address AS tg_user_email_address, tg_user.display_name AS tg_user_display_name, tg_user.created AS tg_user_created FROM tg_user</pre>	[]	5.2230 ms	RESULTS EXPLAIN
<pre>SELECT tg_group.group_id AS tg_group_group_id, tg_group.group_name AS tg_group_group_name, tg_group.display_name AS tg_group_display_name, tg_group.created AS tg_group_created FROM tg_group, tg_user_group WHERE ? = tg_user_group.user_id AND tg_group.group_id = tg_user_group.group_id</pre>	[2]	5.0859 ms	RESULTS EXPLAIN

- Controller methods profiling
- Template Rendering timings
- Query inspector
- Request inspector and so on... The usual things you would expect from a debug bar!



Inventing Mode

How the debugbar relates to rapid prototyping?
Well, it makes life easier, but mostly... It provides the inventing mode!

Place your browser and code editor side by side and start experimenting, your changes will reflect into the browser in real time and it will notify you when you broke something.





Creating pluggables

Once `tgext.pluggable` gets installed the *quickstart-pluggable* command becomes available.

Running `quickstart-pluggable` will create a package that looks a lot like a TurboGears application but provides a *plugme* method inside its `__init__.py`

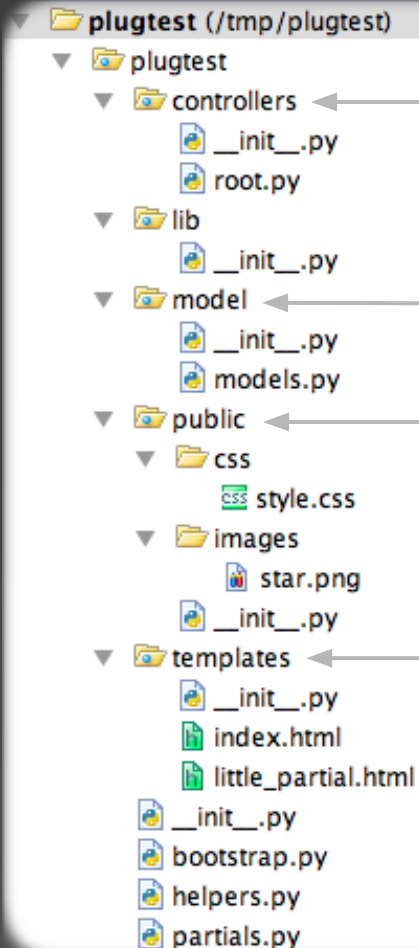
`plugme` method is the entry point of your pluggable application.

```
def plugme(app_config, options):  
    return dict(appid='plugtest', global_helpers=False)
```




Structure of a Pluggable

```
$ paster quickstart-pluggable plugtest
```



Pluggable Applications controllers, root controller of the application is named `RootController` inside the `root.py` file and will be mounted as `/plugtest`

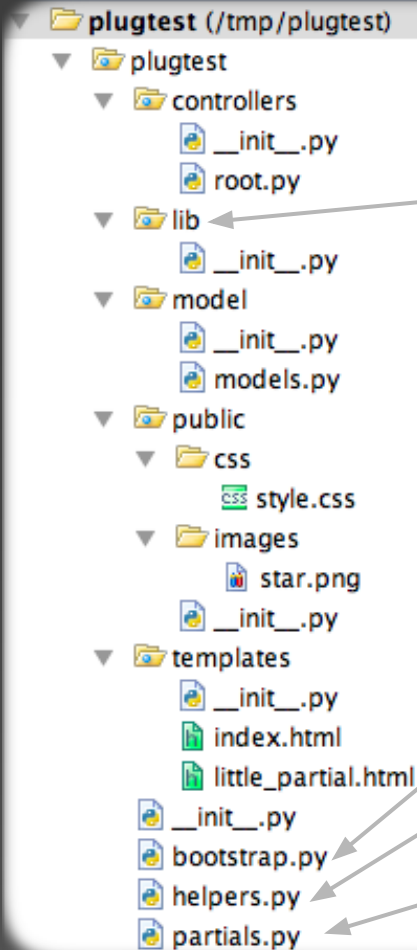
Models of the pluggable applications, will be bound to the session of the master app

Static files of the pluggable application, will be available at `/_pluggable/plugtest`

Templates of the pluggable application, controllers can use them with standard expose syntax: `@expose('plugtest.templates.index')`



Structure of a Pluggable



Here rely all the utility functions of the pluggable application. By default they are not exposed to the master application but are still accessible as a python module

bootstrap is automatically called when initializing the database of the master application.

Pluggables can provide helpers which will be automatically available into the master application as *h.plugtest.helpername*

Partials are evolved helpers, they provide logic and look like controllers with an exposed template. They are accessible inside templates with *h.call_partial('plugtest.partials: partialname')*



Pluggable Utilities

`tgext.pluggable` provides a bunch of utilities that help when working with pluggable applications to override part of their aspect or behavior:

- `replace_template(base_config, 'otherapp.templates.about', 'myapp.templates.index')` permits to replace any template with another one, makes possible to change the look of any plugged web page
- `plug_url('plugtest', '/somewhere')` makes possible to generate urls relative to a plugged application
- `tgext.pluggable.app_model` provides the models of the application where the pluggable app will be plugged.
- `tgext.pluggable.primary_key(Model)` detects the primary key of a model so that it is possible to create relations to models we don't know how they look like.



That's all Folks!

Join turbogears@googlegroups.com for more details!