Caso de Estudio usando el patrón MVC con Google App Engine y Python

Trabajo final Ingeniería Web
2013
Juan Bertinetti

Objetivo

Presentar Google App Engine, mostrando de qué se trata, y más específicamente mostrar un caso de estudio implementado usando las herramientas que nos brinda.

Objetivo

Presentar Google App Engine, mostrando de qué se trata, y más específicamente mostrar un caso de estudio implementado usando las herramientas que nos brinda.

Usando MVC.

Google App Engine

¿Qué es?

Google App Engine

Es una plataforma que permite **alojar y ejecutar** nuestras aplicaciones web en la infraestructura de Google.

Nos brinda una serie de servicios y herramientas

aplicaciones fáciles de crear, de mantener y de ampliar

Servicios de Google App Engine

- Google Accounts
- Extracción de URL
- Correo
- Memcache
- Manipulación de imágenes
- Tareas programadas y colas de tareas
- Otros

Google App Engine

- SDK para Python (versión para Windows, para Mac OS X, y para Linux y otras plataformas)
- SDK para Java
- SDK para Go
- Versión experimental de SDK para PHP

Los SDK son gratuitos y libres.

Google App Engine

- SDK para Python (versión para Windows, para Mac OS X, y para Linux y otras plataformas)
- SDK para Java
- SDK para Go
- Versión experimental de SDK para PHP

Los SDK son gratuitos y libres.

Python

Mostrar un lenguaje alternativo para desarrollar aplicaciones web.

Últimamente es muy usado para este fin.

Es fácil de usar, sencillo, poderoso y entretenido.

Estructura de una aplicación

app.yaml:

```
application: nombre_de_la_aplicación
version: 1
runtime: python27
api_version: 1
threadsafe: yes

handlers:
- url: /.*
  script: modulo.application
```

Estructura de una aplicación

modulo.py:

```
from google.appengine.ext import webapp

class MainPage(webapp.RequestHandler):
    def get(self):
        self.response.out.write('Hola mundo!')

application = webapp.WSGIApplication([('/',
                                       MainPage)], debug=True)
```

webapp

- Framework sencillo para crear aplicaciones que viene con el kit de desarrollo de App Engine.
- Esconde los detalles de CGI.
- Se puede usar cualquier otro framework para Python: Django, CherryPy, Pylons, web.py, etc.

webapp - Java

- webapp.RequestHandler → HttpServlet

Elementos que nos provee:

- self.response → HttpServletResponse
- self.response.out → HttpServletResponse.getWriter()
- self.request → HttpServletRequest

Almacén de datos de App Engine

Base de datos de objetos sin esquema (no relacional) y distribuida.

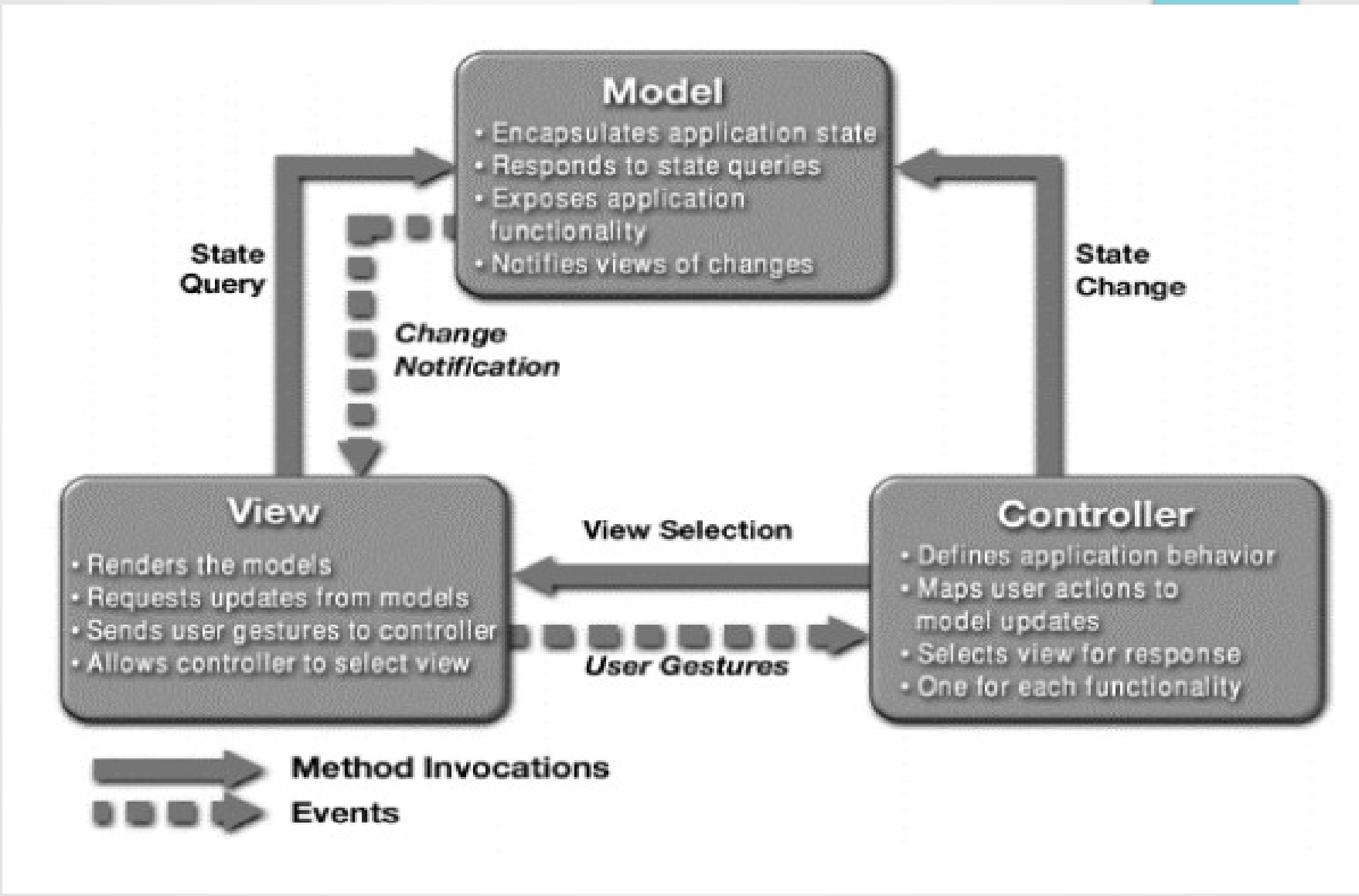
Nuestra aplicación trabaja con entidades.

Persiste, recupera, modifica y elimina a los objetos directamente.

Caso de estudio

Teniendo una lista desplegable con un listado de publicaciones guardadas, permitir seleccionar una y modificarle su precio.

Patrón Modelo-Vista-Controlador



El modelo

Será el encargado de todo lo relacionado con la base de datos.

A él le pedimos los datos y le brindamos la información nueva que debe guardar.

Entidad Publicación

```
class Publicacion(db.Model):
    pub_id = db.StringProperty(required=True)
    titulo = db.StringProperty(required=True)
    year = db.IntegerProperty()
    precio = db.FloatProperty(required=True)
    editorial = db.StringProperty()
    autor = db.ReferenceProperty(Autor)
```

Métodos del modelo:

- Un método que recupere todas las publicaciones guardadas y devuelva una lista con ellas.
- Un método que recupere y devuelva una publicación en particular para mostrar sus datos.
- Un método que modifique y guarde el precio de una publicación en particular.

Métodos del modelo:

Devuelve una lista con las publicaciones guardadas:

```
def listado_publicaciones():
    q = Publicacion.all()
    q.order("titulo")
    resultados = []
    for p in q:
        resultados.append(p)
    return resultados
```

Métodos del modelo:

Devuelve una publicación en particular según su ID de publicación:

```
def get_publicacion(pub_id):
    q = Publicacion.gql(
                    "WHERE pub_id = :1",
                    pub_id)
    return q.get()
```

Métodos del modelo:

Asigna el nuevo precio a una publicación en particular:

```
def modificar_precio(pub_id, nuevo_precio):
    p = get_publicacion(pub_id)
    p.precio = nuevo_precio
    p.put()
    return p
```

La vista

Son los archivos HTML en donde presentamos la información al usuario y donde obtenemos su entrada (nuevo valor para el precio).

La vista

App Engine usa plantillas para construir HTML dinámico según los datos de la aplicación.

Se puede usar cualquier sistema de plantillas para Python (EZT, Cheetah, ClearSilver, Quixote, Django, etc.).

webapp ya viene con el sistema de plantillas de Django incluido.

Plantillas

Listado de publicaciones en una lista desplegable:

```
<select name="pub_id_seleccionada">
    {% for p in publicaciones %}
        <option value="{{ p.pub_id }}">
            {{ p.titulo }}
        </option>
    {% endfor %}
</select>
```

El controlador

Es la conexión entre el modelo y la vista.

Recibe la entrada del usuario, invocando los métodos del modelo y transfiriendo los resultados a la vista.

Servicios del controlador:

- Servicio 1: devuelve la pantalla principal con el listado de publicaciones disponibles.
- Servicio 2: recibe el ID de una publicación (la seleccionada en el listado) y muestra el formulario que permite modificarle el precio, junto con información adicional.
- Servicio 3: recibe el nuevo precio ingresado por el usuario, y es quien le dice al modelo que persista este nuevo valor.

Servicios del controlador:

Cada servicio es una clase distinta que hereda de webapp.RequestHandler.

Como si fueran servlets distintas.

Servicio 1

```
class ListadoPrecios(
                              webapp.RequestHandler):
    def get(self):
        p = PubModel.listado_publicaciones()
        template_values = {"publicaciones": p}
        html = template.render(
                "view/listado.htm",
                 template_values)
        self.response.out.write(html)
```

Servicio 2

```
class Precio(webapp.RequestHandler):
    def post(self):
        pid = self.request.get(
                                "pub_id_seleccionada")
        p = PubModel.get_publicacion(pid)
        template_values = {"pub": p}
        html = template.render(
            "view/input_precio.htm",
            template_values)
        self.response.out.write(html)
```

Servicio 3

```
class Modificacion(webapp.RequestHandler):
    def post(self):
        pid = self.request.get("pub_id_actual")
        precio = float(self.request.get("precio"))
        p = PubModel.modificar_precio(pid,
                                      precio)
        template_values = {"pub": p}
        html = template.render(
                "view/precio_ok.htm",
                template_values)
        self.response.out.write(html)
```

Acceso a los servicios

```
application = webapp.WSGIApplication(
    [
        ("/s1", PubController.ListadoPrecios),
        ("/s2", PubController.Precio),
        ("/s3", PubController.Modificacion),
    ],
    debug=True)
```