UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Academic Master in Geomatic Engineering and Geoinformation
WEB DEVELOP AND GEOPORTALS

Guide of developed contents in lectures.

J. Gaspar Mora Navarro.

Univesitat Politécnica de Valencia.
Departamento de Ingeniería Cartográfica, Geodesia y Fotogrametría Department of Cartographic Engineering and Photogrametry

Valencia, February 2017

WEB DEVELOP AND GEOPORTALES.

J. Gaspar Mora Navarro

5 de mayo de 2017

Índice general

1 Web develop in Linux **1**
- 1.1 Introduction to Linux . . . 2
  - 1.1.1 Apache HTTP server . . . 2
  - 1.1.2 Apache configuration files . . . 4
  - 1.1.3 Project folder organization in this course . . . 5
  - 1.1.4 Check a WSGI project . . . 6
    - 1.1.4.1 Python code of the WSGI application *ejemwsgi* . . . 6
    - 1.1.4.2 Static code of the WSGI application *ejemwsgi* . . . 7

2 HTML5, CSS y JavaScript **8**
- 2.1 Technologies related with the developing and geoportals . . . 9
  - 2.1.1 Base de datos: el componente más importante . . . 9
- 2.2 HTML 5 . . . 11
  - 2.2.1 New features from HTML 5 . . . 11
  - 2.2.2 Publicar una web . . . 11
  - 2.2.3 HTML 5 label classification . . . 11
    - 2.2.3.1 Labels for the structure of the page . . . 12
    - 2.2.3.2 Label for the page structure inside the BODY . . . 12
    - 2.2.3.3 Text groups . . . 12
    - 2.2.3.4 Hipertext and semantics . . . 13
    - 2.2.3.5 Tablas . . . 13
    - 2.2.3.6 HTML5 resources . . . 14
  - 2.2.4 Exercise 1. First steps with HTML 5 . . . 14
  - 2.2.5 Exercise 2. Creation of a web page with structure but without styles . . . 14
- 2.3 Cascade style seeds. CSS . . . 17
  - 2.3.1 Positioning elements in HTML 5 with CSS . . . 19
  - 2.3.2 Exercise 3. CSS. . . . 20
- 2.4 Formularios HTML5 . . . 21
  - 2.4.1 Práctica 4: formularios . . . 21
- 2.5 JavaScript . . . 22
  - 2.5.1 Exercise 5. JavaScript . . . 25
  - 2.5.2 GPS: Geolocation interface . . . 27

3 IDE configuration and *pozossan* projects installation **28**
- 3.1 IDE configuration . . . 29
- 3.2 *pozossan* project installation . . . 30

4 OpenLayers 3.5 **31**
- 4.1 Configure JSONP in GeoServer . . . 32
- 4.2 OpenLayers 3.5 map definition . . . 32
- 4.3 WMS geoserver Styles. SLD . . . 33

4.4 Click, Selec, Draw and Snap interactions . . . 34
4.5 Exercise 6. Map drawing and modify. . . . 34

5 Dynamic sites with Python and WSGI 36
5.1 Before start programming with Python . . . 37
5.1.1 Debugging Python code. File error.log . . . 37
5.1.2 Mostrar errores en el navegador . . . 37
5.1.3 Remote debugging with PyDev . . . 37
5.1.4 Importing modules from a WSGI application . . . 38
5.2 WSGI application example: *dw_wsgi_example* . . . 38
5.2.1 Create the wsgi application index.py . . . 38
5.3 Divide and you will win . . . 40
5.4 Exercise 7. Division of a HTML page in small parts . . . 42
5.5 Send and process GET data in a wsgi application . . . 42
5.5.1 Manage GET data in the urls . . . 42
5.5.2 Server data send from html, using GET method . . . 44
5.5.3 Example of processing GET data in order execute the adequate Python function . 44
5.6 Manage JSON strings . . . 45
5.7 Sending data to the server and to receive data from the server with Ajax . . . 46
5.8 Exercise 8. Ajax . . . 48
5.9 To make changes in PostgreSQL with Pyhton. Use of the *psycopg2* library . . . 49
5.9.1 Table creation . . . 50
5.9.2 To insert rows with geometry . . . 50
5.9.3 Updating rows . . . 51
5.9.4 Selecting rows . . . 52
5.10 Functions to help the student . . . 54
5.10.1 To execute queries with HTML data forms . . . 54
5.10.2 To load a JSON string in a HTML form . . . 55
5.11 Exercise 9. Database update across Internet . . . 55
5.12 Session control . . . 55
5.12.1 Session management example with Beaker . . . 56

6 Evaluation 58
6.1 Evaluation . . . 59
6.1.1 Project minimum requirements of the own project . . . 59

Índice de figuras

2.1 Technologies related with the web develop and geoportals. 9
2.2 Schema client-server. 10
2.3 A NAV element in the HEADER and a ASIDE element in the BODY. 15
2.4 Main section and a subsection inside for the water wells 15
2.5 Section inside of the main section for the pipe data. 16
2.6 Footer, for the body, with the images which are links to other pages 16
2.7 Schema of the parts of the page. 17
2.8 CSS cage model. Source: Tutorial of CSS from HTML.NET. http://www.w3schools.com/CSS/css_boxmodel.asp
2.9 Style effect over the html objects. 19
2.10 Object position and styles from the page using Bootstrap. 20
2.11 User input available checks. 21
2.12 Formularios con HTML5 y Bootstrap. 22
2.13 JavaScript program to sum two numbers. 24
2.14 Change the content of a html element. 25
2.15 Point radiation application. 26

3.1 *pozossan* database relationship schema. 29

5.1 Division of a page in small files. 42

CAPÍTULO 1

Web develop in Linux

Introduction to Linux

In this subject its going to be used a virtual machine with Ubuntu 14.04. The virtual machine can be downloaded from *http://upvusig.car.upv.es/static/mvdesweb/*.

To execute the virtual machine, it can be used the VMWare program, free of charge for not commercial use. It can be downloaded from
https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0

You must remember that, using Linux, the names of folders and files are sensitive to upper case and lower case letters.

Solo existe el usuario *desweb*, con contraseña *desweb111*, que es superusuario.

Only exists the user *desweb*, with password *desweb111*, which is superuser.

```
Linux:
    user: desweb, psw: desweb111

Apache tomcat7
    user: tomcat
    psw: tomcat

Geoserver 2.7.6
    user: admin
    psw: geoserver

Postgres
    user postgres, psw: postgres (it must be configured)
    desweb, psw: desweb111 (it works)
    Both are superusers
```

Each user only can see and write in his own folder. Each user has one folder assigned in the folder *home*

```
/home/user1
/home/user2
/home/user3
...
```

If you want to access to modify the content from other folders with one program, it is necessary to to execute that program with administrator permissions. This can be managed with the console:

```
sudo program-to-execute
```

Apache HTTP server

Apache is a program classified as a daemon, or service. A service or daemon is a program which always is waiting for requests, in a specified port. In the Apache case is the 80 port.

Apache is a service which serves files and executes scripts, but, files and scripts, have to be in the */var/www/* folder. Outside that folder, Apache can't have access. The */var/www/* folder is a folder of the system, to modify its content it is necessary to use the *sudo* command.

Apache se configura mediante directivas o comandos desde fichero de texto. Las configuraciones que se van a explicar se introducen siempre en el mismo fichero:

Apache is configured using directives o commands in text files. The configurations which are going to be explained in this subject are all in the following file:

/etc/apache2/sites-avaiable/000-default.conf

The Apache service is not initialized when the virtual machine is started. In order to start the Apache service and to have able to serve files, you must to execute the following command:

```
sudo service apache2 start
```

Durante la programación, los errores de Pyhon, son reportados por Apache al fichero */var/log/apache2/error.log*. El cliente recibe un mensaje *500: internal server error*. Para saber qué está pasando hay que ver el fichero .log Apache, donde se encuentra la traza del error de python. En el listado siguiente se ve un ejemplo que avisa de que la variable *lista_dic* se ha usado antes de asignarle un valor.

While programming, Python errors are reported, by Apache, into the file */var/log/apache2/error.log*. The client (the person which is visiting the page with a web navigator), will receive 500 internal server message. In order to know what happened it is necessary to have a look to the *error.log* file. In the following listing a example is sowed. The problem is that *lista_dic* variable is used before to assign it a value.

```
[Fri Feb 17 16:51:13.799903 2017] [core:notice] [pid 21144:tid
   140313365424000] AH00094: Command line: '/usr/sbin/apache2'
[Fri Feb 17 16:52:12.396339 2017] [wsgi:error] [pid 21149:tid
   140313066686208] [client 127.0.0.1:60306] Error - <type '
   exceptions.UnboundLocalError'>: local variable 'lista_dict'
   referenced before assignment, referer: http://localhost/static/
   dw_pozossan/dw_pozossan.html
[Fri Feb 17 16:52:51.039097 2017] [wsgi:error] [pid 21149:tid
   140313049900800] [client 127.0.0.1:60318] Error - <type '
   exceptions.UnboundLocalError'>: local variable 'lista_dict'
   referenced before assignment, referer: http://localhost/static/
   dw_pozossan/dw_pozossan.html
```

Si se cambia la configuración en el fichero */etc/apache2/sites-avaiable/000-default.conf*, hay que reiniciar el servicio.

In web developing process It will be necessary to change the configuration of Apache, or to change the Pyhton scripts code very often. In all cases the Apache service have to be reloaded.

```
sudo service apache2 restart
```

Si se quiere modificar el fichero */etc/apache2/sites-avaiable/000-default.conf*. Se escribe en una terminal:

If you want to modify the */etc/apache2/sites-avaiable/000-default.conf* file, you must to write in the console:

```
sudo nautilus
```

Nautilus is the folders explorer. The above command executes Nautilus as administrator. Then it si possible to modify all the files in the file system. Once the file is localized whit Nautilus, click over the file to modify with the right button of the mouse, and select the option *Open con Gedit*, which is like the Notepad program in Windows.

Now we know how to modify the apache configuration file, but it is necessary to know some directives.

Apache configuration files

The most important file is */etc/apache2/sites-avaiable/000-default.conf*. It is necessary to know the following directives:

Indicates the root folder. Is the address where Apache accesses when writing the *http://localhost* address. The document root default is:

```
DocumentRoot "/var/www/html/"
```

The *C:/var/www/html/* folder must no contain any Python script because the user will have able to see the code, and this is dangerous. That folder only must contain static content. Static content are files which are not executed in the server: html, css, image, txt, js, ... Never put in the document root folder files like .py, .php, ...

To allow access to other folder content, inside */var/www/*, the following can be used:

```
<Directory "/var/www/apps/static/">
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

The above code allow the total access to the folder */var/www/apps/static/*. Like document root, never put script which are executed in the server. This folder is commonly used for to serve the static content, instead the document root, despite that both have de same permissions.

All the new directives must be added at the end of the file, before the last label of the file */etc/apache2/sites-avaiable/000-default.conf*.

The above listing allow total access of read to the folders and subfolders in */var/www/apps/static/*, but it is necessary to to one more step more. The access to other folders, different to the document root, is given using *alias*. A alias is a word introduced inside the web address. Apache detect the alias word and redirects to the associated folder. In the following listing there is an example:

```
Alias /static/ /var/www/apps/static/
<Directory /var/www/apps/static>
    Options +Indexes
    Order allow,deny
    Allow from all
</Directory>

Alias /lib-js/ /var/www/apps/lib-js/
<Directory /var/www/apps/lib-js/>
    Options +Indexes
    Order allow,deny
    Allow from all
</Directory>
```

The above listing give access to the */var/www/apps/static/* folder writing the address *http://localhost/static/* in the web navigator.

On the same form, the above listing, give total access of read to the folder */var/www/apps/lib-js/*, writing in the web navigator the address *http://localhost/lib-js/*.

Cada proyecto se va a componer de dos partes: programación en el servidor, que serán ficheros Python, y programación en el cliente (ficheros de imagen, CSS y JavaScript). La programación en el servidor y en el cliente no se puede mezclar. Se usan carpetas separadas y proyectos de desarrollo distintos. Por ejemplo, para un geoportal, denominado rutas-cheste, se crearía la siguiente estructura:

In this course, each project will be divided in two parts: static content and script content. Each part will be located in separated folders with different access permission. Each part will be a folder with the same name, but in different location. For the static content will be used a project name folder inside */var/www/apps/static/*, and, for the scripts, it must be used the same name folder project but inside the folder */var/www/apps/desweb/*. In the following listing its sowed a example from a project called *ejemwsgi*.

```
/var/www/apps/static/ejemwsgi/ /*Static content for the ejemwsgi
    project*/
/var/www/apps/desweb/ejemwsgi/ /*Folder for the script files for
    ejemwsgi project*/
```

La carpeta *lib-js*, se utilizará para poner librerías de terceros: OpenLayers, Jquery, ...

Project folder organization in this course

All the projects must be placed in the following folder:
/var/www/apps
Inside that folder, the project must have divided in two folders with the same name , one of them must be placed in */var/www/apps/static/* and the the other in */var/www/apps/desweb/*. In the following listing its sowed a example from a project called *ejemwsgi*.

```
/var/www/apps/static/ejemwsgi/ /*Static content for the ejemwsgi
    project*/
/var/www/apps/desweb/ejemwsgi/ /*Folder for the script files for
    ejemwsgi project*/
```

The static part of the project is accessible with the url: *http://localhost/static/ejemwsgi/*. The dynamic part of the project, the part which contains the scripts in python will be accessible with other Apache alias, using the *WSGIscriptAlias* Apache directive. This alias allow the access only to one .py file, which is the main file of the application. The Phyton application can be formed by many other .py files, but the access must be granted only to the main .py file across the alias. The main file must import the rest of the .py files which form the entire application. In the following listing are an example, where index.py is the main file of the application.

Listado 1.1: Crear un alias para una aplicación WSGI

```
WSGIScriptAlias /ejemwsgi/ /var/www/apps/desweb/ejemwsgi/index.py
```

El listado anterior, la directiva *WSGIScriptAlias* indica a Apache que en las rutas donde aparexca */ejemwsgi/*, se redirecciona a */var/www/apps/desweb/geops/index.py* y que es es un script WSGI, que debe ejecutar mediante Python. Apache se queda esperando a que el la aplicación termine y le envíe los resultados, en forma de texto, que Apache reenviará al cliente que hizo la petición.

The above listing indicates to Apache that, when Apache finds the string */ejemwsgi/* in a url, for example in the url *http://localhost/ejemwsgi/*, Apache must execute the file */var/www/apps/desweb/-geops/index.py*. Really it is not Apache who execute the scripts. Apache send the script to the interpreter, in this case Python. Apache waits the that Python finishes. When Pyhton finishes send a response to Apache. Apache resend the Pyhton response to the client, which web navigator renders.

The Python script is a normal Pyhton script, then it is possible to access to many resources like databases, any python library, or programs like R.

The technique of communication between Apache and Python is called WSGI. Then the we are going to build WSGI applications.

We are going to use Liclipse IDE to develop. Liclipse uses workspaces to work. Each workspace is the main folder where Liclipse creates each project. Each project is a new folder in the worspace folder. In the virtual machine there are three Liclise workspaces created. One to work in the folder

/var/www/apps/static/, other to work in the folder */var/www/apps/desweb/*, and other to work in other folder in inside */home/desweb/*. To use the first two workspaces it is necessary to execute Lilcipse as super user. In the first workspace the projects must be from *Static web project* type, and in the second workspace, the projects must be from *PyDev* type. In the firs case the develop is in JavaScript, HTML5 and CSS languages, and in the second, the language of develop is Python.

Check a WSGI project

There is an example of WSHI application on the virtual machine. To check it you have to write in the web browser *http://localhost/ejemwsgi/*.

Check the HTML code sent by the server to the client. To see the code press crl + u in the web browser.

To see the Python part of the project, do the following:

- Open a terminal: ctrl + alt + t.
- Write *sudo liclipse* /*It is necessary to introduce la the administrator password each time sudo is used. You must to knw that Linux do not shows anything when a password is introduced.*/
- Chose the workspace */var/www/apps/desweb*
- Check the files form the project ejemwsgi. The file index.py.

To check the static part of the project:

- Open an other terminal: ctrl + alt + t.
- Write *sudo liclipse*.
- Chose the workspace */var/www/apps/static*
- Check the files. In this case only there is a file css/estilos.css.

Python code of the WSGI application *ejemwsgi*

The only file of the project is *index.py*. Its content is showed in the following listing:

```
# -*- coding: utf-8 -*-
'''
Created on 12 de abr. de 2016
@author: Joaquín Gaspar Mora Navarro
'''

#That will be necessary later
import sys, os
dir_base=os.path.dirname(__file__)
sys.path.append(dir_base)

#To debug
#sys.path.append(r'/opt/liclipse/plugins/org.python.pydev_4
    .5.5.201603221237/pysrc')
#import pydevd;

#To debug
#pydevd.settrace()
```

```
def application(environ, start_response):

    html="""
    <!DOCTYPE HTML>
    <html lang="es">
    <head>
        <meta charset="UTF-8"/>
        <meta name="description" content="Asignatura Desarrollo Web
           y Geoportales" />
        <meta name="author" content="Gaspar Mora-Navarro" />
        <meta name="keywords" content="máster, geomática,
           geoinformación, upv, geoportal, mapserver, openlayers,
           javascript"/>
        <title>Ejemplo de aplicación wsgi</title>
        <link rel="stylesheet" type="text/css" href="http://
           localhost/static/ejemwsgi/css/estilos.css">
    </head>
    <body>
    <h1>Desarrollo web y geoportales en Linux</h1>
    <br>
    <p>Funciona la app wsgi</p>
    </body>
    </html>
    """

    #b=10/0#error para probar la lirería paste, de depuración

    status = '200 OK'
    response_headers = [('Content-Type', 'text/html'),('Content-
       Length', str(len(html)))]
    start_response(status, response_headers)
    return [str(html)]

#Descomentar para depuración. Volver a comentar para producción
#from paste.exceptions.errormiddleware import ErrorMiddleware
#application = ErrorMiddleware(application, debug=True)
```

Static code of the WSGI application *ejemwsgi*

In this case is only one CSS file, called *estilos.css*:

```
@CHARSET "UTF-8";
p {color:blue;}/*Los párrafos de color azul*/
h1 {color:maroon;}/*Los títulos H1 de color marrón*/
```

Check how both parts, static an dynamic, are connected because the WSGSI application sends the html content to the client, and in this code, the static content is linked. Check the following line on the Pyhton code and the code received by the client.

```
<link rel="stylesheet" type="text/css" href="http://localhost/static
   /ejemwsgi/css/estilos.css">
```

The */static/* alias give access to the folder */var/www/apps/static/*. Up to here the rest of the address to the css file is specified in the url. The client, when read this line downloads the static files and uses, the css files, to render the page, the js files are used to give functionality, and so on.

CAPÍTULO 2
HTML5, CSS y JavaScript

Technologies related with the developing and geoportals

Base de datos: el componente más importante

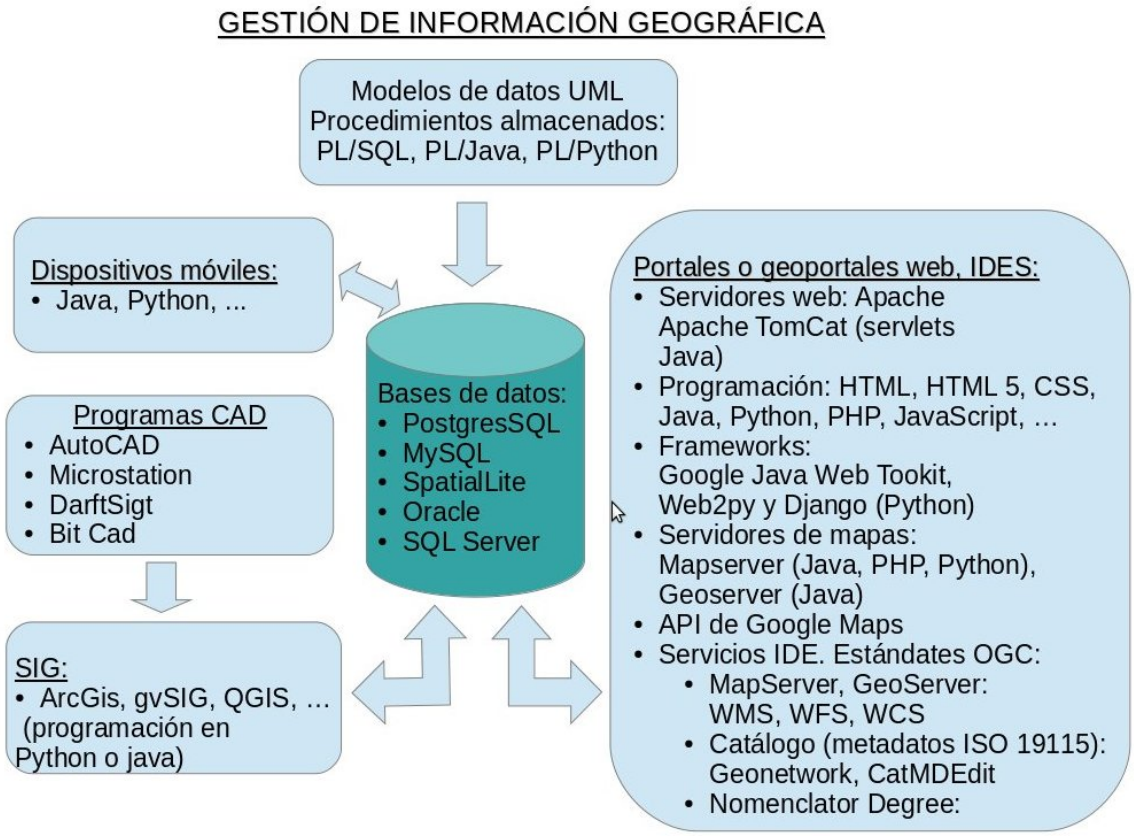


Figura 2.1: Technologies related with the web develop and geoportals.

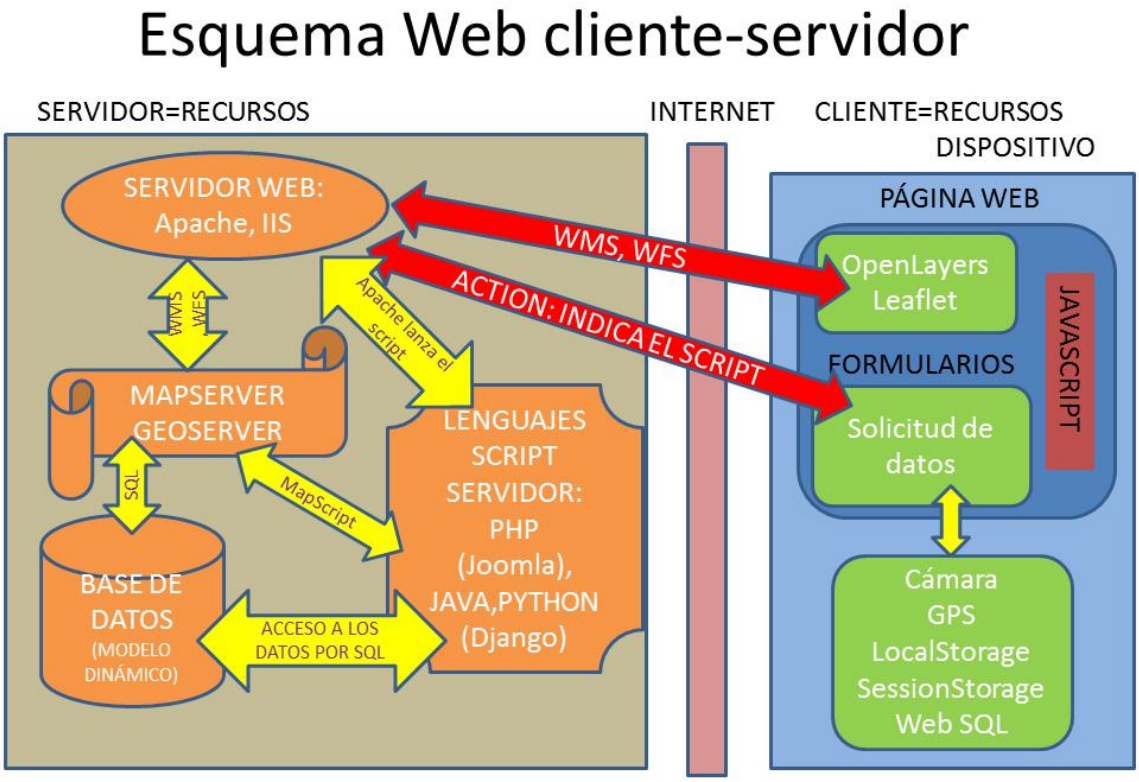


Figura 2.2: Schema client-server.

HTML 5

New features from HTML 5

HTML 5 is a Hipertext Markup Language. The content is marked with labels. Each label wrap the content, and web navigators shows the content depending on that labels. In HTML5, the appearance is managed using external files called CSS (Cascade Style Seeds).

The main characteristics from HTM5 are:

- There are new elements to define the structure, or the parts, from the page. The labels are: HEAD, BODY, HEADER, FOOTER, NAV, SECTION y ASIDE, ARTICLE, TABLE (CAPTION, THEAD, TBODY, TFOOT).
- For the forms, the mos important is the new checks that now have the controls.
- Has been added the possibility of GPS access and the camera of the device. For a more checked access the PhoneGap or Cordova can be used. That libraries avoid the use of Java to access to other services in the device.
- Has been added a control to draw: CANVAS.
- It is added the possibility of to store data in the client, up to 5 mb each page, with the new tools: LocalStorage, sessionStorage and web Storage. That adds the possibility to store data off line, in order to be sent to the server when there was internet connexion.
- All the appearance of the page is managed using CSS external files, across the selectors.

Publicar una web

In order to have able to publish a page, the page have to be stored in a server connected to Internet, and with a HTTP server installed. Personal computers have some problems to be used as a servers. The reasons are: security against attacks, energy cuts, money in energy, and not fixed IP address. The IP address can be transformed in fixed for free with some programs.

There are companies which rent servers where it is possible to install any software. They are called dedicated servers. The operative system can be Windows or Linux. There are companies which allow to have small a dedicate server for free.

Before the page is visible it is necessary to buy a domain to have an not numeric address. Visit the DNS definition in the Wikipedia:
http://es.wikipedia.org/wiki/Domain_Name_System.

HTML 5 label classification

In this section it is going to list the most common labels. The student have to request more extensive documentation to use the each label. The recommended page is *https://www.w3schools.com/*.

Labels for the structure of the page

The labels for the basic structure of the page are: DOCTYPE, HTML, HEAD Y BODY. The first line of the document have to be always the following, which indicates to the web navigator that the page is in HTML 5.

```
<!DOCTYPE HTML>
```

The HEAD label, wrap the metadata of the page. It is really important to fulfilling all the metadata correctly, in order to be found by the searcher robots:

TITLE : Mandatory. Defines the title of the page.

BASE : Folder base to all the not completed addresses in the page.

LINK : Link CSS files.

SCRIPT : Wrap JavaScript code or link a external JavaScript file.

META : Page metadata. There are a lot of types of metadata: author, charset, keywords... The maximum number of metadata must be fulfilled.

STYLE Wrap the styles definition for the page. This is not recommended to use.

The BODY label enclose the real page content.

Label for the page structure inside the BODY

The structure of the page, inside the BODY, is defined with the following labels:

SECTION : Separate independent sections.

ARTICLE : Special for use in articles. Includes labels of metadata for the article, like summary, etc.

ASIDE : Is used to show complementary information like Links to pages, options or explanations. Is located usually on the right or left side.

HEADER : Head of the page, section or article.

FOOTER : Foot of the page, section or article.

ADRESS : Is used to specify addresses of contact.

H1...H6 : Titles from more to less importance.

DIV : A division which enclose a part of the page to be managed. That label is the most commonly used.

Text groups

To group text, there are the following labels:

HR : Line of horizontal separation.

PRE : Preserves the content format: spaces, tabs, lines...

BLOCKQUOTE : Used to mark the content of other authors.

FIGURE Y FIGCAPTION : Wrap a IMG label and adds metadata and title to the image.

UL, OL, LI, DL, DT, DD : Different type of lists.

Hipertext and semantics

Labels to define links, references, internal parts, labels to format text and labels to give mining.

A : To define hiperlinks to other content. Visit *http://es.html.net/tutorials/html/lesson8.php*.

ABBR : To specify what means an abbreviation.

B : Bold.

BR : Line break.

I : Italic font.

MARK : Highlights the content filling the characters.

Q : Introduces double quotation marks.

S : Cross out a text.

SPAN : Modifies the color of the text.

SUB : Enclose a sub-index.

SUP : Enclose a super-index.

TIME : Indicates that inside there is a date. Do not format different the date.

CITE, EM, DFN, STRONG, CODE, SAMP, KBD, VAR : Other special formats for the content.

Tablas

Las tablas son un elemento fundamental y se le han añadido cantidad de etiquetas que ayudan a comprender el contenido.

Tables are a fundamental feature. More labels are been added to tables, in order to understand the content of the tables.

TABLE : Starts the table.

CAPTION : Title.

TR, TD, TH : Row, column content and header.

THEAD : Encloses the headers of the columns.

TBODY : Encloses the content of the table.

TFOOD : Wrap the summary rows.

COLSPAN Y ROWSPAN : Are used to expand a cell.

HTML5 resources

HTML 5 page reference: *http://www.w3schools.com/html/default.asp*.

To check if the page does not have, errors the page *http://validator.w3.org* can be used.

The navigators not hold all the HTML 5 labels. In order to know if a web browser holds a label, you can visit the page: *http://caniuse.com*.

Neither all the mobile devices hold all the HTML 5 labels. It is possible to check it in *http://mobilehtml5.org*.

Exercise 1. First steps with HTML 5

The objective for this Exercise is that the student be familiarized with the labels and the syntax of HTML 5.

Do the following steps:

- Create a new Liclipse project, from type *Static web project*, in the workspace */var/www/apps/static*. The project must be called *prac1*[(1)].
- Inside the project folder, Lilipse creates a folder called *WebContent*. This folder must not be used. In the root project folder, create a HTML file, called *prac1.html*.
- Visit *http://www.w3schools.com* and search each of the labels sowed in this chapter. All the labels must be used in the html file.
- Añadir un alias al servidor web Apache, de forma que el alias */practica1/*, muestre el documento *practica1.html*. Add an alias alias */practica1/* to the server Apache 2. That alias mus show the file *prac1.html*.
- Remember restart Apache server to reload the new configuration.
- Visit the page in the web explorer.

Delibery:

The student must show the page to the lecturer and to answer his questions.

Exercise 2. Creation of a web page with structure but without styles

The goal of this exercise is that the student understand the use of the structural labels of HTML 5: HEADER, FOOTER, NAV, SECTION, ASIDE, LISTAS DE BOTONES, etc. The result will be not very beauty because the page does not have still styles applied. That will be the next exercise.

The student have to use the identifiers used in the figure 2.7. This is important to apply styles later.

Do the next steps:

Create a new Lilipse static project called *prac2*, and a html document with the same name. It must be created a page with the parts sowed in the figures 2.3, 2.4, 2.5 y 2.6, following that sequence.

(1) Forbidden to use spaces accents, jans for the name of the folders or files, inside */var/www/*.

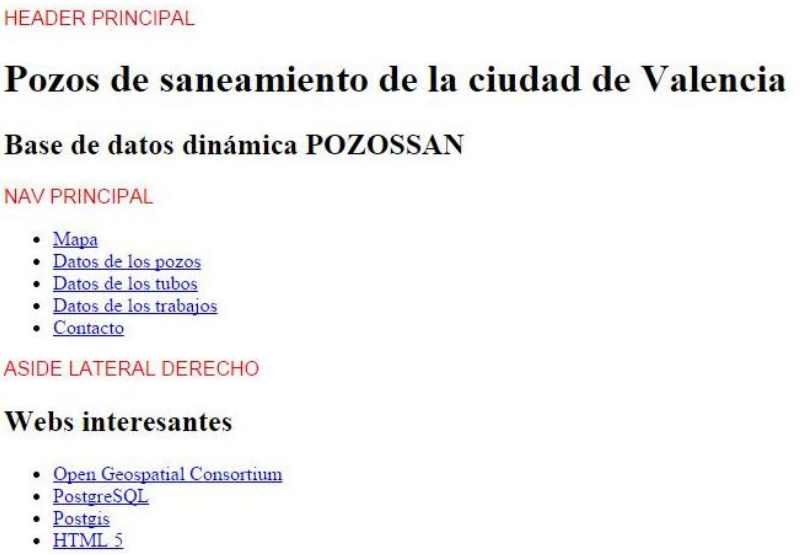


Figura 2.3: A NAV element in the HEADER and a ASIDE element in the BODY.

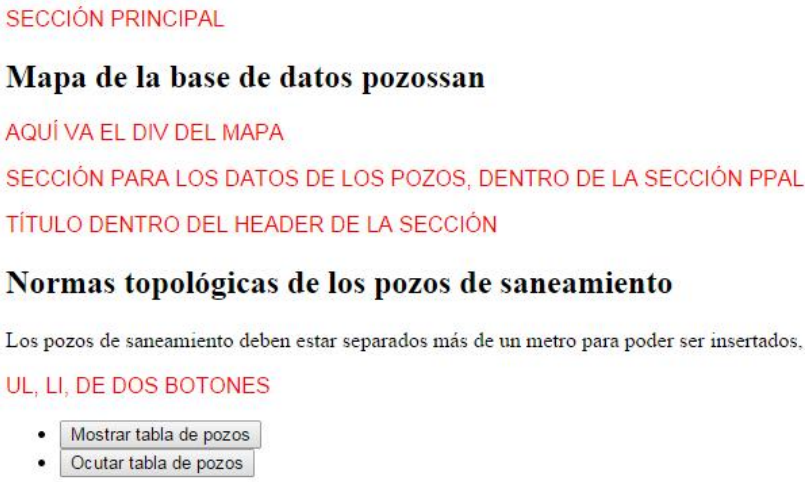


Figura 2.4: Main section and a subsection inside for the water wells

TABLA CON SU CAPTION THEAD, TFOOT y TBODY (border=1)

Tabla de atributos de la capa de pozos

gid	descripción	z_tapa	id_trabajo	profundidad	diametro
1	Pozo normal	10	1	3	1.1
2	La tapa estaba en mal estado	15	1	3	1.1
Medias				**3**	**1.1**

NUEVA SECCIÓN DENTRO DE LA SECCIÓN PRINCIPAL PARA LOS TUBOS

TÍTULO DENTRO DEL HEADER

Normas topológicas para los tubos de saneamiento

Los tubos de saneamiento deben seguir las siguientes normas topológicas:

1. Deben comenzar y acabar exactamente sobre un pozo
2. Denen tener pendiente descendente
3. No pueden intersectarse
4. No pueden autointersectarse

FIN DE LA SECCIÓN PRINCIPAL

Figura 2.5: Section inside of the main section for the pipe data.

FOOTER FUERA DE LA SECCIÓN PRINCIPAL

Asignatura Desarrollo web y geoportales

NAV DENTRO DEL FOOTER

- UNIVERSITAT POLITÈCNICA DE VALÈNCIA
-

Figura 2.6: Footer, for the body, with the images which are links to other pages

In the figure 2.3, in the HEAD element, it must be introduced the following labels: TITLE and META. For the META label it is necessary to fill the following NAME: CHARSET, DESCRIPTION, AUTHOR y KEYWORDS.

In the BODY the HEADER section must be created, and introduce a NAV element, which property *id=nav_principal*. The content of this NAV are the lists which are sowed under the title *Base de datos dinámica POZOSSAN*, in the figure 2.3.

The table which appears in the figure 2.5, must contain the labels CAPTION, THEAD and TFOOT, in addition to the normal labels.

In the figure 2.6, is sowed the FOOTER, where there is a title and a navigation element NAV. The NAV contains a list of UL elements with the images of the UPV and this school. The images contain hyper-links to the respective pages.

In the figure 2.7, a schema is sowed. The navigator shows, up to now, the elements one over other. The position an appearance will be manage later, using CSS.

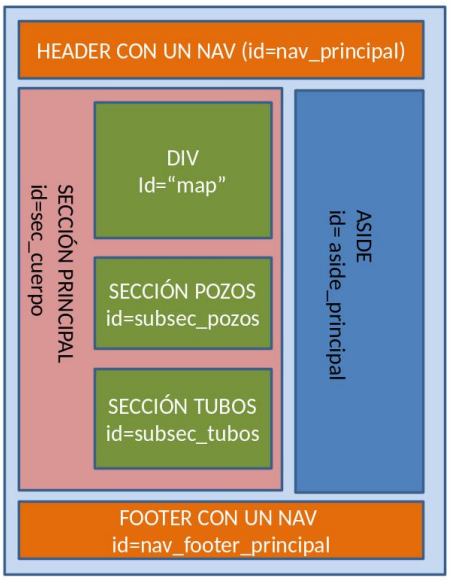


Figura 2.7: Schema of the parts of the page.

Create an Apache alias in order to visit the page.

Delibery:

The student must show the page to the lecturer and to answer his questions.

Cascade style seeds. CSS

CSS indicates how the objects must be rendered in the web browser. CSS can be specified inside the HTML or outside in separate files. The second option is the recommended. To apply styles to elements, the elements must be selected. This is managed using selectors CSS.

Visit all the pages that are sowed in the following.

- Reference page of CSS3: http://www.w3schools.com/css/default.asp. In this page are all the properties and the possible values, with examples.
- The selectors list is in: *http://www.w3schools.com/cssref/css_selectors.asp*. The students have to know the seventh first selectors.

- The CSS code can be checked in *http://www.css-validator.org/validator.html.es*.
- The color elements can be checked in *http://www.tutorialspoint.com/html5/html5_color_names.htm*

Visit the page *http://www.w3schools.com/css/css_syntax.asp* and have a look the syntax. Check all the examples in the page and do the four final exercises.

Watch the figure2.8, where it is possible to see a schema of properties which all the HTML elements hace: *margin*, *padding* and *border*.

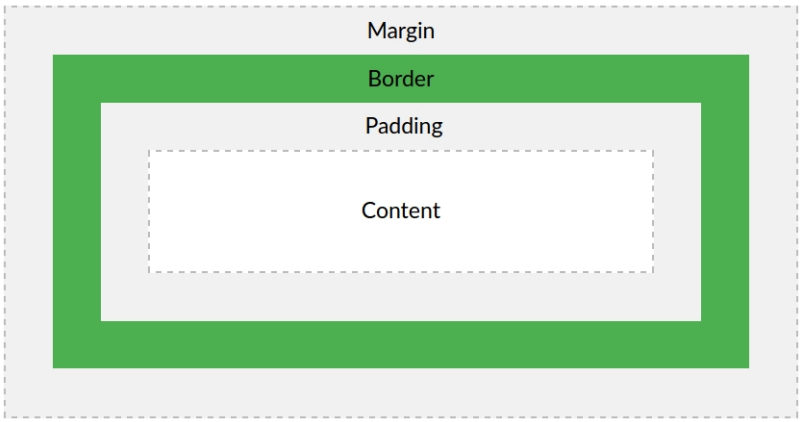


Figura 2.8: CSS cage model. Source: Tutorial of CSS from HTML.NET. http://www.w3schools.com/CSS/css_boxmodel.asp.

Selectors to know:

- Label selector: *p*, selects all the paragraphs.
- Id selector: *#map*, selects only the element with *id=map*. This is the most specific selector.
- Class selector: *.grande*. Selects all the objects which *class=grande*.
- Class of etiqueta: *div.rojo*, selects all the div which *class=rojo*. Select several labels at time: *section, article, ...* Selects the sections an the articles.
- Select labels which are inside others: *section article p*, selects the paragraphs that are inside an article, that are inside a section.
- Combinations: *section#principal article.guerra p*, selects the paragraphs which are inside an article, which classs=guerra, and which are inside the element *id=principal*.

In the following listing a example is presented. The result can be checked in the figure 2.9.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {background-color: lightblue;}
        h1 {color: white; text-align: center;}
        p {font-family: verdana;font-size: 20px;}
        .un_poco_mas_grande{font-family: verdana;font-size: 25px;
           color: green;}
```

```
        #p1{font-family: verdana;font-size: 30px;color: red;}
    </style>
</head>
<body>
    <h1>My First CSS Example</h1>
    <p>Párrafo normal.</p>
    <p class="un_poco_mas_grande">Clase un poco más grande.</p>
    <p class="un_poco_mas_grande">Clase un poco más grande.</p>
    <p id="p1" class="un_poco_mas_grande">Clase un poco más grande,
       pero modificado por un selector más específico.</p>
    <p>Párrafo normal.</p>
    <p>Párrafo normal.</p>
</body>
</html>
```

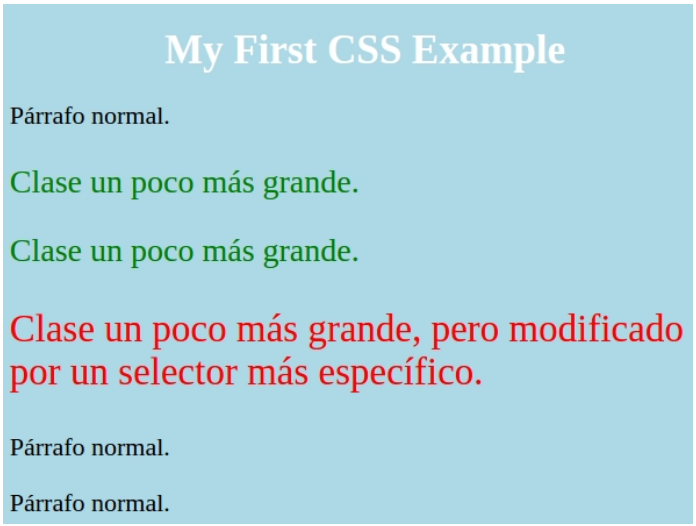


Figura 2.9: Style effect over the html objects.

Positioning elements in HTML 5 with CSS

As a full example of structured page, visit the address *http://upvusig.car.upv.es/static/dw_p2_css/p2_css.html*. The page has the same elements and identifiers that the used in the exercise 2, figura 2.7.

Copy the code HTML and CSS and create a new Liclipse static project. Link the CSS to the page and comment and uncomment part of the CSS file to see the result on the page.

Over all it is important to know how to put the objects in the right position. Check the properties *float:left*, *float:right* and *clear:both* from the objects. Using the property *float* it have been possible to put the section *cuerpo* on the left of the object *aside*. The *clear:both* instruction clean all the objects and make to use the full width to the element.

Exercise 3. CSS.

Manage, using CSS, that the page create in the Exercise 2 have the appearance of the figure 2.10. In this case you must use the Boostrap library *https://www.w3schools.com/bootstrap/default.asp*. This library allows positioning the objects, but is responsible to the size of the screen from the device where the page is being sowed. The student must experiment with the examples and, sowing the figure, solve the exercise.

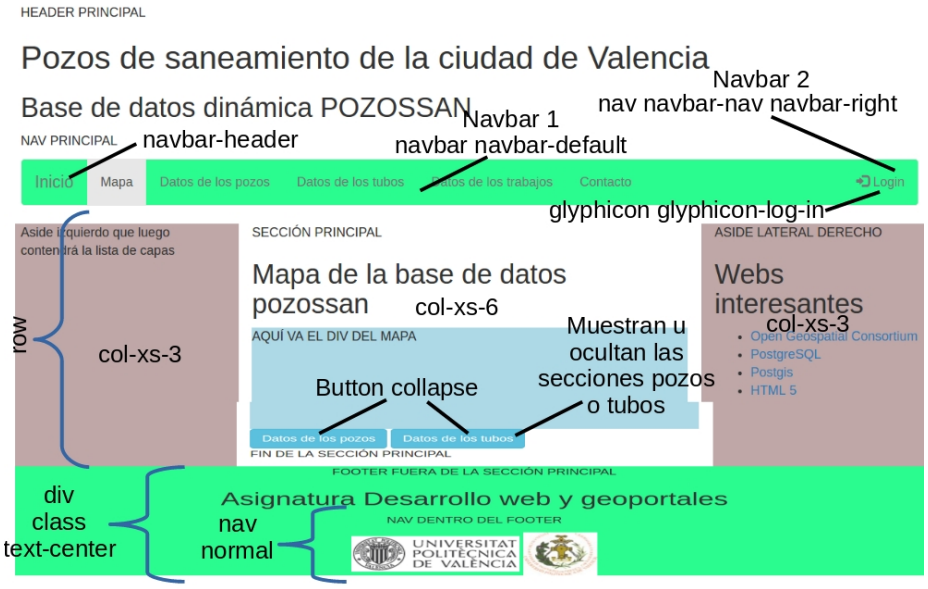


Figura 2.10: Object position and styles from the page using Bootstrap.

Ti solve the exercise it is necessary to add the following content to the CSS of the page:

```
/*
 This is which do that all elements from a row have the same height
*/
.row {
  display: -webkit-box;
  display: -webkit-flex;
  display: -ms-flexbox;
  display:         flex;
}
```

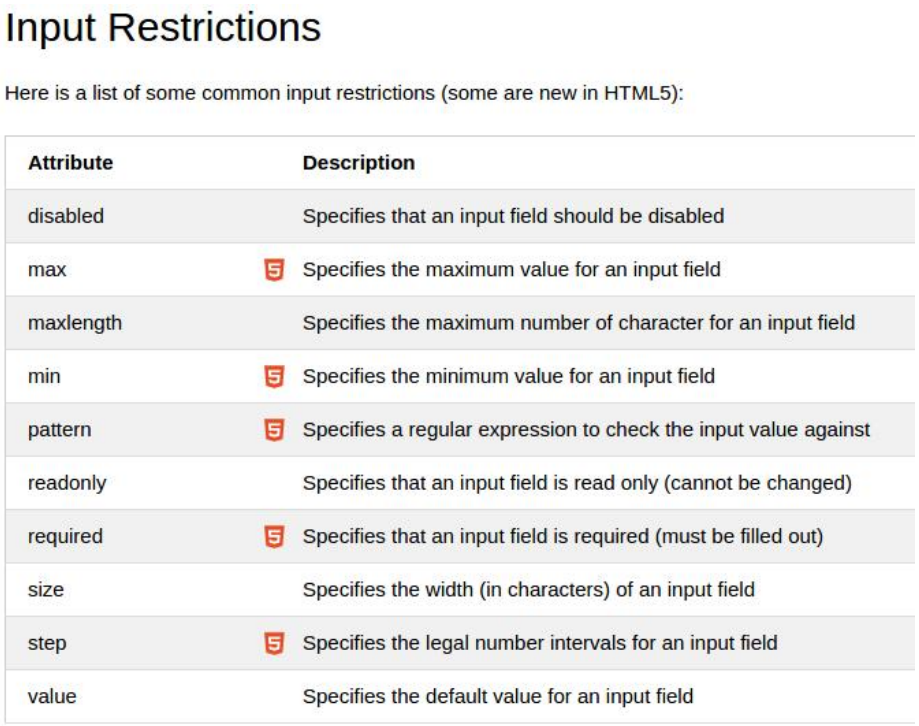
Delibery:
The student must show the page to the lecturer and to answer his questions.

Formularios HTML5

To create forms also we are going to use the Bootstrap library. The student must to get the hand with the available options:

- Visit the page *https://www.w3schools.com/bootstrap/bootstrap_forms.asp* and check the three possible forms available in Bootstrap: vertical, horizontal and inline.
- Visit the page *https://www.w3schools.com/bootstrap/bootstrap_forms_inputs.asp* and try all the examples.

HTML5 adds to the form controls the control of the user input data. In the figure 2.11 it is possible to see the available restrictions.



Input Restrictions

Here is a list of some common input restrictions (some are new in HTML5):

Attribute	Description
disabled	Specifies that an input field should be disabled
max	Specifies the maximum value for an input field
maxlength	Specifies the maximum number of character for an input field
min	Specifies the minimum value for an input field
pattern	Specifies a regular expression to check the input value against
readonly	Specifies that an input field is read only (cannot be changed)
required	Specifies that an input field is required (must be filled out)
size	Specifies the width (in characters) of an input field
step	Specifies the legal number intervals for an input field
value	Specifies the default value for an input field

Figura 2.11: User input available checks.

Práctica 4: formularios

Para que el alumno utilice los controles más comunes en HTML 5, se propone esta práctica. La práctica consiste en crear una página con varios formularios, según la figura figura 2.12.

Todos los controles deben tener un tooltip de Boostrap de ayuda:

```
data-toggle="tooltip" title="Write a description"
```

El script que procesará el formulario debe ser: *http://localhost/desweb/libro/webcontent/py/procFormRegistro.py*. El método de envío debe ser *post*. Pistas: buscar en internet los atributos de formulario *onsubmit* y *method*.

Nota: para el envío de formularios, se va a emplear funciones JavaScript y Ajax. Con esto se tiene la ventaja de poder modificar los datos enviados y controlar la respuesta, además de realizarse la operación en asíncrono.

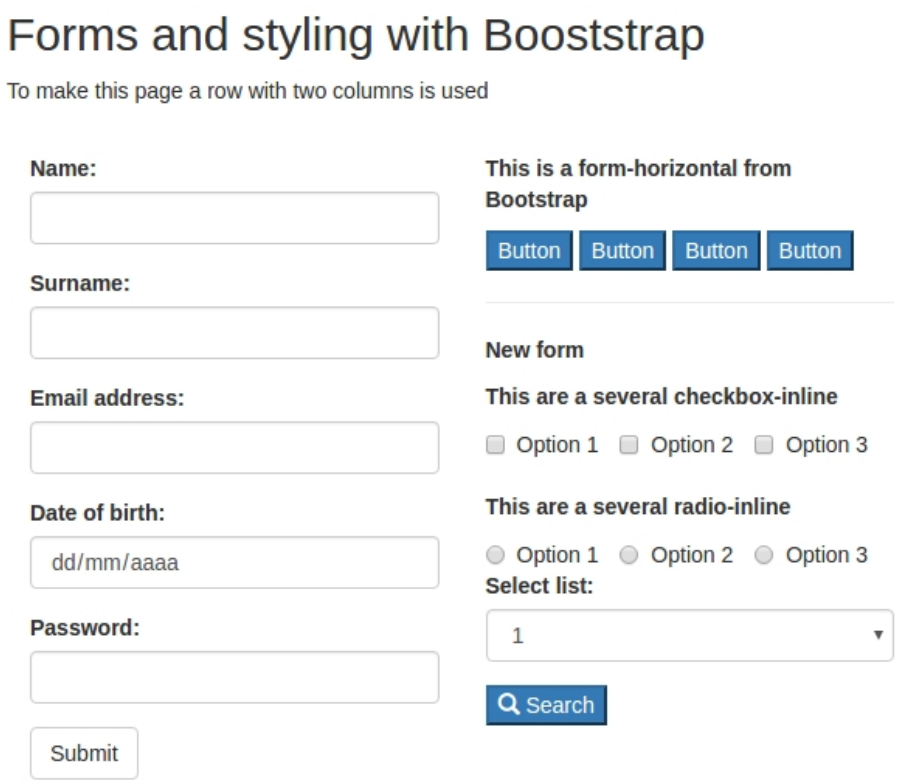


Figura 2.12: Formularios con HTML5 y Bootstrap.

JavaScript

JavaScript is used to give interactivity to the pages. We are going to use JavaScript to:

- Link events over objects to functions.
- Show or hidden parts of the page.
- Use OpenLayers and make pages with interactive maps.
- Use JQuery in order to request information to the server.

Visit the following pages, and check the examples, in order to know the basis of JavaScript:

- How to change html page objects: https://www.w3schools.com/js/js_intro.asp
- Where to put the JavaScript code, how to create functions and how to connect events with functions: https://www.w3schools.com/js/js_whereto.asp
- JavaScript output possibilities: https://www.w3schools.com/js/js_output.asp. Get the develop tools and check the console output.
- Comments are really important.

```
//one line comment

/*
```

```
/*
Several
line
comments
*/

//The recommended form to comment a function is the following
/**
* Change the message showed in the p_message element
* @method update_message
* @param {obj} obj_resp_json - object which have to has two
   properties: ok and mensaje.
*       ok can be true or false.
* @return none
*/
function update_message(resp_json){
    var obj_resp_json=$.parseJSON(resp_json);
    var obj_div=document.getElementById("div_message");
    var obj_p = document.getElementById("p_message");
    var cont;

    if (obj_resp_json.ok) {
        cont="<strong>Éxito!</strong> " + obj_resp_json.mensaje
        obj_div.className = "alert alert-success";
    }else{
        cont="<strong>Problema!</strong> " + obj_resp_json.
           mensaje
        obj_div.className = "alert alert-warning";
    }
    obj_p.innerHTML=cont;
}
```

- The JSon Format. We are going to use JSon format a lot because we are going to send and receive JSon. Is the most used form to communicate whit the server. Check the format in https://www.w3schools.com/js/js_json.asp

When events are linked to a functions, the page must have be loaded completely first. To ensure this the *window.onload* event must to be used.

```
function init(){
    document.getElementById("elemento").addEventListener("change",
       funcion_ejecutar);
}
window.onload = function() {
    init();
}
```

In the above listing, the function *init* only is executed when the document is completely loaded on the web browser. The *init* function links the html event objects with the functions.

The next listings presents a JavaScript program which sum two numbers. The first listing shows the html form definition. The second listing presents the code which do the necessary work. The result is presented in the figure 2.13 .

```
<div id=div_sumar>
    <form name="sumar" id="sumar">
        <h2>Sumar dos números</h2>
```

```
            <fieldset>
                <legend>Números a sumar:</legend>
                <label for="n1">Número 1: </label>
                <input class="ancho_100px" required type="number" id
                    ="n1" name = "n1"/>

                <label for="n2">Número 2: </label>
                <input class="ancho_100px" required type="number" id
                    ="n2" name = "n2"/>
            </fieldset>
            <br>
            <fieldset>
                <legend>Resultado:</legend>
                <label for="res">Suma: </label>
                <input class="ancho_100px" required type="number" id
                    ="res" name = "res"/>
            </fieldset>
            <br>
            <div class="centrado">
                <button type="button" id="calcular_suma" name ="
                    calcular_suma">Calcular</button>
                <button type="reset" id="limpiar_suma" name = "
                    limpiar_suma">Limpiar</button>
            </div>
        </form>
    </div>
```

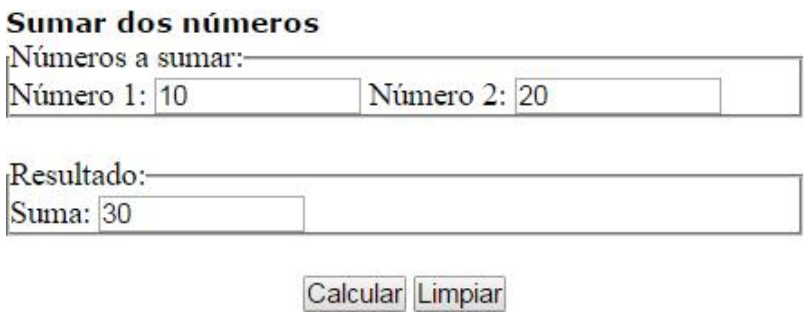


Figura 2.13: JavaScript program to sum two numbers.

The JavaScript code of the program is the following:

```
function sumar(){
    //This function is executed on click to the calculate button
    var n1=Number(document.getElementById("n1").value);
    var n2=Number(document.getElementById("n2").value);
    var resultado=n1+n2;
    document.getElementById("res").value=resultado;
}

function limpiar_suma(){
    //This function is executed on change n1 or n2
    document.getElementById("res").value="";
}
```

```
function init(){
    //gets the html objects and links the events with the functions
    document.getElementById("calcular_suma").addEventListener("click
        ", sumar);
    document.getElementById("n1").addEventListener("change",
        limpiar_suma);
    document.getElementById("n2").addEventListener("change",
        limpiar_suma);
}

window.onload = function() {
    //Executes the init function when the page was completely loaded
    init();
}
```

Exercise 5. JavaScript

The exercise has two parts. In the first part you have to do the following, figure 2.14:

- Create a html document with a control *input* of type *text*, to can to introduce a text.
- Add a button and a title *H1*.
- On doing click over the button, the title content have to change. The new content have to be the text introduced in the *input* of type *text* control.
- You have to link the button click event with the function which makes the change. You have to use the *addEventListener* of the object *document*.
- The event link have to be done from a function which is executed automatically when the document be loaded.
- The JavaScript code have to be outside the html document.

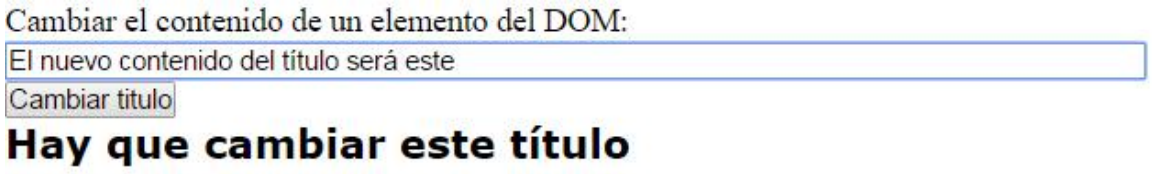


Figura 2.14: Change the content of a html element.

In the second part you have to create a form to calculate XY coordinates from polar coordinates figura 2.15. You have to make the following steps:

- Create a document with the form of the figure 2.15. Use the Boostrap library to style the html controls.
- Use the *onclick=radiar()* attribute in the *Calcular* button definition to connect the event click with the *Calcular* function. This is not the best way to do it but it is necessary to know. Somme times it is necessary its use.

- When a change is realized in the text box of input data, the previous calculated values are outdated, until the *Calcular* button was pressed. The outdated data have to be deleted automatically by using the change event on the text box of the input data. The change event of the four input text box must be linked to a function in charged of clean the out text box. To do this, you have to use the *addEventListener* method of the object *document*.
- The link of the events with the functions have to be done from a function which have to be executed when the page was totally loaded.
- Before to try to do the calculation, the data input have to be checked. If a non valid input is gave, an alert message have to be sowed before to try the calculation.
- The JavaScript code have to be outside the html document.
- Sow the development tools of the web browser. Put some debug stops and run the code, line after line. Check the local variable values in the functions.

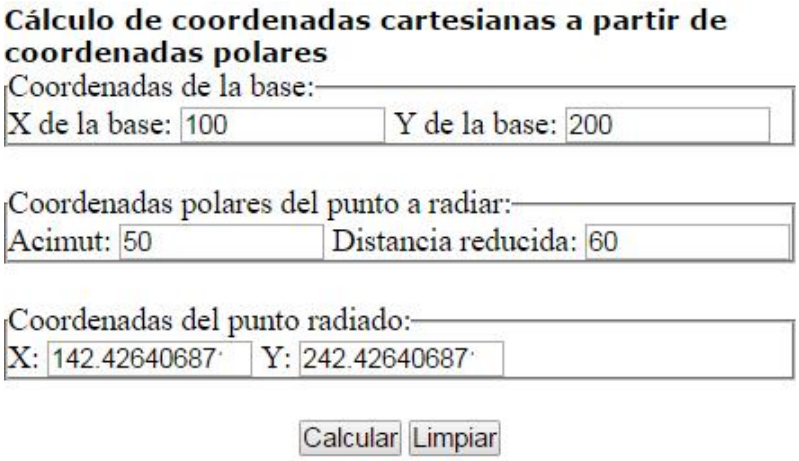


Figura 2.15: Point radiation application.

Useful code for the exercise:

```
//to access to a form element (f is the form  element)
var v=f.elements.namedItem("valor_campo").value;

//To know if a control is empty
if (v==""){
    //is empty
}

//To know if the content can be converted in a number (var num=
   Number(text))
if ((v=="")||(isNaN(v)){
    \\Is not numeric
}
```

Delivery:

It is not necessary to give any document. Only to show the page, make some calculations and answer some questions.

GPS: Geolocation interface

It is very easy to access to the GPS of the devices using geolocation interface. The problem is that, for security reasons the navigators disable the geolocation interface if in pages where the connection is not secure, that is, SSL connections. Have a look to https://www.w3schools.com/html/html5_geolocation.asp

CAPÍTULO 3

IDE configuration and *pozossan* projects installation

IDE configuration

To work with OpenLayers it is necessary to have some data installed. That data have to be in a PostGis database because our goal is to connect with PostgreSQL and update the database across Internet.

A database is prepared to be restored, with a point an a line string layer. The database schema is sowed in the figure 3.1

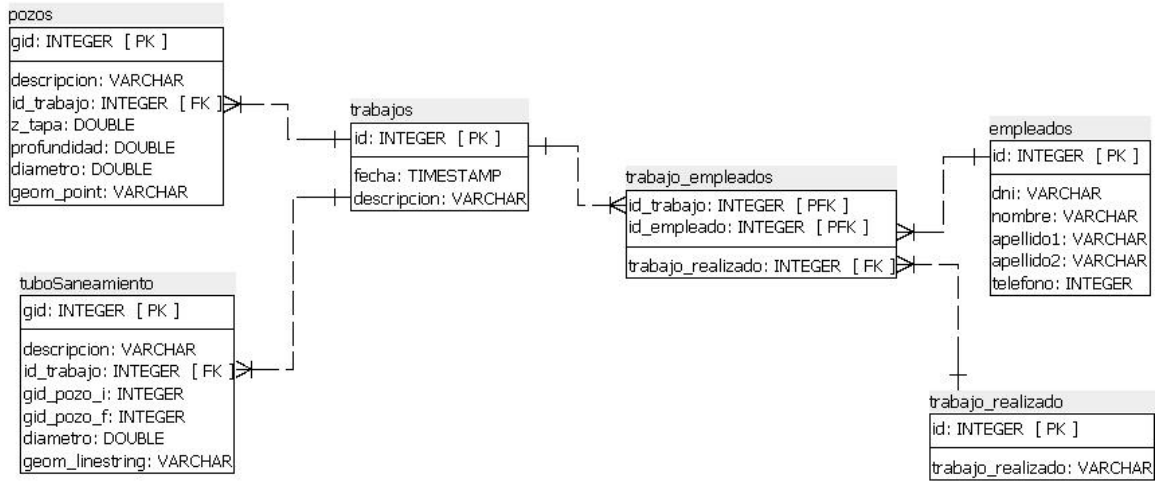


Figura 3.1: *pozossan* database relationship schema.

The database have to be restored in the computer of the student, and the *pozos* and *tubsosaneamiento* have to be published with GeoServer, with WMS and WFS services. To do this visit http://docs.geoserver.org/stable/en/user/gettingstarted/postgis-quickstart/index.html.

To download the database to be restored dowload the file 2017/pozssan_db/pozossan.backup in the resources of the subject, in Poliformat. To restore the database:

- Open PgAdmin, and create the database *pozossan*.
- Over the database, right button and select the option restore, and select the backup file.

To publish the layers *pozos* and *tubsosaneamiento* with GeoServer, with WMS and WFS services, visit
http://docs.geoserver.org/stable/en/user/gettingstarted/postgis-quickstart/index.html:

- The workspace have to be called: *pozossan*
- The data store have to be called: *pozossan*
- The layers have to be called: *pozos* and *tubosaneamiento*

pozossan project installation

We are going to learn web develop with examples contained in a project. The project has about 2500 lines between html, python, css and js code. The code have been divided in small separate files. The code is not going to be copied in this document. This document is going to do references to the files of the project. Then the project have to be installed in the computer of the student and be working.

We are going to use one project:

- A WSGI project, called *dw_pozossan_sessions*, accessible from the alias */pozossans/*.

The project is in 2017/pozossan_web_projects/ in Poliformat.

This WSGI alias have to be created for the projects:

```
WSGIScriptAlias /pozossans/ /var/www/apps/desweb/
    dw_pozossan_sessions/index.py
```

CAPÍTULO 4
OpenLayers 3.5

Configure JSONP in GeoServer

JSONP response is not enabled by default in Geoserver. To enable it some lines have to be uncommented in the file
C:/GeoServer2.6.3/webapps/geoserver/WEB-INF/web.xml in Linux or
/var/lib/tomcat7/webapps/geoserver/WEB-INF/web.xml in Linux. That is already done in the virtual machine of the subject.

```
<context-param>
    <param-name>ENABLE_JSONP</param-name>
    <param-value>true</param-value>
  </context-param>
```

JSONP is a request format which includes the name of the function which will receive the objects. See the parameter callback in the url property in the next listing:

```
/*Capa WFS pozos*/
var vs_pozos = new ol.source.Vector({
      loader: function(extent, resolution, projection) {
        var url = URL_WFS_POZOSSAN + 'service=WFS&' +
            'version=1.1.0&request=GetFeature&typename=pozossan:
               pozos&' +
            'outputFormat=text/javascript&format_options=callback:
               lf_pozos' +
            '&srsname=EPSG:25830&bbox=' + extent.join(',') + ',
               EPSG:25830';
        $.ajax({url: url, dataType: 'jsonp', jsonp: false});
      },
      strategy: ol.loadingstrategy.tile(new ol.tilegrid.XYZ({
        maxZoom: 19
      }))
 });
```

OpenLayers 3.5 map definition

To use OpenLayers it is necessary to load some libraries in the *head* of the page:

```
<!-- OPENLAYERS 3.5-->
<link rel="stylesheet" type="text/css" href="http://upvusig.car.upv.
   es/lib-js/ol3.5.0/css/ol.css" >
<link rel="stylesheet" type="text/css" href="http://upvusig.car.upv.
   es/lib-js/ol3.5.0/ol3-layerswitcher-master/src/ol3-layerswitcher.
   css"/>
<script type="text/javascript" src="http://upvusig.car.upv.es/lib-js
   /ol3.5.0/build/ol.js"></script>
<script type="text/javascript" src="http://upvusig.car.upv.es/lib-js
   /ol3.5.0/ol3-layerswitcher-master/src/ol3-layerswitcher.js"></
   script>
```

In the body, have to have a div which id=map. In that div the map will be draw.

```
<div id="map"></div>
```

The width and the height of the map are specified in the css file.

```
#map{width:100%;height:500px;background-color: #add8e6;}
```

To know how to load layers, to symbolize vector layers, etc, have a look to a the file *ol_pozossan.js*.

WMS geoserver Styles. SLD

There are examples, in the file textitol_pozossan.js, to symbolize vector layers, but any to symbolize WMS layers. WMS layers are not symbolized by OpenLayers, but GeoServer. Geoserver has powerful tools to symbolize using SLD (Style Layer Definition) files. See the page http://docs.geoserver.org/stable/en/user/styling/sld/cookbook/.

The student have to know how symbolize by field values, field ranges, label features, etc. There is some examples in Poliformat, in the folder 2017/geoserver_styles.

One common task is to draw the same layer with different styles, for example to label different field of the layer. In the next listing there is a example:

```
var red_wms_coste= new ol.layer.Tile({
    source: new ol.source.TileWMS(({
      url: URL_WMS_RED,
      params: {"LAYERS": "alginet_red:red", "TILED": "true" ,'
          STYLES': 'alginet_red_coste'},
    })),
    title: "Coste del tramo"
});

var red_wms_axtype= new ol.layer.Tile({
    source: new ol.source.TileWMS(({
      url: URL_WMS_RED,
      params: {"LAYERS": "alginet_red:red", "TILED": "true" ,'
          STYLES': 'alginet_red_axtype'},
    })),
    title: "Tipo de tramo"
});
```

Click, Selec, Draw and Snap interactions

You have documented functions of each map interaction in the following files. You have to know that the interactions are linked. Some interactions need that other interactions be removed and an others enabled.

The interactions select, modify, draw and snap are only applicable to a vector layers, not WMS layers.

map_click.js: Manages the map click event.

map_select.js: Functions to add and remove the select interaction. Select interaction is linked to a function which manages the event selection object. Inside the event selection object there are properties like the vector of elements selected. The function wich manages the select event is *opera_seleccion*, wich is in the file *manage_map_selection.js*.

map_modify.js Functions to add and remove the modify interactions. Needs that the select interaction be activated first.

map_draw.js: Functions to add and remove the select draw interactions.

map_snap_pozos.js: Functions to add and remove the snap to the *pozos* and *tubos* layer.

map_config.js: Functions which configure the interactions on the map. Are triggered by the click on the radio buttons of the page.

The link of the events with the functions and the execution of the functions to initialize the application is made from the file *inicializar.js*.

Exercise 6. Map drawing and modify.

The student already can advance his own project with this exercise, as long as to adds all the functionalities that are demanded here. In the student final geoportal, it is possible that the all the functionalities demanded here do not be applicable. The student then will have able disable that functionalities not applicable.

Follow the following steps:

- Create a new PostGis database.
- Add a polygon table, with some fields: area, perimeter, value, data, ...
- Add a linestring table with some fields. Add the field width.
- Add a point table with some fields.
- Create a map with some base layers. For Spain you can use the PNOA and Cadastre layers... Remember to use the same SRC for the layer in OL3 than the used to create the polygons table, in order to avoid reprojections.
- Add the tables as WFS layers to a map.
- Label at least one WFS layer by one field of the layer.

- Add a WMS layer from the table of lines. The line width of the lines will be in function of the with field value of the table. The widths have to be classified in three ranges. This have to be made with SLD symbology in Geoserver.
- Add select, modify, draw and snap interactions. The user have to have a three radio buttons to chose the action: select, draw or modify.
- Only will draw polygons.
- While drawing or modifying, the snap to the polygons layer have to be active.
- The elements drew have to belong to a new vector layer.
- On each selection the coordinates of the polygon selected have to be showed in a text area control.
- All the new elements drew are temporal. Do not are in the database. If you reload the page, the temporal elements will be deleted. Put a button to clean the temporal layer, in order not to have to reload the page to clean de map. To do this you will find a example at the end of the file *ol_pozossan.js*.
- Add the layer switcher control.
- Add the cursor coordinates to the map.

Delibery:
The student must show the page to the lecturer and to answer his questions.

CAPÍTULO 5

Dynamic sites with Python and WSGI

The server programming part, called *backend*, its going to be done with Python 2.7, with WSGI applications. One application WSGI is a Python program connected to Apache server with a WSGI directive. Apart of this is a normal Python program.

Python in Linux has some differences with Windows. The folder *c:/python27/lib/site-packages* in Windows is located in */usr/lib/python2.7/dist-packages* in Linux. Our new common Python libraries have to be placed in a package in this folder. Later it is necessary to grant read permission to the group www-data in this folder, in order to allow Apache to read it. The *dweb* library, which will be used later, is already in */usr/lib/python2.7/dist-packages* and have all the permission granted.

Before start programming with Python

Debugging Python code. File error.log

If the navigator shows 500 server error, means that there is a error in the Python server code. In this cases it is necessary to see the file */var/log/apache2/error.log*, where, in the last lines, the error description in written.

Mostrar errores en el navegador

The *paste* Python library (*https://pypi.python.org/pypi/Paste*), writes the error in the web browser, and avoids to have to open the file */var/log/apache2/error.log*. This library is already installed in the virtual machine of the subject.

This library only have to be used in the case of develop, not in production.

To use the library two lines have to be written bellow the wsgi function:

```
def application(environ, start_response):
    wsgi code
    return str(html)

#Uncoment to debug. Comment to production
from paste.exceptions.errormiddleware import ErrorMiddleware
application = ErrorMiddleware(application, debug=True)
```

Remote debugging with PyDev

In general, it is necessary, each time the Pyhton code be changed, restart de Apache service. In some cases you will not know what is going wrong in the server code, despite the error messages. In this cases, it is necessary to stop the execution in a line of code and execute the code line after line, seeing the variable values. This is remote debugging.

To remote debug with PyDev:

- Add, where you want stop the execution, the instructions of the following listing:

```
#import sys;sys.path.append(r'/opt/liclipse/plugins/org.
   python.pydev_4.5.5.201603221237/pysrc')
#import pydevd;pydevd.settrace()
```

  Simply writing *pydevd*, Liclipse writes the rest of the sentences.

- In the *Debug* perspective, run the PyDev server.
- Execute the page with a web browser.

With the before steps, the execution is stopped in the line under *pydevd.settrace()* sentence, and it is possible to advance line by line in the code, seeing the variable values.

Importing modules from a WSGI application

Suppose that you have a module called *m1.py*, and other module called *m2.py*, inside the package *py*. The *py* package in where the module *m1.py* is. The following import will work in a normal python program;

```
from py import m2
```

The above import works with standard Python, but not with WSGI applications, due to that the path to *m1.py* is not added to a the *pythonpath* variable. It is necessary to do it manually:

```
import sys, os
DIR_BASE=os.path.dirname(__file__)
sys.path.append(DIR_BASE)
```

Now *from py import m2* will work because the path to *m1.py* module in the *pythonpath*, and the *py* package is in the folder where *m1.py* is. The above listing have to be added in all the principal wsgi modules, usually called *index.py*.

WSGI application example: *dw_wsgi_example*

- Create with Liclipse a PyDev project called *dw_wsgi_example*, inside the workspace *c:/ms4w/apps/desweb*.
- Pay attention in that Liclipse creates a module called *__init__.py*. This module converts the folder *dw_wsgi_example* in a package.
- Create the module of the wsgi web application. The module have to be called *index.py*, despite that the name can be any and the extension also any.
- Create a package, called *py*. This package will have Python modules that will be loaded from the wsgi application.
- Create an alias for Apache, */dw_wsgi_example/*, to load the page with the url: *http://localhost/dw_wsgi_example/*.

Create the wsgi application index.py

Paste in the *index.py* file the following code:

```
# -*- coding: utf-8 -*-
"""
Created on 12 de abr. de 2017
@author: Joaquín Gaspar Mora Navarro
"""

#this is necessary to have able to import other modules Python
import sys, os
dir_base=os.path.dirname(__file__)
sys.path.append(dir_base)

#global variable used to indicate if we are developing or not
DEBUG=True

def application(environ, start_response):

    html="""
<!DOCTYPE HTML>
```

```
<html lang="es">
<head>
    <meta charset="UTF-8"/>
    <base href="http://localhost/static/dw_wsgi_example/"/><!--
       Directorio base para las rutas a los archivos de esta página
       -->
    <meta name="description" content="Asignatura Desarrollo Web y
       Geoportales" />
    <meta name="author" content="Gaspar Mora-Navarro" />
    <meta name="keywords" content="máster, geomática, geoinformación
       , upv, geoportal, mapserver, openlayers, javascript"/>
    <title>Geopotal de la base de datos POZOSSAN</title>
    <link href="css/estyles.css" rel="stylesheet" type="text/css
       "><!--Indica cómo se muestran los elementos y su distribución
       -->
    <script type="text/javascript" src="js/code.js"></script>
</head>
<body>
    <div id="div1">
        <p>This is the content of the div1</p>
        <p>This is the content of the div1</p>
        <p>This is the content of the div1</p>
        <p>This is the content of the div1</p>
    </div>
    <div id="div2">
        <p>This is the content of the div2</p>
        <p>This is the content of the div2</p>
        <p>This is the content of the div2</p>
        <p>This is the content of the div2</p>
    </div>

    <button id="show_div1">Show div1</button>
    <button id="show_div2">Show div2</button>

</body>
</html>
"""
    #Envío al servidor. Siempre igual.
    status = "200 OK"
    response_headers = [("Content-Type", "text/html"),("Content-
       Length", str(len(html)))]
    start_response(status, response_headers)
    return [str(html)]

if DEBUG:
    from paste.exceptions.errormiddleware import ErrorMiddleware
    application = ErrorMiddleware(application, debug=True)
```

The project uses two static files: js/code.js and css/styles.css, with are locates because the label *<base>* from the *<head>* section. It is necessary to create a static Liclipse project, called *dw_wsgi_example/*. In the folders *js* and *css* paste the following content:

File css/styles.css

```
@CHARSET "UTF-8";
```

```
#div1{background: green}
#div2{background: blue}
```

File js/code.js

```
function show_div1(){
    document.getElementById("div1").style.display = "block";
    document.getElementById("div2").style.display = "none";
}

function show_div2(){
    document.getElementById("div1").style.display = "none";
    document.getElementById("div2").style.display = "block";
}

function init(){
    //web section navigation. File: section_navigation.js
    document.getElementById("show_div1").addEventListener("click",
       show_div1);
    document.getElementById("show_div2").addEventListener("click",
       show_div2);
}

window.onload = function() {
    init();
};
```

In this simple wsgi application you have many important things: one part, in Python, that generates the html string that is sent to the client. That string is a complete html page. Pay attention that, in the <head>section there are several links to the static part of the project: *js/code.js*, and *css/styles.css*. Apache has access to that files because the *<base>* label, which uses the */static/* alias, that drives Apache to the folder */var/www/apps/static/*.

The WSGI applications have always a point of entry, which is the function *application*, in the file *index.py*. The *application* function receives two parameters, which will be described later.

Check the applicartion: *http://localhost/dw_wsgi_example/*

The js code shows or hides the elements from the html code. This is the technique frequently used to navigate by the page. The full page is sent to the client, and the adequate sections are sowed in depending of the menu option selected.

You can download the application in Poliformat, in the resources of the subject, in 2017/ejemplos_clase/wd_wsgi_example.zip.

Divide and you will win

Programming is a challenge of organization. Is you arrive to a point where you can not extend the program, or the cost in do it is too big, is because the program is not well organized. A good organization requires the division of the project in many independent, small parts. This is the key.

In the following project a way of how to divide the HTML code is presented: sections and forms. Each part is a independent file. All the files are joined in one file called *base.html*. In the following listing a base file example is showed:

```
<!DOCTYPE HTML>
<html lang="es">
<head>
```

```
    {{head_content.html}}
</head>
<body class="container">
    {{header.html}}
    <section id="sec_mapa">
        <row class="row">
            <aside id="aside_izquierdo" class="col-xs-3">
                 {{div_form_ini_session.html}}
            </aside>
            <section id="sec_cuerpo" class="col-xs-7">
                <div id="map"></div>
            </section><!--cuerpo-->
            <div id="div_aside_principal" class="col-xs-2">
                {{aside_principal.html}}
            </div>
        </row>
    </section><!-- sec_mapa -->
</body>
</html>
```

This technique is copied from the Django framework. Django call this separate html files *templates*. Actually is a substitution string technique. The content of the base.html file is loaded in a string, and the {{file_names.html}} sub strings are replaced by the real content of the corresponding file. The result is one big html file divided in small separate files, and joined in the base.html file.

To facilitate this division in small files, in the *dw_pozossan_sessions* project, in the *templates* package is the *mount_page.py* module, which do precisely that. Receives the base file name y a list of strings, which represent folders. The *mount_page* function loads all the files and search the name of each file in the base file. If there is a match replaces the file name by the file content. In the following listing there is a example:

```
dir_base=os.path.dirname(__file__)
base_file=dir_base + '/sections/base.html'
print base_file
sections=dir_base + '/sections'
forms=dir_base + '/forms'
list_folders_templates=[sections,forms]
html=mount_page(base_file,list_folders_templates, False,'')
print html
```

In the figura 5.1 you can see the folder and file structure.

In the resources of the subject, in Poliformat, in the folder *ejemplos_clase*, there is the file *mount_page.zip*, ready to be checked. This is not a wsgi application is a normal Python program.

The function *mount_page* uses other functions, which are installed in a module called *dweb*, installed in the package *dweb*, which is in the *dist-packages* folder, in the virtual machine of the subject.

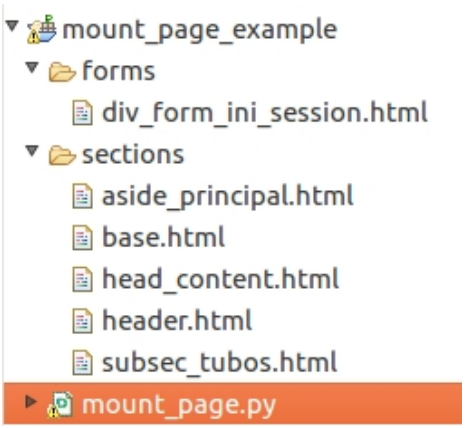


Figura 5.1: Division of a page in small files.

Exercise 7. Division of a HTML page in small parts

The student already has to work with his own project and apply this exercise. It consists in to divide the web page in small parts, the parts which the student think: sections, forms, head, header, footer, ... To do it:

- Create a wsgi project.
- Create a folder called *templates*, and inside two sub folders *secctions* and *forms*.
- Divide the page in parts, the parts which the student think: sections, forms, head, header, footer, ...
- The page has to have a navigation menu and, on click over it, different sections have to appears and disappears. You have to use the technique *style.display*, used in the section sección 5.2.1, página 38.
- Has to have at least a form in the page, to the session initialization. It has to have a option in the menu to sow or hide the form. .
- The page has to have a OpenLayers map. This map could be the map created in the before exercise.
- Has to have a help section. When the help option in the menu be clicked, the map section disappears an the help section appears.

Delivery: You have to sow the project working to the teacher and answer some questions.

Send and process GET data in a wsgi application

Manage GET data in the urls

The data specified in the urls is caller GET data.

In many occasions hiperlinks are used on order to show different parts of the page. For example: http://www.site.com/alias/?option=init, or http://www.site.com/alias/?option=init-session.

As you see, in both cases the same alias is used in the url. That means that the same wsgi application will be executed. Inside of, wsgi function, in the file index.py, the chosen option is obtained and

the adequate answer is generated. To know the chosen option the package *urlparse* is used, able in the Python 2.7 standard installation. In the following listing a use example is showed:

```
C:\Postgres91\bin>python
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (
   Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
   information.
>>> import urlparse
>>> a=urlparse.urlparse("http://localhost/geops/?aplicacion=inicio")
>>> print a
ParseResult(scheme='http', netloc='localhost', path='/geops/',
   params='', query='aplicacion=inicio', fragment='')
>>> a.scheme
'http'
>>> a.query
'aplicacion=inicio'
>>> b=urlparse.parse_qs(a.query)
>>> b
{'aplicacion': ['inicio']}
>>> b['aplicacion']
['inicio']
>>> b['aplicacion'][0]
'inicio'
>>>
```

In the following code a function which returns a Python dictionary with the values of a GET request. This is the method commonly used in the hyperlinks with parameters: http://www.site/alias/?variable1=value1&variable2=

The *environ* parameter is given by Apache to the wsgi function *application*. Inside the *environ* parameter there are a lot of information: the GET string, IP of the client. web browser name, ...

```
from urlparse import parse_qs #librería estándar de python
def devuelve_dict_get(environ):
    qs = environ['QUERY_STRING']
    d=parse_qs(qs, True)#el parámetro opcional true conserva los
       valores en blanco
            #la llave estará pero vale ''
    return d
```

The use of the above function is as is sowed in the following listing. In the next example the url http://www.site/script/?variable1=value&variable2=value2 is passed to the server.

```
from dweb import dweb
d=dweb.devuelve_dict_get(environ)
print d
{'variable1':[value1],'variable2':[value2]}

print d['variable1'][0]
value1
```

The function *devuelve_dict_get* resturns a normal Python dictionary. Note that the values of each key are inside a list, then, to access to the values, you have to access to the element 0 in the list (d["variable1"][0]).

To extract the values from the dictionary is better to use the method *get*, that all Python dictionaries have, because d["variable1"][0] raise a error in the case the key "variable1" does not exist. See the following code:

```
from dweb import dweb
d=dweb.devuelve_dict_get(environ)
print d
{'variable1':[valor1],'variable2':[valor2]}

print d['variable555'][0]
Error the key 'variable555' does not exists in the dictionary d
```

The *get* method in the dictionaries allows, in the case of the key does not exist, to return an other specified values in the method. See the following example:

```
from dweb import dweb
d=dweb.devuelve_dict_get(environ)
print d
{'variable1':[value1],'variable2':[value2]}

print d.get('variable555',[''])[0]
Returns '' without error, because the key 'variable555' does not
   exists
```

Server data send from html, using GET method

In the urls you can send data to the server using keys and values. This method of data send is called GET. This keys and values, in the url, indicate to the server what the client wants. For example, you can send to the server the variable *operation*, specifying one of the values *insert*, *select*, *update* or *delete*. The wsgi application retrieves the *opetation* variable value, and executes a function that generates the adequate answer. See the following url examples:

```
- To insert a reccord
http://localhost/myalias/?operacion=insert&tabla=d.empleados&pk_name
   =id&registro={nombre=juan, dni=25252525}

- Para seleccionar un registro
http://localhost/myalias/?operacion=select&tabla=d.empleados&pk_name
   =dni&pk_value=25252525&campos={id, nombre,dni}

- Para editar un registro
http://localhost/myalias/?operacion=update&tabla=d.empleados&pk_name
   =dni&pk_value=25252525&registro={nombre=felipe}

- Para borrar un registro
http://localhost/myalias/?operacion=delete&tabla=d.empleados&pk_name
   =dni&pk_value=25252525
```

Example of processing GET data in order execute the adequate Python function

In the server *myalias*, indicates Apache which wsgi application to execute. In the wsgi application, to decide what to do in the case of each url of the previous section (5.5.2), it is necessary to do something like:

```
import json#to make json strings from python dictionaries
from dweb import dweb
from py import operations

def application(environ, start_response):
    dic_get = dweb.devuelve_dict_get(environ)
```

```
operacion=dic_get.get('operacion',[''])[0]
tabla=dic_get.get('tabla',[''])[0]
pk_name=dic_get.get('pk_name',[''])[0]
pk_value=dic_get.get('pk_value',[''])[0]
registro=dic_get.get('registro',[''])[0]
campos=dic_get.get('campos',[''])[0]

#in the case in which a key of the dictionary does not exists in
    the dictionary, the value will be '', but any error is
   generated

if operacion=='insert':
    resp_json=operations.insert(tabla,pk_name,registro)
elif operacion=='select':
    resp_json==operations.select(tabla,pk_name, pk_value, campos
       )
elif operacion=='update':
    resp_json==operations.update(tabla,pk_name, pk_value,
       registro)
elif operacion=='delete'
    resp_json==operations.delete(tabla,pk_name, pk_value)
else:
    d={}
    d['ok']=False
    d['json']='vacio'
    d['mensaje']='<p>Any valid operation</p>'
    resp_json=json.dumps(d)

status = '200 OK'
response_headers = [('Content-Type', 'text/html'),('Content-
   Length', str(len(resp_json)))]
start_response(status, response_headers)
return [str(resp_json)]
```

Manage JSON strings

In the previous listing is introduced a new important element: the send of json answers to the client. A json answer is a string that inside has a dictionary key: value. From JavaScript it is possible to convert json strings to objects. That objects have properties which are called like the keys of the json dictionary, and the values of that properties are the values of the json dictionary. **This technique, the json strings, allow multiple data send to the client in each answer**.

In the next listing a example is presented:

Listado 5.1: Crear cadenas json con python

```
import json#to make json strings from python dictionaries

d={}
d['ok']=False
d['json']='vacio'
d['mensaje']='<p>Any valid operation</p>'
resp_json=json.dumps(d)
```

In the above example, a Pyhton dictionary is created. This dictionary is converted in a json string. This string is stored in the *resp_json* variable, whic will be send to the client. The content of the *resp_json* variable is:

```
"'ok':False,'json':'vacio','mensaje':'<p>Any valid operation</p>'"
```

Is a json string is received in the client, with JavaScript it is possible convert it into a object. In the following code, a example is showed:

Listado 5.2: Convert json string to object in JavaScritp

```
function functionWichReceivesTheAnswer(resp_json){
    var obj_resp_json=$.parseJSON(resp_json);//$ mins JQUERY. It is
       necessary to load JQUERY library before
    //access to the field values
    alert(obj_resp_json.ok);//shows False
    alert(obj_resp_json.mensaje;//shows 'Any valid operation'
}
```

From the same form, it is possible to send a string to the server from JavaScript. To create a json string with JavaScript:

```
var str_json= '{"field1":"value1","field2":"value2"}'
```

When a json is received from Python, it is possible to convert it into a Python dictionary. In the following code there is an example:

Listado 5.3: Convertir cadena json en diccionario con Python

```
import json#to make json strings from python dictionaries
var dic_json= json.loads('{"field1":"value1","field2":"value2"}')
print dic_json["field1"]
```

A very common operation is to have to create a json string whit all the values of a HTML form. This task can be done easily with the function *formToJSONString*. This function is in the file *general_code.js* in the *dw_pozossan_sessions* project.

Sending data to the server and to receive data from the server with Ajax

Ajax allows to send data to the server and to retrieve data from the server, without to send or receive the complete page. In addition, is executed on asynchronous form, that is, while the page is sending and receiving data, the page is not locked, continues working.

The library JQuery has to be loaded (*https://jquery.com*). To load this JavaScript library, you have to write in the *head* of the document:

```
<script type="text/javascript" src="http://code.jquery.com/jquery
   -2.1.3.min.js"></script>
```

In the section 5.5.2, it was sowed how to send data to the server using urls. This can be done with hiperlinks:

```
<a href="http://localhost/myalias/?operacion=select&tabla=d.
   empleados&pk_name=dni&pk_value=25252525&campos={id, nombre,dni}">
   Seleccionar el empleado DNI=25252525</a>
```

On pressing over the hiperlink, the GET request in sent, and the server generates a answer which could be showed in a new page. This is not very useful. The correct form is to have able to send data to the server, then the server do the tasks requested and send the answer to the client. The client, on the other hand, once received the answer, instead of change the complete page, makes the changes in

the same page. Therefore, the user could not lock the page, that is, all this process have to be done in asynchronous mode. All this process is what Ajax do. Ajax is a method of the JQuery JavaScript library.

The Ajax syntax is the following:

```
var URL="http://localhost/myalias/?";
var datos="operacion=select&tabla=d.empleados&pk_name=dni&
   pk_value=25252525&campos={id, nombre,dni}"
$.ajax({
    type: "GET",//mode of sending data. Can be 'GET' or 'POST'
    url: URL,//url where the data is sent
    data: datos, //This is the data sent to the server.
    success: update_message //function that will receive the
       data answer, and will be executed when the data will be
       received
  });
```

The function *update_message* will be execute on automatic form. That function have to receive one parameter, which is the server answer. Usually the response will be a *json* string. In the following listing there is an example:

```
/**
* Change the message showed in the p_message element, and the color
   from the div whic contains the p_message element, called
   div_message
* @method update_message
* @param {obj} obj_resp_json - object which have to has two
   properties: ok and mensaje.
*       ok can be true or false.
* @return none
*/
function update_message(resp_json){
    var obj_resp_json=$.parseJSON(resp_json);
    var obj_div=document.getElementById('div_message');
    var obj_p = document.getElementById('p_message');
    var cont;

    if (obj_resp_json.ok) {
        cont='<strong>Éxito!</strong> ' + obj_resp_json.mensaje
        obj_div.className = "alert alert-success";
    }else{
        cont='<strong>Problema!</strong> ' + obj_resp_json.mensaje
        obj_div.className = "alert alert-warning";
    }
    obj_p.innerHTML=cont;
}
```

In the Poliformat subject resource, in *2017/ejemplos_clase/dw_ajax.zip*, there is an example. Install the example and check how it works. In order the example cloud work, you have to create the Apache alias */dw_ajax/* for the wsgi application.

Exercise 8. Ajax

The student have to do the following tasks in his own project, or in a new project. The goal is that the student could learn:

- To send json strings to the server with Ajax.
- To process GET request from the server.
- To generate json answers from the server.
- To receive the json server answer in a JavaScript function and make change in the page.

The communication client-server by using Ajax-json is the most important knowledge in this subject.

In the next exercise the student will make changes in the data base with the data of the forms in this Exercise. Then, the student have to be organized and to try that this Exercise be useful to his own project or the next exercise.

- Create a new WSGI application, or use the application of your own project, and divide the HTML code in different HMML files.
- One of the HTML files have to have a HTML form.
- The application, on the firs time that the client request the page, have to send the complete page.
- You have to create operations, in the wsgi application, in order to the application reacts to the GET requests. At least have to have one operation that call to one Pyhton function that manage the form. In the GET data, of the request that the page sends to the server, have to be the operation name, the form name and the data form. The data form have to be sent to the server, in the GET data of the request, in a string json format.
- To create the json string with the form data in JavaScript, you have to use the function *formToJSONString*. That function makes the task for you. The *formToJSONString* functionis in the file *general_code.js* of the project *dw_pozossan_sessions*.
- The server have to do some operation whit the form data, therefore you have to use the Python *json* library, in order to transform the json string of the form data, received by GET, in a Python dictionary. The server have to do some operation with the form data, in order the teacher has able to know that the student knows to process the form data. You can transform the form data to upper-case, to sum some values,... In the next exercise, this data form will be used to insert, update or delete some row in the database.
- Once processed the data form in the server, you have to generate a Python dictionary, with several fields, and send it to the client in a json string.
- In the client side, from JavaScript, you have to process the json string received from the server, transform it in a object, and give messages on the page with the data answer.
- Para el envío de los datos al servidor y la gestión de la respuesta, se debe usar Ajax, o la función *mi_ajax*, contenida en el archivo *mi_ajax.js* del proyecto *dw_pozossan_sessions*.
- To send the data to the server, and the server answer management, you have to use Ajax, or the function textitmi_ajax, from the file *mi_ajax.js* from the project *dw_pozossan_sessions*.

For example, you can send a form which match with the *d.edificios* (d.buildings): a description, the geometry type (always Polygon), and the polygon complete string coordinates x,y,x,y, ..., in the OpenLayers format. In the server, you transform the description field to upper-caps and transform the coordinates into PostGis format: x y, x y, ... You have to build a dictionary with the transformed data, transform it into a json string and send it to the client. The client shows the same values, in the same form, but transformed by the server (upper-caps, and the transformed string coordinates).

Para cargar una cadena json en un formulario, leer la sección 5.10.2 55. To load into a form a json string whit the form data, see the section 5.10.2, page 55.

To make changes in PostgreSQL with Pyhton. Use of the *psycopg2* library

To make queries or changes in a PostgreSQL database with Pytjon, we are going to use the *psycopg2* library,*http://initd.org/psycopg/*. This is the penultimate step to complete the student training: to make changes in a spatial database.

Lo primero que hay que hacer es configurar psycopg2 para que, ya que nuestras bases de datos están en UTF8, que nos devuelva las cadenas con esa codificación.

The first you have to do is to configure the psycopg2 library, because our databases are in UTF8 coding:

```
import psycopg2
import psycopg2.extensions
psycopg2.extensions.register_type(psycopg2.extensions.UNICODE)
psycopg2.extensions.register_type(psycopg2.extensions.UNICODEARRAY)
```

Now you can connect with the pozossan database:

```
database='pozossan'
user='postgres'
password='postgres'
host='localhost'
port=5432

#conexion
conn=psycopg2.connect(database=database, user=user, password=
   password, host=host, port=port)
```

The constructor of the object *connect* created a object, which has been denominated *conn*. This object has a property, called *cursor*, which is an other object which allow to send SQL queries to the database.

```
cursor=conn.cursor()
```

With the *cursor*, it is already able to send SQL sentences to the database. The sentences can have parameters or not. If the sentence does not have parameters then is a normal string. In the following examples, it is assumed that the variables *conn* y *cursor* have been created before as global variables.

Table creation

In the following example a table is created from Python code:

```
def crea_tabla(nom_tabla):
    """
    The queries where there are not field values are created with
       strings' normal techniques
    """
    sql_tabla="create table {0}(gid serial primary key, descripcion
       text, area double precision)".format(nom_tabla)
    sql_geom="select addgeometrycolumn('d','{0}','geom',25830,'
       POLYGON',2, true)".format('edificios')
    print 'Creando la tabla'
    print sql_tabla
    print 'Añadiendo columna de geometría'
    print sql_geom
    cursor.execute(sql_tabla)# adds the sentence to the transaction
    cursor.execute(sql_geom)# adds the sentence to the transaction
    conn.commit()# executes the transaction. IT IS NECESSARY. IF NOT
        NOTHING WILL CHANGE IN THE DATABASE
```

As you can see in the above listing, after each sent sentence, it is necessary to call the method *commit* from the object *conn*, in order to make the changes in the database. If you do not call this method, any change will be done in the database.

On executing the *crea_tabla* function, the following result is obtained.

```
Creando la tabla
create table d.edificios(gid serial primary key, descripcion text,
   area double precision)
Añadiendo columna de geometría
select addgeometrycolumn('d','edificios','geom',25830,'POLYGON',2,
   true)
```

In the previous function, and in the following ones, it is going to be used variables with table names, field names and field values. The field values usually are also parameters of the functions. This is done like that in order to the student learn how it is done, being that, without using this techniques it is impossible to automate nothing.

When you have finished of using the connexion, it is very important to close it, to not to use resources on the server:

```
cursor.close()
conn.close()
```

Once the connexion is closed, the variables *cursor* ni *conn* can not be used any more.

To insert rows with geometry

In the following listing, a form of to insert geometries in tables with a field geometry is presented. The trick consists in to use the PostGIS function *st_geometryfromtext*, and transform the geometry, in a text coordinates, to a valid column geometry type. The previous function also needs the SRC, specified in the EPSG code:

```
st_geometryfromtext(geometria_en_texto, EPSG)
```

In the following listing a function that inserts building's geometries, int the previously created table *d.edificios*, is presented:

```
def inserta_poligonos(nom_tabla, descripcion, pol_wkt):
    """
    The queries where there are field values are generated with the
       same system, using string, but you NEVER have to specify the
       values inside the string. You have to put '%s' instead the
       real field value. The field values are specified in a vector
       as a second parameter of the execute function
    """
    cons_ins='insert into {0} (descripcion,geom) values (%s,
       st_geometryfromtext(%s,25830))'.format(nom_tabla)
    print 'Insertando polígono'
    print cons_ins
    cursor.execute(cons_ins,(descripcion,pol_wkt))
    conn.commit()
```

As you can see in the above listing, the field values position are specified in the string query, using *%s*. The real values are specified as second parameter of the *execute* method, inside of a list or *tuple*. The execute method introduces the real values in the query string, replacing each %s by his real value.

In the next listing a function use example is showed:

```
# Pay attention in that, in the case of polygons, the last point
   have to be the same as the first
p1='POLYGON((727844 4373183,727896 4373187,727893 4373028,727873
   4373018,727858 4372987,727796 4372988,727782 4373008,727844
   4373183, 727844 4373183))'
p2='POLYGON((727988 4373188,728054 4373192,728051 4373093,727983
   4373093,727988 4373188))'

inserta_poligonos('d.edificios','Primera descripcion', p1)
inserta_poligonos('d.edificios','Segunda descripcion', p2)
```

The execution of the above listing generates the following result:

```
Insertando polígono
insert into d.edificios (descripcion,geom) values (%s,
   st_geometryfromtext(%s,25830))
Insertando polígono
insert into d.edificios (descripcion,geom) values (%s,
   st_geometryfromtext(%s,25830))
```

Updating rows

In the following listing a example of how to update rows is sowed. The listing present a function which updates de *area* field of the buildings table. The function is a example of how to build queries dynamically, using variables which contain the name of tables and fields.

```
def actualiza_areas(nom_tabla,nom_campo_area, nom_campo_geom):
    cons='update {0} set {1}=st_area({2})'.format(nom_tabla,
       nom_campo_area,nom_campo_geom)
    print 'Actualizando areas'
    print cons
    cursor.execute(cons)
    conn.commit()
```

In the following listing the above function is used.

```
actualiza_areas(nom_tabla='d.edificios', nom_campo_area='area',
   nom_campo_geom='geom')

Actualizando areas
update d.edificios set area=st_area(geom)
```

Selecting rows

To retrieve the database data, a sentence *select* have to be used. The results remain stored in the *cursor* variable used to execute the *select* sentence. The *fetchall* method returns all the records in a *list* of *tuples*, and empties the cursor. That means that the records can not be retrieves again. Each row selected in the *select* query is represented in a tuple, inside the list returned by *fecthall*.

A tuple is a immutable list, which is used like a list to retrieve its elements. The first tuple element contains the field values in the order done by *select* query.

The *'u* that appears before the string values indicates that is a UNICODE string, that is, the string can contains Latin characters.

In the below example a function that selects and returns some rows from the *d.buildings* is showed. The function parameter description is inside the function.

```
def selecciona_datos_edificios(cond_where=None, lista_valores=None):
    """
    Selects some rows from the d.buildings table.
    Use examples:
        selecciona_datos_edificios()#selects all rows
        selecciona_datos_edificios('nombre=%s and edad<%s', ['juan',
            40])
        selecciona_datos_edificios('gid=%s', [8])#selects only the
           row which gid=8

        cond_where: string with the row's accomplish requirements.
           YOU CAN NOT SPECIFY THE QUERY FIELD VALUES DIRECTLY.
           NEVER DO THAT. Instead you have to put a '%s' in the
           field value position.
        lista_valores: List which has the query field values, in the
            order in which they appears in the query.
    returns: A list of tuples with the selected rows.
        [(row1),(row2),(row3), ...]
        Each row conains:
            (gid, descripcion, geometría en formato texto) -->
            (1, u'Primera descripcion', u'POLYGON((727844
               4373183,727896 4373187,727893 4373028,727873
               4373018,727858 4372987,727796 4372988,727782
               4373008,727844 4373183,727844 4373183))'
    """
    cons='select gid, descripcion, st_astext(geom) from d.edificios'
    if cond_where <> None:
        cons += ' where ' + cond_where
        cursor.execute(cons,lista_valores)
    else:
        cursor.execute(cons)
    print 'Consulta: ' + cons
    return cursor.fetchall()
```

In the following listing, a function which retrieves the rows selected is showed:

```
def print_registros_completos(lista_registros):
    print "Full rows"
    for registro in lista_registros:
        print registro

def print_registros_campos(lista_registros):
    print "Prints each field name and field value"
    for registro in lista_registros:
        print "gid: " + str(registro[0])
        print "descripcion: " + str(registro[1])
        print "geometria: " + str(registro[2])
```

In the following listing a example of use from the three previous functions is presented:

```
lista_registros= selecciona_datos_edificios()
Consulta: select gid, descripcion, st_astext(geom) from d.edificios

print_registros_completos(lista_registros)
Registros completos
(1, u'Primera descripcion', u'POLYGON((727844 4373183,727896
   4373187,727893 4373028,727873 4373018,727858 4372987,727796
   4372988,727782 4373008,727844 4373183,727844 4373183))')
(2, u'Segunda descripcion', u'POLYGON((727988 4373188,728054
   4373192,728051 4373093,727983 4373093,727988 4373188))')

print_registros_campos(lista_registros)
Registros campos
gid: 1
descripcion: Primera descripcion
geometria: POLYGON((727844 4373183,727896 4373187,727893
   4373028,727873 4373018,727858 4372987,727796 4372988,727782
   4373008,727844 4373183,727844 4373183))
gid: 2
descripcion: Segunda descripcion
geometria: POLYGON((727988 4373188,728054 4373192,728051
   4373093,727983 4373093,727988 4373188))
```

In the following example, a *where* condition is used to filter the rows selected:

```
lista_registros=selecciona_datos_edificios(cond_where='area<%s',
   lista_valores=[10000])
Consulta: select gid, descripcion, st_astext(geom) from d.edificios
   where area<%s

print_registros_completos(lista_registros)
Registros completos
(2, u'Segunda descripcion', u'POLYGON((727988 4373188,728054
   4373192,728051 4373093,727983 4373093,727988 4373188))')

print_registros_campos(lista_registros)
Registros campos
gid: 2
descripcion: Segunda descripcion
geometria: POLYGON((727988 4373188,728054 4373192,728051
   4373093,727983 4373093,727988 4373188))
```

Functions to help the student

To execute queries with HTML data forms

Some functions are available to the student, in order to facilitate his own project develop. That functions are general and work whit any table and form. The requirement to use it in table editions, is that the control input names have to match with whit the field table names. All the functions described here are in the project *dw_pozossan_sessions*, with long comments.

We have to join the data user introduced in the web, in HTML forms, with the execution of some Python functions. A JavaScript function called *formToJSONString* has been created. That function transforms the HTML form data in a JSON string, and that string is the data which the python functions created need to know the introduced user data. The *formToJSONString* function only receives the the page's HTML form name. That function is located in the file *general_code.js*. To make the changes in the tables, the Python functions receive that JSON sring, in the *registro* parameter. That Pyhton functions are located in the *py* project package, in the following modules:

actualizar_tabla.py: Updates table rows. Contains a function which receives a table name, the primary key table name, the primary key field value from the row to update, and a JSON row data (*registro*). The function can also update the table geometries. The latest version of that file is in Poliformat (recursos/2017/pozossan_web_projects/).

borrar.py: Deletes table rows. Contains a function which receives a table name, the primary key name field, and the primary key name field value from the row to delete.

conectar.py: Contains a function that makes a psycopg2 connection, and makes a Python dictionary with de Psycopg2 connection and cursor objects.

consultar_tabla.py: Contains a function that receives a table name, the primary key field name, the primary key field name value, and a JSON string (*registro*). The function selects the table row which the primary key field value matches. The JSON string (*registro*), is used to know the table field names. The function returns a JSON, with some key and values, one of this is other JSON which contains a list whit the row selected. If the primary key field values is not gave, all the rows are returned in the second JSON.

insertar.py: Inserts a row. Contains a function that receives a table name, the primary key filed name, and a JSON string (*registro)*, with the names and field values of the row to be inserted. The function inserts the row, with or without geometry.

var_globales.py: Global variables with the data conexion and the program configuration.

transformar_coords_ol_to_postgis.py: Contains a function that receives a OL3 coordinates string, and returns a string coordinates in PostGIS format. This transformation is necessary to insert geometries in the database.

check.py: Contains use example of each function. That module have to be downloaded from Poliformat (recursos/2017/pozossan_web_projects/) an copied in the *py* package.

The function *formToJSONString*, and the Pyhton functions described in this section, is easier to make changes in PostGIS tables.

To load a JSON string in a HTML form

The described Python functions return a JSON string with a row. That JSON is received by the client in JavaScript. To show directly the row data in the corresponding HTML form, the function *load_values_form*, in the file *load_values_form.js* has been created.

Exercise 9. Database update across Internet

The goal of this exercise is that the student have able to modify the database with the data gave by the user in the geoportal.

The functionalities requested here can be programmed directly in the student personal project. If you do not have it jet, you can do the exactly following tasks.

- : Create a geoportal where the *d.edificios* layer be published by WFS. Create a form with the field table names: gid, description, type (type of geometry, in this case *Polygon*), and coordinates (to store the polygon coordinates).
- : Each time you click over a building, a query have to be done to the database, and show the field values and its coordinates. All this information have to be retrieved from the server. You can't use the layer WFS data.
- It has to have a button in the buildings form to delete the building in the database.
- : It has to have a button in the buildings form to save the changes done in the HTML form, in the data base table row which corresponds whit the row showed in the HTML form.
- It has to have a button to insert a new building with the current data stored in the HTML form. The form building coordinates have to be able to be drown in the map, with the snapping interaction to the existing buildings activated.

Delibery You do not have to deliver nothing, only to show the functionalities to the lecturer.

Session control

Cuando es necesario diferenciar usuarios, por las cosas que puede hacer cada uno de ellos, es necesario identificarlos mediante usuario y contraseña. Una vez identificado, el servidor debe recordar quién es cada usuario, ya que el usuario está usando la página y está continuamente haciendo peticiones al servidor. Como el servidor recuerda qué usuario, es puede dar acceso a las partes correctas del programa. El servidor debe olvidar al usuario si este cierra el navegador. Para esto hay dos cosas necesarias:

When it is necessary to difference users, by the permissions that each one have, it is necessary to identify each user by its credentials. Once identified the user, the server have to remember whose user is each request from, to allow the request or not. The server have to forget the user only if the user close the web browser. This is only possible using session variables. The functioning is the following:

- First: the web browser genetates a random number each time that a page is visited. That number is the session number.
- Second: it is needed a library to save in the hard disk the session number and the user data, whilst the user do not close de web browser. That library have to be installed, and is the developer who program which data must to be saved in each user session.

Each time that the user, browsing, makes a request, the server checks the session number and if the user is already identified, that is, if the user has already initialized a session. After that goes ahead with the request or not.

When the user closes the web browser, changes the session number and the user have to identify again.

Para la gestión de variables de sesión se va a instalar una nueva librería, denominada *Beaker*. Se instala con sudo apt-get install pyton-beaker. Luego hay que reiniciar el servicio de Apache.

To the session variables management you need to install a new Python library called *Beaker*. You have to execute the command: *sudo apt-get install pyton-beaker*. Once installed the Apache service have to be initialized.

Beaker can write the session variables in a database or a folder. In this case the folder mode has been chosen. Then you have to create a new folder, and to give write permission to Apache to it. Apache belongs to the *www-data*, group, an uses the *www-data* user.

To create the folder an give permissions to Apache:

```
sudo mkdir /var/www/apps/sessions
sudo chown -R www-data:www-data /var/www/apps/sessions
sudo chmod -R og-r /var/www/apps/sessions
```

Session management example with Beaker

Coming up a example of session variables saving is sown. The variable will be accessible until the user closes the web browser.

Listado 5.4: Ejemplo de gestión de sesiones con Beaker

```
def application(environ, start_response):
    session = environ['beaker.session']
    if not session.has_key('value'):
        session['value'] = 0
    session['value'] += 1
    session.save()#save the data session in the memory
    session.persist()#save the data session in the hard disk

    start_response('200 OK', [('Content-type', 'text/plain')])
    return ['The current value is: %d' % session['value']]

####To manage sessions in /var/www/apps/sessions/ folder##########
session_opts = {
    'session.type': 'file',
    'session.key':'session_pozossan',
    'session.secret':'secretkey_pozossan',
    'session.data_dir':'/var/www/apps/sessions/',
    'session.cookie_expires': True,
    'session.key':'session_pozossan',
    'session.secret':'secretkey_pozossan',
    'session.validate_key':'some secret value'
}
#    'session.encrypt_key':'some other value'
from beaker.middleware import SessionMiddleware
application = SessionMiddleware(application,session_opts)
```

In the above listing, the *SessionMiddleware* class, through its constructor, adds to the dictionary *environ* a new key called *beaker.session*, which value is a object, similar to a Python dictionary. The *session* object, that is retrieved through the *beaker.session* key, allows to add values inside the object

session, with the same form that a dictionary: *session[ćlave]Í=valor*. Once the values are added, the method *save* and *persist* have to be called, in order to save the new data in the session object.

The *session* object values are retrieved as the same form as a Python dictionary values: *valor=session[‘clave’]*.

This can be used to know if a user is already identified or not. To do this:

- A register form is sent to the user. The user fills the form and send it to the server.
- The server receives de form data, and uses higher user credentials to connect to the database and check if the user name and password are correct. If it is right gets the kind of user. To do that a user table (*users*) is usually used. The users table usually have the user login, user name, other data and the kind of user. Depending of the king of user, the WSGI application will give access to a request or not. Usually there are that types of user: editor, consultant, and administrator. In the case of the textitdw_pozossan_sessions project, the user authentication data are in the table *d.empleados*. The user login is in the *dni* field, and the password is in the *apellido1* field.
- Once the server knows that the user is right, and the kind of user, all the data user are saved, using the *SessionMiddleware* class. Then the server sends a answer to the client and the request execution in the server finishes.

On finishing the request server execution, the WSGI application finishes. That means all the variables are cleaned, except the session variables, which remain in the *environ[‘beaker.session’]*) object. Thanks to that object it is possible to know if the user has already initialized a session or not. Therefore, to check if the user is identified or not is always a task to do, when the user wants to do something in the page which requires some privileges.

In the session variables you can save almost any thing, even complex objects.

CAPÍTULO 6
Evaluation

Evaluation

The subject evaluation is divided in two parts: the class exercises (40%) and a own project (60%). The exercises an the project can be done by pairs.

The exercises are the ones which appears in the DesarrolloWeb2017_en.pdf document. The delivery is done in class time, while an other exercise is being done. The delivery of the exercises consists in show the exercise to the lecturer and to answer some questions.

The own project consists in a geoportal, chosen by the student. The geoportal has a minimum requirements. The class exercises can be applied to the own project, instead of to do the exercise, always that all the requirements in the exercise be accomplished. In an other form: instead of doin the exercises in the pdf file, you can advance your own project, if all the steps in each exercise are done in your own project.

The project will be showed the last class, and will be evaluated in that moment.

Project minimum requirements of the own project

The geoportal has to have at least the following elements:

- A navigation menu, where appears and disappears parts of the page, on pressing the menu elements.
- A map, done with OpenLayers, Leaflet or other JS library.
  - Layers wms and wfs. The layers have to be created by the student and be published by GeoServer or MapServer.
  - Base layers like ortophotos, OSM, etc.
  - One of the wfs layers has to be labeled.
  - One of the wms layers has to have a field value simbology, using a SLD file.
  - Cursor coodinates indicator and a layer control.
  - Element selection, at least from a one layer. On each element selection, a form have to be showed with the selected element database data. You have to retrieve the data from the server. The request has to be done with Ajax.
  - Has to be able to modify elements from a one layer, and to change all the data in the database: geometries and data fields.
  - You have to be able to draw new elements and to add them to the database, sending the corresponding form.
  - You have to be able to delete elements from the database.
  - On drawing or modify elements from a layer, the snap interaction has to work over the elements from at least one layer.
- The session management is optional.

Bibliografía

[1] Jose Carlos Martínez Llario, *PostGIS 2. Análisis Espacial Avanzado*, 2012.

[2] Adrian Holovaty, Jakob Kaplan-Moss, *La guía definitiva de Django*, 2009.

[3] Joseph W. Lowerly, Mark Fletcher, *HTML 5 para desarrolladores*, 2011.

[4] Juan Diego Gauchat, *El gran libro de HTML5, CSS y JavaScript, 2012*

Páginas web.

[5] Página de referencia HTML5: http://www.w3schools.com/html

[6] Página de referencia de CSS3: http://www.w3schools.com/css/default.asp

[7] Tutorial: http://www.tutosytips.com/aprende-html5-desde-0

[8] Referencia HTML 5 para desarrolladores. http://www.html-5.com

[9] Compatibilidad de HTML 5 con dispositivos móviles: http://mobilehtml5.org

[10] Tipos de controles input: http://www.w3schools.com/html/html_form_input_types.asp

[11] Control canvas http://www.html5canvastutorials.com/advanced/html5-canvas-mouse-coordinates/

[12] Open Geospatial Consortium, OGC. www.opengeospatial.org

[13] PostgreSQL. http://www.postgresql.org

[14] PostGis. http://postgis.refractions.net. Extensión que da soporte de datos espaciales a la base de datos

[15] Descripción del lenguaje SQL http://www.postgresql.org/docs/9.1/static/tutorialsql.html

[16] Descripción del lenguaje PL/SQL http://www.postgresql.org/docs/9.1/static/plpgsql.html

[17] Quantum Gis, Qgis. http://www.qgis.org

[18] Python. http://www.python.org

[19] PyQt4. http://www.riverbankcomputing.co.uk/software/pyqt/intro

[20] PIL. http://www.pythonware.com/products/pil. Biblioteca Python el trabajo con imágenes

[21] psycopg2. http://pypi.python.org/pypi/psycopg2. Biblioteca Python para la conexión con PostgreSQL

[22] Eclipse. http://www.eclipse.org. Entorno de desarrollo (IDE)

[23] PyDev http://pydev.org. Plugin para Eclipse para el desarrollo en el lenguaje Python