# Quick web application building with TurboGears

## A short tutorial

RuPy conference, Poznań

14.04.2007

Christopher Arndt <chris@chrisarndt.de>

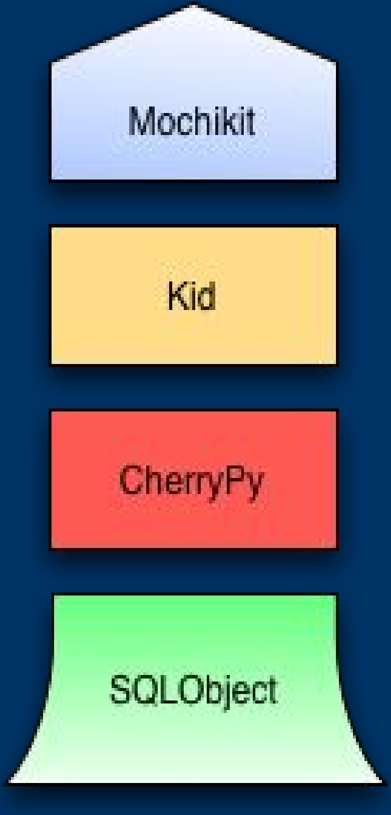
## What is TurboGears?

- A Python web meta framework!
- Comparable to Django and Ruby on Rails
- Open Source (MIT License)
- Still young (1st public version autumn 2005)
- Buzzword compliant: MVC, AJAX(J), REST etc.

## *What can it be used for?*

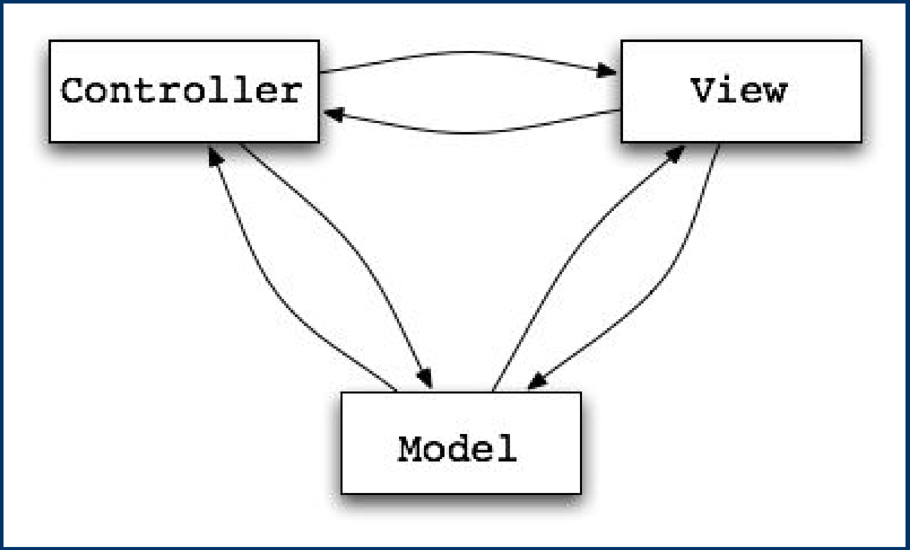
- „Classic“ web apps, e.g. Blogs, Wikis, CMS
- Intranet apps, e.g. *WhatWhat Status*
- Web administration front ends, e.g. *WebFaction.com Control Panel*
- „Microapps“ (http://microapps.org/) à la *Gravatar.com*, *Websnapr.com*, etc.
- See http://docs.turbogears.org/1.0/SitesUsingTurboGears

## Which components make up the TurboGears framework?



- Database abstraction: *SQLObject*
- Application server: *CherryPy*
- Template engine: *Kid*
- Client-side JavaScript: *MochiKit*
- plus several other bits, including:
  - *FormEncode* (Validierung),
  - *Nose* (Unit tests),
  - *simplejson* (JSON) and many more...

## The Model-View-Controller pattern and you



- MVC = Model / View / Controller
- Web applications:
  database / data retrieval methods / templates
- Goal: separation of components for easier replacement
- Easier to grasp by example later

## 10 steps to your TurboGears application

1. Quickstart your project
2. Code your data model
3. Create the database
4. Add some bootstrap data using CatWalk
5. Design your URLs
6. Write your controller methods
7. Write your templates
8. Add some CSS and/or JavaScript
9. Build an egg
10. Deploy!

## *A simple example*

**Yet another Bookmark directory**

## Step 1: Quickstart your application

```
$ tg-admin quickstart
Enter project name: Bookmarker
Enter package name [bookmarker]:
Do you need Identity (usernames/passwords) in
 this project? [no] yes

[ long output follows...]

$ cd Bookmarker
```

## Step 2: Code you data model

- Two application-specific data objects:
  - Bookmarks
  - Tags
- TurboGears creates standard data objects for us:
  - Users
  - Groups
  - Permissions

## Step 2 (cont.): Data model - Bookmarks

**Bookmark** properties:

- Title (text, one-line)
- URL (text, one-line)
- Description (text, multi-line)
- Creation time (timestamp)
- Owner (one-to-many: User )
- Tags (many-to-many: Tag)

## Step2 (cont.): Data model - Bookmark objects

```
# in model.py:

class Bookmark(SQLObject):

    title = UnicodeCol(length=255, notNull=True)
    url = UnicodeCol(length=255, notNull=True)
    description = UnicodeCol()

    tags = RelatedJoin('Tag', orderBy='name')

    # meta data
    created = DateTimeCol(default=datetime.now)
    owner = ForeignKey('User', notNull=True)
```

## Step 2 (cont.): Data model - Tags

**Tag** properties:

- Label (text, one-line)
- Name (text, one-line)
- Creation time (timestamp)
- Owner (one-to-many: Tag)
- Bookmarks (many-to-many: Bookmark)

## Step2 (cont.): Data model - Tag objects

```
 # still in model.py:

class Tag(SQLObject):

    name = UnicodeCol(length=100, notNull=True)
    label = UnicodeCol(length=100, notNull=True)

    bookmarks = RelatedJoin('Bookmark',
      orderBy='-created')

    # meta data
    owner = ForeignKey('User', notNull=True)
    created = DateTimeCol(default=datetime.now)
```

## Step 3: Create the database

Everything is already set up for the default **SQLite** backend:

```
$ tg-admin sql create
Using database URI
sqlite:///home/chris/Bookmarker/devdata.sqlite
$
```

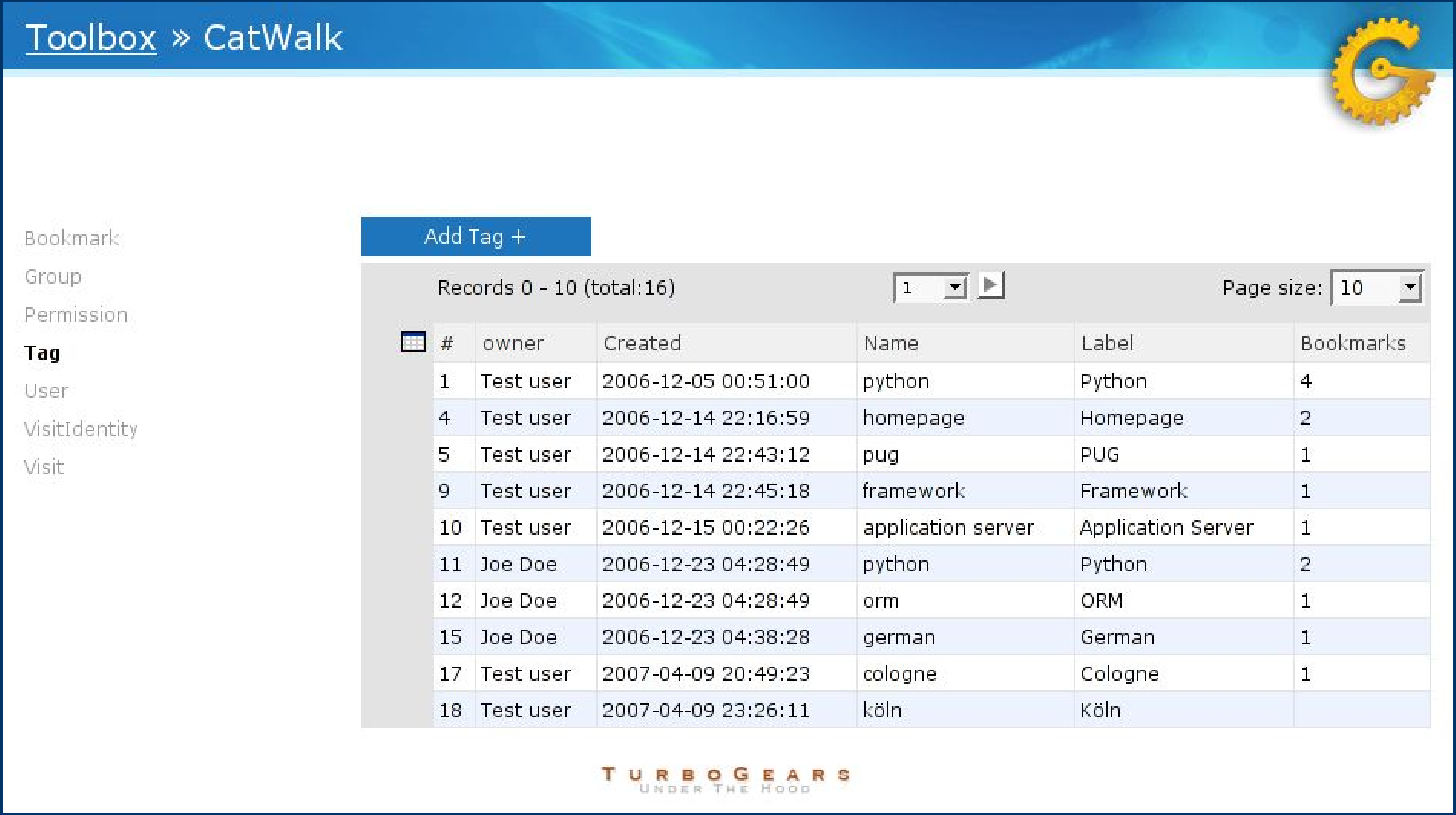
## Step 4: Add bootstrap data

- TurboGears comes with a nice web administration interface called **CatWalk**.
- We'll add groups, users and permissions and a few bookmarks and tags.

```
$ tg-admin toolbox
[...]
HTTP INFO Serving HTTP on http://localhost:7654/
[...]
```

- Open web browser at *http://localhost:7654/*

# Step 4 (cont.): CatWalk



Records 0 - 10 (total:16)    1    Page size: 10

| # | owner | Created | Name | Label | Bookmarks |
|---|---|---|---|---|---|
| 1 | Test user | 2006-12-05 00:51:00 | python | Python | 4 |
| 4 | Test user | 2006-12-14 22:16:59 | homepage | Homepage | 2 |
| 5 | Test user | 2006-12-14 22:43:12 | pug | PUG | 1 |
| 9 | Test user | 2006-12-14 22:45:18 | framework | Framework | 1 |
| 10 | Test user | 2006-12-15 00:22:26 | application server | Application Server | 1 |
| 11 | Joe Doe | 2006-12-23 04:28:49 | python | Python | 2 |
| 12 | Joe Doe | 2006-12-23 04:28:49 | orm | ORM | 1 |
| 15 | Joe Doe | 2006-12-23 04:38:28 | german | German | 1 |
| 17 | Test user | 2007-04-09 20:49:23 | cologne | Cologne | 1 |
| 18 | Test user | 2007-04-09 23:26:11 | köln | Köln | |

## Step 5: Designing your URLs

- http://mysite/bookmarks/
  /bookmarks/
  } List of bookmarks
- /bookmarks/<*id*>
  /bookmarks/<*id*>/view
- /bookmarks/<*id*>/edit
  /bookmarks/<*id*>/add
  } Show bookmark details / Show edit form
- /bookmarks/<*id*>/delete → Delete bookmark
- /bookmarks/<*id*>/update → Update bookmark

## Step 5 (cont.): URL mapping

- URL mapping is the process of turning a request for a certain URL into a function or method call in your web application.
- Example:

**http://mysite.com/bookmarks/edit/1**

- Question: which part of the URL is the method name and which are the parameters?

## Step 5 (cont.): URL mapping à la CherryPy

```
# in controllers.py:

class BookmarkController(controller.Controller):
    @expose()
    def edit(self, id):
        return “The given ID is %s” % id

class Root(controller.RootController):
    bookmarks = BookmarkController()
```

URL: *http://mysite/bookmarks/edit/1*

Resulting call: `Root().bookmarks.edit(1)`

## Step 5 (cont.): CherryPy REST URL mapper

```
@expose()
def default(self, *params, **kw):
    if len(params) == 1:
        id = params[0]
        redirect(url('%s/view') % id)
    elif len(params) >= 2:
        id, verb = params[:2]
        action = getattr(self, verb, None)
        if not action or not \
          getattr(action,'exposed'):
            raise cherrypy.NotFound
        action(item, *params[2:], **kw)
```

## Step 6: Write controller methods

We need the following methods:

1. Show a welcome page*
2. Show list of bookmarks
3. Show bookmark details / edit form
4. Show form for new bookmark*
5. Create/Update bookmark from form submission
6. Delete bookmark

* left as exercise for the reader

## Step 6 (cont.): Controller methods List of bookmarks

```
# in controllers.py:

class BookmarksController(controllers.Controller):

    @expose(template='.templates.bookmarks.list')
    def index(self):
        bookmarks = Bookmark.select()
        return dict(entries=bookmarks)

    list = index
```

## Step 6 (cont.): Controller methods Show bookmark details / edit form

```
# still in controllers.py:

class BookmarksController(...):
    ...

    @expose(template='.templates.bookmarks.edit')
    def view(self, id, *params, **kw):
        try:
            bookmark = Bookmark.get(id)
        except SQLObjectNotFound:
            flash('Bookmark not found.')
            redirect('/')
        return dict(entry=bookmark)
```

## Step 6 (cont.): Controller methods Update/Create bookmark

```
@expose()
def update(self, id, *params, **kw):
    try:
        bookmark = Bookmark.get(id)
    except SQLObjectNotFound:
        bookmark = Bookmark(
          title = kw.get('title'),
          url = kw.get('url'),
          description = kw.get('description'))
    else:
        bookmark.set(
          title = kw.get('title'), url=...)
    # TODO: handle tags specially
    redirect('/bookmarks/')
```

## Step 6 (cont.): Controller methods Delete bookmark

```
@expose()
def delete(self, id, *params, **kw):
    try:
        Bookmark.delete(id)
    except SQLObjectNotFound:
        flash('Bookmark not found.')
    else:
        flash('Bookmark deleted.')
    redirect('/bookmarks')
```

## Step 7: Edit templates List of bookmarks

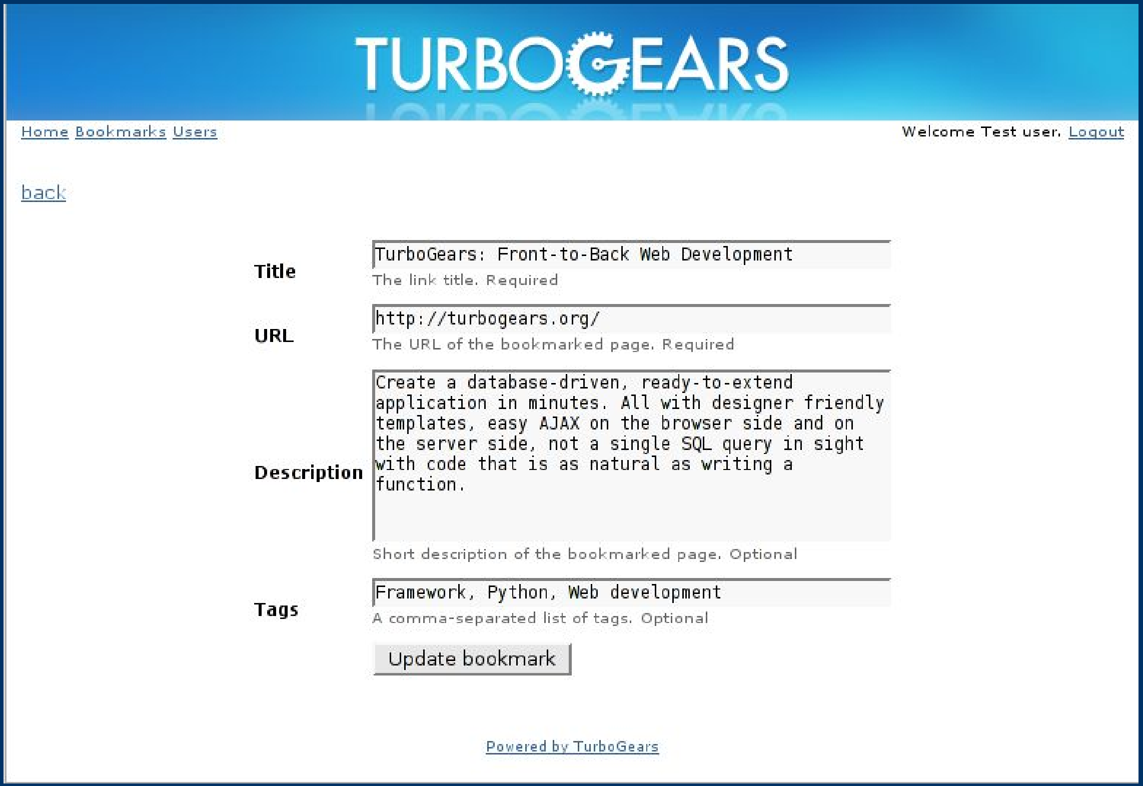
```
<!-- templates/list.kid -->

<?python item = tg.ipeek(entries) ?>

<div py:if="item" class="bookmarks">
  <dl py:for="bookmark in entries">
    <dt><a href="${bookmark.url}"
      py:content="bookmark.title" /></dt>

    <dd><p py:content="bookmark.description" />
      <p><a href="${tg.url('/bookmarks/%i/edit' %
        bookmark.id)}">Edit</a></p></dd>
  </dl>
</div>
<div py:if="not item" class="bookmarks">
  No bookmarks found
</div>
```

## Step 7 (cont.): Edit templates Show bookmark / edit form

```
<!-- templates/edit.kid -->

<form action="update" method="POST">
 <input type="text" name="title" value="${entry.title}" />

 <input type="text" name="url" value="${entry.url}" />

 <textarea name="description">
  ${entry.description}
 </textarea>

 <input type="text" name="tags"
  value="${','.join([tag.name for tag in entry.tags])}" />

 <input type="submit" value="Save">
</form>
```

## Step 8: Add CSS and/or JavaScript

- Edit `static/css/style.css` and give your application a facelift:



|  |  |
|---|---|
| Title | TurboGears: Front-to-Back Web Development<br>The link title. Required |
| URL | http://turbogears.org/<br>The URL of the bookmarked page. Required |
| Description | Create a database-driven, ready-to-extend application in minutes. All with designer friendly templates, easy AJAX on the browser side and on the server side, not a single SQL query in sight with code that is as natural as writing a function.<br>Short description of the bookmarked page. Optional |
| Tags | Framework, Python, Web development<br>A comma-separated list of tags. Optional |

## Step 9: Build an egg

- Edit `release.py` to add package meta data.
- `python setup.py bdist_egg`
- Copy egg to target host and do

  ```
  easy_install <egg-file>
  ```
- See http://docs.turbogears.org/1.0/DeployWithAnEgg for more information

## Step 10: Deployment options

- Pure CherryPy-Server (for development/testing)
- Apache with **mod_proxy** (recommended)
- Apache with **mod_python**
- Alternative light-weight webservers:
  - **nginx** (my favourite)
  - **LighTTP**

## Conclusion

- We edited **3 Python source code** files:
  - model.py
  - controllers.py
  - release.py
- We edited **3 Kid template** files:
  - welcome.kid
  - list.kid
  - edit.kid
- Plus some CSS
- and **no SQL statement** in sight!

## *What's next?*

- Read the book:
  - http://www.turbogearsbook.com/
- Visit the Wiki:
  - http://docs.turbogears.org/
- Easy forms with TurboGears **widgets**:
  - http://docs.turbogears.org/1.0/Widgets
- The future: **SQLAlchemy** and **Genshi**:
  - http://docs.turbogears.org/1.0/SQLAlchemy
  - http://docs.turbogears.org/1.0/GenshiTemplating

## *Thank you for listening!*

## *Questions?*

**Slides and sample code at:**

***http://chrisarndt.de/talks/rupy/***

## Appendix
## Easy controllers with FastData

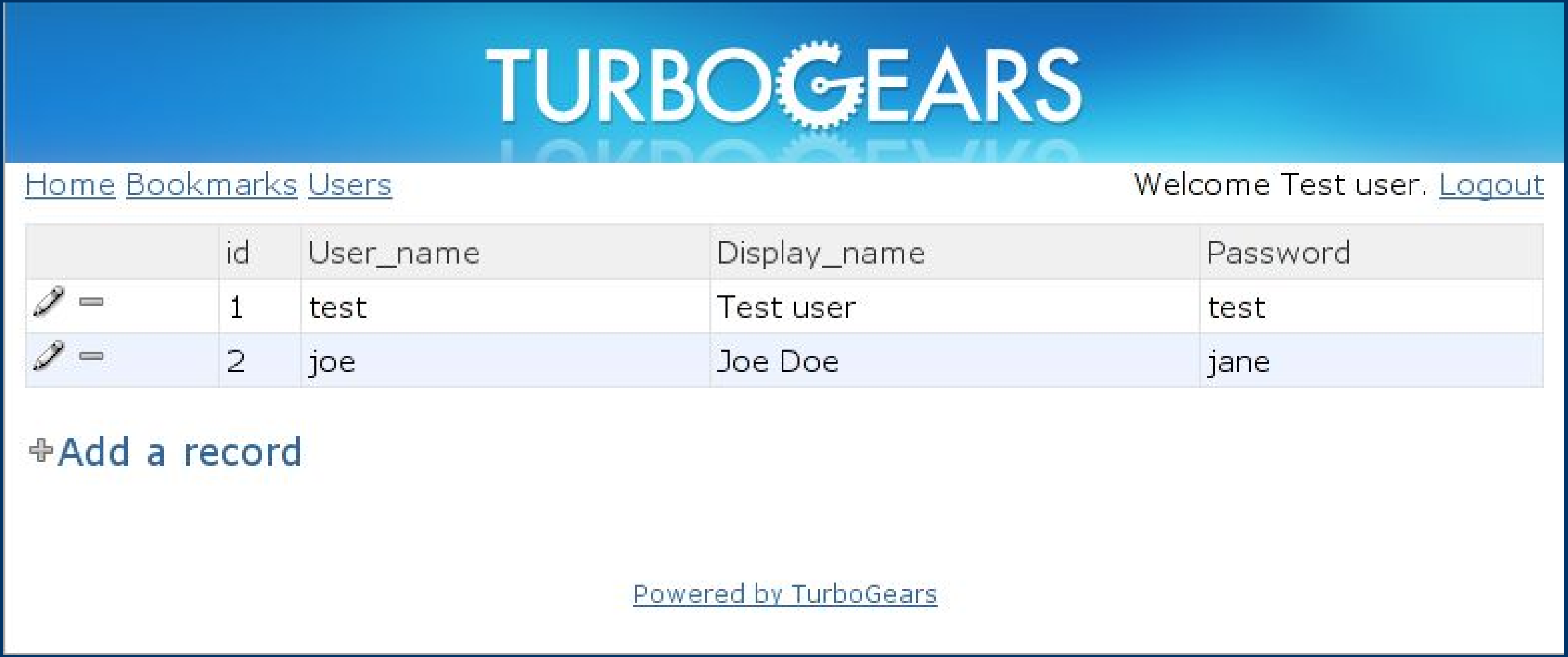
- **FastData** is a TurboGears extension.
- Build CRUD interface for your model objects with <10 lines of code
- Current version needs my patch which will be in SVN soon (hopefully).
- Works only with SQLObject.

## *FastData example: User administration*

```
from turbogears import controllers, identity
from tgfastdata import DataController
from bookmarker.model import User

class Root(controllers.RootController):

    # the FastData controller
    users = DataController(User,
        object_name = 'User',
        list_fields = ['id', 'user_name',
          'display_name', 'password'])

    # only users with permission 'admin' can
    # access the user administration
    users = identity.SecureObject(users,
        identity.has_permission('admin'))
```

# FastData example: User administration

http://mysite/bookmarks/users/



Home Bookmarks Users                Welcome Test user. Logout

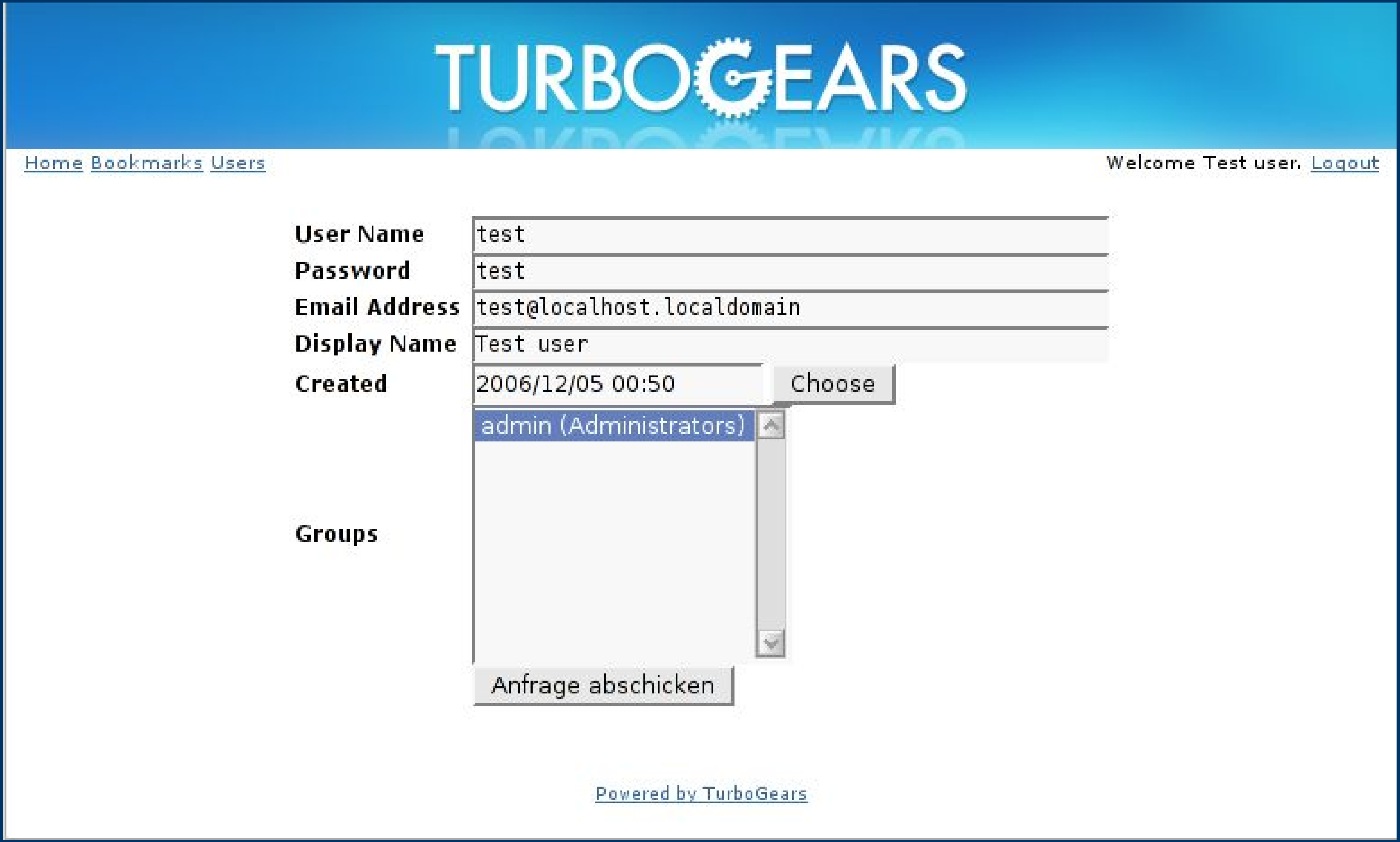
| | id | User_name | Display_name | Password |
|---|---|---|---|---|
| | 1 | test | Test user | test |
| | 2 | joe | Joe Doe | jane |

+Add a record

Powered by TurboGears

## *FastData example: User administration*

http://mysite/bookmarks/users/1/edit



| | |
|---|---|
| User Name | test |
| Password | test |
| Email Address | test@localhost.localdomain |
| Display Name | Test user |
| Created | 2006/12/05 00:50 |
| Groups | admin (Administrators) |