

Gestión de Proxmox mediante Python



Jesús Pulido Porcuna
Ciclo Superior de Administración de Sistemas Informáticos en Redes
IES Medina Azahara
08/06/2023



Administración de proxmox mediante Python © 2024 by Jesús Pulido is licensed under Creative Commons Attribution-NonCommercial 4.0 International. To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc/4.0/>

Índice

1. DEFINICIÓN Y ANÁLISIS CONTEXTUAL	3
1.1 Contexto y Justificación	3
1.2 MARCO LEGAL	4
1.3 ALCANCE DEL PROYECTO	4
1.4 ALTERNATIVAS Y PROPUESTA DE SOLUCIÓN	4
2. ANÁLISIS DE REQUISITOS	6
2.1 Recursos humanos:	6
2.1 Requisitos legales:	6
2.3 Hardware:	6
2.4 Software:	7
3. TEMPORALIZACIÓN	8
3.1 IDENTIFICACIÓN DE FASES Y TAREAS	8
4. DOCUMENTACIÓN TÉCNICA	9
4.1 Diseño de la base de datos	9
4.1.1 Diseño del esquema entidad relación	10
4.1.2 Esquema de entidades	10
4.1.3 Transformación	11
4.1.4 Normalización	13
4.1.5 Creación de la base de datos	15
4.2 Diseño de las funciones de gestión de proxmox	20
4.3 Diseño de la aplicación	28
4.3.1 Instalaciones previas	28
4.3.2 Creación del proyecto	28
4.3.3 Configuración del proyecto	29
4.3.4 Creación de la app	30
4.3.5 Migración de la base de datos.	31
4.3.6 Creación de las vistas	32
4.3.7 Creación de la página de login	36
4.3.8 Página de inicio	37
4.3.9 Página de usuarios	39
4.3.8 Página de máquinas virtuales	46
4.3.8 Página de contenedores	51
5. ANÁLISIS ECONÓMICO	56
6. SEGUIMIENTO Y CONTROL	58
6.1 EVALUACIÓN GENERAL	58
7. POSIBLES MEJORAS	60
8. FUENTES DE DOCUMENTACIÓN	61
9. Anexos	62

1. DEFINICIÓN Y ANÁLISIS CONTEXTUAL

1.1 Contexto y Justificación

La virtualización de equipos y servicios es una práctica que cada día gana más fuerza. Esta práctica es muy útil por diversos motivos.

En primer lugar, permite un mayor aprovechamiento de los recursos. Muchas veces en empresas, o incluso en el ámbito educativo, se necesita que un equipo trabaje a un alto rendimiento durante un periodo de tiempo (una compilación o la ejecución de un programa pesado), pero la mayoría del tiempo se desperdician recursos. Mediante el uso de la virtualización podemos repartir recursos entre todos los usuarios del sistema, distribuyendo mejor los recursos haciendo que cada usuario solamente use los que necesita en el momento en el que le hace falta.

Por otro lado, la virtualización nos da la posibilidad de crear entornos aislados, aislando posibles errores y eliminando los errores de compatibilidad entre proyectos, como librerías incompatibles o interferencias entre direcciones ip o redes en general, errores más comunes de lo que pueda parecer a priori.

Una correcta gestión de la virtualización también permite asignar distintos recursos a distintos tipos de usuarios, limitando el acceso de estos grupos de usuarios y dejándolos aislados de los recursos o datos que no les competen

Como podemos ver, la virtualización es un proceso muy útil en varios aspectos. Sin embargo, el problema es la propia creación de entornos virtuales, ya que no todos los usuarios están familiarizados con estos y se necesita de una buena base de conocimientos que no todos los usuarios tienen por qué tener.

Otra desventaja, sería el tiempo necesario para la creación de entornos virtuales, ya que si se hacen de forma manual, requerirá de una persona dedicada específicamente a esta labor.

Este proyecto surge como solución a estos problemas. En primer lugar, la creación de scripts automatiza y facilita la gestión de un entorno virtual, en este caso, proxmox, ayudando tanto en la gestión de usuarios como de máquinas virtuales y/o contenedores. La segunda parte del proyecto estará enfocada en la creación de una interfaz que ayude a un usuario estándar, sin necesidad de grandes conocimientos sobre el tema, a acercarse a la gestión de proxmox y de sus recursos propios.

1.2 MARCO LEGAL

En este proyecto, no existe un marco legal que nos afecte de manera directa.

1.3 ALCANCE DEL PROYECTO

- Análisis de las necesidades del ámbito específico en el que se realizará el proyecto.
- Diseño de una base de datos que recoja los datos según las necesidades analizadas.
- Creación de scripts para cumplir con las funcionalidades requeridas (crear usuarios, crear máquinas...).
- Diseño de una página web, a modo de interfaz de usuario con proxmox, sencilla, actualizada y realizable:
 - Diferentes vistas que se muestran al usuario.
 - Definición del modelo que refleje la base de datos.
 - Implementación de las funciones necesarias.

1.4 ALTERNATIVAS Y PROPUESTA DE SOLUCIÓN

Para este proyecto, nos hemos basado en 3 pilares:

- Python como lenguaje de programación
- Proxmox como sistema de virtualización
- MySQL como base de datos

Sin embargo, existen alternativas a estos, como las siguientes

- Lenguajes de programación como C#, php o JavaScript: la principal razón por la que se ha decidido utilizar python, es principalmente por su sencillez y librerías para interactuar con proxmox. Utilizar cualquier otro lenguaje, habría requerido más tiempo de investigación y la necesidad de aprender un nuevo lenguaje, lo cual supone una inversión de tiempo bastante grande
- Software de virtualización de servidores como VMware vSphere, Red Hat virtualization u Oracle VM Server: en este caso, se ha decantado utilizar proxmox por diversos motivos, como que tiene una buena comunidad, la cual lo respalda, es gratuito y de código abierto.
- Gestor de bases de datos como SQLite, MongoDB, MariaDB o Influx: Si bien podríamos habernos decantado por SQLite, siendo por defecto el que usa Django, la mayor seguridad que aporta MySQL hace que nos decantemos por este último. En cuanto a Mongo, al dar gran importancia a la coherencia entre los datos y las relaciones entre tablas, nos hace descartarlo. Por último, la razón principal para utilizar MySQL en lugar de cualquier otra base de datos, es la familiaridad que tenemos con esta, mientras que cumple los puntos anteriormente señalados.

2. ANÁLISIS DE REQUISITOS

2.1 Recursos humanos:

Para la realización de este proyecto, podemos disponer de una sola persona que lo haga todo. Sin embargo, la mejor opción es contar con al menos un equipo de 2 personas, ya que el proyecto cubre varios campos muy diferenciados. Por un lado, la parte de administración de sistemas, como es la creación de funciones de gestión del sistema y la configuración del propio proxmox. Por otro lado, tenemos la programación web, tanto el estilo de la página en sí como la optimización de esta. También podríamos disponer de un tercer integrante que gestione la base de datos y los accesos a esta de disponer de los recursos necesarios.

2.1 Requisitos legales:

En principio, no requerimos de unos requisitos legales específicos. Como complemento, y ya que utilizamos datos personales de los usuarios, podríamos incluir la ley de protección de datos, es decir, la autorización de los usuarios para el uso de sus datos.

2.3 Hardware:

En cuanto a hardware, el servidor en el que se instale proxmox debe de tener al menos las siguientes características:

- Procesador en 64bits (Intel EMT64 ó AMD64), de preferencia con múltiples núcleos.
- Tarjeta Madre con soporte para virtualización.
- 2 GB de RAM por máquina virtual que vayamos a tener levantada a la vez.
- Discos duros rápidos (15k rpm SAS, SSD) para el SO y al menos 1T de capacidad para almacenamiento.
- Soporte para RAID por hardware o ZFS.
- Tarjetas de red.

Estos requisitos podrían ser ampliables durante el seguimiento del proyecto de ser necesario, sobre todo, una vez conocidos la cantidad de usuarios y máquinas que trabajarán con el.

2.4 Software:

Necesitaremos el siguiente software:

- SO proxmox instalado en el servidor
- Python3 y django
- Servidor mysql

3. TEMPORALIZACIÓN

3.1 IDENTIFICACIÓN DE FASES Y TAREAS

ABRIL 2024			
Semana 1	Semana 2	Semana 3	Semana 4
	1	2	3
	Búsqueda de librerías de python para Proxmox		
7	8	9	10
	Pruebas con Proxmoxer y recabación de información sobre ella		

MAYO 2024			
Semana 1	Semana 2	Semana 3	Semana 4
	1	2	3
	Diseño de las funciones de gestión		
7	8	9	10
	Pruebas de las funciones		
21	22	23	24
	Modificación de las funciones		
28	29	30	31
		Diseño e implementación de la base de dato	

JUNIO 2024			
Semana 1	Semana 2	Semana 3	Semana 4
	1	2	3
	Diseño de la aplicación web		
7	8	9	10
	Pruebas finales del proyecto		
21	22	23	24
	Elaboración de la memoria del proyecto		

4. DOCUMENTACIÓN TÉCNICA

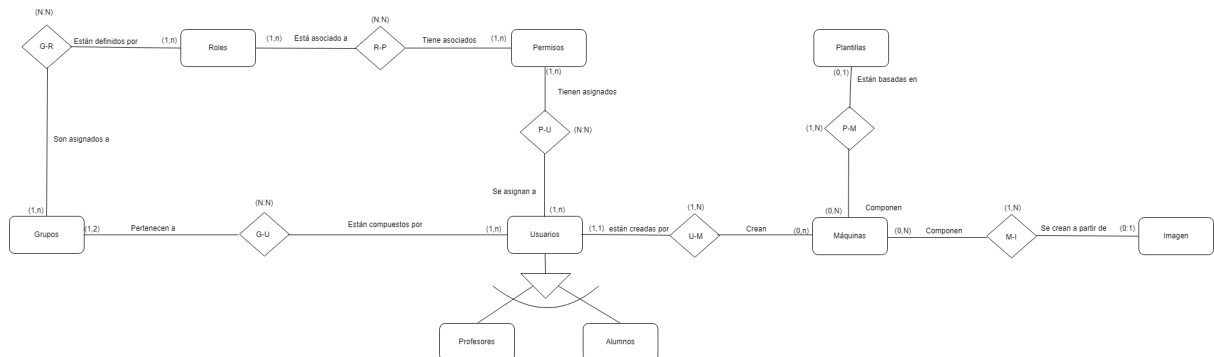
El proyecto estará dividido en 3 apartados, a su vez divididos en varios subapartados

4.1 Diseño de la base de datos

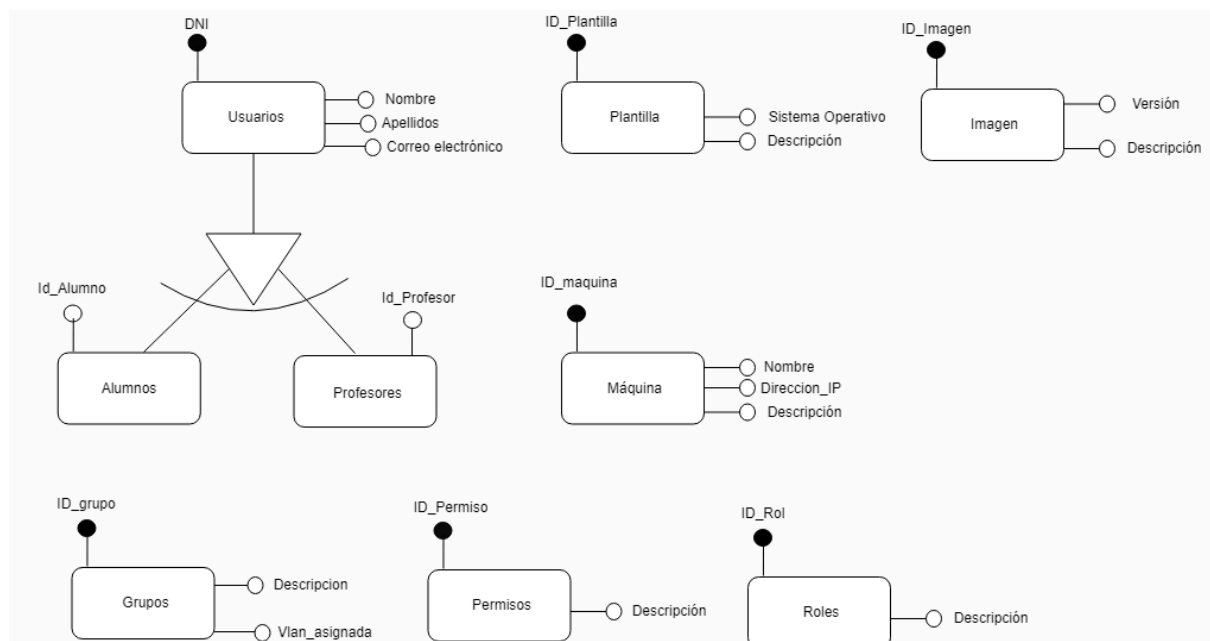
Nuestra base de datos deberá recoger la siguiente información:

1. Información referente a los usuarios. Los usuarios podrán ser alumnos o profesores y de estos se desea conocer su nombre, apellidos, dni y correo. En cuanto a los alumnos, estos estarán identificados con un ID_Alumno y los profesores con un ID_Profesor
2. Información referente a los grupos. Cada curso de alumnos tendrán su correspondiente grupo en la base de datos. En este grupo se guarda la información de la vlan asignada a cada clase y una descripción opcional del grupo
3. Información referente a los roles definidos en proxmox. Se definirán varios roles en proxmox. Estos roles se identifican con un id roll y estarán conformados por permisos.
4. Información referente a los permisos. En esta tabla se recogerán los distintos permisos definidos en proxmox. Estos estarán identificados por el ID que tienen dentro de proxmox y se relacionarán con los roles. Se considera que un usuario puede tener permisos especiales a parte de los que les asigna el grupo.
5. Información sobre las máquinas virtuales. Se ha decidido añadir una tabla específica para las máquinas virtuales, ya que estas tendrán información que puede ser útil a futuro. Sobre estas máquinas recogeremos los siguientes datos: ID de la máquina, nombre de esta, una descripción con datos relevantes sobre la máquina (servicios que tiene instalados, etc...) y la dirección ip de la máquina. Las máquinas deberán estar relacionadas con un único usuario.
6. Información de las plantillas almacenadas. Estas plantillas estarán basadas en un sistema operativo y tendrán una descripción al igual que las máquinas virtuales.
7. Información sobre las imágenes almacenadas. Estas imágenes tendrán un Id para identificarlas, una breve descripción y también se guardará la versión a la que pertenecen.

4.1.1 Diseño del esquema entidad relación



4.1.2 Esquema de entidades



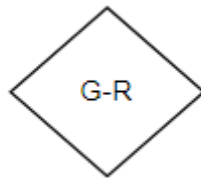
4.1.3 Transformación

En este apartado, seguiremos las reglas de transformación para pasar del modelo entidad relación al modelo relacional

Pasos previos:

Comenzaremos por analizar la jerarquía usuarios -> alumnos, profesores
Dentro de esta jerarquía, el padre es el que cuenta con casi todos los atributos y relaciones, por lo que eliminaremos a los hijos, quedándonos con el supertipo.

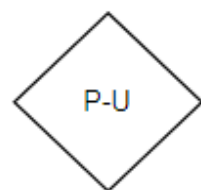
Por otra parte, la información proporcionada por las claves tanto de alumno como profesor, pueden unificarse en un solo campo llamado rol, eliminando ambos IDs ya que los usuarios estarán identificados por el DNI



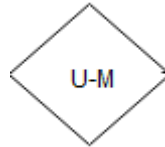
En la relación grupo usuario, técnicamente tenemos una relación 2:N, por lo que podríamos crear simplemente un atributo grupo1 y grupo2 en la tabla usuarios que se relaciona con la tabla grupos. Sin embargo y con la finalidad de no tener un atributo que la mayoría de veces estará vacío, trataremos la relación como una N:N. Al tratarse de una relación N:N aplicaremos la regla de transformación correspondiente, donde se crea una tabla para cada entidad y una para la relación, teniendo esta última los identificadores de las otras dos, siendo la clave principal la suma de las claves principales de las otras tablas.



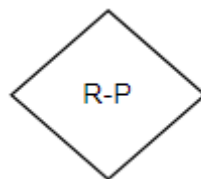
De nuevo, una relación N:N, por lo que aplicaremos la misma regla, creando de nuevo las 3 tablas correspondientes



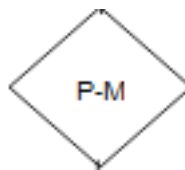
Relación N:N entre dos entidades, por lo que seguiremos la metodología anterior. Creamos una tabla por cada entidad y una más por la relación. Esta última tendrá como clave principal la suma de las claves principales de las otras dos entidades



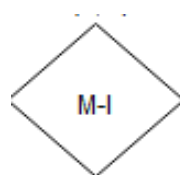
En este caso, tenemos una relación 1:N donde la que participa con cardinalidad máxima 1 lo hace de forma total. Crearemos una tabla para cada entidad y la clave principal de la que participa con cardinalidad máxima 1 pasará a ser clave foránea de la otra.



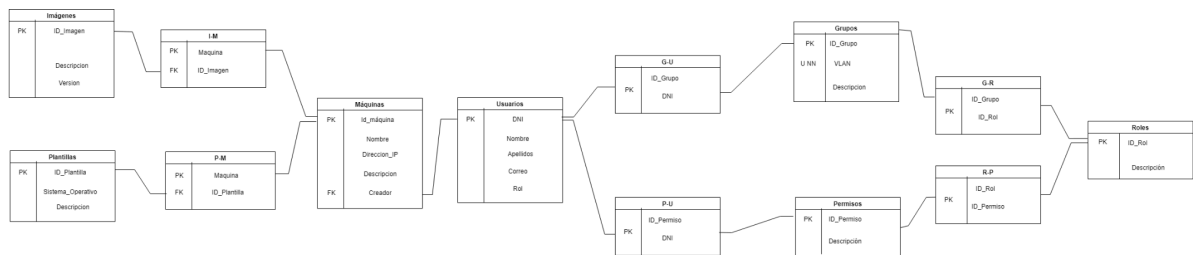
Se vuelve a repetir el caso anterior. Tendremos una tabla con los atributos de la relación(si los hubiera) y las claves de las tablas correspondientes a las dos entidades que participan en la relación.



Se trata de una relación 1:N donde la que participa con cardinalidad máxima 1 lo hace de forma parcial. Crearemos 1 tabla para cada entidad y una para la relación. La tabla de la relación tendrá como clave principal la de la que participa con cardinalidad n y la clave principal de la que participa con cardinalidad 1 pasará a ser clave foránea.



Volvemos a encontrar una relación 1:N, donde la entidad que participa con cardinalidad máxima 1, lo hace de forma parcial. Aplicando la regla de transformación anterior, crearemos una tabla para cada una de las entidades y una tabla para la relación, teniendo esta última como clave principal la clave de la entidad que participa con cardinalidad máxima n y como clave foránea la clave principal de la entidad que participa con cardinalidad máxima 1



4.1.4 Normalización

Tabla usuarios:

- Análisis de dependencias funcionales: Todos los atributos dependen de la clave candidata y no existen dependencias entre claves externas a esta (consideramos que nombre + apellidos puede repetirse).
- La tabla cumple con BC, por tanto no hay necesidad de modificaciones

Tabla grupos:

- Análisis de dependencias funcionales: Todos los atributos dependen de la clave candidata. Existen dependencias funcionales:
 - Todas las claves dependen de VLAN, ya que esta identifica a las demás (Cada grupo tiene una única VLAN asignada y esta VLAN no se puede repetir)
- La tabla no cumple con BC.
- Solución: La clave candidata con la que existen dependencias pasa a ser clave secundaria, pasando esta a ser única y no nula.

Tabla permisos

- Análisis de dependencias funcionales: Todas las claves dependen de la clave candidata y no existen dependencias entre atributos externos a esta.
- La tabla cumple con BC.

Tabla roles

- Análisis de dependencias funcionales: Todas las claves dependen de la clave candidata y no existen dependencias entre atributos externos a esta.
- La tabla cumple con BC.

Tabla máquinas

- Análisis de dependencias funcionales: Todas las claves dependen de la clave candidata y no existen dependencias entre atributos externos a esta.
- La tabla cumple con BC

Tabla Plantillas

- Análisis de dependencias funcionales: Todas las claves dependen de la clave candidata y no existen dependencias entre atributos externos a esta.
- La tabla cumple con BC

Tabla Imágenes

- Análisis de dependencias funcionales: Todas las claves dependen de la clave candidata y no existen dependencias entre atributos externos a esta.
- La tabla cumple con BC

4.1.5 Creación de la base de datos

Como ya hemos explicado con anterioridad, crearemos la base usando el SGBD MySQL. En este caso, trabajaremos con la herramienta SQL Workbench. Se ha añadido al principio del archivo las órdenes `DROP DATABASE IF EXISTS` y `CREATE DATABASE` para que al ejecutarlo, limpie la base de datos, creando las tablas limpias.

Para todas las tablas que tengan un campo descripción, se ha definido que sea de tipo texto sin ninguna restricción. Esto es para que el usuario pueda poner lo que necesite en este campo.

En primer lugar crearemos la tabla usuarios, cuya clave principal será dni. Esta clave debe tener 8 dígitos y 1 letra. El campo nombre no puede contener dígitos, al igual que los apellidos. El campo email será de la forma letras@letras.letras y por último el campo roll será de tipo enum, es decir, se puede elegir entre una de las opciones disponibles únicamente. en este caso alumno o profesor.

```

1 • DROP DATABASE if exists tfg;
2
3 • CREATE DATABASE tfg;
4
5 • USE tfg;
6
7 -- Creación de la tabla Usuarios
8 • CREATE TABLE Usuarios (
9     DNI CHAR(9) PRIMARY KEY,
10    Nombre VARCHAR(50) NOT NULL,
11    Apellidos VARCHAR(50) NOT NULL,
12    Correo VARCHAR(100) NOT NULL,
13    Rol ENUM('profesor', 'alumno') NOT NULL,
14    CONSTRAINT chk_DNI CHECK (DNI REGEXP '^[0-9]{8}[A-Z]$'),
15    CONSTRAINT chk_Nombre CHECK (Nombre REGEXP '^[a-zA-Z]+$'),
16    CONSTRAINT chk_Apellidos CHECK (Apellidos REGEXP '^[a-zA-Z]+$'),
17    CONSTRAINT chk_Correo CHECK (Correo REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]+$')
18 );

```

Para la tabla grupos, la clave principal será id_grupo y la única restricción que tenemos es que vlan no puede ser nulo y debe de estar formada por VLAN y un número.

En la tabla permisos no habrá ninguna restricción y su clave principal será id_permiso

```

20 -- Creación de la tabla Grupos
21 • CREATE TABLE Grupos (
22     ID_Grupo VARCHAR(20) PRIMARY KEY,
23     Descripcion TEXT,
24     Vlan VARCHAR(10) NOT NULL,
25     CONSTRAINT chk_Vlan CHECK (Vlan REGEXP '^[VLAN][0-9]{2}$')
26 );
27
28 -- Creación de la tabla Permisos
29 • CREATE TABLE Permisos (
30     ID_Permiso VARCHAR(20) PRIMARY KEY,
31     Descripcion TEXT
32 );

```

Para los roles, en este caso referentes a los roles de proxmox, es decir, al conjunto de permisos, la clave principal será id_rol y no podrá contener caracteres especiales.

En cuanto a las máquinas, la clave principal será id_maquina. La clave creador será una clave foránea y referencia a dni de la tabla usuarios. El nombre de la máquina no podrá contener caracteres especiales

```
34  -- Creación de la tabla Roles
35  • CREATE TABLE Roles (
36      ID_Rol VARCHAR(20) PRIMARY KEY,
37      Descripcion TEXT,
38      CONSTRAINT chk_ID_Rol CHECK (ID_Rol REGEXP '^[a-zA-Z0-9]+$')
39  );
40
41  -- Creación de la tabla Máquinas
42  • CREATE TABLE Maquinas (
43      ID_Maquina INT PRIMARY KEY,
44      Nombre VARCHAR(50) NOT NULL,
45      Descripcion TEXT,
46      Direccion_IP VARCHAR(15) NOT NULL,
47      Creador VARCHAR(20) NOT NULL,
48      FOREIGN KEY (Creador) REFERENCES Usuarios(DNI),
49      CONSTRAINT chk_Maquina CHECK (Nombre REGEXP '^[a-zA-Z0-9]+$')
50  );
```

La tabla imágenes tendrá como clave principal id_imagen y no tendrá ninguna restricción. De igual forma, la tabla plantillas tendrá como clave id_plantilla y tampoco contendrá ninguna restricción.

```
52  -- Creación de la tabla Imagenes
53  • CREATE TABLE Imagenes (
54      ID_Imagen VARCHAR(20) PRIMARY KEY,
55      VersionImagen VARCHAR(20),
56      Descripcion TEXT
57  );
58
59  -- Creación de la tabla Plantillas
60  • CREATE TABLE Plantillas (
61      ID_Plantilla VARCHAR(20) PRIMARY KEY,
62      Sistema_Operativo VARCHAR(50),
63      Descripcion TEXT
64  );
```


Para las tablas de las relaciones, se implementarán como se han definido anteriormente.

En primer lugar, la tabla grupos-roles tendrá como primary key la combinación de id grupo e id roll, cada una foreign key relacionada con su respectiva tabla.

Por otra parte, la tabla roles-permisos estará compuesta de id rol e id permiso, ambas claves principales en su conjunto.

```
66  -- Creación de las relaciones
67
68  -- Relación entre Grupos y Roles
69 • CREATE TABLE Grupos_Roles (
70      ID_Grupo VARCHAR(20),
71      ID_Rol VARCHAR(20),
72      PRIMARY KEY (ID_Grupo, ID_Rol),
73      FOREIGN KEY (ID_Grupo) REFERENCES Grupos(ID_Grupo),
74      FOREIGN KEY (ID_Rol) REFERENCES Roles(ID_Rol)
75  );
76
77  -- Relación entre Roles y Permisos
78 • CREATE TABLE Roles_Permisos (
79      ID_Rol VARCHAR(20),
80      ID_Permiso VARCHAR(20),
81      PRIMARY KEY (ID_Rol, ID_Permiso),
82      FOREIGN KEY (ID_Rol) REFERENCES Roles(ID_Rol),
83      FOREIGN KEY (ID_Permiso) REFERENCES Permisos(ID_Permiso)
84  );
```

Para la tabla permisos-usuarios, la clave principal estará formada por id permiso y dni.

```
86  -- Relación entre Permisos y Usuarios
87 • CREATE TABLE Permisos_Usuarios (
88      ID_Permiso VARCHAR(20),
89      DNI CHAR(9),
90      PRIMARY KEY (ID_Permiso, DNI),
91      FOREIGN KEY (ID_Permiso) REFERENCES Permisos(ID_Permiso),
92      FOREIGN KEY (DNI) REFERENCES Usuarios(DNI)
93  );
94
95  -- Relación entre Usuarios y Grupos
96 • CREATE TABLE Usuarios_Grupos (
97      Usuario CHAR(9),
98      ID_Grupo VARCHAR(20),
99      PRIMARY KEY (Usuario, ID_Grupo),
100     FOREIGN KEY (Usuario) REFERENCES Usuarios(DNI),
101     FOREIGN KEY (ID_Grupo) REFERENCES Grupos(ID_Grupo)
102  );
```

La clave principal de la tabla maquinas-plantillas será id maquina mientras que id plantilla será una clave foránea

De igual forma, la clave principal de maquinas-imagenes también será id maquina mientras que id imagen será una clave foránea

```
104 -- Relación entre Máquinas y Plantillas
105 • CREATE TABLE Maquinas_Plantillas (
106     ID_Maquina INT,
107     ID_Plantilla VARCHAR(20),
108     PRIMARY KEY (ID_Maquina),
109     FOREIGN KEY (ID_Maquina) REFERENCES Maquinas(ID_Maquina),
110     FOREIGN KEY (ID_Plantilla) REFERENCES Plantillas(ID_Plantilla)
111 );
112
113 -- Relación entre Máquinas e Imágenes
114 • CREATE TABLE Maquinas_Imagenes (
115     ID_Maquina INT,
116     ID_Imagen VARCHAR(20),
117     PRIMARY KEY (ID_Maquina),
118     FOREIGN KEY (ID_Maquina) REFERENCES Maquinas(ID_Maquina),
119     FOREIGN KEY (ID_Imagen) REFERENCES Imagenes(ID_Imagen)
120 );
121
```

Por último, la relacion usuarios maquinas tendrá como clave principal dni, clave foránea proveniente de la tabla usuarios, combinada con id_maquina.

También se ha añadido a posteriori, un trigger para comprobar que el id de la máquina esté entre 100 y 999. El objetivo de esta comprobación es que mantenga coherencia con la nomenclatura que utiliza proxmox y evitar errores futuros.

```
122 -- Relación entre Usuarios y Máquinas
123 • CREATE TABLE Usuarios_Maquinas (
124     DNI CHAR(9),
125     ID_Maquina INT,
126     PRIMARY KEY (DNI, ID_Maquina),
127     FOREIGN KEY (DNI) REFERENCES Usuarios(DNI),
128     FOREIGN KEY (ID_Maquina) REFERENCES Maquinas(ID_Maquina)
129 );
130
131 -- Trigger para comprobar que el id esté en el rango aceptado
132
133 DELIMITER //
134 • CREATE TRIGGER trg_check_id_maquina
135 BEFORE INSERT ON Maquinas
136 FOR EACH ROW
137 BEGIN
138     IF NEW.ID_Maquina < 100 OR NEW.ID_Maquina > 999 THEN
139         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ID_Maquina debe estar entre 100 y 999';
140     END IF;
141 END;
142 //
143 DELIMITER ;
```

4.2 Diseño de las funciones de gestión de proxmox

Para automatizar la gestión de proxmox, se ha decidido utilizar como base la librería proxmoxer de python, la cual nos permite hacer peticiones a la API de proxmox de manera sencilla. También decir que como editor de textos y terminal usaremos los que nos proporciona Visual Studio Code

En primer lugar, deberemos instalar la librería proxmoxer:

```
usuario@usuario:~/Documentos/ProyectosDjango/Proyecto1$ sudo pip install proxmoxer
[sudo] contraseña para usuario:
Collecting proxmoxer
  Downloading proxmoxer-2.0.1-py3-none-any.whl (16 kB)
Installing collected packages: proxmoxer
Successfully installed proxmoxer-2.0.1
```

Para poder utilizarla en nuestras funciones, deberemos de importar la librería y definir la conexión con nuestro servidor. Para esto último, deberemos de indicar la dirección ip de nuestro servidor, así como un usuario registrado en Proxmox con permisos necesarios para llevar a cabo las tareas definidas y la contraseña de este. Por último, Como nuestro servidor proxmox no tiene definido un protocolo seguro de transferencia de hipertexto (HTTPS), deberemos indicarle que no existe la verificación ssl necesaria para este.

```
1  from proxmoxer import ProxmoxAPI
2
3  proxmox = ProxmoxAPI('192.168.0.18', user='root@pam', password='usuario', verify_ssl=False)
4
```

Ahora procederemos a definir las funciones. Por comodidad y por una visualización más sencilla, en las funciones que requieren una larga lista de parámetros hemos definido una variable de tipo json, la cual será la que contenga la información que necesita recibir la API.

Funciones para la administración de usuarios:

```
10 #####
11 ## FUNCIÓN QUE CREA UN USUARIO PVE PROXMOX ###
12 ## #####
13 ## PARÁMETROS DE ENTRADA #####
14 ## nombre_usuario: nombre de usuario ###
15 ## correo_usuario: correo del usuario ###
16 ## grupo: Grupo del usuario ###
17 ## #####
18 #####
19
20 def creaUsuario(nombre_usuario, correo_usuario, grupo):
21     try:
22         # Define los parámetros del nuevo usuario
23         nuevo_usuario = {
24             'userid': nombre_usuario+'@pve',
25             'email': correo_usuario,
26             'comment': f'Usuario creado mediante script de Python',
27             'groups': grupo
28         }
29
30         #Crea el usuario enviando un json al endpoint de la API de proxmox
31         proxmox.access.users.create(**nuevo_usuario)
32
33         #En caso de haberla, recogemos la excepcion
34     except Exception as e:
35         print(f"Error al crear usuario '{nombre_usuario}': {str(e)}")
36
```

Como la API no nos permite añadir directamente una contraseña al usuario durante la creación, implementaremos una función que lo haga. Estas dos funciones siempre deben ir de la mano por motivos obvios de seguridad. Otra forma de hacerlo, más segura aunque hemos considerado innecesaria, es la de crear el usuario inactivo, añadirle contraseña y activarlo posteriormente.

Cabe resaltar, que se ha separado en dos funciones distintas porque, si se utilizan dentro de la misma, el servidor proxmox no se actualiza correctamente a tiempo y se produce un error al no encontrar al usuario entre los usuarios existentes.

```

41 #####
42 ## FUNCIÓN QUE AÑADE CONTRASEÑA AL USUARIO ###
43 ##                                     ###
44 ## PARÁMETROS DE ENTRADA             ###
45 ##     nombre_usuario: nombre de usuario     ###
46 ##     contra_aUsuario: contraseña del usuario ###
47 ##                                     ###
48 #####
49 def creaContra(nombre_usuario, contra_aUsuario):
50     try:
51         contra = {
52             "password": contra_aUsuario,
53             "userid": nombre_usuario+'@pve'
54         }
55     }
56
57     #Añadimos la contraseña al usuario
58     proxmox.access.password.put(**contra)
59
60     except Exception as e:
61         {
62             print(f"Error al crear usuario '{nombre_usuario}': {str(e)}")
63         }
64
65
66
67
68
69

```

```

73 #####
74 ## FUNCIÓN QUE ELIMINA UN USUARIO     ###
75 ##                                     ###
76 ## PARÁMETROS DE ENTRADA             ###
77 ##     nombre_usuario: nombre de usuario     ###
78 ##                                     ###
79 #####
80
81 def borraUsuario(nombre_usuario):
82
83     try:
84
85         proxmox.access.users(nombre_usuario+'@pve').delete()
86
87     except Exception as e:
88         {
89             print(f"Error al borrar usuario '{nombre_usuario}': {str(e)}")
90         }
91
92

```

Funciones para la gestión de las máquinas:

```
8 #####
9 ## FUNCIÓN QUE CREA UNA MAQUINA VIRTUAL    ###
10 ##                                         ###
11 ## PARÁMETROS DE ENTRADA                  ###
12 ##   idMaquina: id de la maquina          ###
13 ##   nodo: nodo de proxmox                 ###
14 ##   imagen: imagen a partir de la que se va ###
15 ##   a crear la maquina                    ###
16 ##   nombre: nombre de la maquina          ###
17 ##   memoria: ram de la maquina            ###
18 ##   sockets: sockets de la maquina        ###
19 ##   cores: nombre de la maquina           ###
20 ##   pool: pool a la que pertenece          ###
21 ##   storage: lugar de almacenamiento       ###
22 ##                                         ###
23 #####
24
25 def creaMaquina(idMaquina, nodo, imagen, nombre, memoria, sockets, cores, pool , storage ='local-lvm'):
26
27     nuevaMaquina ={
28         'vmid' : idMaquina,
29         'name':nombre,
30         'memory':memoria,
31         'sockets':sockets,
32         'cores':cores,
33         'storage':storage,
34         'pool': pool,
35         'cdrom':f'local:iso/{imagen}',
36         'net0':'virtio,bridge=vbr0'
37     }
38
39     try:
40         proxmox.nodes(nodo).qemu.create(**nuevaMaquina)
41     except Exception as e:
42         print(f"Error creando la VM: {e}")
43
44
45 #####
46 ## FUNCIÓN QUE CREA UNA MAQUINA VIRTUAL    ###
47 ## DESDE UNA PLANTILLA                      ###
48 ##                                         ###
49 ## PARÁMETROS DE ENTRADA                  ###
50 ##   idMaquina: id de la maquina          ###
51 ##   nodo: nodo de proxmox                 ###
52 ##   idPlantilla: plantilla a partir de la  ###
53 ##   que se va a crear la maquina          ###
54 ##   pool: pool a la que pertenece          ###
55 ##   storage: sockets de la maquina        ###
56 ##                                         ###
57 #####
58
59 def creaMaquinaPlantilla(idMaquina, nodo, idPlantilla, nombre, pool, storage ='local-lvm'):
60
61     try:
62         proxmox.nodes(nodo).qemu(idPlantilla).clone.newid(vmid=idMaquina).name(nombre).target(nodo).storage(storage).create()
63         proxmox.pools(pool).addmembers(vmid=idMaquina)
64     except Exception as e:
65         print(f"Error creando la VM: {e}")
66
67
68
69
70
71
```

```

75 #####
76 ## FUNCIÓN QUE CREA UN CONTENEDOR      ###
77 ##                                     ###
78 ## PARÁMETROS DE ENTRADA               ###
79 ##   idContenedor: id del contenedor   ###
80 ##   contra: contraseña del contenedor ###
81 ##   nodo: nodo de proxmox             ###
82 ##   imagen: nombre de la imagen       ###
83 ##   nombre: nombre de usuario del contenedor ###
84 ##   net0: configuración de red        ###
85 ##   cores: cores de la maquina        ###
86 ##   rootfs: Root filesystem especificando ###
87 ##   tamaño y almacenamiento,          ###
88 ##   ej. "local-lvm:4"                 ###
89 ##   storage: lugar de almacenamiento  ###
90 ##                                     ###
91 #####
92
93
94 def creaContenedor(idContenedor, contra, nodo, imagen, nombre, net0, cores, rootfs, storage='local-lvm'):
95
96     nuevoContenedor = {
97         'vmid':idContenedor,
98         'ostemplate': imagen,
99         'storage':storage,
100         'hostname':nombre,
101         'cores':cores,
102         'rootfs':rootfs,
103         'net0':net0,
104         'password':contra
105     }
106     try:
107         proxmox.nodes(nodo).lxc.create(**nuevoContenedor)
108     except Exception as e:print(f"Error creando el contenedor {e}")
109
110
111

```

```

113 #####
114 ## FUNCIÓN QUE CREA UN CONTENEDOR desde una  ###
115 ##   plantilla                             ###
116 ##                                     ###
117 ## PARÁMETROS DE ENTRADA               ###
118 ##   idContenedor: id del contenedor   ###
119 ##   contra: contraseña del contenedor ###
120 ##   nodo: nodo de proxmox             ###
121 ##   idPlantilla: nombre de la plantilla ###
122 ##   nombre: nombre de usuario del contenedor ###
123 ##   net0: configuración de red        ###
124 ##   cores: cores de la maquina        ###
125 ##   storage: lugar de almacenamiento  ###
126 ##                                     ###
127 #####
128
129 def creaContenedorPlantilla(idContenedor, contra, nodo, idPlantilla, nombre, net0, cores, storage='local-lvm'):
130
131     nuevoContenedor = {
132         'vmid':idContenedor,
133         'storage':storage,
134         'hostname':nombre,
135         'cores':cores,
136         'net0':net0,
137         'password':contra
138     }
139
140     try:
141         proxmox.nodes(nodo).lxc(idPlantilla).clone.create(**nuevoContenedor)
142     except Exception as e:print(f"Error creando el contenedor {e}")
143

```

Gestión de pools de los usuarios

```
8 #####
9 ## FUNCIÓN QUE CREA UNA POOL      ###
10 ##                                ###
11 ## PARÁMETROS DE ENTRADA          ###
12 ##   pool_id: nombre de la pool   ###
13 ##   comment: Comentario opcional ###
14 ##                                ###
15 #####
16
17 def crear_pool(poolid, comment=None):
18
19     try:
20         proxmox.pools.create(poolid=poolid, comment=comment)
21     except Exception as e:
22         print(f"Error crear la pool: {e}")
23
24
```

```
26 #####
27 ## FUNCIÓN QUE ASIGNA POOL A UN USUARIO  ###
28 ##                                        ###
29 ## PARÁMETROS DE ENTRADA                ###
30 ##   nombre_usuario: nombre de usuario  ###
31 ##                                        ###
32 #####
33
34 def asignaPool(pool, idUsuario):
35     try:
36         proxmox.pools(pool).update(users=f'{idUsuario+'@pve'}')
37     except Exception as e:
38         print(f"Error asignando el usuario al pool: {e}")
39
40
```



```
46 #####
47 ## FUNCIÓN QUE BORRA UNA POOL      ###
48 ##                                ###
49 ## PARÁMETROS DE ENTRADA           ###
50 ##   pool_id: nombre de la pool    ###
51 ##                                ###
52 #####
53 def borrar_pool(pool_id):
54
55     try:
56         proxmox.pools(pool_id).delete()
57     except Exception as e:
58         print(f"Error al borrar la pool: {e}")
59
```

Gestión de roles

```
8 #####
9 ## FUNCIÓN QUE ASIGNA ROL A UN USUARIO      ###
10 ##                                           ###
11 ## PARÁMETROS DE ENTRADA                    ###
12 ##   idUsuario: id de usuario               ###
13 ##   nombreRol: nombre del rol              ###
14 ##                                           ###
15 #####
16
17 def asignaRol(idUsuario, nombreRol):
18
19     nuevoUsuarioRoll = {
20         'path': '/',
21         'roleid': nombreRol,
22         'ugid': idUsuario+'@pve',
23         'type': 'user '
24     }
25
26     try:
27
28         proxmox.access.acl.set(**nuevoUsuarioRoll)
29
30     except Exception as e:
31         print(f"Error al asignar el rol a '{idUsuario}': {str(e)}")
32
```

4.3 Diseño de la aplicación

Como se ha indicado anteriormente, utilizaremos Django como framework para crear la aplicación web. Cabe mencionar que seguiremos el patrón de arquitectura software Modelo-Vista-Controlador, donde se diferenciarán el acceso a datos (Modelo), la interfaz de usuario (Vista) y el manejo de eventos (Controlador). Si bien tiene otro nombre, en este caso Model-Template-View (o MTV), es el recomendado y usado por defecto en Django.

Siguiendo esta lógica, dividiremos los archivos de la app en 3 carpetas: modelos, plantillas y funciones

4.3.1 Instalaciones previas

Necesitaremos tener instalado el siguiente software:

- Python3: se puede instalar mediante el comando `sudo apt install python3` en ubuntu o desde la página oficial de windows.
- Django: se instala con el comando `pip install django`. Esto instalará la última versión disponible

4.3.2 Creación del proyecto

Para crear el proyecto, primero crearemos la carpeta donde se almacenará. Una vez en esta carpeta, ejecutaremos el siguiente comando:

```
usuario@usuario:~/Documentos/ProyectosDjango$ django-admin startproject Proyecto1
```

Esto nos creará 4 archivos:

- `manage.py`: contiene los distintos comandos de django para manejar el proyecto, como creación de un superusuario, importar bases de datos, etc..
- `settings.py`: Fichero con la configuración del proyecto. Se verá en profundidad en el siguiente apartado
- `urls.py`: Fichero que contiene las urls que tendrá la página.
- `wsgi.py`: Fichero para la comunicación con el servidor virtual

4.3.3 Configuración del proyecto

Una vez creado, editaremos el archivo `settings.py` para configurar las distintas partes de nuestro proyecto. En primer lugar, cambiaremos el idioma al español. Se recomienda dejar comentadas las líneas que vienen por defecto y editar una copia de estas, por si en algún momento se quiere volver atrás en los cambios.

```
#LANGUAGE_CODE = 'en-us'  
LANGUAGE_CODE = 'es-es'
```

Ahora debemos configurar la base de datos. Django usa por defecto SQLite, por lo que debemos indicarle que trabajaremos con una base de datos MySQL, así como el nombre de la base de datos, las credenciales de conexión y la ubicación de la base de datos.

```
# Database  
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases  
  
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'tfg',  
        'USER': 'root',  
        'PASSWORD': 'usuario',  
        'HOST': 'localhost',  
        'PORT': '3306',  
    }  
}
```

Por último, hemos de indicarle las apps que tendrá el proyecto. En nuestro caso, debemos de añadir crispy forms, que usaremos para el formulario de login, y una app llamada gestión proxmox que crearemos más adelante. También utilizaremos bootstrap para darle forma al proyecto, por lo que debemos de indicarlo.

```
30
31 CRISPY_TEMPLATE_PACK = 'bootstrap4'
32
33 # Application definition
34
35 INSTALLED_APPS = [
36     'django.contrib.admin',
37     'django.contrib.auth',
38     'django.contrib.contenttypes',
39     'django.contrib.sessions',
40     'django.contrib.messages',
41     'django.contrib.staticfiles',
42     'gestionProxmox',
43     'crispy_bootstrap4',
44 ]
```

4.3.4 Creación de la app

Es recomendable separar nuestros proyectos en varias apps. Esto permite reciclar código en un futuro o, directamente, utilizar una app en varios proyectos. Para crear esta app, nos situamos en el directorio de nuestro proyecto y ejecutamos el siguiente comando:

```
o usuario@usuario:~/Documentos/ProyectosDjango$ python3 manage.py startapp gestionProxmox
```

De no estar situados en el directorio donde se encuentra manage.py, nos dará un error y no se creará la app.

Tras crear la app, deberemos definirla en un fichero apps.py. En este fichero es donde irán definidas las siguientes apps de nuestro proyecto, en este caso, solamente 1.

```
gestionProxmox > apps.py > ...
1  from django.apps import AppConfig
2
3
4  class GestionproxmoxConfig(AppConfig):
5      default_auto_field = 'django.db.models.BigAutoField'
6      name = 'gestionProxmox'
7
8
```

4.3.5 Migración de la base de datos.

Ya se ha mencionado que django utiliza el modelo MTV, por lo que necesitaremos importar los modelos de la base de datos. Para ello, en un archivo, llamado models.py por convenio, deberemos definir cada tabla existente en nuestra base de datos. Si bien se podría hacer a mano, django nos proporciona una manera automática de hacerlo.

Utilizando el comando `python3 manage.py inspectdb`, django buscará automáticamente la base de datos definida en el archivo `settings.py` y mostrará su estructura por pantalla. Si redirigimos esta salida a un archivo, el modelo se cargará automáticamente.

```
o usuario@usuario:~/Documentos/ProyectosDjango$ python3 manage.py inspectdb > models.py
```

```
gestionProxmox > models.py > Grupos
1 # This is an auto-generated Django model module.
2 # You'll have to do the following manually to clean this up:
3 # * Rearrange models' order
4 # * Make sure each model has one field with primary_key=True
5 # * Make sure each ForeignKey and OneToOneField has 'on_delete' set to the desired behavior
6 # * Remove 'managed = False' lines if you wish to allow Django to create, modify, and delete the table
7 # Feel free to rename the models, but don't rename db_table values or field names.
8 from django.db import models
9
10
11 class Grupos(models.Model):
12     id_grupo = models.CharField(db_column='ID_Grupo', primary_key=True, max_length=20) # Field name made lowercase.
13     descripcion = models.TextField(db_column='Descripcion', blank=True, null=True) # Field name made lowercase.
14     vlan = models.CharField(db_column='Vlan', max_length=10) # Field name made lowercase.
15
16     class Meta:
17         managed = False
18         db_table = 'Grupos'
19
20
21 class GruposRoles(models.Model):
22     id_grupo = models.OneToOneField(Grupos, models.DO_NOTHING, db_column='ID_Grupo', primary_key=True) # Field name made lowercase.
23     id_rol = models.ForeignKey('Roles', models.DO_NOTHING, db_column='ID_Rol') # Field name made lowercase.
24
25     class Meta:
26         managed = False
27         db_table = 'Grupos_Roles'
28         unique_together = (('id_grupo', 'id_rol'),)
29
30
31 class Imagenes(models.Model):
32     id_imagen = models.CharField(db_column='ID_Imagen', primary_key=True, max_length=20) # Field name made lowercase.
33     versionimagen = models.CharField(db_column='VersionImagen', max_length=20, blank=True, null=True) # Field name made lowercase.
34     descripcion = models.TextField(db_column='Descripcion', blank=True, null=True) # Field name made lowercase.
35
36     class Meta:
```

Esta estructura se repetirá por cada tabla de la base de datos

4.3.6 Creación de las vistas

Para las vistas, crearemos una carpeta, en nuestro caso la hemos llamado Plantillas. Esta carpeta contendrá los distintos archivos html de nuestro proyecto. También crearemos otra carpeta llamada static. Esta contendrá elementos como el css o las imágenes de nuestro proyecto.

Con el fin de simplificar el proyecto, ahorrar tiempo y mantener una coherencia entre páginas, crearemos una página base. El resto de páginas serán una extensión de esta (heredan de ella), manteniendo así un formato parecido .

Para ello, creamos una página html como haríamos normalmente. En este caso, tendrá un menú, una cabecera, un fondo y un pie de página. La única diferencia, es que deberemos indicar la parte que variará con un block content. También indicaremos que cargue los ficheros estáticos, como imágenes css etc, definidos en el fichero static creado anteriormente

```
1 <html>
2
3 <head>
4
5     {{ load static %}}
6
7     <!-- Bootstrap -->
8     <link href="{% static 'gestionProxmox/vendor/bootstrap/css/bootstrap.min.css' %}" rel="stylesheet">
9
10    <!-- Fonts -->
11
12    <link href="https://fonts.googleapis.com/css2?family=Raleway:wght@300&display=swap" rel="stylesheet">
13    <link href="https://fonts.googleapis.com/css2?family=Lora:400,400i,700,700i" rel="stylesheet">
14
15    <!-- Styles -->
16    <link href="{% static 'gestionProxmox/css/gestion.css' %}" rel="stylesheet">
17 </head>
18
19 <body>
20
21     <h1 class="site-heading text-center text-white d-none d-lg-block">
22         <span class="site-heading-lower">Gestión de Proxmox</span>
23     </h1>
24
25     <!-- Navbar -->
26     <nav class="navbar navbar-expand-lg navbar-dark py-lg-4" id="mainNav">
27         <div class="container">
28             <a class="navbar-brand text-uppercase text-expanded font-weight-bold d-lg-none" href="{% url 'home' %}">Gesti
29             <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-con
30             <span class="navbar-toggler-icon"></span>
31             </button>
32             <div class="collapse navbar-collapse" id="navbarResponsive">
33                 <ul class="navbar-nav mx-auto">
34                     <li class="nav-item {% if request.path == '/home/' %} active {% endif %} px-lg-4">
35                         <a class="nav-link text-uppercase text-expanded" href="{% url 'home' %}">Inicio</a>
36                     </li>
37                     <li class="nav-item {% if request.path == '/usuarios/' %} active {% endif %} px-lg-4">
38                         <a class="nav-link text-uppercase text-expanded" href="{% url 'usuarios' %}">Usuarios</a>
39                     </li>
40                 </ul>
41             </div>
42         </div>
43     </nav>
```

```

52 <!-- Contenido que cambia respecto a la plantilla-->
53
54 {% block content %}
55
56 {% endblock %}
57
58 <!-- Footer -->
59 <footer class="footer text-faded text-center py-5">
60   <div class="container">
61     <p class="m-0">
62       <a href="#" class="link">
63         <span class="fa-stack fa-lg">
64           <i class="fa fa-circle fa-stack-2x"></i>
65           <i class="fa fa-twitter fa-stack-1x fa-inverse"></i>
66         </span>
67       </a>
68       <a href="#" class="link">
69         <span class="fa-stack fa-lg">
70           <i class="fa fa-circle fa-stack-2x"></i>
71           <i class="fa fa-facebook fa-stack-1x fa-inverse"></i>
72         </span>
73       </a>
74       <a href="#" class="link">
75         <span class="fa-stack fa-lg">
76           <i class="fa fa-circle fa-stack-2x"></i>
77           <i class="fa fa-instagram fa-stack-1x fa-inverse"></i>
78         </span>
79       </a>
80     </p>
81     <p class="m-0 mbt1">&copy; Gestión de Proxmox 1.0</p>
82   </div>
83 </footer>
84
85 <!-- Bootstrap -->
86 <script src="{% static 'gestionProxmox/vendor/jquery/jquery.min.js' %}"></script>
87 <script src="{% static 'gestionProxmox/vendor/bootstrap/js/bootstrap.bundle.min.js' %}"></script>
88
89

```

Si creamos ahora un archivo home.html y le decimos que herede de base.html de la siguiente forma:

```

Proyecto1 > Plantillas > <> home.html > ...
1  {% extends "base.html" %}
2
3  {% load static %}
4
5  {% block content %}
6
7
8  {% endblock %}
9

```

Crearemos posteriormente un archivo views.py que contendrá nuestras vistas, así como la lógica de estas:


```
views.py 1
gestionProxmox > views.py > ...
1
2 from django.shortcuts import render
3
4
5
6 def home(request):
7
8
9     return render(request, "home.html")
10
11
```

Y por último, definimos el path en el archivo urls.

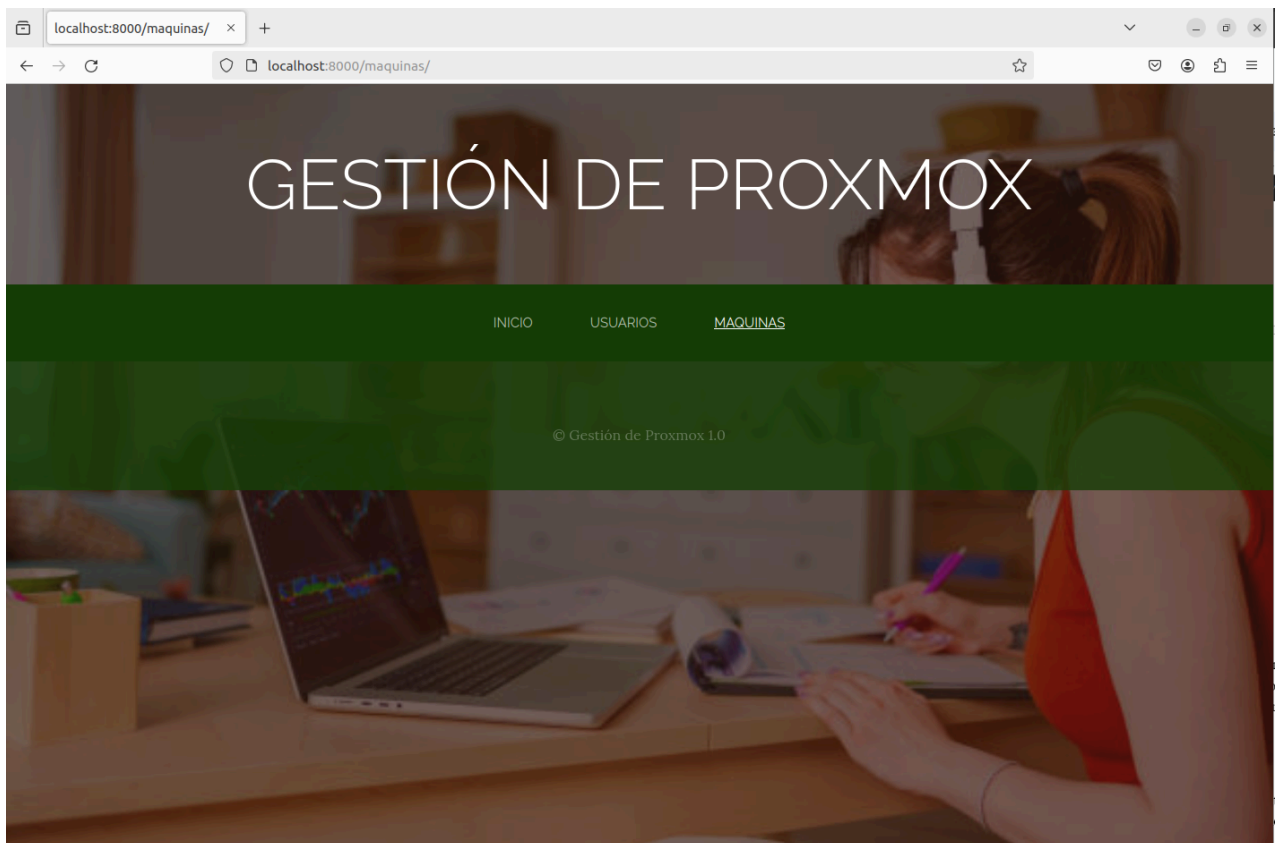
```
views.py  urls.py
Proyecto1 > urls.py > ...
1 """
2 URL configuration for Proyecto1 project.
3
4 The 'urlpatterns' list routes URLs to views. For more information please see:
5     https://docs.djangoproject.com/en/5.0/topics/http/urls/
6 Examples:
7 Function views
8     1. Add an import:  from my_app import views
9     2. Add a URL to urlpatterns:  path('', views.home, name='home')
10 Class-based views
11     1. Add an import:  from other_app.views import Home
12     2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
13 Including another URLconf
14     1. Import the include() function: from django.urls import include, path
15     2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
16 """
17
18 from django.urls import path
19 from gestionProxmox import views
20
21
22 urlpatterns = [
23
24     path('home/', views.home, name="home"),
25
26 ]
27
```

Podemos hacer correr el servidor virtual con el siguiente comando:

```
^X^Cusuario@usuario:~/Documentos/ProyectosDjango/Proyecto1$ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 11, 2024 - 09:37:34
Django version 5.0.6, using settings 'Proyecto1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Quedando de la siguiente forma:



Esta será la estructura general de la página web.

4.3.7 Creación de la página de login

La página de login será la única que no siga la estructura anteriormente mencionada. Será la página de inicio y el usuario no podrá ver otras páginas hasta que no se identifique correctamente. Constará de un formulario, donde se pide al usuario que introduzca su nombre y contraseña. Para el formulario, utilizaremos el formulario de autenticación por defecto de django, implementandolo de la siguiente manera

En el archivo views.py, insertamos el siguiente código para definir el formulario y la página a la que redirige si se completa con éxito el logueo

```
def logear(request):  
    form=AuthenticationForm()  
  
    return render(request, "login.html", {"form":form})
```

Creamos el archivo html login.html y añadimos el siguiente código para crear el formulario. Para ello, utilizamos un contenedor de fondo blanco, importamos crispy forms para utilizar el formulario por defecto de django y por último añadimos un botón de envío.

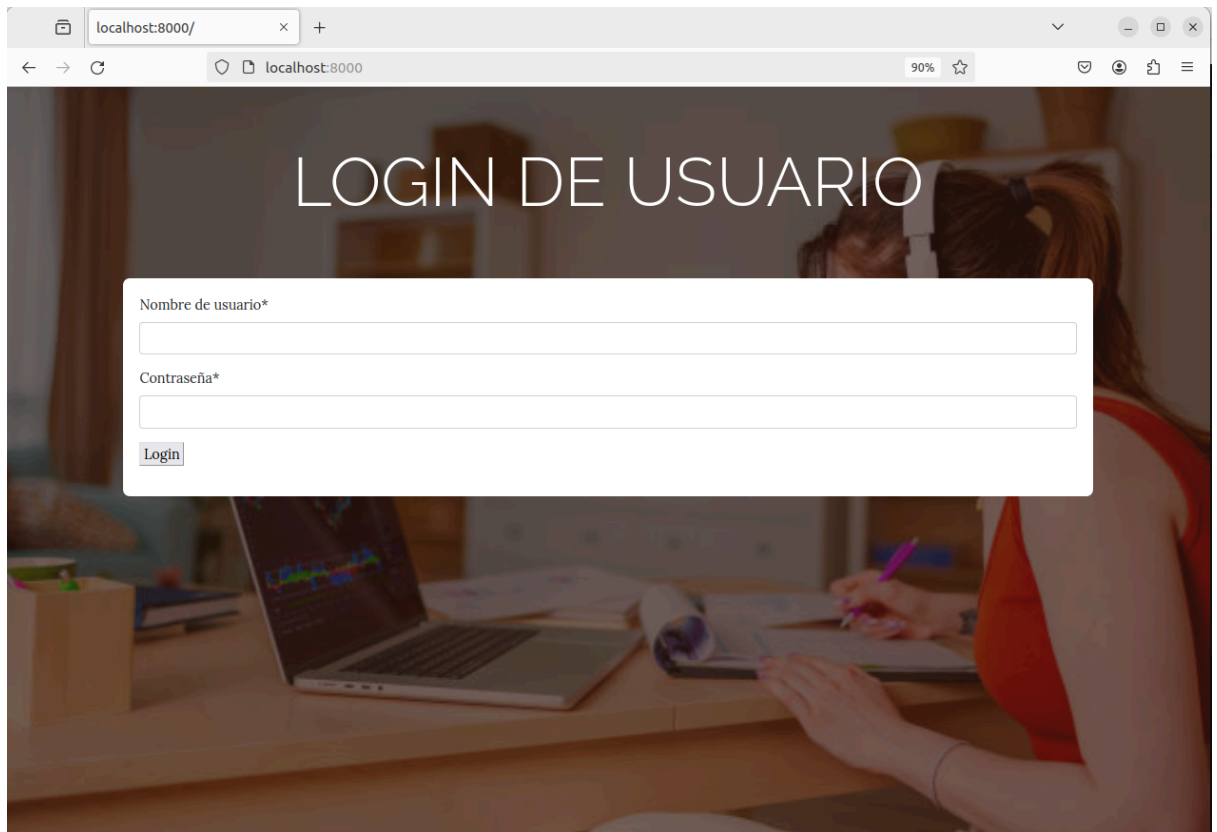
```
{% load crispy_forms_tags %}  
  
<div class="container" style="background-color: white; padding: 20px; border-radius: 8px;">  
  <form method="post">  
    {% csrf_token %}  
    {{ form|crispy }}  
    <button type="submit">Login</button>  
  </form>  
</div>
```

Para definir que sea la página por defecto, en el archivo urls.py añadimos la siguiente línea

```
path('', LoginView.as_view(template_name='login.html'), name='logear'),
```

Como podemos ver, el primer campo hace referencia a la url. Al estar vacía (‘ ‘) coge como url la dirección base del servidor. Con esto tambien enlazamos la vista (login.html) con el controlador (logear)

Por último, añadimos una imagen de fondo y aplicamos un poco de css para dar estilo a la página, quedando de la siguiente manera



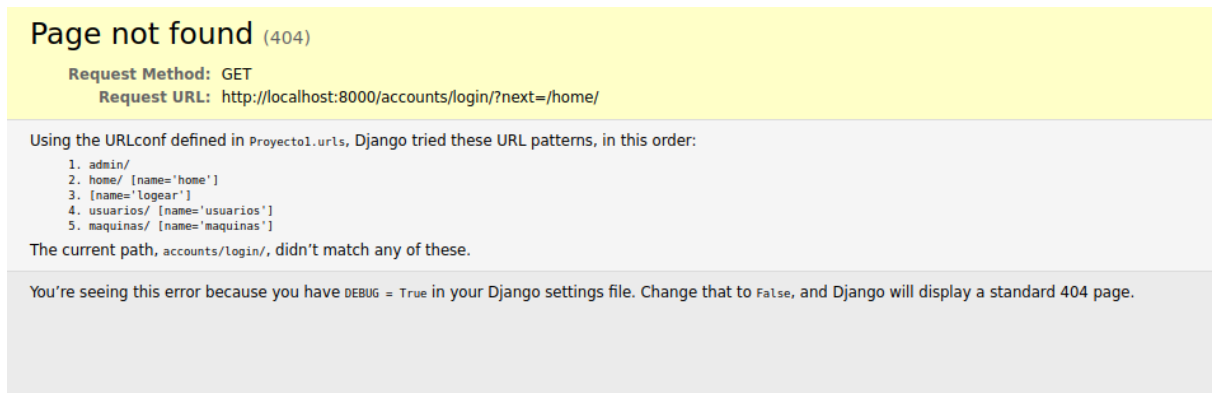
4.3.8 Página de inicio

Una vez identificado el usuario, deberá de ser redirigido a una página de bienvenida o home. Para ello, seguimos la metodología anterior

Definimos el controlador de la página

```
11
12 @login_required
13 def home(request):
14
15
16     return render(request, "home.html")
17
```

Con la línea login required indicamos que esta página no es accesible sin un logueo, por lo que dará un error al usuario si intenta acceder sin identificarse.



Esta ventana, una vez en producción y tal y como indica django, no será visible y en su lugar se mostrará un mensaje de error 404. De esta forma no se da información relevante a un posible atacante.

Para la vista, utilizaremos el siguiente código

```
Proyecto1 > Plantillas > home.html > ...
1 {% extends "base.html" %}
2
3 {% load static %}
4
5 {% block content %}
6
7     <!-- Heading -->
8     <section class="page-section clearfix">
9         <div class="container">
10             <div class="intro d-flex justify-content-center align-items-center">
11                 
12             </div>
13         </div>
14     </section>
15
16 {% endblock %}
```

Utilizamos la plantilla base, cargamos la carpeta static para utilizar una imagen. Centramos la imagen y quitamos los bordes.

Por último, definimos la url de la página en el archivo urls.py

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

Quedando de la siguiente manera



4.3.9 Página de usuarios

Esta página estará destinada a la creación de usuarios. Consistirá, básicamente, en dos apartados: uno para crear un usuario y otro para subir un archivo, con la información de los usuarios, procesarlo y crear tantos usuarios como se definan en el archivo. Estos usuarios se crearán tanto en la base de datos como en el servidor proxmox.

Comenzaremos por definir la url, como hemos hecho anteriormente, en el archivo urls.py.

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

Creamos la vista del archivo extendiendo la página base.

```
Proyecto1 > Plantillas > usuarios.html > div.container > form > div.form-group
1  {% extends "../base.html" %}
2
3  {% load static %}
4
5  {% block content %}
6
7  <div class="container" style="background-color: #white; padding: 20px; border-radius: 8px;">
8    <h1 class="mb-4">Suba un archivo de texto con los datos</h1>
9    <form method="post" enctype="multipart/form-data">
10
11      {% csrf_token %}
12
13      <div class="form-group">
14        <label for="file">Seleccionar archivo</label>
15        <input type="file" class="form-control-file" id="file" name="file">
16      </div>
17
18      <button type="submit" name="file_submit" class="btn btn-primary">Subir archivo</button>
19
20    </form>
21
22
```

Con la línea csrf token rescatamos el token que se crea automáticamente al logear. En la página anterior (home) no hacía falta ya que, al redirigir directamente desde login, esta redirección ya lo lleva implícito. Sin embargo, debemos de utilizarlo en otras páginas que requieran login.

Definimos un contenedor y un fondo de texto blanco junto con los bordes. Este contenedor se ajustará si cambiamos las dimensiones de la ventana del navegador. Por último, añadimos un pequeño título junto con dos botones, uno para subir el archivo y otro para aceptar y enviar el archivo. Es importante que el primer archivo pertenezca a la clase form-group y definir que será de tipo form control file.

Posteriormente, definiremos el formulario para que el usuario introduzca de forma manual los datos. para ello, creamos un pequeño título, junto con diferentes campos de texto de tipo form control, dando un nombre y un id a cada uno. junto con estos campos, definimos también como acceder a los datos que se almacenen en estos campos.

```

<h2 class="mt-4">0 introduzca los datos del Usuario manualmente</h2>
<form method="post">
  {% csrf token %}
  <div class="form-group">
    <label for="dni">DNI:</label>
    <input type="text" class="form-control" id="dni" name="dni" value="{{ user_data_form.dni.value }}">
  </div>
  <div class="form-group">
    <label for="nombre">Nombre:</label>
    <input type="text" class="form-control" id="nombre" name="nombre" value="{{ user_data_form.nombre.value }}">
  </div>
  <div class="form-group">
    <label for="apellidos">Apellidos:</label>
    <input type="text" class="form-control" id="apellidos" name="apellidos" value="{{ user_data_form.apellidos.value }}">
  </div>
  <div class="form-group">
    <label for="correo">Correo:</label>
    <input type="email" class="form-control" id="correo" name="correo" value="{{ user_data_form.correo.value }}">
  </div>
  <div class="form-group">
    <label for="password">Contraseña:</label>
    <input type="password" class="form-control" id="password" name="password" value="{{ user_data_form.password.value }}">
  </div>
  <button type="submit" name="manual_submit" class="btn btn-primary">Subir usuario</button>
</form>
</div>

{% endblock%}

```

Ahora debemos definir las formas, tanto de archivo que subamos, como del formulario. Esto con el formulario de login no hace falta, ya que utilizamos el login de usuario de django. Crearemos por tanto, el archivo forms.py. Dentro de este archivo, se definirá los campos de cada clase.

```

login.html  urls.py  forms.py  admin.py  usuarios.html  views.py  home.html  models.py
gestionProxmox > forms.py  UserDataForm
1  from django import forms
2
3
4  class UploadFileForm(forms.Form):
5      title = forms.CharField(max_length=50)
6      file = forms.FileField()
7
8
9  class UserDataForm(forms.Form):
10     dni = forms.CharField(max_length=10, label='DNI', widget=forms.TextInput(attrs={'class': 'form-control'}))
11     nombre = forms.CharField(max_length=100, label='Nombre', widget=forms.TextInput(attrs={'class': 'form-control'}))
12     apellidos = forms.CharField(max_length=100, label='Apellidos', widget=forms.TextInput(attrs={'class': 'form-control'}))
13     correo = forms.EmailField(label='Correo', widget=forms.EmailInput(attrs={'class': 'form-control'}))
14     password = forms.CharField(widget=forms.PasswordInput(attrs={'class': 'form-control'}), label='Contraseña')

```

El archivo de texto se compondrá de un título (el nombre del propio archivo) y del archivo en sí. Los usuarios tendrán dni, nombre, apellidos, correo y contraseña. es importante que las restricciones de la clase, coincidan con los de la base de datos en el caso de los usuarios, ya que si introducimos un usuario cuyos campos tengan una restricción en la base de datos, pero no esté reflejada en la clase, se producirá un error interno.

Ahora pasamos a definir cómo se tratarán estos datos en views.py. Importamos las funciones creadas anteriormente hemos almacenado en la carpeta funciones y, dentro de esta, las de gestión de usuarios que están en el archivo usuarios.py

```

from gestionProxmox.funciones import Usuarios

```


Para el caso del archivo, y a fin de modularizar el código, crearemos un manejador que se encargue de leer el archivo, guardar en la base de datos, y crear el usuario en proxmox. También es importante recordar que hay que crear la contraseña del usuario. Para ello, inmediatamente después de crearlo, llamamos a la función que se encarga de esto

```
28 def handle_uploaded_file(f):
29     # Leer el contenido del archivo
30     archivo = f.read().decode('utf-8').splitlines()
31     for linea in archivo:
32         # Crear un nuevo usuario y guardarlo en la base de datos
33         partes = linea.split()
34         if len(partes) == 6: # Asegurarse de que hay tres elementos en la fila
35             usuario = usuarios(dni=partes[0], nombre=partes[1], apellidos=partes[2], correo=partes[3], rol='alumno', contra = partes[4])
36             usuario.save()
37             GestionUsuarios.creaUsuario(partes[0], partes[3], partes[5])
38             GestionUsuarios.creaContra(partes[0], partes[4])
39
```

De esta forma, si el botón pulsado es el de subir archivo se llamará a este manejador. De otra forma, se tratará la información del formulario, quedando la función de la siguiente manera

```
@login_required
def usuarios(request):

    form = UploadFileForm()
    user_data_form = UserDataForm()
    if request.method == 'POST':

        if 'file_submit' in request.POST:
            form = UploadFileForm(request.POST, request.FILES)
            handle_uploaded_file(request.FILES['file'])

        elif 'manual_submit' in request.POST:
            user_data_form = UserDataForm(request.POST)
            datos = user_data_form.data
            dni = datos['dni']
            nombre = datos['nombre']
            apellidos = datos['apellidos']
            correo = datos['correo']
            contra = datos['password']
            grupo = datos['grupo']
            usuario = GestionUsuarios(dni=dni, nombre=nombre, apellidos=apellidos, correo=correo, contra=contra, rol='alumno')
            usuario.save()
            GestionUsuarios.creaUsuario([dni, correo, grupo])
            GestionUsuarios.creaContra(dni, contra)

        else:
            pass

    return render(request, 'usuarios.html', {'form': form, 'user_data_form': user_data_form})
```

La página quedará, por tanto, de la siguiente manera

GESTIÓN DE PROXMOX

INICIO USUARIOS MAQUINAS

Suba un archivo de texto con los datos

Select a file:
 No se ha seleccionado ningún archivo.

O introduzca los datos del Usuario manualmente

DNI:

Nombre:

Apellidos:

Correo:

Contraseña:

Para comprobar que funciona la página, verificamos los usuarios de la base de datos. Esto podemos hacerlo conectándonos a esta por cualquier método (terminal, workbench...) En este caso, lo haremos mediante phpmyadmin.

Si miramos la tabla usuarios, podemos ver que solo hay registrado un usuario.

phpMyAdmin

Reciente Favoritas

- mysql
- performance_schema
- phpmyadmin
- sys
- tfg
 - Nueva
 - auth_group
 - auth_group_permissions
 - auth_permission
 - auth_user
 - auth_user_groups
 - auth_user_user_permissions
 - django_admin_log
 - django_content_type
 - django_migrations
 - django_session
 - Grupos
 - Grupos_Roles
 - Imágenes
 - Maquinas
 - Maquinas_Imágenes
 - Maquinas_Plantillas
 - Permisos
 - Permisos_Usuarios
 - Plantillas
 - Roles
 - Roles_Permisos
 - Usuarios
 - Usuarios_Grupos
 - Usuarios_Maquinas

Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios Operaciones Seguimiento Disparadores

Mostrando filas 0 - 0 (total de 1. La consulta tardó 0.0022 segundos.)

SELECT * FROM `Usuarios`

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

☐ Mostrar todo Número de filas: 25 Filtrar filas: Buscar en esta tabla

	DNI	Nombre	Apellidos	Correo	Contra	Rol
<input type="checkbox"/>	123456788	Pepito	Martinez Gonzalez	gmail@gmail.com	123456789	alumno

☐ Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar Exportar

☐ Mostrar todo Número de filas: 25 Filtrar filas: Buscar en esta tabla

Operaciones sobre los resultados de la consulta

Imprimir Copiar al portapapeles Exportar Mostrar gráfico Crear vista

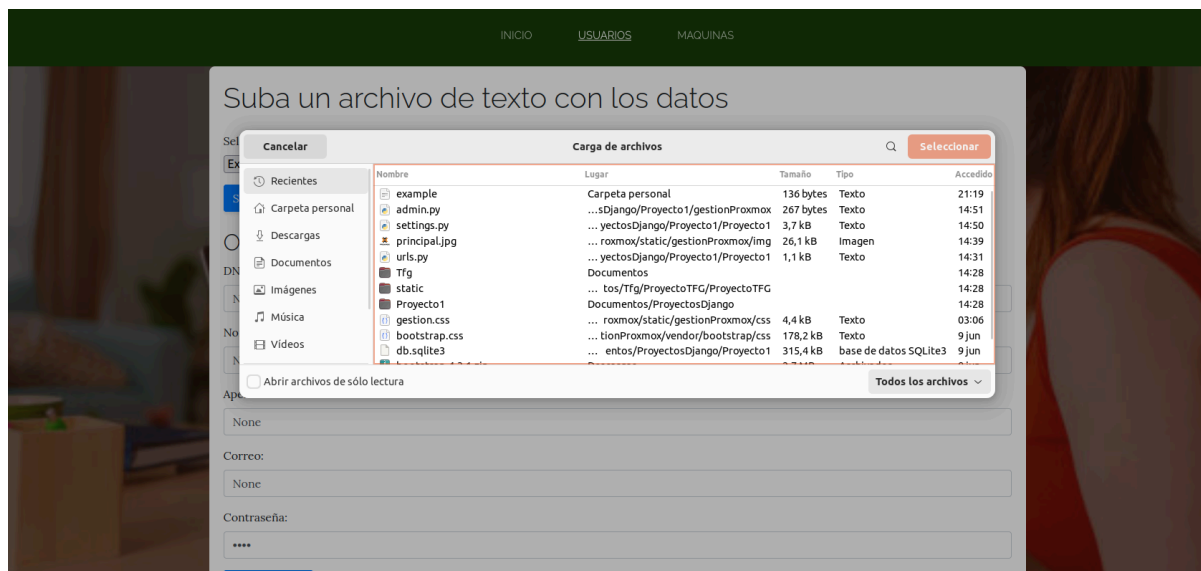
Guardar esta consulta en favoritos

Etiqueta: ☐ Permitir que todo usuario pueda acceder a este favorito

Guardar esta consulta en favoritos

Consola

Nos dirigimos ahora a la página y hacemos click en el botón de subir archivo



En este caso, subiremos el archivo examples, el cual contiene los datos de dos usuarios. Una vez seleccionado, nos aparecerá junto al botón de examinar. Hacemos click en subir archivo.



Si comprobamos la base de datos de nuevo, veremos que se han creado dos nuevos usuarios con los datos correspondientes.

+ Opciones												
<div>← T →</div>				DNI	Nombre	Apellidos	Correo	Contra	Rol			
<input type="checkbox"/>		Editar		Copiar		Borrar	12345678B	Pepito	Martinez Gonzalez	gmail@gmail.com	123456789	alumno
<input type="checkbox"/>		Editar		Copiar		Borrar	69673483F	Fulanito	VenturaMartinez	fulanventura@gmail.com	usuario	alumno
<input type="checkbox"/>		Editar		Copiar		Borrar	87654321D	Manolito	GutierrezFernandez	manolitoguti@gmail.com	usuario	alumno

De igual forma, comprobamos que se pueden crear usuarios manualmente

Introducimos los datos en el formulario

O introduzca los datos del Usuario manualmente

DNI:

29384756D

Nombre:

Juanito

Apellidos:

Lopez Valera

Correo:

jlopezcalera@gmail.com

Contraseña:

Subir usuario

Pulsamos en subir usuario y comprobamos la base de datos

✓ Mostrando filas 0 - 3 (total de 4, La consulta tardó 0.0006 segundos.)

SELECT * FROM `Usuarios`

☐ Perfilando [[Editar en línea](#)] [[Editar](#)] [[Explicar SQL](#)] [[Crear código PHP](#)] [[Actualizar](#)]

☐ Mostrar todo | Número de filas: 25 | Filtrar filas: Sort by key: Ninguna

+ Opciones

				DNI	Nombre	Apellidos	Correo	Contra	Rol
<input type="checkbox"/>				12345678B	Pepito	Martinez Gonzalez	gmail@gmail.com	123456789	alumno
<input type="checkbox"/>				29384756D	Juanito	Lopez Valera	jlopezcalera@gmail.com	usuario	alumno
<input type="checkbox"/>				69673483F	Fulanito	VenturaMartinez	fulanventura@gmail.com	usuario	alumno
<input type="checkbox"/>				87654321D	Manolito	GutierrezFernandez	manolitoguti@gmail.com	usuario	alumno

☐ Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar Exportar

☐ Mostrar todo | Número de filas: 25 | Filtrar filas: Sort by key: Ninguna

Como podemos comprobar, el usuario se ha registrado en la base de datos de forma correcta. Tanto en este método de creación de usuarios como en la creación mediante fichero, los usuarios son usuarios estándar de django, es decir, no tienen acceso al panel de administración de la página ni a la base de datos, ya que esto podría suponer un riesgo de seguridad. Para crear usuarios administradores de django o de la base de datos, habría que

crearlos en sus respectivos ámbitos, es decir, en el panel de administración para django o en mysql para la base de datos.

4.3.8 Página de máquinas virtuales

En primer lugar, crearemos el archivo html para la página. Este consistirá básicamente, al igual que los anteriores, en una extensión de la página base, que contendrá un formulario donde ingresar los datos de la máquina.

```
Proyecto1 > Plantillas > maquinas.html > div.container > form > div.form-group
1  {% extends "../base.html" %}
2
3  {% load static %}
4
5  {% block content %}
6
7  <div class="container" style="background-color: #white; padding: 20px; border-radius: 8px;">
8    <h2 class="mt-4">Introduzca los datos de la Máquina</h2>
9
10   <!-- Formulario para datos de la máquina -->
11   <form method="post">
12     {% csrf_token %}
13     <div class="form-group">
14       <label for="idMaquina">ID Máquina:</label>
15       <input type="number" class="form-control" id="idMaquina" name="idMaquina" value="{{ machine_data_form.idMaquina.value }}">
16     </div>
17     <div class="form-group">
18       <label for="nodo">Nodo:</label>
19       <input type="text" class="form-control" id="nodo" name="nodo" value="{{ machine_data_form.nodo.value }}">
20     </div>
21     <div class="form-group">
22       <label for="imagen">Imagen:</label>
23       {{ machine_data_form.id_imagen }}
24     </div>
25     <div class="form-group">
26       <label for="nombre">Nombre:</label>
27       <input type="text" class="form-control" id="nombre" name="nombre" value="{{ machine_data_form.nombre.value }}">
28     </div>
29     <div class="form-group">
30       <label for="memoria">Memoria:</label>
31       <input type="number" class="form-control" id="memoria" name="memoria" value="{{ machine_data_form.memoria.value }}">
32     </div>
33     <div class="form-group">
34       <label for="sockets">Sockets:</label>
35       <input type="number" class="form-control" id="sockets" name="sockets" value="{{ machine_data_form.sockets.value }}">
36     </div>
37     <div class="form-group">
38       <label for="cores">Cores:</label>
39       <input type="number" class="form-control" id="cores" name="cores" value="{{ machine_data_form.cores.value }}">
40     </div>
41     <div class="form-group">
42       <label for="net0">Direccion IP:</label>
43       <input type="text" class="form-control" id="net0" name="net0" value="{{ machine_data_form.net0.value }}">
44     </div>
45     <button type="submit" name="machine_submit" class="btn btn-primary">Subir máquina</button>
46   </form>
47 </div>
48
49 {% endblock %}
```

Para recoger los datos de este formulario, crearemos la clase correspondiente con los campos que devuelve el cuestionario, así como los tipos de cada dato. en este caso, tenemos varios campos de tipo texto, numérico y un menú desplegable con los datos almacenados en la tabla imagenes. Estas imágenes se han introducido a mano tanto en la base de datos como en el proxmox.

```
class Maquina(forms.Form):
    idMaquina = forms.CharField(max_length=100, widget=forms.TextInput(attrs={'class': 'form-control'}))
    nodo = forms.CharField(max_length=100, widget=forms.TextInput(attrs={'class': 'form-control'}))
    id_imagen = forms.ModelChoiceField(queryset=Imagenes.objects.all(), to_field_name="id_imagen", widget=forms.Select(attrs={'class': 'form-control'}))
    nombre = forms.CharField(max_length=100, widget=forms.TextInput(attrs={'class': 'form-control'}))
    memoria = forms.IntegerField(widget=forms.NumberInput(attrs={'class': 'form-control'}))
    sockets = forms.IntegerField(widget=forms.NumberInput(attrs={'class': 'form-control'}))
    cores = forms.IntegerField(widget=forms.NumberInput(attrs={'class': 'form-control'}))
```

Al igual que en la gestión de usuarios, tratamos los datos del formulario, creando un modelo con los campos que tiene nuestra tabla en la base de datos, almacenando este modelo, y llamando a la función correspondiente para crear la máquina.

```
@login_required
def maquinas(request):
    if request.method == 'POST':
        machine_data_form = Maquina(request.POST)
        if machine_data_form.is_valid():
            idMaquina = machine_data_form.cleaned_data['idMaquina']
            nodo = machine_data_form.cleaned_data['nodo']
            imagen = machine_data_form.cleaned_data['imagen']
            nombre = machine_data_form.cleaned_data['nombre']
            memoria = machine_data_form.cleaned_data['memoria']
            sockets = machine_data_form.cleaned_data['sockets']
            cores = machine_data_form.cleaned_data['cores']
            net0 = machine_data_form.cleaned_data['net0']

            usuario_instancia = Usuarios.objects.get(dni=request.user)

            maquina = Maquinas(id_maquina=idMaquina, nombre=nombre, creador=usuario_instancia, direccion_ip=net0)
            maquina.save()
            creaMaquina(idMaquina, nodo, imagen, nombre, memoria, sockets, cores)
        else:
            machine_data_form = Maquina()
```

Si ahora recargamos la página, debería quedar de la siguiente forma. Es posible que en algunos casos la página no se actualice correctamente al modificarla. Para evitar esto, es recomendable usar una ventana de incógnito del navegador para evitar las cookies y, tras cada modificación, parar el servidor y volverlo a levantar.

INICIO USUARIOS MAQUINAS CONTENEDORES

Introduzca los datos de la Máquina

ID Máquina:

Nodo:

Imagen:

Nombre:

Memoria:

Sockets:

Cores:

© Gestión de Proxmox 1.0

Como podemos ver, la tabla maquinas de la base de datos está vacía. Probaremos a rellenar el formulario y a pulsar el botón para comprobar que se ejecuta correctamente nuestro programa.


✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0003 segundos.)


SELECT * FROM `Maquinas`

☐ Perfilando [[Editar en línea](#)] [[Editar](#)] [[Explicar SQL](#)] [[Crear código PHP](#)] [[Actualizar](#)]

ID_Maquina	Nombre	Descripcion	Direccion_IP	Creador
------------	--------	-------------	--------------	---------

Operaciones sobre los resultados de la consulta

 Crear vista

 Guardar esta consulta en favoritos

Etiqueta: ☐ Permitir que todo usuario pueda acceder a este favorito

Introducimos unos datos de ejemplo que coincidan con las restricciones de la base de datos.

Como podemos ver, si introducimos un dato que no corresponde con lo que se espera en el modelo, la página nos dará un aviso. En este caso, espera un número para el id y, sin embargo, le estamos introduciendo letras, por lo que no funcionará.

ID Máquina:

Introduzca un número.

Ahora si, introducimos los datos de manera correcta.

INICIOUSUARIOSMAQUINASCONTENEDORES

Introduzca los datos de la Máquina

ID Máquina:

101

Nodo:

Nodo 1

Imagen:

Ubuntu

Nombre:

maquina1

Memoria:

2

Sockets:

2

Cores:

2

Direccion IP:

192.168.3.11

Subir máquina

© Gestión de Proxmox 1.0

Abrimos php phpmyadmin para comprobar los datos y podemos comprobar que ahora existe una nueva máquina con los datos anteriores

✓ Mostrando filas 0 - 1 (total de 2, La consulta tardó 0.0011 segundos.)

SELECT * FROM `Maquinas`

☐ Perfilando [[Editar en línea](#)] [[Editar](#)] [[Explicar SQL](#)] [[Crear código PHP](#)] [[Actualizar](#)]

☐ Mostrar todo | Número de filas: 25 | Filtrar filas: | Sort by key: Ninguna

+ Opciones

Editar

Copiar

Borrar

	ID_Maquina	Nombre	Descripcion	Direccion_IP	Creador
<input type="checkbox"/>	101	maquina1	NULL	192.168.3.11	12345678B

Seleccionar todo

Para los elementos que están marcados:

Editar

Copiar

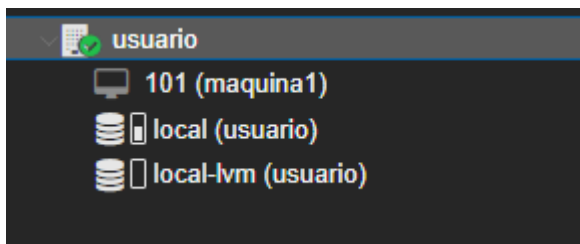
Borrar

Exportar

☐ Mostrar todo | Número de filas: 25 | Filtrar filas: | Sort by key: Ninguna

Operaciones sobre los resultados de la consulta

De igual forma, si comprobamos nuestro servidor proxmox podremos ver que se ha creado una máquina virtual con las características especificadas.



4.3.8 Página de contenedores

La creación de contenedores funcionará prácticamente igual que la de las máquinas. Hay que tener en cuenta, que un contenedor necesita que definamos también el nombre de usuario y una contraseña para acceder al contenedor, por lo que hay que indicarlo también en el formulario.

```
Proyecto1 > Plantillas > > contenedores.html > div.container > form > div.form-group
1  {% extends "../base.html" %}
2  {% load static %}
3
4  {% block content %}
5  <div class="container" style="background-color: #white; padding: 20px; border-radius: 8px;">
6    <h2 class="mt-4">Introduzca los datos del Contenedor</h2>
7
8    <!-- Formulario para datos del contenedor -->
9    <form method="post">
10      {% csrf token %}
11      <div class="form-group">
12        <label for="idContenedor">ID Contenedor:</label>
13        <input type="number" class="form-control" id="idContenedor" name="idContenedor" value="{{ container_data_form.idContenedor.value }}">
14      </div>
15      <div class="form-group">
16        <label for="nodo">Nodo:</label>
17        <input type="text" class="form-control" id="nodo" name="nodo" value="{{ container_data_form.nodo.value }}">
18      </div>
19      <div class="form-group">
20        <label for="imagen">Imagen:</label>
21        {{ container_data_form.imagen }}
22      </div>
23      <div class="form-group">
24        <label for="nombreUsuario">Nombre de usuario:</label>
25        <input type="text" class="form-control" id="nombreUsuario" name="nombreUsuario" value="{{ container_data_form.nombreUsuario.value }}">
26      </div>
27      <div class="form-group">
28        <label for="contra">Contraseña de la máquina:</label>
29        <input type="password" class="form-control" id="contra" name="contra" value="{{ container_data_form.contra.value }}">
30      </div>
31      <div class="form-group">
32        <label for="contra">Contraseña de la máquina:</label>
33        <input type="password" class="form-control" id="contra" name="contra" value="{{ container_data_form.contra.value }}">
34      </div>
35      <div class="form-group">
36        <label for="nombre">Nombre:</label>
37        <input type="text" class="form-control" id="nombre" name="nombre" value="{{ container_data_form.nombre.value }}">
38      </div>
39      <div class="form-group">
40        <label for="rootfs">Sistema de archivos:</label>
41        <input type="text" class="form-control" id="rootfs" name="rootfs" value="{{ container_data_form.rootfs.value }}">
42      </div>
43      <div class="form-group">
44        <label for="net0">Direccion IP:</label>
45        <input type="text" class="form-control" id="net0" name="net0" value="{{ container_data_form.net0.value }}">
46      </div>
47      <div class="form-group">
48        <label for="cores">Cores:</label>
49        <input type="number" class="form-control" id="cores" name="cores" value="{{ container_data_form.cores.value }}">
50      </div>
51      <button type="submit" name="container_submit" class="btn btn-primary">Subir contenedor</button>
52    </form>
53  </div>
54  {% endblock %}
```

De nuevo, definimos la clase que recogerá los datos del formulario:

```

class Contenedor(forms.Form):
    idContenedor = forms.CharField(max_length=100, widget=forms.TextInput(attrs={'class': 'form-control'}))
    contra = forms.CharField(max_length=100, widget=forms.PasswordInput(attrs={'class': 'form-control'}))
    nodo = forms.CharField(max_length=100, widget=forms.TextInput(attrs={'class': 'form-control'}))
    imagen = forms.ModelChoiceField(queryset=imagenes.objects.all(), to_field_name="id_imagen", widget=forms.Select(attrs={'class': 'form-control'}))
    nombre = forms.CharField(max_length=100, widget=forms.TextInput(attrs={'class': 'form-control'}))
    nombreUsuario = forms.CharField(max_length=100, widget=forms.TextInput(attrs={'class': 'form-control'}))
    net0 = forms.CharField(max_length=100, widget=forms.TextInput(attrs={'class': 'form-control'}))
    cores = forms.IntegerField(widget=forms.NumberInput(attrs={'class': 'form-control'}))
    rootfs = forms.CharField(max_length=100, widget=forms.TextInput(attrs={'class': 'form-control'}))

```

Y también la lógica que se utilizará en la página para procesar los datos y almacenarlos en la base de datos, creando el contenedor correspondiente:

```

gestionProxmox > views.py > contenedores
19
20 @login_required
21 def contenedores(request):
22
23
24     if request.method == 'POST':
25
26         container_data_form = Contenedor(request.POST)
27         if container_data_form.is_valid():
28             print('formulario válido')
29             idContenedor = container_data_form.cleaned_data['idContenedor']
30             contra = container_data_form.cleaned_data['contra']
31             nodo = container_data_form.cleaned_data['nodo']
32             imagen = container_data_form.cleaned_data['imagen']
33             nombre = container_data_form.cleaned_data['nombre']
34             net0 = container_data_form.cleaned_data['net0']
35             cores = container_data_form.cleaned_data['cores']
36             rootfs = container_data_form.cleaned_data['rootfs']
37             print('entro')
38
39             usuario_instancia = Usuarios.objects.get(dni=request.user)
40             maquina = Maquinas(id_maquina=idContenedor, nombre=nombre, creador=usuario_instancia, direccion_ip=net0)
41             maquina.save()
42             creaContenedor(idContenedor, contra, nodo, imagen, nombre, net0, cores, rootfs)
43
44         else:
45             container_data_form = Contenedor()
46         return render(request, 'contenedores.html', {'container_data_form': container_data_form})
47
48
49 @login_required

```

La página quedará de la siguiente forma:

INICIOUSUARIOSMAQUINASCONTENEDORES

Introduzca los datos del Contenedor

ID Contenedor:

Nodo:

Imagen:

Nombre de usuario:

Contraseña de la máquina:

Nombre:

Sistema de ficheros:

Dirección IP:

Cores:

Subir contenedor

Rellenamos el formulario para comprobar su funcionamiento, teniendo en cuenta como se ha dicho antes, que deben tener la forma correcta.

INICIO

USUARIOS

MAQUINAS

CONTENEDORES

Introduzca los datos del Contenedor

ID Contenedor:

102

Nodo:

Nodo 1

Imagen:

Ubuntu

Nombre de usuario:

usuario

Contraseña de la máquina:

Nombre:

contenedor1

Sistema de ficheros:

local-lvm:4

Dirección IP:

192.159.10.20

Cores:

2

Subir contenedor

Procedemos a enviar los datos y consultar la base de datos. Anteriormente hemos creado ya una máquina virtual y, cómo utilizamos la misma tabla para almacenar máquinas y contenedores, tendremos la máquina creada en el apartado anterior y la que acabamos de crear.

Examinar

Estructura

SQL

Buscar

Insertar

Exportar

Importar

Privilegios

Operaciones

Seguimiento

Disparadores

Mostrando filas 0 - 1 (total de 2, La consulta tardó 0.0005 segundos.)

SELECT * FROM 'Maquinas'

Perfilando

Editar en línea

Editar

Explicar SQL

Crear código PHP

Actualizar

Mostrar todo

Número de filas: 25

Filtrar filas: Buscar en esta tabla

Sort by key: Ninguna

+ Opciones

ID_Maquina

Nombre

Descripcion

Direccion_IP

Creador

101

maquina1

NULL

192.168.3.11

12345678B

102

contenedor1

NULL

192.159.10.20

12345678B

Seleccionar todo

Para los elementos que están marcados:

Editar

Copiar

Borrar

Exportar


Mostrar todo


Número de filas: 25



Filtrar filas: Buscar en esta tabla



Sort by key: Ninguna

Si comprobamos el proxmox, podemos ver que se ha creado correctamente el contenedor

 102 (contenedor1)

 101 (maquina1)

  local (usuario)

  local-lvm (usuario)

5. ANÁLISIS ECONÓMICO

Para llevar a cabo este proyecto, se ha optado por la creación de una empresa, cuya forma jurídica sea la de la sociedad limitada. Esto es debido a que necesitaremos un menor capital inicial, y en caso de quiebre de la empresa no deberemos responder con el capital de los socios. En adición, sólomente necesitaremos de 3 socios para crearla, lo cual casa perfectamente con los recursos humanos definidos.

En cuanto al presupuesto, teniendo en cuenta que el servidor físico será proporcionado por la empresa contratante y que todo el software que usamos es gratuito, podríamos dividirlo en los siguientes apartados:

Instalaciones: deberemos de tener un lugar físico donde atender a clientes, donde también puedan trabajar nuestros empleados. Para ello la mejor opción al empezar es alquilar una oficina, no muy grande y a ser posible céntrica, para atraer a una clientela cercana. En total, estaríamos hablando de un aproximado de 600 euros al mes.

Mobiliario: Dispondremos de 4 puestos de trabajo, por lo que necesitaremos de una mesa para cada uno, una silla y un par de estanterías para guardar documentos. Es importante pensar en la ergonomía de las sillas para que los empleados puedan desarrollar su trabajo cómodamente. También necesitaremos un par más de sillas para las posibles personas que acudan al local. En total, el presupuesto sería de 500 euros aproximadamente

Equipos informáticos: necesitaremos 4 ordenadores, ratones, monitores y teclados. Relativo al software, haremos uso de una licencia de office, mientras que el resto de programas serán de software libre. Con todo, el presupuesto aproximado será de 2.000 euros.

Elementos de transporte: En principio, los trabajadores usarán el vehículo propio. Esto implica que la empresa deberá pagar el combustible de los vehículos, pero no requiere de una inversión inicial.

Presupuesto de inicio: El presupuesto inicial, será la suma de los materiales más los 3.000 euros de depósito inicial que requiere fundar una sociedad limitada. En total, y sumando algunos elementos decorativos para la oficina, el presupuesto inicial será de 6.200 euros.

Presupuesto anual: Este presupuesto dependerá del número de desplazamientos realizados, la electricidad consumida, etc ... por lo que solamente podemos hacer un aproximado. En total, reservaremos unos 30.000 euros.

Plan de financiación: Aparte del capital aportado por los socios, haremos uso de las siguientes ayudas y subvenciones:

- Programa de Promoción del Empleo Autónomo (PAE)
- Ayudas para la digitalización
- El Instituto de Crédito Oficial (ICO)

6. SEGUIMIENTO Y CONTROL

Al proporcionar un servicio, se hace necesario un seguimiento y control del sistema. Principalmente, el usuario final del servicio, es decir, el cliente, puede necesitar nuevas funcionalidades en el sistema. Estas actualizaciones se llevarán a cabo cuando el cliente las solicite.

Por otro lado, necesitamos controlar que el funcionamiento del sistema sea el correcto, monitorizando cada una de sus partes.

En primer lugar, para la monitorización podemos utilizar Netdata, un software libre y gratuito destinado a la monitorización de sistemas. Este software nos permite monitorizar el servidor proxmox, viendo el uso de recursos consumidos o distintos logs de errores. Esto nos permite adaptar el servidor de ser necesario. Por ejemplo, si la cantidad de usuarios es mayor a la calculada inicialmente, podría ser necesario ampliar la memoria RAM o el almacenamiento.

El mismo software, nos permite monitorizar aplicaciones del sistema, esto incluye la base de datos. Podemos monitorizar MySQL y ver si se produce algún error, el tiempo de las consultas, etc... todo esto, con la finalidad de optimizar la base de datos y corregir posibles fallas.

En cuanto a la frecuencia, se debería de hacer un chequeo semanal, o al menos mensual del sistema en general. Esto sumado, claro, a errores producidos, los cuales serán notificados por la aplicación de monitoreo o, en el peor de los casos, por el cliente al notar algún fallo.

Las acciones correctivas pueden ir desde añadir elementos a la base de datos, como reprogramar funciones las cuales tengan alguna falla no detectada durante el periodo de pruebas, ampliación de software o rediseño de las vistas de la aplicación web

6.1 EVALUACIÓN GENERAL

En este apartado, haremos un análisis DAFO sobre nuestro proyecto.

Debilidades: Dentro de las debilidades del proyecto, se encuentra la poca experiencia en el sector. La experiencia a la hora de resolver problemas es muy importante y no tenerla puede ser un factor que juegue en contra. Otra de las debilidades sería el no tener personal

especializado en el marketing, ya que al ser la virtualización una práctica relativamente joven, hay que localizar y convencer a empresas del sector para implementar un servidor Proxmox.

Fortalezas: En cuanto a las fortalezas, el servicio ofrecido es muy personalizable, adaptándose a lo que necesita el cliente. Por otro lado, no hay muchas empresas que ofrezcan este servicio.

Oportunidades: En primer lugar, el auge de la virtualización nos ofrece una gran oportunidad de mercado. Muchas empresas se están virtualizando e incluso desde la pandemia, el trabajo en remoto está en auge, lo que favorece aún más la creación de un servidor sobre el que trabajar telemáticamente. Otra de las oportunidades es el crecimiento de las empresas tecnológicas en Córdoba. Las empresas pequeñas o medianas no suelen tener un departamento específico para la administración de sistemas, por lo que podrían recurrir a servicios externos para ello.

Amenazas: Una de las principales amenazas son los servicios ofrecidos por grandes empresas, como Amazon. Muchas empresas recurren a Amazon para la contratación de servicios virtuales. Al ser un proyecto nuevo, no podemos competir contra el marketing ni los servicios ofrecidos por empresas de tal envergadura.

7. POSIBLES MEJORAS

En este apartado indicaremos posibles mejoras para el proyecto que, por distintas razones, no se han podido implementar durante la elaboración del mismo

- Ampliación de la base de datos: es posible que la base de datos actual, no cubra todas las necesidades o que se incorporen algunas nuevas. Por esto, una posible mejora es ampliar la base de datos respecto a los requerimientos que se vayan descubriendo
- Mejora visual de la página web: la página web es una página muy sencilla, la cual está diseñada para ser funcional. Se pueden implementar varias mejoras visuales o incluso algunas que los usuarios consideren necesarias a la hora de mostrar información.
- Mejora en la seguridad: Si bien la página tiene medidas de seguridad informática implementadas, como la protección contra inyección sql, hay muchas otras que podrían implementarse, como establecer requisitos mínimos para las contraseñas o instalar un proxy para proteger de ataques externos.
- Aumentar funcionalidades de la aplicación: Se pueden añadir funcionalidades utilizando las funciones que ya tenemos, como borrar usuarios, máquinas o contenedores, clonar máquinas, crear pools y asignarlas, etc.

8. FUENTES DE DOCUMENTACIÓN

<https://genos.es/proxmox-ve/>

<https://www.hostdime.com.pe/blog/mejor-software-de-virtualizacion-de-servidores/>

<https://pve.proxmox.com/pve-docs/api-viewer/index.html>

<https://www.redhat.com/es/topics/virtualization/what-is-virtualization>

<https://www.godaddy.com/resources/latam/desarrollo/prototipo-interfaz-usuario-definicion-herramientas>

<https://github.com/iesmedina/scriptsproxmox/tree/main>

<https://www.youtube.com/playlist?list=PLU8oAlHdN5BmfvwxF07HdPciOCmmYneA>

B

<https://getbootstrap.com/>

<https://www.djangoproject.com/>

<https://stackoverflow.com/>

https://developer.mozilla.org/es/docs/Learn/Server-side/Django/skeleton_website

<https://medium.com/@a01207543/django-conecta-tu-proyecto-con-la-base-de-datos-mysql-2d329c73192a>

<https://www.canva.com/>

<https://www.python.org/doc/>

9. Anexos

Formularios:

<https://getbootstrap.com/docs/4.0/components/forms/>

Botones:

<https://getbootstrap.com/docs/4.0/components/buttons/>

Barras de navegación:

<https://getbootstrap.com/docs/4.0/components/navbar/>

Documentación de proxmoxer:

<https://proxmoxer.github.io/docs/2.0>