Software Distribuït

Pràctica 2 – App Web

Blai Ras i Albert Morales

Introducció

En la última pràctica del curs se'ns proposa desenvolupar una aplicació web amb Django. Concretament, anirem realitzant iteracions en un projecte d'una pàgina web de mobles i objectes per la llar en correspondència amb la teoria vista a classe.

En una primera iteració, ens centrarem amb Django, quines eines ens dona per gestionar una aplicació web i realitzarem els primers items d'aquesta.

En segon lloc, desenvoluparem el carro de la compra dels nostres usuaris i veurem com es tracta la nostre informació a la base de dades.

En una tercera iteració realitzarem el *login* i *logout* dels usuaris a part de fer maca la web amb CSS.

Finalment, aplicarem RESTFul i farem el Deployment del projecte amb Heroku.

Objectius

- Treballar les aplicacions web vistes a classe amb un projecte d'una pàgina web de mobles i l'ajuda de Django
- Entendre la gestió d'usuaris d'una aplicació web i el tractament de la seva informació
- Familiaritzar-nos amb HTML, una mica de Javascript i CSS.
- Entendre l'aplicació de RESTFul a una aplicació web.
- Realitzar el Deployment d'una aplicació web.

A continuació destaco la els mètodes de la classe *Views.py* on s'ha fet la majoria de codi d'aquest projecte:

- Index: mètode que gestiona la pàgina principal de Ykea. En ella, he decidit mostrar les categories dels nostres items. Per a fer-ho, simplement hi accedim gràcies al *import* de la classe Item que posseeix l'atribut *CATEGORIES*. Realitzem una llista i la passem amb *context*.

  La plana principal també ha d'oferir al usuari registrar-se, iniciar sessió o sortir. Per tant, aquest mètode mira si l'usuari actual està registrat. Si així es, mostra el seu nom, els seus diners i la possibilitat de fer *logout*. Si no, ofereix registar-se o iniciar sessió.

- Items: aquest mètode no ha estat modificat respecte al del enunciat amb excepció de la possibilitat d'iniciar sessió, registrar-se o sortir.
- findItem: aquest mètode s'encarrega de mostrar la informació detallada d'un item. Per tant, té un *try-catch* on a dins del *try* crea una variable *item* amb la informació passada per paràmetre i salta el *catch* si no ha pogut trobar l'item. El tipus d'excepció l'he creat jo amb una classe anomenada *ItemNotFound*, la qual simplement mostra un missatge d'error.

- Shoppingcart: aquest mètode gestiona la creació o recuperació del carro de la compra. Cada usuari té el seu propi carro, per tant, el primer que hem de fer és obtenir el carro d'aquest. Què passa si l'usuari no està registrat? No podem trobar el seu carro! Doncs aquest mètode va protegit amb un *@login_required*, una instrucció que redirigeix al usuari no logat cap a un camp especificat en el meu fitxer *Settings.py* anomenat LOGIN_URL. L'he hagut d'especificar perquè el meu registre es realitza a la *main page*, no pas a la URL *accounts/login* (URL cap on redirigeix el mètode *@login_required* per defecte).

  Bé, per agafar el carro utilitzem un get passant l'usuari com a paràmetre. Si no existeix "usuari", el creem i l'hi afegim un carro nou. Finalment, afegim els productes que l'usuari ha marcat amb la *checkbox*, mirem la seva quantitat i redirigim la pàgina cap a *buy*.
- Buy: aquest mètode gestiona la compra d'un carro. El primer que fem és agafar l'usuari de la *request*. En segon lloc poden passar dos coses: que l'usuari decideixi eliminar un ítem o que l'usuari decideixi procedir a comprar el carro.

  En la primera opció simplement eliminem una unitat del producte del carro. En la segona opció, hem d'agafar els ítem que ha seleccionat, la seva quantitat i calcular el preu final. Aquesta informació es passa una *template* diferent, anomenada *checkout*, on hi ha el botó de comprar. Al clicar aquest botó, es procedeix a restar el total dels diners de l'usuari i finalment redirigeix a la persona cap a la pàgina principal. Aquest botó no apareix si l'usuari no t'he prou diners, sinó que es mostra un *warning* avisant.
- Logout_view: tal i com el seu nom indica, realitza la desautenticació del usuari en qüestió. Està protegit amb un *@login_required*, de manera que mai hi podrem arribar si no hem iniciat sessió. Així doncs, fa el *logout* de l'usuari i mostra un missatge de confirmació. Al final, et dona la possibilitat de tornar a la pàgina principal.
- Register: mètode que realitza el registre d'un nou usuari dins d'Ykea. La creació d'un nou usuari amb Django és bastant senzilla, és a dir, només hem hagut de modificar respecte al vostre codi dos línies: una per crear el carro amb l'usuari just creat i una altre per crear l'usuari (de tipus Client) amb el carro i l'usuari just acabats de crear. Si hi ha algun error al formulari et redirigeix cap a la mateixa *view*. Si el registre és correcte, es torna a la pàgina principal on t'hi has de registrar.
- ItemViewSet: mètode que gestiona les entrades de la API pels items d'Ykea. La primera funció que trobem és *get_queryset*. Tal i com el seu nom indica, retorna una *query* segons la *request* que li entri. Ens diuen que pot haver-hi 3 possibilitats a filtrar: el preu, la categoria o si "és nou".

  Per retornar els items que compleixen un o més d'aquets paràmetres, simplement realitzarem un *filter*, tot seguit després d'haver obtingut el valor en qüestió de la *request* amb un *get.*
- Comparator: mètode que redirecciona al usuari cap al comparador de la API. Per paràmetre, es passa la llista d'IP's.

Models.py

En aquesta classe hi ha representada la definició d'un ítem (donada per l'enunciat), la definició d'un carro de la compra, la seva classe intermediària *ManyToMany* i la classe Client.

Shoppingcart té 3 atributs: un usuari, un seguit d'Items i la quantitat de diners que val aquest carro en un moment determinat. La seva classe intermediària té dos atributs: l'ítem en qüestió i la quantitat d'aquest.

Client té 3 atributs també. Un usuari (*user*) per saber de qui estem parlant, el seu corresponent carro (*Shoppingcart*) i els diners d'aquest. Per defecte, en té 2000.

CSS

El nostre projecte té una mica de CSS tret de *Bootstrap.com*. S'han usat "caixes" per representar un ítem, *breadcrumb*s per fer la barra de navegació de cada plana, barres de navegació, missatges d'error personalitzats, botons...

Deployment i sessió de test

El Deployment es va realitzar el dia 29 de maig a Heroku. Es va crear un repositori anomenat YkeaWebsite. En aquell moment, cap part de "deures" de RESTFul estava implementada.

Per tant, el dia de la sessió de test es va realitzar sense aquesta part. Les dos hores van anar força bé. Es va realitzar una fulla de càlcul amb els resultats que pots trobar adjuntada amb aquest projecte. Per part nostre, el *feedback* va ser el següent:

Grup	F04
Disseny i usabilitat	*(No ens posarem nota nosaltres mateixos)*
Registrar un usuari	Sí
Iniciar sessió amb un usuari existent	Sí
Límit de diners?	Sí, però en aquell moment no es visualitzava
Realitzar la compra d'un ítem	Sí
Es pot comprar un ítem sense tenir els diners necessaris?	Sí
Històric de factures?	No
Esborrar les *cookies*. Iniciar sessió un altre cop. Tot correcte?	Sí
WebServicie, buscar cert ítem	No implementat en aquell moment
Comprar ítem mes barat	No implementat en aquell moment
Observacions	1. Si seleccionàvem un ítem per comprar sense estar registrats, l'aplicació "petava". Això era degut a que el nostre paràmetre LOGIN_URL de *Settings.py* no es va canviar al realitzar el Deployment, i per tant, encara era "*localhost:...*" i no "*sd-2018...*". 2. En segon lloc, hi havia un error similar amb el *Sign In* a les *templates* de ítem i *itemInfo*. El paràmetre *href* que fa de *link* cap a la URL de *login* era incorrecte. Estava a "register" (com a la *mainPage*) i havia de ser "/ykea/register". 3. Finalment, també se'm va comentar que només mostràvem la quantitat de diners a l'hora de fer el *checkout*. Clar, després de comprar un ítem, hauríem de veure la quantitat de diners que ens queden a la pàgina principal (ara ja es mostra).

Mapa Web

El següent diagrama mostra el mapa web de la nostre aplicació principal. He decidit col·locar com a primera URL la de "Heroku", és a dir, *https://sd2018-ykeaf4.herokuapp.com* , però en veritat el *home* és la URL anterior més /ykea.

Per tant, divideixo aquesta URL en */ykea, /admin, /api* i */accounts*.

- A */ykea/items* he col·locat totes les categories i llavors el diagrama queda una mica estrany.
- En canvi, a */ykea/item* he decidit posar el nom *item_number* en comptes de tots els números de cada item de la app.
- A */accounts* només hi he posat *logout* ja que nosaltres no tenim */accounts/login*. El *login* es fa des de la *mainpage* o des de qualsevol altre pantalla.
- A */api/items* he posat els tres paràmetres que et poden pasar (preu, si és nou i la categoria) per separat, però ja sabem que et poden donar una URL que contingui tots aquets 3 camps.

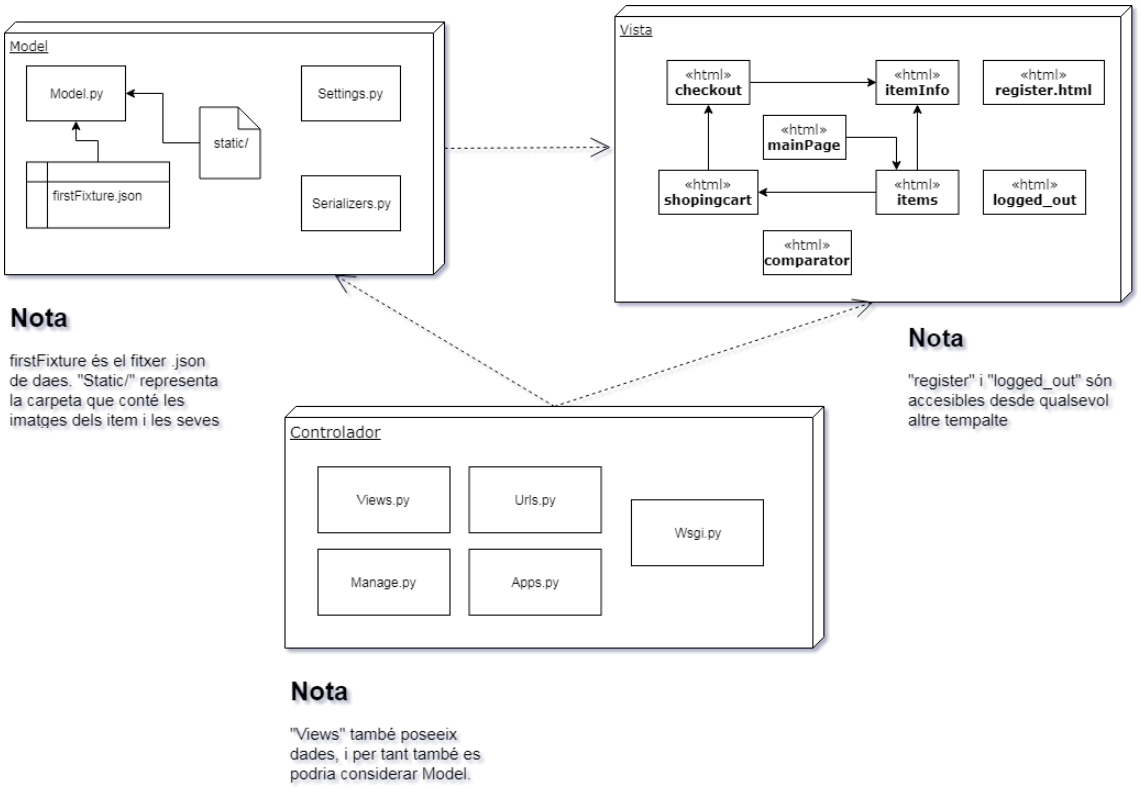
Pots veure en format *.png* i *.xml* a la carpeta d'aquest mateix *.PDF*. No el poso aquí perquè és massa gran!

Diagrama de classes

Finalment, he realitzat el diagrama de classes d'Ykea. Ens demanaven que s'interpretés com a Model-Vista-Controlador. A comentar...:

- La Vista he considerat que eren tots els *templates* en *html* que posseeix l'aplicació web
- El Controlador per mi és principalment *views.py*, ja que redirigeix i indica cap on s'ha d'anar segons quina acció faci l'usuari. De totes maneres, també te dades, així que no crec que sigui incorrecte situar-lo a Model. La classe que sens dubte pertany a Controlador és *urls.py*, ja sigui de PracticaWeb com d'Ykea. *Apps.py* i *Manage.py* són més dubtoses però no crec que encaixin enlloc més.
- A Model, les dades, he col·locat Model.py (redundant) i el fitxer de *fixtures* on estàn definits els items. També hi he col·locat, per tant, la carpeta *static* que conté les imatges i les instruccions del ítems i de l'aplicació web en sí. Amb *Settings.py* també vaig dubtar. Per una part, té estructura de "diccionari" i per tant pots pensar que es més Controlador, però com que allà defineix diferents paràmetres de l'aplicació, potser encaixa més en Model.

Pots veure el diagrama a continuació. De totes maneres, també el pots veure en format *.png* a dins d'aquest projecte.



Conclusions

Hem realitzat una aplicació web amb Django i aplicant els coneixements explicats a teoria. Concretament, hem vist com es tracta la informació per usuari, l'estructura de la base de dades, una mica de *html* i *javascript* i finalment hem aplicat serveis web RESTFul on hem vist com es gestiona una API + Ajax.

La part que més m'ha agradat ha sigut la de *html* i CSS. La part que ens ha portat més hores és la part dos d'aquesta pràctica, a l'hora de realitzar les classes *shoppingcart*, etc. No per la dificultat, sinó per entendre-ho tot. La part final de RESTFul, sobretot el punt 3, també ens ha portat força hores. De fet, la part de comparador m'han hagut d'ajudar bastant, sobretot perquè no vaig poder anar a la classes d'Ajax.