kiss.py Documentation

Release 0.3.3

Stanislav Feldman

January 15, 2015

Contents

MVC web framework in Python with Gevent, Jinja2, Werkzeug.

CHAPTER 1

Contents

1.1 Overview

Kiss.py is MVC web framework in Python with Gevent, Jinja2, Werkzeug.

You can break your app to views, controllers and models.

Controller is class inherited from class Controller and may have methods get, post, put, delete. These methods get Request object param and return Response object. Request and Response objects inherited from Werkzeug.

1.1.1 Features

- Django-like templates from Jinja2.
- You can add your static path in settings and all css and javascript files will be minified.
- Also html templates will be minified.
- In css files you can use SCSS syntax.
- ORM with PostgreSQL, MySQL and SQLite(Peewee).
- Event dispatcher. You can subscribe to event or publish event.
- Rest controller. You can create rest interfaces to your models with RestController.

1.2 Installation

1.2.1 Installing with pip or easy_install

```
pip install kiss.py
```

1.2.2 Installing with git

```
git clone https://github.com/stanislavfeldman/kiss.py.git
cd kiss.py
python setup.py install
```

1.3 Example

Git repository includes example minimal project. You can use it as start point.

1.3.1 main.py

```
from settings import options
from kiss.core.application import Application
app = Application(options)
app.start()
```

1.3.2 settings.py

```
from os import path
current_dir = path.dirname(path.abspath(__file__))
import sys
sys.path.append(path.join(current_dir, "../../kiss.py"))
sys.path.append(path.join(current_dir, "../../compressinja/"))
sys.path.append(path.join(current_dir, "../../putils/"))
from kiss.core.application import Application
from controllers.controller1 import Controller1
from controllers.controller2 import Controller2
from kiss.core.events import ApplicationStarted
from kiss.controllers.events import BeforeControllerAction
from kiss.models import SqliteDatabase
from kiss.core.exceptions import InternalServerError
from kiss.controllers.page import PageController
from kiss.controllers.rest import RestController
from models.models import Blog
options = {
        "application": {
                "address": "127.0.0.1",
                "port": 8080
        },
        "urls": {
                "": Controller1,
                "users": {
                        "(?P<user>\w+)": Controller2
                },
                "2": {
                        "3": Controller1(),
                        "4": Controller2
                },
                "3": PageController("static_view.html", {"foo": "bar"}),
                RestController(Blog).url: RestController(Blog).controller
        },
        "views": {
                "templates_path": "views.templates",
                "static_path": "views.static"
        },
        "events": {
                ApplicationStarted: Controller2.application_after_load,
                BeforeControllerAction: Controller2.before_controller_action,
                InternalServerError.code: Controller2.internal_server_error
        },
```

```
        "models": {
                "engine": SqliteDatabase,
                #"host": "localhost",
                "database": path.join(current_dir, "kiss_py_project.sqldb")#,#,
                #"user": 'postgres',
                #"password": "postgres"
        }
}
```

1.3.3 models/models.py

```
from kiss.models import Model, CharField, TextField, DateTimeField, BooleanField, ForeignKeyField
class Blog(Model):
        creator = CharField()
        name = CharField()
class Entry(Model):
        blog = ForeignKeyField(Blog)
        title = CharField()
        body = TextField()
        pub_date = DateTimeField()
        published = BooleanField(default=True)
```

1.3.4 controllers/controller2.py

```
from kiss.views.templates import TemplateResponse
from kiss.core.events import Eventer
from models.models import Blog, Entry
import datetime
class Controller2(object):
        def get(self, request):
                #publish some event
                eventer = Eventer()
                eventer.publish("some event", self)
                if not "foo" in request.session:
                        request.session["foo"] = 0
                request.session["foo"] += 1
                #blog = Blog()
                blog = Blog.get(id=1)
                blog.name = "super blog"
                blog.creator = "Stas"
                blog.save()
                #entry = Entry()
                entry = Entry.get(id=2)
                entry.blog = blog
                entry.title = "super post"
                entry.body = "lkoeirsldfkwierj"
                entry.pub_date = datetime.datetime.now()
                entry.save()
                return TemplateResponse("view.html", {
                        "foo": request.session["foo"],
                        "users": [{"url": "google.com", "username": "brin"}],
                        "blog": blog
                })
        #on load handler via eventer
        def application_after_load(self, application):
```

```
print "app loaded"
#Blog.create_table()
#Entry.create_table()
```

1.3.5 views/templates/view.html

```
<html>
        <head>
                <title>{% block title %}{% endblock %}</title>
                <script src="/scripts/j.js"></script>
        </head>
        <body>
                <div>{{foo}}</div>
                <ul>
                {% for user in users %}
                  <li><a href="{{ user.url }}">{{ user.username }}</a></li>
                {% endfor %}
                </ul>
        </body>
</html>
```

1.4 Core

1.4.1 Application

class kiss.core.application.**Application**(*options*)
Main class of your application. Pass options to constructor and all subsystems(eventer, router, db_engine) will be configured.

1.4.2 Events

class kiss.core.events.**Eventer**(*mapping={}*)
Event dispatcher. You can subscribe to event or publish event.

1.5 Controllers

1.5.1 Events

1.5.2 Router

class kiss.controllers.router.**Router**(*options*)
Router implements unique hierarhical url mapping. Pass dictionary with mapping of regex and controller.

1.5.3 Core

class kiss.controllers.core.**Controller**
Base class of all controllers.

1.5.4 Page

class kiss.controllers.page.**PageController**(*page*, *context={}*)
If you need just to show page, create PageController and pass to it your page and optional context. Use it like another controllers in urls settings of your app.

1.5.5 Rest

class kiss.controllers.rest.**RestController**(*model*, *id_regex='(?P<id>\d+)'*)
Controller that creates REST API to your model. Pass model class to it and use url property and controller property in your urls settings.

1.5.6 Auth

1.6 Views

1.6.1 Core

class kiss.views.core.**JsonResponse**(*inp*, ***argw*)
Json response. Pass any object you want, JsonResponse converts it to json.

class kiss.views.core.**RedirectResponse**(*response=None*, *status=None*, *headers=None*, *mimetype=None*, *content_type=None*, *direct_passthrough=False*)
Response for redirect. Pass path and server will do 302 request.

class kiss.views.core.**Request**(*options*, ***argw*)
Base request object inhereted from werkzeug Request. Added session object.

class kiss.views.core.**Response**(*text*, ***argw*)
Base response object inhereted from werkzeug Response. Text/html mimetype is default.

1.6.2 Templates

class kiss.views.templates.**TemplateResponse**(*path*, *context={}*, ***argw*)
Template response via Jinja2. Pass template path and context.

1.6.3 Static files

class kiss.views.static.**StaticBuilder**(*path*)
Uses StaticCompiler to minify and compile js and css.

class kiss.views.static.**StaticCompiler**(*path*)
Static files minifier.

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

k