Pràctica 2: Aplicacions web

Interfícies d'usuari — Enginyeria de Sistemes TIC

Sebastià Vila-Marta Aleix Llusà Serra

14 d'abril de 2023

Índex

Resum

Llenguatges web: HTML, CSS, JS
Llenguatges de plantilles: TAL, METAL
Framework web: Python/aiohttp

1 Organització

Aquesta pràctica segueix a l'anterior i dissenyarem la part del servidor de l'aplicació i la connexió des de la interfície web.

En aquesta segona part de la pràctica, podeu crear un projecte d'aiohttp tot seguint el tutorial bàsic a `http://demos.aiohttp.org/en/latest`. És important estructurar el codi en diversos

directoris.

També podeu escollir un web framework diferent i aplicar-hi els mateixos passos d'aquesta pràctica, adaptant el codi allà on calgui.

1.1 Condicions

- La pràctica està calibrada per a ésser treballada en equips de dues persones.
- La durada de la pràctica és de 5 setmanes.

2 aiohttp

Aiohttp és un client/servidor asíncron que es programa en Python. En aquest apartat primer seguim el tutorial bàsic d'aiohttp i després programem una aplicació per a renderitzar les plantilles TAL anteriors.

Podeu trobar la documentació sobre aiohttp servidor a `https://docs.aiohttp.org/en/stable/web.html` i una referència ràpida a`https://docs.aiohttp.org/en/stable/web_quickstart.html`.

> TASCA PRÈVIA 1 No continueu als punts següents fins que pugueu visitar una pàgina com per exemple `http://localhost:8080/bondia/itic` i obtingueu resposta.
>
> Podeu seguir el tutorial d'aiohttp a `http://demos.aiohttp.org/`.
>
> També podeu instal · lar les eines de desenvolupadors d'aiohttp a
>
> `https://create-aio-app.readthedocs.io/`. En aquest cas, us serà més senzill començar sense base de dades:
>
> ```
> create-aio-app --without-postgres iu
> pip install -r iu/requirements/development.txt
> cd iu
> make adev
> ```

2.0.1 La primera plantilla

Referència principal: `https://aiohttp-tal.readthedocs.io`

Abans d'implementar les vostres plantilles, us cal entendre l'estructura del paquet d'aiohttp per a localitzar l'organització del codi. Podeu trobar un exemple ja fet amb Jinja2 a `iu/templates/index.html`.

Us proposem el programa Python següent per a renderitzar la plantilla de la pràctica 1 `plantilla1.pt`.

1. Afegiu el renderitzador de TAL per a aiohttp. Cal afegir el paquet `aiohttp-tal` a `iu/requirements/production.txt` i tornar a executar
   `pip install -r iu/requirements/development.txt`.
2. Deseu la plantilla de la pràctica 1 a `iu/templates/plantilla1.pt`.

3. Configureu l'entorn d'aiohttp-tal a `iu/app.py` (de manera semblant a com es configura l'entorn d'aiohttp-jinja2):

```
import aiohttp_tal
import chameleon
[...]
    # aiohttp_jinja2.setup(
    # app,
    # loader=jinja2.FileSystemLoader(str(path / 'templates'))
    # )
    aiohttp_tal.setup(app, loader=chameleon.PageTemplateLoader(
        str(path / 'templates'),
        auto_reload=True # debugging
        )
    )
[...]
```

4. Definiu la ruta a `iu/routes.py`:

```
from iu.main import views

[...]
    add_route('GET', '/doc1', views.plantilla1)
[...]
```

5. Definiu la vista que renderitza la plantilla a `iu/main/views.py`:

```
import aiohttp_tal
[...]
@aiohttp_tal.template('plantilla1.pt')
async def plantilla1(request):
    return {
        'titol':'Plantilles TAL',
        'menu': {
            'actiu': True,
            'actual': 'inici',
        }
    }
```

Fixeu-vos que definim una ruta `/doc1` que renderitza el fitxer TAL `plantilla1.pt` mitjançant les variables que retorna la funció *plantilla1* en un diccionari.

2.0.2 Les plantilles amb macros

Per al segon cas de plantilles amb macro, us proposem d'estendre el programa anterior amb el codi següent.

1. Deseu la macro a `iu/templates/_master.pt` i la plantilla a `iu/templates/inici.pt`.
2. Definiu la ruta a `iu/routes.py`:

```
[...]
    add_route('GET', '/inici.html', views.inici)
    add_route('GET', '/amistats.html', views.amistats)
[...]
```

3. Definiu les vistes que renderitzen la plantilla a `iu/main/views.py`:

```
[...]
@aiohttp_tal.template('inici.pt')
async def inici(request):
    return {
        'inici':{
            'titol':'Plantilles TAL amb macro',
            'menu': {
                'actiu': True,
                'actual': 'inici',
            }
        }
    }


@aiohttp_tal.template('inici.pt')
async def amistats(request):
    return {
        'inici':{
            'titol':'Amistats',
            'menu': {
                'actiu': True,
                'actual': 'amistats',
            }
        }
    }
```

Fixeu-vos que hi ha dues rutes, una per a la pàgina `/inici.html` i l'altra per a `/amistats.html`, i una funció per a cada una que retorna les variables particulars de cada pàgina.

3 Entrada de dades: formularis

Els formularis HTML són la base de la interacció entre els usuaris i les interfícies web. Els formularis permeten enviar dades cap al servidor.

3.1 Exemple

Referència principal: `https://docs.aiohttp.org/en/stable/web_quickstart.html#http-forms`

A continuació hi ha un exemple molt senzill de com atendre un formulari que sol·licita noms en un servidor que els emmagatzema temporalment i retorna a la vista la llista de tots els noms que s'han afegit.

A la plantilla HTML:

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  ...
</head>
<body>
<h1>Llista de noms</h1>
<ul>
 <li tal:repeat="nom noms" tal:content="nom">Nom</li>
</ul>

<h2>Afegeix un nou usuari</h2>
<p tal:condition="afegit">Usuari afegit!</p>
<form method="POST" action="afegeixUsuari">
  <input type="text" name="nom"/>
  <input type="submit" value="Afegeix"/>
</form>
</body>
```

A les rutes:

```
add_route('*', '/afegeixUsuari', views.formulari)
```

A les vistes:

```
noms = []

@aiohttp_tal.template('formulari.pt')
async def formulari(request):
    afegit = False

    if request.method=='POST':
        data = await request.post()
        nom = data['nom']
        noms.append(nom)
        afegit = True

    return {'noms': noms, 'afegit': afegit}
```

3.2 Base de dades

En l'exemple anterior hem utilitzat una variable per emmagatzemar temporalment les dades, però un cop tanquem el servidor aquestes dades no persisteixen.

En el cas que ja hagueu fet l'assignatura de Bases de Dades, ara és l'hora de connectar el servidor amb una base de dades on emmagatzemeu les dades del formulari o d'on les recupereu per a mostrar-les. A aiohttp podeu trobar documentació a `https://create-aio-app.readthedocs.io/pages/commands.html#database` on s'utilitza SQLAlchemy per a enllaçar amb la base de dades amb la tècnica de *Object Relational Mapper* (ORM). També podeu utilitzar SQLAlchemy per a escriure directament les operacions en SQL sense utilitzar ORM. O també podeu utilitzar altres llibreries de Python per a connectar amb el servidor de bases de dades que utilitzeu habitualment.

En el cas que no hagueu fet l'assignatura de Bases de Dades, podeu seguir utilitzant l'emmagatzematge no persistent. A més, podeu inicialitzar aquestes variables amb valors per poder tenir continguts disponibles cada cop que inicieu el servidor. Alternativament, podeu emmagatzemar les dades en fitxers.

4 AJAX

Referència principal: `https://www.w3schools.com/js/js_ajax_intro.asp`

Asynchronous JavaScript And XML (AJAX) és una tècnica web que permet a les aplicacions enviar i rebre dades del servidor asíncronament i canviar dinàmicament el contingut dels documents. El format d'intercanvi de dades pot ser XML, JSON o text pla.

Els components principals que intervenen a AJAX són: Javascript, el DOM i l'objecte XMLHttpRequest.

4.1 Cerca AJAX de ciutats

Com a exemple, a continuació teniu el codi simplificat per a una cerca AJAX de noms de ciutats. Quan l'usuari escriu una cadena `input`, es crea una sol·licitud HTTP GET a `/cerca?text=cadena`. El servidor respon en format JSON les ciutats que concorden amb aquesta cadena i es mostren en una llista al document.

El codi HTML amb l'`input` i les funcions de JS que envien la sol · licitud i processen la resposta:

```
<html>
<head>
...
<script>
function crea_li(text, ul){
  var resultats = document.getElementById('resultats')
  var li = document.createElement("li");
  li.appendChild(document.createTextNode(text));
  resultats.appendChild(li);
}

function resultats(xhr){
  document.getElementById('resultats').innerHTML = "";
  var resposta = JSON.parse(xhr.response);
  resposta.forEach(crea_li);

}

function cerca(text){
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200) {
    resultats(this);
    }
  };
  xhr.open("GET", "cerca?text="+text, true);
  xhr.send();
}
</script>
</head>
```

```
<body>

<form>
  <input type="text" placeholder="Cerca ciutats" onkeyup="cerca(this.value)" />
</form>

<h2>Resultats</h2>
<ul id="resultats">
  <li> Sense resultats </li>
</ul>

</body>
</html>
```

La ruta:

```
add_route('GET', '/cerca', views.cerca)
```

I el codi per a aiohttp associat al camí `/cerca`:

```
from aiohttp import web

ciutats = ['manresa', 'mollerussa', 'manlleu', 'cardona', 'cervera']

async def cerca(request):
    text = request.query.get('text', '')
    filtre = [ciutat for ciutat in ciutats if ciutat.startswith(text)]
    data = sorted(filtre)
    return web.json_response(data)
```

5 Server-Sent Events

Referència principal: `https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events`

HTTP és un protocol client/servidor en què per a rebre una resposta primer cal que el client enviï una petició. *Server-Sent Events* és una tècnica *push* que permet que el servidor enviï dades cap al client en qualsevol moment, en format d'esdeveniments `text/event-stream`. Hi ha altres tècniques *push* com Websockets, Push API o BOSH.

Podeu trobar un exemple de SSE per a aiohttp a `https://github.com/aio-libs/aiohttp-sse`.

6 I18N L10N

Referència principal: `https://chameleon.readthedocs.io/en/latest/reference.html#translation-i18n`

La *internacionalització* (i18n) és el disseny i el desenvolupament d'una interfície d'usuari que en permeti l'adaptació a diferents llengües, cultures o regions. La *localització* (L10n) és l'adaptació a una llengua particular; de fet normalment a una parella de llengua i país particulars, cosa que s'anomena *configuració local* (*locale* en anglès).

Així doncs, en aquest apartat cal que internacionalitzeu la vostra interfície d'usuari i que la localitzeu a dues llengües com a mínim. Seguiu la documentació d'i18n i de l10n per a `aiohttp` a `https://github.com/jie/aiohttp_babel` i a `https://aiohttp-tal.readthedocs.io/en/latest/examples.html#translation-i18n`.

Cada cadena de text que s'internacionalitza s'anomena *missatge*. El mateix text del missatge pot servir d'*identificador* en les traduccions o bé es pot especificar un identificador diferent. I cada missatge sol estar associat a un *domini*, el qual agrupa els missatges per distingir-los d'altres i18n que tinguin el mateix *identificador*.

6.1 Exemple

A continuació teniu algunes pistes per a desenvolupar l'i18n i la l10n de la vostra interfície.

En aquest exemple internacionalitzem el text de la interfície. Com a domini fem servir *ton.domini*, noteu que a Python normalment s'utilitza el *namespace* del paquet com a domini d'i18n.

TASCA 2 Afegiu els paquets `aiohttp_babel` i `babel-lingua-chameleon` a `iu/requirements/production.txt` i torneu a executar
`pip install -r iu/requirements/development.txt`.

6.1.1 I18N de les plantilles

Per a l'i18n mitjançant les plantilles TAL, bàsicament hi ha tres atributs:

- `i18n:translate`
- `i18n:attributes`
- `i18n:domain`

Podeu veure tots els atributs a `http://chameleon.readthedocs.io/en/latest/reference.html#id49`. Aneu alerta si llegiu les altres seccions d'aquesta documentació d'i18n perquè hi ha algunes diferències entre Chameleon i aiohttp a l'hora d'implementar les funcions de traducció.

```
<!DOCTYPE html>
<html i18n:domain="ton.domini">

...

<body>

<p i18n:translate="">Bon dia</p>

<p>
<a href="http://itic.cat" title="Enginyeria iTIC" i18n:attributes="title" i18n:translate="">
  Enginyeria iTIC
</a>
</p>

</body>
</html>
```

6.1.2 I18N del codi Python

Per al codi Python utilitzem les eines de `aiohttp-babel`, que registren una funció per declarar els missatges del domini. Per convenció, aquesta funció es defineix amb guió baix com a nom.

L'estructura bàsica és la següent:

```
from aiohttp_babel.middlewares import _

benvingut = _('Benvingut')
```

6.1.3 Traducció de missatges amb gettext

Per a extreure i traduir els missatges utilitzem GNU gettext. GNU gettext utilitza tres tipus de fitxers en el procés de traducció: `.pot`, `.po` i `.mo`.

Seguiu la documentació de `http://babel.pocoo.org/en/latest/cmdline.html`. Si heu installat `aiohttp_babel` ja inclou la dependència a `babel` i `gettext`.

1. Creeu un directori `locales` en el vostre paquet de Python i efectueu-hi les localitzacions seguint les instruccions de gettext.
2. Definiu el fitxer de configuració `iu/babel.cfg`:

   ```
   [extractors]
   chameleon = babel_lingua_chameleon.extractors:extract_chameleon

   [chameleon: templates/**.*]
   [python: **.py]
   ```

3. Extraieu els missatges i18n del codi i de les plantilles:
   `pybabel extract -F iu/babel.cfg -o iu/locales/ton.domini.pot iu/`
4. Per cada idioma inicieu el fitxer a la l10n:
   `pybabel init -D ton.domini -i iu/locales/ton.domini.pot -d iu/locales -l ca`
5. Compileu a `.mo` la l10n:
   `pybabel compile -D ton.domini -d iu/locales`

Per a l10n només cal que traduïu els missatges en el fitxer de cada idioma, p.ex. a `iu/locales/en/LC_MESSAGES/ton.domini.po`:

```
#: iu/main/views.py:11
msgid "Benvingut"
msgstr "Wellcome"

#: iu/templates/traduccio.pt:19
msgid "Bon dia"
msgstr "Hello World"
```

Si feu canvis en el codi, les plantilles o els fitxers `.po`, només cal que sincronitzeu els canvis de i18n i l10n fent:

1. `pybabel extract -F iu/babel.cfg -o iu/locales/ton.domini.pot iu/`

2. `pybabel update -D ton.domini -i iu/locales/ton.domini.pot -d iu/locales`
3. `pybabel compile -D ton.domini -d iu/locales`

6.1.4 L10N a aiohttp

Un cop tingueu a punt els fitxers de gettext, ja podeu utilitzar els locals a la vostre interfície.

Registreu a aiohttp el directori de traduccions, a `iu/app.py`:

```
from aiohttp_babel.locale import load_gettext_translations, set_default_locale
from aiohttp_babel.middlewares import babel_middleware, _

path = Path(__file__).parent
set_default_locale('ca')
load_gettext_translations(str(path / 'locales'), 'ton.domini')


def translate(msgid, domain=None, mapping=None, context=None,
              target_language=None, default=None):
    # _(message, plural_message=None, count=None, **kwargs)
    return str(_(msgid))


[...]
    aiohttp_tal.setup(app, loader=chameleon.PageTemplateLoader(
        str(path / 'templates'),
        translate=translate,
        auto_reload=True # debugging
        )
    )
[...]

def init_app(config):
    # app = web.Application()
    app = web.Application(middlewares=[babel_middleware])
```

I, de manera senzilla, ja podeu accedir a cada pagina sol·licitant diferents localitzacions. De manera més habitual, els locals es negocien en el protocol HTTP segons la capçalera *Accept-Language* que envia el navegador. També de manera habitual, es permet a l'usuari configurar un local mitjançant cookies o localStorage perquè s'apliqui a tota la interfície. Per defecte `aiohttp_babel` primer cerca si hi ha una cookie de nom `locale` amb l'idioma configurat, si no hi és obté l'idioma de la capçalera *Accept-Language* i si aquest tampoc hi és mostra l'idioma configurat amb `set_default_locale`.