INSTITUTO TECNOLÓGICO DE COLIMA

INGENIERIA EN SISTEMAS COMPUTACIONALES

MATERIA: SERVICIOS EN LA NUBE

- Examen Unidad 2: Hangman Game-

ALUMNO: Alejandro Pérez Toga

MCC. Patricia Elizabeth Figueroa Millán

Villa de Álvarez, Colima a Jueves 5 de Octubre de 2015

Examen Unidad *2: Hangman Game*

OBJETIVO

Como examen de la unidad II, se realizará la creación de del juego hangman (ahorcado) con el sdk de Google App Engine para después publicarlo para que se pueda agregar en la nube.

METODOLOGÍA UTILIZADA

1. Codificación del programa .py
2. Probar juego desde la terminal
3. Creación de la carpeta raiz del proyecto AppEngineProjects/Examen
4. Creación de las siguientes carpetas Examen/templates/ y Examen/statics/
5. Creación de las vistas que contendrá la aplicación dentro de la carpeta templates
6. Colocar los hojas de estilos dentro de la carpeta statics/css
7. Creación del archivo de configuración app.yaml donde se declaran los archivos html y la carpeta de los archivos css que se usarán dentro de la aplicación
8. Creación de la lógica de la aplicacion dentro del archivo examen.py
9. Correr la aplicación

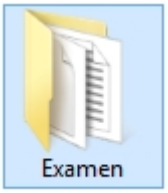
MATERIALES

- Computadora Sistema Operativo Ubuntu 14.04 LTS (Trusty)
- Python
- SDK Google App Engine
- Plantilla Jinja2 (incluida en SDK de google)
- Editor Atom
- Terminal de Ubuntu

DESARROLLO

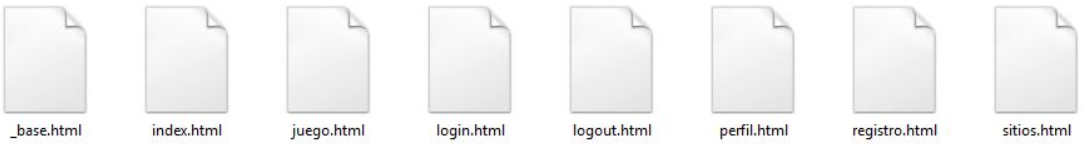
- Creación de la aplicación

1. Dentro de la carpeta AppEngineProjects se crea el directorio raíz de la aplicación con el nombre de Examen.



2. Dentro de Examen se crearán las carpetas de templates/ y de static/

3. En la carpeta de templates estarán los archivos html que manejaremos como vistas que serán manipuladas en el archivo examen.py y estarán declaradas en el archivo app.yaml



4. Colocamos los archivos css que utilicemos en el la carpeta static/css

5. Agregamos los archivos html y la carpeta de css al archivo de configuracion app.yaml

```yaml
application: examen-toga
version: 1
runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /fonts
  static_dir: static/fonts

- url: /css
  static_dir: static/css

- url: /js
  static_dir: static/js

- url: /img
  static_dir: static/img

- url: /.*
  script: examen.app

libraries:
- name: jinja2
  version: latest
```

6. El archivo index.html será la página principal de la aplicación, pedir y tendrá este código en el bloque de cuerpo:

Este Archivo hereda el cuerpo html del archivo base como se ha venido trabajando conforme a a la practica 3 de la unidad II.

```
{% block bodycontent %}
    <h2>Acerca del Sitio</h2>
    <p>
  <div>
    {% if user.user %}
      Bienvenido, {{user.user}}
    {% endif %}
  </div>
    Este Sitito dedicado para el aprendizaje
  de Google App Engine y Python. Esperamos sea de su agrado, pero sobre todo
  de gran utilidad.
  <a href="http://www.google.com.mx">Google</a>

    </p>
{% endblock %}
```

7. El archivo juego.html será la página con la dinámica del juego tendrá este código:

Esta vista recibirá desde la aplicación de python un 1 lista que contendrá las letras que se podrán utilizar en esta vista y por cada letra se hará un botón que enviara el valor de dicho botón para que se almacene en otra lista de las teclas que han sido seleccionadas para que se les agregue el valor disabled y no pueda volver a mandar esa letra.

Los mensajes se mandaran desde la clase que se encuentra en el archivo de examen .py y se mostrará en un clase que nos brinda bootstrap para manejar este tipo de mensajes.

Al final se va mostrando cada una de las letras que se han ido adivinando para que se pueda visulizar la continuidad del juego mediante una lista que va sustituyendo la letra correcta por guiones bajos y así se le va mandando a la vista y este pueda ser manipulado con un ciclo for, para poder agregar dicha letra en una etiqueta label.

```
{% block bodycontent %}
<h1 class="text-center">Tiempo de jugar al Ahorcado, {{user.user}}!!</h1>
<h2>Turnos: {{turns}}</h2>

<form action="/juego" method="post" class="text-center">
<h3>La Categoría es Animales (en Inglés)</h3>

  {%for letter in letters%}

        {% if letter in desac %}
         <button name="letra" class="btn btn-default" value="{{letter}}" disabled>{{letter}}</button>
        {% else %}
         <button name="letra" class="btn btn-default" value="{{letter}}">{{letter}}</button>
        {% endif %}


  {%endfor%}
  <div id="listas">

  </div>
</form>
{%if msg %}
<div class="alert alert-warning alert-dismissible" role="alert">
  <button type="button" class="close" data-dismiss="alert" aria-label="Close">
    <span aria-hidden="true">&times;</span></button>
  <strong> {{msg}}</strong>
</div>
{%endif%}
```

```
missed: {{missed}}<p>

<div id="hangmanImagen">
{%if img%}
<img src="{{img}}" alt="imagen">
{%endif%}


</div>
  <!-- {{secret}}</p> -->
  <hr>
  <hr>

  <div class="form-group text-center">
      {%for letra in animal%}
       <label for="" class="letrasSalida">{{letra}}</label>
    {%endfor%}
  </div>
<hr>
<hr>
  <!-- {{animal}} -->
<!-- <hr>
<p>{{letra}}</p>
<hr>
<p>{{guesses}}</p> -->


{% endblock %}
```

8. El archivo perfil.html mostrará la información del usuario, así como los juegos ganados y perdidos que a acumulado.

En esta vista se le pasa como parámetro toda la info relacionada con el usuario y simplemente se le da salida en las etiquetas correspondientes para que se pueda ver en la interfaz.

```
{% block bodycontent %}

<div class="container">

  <div class="col-xs-12 col-sm-12 col-md-6 col-lg-6 col-xs-offset-0 col-sm-offset-0 col-md-offset-3 col-lg-offset-3 toppad" >


         <div class="panel panel-info">
           <div class="panel-heading">
             <h3 class="panel-title">{{usuario.username}}</h3>
           </div>
           <div class="panel-body">
             <div class="row">
               <div class="col-md-3 col-lg-3 " align="center">
                 <img alt="User Pic" src="img/{{usuario.gender}}_user_icon.png"
                 class="img-circle img-responsive"> </div>

               <div class=" col-md-9 col-lg-9 ">
                 <table class="table table-user-information">
                   <tbody>
                     <tr>
                       <td>Fullname:</td>
                       <td>{{usuario.fullname}}</td>
                     </tr>
                       <tr>
                       <td>Gender</td>
                       <td>{{usuario.gender}}</td>
                     </tr>
                       <tr>
                       <td>Home Address</td>
                       <td>{{usuario.Address}}</td>
                     </tr>
                     <hr>
                     <tr>
                     <td>Wins</td>
                     <td>{{usuario.wins}}</td>
                     </tr>
                     <tr>
                     <td>Losses</td>
                     <td>{{usuario.losses}}</td>
                     </tr>
                   </tbody>
                  </table>


               </div>
             </div>
            </div>
                <div class="panel-footer">
                   <a data-original-title="Broadcast Message" data-toggle="tooltip" type="button" class="btn btn-sm btn-primary">
                       <i class="glyphicon glyphicon-envelope"></i></a>
                     <span class="pull-right">
                       <a href="edit.html" data-original-title="Edit this user" data-toggle="tooltip"
                        type="button" class="btn btn-sm btn-warning">
                         <i class="glyphicon glyphicon-edit"></i></a>
                       <a data-original-title="Remove this user" data-toggle="tooltip"
                       type="button" class="btn btn-sm btn-danger">
                         <i class="glyphicon glyphicon-remove"></i></a>
                     </span>
                   </div>

          </div>
        </div>
      </div>
    </div>
 {% endblock %}
```

9. Dentro de la carpeta Examen/ se edita el archivo *examen.py* con el siguiente código:

Se importan las librerías necesarias para poder utilizar algunos métodos que ocuparemos para la visualización del juego. Se utilizan las mismas funciones que nos permiten crear sesiones y para poder utilizar las paginas html junto con las plantillas de jinja2 como se vieron en practicas pasadas.

Se declaran algunas variables globales, las cuales nos servirán obtener la captura del botón presionado en la vista, la cantidad de letras que se encuentran si adivinar, los turnos, las letras que se mostrarán, los mensajes, la ruta de las imagines y la letra secreta.

```
# -*- encoding: utf-8 -*-
import os
import webapp2
import jinja2
import logging
import random
import urllib

from google.appengine.ext import ndb
from webapp2_extras import sessions

template_dir = os.path.join(os.path.dirname(__file__), 'templates')
jinja_env = jinja2.Environment(loader = jinja2.FileSystemLoader(template_dir),
                               autoescape = True)

winned= 0
lost = 0
guesses = ""
guess = ""
secret = ""
turns = 0
missed= 0
missed2=0
msg=""
letters = ['b', 'c', 'd','f', 'g', 'h',
       'j', 'k', 'l', 'm', 'n','p', 'q', 'r', 's',
       't','v', 'w', 'x', 'y', 'z']
desactivar=[]
animal=[]

img=""
```

Se crea una clase cuentas que tendrá los datos registrados de la vista registro y se almacenaran en el datastore para poder manipularla conforme a lo que ocupemos. La clase login renderizará la vista de la pagina login.html que contiene el usuario y la contraseña y los mandará como tipo post, donde la misma clase recibirá dichos datos y verificará que existan en el datastore para después crear la sesión.

```
#Describe las entidades del data Storex
class Cuentas(ndb.Model):
    username = ndb.StringProperty()
    password = ndb.StringProperty()
    fullname = ndb.StringProperty()
    Address   = ndb.StringProperty()
    gender   = ndb.StringProperty()
    wins     = ndb.IntegerProperty(default=0)
    losses     = ndb.IntegerProperty(default=0)

class Login(Handler):
    def get(self):
        self.render("login.html")

    def post(self):
        user = self.request.get('lg_username')
        pw = self.request.get('lg_password')

        logging.info('Checking user='+ str(user) + 'pw='+ str(pw))
        msg = ''
        if pw == '' or user == '' :
            msg = 'Please specify Account and Password'
            self.render("login.html", error=msg)
        else:
            consulta=Cuentas.query(ndb.AND(Cuentas.username==user, Cuentas.password==pw )).get()
            if consulta is not None:
                logging.info('POST consulta=' + str(consulta))
                #Vinculo el usuario obtenido de mi datastore con mi sesion.
                self.session['user'] = consulta.username
                logging.info("%s just logged in" % user)
                self.redirect('/')
            else:
                logging.info('POST consulta=' + str(consulta))
                msg = 'Incorrect user or password.. please try again'
                self.render("login.html", error=msg)
```

La clase registro obtendrá los datos de la vista y los almacenará en el datastore, con previa validación por javascript en el formulario para evitar mandar datos en blanco.

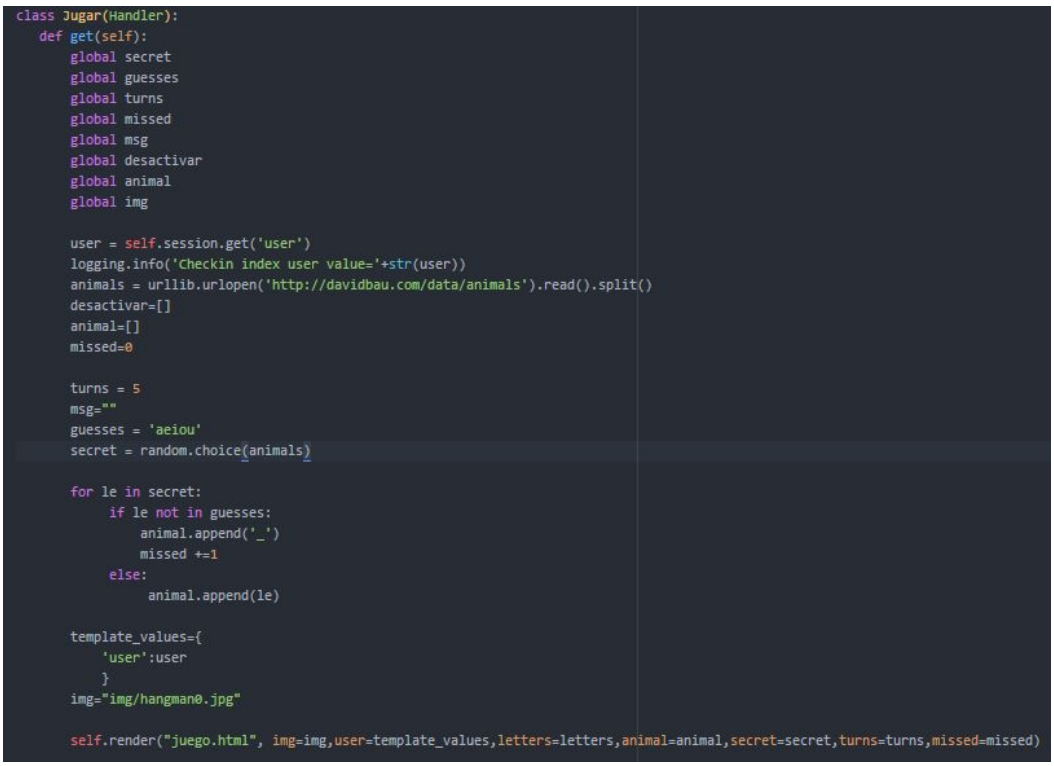
El index mostrará la pagina principal.

```
class Registro(Handler):
    def get(self):
            self.render("registro.html")

    def post(self):
        user= self.request.get('reg_username')
        pw=self.request.get('reg_password')
        fullname=self.request.get('reg_fullname')
        address=self.request.get('reg_address')
        gender=self.request.get('reg_gender')

        cuenta=Cuentas(username=user,password=pw,fullname=fullname,Address=address,gender=gender)
        cuentakey=cuenta.put()
        cuenta_user=cuentakey.get()

        if cuenta_user == cuenta:
            print "Cuenta de usuario: ",cuenta_user
            print
            msg= "Gracias Por Registarse..."
            self.render("registro.html",error=msg)

        query = Cuentas.query()
        for resultado in query:
            print "Resultado.username: ",resultado.username
            print "resultado: ",resultado



class Index(Handler):
    def get(self):
        user = self.session.get('user')
        logging.info('Checkin index user value='+str(user))
        template_values={
            'user':user
            }
        self.render("index.html", user=template_values)
```

La clase jugar lleva el control del juego es donde se utilizan las mayorías de las variables

En el método get se inicializan las variables y se pasan a la vista para que se puedan seguir manipulando en el método post.

Aquí se obtiene una lista desde la url que contiene los nombres de varios animales , para después aplicar un método random para que escoja un animal, el ciclo for con la variable accedemos a la lista que contiene al animal secreto para cambiar todas letras que no sean las vocales por guiones bajos y cuenta el numero de guiones bajos para determinar el numero de letras que hacen falta por descubrir.



```
class Jugar(Handler):
   def get(self):
       global secret
       global guesses
       global turns
       global missed
       global msg
       global desactivar
       global animal
       global img

       user = self.session.get('user')
       logging.info('Checkin index user value='+str(user))
       animals = urllib.urlopen('http://davidbau.com/data/animals').read().split()
       desactivar=[]
       animal=[]
       missed=0

       turns = 5
       msg=""
       guesses = 'aeiou'
       secret = random.choice(animals)

       for le in secret:
            if le not in guesses:
                animal.append('_')
                missed +=1
            else:
                 animal.append(le)

       template_values={
           'user':user
           }
       img="img/hangman0.jpg"

       self.render("juego.html", img=img,user=template_values,letters=letters,animal=animal,secret=secret,turns=turns,missed=missed)
```

El método post lo que hace es mandar la letra a que se revise si esta en la palabra secreta, si lo esta se descubrirá dicha letra y se mostrará de lo contrario se restará un turno y se mandara la dirección de la imagen correspondiente.

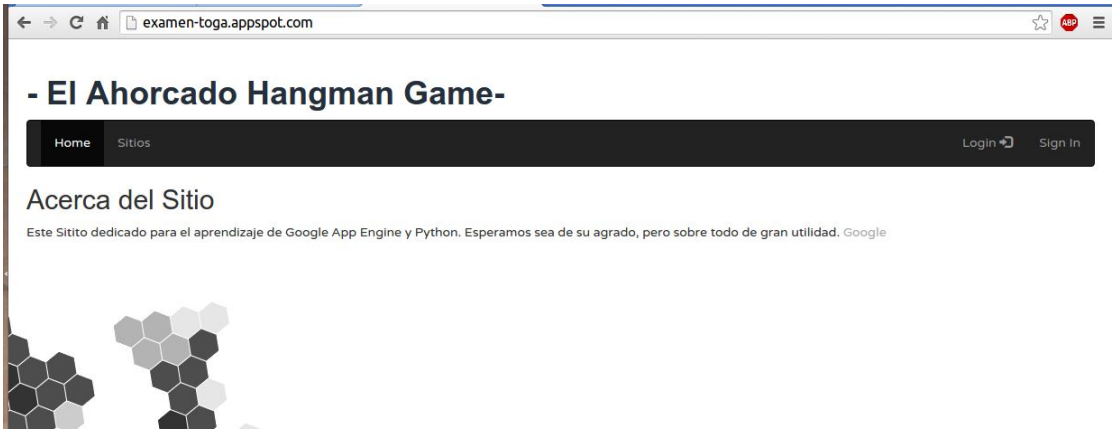
```
def post(self):
    global turns
    global guess
    global guesses
    global secret
    global missed
    global msg
    global letters
    global desactivar
    global winned
    global lost
    global img

    missed = 0
    msg=""
    user = self.session.get('user')
    letra= self.request.get('letra')
    letra=str(letra)
    desactivar.append(letra)
    guesses += letra
    index=0

    for le in secret:
         if le not in guesses:
             missed +=1
         else:
              animal[index]=le
         index+=1
  if missed == 0:
       msg="Ganasteeeee!!!"
       desactivar=letters
       winned += 1
       query = Cuentas.query(Cuentas.username == user).get()
       if query is not None:
           query.wins += 1
           query.put()
       # if consulta is not None:
       #     consulta.winned = wined
       #     consulta.put()

  if letra not in secret:
      turns -=1
      msg = "No es la letra correcta."
      if turns < 5: img="img/hangman1.jpg"
      if turns < 4: img="img/hangman2.jpg"
      if turns < 3: img="img/hangman3.jpg"
      if turns < 2: img="img/hangman4.jpg"
      if turns < 1: img="img/hangman5.jpg"
      if turns == 0:
          msg='La respuesta correcta es..'+str(secret)
          desactivar=letters
          lost += 1
          query = Cuentas.query(Cuentas.username == user).get()
          if query is not None:
              query.losses += 1
              query.put()
       #    if consulta is not None:
       #        consulta.losses = lost
       #        consulta.put()

  template_values={
      'user':user
   }
  self.render("juego.html", img=img,animal=animal,missed=missed,desac=desactivar,user=template_values,letters=letters,
    secret=secret,letra=letra,guesses=guesses,msg=msg,turns=turns)
```

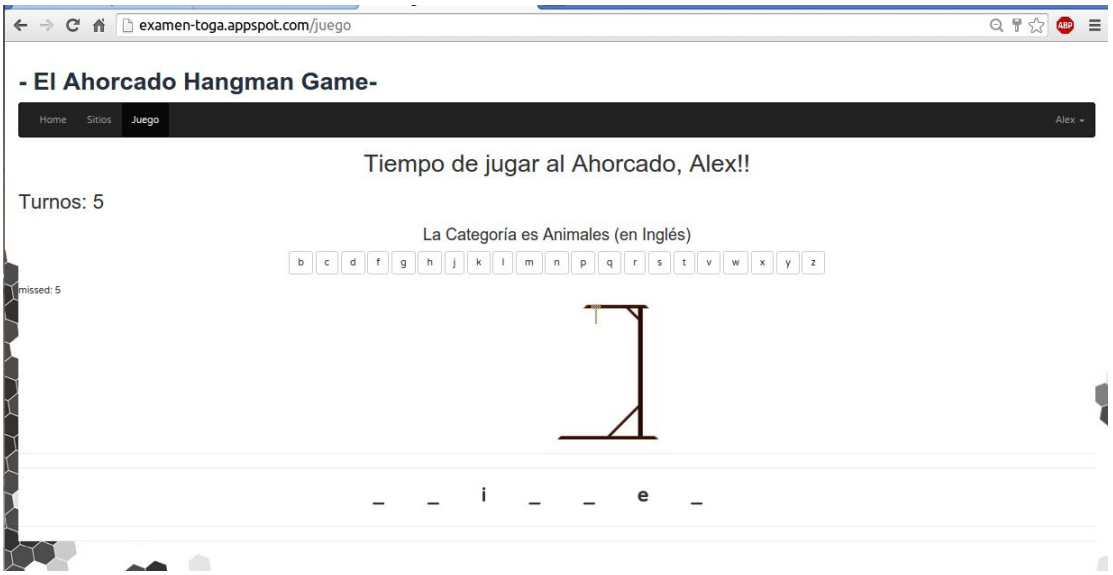
La clase profile hace la consulta para obtener los campos del usuario que inicio sesión y lo manda a la vista de profile.html

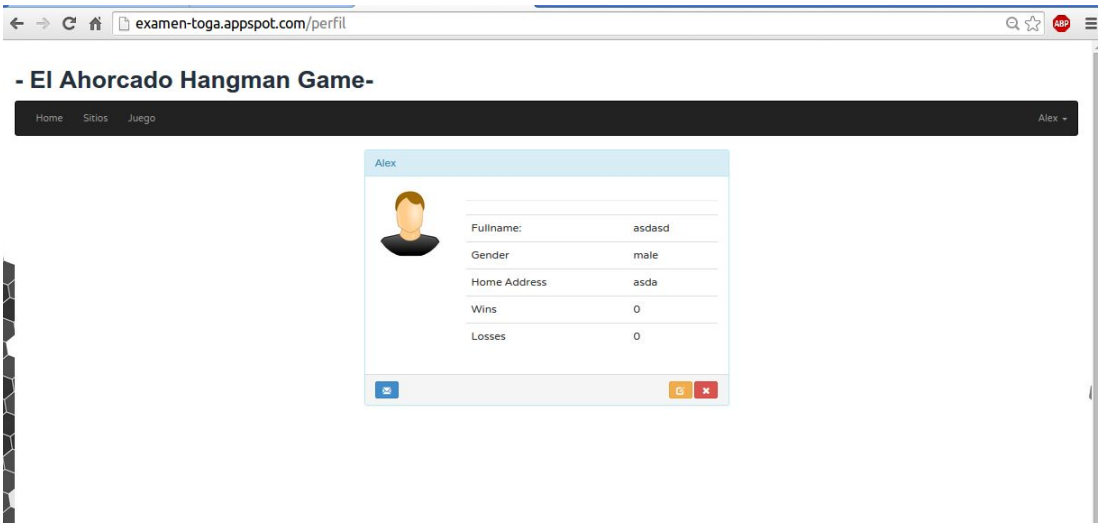
Logout destruye la sesión y redirecciona al index.

```
class Profile(Handler):
   def get(self):
       user = self.session.get('user')
       query = Cuentas.query(Cuentas.username == user).get()
       print query
       logging.info('Checkin index user value='+str(user))
       template_values={
           'user':user
           }
       self.render("perfil.html", user=template_values,usuario=query)

class Logout(Handler):
    def get(self):
        if self.session.get('user'):
            msg = 'You are loging out..'
            self.render("logout.html", error=msg)
            del self.session['user']
            self.redirect('/')

config = {}
config['webapp2_extras.sessions'] = {
    'secret_key': 'some-secret-key',
}

app = webapp2.WSGIApplication([('/', Index),
                               ('/juego',Jugar),
                               ('/registro',Registro),
                               ('/sitios',Sitios    ),
                               ('/login',Login),
                               ('/logout',Logout),
                               ('/perfil',Profile)
                              ],
                              debug=True, config=config)
```

RESULTADO


examen-toga.appspot.com
- El Ahorcado Hangman Game-
Home
Sitios
Login
Sign In
Acerca del Sitio
Este Sitito dedicado para el aprendizaje de Google App Engine y Python. Esperamos sea de su agrado, pero sobre todo de gran utilidad. Google


examen-toga.appspot.com/juego
- El Ahorcado Hangman Game-
Home
Sitios
Juego
Alex
Tiempo de jugar al Ahorcado, Alex!!
Turnos: 5
La Categoría es Animales (en Inglés)
b c d f g h j k l m n p q r s t v w x y z
missed: 5
_ _ i _ _ e _


examen-toga.appspot.com/perfil
- El Ahorcado Hangman Game-
Home
Sitios
Juego
Alex
Alex
Fullname: asdasd
Gender male
Home Address asda
Wins 0
Losses 0

CONCLUSIONES

En esta practica el uso de las listas fue importante ya que en estas se podía acceder y cambiar los valores.

El datastore de google nos permite almacenar datos de una menera diferente a la que utilizaramos una base de datos relacional ya que el datastore son objetos.