



sergiomhdw@gmail.com

2021 TRABAJO DAW

RESERVA DE CLASES

Sergio Moreno Hedrera

Tabla de contenido

INTRODUCCIÓN:.....	3
Mi idea inicial:.....	3
Material Utilizado:	3
DESCRIPCIÓN:.....	3
INSTALACION:.....	4
GUÍA DE ESTILOS Y PROTOTIPADO	8
Identidad Visual:.....	8
Colores seleccionados:	8
Tipografía:	8
Wireframe:	9
Mockups:	10
DISEÑO	11
Diseño UML	11
Casos de uso.....	12
DESARROLLO	13
Django:.....	13
Flask:.....	14
Ejemplo de un recorrido de una llamada MVT.....	15
Ejemplo de un recorrido de MVT con Ajax.....	16
Dificultades:.....	19
Generar todos los profesores de forma automática	20
Control de versiones:	22
PRUEBAS	22
Pruebas de servidor	22
Pruebas de Diseño	25
DESPLIEGUE.....	25
MANUAL	30
CONCLUSIONES.....	36
ÍNDICE DE TABLAS E IMÁGENES.....	38
BIBLIOGRAFÍA	39

INTRODUCCIÓN:

Mi proyecto trata en añadir una mejora a la plataforma del centro. Consiste en la creación de una aplicación para que los profesores puedan ver todas las aulas que se pueden reservar, realizar reservas, ver cuándo y a qué hora hay reservas hechas por otros profesores, etc.

Mi idea inicial:

La primera opción era crear un mapa donde se vieran las salas para reservar, pero después al pensarlo quizá era poco intuitivo, y difícil de ampliar si se hacen cambios en el plano. Así que cambie la idea y decidí mostrar todas las aulas que son reservables y mostrarlas de la mejor forma posible para que fuera fácil su acceso y claro. Una vez dentro del aula que saliera un calendario y las reservas de los diferentes días con las horas etc.

Material Utilizado:

- Framework de servidor **Django** que utilizada como lenguaje **Python**.
- Para la parte de Cliente he utilizado **JavaScript, Ajax, JQuery**, y estuve leyendo en varios sitios que **Django** se puede considerar Framework de Frontend.
- En diseño he utilizado **HTML5** y **CSS** y he añadido librerías para iconos como <https://lordicon.com/icons>
- **Dockers** para despliegue.

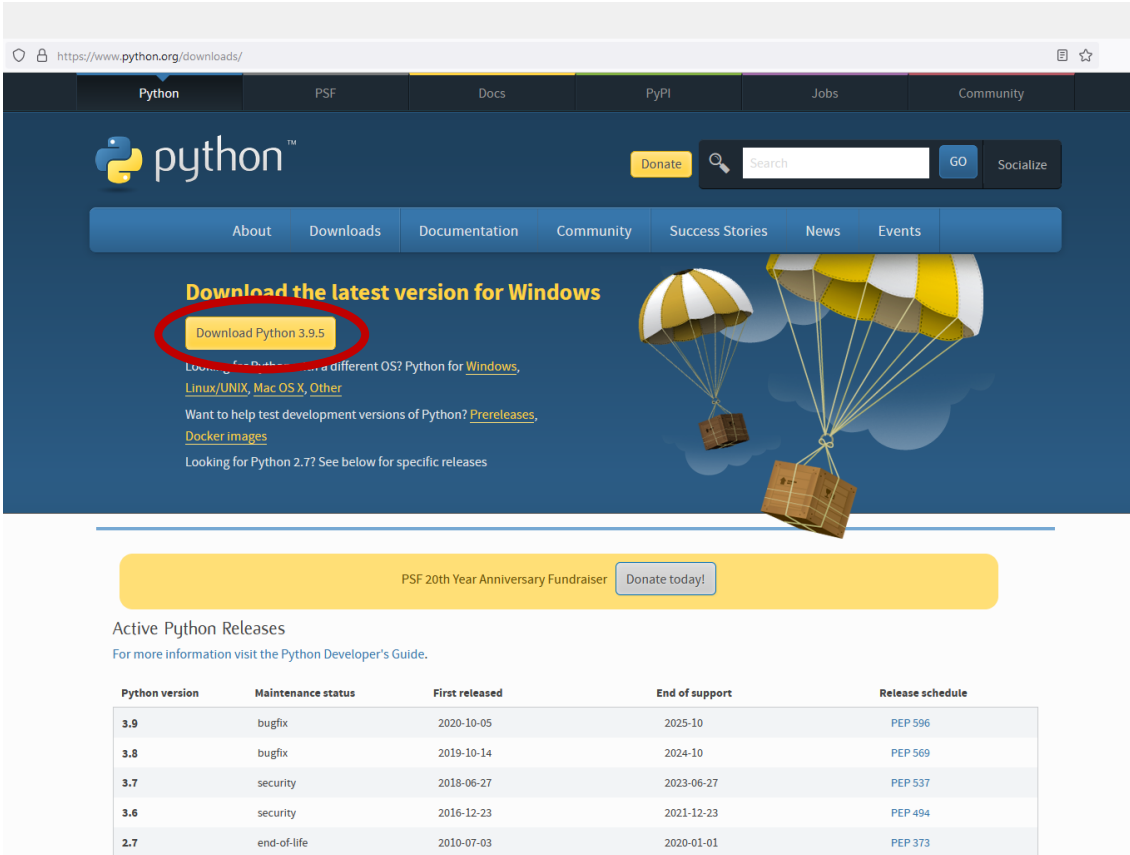
DESCRIPCIÓN:

El Resultado final de la aplicación consiste en ver todas las clases, hacer una reserva, ver las reservas de un día, ver todas mis reservas en general, poder borrar una reserva, un apartado para administrador, el administrador puede ver todas las reservas de todos los profesores, y se puede filtrar por día, y evento.

También se pueden cargar todos los profesores de séneca para no tener que meterlos uno a uno

INSTALACION:

Para la instalación necesitaremos tener en nuestra maquina Python 3.9.5, para ello nos iremos a la página oficial de Python y descargaremos nuestra versión.



The screenshot shows the Python.org website with the URL <https://www.python.org/downloads/>. The navigation bar includes links for Python, PSF, Docs, PyPI, Jobs, and Community. The main content area features the Python logo, a search bar, and a 'Download the latest version for Windows' section. A yellow button labeled 'Download Python 3.9.5' is circled in red. Below this button, there are links for 'Python for Windows', 'Linux/UNIX, Mac OS X, Other', 'Prereleases', 'Docker images', and 'Looking for Python 2.7? See below for specific releases'. A yellow banner for the 'PSF 20th Year Anniversary Fundraiser' is also visible.

Python version	Maintenance status	First released	End of support	Release schedule
3.9	bugfix	2020-10-05	2025-10	PEP 596
3.8	bugfix	2019-10-14	2024-10	PEP 569
3.7	security	2018-06-27	2023-06-27	PEP 537
3.6	security	2016-12-23	2021-12-23	PEP 494
2.7	end-of-life	2010-07-03	2020-01-01	PEP 373

Ilustración 1 Página de Python para descargarlo

Si la versión que apareciera fuera distinta iremos más abajo donde nos saldrán todas las versiones y podremos descargar la 3.9.5

Una vez la hayamos descargado lo abrimos y lo instalamos.

Para ver si lo hemos instalado correctamente abrimos nuestra cmd y ejecutamos el siguiente comando: **Python -- version**

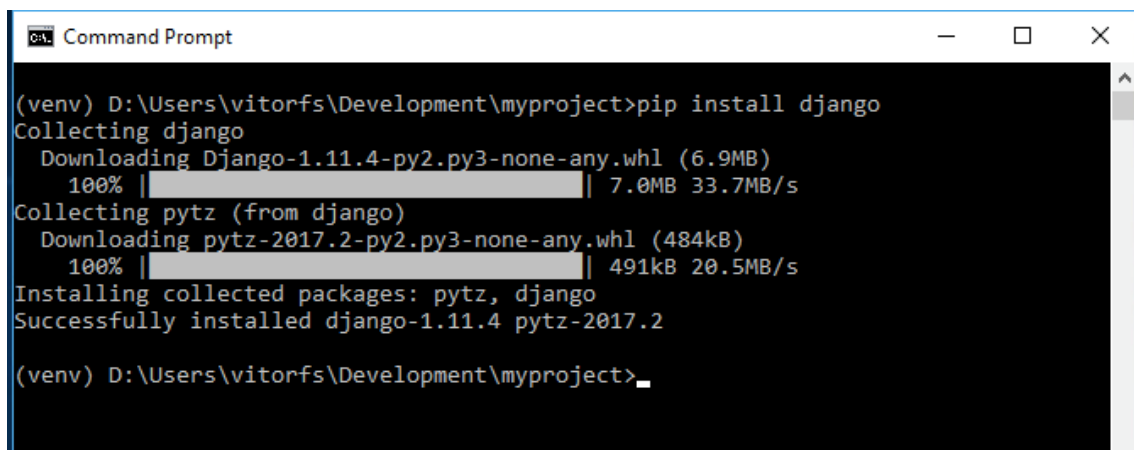
```
C:\Users\sergi\OneDrive\Escritorio\DAW PROYECTO\proyecto>python --version
Python 3.9.5
```

Ilustración 2 Prueba de que Python está instalado

Una vez nos hemos asegurado que Python lo tenemos instalado vamos a instalar Django, para instalar Django es necesario tener antes instalado pip. Esta herramienta en la versión de Python 3.9 ya viene instalada así que no tendremos que preocuparnos, si utilizamos otra versión de Python tendremos que comprobar si tenemos el pip instalado.

```
C:\Users\sergi>pip --version
pip 21.1.1 from c:\python39\lib\site-packages\pip (python 3.9)
```

Después de asegurarnos que lo tenemos vamos a instalar Django, usamos el comando en nuestra cmd: **py -m pip install Django**



```
Command Prompt

(venv) D:\Users\vitorfs\Development\myproject>pip install django
Collecting django
  Downloading Django-1.11.4-py2.py3-none-any.whl (6.9MB)
    100% |#####| 7.0MB 33.7MB/s
Collecting pytz (from django)
  Downloading pytz-2017.2-py2.py3-none-any.whl (484kB)
    100% |#####| 491kB 20.5MB/s
Installing collected packages: pytz, django
Successfully installed django-1.11.4 pytz-2017.2

(venv) D:\Users\vitorfs\Development\myproject>
```

Ilustración 3 instalación de Django

Una vez haya finalizado comprobaremos en nuestra cmd que lo hemos instalado

```
C:\Users\sergi>python
Python 3.9.5 (tags/v3.9.5:0a7dcdb, May 3 2021, 1
Type "help", "copyright", "credits" or "license"
>>> import django
>>> django.VERSION
(3, 2, 3, 'final', 0)
>>> ^Z

C:\Users\sergi>
```

Ilustración 4 Prueba de que Django está instalado

Ahora instalaremos nuestro servicio de base de datos (MySQL) con el comando **pip install mysqlclient**

Una vez tengamos todo instalado, tendremos que configurar un archivo llamado settings en el cual le vamos a decir como conectar a nuestra base de datos.

```
# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'proyecto',
        'USER': 'root',
        'PASSWORD': 'pestillo',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Ilustración 5 Configuración de db en el archivo settings

Antes de poder conectar con nuestra base de datos debemos tenerla creada. Debemos de asegurarnos que los puertos que vamos a utilizar estén disponibles ya que si están en uso no nos dejará conectarnos por esos puertos.

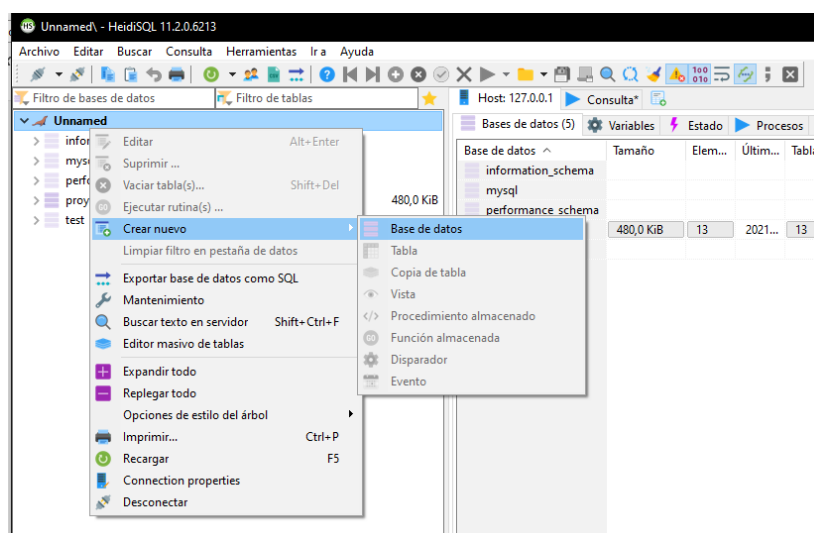


Ilustración 6 Creacion de la db en Heidi

En mi caso para el control de base de datos he utilizado HeidiSQL, para crear la base de datos tendremos que hacer click derecho crear nuevo, base de datos.

Para que la base de datos coja la información de nuestro Django tendremos que hacer migraciones con **Python manage.py makemigrations**

```
C:\Users\sergi\OneDrive\Escritorio\DAW PROYECTO\proyecto>python manage.py makemigrations
Migrations for 'reserva':
  reserva\migrations\0003_alter_reserv_id.py
    - Alter field id on reserv

C:\Users\sergi\OneDrive\Escritorio\DAW PROYECTO\proyecto>
```

Ilustración 7 Hacer migraciones para Django 1.0

Y una vez esto tendremos que hacer **Python manage.py migrate** y así se nos subirá la información a nuestra base de datos.

```
C:\Users\sergi\OneDrive\Escritorio\DAW PROYECTO\proyecto>py manage.py migrate
Operations to perform:
  Apply all migrations: admin, app, auth, contenttypes, reserva, sessions
Running migrations:
  No migrations to apply.
```

Ilustración 8 Hacer migraciones para Django 1.1

Por último, instalaremos Pillow con **pip install pillow**

Pillow es una herramienta para poder subir y trabajar con imágenes con Django.

```
C:\Users\sergi\OneDrive\Escritorio\DAW PROYECTO\proyecto>pip install pillow
Requirement already satisfied: pillow in c:\users\sergi\appdata\local\programs\python\python39\l...
WARNING: You are using pip version 21.1.1; however, version 21.1.2 is available.
You should consider upgrading via the 'c:\users\sergi\appdata\local\programs\python\python39\pyt...
-upgrade pip' command.

C:\Users\sergi\OneDrive\Escritorio\DAW PROYECTO\proyecto>
```

Ilustración 9 Instalación de Pillow

Para el despliegue he utilizado dockers, así que tendremos que instalarlo en nuestro equipo también:

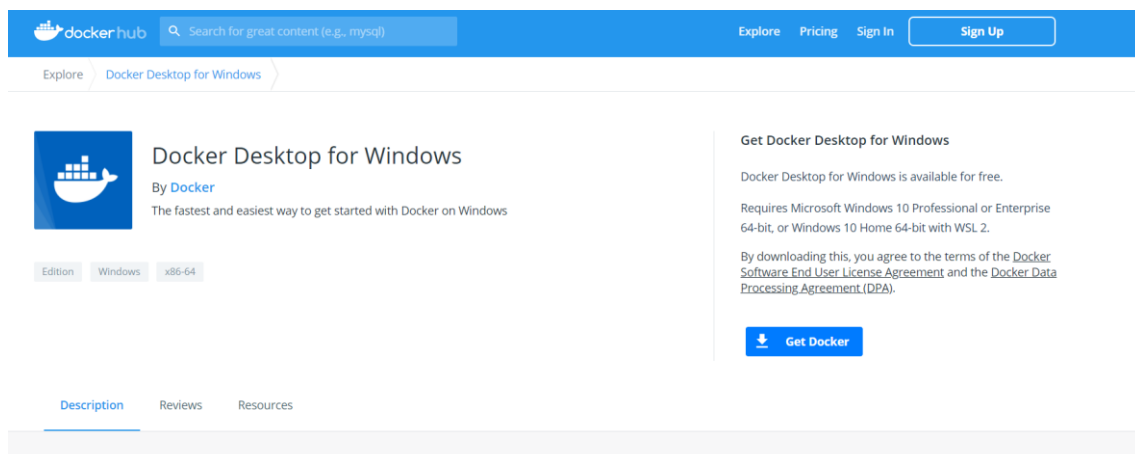


Ilustración 10 Página de descarga a Docker

GUÍA DE ESTILOS Y PROTOTIPADO

Identidad Visual:

Al comenzar el proyecto tuve claro que esta aplicación se iba a usar sobre todo en los ordenadores del centro, así que enfoqué el diseño para desktop en vez de mobile first.

Colores seleccionados:

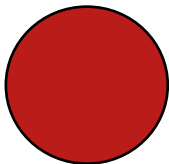
En cuanto al color del diseño estuve pensando si escoger el azul de la Moodle del centro o en usar unos colores rojos ya que son realmente los colores que representa nuestro centro, al final me decanté por usar unos colores rojizos y corporativos, como podemos ver en el logo son los colores que utilizan.



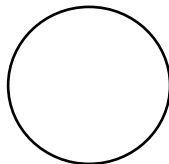
Ilustración 11 Logo IES Rafael Alberti

La tonalidad de colores que he escogido es la misma, ya que el centro tiene su página web con estos mismos colores para estar en sintonía.

#BA1C1A



#FFFFFF



Tipografía:

En cuanto a la tipografía al principio decidí utilizar la misma de la plataforma (Poppins sans-serif), pero no me convencía ya que parecía que era una web

antigua y la cambie por una ROBOTO, así le doy un toque más actual a la web, aparte de esto, roboto es una letra clara, fácil de leer y de entender y para una aplicación que se va a utilizar en un Instituto creo que esto es una de las cosas que hay que tener en cuenta.

ROBOTO:

ABCČĆĐĖFGHIJKLMNOPQRSŠTUVWXYZŽabcčćđđ
 efghijklmnopqrsštuvwxyzžАБВГГґДґЕЕЁЖЗСИІЇЈК
 ЛЉМНЊОПРСТҢУЎФХЦЧШЩЪЫЬЭЮЯабвггђ
 еёежзсииііјкљмнњопрстђуўфхцчшщъыьэюяАВ
 ΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩαβγδεζηθικλμνξοπρ
 στυφχψωάΆΈΕΉΊΰΐΌόΰϋϚϛϜϝϞϟϠϡϢϣϤϥϦϧϨϩ
 ϪϫϬϭϮϯϰϱϲϳϴϵ϶ϷϸϹϺϻϼϽϾϿϿ1234
 567890'?"'!" (%) [#] {@} /& \<-+÷×=>®©\$€£¥¢;,:.*

Wireframe:

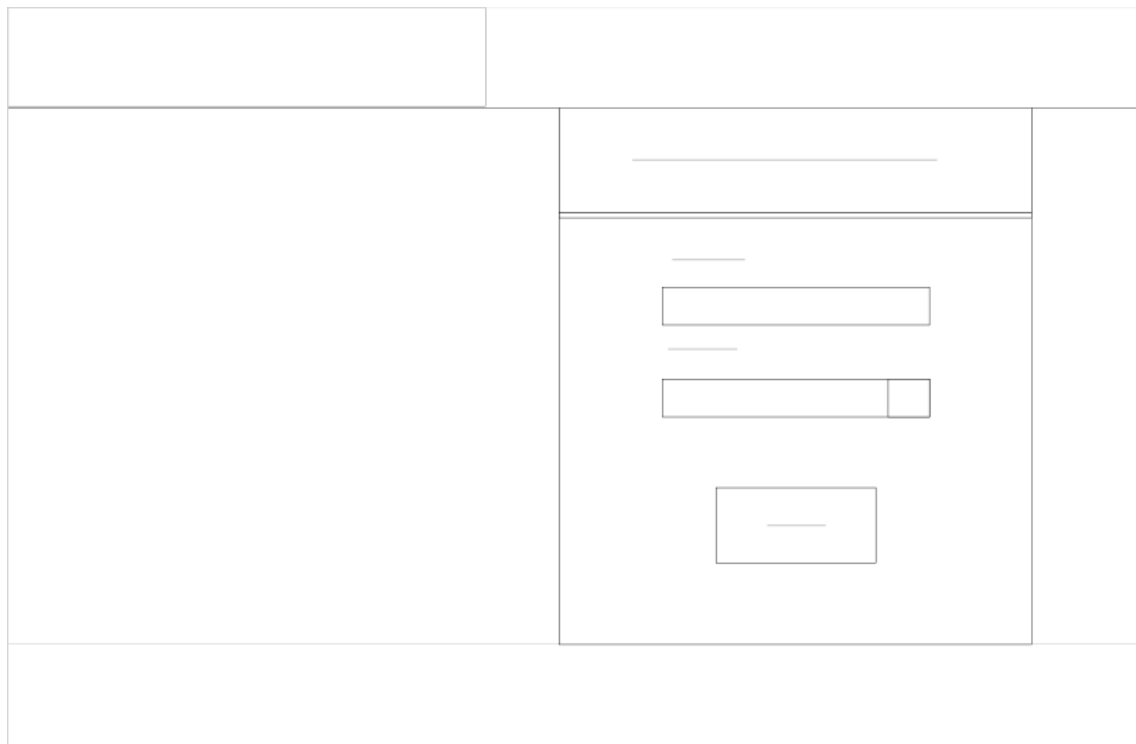


Ilustración 12 Wireframe 1

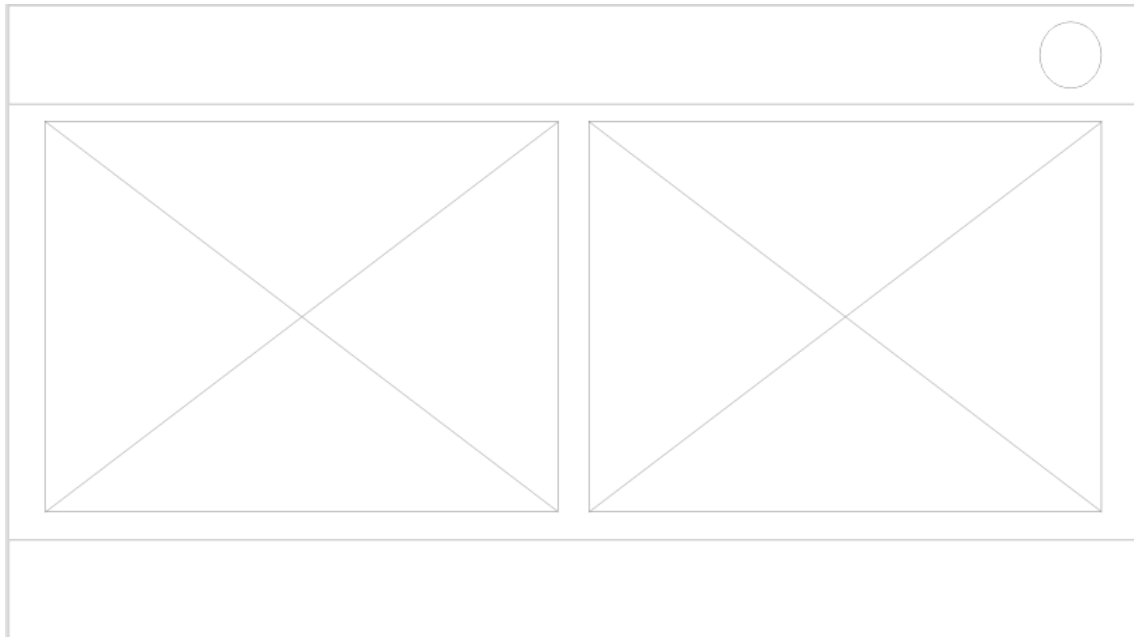


Ilustración 13 Wireframe 2

Mockups:

Login:



Ilustración 14 Login del mockup

[Para ver el resto de documentos de wireframe y del mockup.](#)

DISEÑO

Diseño UML

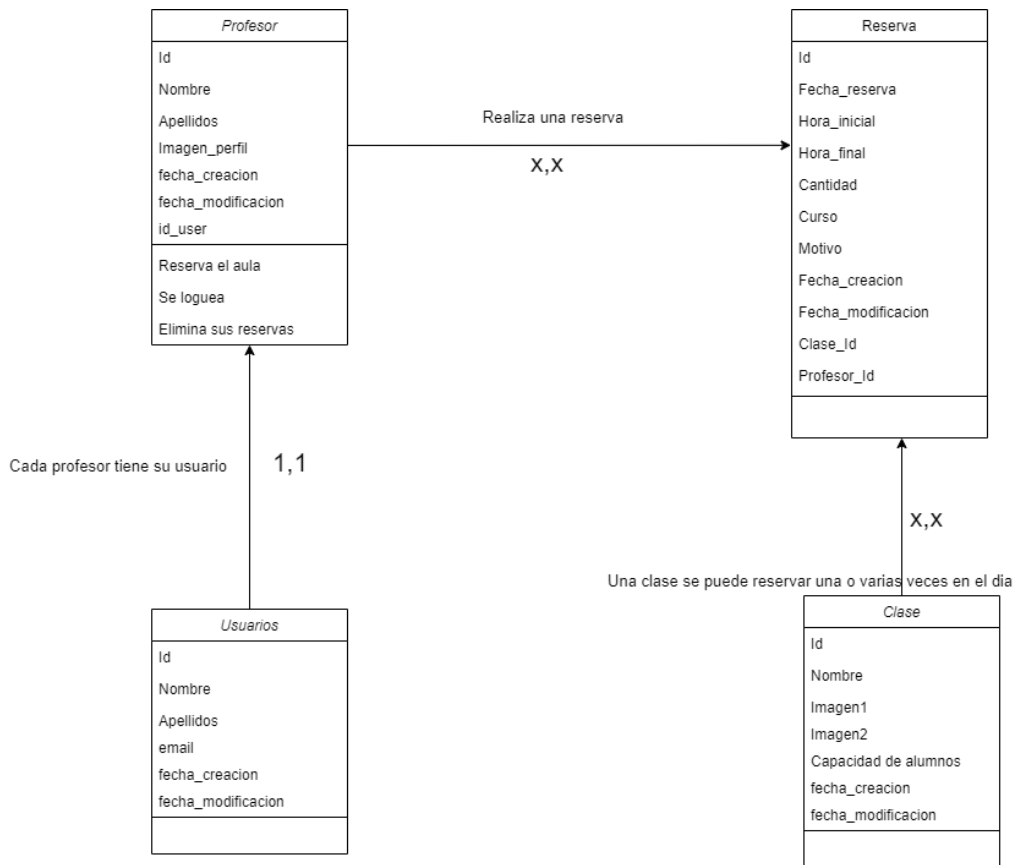


Ilustración 15 Diseño UML

Tenemos varias entidades

- **Usuarios:** Los usuarios son los profesores que se loguean y cada usuario va relacionado con un profesor así que tienen una relación de uno a uno.
- **Profesores:** Son los que realmente contienen todos los datos de los profesores, nombre, apellidos, imagen, etc... Los profesores son los que pueden hacer una reserva de una clase. Un profesor puede reservar una o varias clases y una clase puede ser reservada por un profesor o por varios lo que significa que entre profesor y clases existe una relación de muchos a muchos. Al haber una relación de muchos a muchos tengo que crear una tabla que se llama reserva para controlar todas las aulas que se reservan y que profesores la reservan.

- **Clase:** Las clases como hemos especificado anteriormente tienen una relación de muchos a muchos y su tabla guarda el nombre, el id, la cantidad de alumnos etc.
- **Reserva:** Esta tabla generada al haber una relación muchos a muchos guardan las fechas de las reservas, las horas, los id de las diferentes tablas etc.

Casos de uso

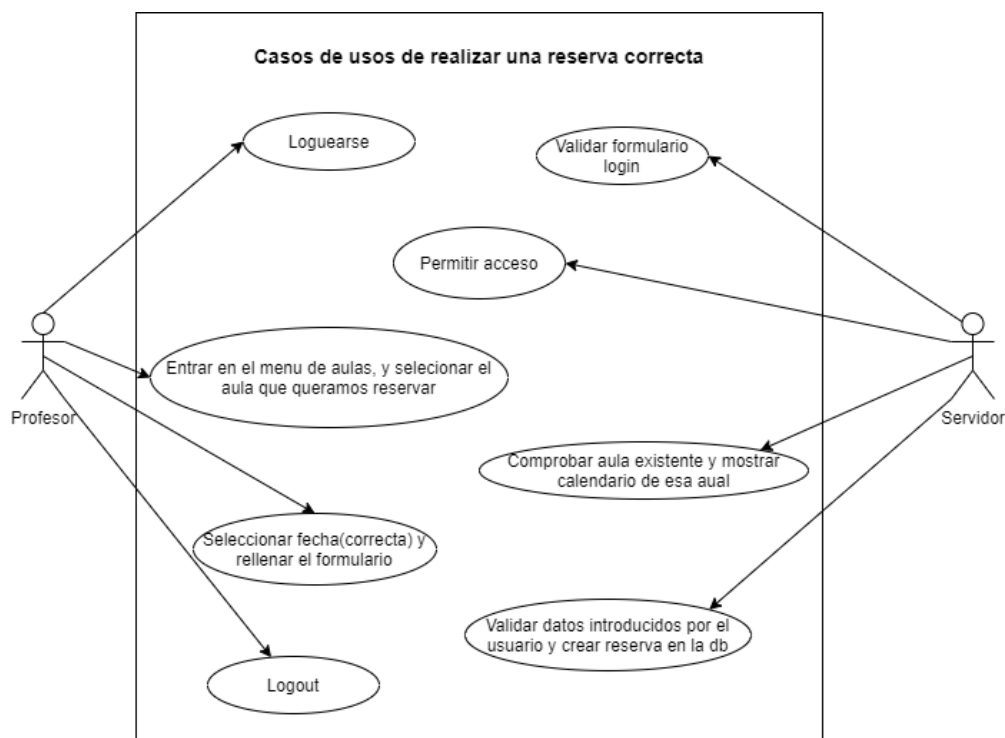


Ilustración 16 Diseño de casos de uso

Un caso de uso de una reserva correcta entre el usuario y el servidor.

- Primero el usuario entra en la aplicación y lo primero que se encuentra es el login.
- El servidor comprueba que los datos son correctos, si lo son pasa a la siguiente página o si no muestra un error
- El usuario navega por la página hasta que llega a la zona donde se encuentran todas las aulas y selecciona una.

- El servidor comprueba si el aula existe y si no lanza un 404, si la página es correcta manda al usuario a la web de esa aula en concreto y puede realizar una reserva.
- El usuario tendrá que elegir un día y rellenar un formulario con datos correctos para que la petición se tramita exitosamente.
- Por último, el servidor comprueba ese formulario y si es correcto crear una reserva en la base de datos.

DESARROLLO

Debido a un cambio de planes y falta de tiempo he tenido que trabajar con una secuencia de desarrollo rápida. Antes de empezar a instalar nada tenía que tener claro lo que quería, las tablas de bases de datos que iba a tener, las relaciones que podía existir, así que creé el modelo UML para tenerlo claro. Una vez hecho esto, estuve barajando dos posibles Frameworks para trabajar en la parte de servidor, Flask o Django.

Los dos están escritos en lenguaje Python, lenguaje que habíamos dado el año anterior durante todo el curso y por eso me sentía bastante cómodo con este lenguaje, aparte que es flexible y cómodo a la hora de trabajar.

Estuve buscando información sobre cuál de los dos utilizar:

Django:

Django es lo que se conoce como un framework “full stack” para Python con el cual se pueden abordar todo tipo de proyectos en este lenguaje como el desarrollo web escalable y de alta calidad.

Se trata del entorno de desarrollo de Python más popular, ya que permite un desarrollo ágil al encargarse de manejar los controladores.

Las ventajas que tiene Django son:

- Cuenta con un sistema de autenticación de usuarios.
- Ofrece un gran rendimiento y flexibilidad, pudiendo escalar proyectos de forma sencilla.
- Trabajar bajo un patrón MVC (Modelo Vista Controlador), lo que permite un desarrollo ágil y reutilizable.
- Incorpora una amplia variedad de paquetes de librerías (más de 4000).
- Dispone de una inmensa comunidad de usuarios en internet, su documentación está actualizada y es muy amplia, aparte en la misma documentación tiene un curso para empezar a trabajar con django.
- Cuenta con panel de administración para bases de datos.

Flask:

Flask es un framework perfecto para personas que quieran hacer una pequeña aplicación y estén empezando ya que es muy flexible. Con Flask se utilizan las líneas de código necesarias para realizar cualquier acción por lo que es mucho más sencillo comprender la estructura de cualquier aplicación o script, y saber qué es lo que realiza.

Ventajas de Flask:

- Se adapta a cada proyecto instalando extensiones específicas para el mismo.
- Incluye servidor web propio para pruebas.
- El diseño minimalista de su estructura le permite ser rápido y con un gran desempeño.
- Cuenta con documentación extensa para el desarrollo de aplicaciones.
- Es muy sencillo de utilizar, por lo que es el indicado para empezar a programar con Python.

- Se integra con otras herramientas para incrementar sus funciones, como Jinja2 (motor de plantillas web) o SQLAlchemy (kit de herramientas SQL de código abierto).

Después de haber visto todas sus ventajas, ver como funcionaban, como se trabajaban con ellas, decidí utilizar django, no solo porque me parecía más cómoda, si no por la gran cantidad de información que había en internet ya que era más conocida que flask, aparte Flask estaba enfocada a hacer pequeñas aplicaciones y a trabajar con API y django para hacer aplicaciones web a un nivel más profesional. Por otro lado, ambos tienen desventajas y las de flask me hicieron elegir definitivamente django. Desventajas de flask como no contener librerías integradas, Se trata de un entorno que genera dificultades a la hora de realizar migraciones o pruebas unitarias. También es una desventaja tener que recurrir a un mapeo de objetos relacionales (ORM) externo para conectar con bases de datos.

Al empezar con Django y ver su forma estructurada de trabajar no me resulto difícil.

Ejemplo de un recorrido de una llamada MVT.

En este ejemplo voy a situarme en la página ramas, que es la página justo después de loguearnos en la aplicación, luego vamos a ir a la parte donde muestro todas las aulas. En el archivo HTML tengo que tener un enlace el cual tenga la ruta que esta está controlada por el archivo url:

```
<div class="grid-item"><div class="imagen"><a href="{% url 'aulas' %}">
```

En la línea de código de arriba indico en la etiqueta href a la ruta que quiero ir es a aulas, en la imagen de abajo es la configuración para llamar a la vista de aulas que tengo en el archivo urls.

```
path('', views.aulas, name="aulas"),
```

La ruta está definida y llama a la vista de aulas.

```
@login_required
def aulas(request):
    clases=Clase.objects.all()
    return render(request, "reserva/aulas.html", {"clases":clases})
```

En este caso la vista no tiene mucha dificultad, tiene un decorador que indica que el usuario tiene que estar logueado para poder acceder, luego guardo todas las clases para poder trabajar con la información dentro del template, por último, en el "return" digo que me lleve a la siguiente página con la lista de clases para usarlo posteriormente.

Si en la página tuviera que controlar datos, validación o etc, se configura en la parte de la vista ya que estas funcionan como controladores.

Ejemplo de un recorrido de MVT con Ajax

Para este ejemplo voy a mostrar la consulta en la que recibo todas las reservas de los profesores.

Empezamos en el código Javascript de misreservas.js. Como quiero que nada mas entrar en la página me devuelva las reservas, esta parte del código no la meto en ninguna función para que se ejecute nada mas entrar.


```

1
2 let staff = document.getElementById("staff");
3 let profesor = "";
4 if (staff.innerHTML == "True"){
5     profesor = "";
6 }
7 else{
8     profesor = staff.className;
9     console.log(profesor)
10 }
11 let info = document.getElementById("informacion");
12 while (info.firstChild) {
13     info.removeChild(info.firstChild);
14 }
15
16
17 $.ajax({
18     url: "/reserva/detallemisreservas_inicio",
19     method: "GET",
20     dataType: "json",
21     data: {
22         'profesor': profesor,
23     },
24     success: function(datos){
25         console.log("success");
26         console.log(datos);
27         if(datos.reservas==" "){
28             document.getElementById("error").style.display="block";
29             document.getElementById("contador").innerHTML="<p>Reservas activas: 0 </p>";
30         }else{
31             reservas=JSON.parse(datos.reservas);
32             clases=JSON.parse(datos.clases);
33             profesores=JSON.parse(datos.profesores);
34

```

Al principio guardo en la variable staff si el usuario que tengo logueado es staff o no. Si es staff lo dejo vacío, y si no es staff guardo el valor del id del profesor y luego borro toda la información que pueda tener en el div información, para que siempre se actualice.

Una vez obtenida esta información entramos en el Ajax:

Le decimos a que url tiene que ir, el método, el tipo de datos, y le vamos a pasar el profesor, una vez hecho esto, ajax manda la petición a nuestro archivo donde tenemos las rutas en este caso a detallemisreservas_inicio.

```
path('detallemisreservas_inicio/', views.detallemisreservas_inicio, name="detallemisreservas_inicio"),
```

Y como podemos ver en esta ruta nos manda a la vista detallemisreservas_inicio.

```
@login_required
def detallemisreservas_inicio(request):
    profesor=request.user.profesor.id

    if request.user.profesor.is_superuser:
        dat = Reserv.objects.filter(reservationday__gte=datetime.now())
    else:
        profesor = int(profesor)
        dat = Reserv.objects.filter(reservationday__gte=datetime.now()).filter(profesor_id=profesor)
    if dat.count() > 0 :
        datos={
            'reservas':serialize('json', dat),
            'profesores':serialize('json',Profesor.objects.all()),
            'clases':serialize('json',Clase.objects.all()),
        }
    else:
        datos = {
            'reservas': " "
        }
    return JsonResponse(datos)
```

En esta imagen en el request tenemos la información que nos ha mandado el ajax y en este caso almacenamos en una variable la Id que hemos pasado del profesor y comprobamos:

- Si el profesor es superusuario vamos a filtrar todas las reservas por el día actual hacia delante
- Si no es superusuario, las reservas se van a filtrar por el id de profesor que recibo desde Ajax, y por el día actual hacia delante.

Una vez he comprobado el rol del usuario logueado, vamos a comprobar si existen reservas si al comprobar el usuario y filtrar recibo reservas pues voy a guardar en datos toda la información para poder mostrarlos por pantalla, si no hay reservas mando nada. (Es importante que datos lleve algo, Si no declaro datos obviamente el servidor da un error 500).

Una vez termino todo mando los datos a Ajax de nuevo:

```
success:function(datos){
    console.log("success");
    console.log(datos);
    if(datos.reservas==" "){
        document.getElementById("error").style.display="block";
        document.getElementById("contador").innerHTML="<p>Reservas activas: 0 </p>";
    }else{
        reservas=JSON.parse(datos.reservas);
        clases=JSON.parse(datos.clases);
        profesores=JSON.parse(datos.profesores);

        creacionreserva(reservas,profesores,clases);
    }
}
```

Si la petición se ha hecho exitosamente, se comprueban los datos que han llegado. Si llega vacío ya se sabe que no hay reservas, si llegan datos pues llamo a una función para que trabaje con esos datos y me cree los elementos para que se vean por pantalla.

Si la petición es errónea:

```
error:function(datos){
    console.log("Error controlado por el servidor");
}
```

Muestro un console.log que me diga que el error está controlado por el servidor.

Dificultades:

Durante el trabajo me he encontrado varias dificultades, una de ellas fue a la hora de hacer el calendario, estuve mirando por internet ya que había muchos hechos y plugins que te ayudaban a hacerlo, pero al final viendo varios ejemplos conseguí hacer el mío propio.

Otra dificultad que tuve fue a la hora de hacer las reservas y comprobar cuando una reserva ya esté en una hora determinada.

Intenté desde el lado del servidor guardar las aulas que tenía reservadas para cuando coincidiera con la hora no dejara realizar la reserva, pero no es tan fácil ya que si una reserva dura varias horas en medio hay horas sueltas. Para resolver

este problema desde el lado del cliente no dejo seleccionar las horas que estén reservadas así si el cliente no rellena el formulario dará un error, y si lo rellena tendrá que seleccionar las horas que haya disponibles.

A la hora de implementar Ajax tuve algunos problemas ya que la petición no se realizaba correctamente al principio, después de ver un par de tutoriales conseguí hacer varias peticiones de esta forma.

A la hora de montar el proyecto en Docker tuve bastantes problemas con la base de datos, me costó bastante conectarla, aparte el lio de puertos que tuve porque el 3306 lo tenía en uso, luego el puerto 8000 es el de django por defecto y lo tenía cogido etc...

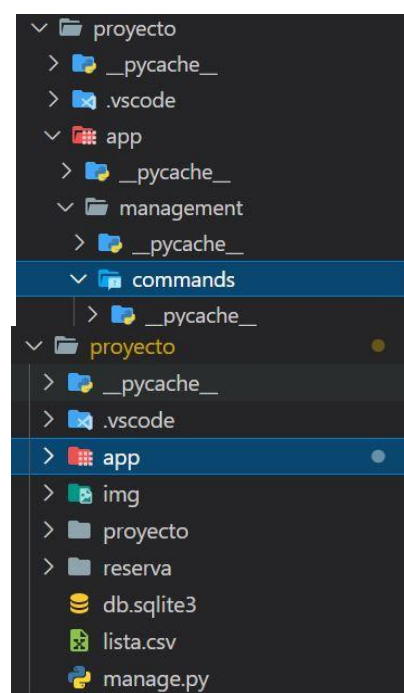
Generar todos los profesores de forma automática

A la hora de generar todos los profesores tuve dificultades ya que no puedo soltar el archivo en la misma carpeta de Django y ejecutarlo, Django no te lo reconoce.

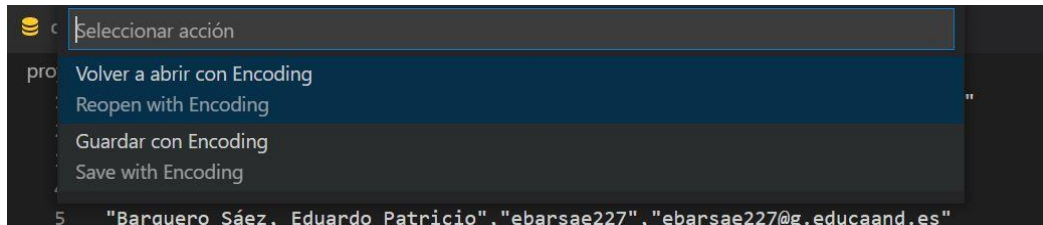
Para que funcionara tenía que guardarlo en un sitio con nombres de carpetas que Django reconociera y una vez hecho esto si podía ejecutarlo como un comando.

En mi caso como lo que iba a generar son profesores pues lo guarde en la aplicación de mi proyecto Django llamada App.

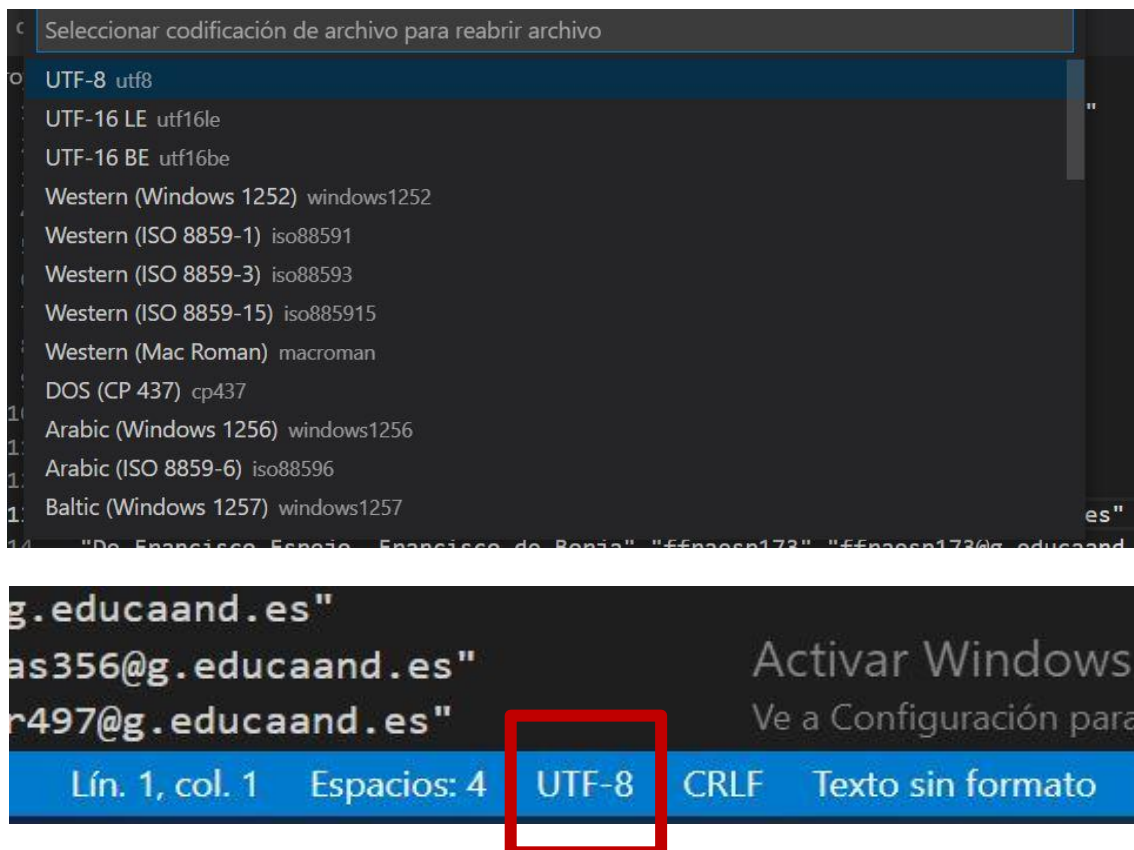
La estructura que tiene que tener es management dentro de la app, dentro de management un `__init__.py` y la carpeta commands dentro de esta otro `__init__.py` y nuestro script en este ejemplo `registro.py`, por último, la lista de profesores se debe de encontrar al nivel donde tenemos el archivo `manage.py`.



A la hora de recorrer todas las líneas de profesores del archivo.csv tuve bastantes problemas ya que me fallaba el codeado UTF-8 y tuve que modificar la de este archivo, para ello pulsamos f1 y escribimos encoding.



Aquí seleccionamos Guardar con Encoding y nos mostrará todas las formas que tenemos para transformar nuestro archivo, Seleccionamos utf-8 y abajo en el visual nos tendrá que salir

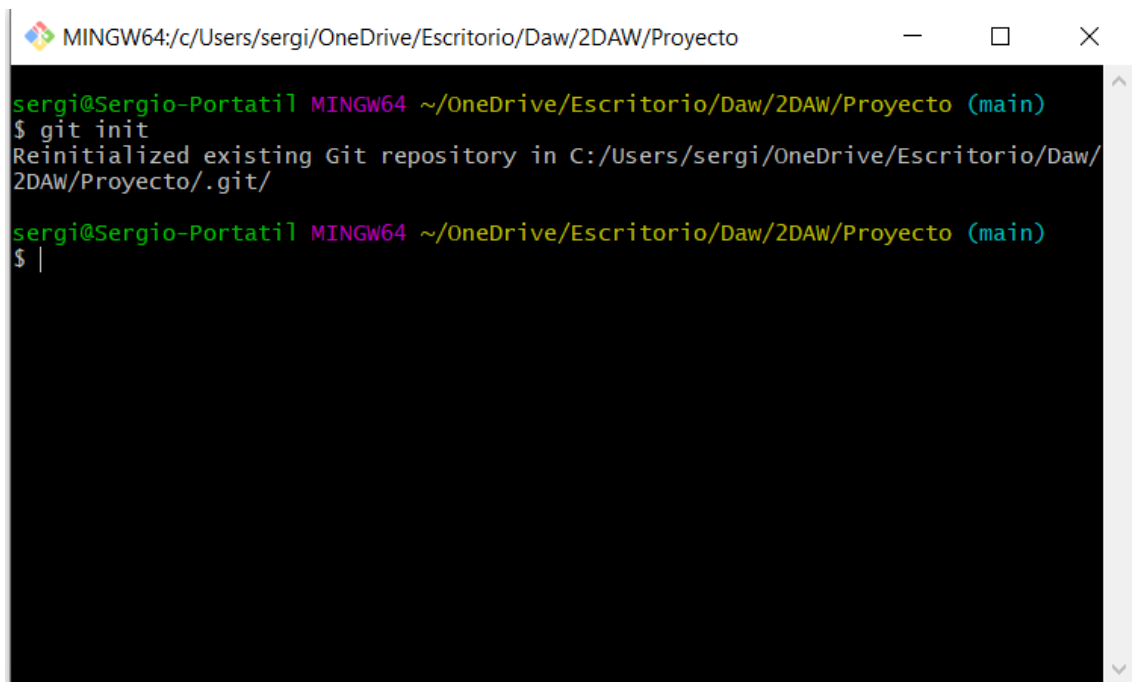


Control de versiones:

Para el control de versiones, he ido subiendo varias veces el proyecto a github, Normalmente lo he ido subiendo después de haber conseguido un objetivo o después de un día entero trabajando, aparte he subido otros archivos por si modificaba el original tener una copia en GitHub segura.

Al final he creado dos repositorios, uno lo he ido usando cuando he ido trabajando poco a poco con el proyecto y el otro lo he usado para configuraciones de Docker y el proyecto en docker.

Para subir la información al repositorio he trabajado con git bash:

A screenshot of a Windows command prompt window titled "MINGW64:/c:/Users/sergi/OneDrive/Escritorio/Daw/2DAW/Proyecto". The window shows the execution of the 'git init' command. The output indicates that an existing Git repository was reinitialized. The prompt is currently at the start of a new line.

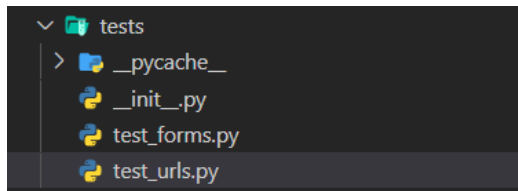
```
MINGW64:/c:/Users/sergi/OneDrive/Escritorio/Daw/2DAW/Proyecto
sergi@Sergio-Portatil MINGW64 ~/OneDrive/Escritorio/Daw/2DAW/Proyecto (main)
$ git init
Reinitialized existing Git repository in C:/Users/sergi/OneDrive/Escritorio/Daw/2DAW/Proyecto/.git/
sergi@Sergio-Portatil MINGW64 ~/OneDrive/Escritorio/Daw/2DAW/Proyecto (main)
$ |
```

Ilustración 17 Consola de git

PRUEBAS

Pruebas de servidor

He realizado varios test unitarios dentro del proyecto para comprobar que funciona, he hecho test a las Urls, a los modelos....



Para hacer los test he creado una carpeta y dentro los diferentes test que puedo hacer, y el archivo `__init__.py` el archivo `init`

es como un enrutador o un lanzador para poder llamar a los test y que se ejecuten, una vez hemos creados nuestros archivos vamos a configurarlos por dentro.

Entro en el test de las urls:

```
app > tests > test_urls.py > ...
1  import django
2  from django.test import SimpleTestCase
3  from django.urls import reverse, resolve
4  from app.views import ramas
5
6  class TestUrls(SimpleTestCase):
7
8      def test_login_url_resolved(self):
9          url = reverse('login')
10         self.assertEqual(resolve(url).func.view_class, django.contrib.auth.views.LoginView)
11
12     def test_logout_url_resolved(self):
13         url = reverse('logout')
14         self.assertEqual(resolve(url).func.view_class, django.contrib.auth.views.LogoutView)
15
16     def test_ramas_url_resolved(self):
17         url = reverse('ramas')
18         self.assertEqual(resolve(url).func, ramas)
```

Como la aplicación la tengo dividida en dos pequeñas partes no tengo todas las Url en el mismo, cada app del proyecto tiene su propia carpeta de test.

Por ejemplo, si tenemos que hacer un test al que hay que pasarle un argumento como el ID de la clase se pondría así:

```
def test_un_aula_url_resolved(self):
    url = reverse('un_aula', args=[6])
    self.assertEqual(resolve(url).func, un_aula)
```

Como vemos le estamos pasando el argumento el número 6 que es el Id de

una clase de prueba.

Como dije Antes se pueden hacer diferentes tipos de test ahora voy a mostrar los test que he hecho a los formularios que he creado con Django.

```

reserva > tests > test_form.py > ...
1  from django.test import SimpleTestCase
2  from reserva.forms import Reserva
3
4  class TestForms(SimpleTestCase):
5
6      def test_ReservaForm_valido_data(self):
7          form = Reserva(data={
8              'horainicio': '11:00:00Z',
9              'horafinal': '13:30:00Z',
10             'descripcion': 'examen',
11             'alumnos': 35,
12             'cursos': '3ESOA',
13         })
14
15         self.assertTrue(form.is_valid())
16
17
18      def test_ReservaForm_noValido_data(self):
19          form = Reserva(data={
20              'horainicio': '11:00:00Z',
21              'horafinal': '13:30:00Z',
22              'descripcion': '',
23              'alumnos': '',
24              'cursos': '3ESOA',
25          })
26
27          self.assertFalse(form.is_valid())
28
29      def test_expense_form_no_data(self):
30          form = Reserva(data={})
31
32          self.assertFalse(form.is_valid())

```

Dentro de los test del formulario he creado test con todos los datos bien puesto (un caso de test valido), y también he creado test a los que le faltaban un campo, o sin información (test no validos).

El primero contiene todos los datos lo cual es exitoso.

El segundo le falta el número de alumnos y el ultimo test no le paso datos directamente.

A la hora de probar los test nos iremos a nuestra cmd y

escribiremos:

Python manage.py proyecto reserva

En mi caso proyecto es el nombre de mi proyecto de Django y reserva es el nombre del módulo al cual le voy a hacer los test, como tengo dos tendré que hacer reserva y luego app

```

C:\Users\sergi\OneDrive\Escritorio\DAW PROYECTO\proyecto>py manage.py test proyecto reserva
System check identified no issues (0 silenced).
.....
-----
Ran 10 tests in 0.006s
OK

```

Ilustración 18 Test realizados

```

C:\Users\sergi\OneDrive\Escritorio\DAW PROYECTO\proyecto>py manage.py test proyecto app
System check identified no issues (0 silenced).
...
-----
Ran 3 tests in 0.003s
OK

```


Pruebas de Diseño

Aparte de pruebas de código, he hecho pruebas de diseño con la extensión AXE. Estas pruebas las he realizado en Firefox

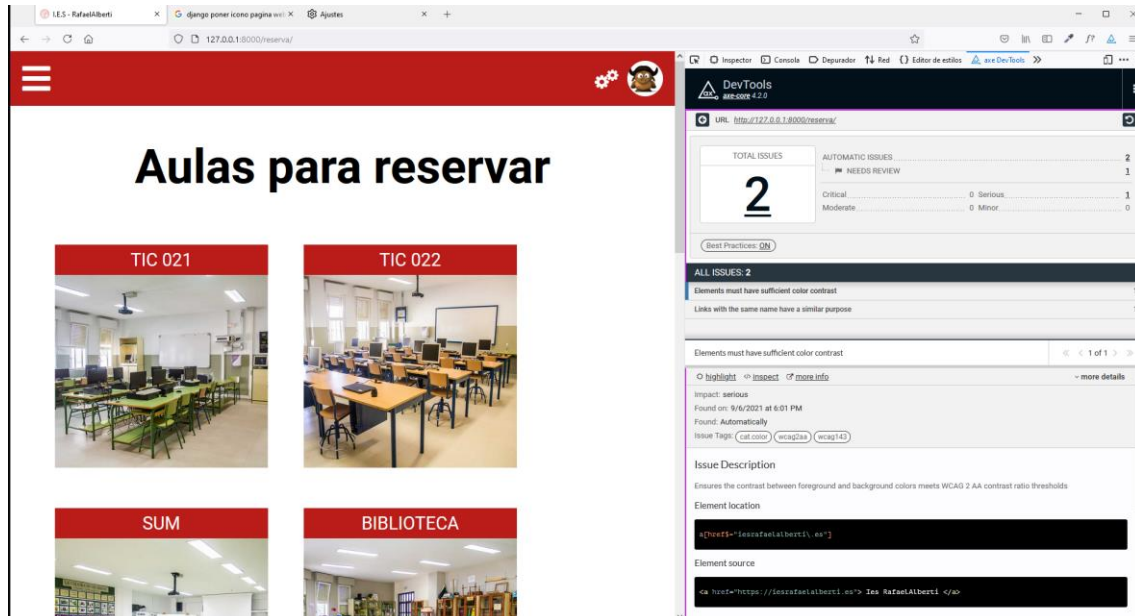


Ilustración 19 Prueba de Axe

Casi todos los errores que me ha dado han sido por contrastes de colores, y porque los botones que tengo con un icono me dicen que tienen que tener texto dentro. Como el botón de la papelera o el del ojo del login.

DESPLIEGUE

Para el despliegue de la aplicación he utilizado un Docker.

Una vez que tenía el proyecto medio terminado decidí montarlo en un docker, para ello tuve que preparar varios archivos. Un docker file, un requirements.txt en el cual vamos a poner las versiones y aplicaciones que vamos a instalar.

```
requirements.txt
1 Django>=3.0,<4.0
2 Pillow
3 pymysql
4 mysqlclient
5 django-mysql
```

Ilustración 20 Configuración requirements.txt

El Dockerfile

```
Users > sergi > AppData > Local > Temp > Rar$Dla8472.38567 > Dockerfile
FROM python:3.9.5
ENV PYTHONUNBUFFERED 1
WORKDIR /proyecto
COPY requirements.txt /proyecto/requirements.txt
RUN pip install -r requirements.txt
COPY . /proyecto

CMD python ./proyecto/manage.py runserver 0.0.0.0:8787
```

En el podemos ver que versión de Python voy a utilizar, digo que la carpeta de trabajo sea proyecto, copio el txt requirements

que como hemos visto anteriormente son aplicaciones que tengo que instalar para preparar el contenedor para poder desplegar django en el docker. Instalo el requirements, y lanzo el proyecto con el comando **py manage.py runserver** con un puerto que yo le asigno en el Docker-compose.

Una vez tenemos el Dockerfile vamos a necesitar un docker-compose.yml para meterle contenido al contenedor:

El Docker-compose:

```
services:
  base:
    image: mariadb
    # restart: always
    environment:
      MYSQL_DATABASE: proyecto
      MYSQL_USER: root
      MYSQL_PASSWORD: root
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - .database:/var/lib/mysql
    expose:
      - 3312
    ports:
      - "3388:3306"

  backend:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 8787:8787
    volumes:
      - ../proyecto
    depends_on:
      - base

  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
volumes:
  .database:
```

Primero tenemos el Docker-compose dividido en tres partes la primera base, backend, y adminer.

Base se encarga de almacenar la base de datos, uso una imagen mariadb, configuro los datos necesarios para entrar en la db, nombre de la db, usuario, contraseña. Le indico el volumen en el cual se va a almacenar, digo que el puerto 3312 se quede expuesto, y ahora en puertos utilizo el 3388:3306 uno es para conectarla con la db y el otro para conectarla con nuestro django.

La parte de backend, como su nombre indica va a llevar el proyecto de Django, aquí tendremos que vincular nuestro Docker file el cual instala las aplicaciones necesarias

para Django y el comando que lanza nuestro servidor, le doy el puerto 8787, en el volumen le asigno la carpeta de mi proyecto, y en depends_on le vinculo la base de datos creada anteriormente.

El apartado de adminer es una forma para poder visualizar los datos de la db desde el navegador como por ejemplo phpmyadmin

Al principio el puerto 8080 era para mí django pero el adminer estaba configurado para usar este puerto así que lo he dejado de esta forma.

Language: English

[MySQL](#) » [Server](#) » [cds](#) » [albums](#) » Alter table

Adminer 3.2.1-dev

[SQL command Dump](#) [Logout](#)

cds

[Create new table](#)

[select albums](#)
[select interprets](#)
[select songs](#)

Alter table: albums

Table name: albums InnoDB utf8_czech_ci [Save](#)

Column name	Type	Length	Options	NULL	AI	+
id	int	11		<input type="checkbox"/>	<input checked="" type="radio"/>	+ ↑ ↓ ×
interpret	interprets		(ON DELETE)	<input type="checkbox"/>	<input type="radio"/>	+ ↑ ↓ ×
title	varchar	100	utf8_czech_ci	<input type="checkbox"/>	<input type="radio"/>	+ ↑ ↓ ×
issued	year	4		<input checked="" type="checkbox"/>	<input type="radio"/>	+ ↑ ↓ ×

Auto Increment: ☐ ☐ Default values ☐ Comment

[Save](#) [Drop](#)

[Partition by](#)

Ilustración 21 Vista de datos desde adminer

Una vez hago el **docker-compose build** y **docker-compose up**. Tendré los contenedores creados, pero tengo que conectar el Django a la db.

Para ello me he tenido que instalar una extensión en visual estudio que es:

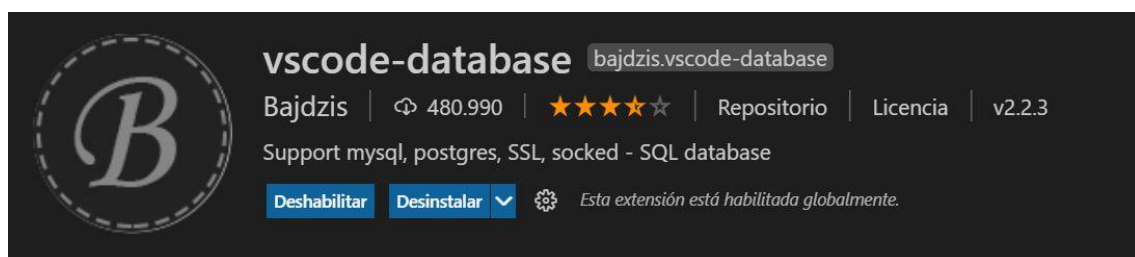
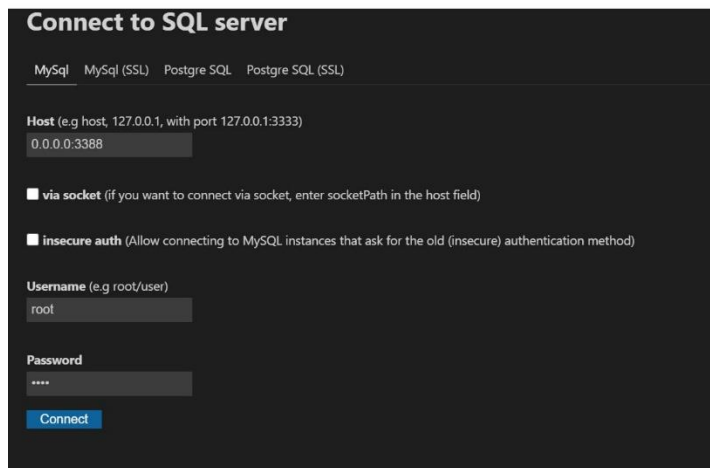


Ilustración 22 Extensión de visual studio para base de datos

Creo una conexión a mi db tengo que poner localhost o 0.0.0.0 y el puerto que he configurado en la parte de db una vez esté conectada nos mostrará la base la izquierda del visual estudio code donde salen los



Connect to SQL server

MySql MySql (SSL) Postgre SQL Postgre SQL (SSL)

Host (e.g host, 127.0.0.1, with port 127.0.0.1:3333)
0.0.0.0:3388

☐ via socket (if you want to connect via socket, enter socketPath in the host field)

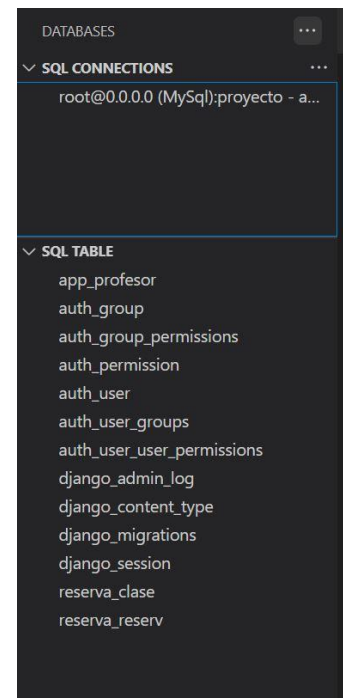
☐ insecure auth (Allow connecting to MySQL instances that ask for the old (insecure) authentication method)

Username (e.g root/user)
root

Password

Connect

ficheros.



Por último, tenía que desplegar la aplicación y generar todos los profesores del centro para ello me he creado un comando custom de Django y lo que hago es que cuando el administrador quiera ejecuta en la consola del Dokcer Python manage.py registro y se almacenarán todos los profesores, sin foto, ya que no tengo todos los recursos para ponerles fotos a todos.

Ejecutar el proyecto

Para ejecutar el proyecto debemos de tener nuestros archivos de docker y el proyecto situados en la misma página:









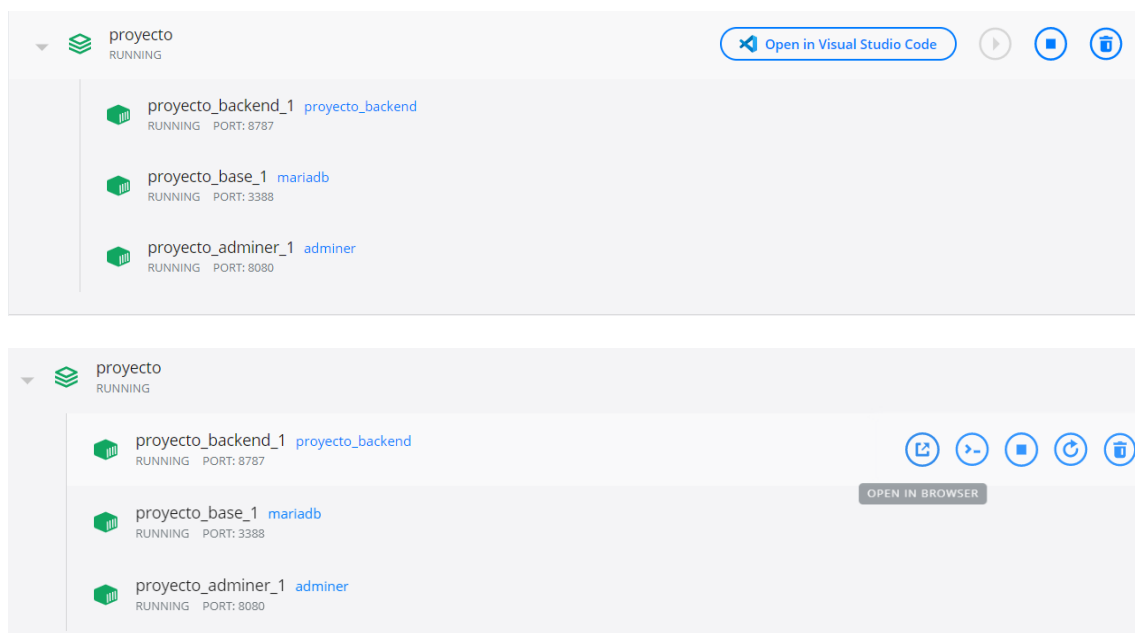
 .database		11/06/2021 9:11	
 proyecto		10/06/2021 13:27	
 docker-compose.yml		10/06/2021 16:42	
 Dockerfile		10/06/2021 16:25	
 proyecto.rar		10/06/2021 16:31	
 requirements.txt		09/06/2021 16:11	

Ilustración 23 Estructura de la carpeta

Abrimos nuestra powershell y nos dirigimos a la ruta de la carpeta con **cd**.

Una vez estemos dentro de la carpeta tendremos que ejecutar el docker-compose para ello vamos a utilizar dos comandos, **docker-compose build**, monta las imágenes y prepara las maquinas, y **docker-compose up** activa los contenedores Dockers para poder trabajar con ellas.

Una vez hayan terminado de lanzarse, podemos ir a nuestro docker



Le damos al botón para que te lo habrá en el navegador, de los tres contenedores que se crean solo dos se pueden abrir el proyecto de Django que es el que se llama **proyecto_backend**. El contenedor de base solo almacena la base de datos y para poder ver todos los datos, podemos verla desde visual estudio con la extensión vista anteriormente o entrando al contenedor de **proyecto_adminer**.

A la hora de descarga el contenido desde dockers la carpeta copy.database hay que cambiarle el nombre a .database si no la base de datos estará vacía.

MANUAL

Al acceder a la aplicación web lo primero que nos mostrará será la página de inicio de sesión del usuario. En la cual podremos nuestras credenciales



I.E.S. Rafael Alberti

INICIO DE SESIÓN

Nombre:

Contraseña:

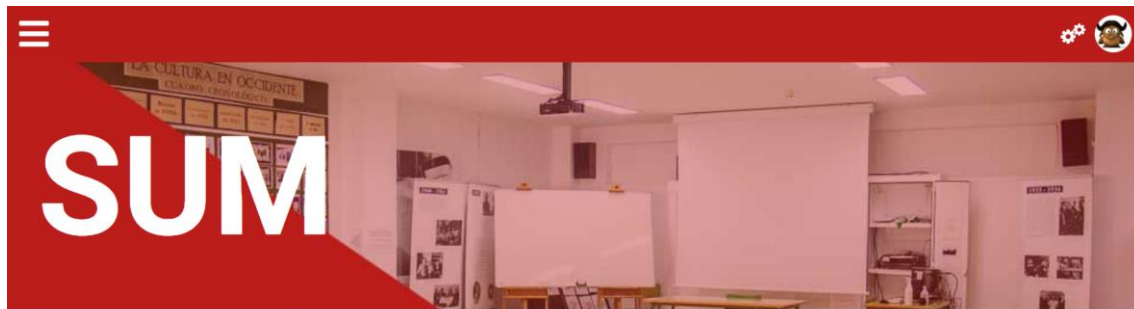
Una vez entremos nos mostrará dos opciones, reservar un aula o temas de convivencia, este manual se centra en la reserva de aulas con lo cual hacemos clic en nuestro apartado.



Al entrar en la zona de Reservar un aula nos mostrará todas las aulas que se suelen reservar en el colegio.



Para reservar un aula, tendremos que seleccionar la clase que queremos reservar, en este ejemplo elegiremos la sala de usos múltiples (SUM)



JUNIO 2021						
Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Dentro nos encontraremos este calendario en el cual se pueden seleccionar todos los días laborales, ya que un domingo o un sábado no se pueden reservar las clases.

Al seleccionar un día nos mostrará que eventos hay ese día y a su vez un formulario en el cual podemos hacer una reserva si queremos.



JUNIO 2021						
Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Reservas

No hay reserva este día

Formulario

Día seleccionado:

2021-6-11

Hora de inicio:

08:00

Hora final:

09:00

Motivo:

En el formulario podemos ver que para el día 11 de junio no hay reservas hechas así que vamos a crear una nosotros.

Reservas

No hay reserva este día

Formulario

Día seleccionado:

2021-6-11

Hora de inicio:

08:00

Hora final:

10:00

Motivo:

Otro

Nº Alumnos:/ Max alumnos: 100

75

Curso:

Otros

Enviar

- Deberemos de rellenar el formulario adecuadamente para poder realizar la reserva con éxito.
- Deberemos de elegir la hora de inicio menor a la hora final.
- Si la hora de inicio es la misma a la hora final también dará un error.
- Tendremos que seleccionar un día que no haya pasado, ya que si la fecha ha pasado no tiene mucho sentido realizar una reserva.
- El número de alumnos tiene que ser menor al número máximo que soporta una clase.
- Una vez hayamos rellenado todos los campos adecuadamente la reserva se realizará, si por lo que sea hemos rellenado mal el formulario la web nos

devolverá el siguiente error



El formulario no es correcto

JUNIO 2021

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Y volveremos a poder realizar el formulario por ultimo si pinchamos en una clase que ya hay reservas hecha la pagina nos lo mostrará de la siguiente forma



JUNIO 2021

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Reservas 

Javier Ortega Con el curso: otros
examen de 08:00 a 10:00

Formulario

Día seleccionado:

Hora de inicio:

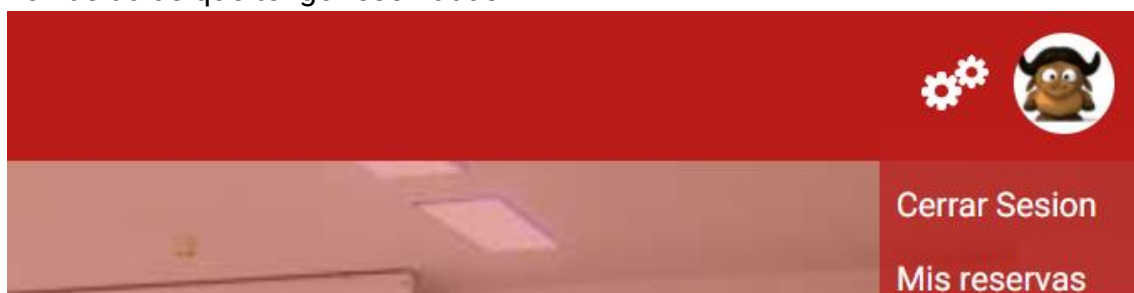
Hora final:

Motivo:

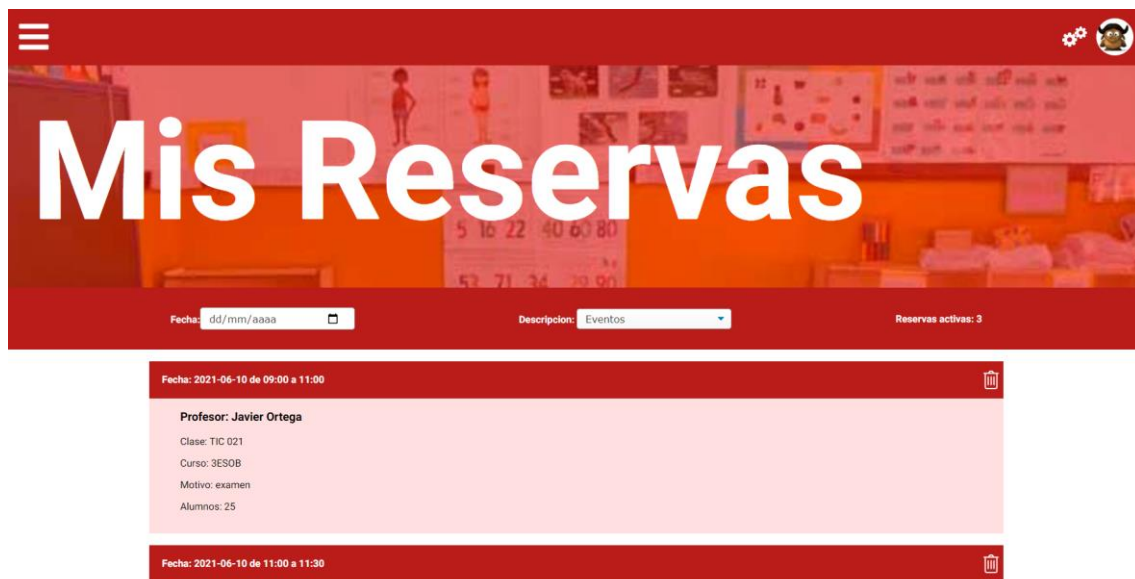
Nº Alumnos:/ Max alumnos: 100

Curso:

Por ultimo hablar de la existencia de dos tipos de menus diferentes en la esquina superior derecha tendremos la opcion de cerrar sesion y la opcion de poder ir a ver las aulas que tengo reservadas



Si le damos a la Parte de Mis reservas:



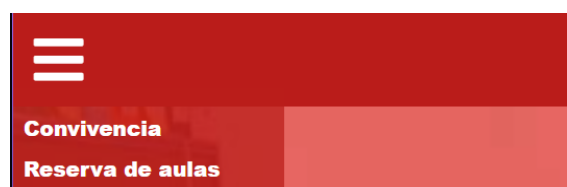
Nos mostrará todas las reservas de un profesor desde el día actual hacia delante, aunque podremos filtrar por día en concreto y por tipo de evento.

Hay dos tipos de roles, el usuario normal que son todos los profesores y el administrador y la diferencia que hay entre uno u otro es que el administrador puede ver todas las reservas de todos los profesores, mientras que el profesor solo puede ver las suyas.

Si nos fijamos en la foto del perfil del usuario al lado hay unos engranajes eso quiere decir que es administrador.

Podremos eliminar también nuestras reservas si nos fijamos en cada reserva. En la esquina superior derecha hay un icono de una papelera, pues desde ahí podremos eliminar la reserva.

Por último, en la esquina superior izquierda tendremos otro menú. En el cual podremos acceder a él si pinchamos encima de las tres barritas. Si hacemos clic en 'Reserva de aulas' nos volverá a mandar al menú donde aparecen todas las aulas para poder realizar otra reserva de la clase que queramos.



CONCLUSIONES

La idea principal se parecía al resultado final ya que el tema no ha variado mucho, pero si ha variado mucho la forma con la que iba a representar las clases y las ideas que tenía iniciales con el resultado conseguido.

La primera idea que tuve fue hacer un mapa, y que en el mapa se vieran todas las clases que se puedan reservar, pero después de pensarlo un par de veces creo que era complicado de realizar y poco intuitivo, ya que no todas las aulas se pueden reservar, después si se quieren meter más aulas es difícil de ampliar, y menos si hablamos de que tenemos que cambiar el mapa del centro.

Otra desventaja es que el centro tiene dos plantas y al principio no sabía si algunas iban a estar arriba otras abajo en fin que esta idea la descarté.

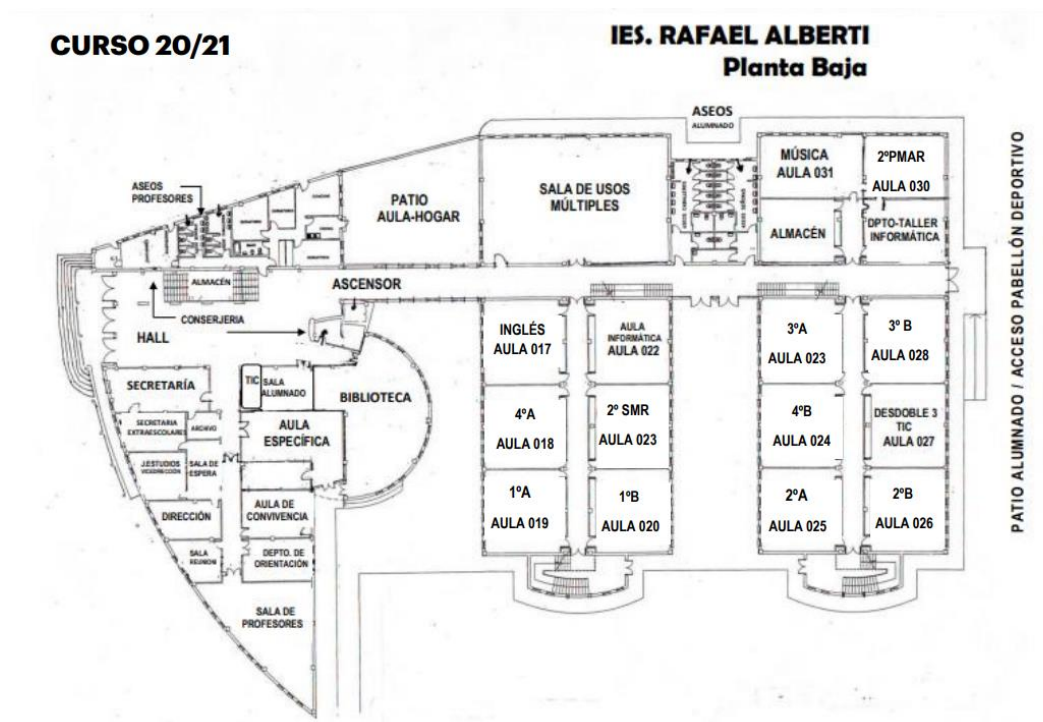


Ilustración 24 Mapeado del instituto

Un plano del mapa

Mi segunda idea fue el mostrar todas las aulas como están ahora, creo que con sus fotos y el nombre de cada zona cualquier profesor sabe a que aula se está refiriendo, aparte de que si queremos meter aulas nuevas solo con meter una entrada en la base de datos ya estarían introducidas.

Por otra parte, sabia al 100% que un calendario tenía que utilizar ya que se tiene que ver reflejada la fecha en la cual vamos a hacer nuestra reserva, al principio pensé en un calendario global en la que se pudiera ver que aulas estaban reservadas y a qué horas, pero al tener 10 aulas se me iban a pisar muchas de estas horas con las diferentes aulas y quizá podría ser un agobio al tener tanta información a la vez. Así que decidí mostrar dentro de cada aula sus eventos y en vez de en el calendario fuera de este con todos los datos.

Por último, como una especie de historial de quien ha reservado aulas o para asegurar que aulas he reservado añadí el apartado “Mis reservas” en el cual podemos ver todas las reservas que tangamos.

Mejoras futuras añadirle el otro apartado de convivencia para poder tener una aplicación con una mayor funcionalidad.

Añadir al calendario una función para poder reservar todos los días a una misma hora durante x días del año.

A la hora de generar los usuarios poder añadirle la foto automáticamente, yo no se lo he podido añadir ya que no disponía de las fotografías. Los profesores que tienen fotografías se las he puesto de forma manual.

Añadir una función para que genere PDF de las reservas que hay a la semana, pero tenía problemas a hora de especificar el número de días.

ÍNDICE DE TABLAS E IMÁGENES

Ilustración 1 Página de Python para descargarlo.....	4
Ilustración 2 Prueba de que Python está instalado.....	4
Ilustración 3 instalación de Django	5
Ilustración 4 Prueba de que Django está instalado.....	5
Ilustración 5 Configuración de db en el archivo settings.....	6
Ilustración 6 Creacion de la db en Heidi.....	6
Ilustración 7 Hacer migraciones para Django 1.0	7
Ilustración 8 Hacer migraciones para Django 1.1	7
Ilustración 9 Instalación de Pillow	7
Ilustración 10 Página de descarga a Docker	7
Ilustración 11 Logo IES Rafael Alberti	8
Ilustración 12 Wireframe 1	9
Ilustración 13 Wireframe 2	10
Ilustración 14 Login del mockup.....	10
Ilustración 15 Diseño UML	11
Ilustración 16 Diseño de casos de uso	12
Ilustración 17 Consola de git.....	22
Ilustración 18 Test realizados	24
Ilustración 19 Prueba de Axe	25
Ilustración 20 Configuración requirements.txt	25
Ilustración 21 Vista de datos desde adminer	27
Ilustración 22 Extensión de visual studio para base de datos	27
Ilustración 23 Estructura de la carpeta	29
Ilustración 24 Mapeado del instituto.....	36

BIBLIOGRAFÍA

Material utilizado:

Descargar Docker (2021).

Docker: <https://hub.docker.com/editions/community/docker-ce-desktop-windows>

Descargar Python. (2021). Python. <https://www.python.org/downloads/>

Pillow. (2021). Pillow. <https://pypi.org/project/Pillow/>

MySQLClient. (2021). MySQLClient. <https://pypi.org/project/mysqlclient/>

Descargar Django. (2021). Django. <https://www.djangoproject.com/download/>

Webs donde he sacado información

Para servidor y despliegue

Documentación de Django (2021).

Django <https://docs.djangoproject.com/en/3.2/>

Preguntas en Stackoverflow. (2021). Stackoverflow.

<https://stackoverflow.com/>

Documentación Docker. (2021). Docker. <https://docs.docker.com/get-started/>

Para Diseño:

Al principio mire muchas páginas para ver algunas ideas de menú, de composición de fotografías, distribución, botones, etc.

Paginas como:

Awwwards. (2021). Awwwards. <https://www.awwwards.com/>

30 Secondsofcode. (2021) <https://www.30secondsofcode.org/css/p/1>

Y páginas que me han ayudado:

W3schools. (2021). W3schools. <https://www.w3schools.com/>

Lenguajecss. (2021). Lenguajecss. <https://lenguajecss.com/>

Para cliente

w3schools. (2021). w3schools. <https://www.w3schools.com/js/DEFAULT.asp>

Ajax. (2021). Ajax. <https://developer.mozilla.org/es/docs/Web/Guide/AJAX>

Por ultimo las imágenes utilizadas en mi proyecto has sido sacadas por mí con ayuda de Alejandro, y la de los profesores de la Moodle: