Universidad Autónoma de Madrid

Departamento de Informática

Proyecto de Sistemas Informáticos
Práctica - 4

Roberto Marabini

Índice

1. Objetivos

En esta práctica se acabará de implementar el sistema de elección de grupo de laboratorio. Las principales modificaciones serán: (1) la introducción de la posibilidad de "disolver" parejas y (2) una mejora de la estructura de las páginas web. Por ejemplo, os pediremos que uséis ficheros de estilo `css` y definái una plantilla base que luego extenderemos en cada página del sitio web.

2. Trabajo a realizar durante la primera semana

El trabajo a realizar en esta práctica parte del código desarrollado en la práctica anterior así que se recomienda que sigas usando el mismo repositorio privado que usaste en la practica 3. Si por algún motivo decides crear uno nuevo añade a tu profesor al repositorio. . En la implementación de la práctica 3 nos quedó pendiente incluir la posibilidad de que las parejas se disuelvan o deshagan y los miembros de las mismas pasen a formar parejas nuevas.

2.1. Disolución de parejas, requisitos a implementar

- Siempre que los miembros de la pareja no hayan seleccionado un grupo debe ser posible disolver una pareja.
- Si la pareja no está validada bastará con que cualquiera de sus miembros solicite su disolución. Tras la solicitud la pareja será borrada de la base de datos.
- Si la pareja esta validada ambos usuarios deben solicitar su borrado. Cuando cualquiera de los miembros de la pareja validada solicite la disolución de la misma el sistema debe comprobar el valor de la variable `Pair.studentBreakRequest`. Si está variable esta vacía se almacenará en la misma al usuario que realiza la petición. Si esta llena se procederá a borrar la pareja siempre y cuando el usuario almacenado en `Pair.studentBreakRequest` sea distinto al usuario que realiza la petición. Esto es, ambos miembros de la pareja deben solicitar la disolución no uno dos veces.

- Obviamente solo los miembros de la pareja pueden solicitar el borrado de la misma.

En resumen, se solicita la implementación del servicio:

- **breakpair:** Solicitar la disolución de una pareja.
  Entrada: Mediante un formulario se ofrecerán al usuario todos las parejas de las que forma parte tanto como `student1` como como `student2`
  Resultado: Borrado de la pajera (`Pair`) seleccionado
  Accesibilidad: Solo puede ser invocado por usuarios previamente autenticados. Implementad este requisito usando el sistema de verificación de *Django*. Un usuario dado sólo puede solicitar la disolución de una pareja de la que él sea uno de los miembros.

2.2. Tests

Para verificar la correcta implementación de los nuevos requerimientos hemos usad la clase `BreakPairServiceTests` definida en `tests_services.py`.

Una vez que hayáis conseguido que todos los test (tanto los nuevos como los introducidos en la práctica 3) se ejecuten de forma satisfactoria verificad la cobertura:

```
coverage erase
coverage run --omit="*/test*" --source=core ./manage.py test core
coverage report -m -i
```

en caso de que el fichero `views.py` no tenga una cobertura cercana al 100 % añadid al fichero de tests los test que hagan falta para alcanzar este valor.

3. Trabajo a realizar durante la segunda semana

Esta semana nos vamos a limitar a modificar el frontal general de la aplicación para hacerla atractiva y mejorar su estructura.

Con pequeñas variaciones la mayor parte de vosotros habrá creado una página web similar estructuralmente a la existente en la Figura 1 donde se aprecian varias áreas:

cabecero (header), pie (footer), menú de navegación, etc que tienden a ser siempre idénticas para todas las páginas de la aplicación y una zona central (content) donde se producen la mayor parte de los cambios.

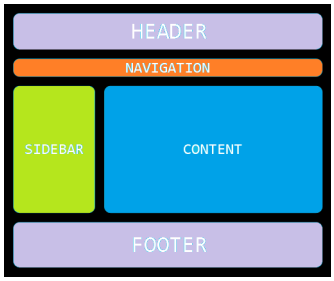

Figura 1: Estructura típica de una página web.

Si no habéis diseñado vuestras vuestras *templates* para que usen el mecanismo de herencia implementado en *Django* hacedlo. Esto es, debe existir una *template* principal (pongamos que se llama `base.html`) de la que heredarán el resto de las *templates* usando la directiva de *Django* {`% extends "base.html" %`}. Dentro de la *template* base se crearán distintos bloques (con la directiva {`% block xxx %`} {`% endblock %`}) que serán substituidos por las distintas *templates* hijas. Igualmente modificar vuestros formularios si no habéis usado la infraestructura de *Django* para crearlos.

Queda a vuestra discreción decidir el diseño y la estructura de vuestras páginas con las restricciones de que el resultado: (1) no debe parecerse visualmente a la aplicación que os damos como referencia en `https://desolate-sands-30591.herokuapp.com/`, (2) debe ser estéticamente gratas y funcionalmente ágiles y (3) el sitio web debe ser autocontenido, explicándose su propósito y funcionamiento a los usuarios. Por favor describir el funcionamiento usando vuestras palabras y no copiando párrafos de la aplicación que os ofrecemos como referencia.

Finalmente se espera que las decisiones de estilo estén contenidas en un fichero css y no en el código html de las *templates*.

4. Trabajo a presentar al finalizar la práctica

- Aseguraros que vuestro código satisface **todos** los tests proporcionados en `tests_models.py`, `tests_services.py` (ambos proporcionados en la práctica anterior). No es admisible que se modifique el código de los tests excepto las variables situadas entre las lineas `# You may modify the following variables` y `# Please do not modify anything below this line`.
- Implementar todos los tests que consideres necesarios para cubrir totalmente la funcionalidad desarrollada. Estos tests se deben incluir en clases dentro de un fichero llamado `tests_additional_P4.py`.
- Incluir en la raíz del proyecto un fichero llamado coverage.txt que contenga el resultado de ejecutar el comando `coverage erase; coverage run --omit="*/test*-source=core ./manage.py test core ; coverage report -m -i`.
- Desplegar, poblar y probar la aplicación en *Heroku*.
- Incluir un **breve** manual de usuario de la aplicación de no más de 5 páginas en formato pdf. El manual contendrá las instrucciones adecuadas para que los usuarios de vuestra aplicación puedan utilizarla. No se os pide un manual para el desarrollador en el cual se describa la arquitectura de la aplicación, ni una descripción sobre la arquitectura de *Django*, ni una valoración sobre la dificultad de implementar los diferentes requisitos.
- Subir a *Moodle* el fichero obtenido al ejecutar el comando `zip -r ../assign4_final.zip .git` desde la raíz del proyecto. Recordad que hay que añadir y "comitir" los ficheros a git antes de ejecutar el comando. Si queréis comprobar que el contenido del fichero zip es correcto lo podéis hacer ejecutando la orden: `cd ..; unzip assign4_final.zip; git clone . tmpDir; ls tmpDir`. En este punto es importante asegurarse que la variable `ALLOWED_HOSTS` del fichero `settings.py` incluido en la entrega contiene tu dirección de despliegue en *Heroku* (si no aparece corregiremos la práctica como si el proyecto no estuviera desplegado en *Heroku*). Asimismo, comprobar que tanto el nombre de usuario como la clave del usuario de administración son *alumnodb*.

5. Criterios de evaluación

Para aprobar con 5 puntos es necesario satisfacer en su totalidad los siguientes criterios:

- Todos los ficheros necesarios para ejecutar la aplicación se han entregado a tiempo.
- El código se ha guardado en un repository git y este repositorio es privado.
- Es posible romper parejas tal y como se describe en el enunciado. Los usuarios implicados deben poder crear parejas nuevas tras la rotura.
- Al ejecutar en local los tests contenidos en los ficheros `test_models.py` y `test_services.py` el número de fallos no es superior a seis y el código que los satisface es funcional.
- Resulta imposible suplantar a un usuario sin conocer su nombre de usuario y clave. Por ejemplo realizando una llamada directamente a un método usando `curl` o similar.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 5.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- La aplicación está desplegada en *Heroku*. En el fichero `settings.py` se encuentra añadida la dirección de *Heroku* a la variable `ALLOWED_HOSTS`. Además de estar desplegada, la aplicación funciona correctamente en *Heroku*.
- La aplicación de administración de base de datos (admin/) se ha desplegado y está accesible en *Heroku* usando el username/password *alumnodb*.
- Usando la aplicación de administración es posible crear o borrar objetos pertenecientes a todos los modelos solicitados.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 6.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- Se utiliza correctamente el sistema de autentificación de usuarios de *Django*.
- Se ha usado herencia a la hora de crear las paginas web
- Los formularios se construyen usando el sistema de formulario implementado por *Django*.
- Se han usado ficheros css para definir los estilos
- Al ejecutar en local los tests contenidos en los ficheros `test_models.py` y `test_services.py` el número de fallos no es superior a cuatro y el código que los satisface es funcional.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 7.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- El código es legible, eficiente, está bien estructurado y comentado.
- Se utilizan las herramientas que proporciona el framework.
- Sirva como ejemplo de los puntos anteriores:
  - Las búsquedas las realiza la base de datos no se cargan todos los elementos de una tabla y se busca en las funciones definidas en `views.py`.
  - Los errores se procesan adecuadamente y se devuelven mensajes de error comprensibles.
  - El código presenta un estilo consistente y las funciones están comentadas incluyendo su autor. Nota: el autor de una función debe ser único.
  - Se es coherente con los criterios de estilos marcados por *Flake8*. *Flake8* no devuelve ningún error al ejecutarse sobre las líneas de código programadas por el estudiante.

- Se ha incluido el manual de usuario y es correcto.
- La aplicación es estéticamente atrayente.
- El propósito y el funcionamiento del sitio web quedan claramente explicados en las paginas creadas por la aplicación de forma que un usuario sin conocimientos previos sobre su uso es capaz de elegir su grupo de prácticas.
- Al ejecutar en local los tests contenidos en los ficheros `test_models.py` y `test_services.py` el número de fallos no es superior a dos y el código que los satisface es funcional.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 8.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- La aplicacion es robusta y responde adecuadamente incluso si se proporcionan parametros invalidos.
- Todos los tests y todas las pruebas ejecutadas dan resultados satisfactorios.
- Si reducimos el tamaño de la ventana del navegador o utilizamos el zoom, todos los elementos de la página siguen resultando accesibles y no se pierde funcionalidad.

Para optar a la nota máxima se debe cumplir lo siguientes criterios:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- Se reporta la cobertura (programa coverage) obtenida por los tests antes y después de añadir tus nuevos tests. La cobertura debe incrementarse.
- La cobertura para los ficheros que contienen los modelos, las vistas y los formularios es superior al 99 %.

Nota: Entrega final fuera de plazo → substraer un punto por cada día (o fracción) de retraso en la entrega.
Nota: El código usado en la corrección de la práctica será el entregado en *Moodle*. Bajo ningún concepto se usará el código existente en *Heroku*, *Github* o cualquier otro repositorio.