

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH



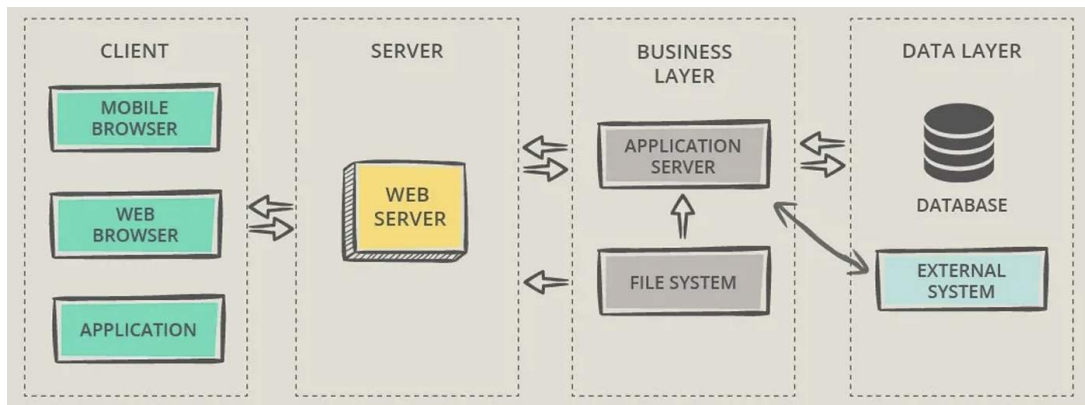
Desarrollo de aplicaciones para Internet

@ Email	jmguirao@ugr.es
# Teoría	25%
# Prácticas	75%
# Curso	5

Exámenes de otros años

Examen - febrero 2016

▼ Bases de Datos



Archivos indexados

Los archivos indexados son estructuras de datos que permiten acceder eficientemente a registros o datos almacenados, utilizando **índices** para facilitar la búsqueda y recuperación de información.

Permiten una **búsqueda eficiente** y un **acceso rápido** pero requieren de **espacio adicional** y **mantenimiento**.

Son utilizados con la misma interfaz que los diccionarios en memoria. Se pueden implementar mediante la librería `dbm`.

```
import dbm

db = dbm.open('file.dbm', 'c')
db['one'] = 'un'
db['two'] = 'dos'
db['three'] = 'tres'

print(db['two'])
print(db.get('two'))




























db.close()
```

Serialización y deserialización

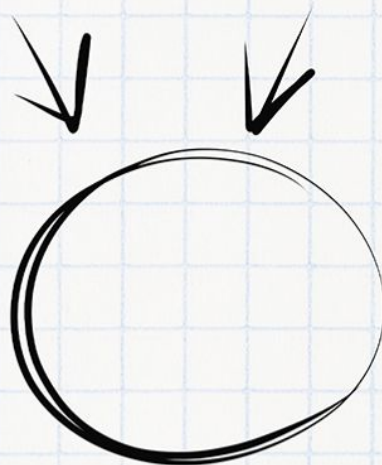
Es posible añadir serialización/deserialización para guardar cualquier tipo de dato utilizando la librería `shelve`. Este enfoque permite almacenar estructuras de datos más complejas.

Imagínate aprobando el examen

Necesitas tiempo y concentración

Planes	 PLAN TURBO	 PLAN PRO	 PLAN PRO+
 Descargas sin publi al mes	10 	40 	80 
 Elimina el video entre descargas			
 Descarga carpetas			
 Descarga archivos grandes			
 Visualiza apuntes online sin publi			
 Elimina toda la publi web			
 Precios Anual <input type="checkbox"/>	0,99 € / mes	3,99 € / mes	7,99 € / mes

Ahora que puedes conseguirlo,
¿Qué nota vas a sacar?



WUOLAH

```
import shelve

db = shelve.open('file.dbm', 'c')
db['one'] = ['Una', 'lista']
db['two'] = 'un string'
db['three'] = {'un': 'diccionario'}

print(db['two'])
print(db.get('two'))
print(db.keys())
print(db.values())

db.close()
```

Bases de Datos Relacionales (SQL):

Las bases de datos relacionales son gestionadas con Object Relational Mappers (ORM), como Django models.



Un **ORM** (Object Relational Mapper) es una herramienta de software utilizado para **traducir las representaciones de datos** usadas por las bases de datos (como MongoDB o SQL) y aquellas usadas por la programación orientada a objetos.

Actúa como una **capa de abstracción**, de forma que es irrelevante la base de datos subyacente que está siendo utilizada en un proyecto.

Estos permiten manejar tablas con clases y métodos, sin la necesidad de escribir SQL directamente, lo que facilita la manipulación de bases de datos relacionales sin la necesidad de SQL directo. Ejemplos en Django serían:

```
import Producto

producto = Producto(nombre="camisa de hombre", precio=16.15)
producto.save()
```


Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



```
un_producto = Producto.objects.get(pk=1)
todos_los_productos = Producto.objects.all().order_by('precio')
productos_caros = Producto.objects.filter(precio__gt=1000.)
ropa_de_hombre = Producto.objects.filter(categoria="Ropa de hombre")
```

Bases de Datos NoSQL:

Las bases de datos NoSQL surgieron para manejar Big Data, abordando las '3 Vs': **Volumen**, **Velocidad** y **Variedad**.

Tipos de BD NoSQL:

- **Key-Value Stores:**

Sirven como caches temporales en aplicaciones asíncronas o cuando los datos se tratan a distintas velocidades.

Ejemplos: Redis y Memcache.

- **Motores de Búsqueda:**

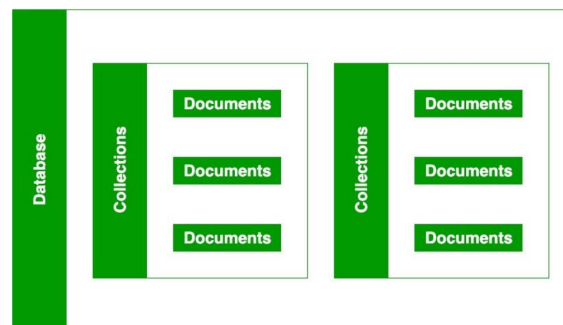
Motores de búsqueda que ofrecen soluciones para búsquedas en texto, páginas web y logs centralizados.

Ejemplos: Elasticsearch

- **Basadas en Documentos:**

MongoDB es una base de datos orientada a documentos, formada por colecciones, que son un conjunto de documentos.

Ejemplos: MongoDB



MongoDB y CRUD

En MongoDB, el esquema CRUD representa las operaciones básicas que se pueden realizar sobre los datos en una base de datos. CRUD es un acrónimo que representa:

- **Create (Crear)** → `insertOne()` o `insertMany()`
- **Read (Leer)** → `find()` o `findOne()`
- **Update (Actualizar)** → `updateOne()` o `updateMany()`
- **Delete (Eliminar)** → `deleteOne()` o `deleteMany()`

Esquema de las Bases de Datos NoSQL:

A diferencia de las bases de datos relacionales, las NoSQL tienen un esquema flexible. La librería `Pydantic` es una opción para esquematizar y validar datos en este contexto.

SQL vs NoSQL

	SQL	NoSQL
Estructura	Predefinida	Flexible
Esquema	Fijo (definido antes de ingresar datos)	Dinámico (no requiere ser predefinido, se pueden añadir nuevos campos sin cambiar toda la BD)
Consulta	Usan SQL	Personalizado (Cada BD NoSQL tiene el suyo propio)
Escalabilidad	Vertical (añadir más recursos al servidor)	Horizontal (se pueden distribuir en más servidores)

▼ Preguntas de examen:

▼ ¿Cuál es la diferencia entre bases de datos relacionales y NoSQL?

- **Relacionales (SQL):** Utilizan tablas con esquemas fijos, SQL para consultas, aseguran integridad y ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad).

- NoSQL: Flexibles en estructura, pueden usar varios formatos, escalables horizontalmente, diversidad en modelos y lenguajes de consulta.

▼ ¿Cómo es la gestión de archivos indexados en Python con `dbm` ?

La librería `dbm` proporciona un acceso simple a archivos indexados, permitiendo almacenar y recuperar datos utilizando claves.

Ejemplo:

```
import dbm

# Abrir un archivo indexado (se crea si no existe)
with dbm.open('archivo.db', 'c') as db:
    # Agregar datos al archivo indexado
    db[b'clave'] = b'valor'
# Recuperar valores del archivo indexado
print(db[b'clave']) # Imprime: b'valor'
print(db.get(b'otra_clave')) # Imprime: b'otro_valor'

# Eliminar una clave del archivo indexado
del db[b'clave']
```

▼ ¿Qué es un ORM? En Django, ¿cómo se llama el archivo relacionado con el ORM?

Un **ORM** (Mapeador Objeto-Relacional) es una herramienta de programación que permite interactuar con bases de datos utilizando objetos en lugar de consultas SQL directas.

En Django, el archivo relacionado con el ORM se llama `models.py`, donde se definen los modelos de datos que se mapearán a las tablas de la base de datos.

▼ Protocolo HTTP

Cliente vs Servidor

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali oohh
esto con 1 coin me
lo quito yo...

WUOLAH

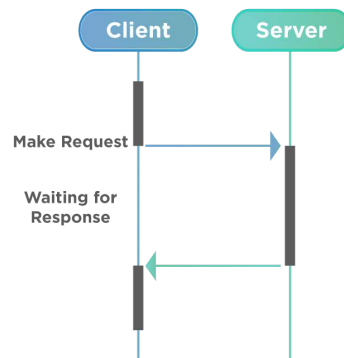
Las aplicaciones web se distribuyen en dos componentes fundamentales: el Cliente y el Servidor. Esta arquitectura implica que el Cliente se ocupa de la interfaz de usuario, mientras que el Servidor maneja la lógica de la aplicación y la comunicación con la Base de Datos.

	Cliente	Servidor
Tipo	Proactivo (inicia la comunicación)	Reactivo (siempre a la escucha)
Manda	Requests	Responses
Comunicación con	El usuario	La Base de Datos

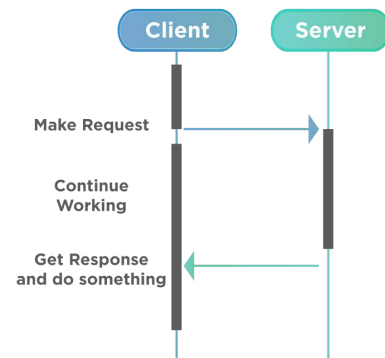
Comunicación síncrona vs asíncrona

La comunicación puede ser síncrona o asíncrona, siendo AJAX y fetch ejemplos de llamadas asíncronas.

Synchronous



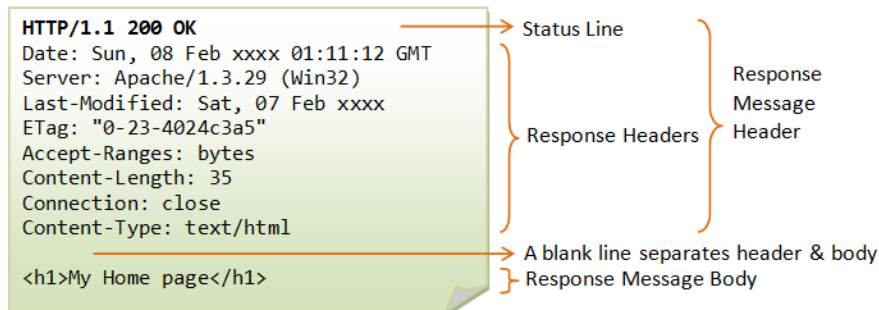
Asynchronous



Catacterísticas de HTTP

- Es **state-less**: Cada solicitud es independiente de las anteriores.
- Usa **ASCII** para requests y responses.
- Usa **verbos HTTP** que están asociados a operaciones **CRUD** cuando se usan en una API.
- Tiene **sesiones** mediante el uso de las **cookies**.

- Es **full-duplex**: la comunicación es **bidireccional** (permite la transferencia simultánea de datos en ambas direcciones entre el Cliente y el Servidor). Para ello, usa **web-socket** (que establece una conexión persistente).



Ejemplo de un Response

GET

Los parámetros en el GET se incluyen en la URL.

```
https://ejemplo.com/recurso?parametro1=valor1&parametro2=va
```

POST

Los parámetros del POST, se encuentran en el body, comúnmente asociado a formularios.

```
POST /recurso HTTP/1.1
Host: ejemplo.com

parametro1=valor1&parametro2=valor2
```

WSGI

El protocolo HTTP evolucionó desde versiones como HTTP 0.9 hasta HTTP 2. Se introduce el concepto de **WSGI** (Web Server Gateway Interface) para adaptar HTTP a Python, permitiendo la comunicación entre servidores web y aplicaciones Python de manera eficiente.

▼ Preguntas para el Examen:

▼ **Explique la diferencia entre el Cliente y el Servidor en una arquitectura web y cómo se comunican mediante HTTP.**

El Cliente es proactivo e inicia la comunicación, mientras que el Servidor es reactivo y responde a los requerimientos del Cliente mediante requests y responses en el protocolo HTTP.

▼ **¿Cómo se manejan los diferentes tamaños de pantalla en el diseño web y por qué se busca que sea "responsive"?**

Se busca un diseño "responsive" para adaptarse eficientemente a distintos tamaños de pantalla en los clientes, asegurando una experiencia de usuario consistente y efectiva.

▼ **Explique la diferencia entre comunicación síncrona y asíncrona en el desarrollo web. Dé ejemplos de tecnologías que permitan llamadas asíncronas.**

La comunicación síncrona espera a que se complete una tarea antes de continuar, mientras que la asíncrona no espera y permite que otras tareas continúen. Ejemplos de tecnologías asíncronas son AJAX y fetch.

▼ **Describa el proceso de solicitud (request) en HTTP, incluyendo los verbos del HTTP y cómo se manejan los parámetros en el GET y POST. Proporcione un ejemplo de solicitud POST.**

El request incluye un header y puede tener verbos HTTP asociados a operaciones CRUD. Los parámetros en el GET están en la URL, y en el POST, en el body. Ejemplo POST: `POST /recepcion_formulario HTTP/1.1 ... variable_1=valor_1&variable_2=valor_2 .`

▼ **¿Qué significa que HTTP sea un protocolo sin estado? Explique cómo se gestionan las sesiones a pesar de ello, incluyendo el uso de cookies. ¿Qué son las third party cookies?**

HTTP sin estado (stateless) significa que cada solicitud es independiente. Las sesiones se gestionan con cookies, permitiendo persistencia. Third party cookies son cookies utilizadas por dominios diferentes al que el usuario está visitando.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

▼ Explique brevemente la evolución del protocolo HTTP desde sus primeras versiones hasta HTTP 2. ¿Qué mejoras trajo HTTP 2?

El protocolo evolucionó desde HTTP 0.9 a HTTP 2. HTTP 2 introdujo mejoras en la velocidad, eficiencia y seguridad mediante técnicas como la multiplexación y la compresión de encabezados.

▼ ¿Qué es WSGI y cómo adapta HTTP a Python en el desarrollo web?

WSGI (Web Server Gateway Interface) es una interfaz que facilita la comunicación entre servidores web y aplicaciones Python, permitiendo una integración eficiente y flexible.

▼ Explique la capacidad full duplex de WebSocket y su utilidad en el desarrollo web.

WebSocket permite una comunicación bidireccional full duplex, facilitando la transferencia de datos en ambas direcciones de forma simultánea, lo que es útil para aplicaciones que requieren una interacción en tiempo real, como chats y juegos en línea.

▼ ¿Cuál es la diferencia en el envío de parámetros entre una llamada GET y otra POST de HTTP?

La diferencia entre GET y POST en HTTP radica principalmente en cómo se envían los datos:

- **GET** envía los datos a través de la URL. Es visible para todos y tiene restricciones en la cantidad de datos que se pueden enviar.

```
https://ejemplo.com/recurso?parametro1=valor1&parametro2=valor2
```

- **POST** envía los datos en el cuerpo de la solicitud HTTP, es más seguro y no tiene limitaciones en el tamaño de los datos.

```
POST /recurso HTTP/1.1
```

```
Host: ejemplo.com
```

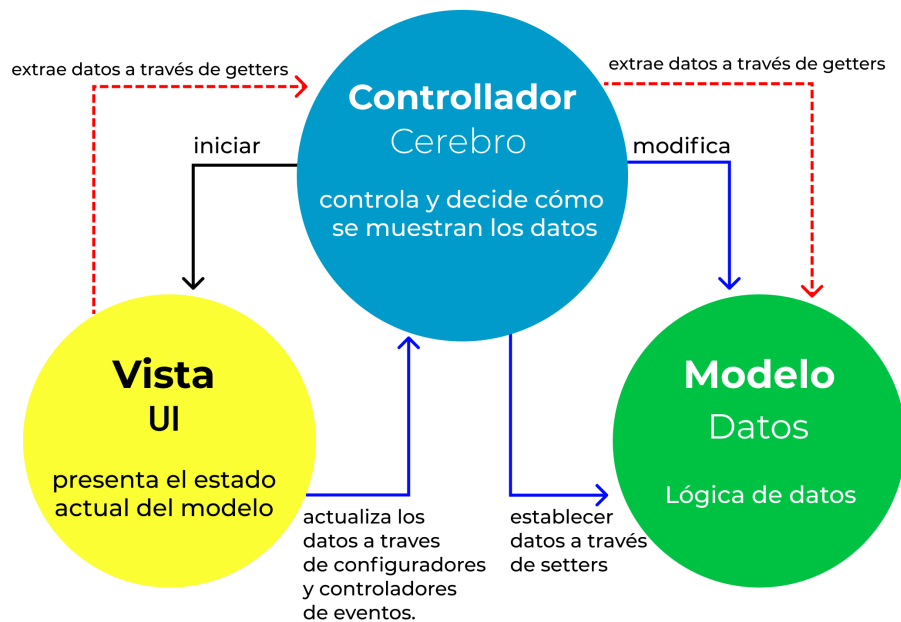
```
parametro1=valor1&parametro2=valor2
```

▼ Modelo Vista Controlador (MVC)



MVC es un patrón de diseño arquitectónico que divide una aplicación en tres componentes principales: **Modelo** (Model), **Vista** (View) y **Controlador** (Controller).

Patrones de Arquitectura MVC



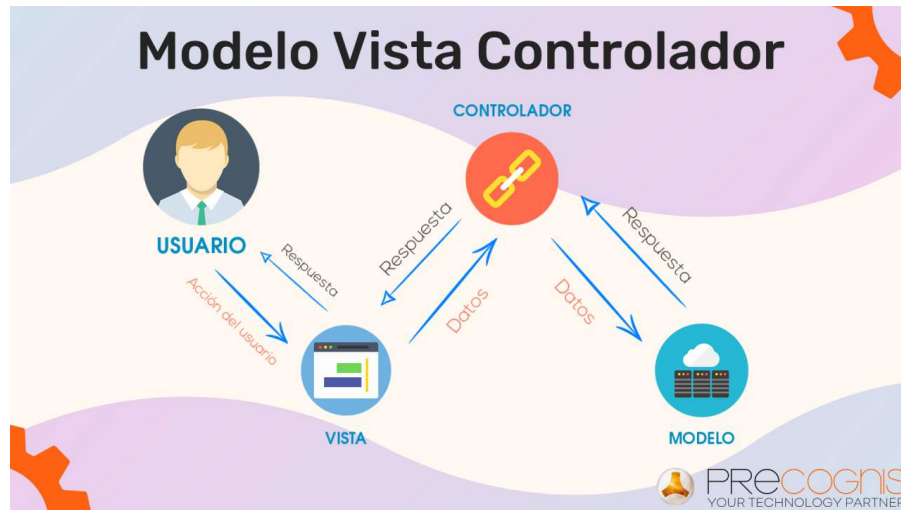
Modelo (Model):

- Interacción con la **Base de Datos** (`SELECT`, `INSERT`, `UPDATE`, `DELETE`)
- Lógica relacionada con los datos, como restricciones y campos derivados.
- Comunica con el [controlador](#).
- Utiliza clases, validaciones, y puede hacer uso de un [ORM \(Object-Relational Mapper\)](#).

```
# models.py
from django.db import models
```

```
class Person(models.Model):
    email = models.EmailField(primary_key=True)
    firstname = models.CharField(max_length=100)
    lastname = models.CharField(max_length=100, null=False)
```

Vista (View):



- Interactúa con el usuario.
- Usualmente HTML, CSS y JavaScript.
- Utiliza un motor de plantillas (templates).
- Comunica con el [controlador](#).

```
<html>
...
<div>
  Person {{ id }} <br />
  {{ datos_personales }}
</div>
...
</html>
```

Controlador (Controller):

- Recibe URLs (routing).

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



- Procesa los requests (GET, POST, PUT, DELETE)
- Interactúa con el modelo.
- Rellena y envía las plantillas (templates de la vista).

```
from flask import render_template

@app.route('/mostrar/usuario/<id>')
def datos_usuario():
    datos = Consulta_Base_de_Datos_Usuarios(id)
    return render_template('usuario.html', id=id, datos_per
```

▼ Preguntas de examen

▼ Explica el concepto de Modelo-Vista-Controlador (MVC) en el desarrollo de aplicaciones web. ¿Cuál es el propósito de cada componente?

El MVC es un **patrón arquitectónico** que divide una aplicación en tres componentes: **Modelo** (interacción con la BD y lógica de datos), **Vista** (interfaz de usuario), y **Controlador** (gestiona las solicitudes y interactúa con el modelo).

El **Modelo** se encarga de la lógica de datos y se comunica con el Controlador.

La **Vista** interactúa con el usuario y se comunica con el Controlador.

El **Controlador** procesa solicitudes, interactúa con el Modelo y envía datos a la Vista.

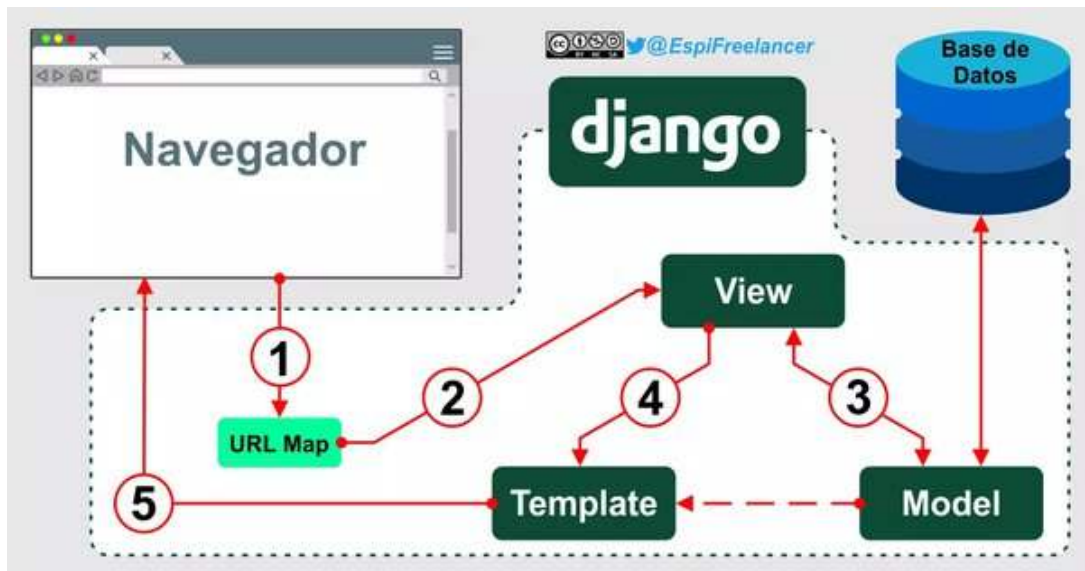
▼ Describe la Vista en el patrón MVC. ¿Qué tecnologías suelen utilizarse en la implementación de la Vista en una aplicación web?

La **Vista** en el patrón MVC es donde interactúa el usuario. Se implementa comúnmente con HTML, CSS y JavaScript, utilizando un motor de plantillas para renderizar datos dinámicos. Además, se comunica con el **Controlador**.

▼ Django



Django es un **framework** web de **alto nivel**, escrito en Python, que fomenta un desarrollo rápido y limpio de aplicaciones web.



Flujo de Datos de Django

Características:

- Es uno de los **frameworks** más utilizado.
- Sigue el patrón **MTV** (Modelo, Template, Vista), que es equivalente a MVC. Los **modelos** representan la estructura de la **base de datos**, las **vistas** procesan las **solicitudes** y las **plantillas** definen la **presentación** de los datos.
- Utiliza:
 - [Object Relational Mapper](#).
 - Templates.
 - Interfaz de administración (CRUD).
 - Autenticación, Cache, Sesiones, etc.
- Django se maneja con comandos del `manage.py` como `startproject`, `startapp`, `makemigrations`, `migrate`, y `runserver`.



Algunas alternativas a Django son *Flask*, *FastApi* y *Express*.

▼ Pasos para crear una app en Django

1. Instala Django:

```
pip install django
```

2. Crea un proyecto Django:

```
django-admin startproject nombre_proyecto
```

3. Crea una aplicación dentro del proyecto:

```
cd nombre_proyecto  
python manage.py startapp nombre_app
```

4. Configura la aplicación en el proyecto:

Agrega el nombre de la aplicación (`nombre_app`) en la lista `INSTALLED_APPS` dentro de `settings.py` en `nombre_proyecto` .

5. Define modelos en la aplicación:

En el archivo `models.py` de `nombre_app` , define los modelos que representan la estructura de la base de datos.

6. Haz migraciones:

```
python manage.py makemigrations  
python manage.py migrate
```

7. Crea vistas y plantillas:

Define vistas y plantillas HTML para manejar solicitudes y mostrar información.

8. Ejecuta el servidor de desarrollo:

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



```
python manage.py runserver
```

Flask vs Django

Las diferencias clave entre **Flask** y **Django** incluyen:

- Flask es más ligero y flexible, mientras que Django ofrece más funcionalidades integradas.
- Django proporciona un ORM integrado, sistema de administración y un conjunto completo de funcionalidades, mientras que Flask es más modular y requiere configuración adicional para ciertas características.

▼ Preguntas para el Examen:

▼ En el contexto de Django, ¿cuál es la función del Modelo y cómo se define? Proporciona un ejemplo de modelo en Django.

En Django, el Modelo interactúa con la Base de Datos y define la lógica relacionada con los datos. Se define mediante clases y puede hacer uso de un ORM. Ejemplo:

```
# model.py
from django.db import models

class Person(models.Model):
    email = models.EmailField(primary_key=True)
    firstname = models.CharField(max_length=100)
    lastname = models.CharField(max_length=100, null=F
```

▼ Explica el papel del Controlador en una aplicación web basada en MVC. Proporciona un ejemplo de cómo se codificaría el Controlador en Flask.

El Controlador recibe URLs, procesa requests, interactúa con el Modelo y rellena/envía plantillas. En Flask, el Controlador se codifica mediante funciones o métodos decorados. Ejemplo:

```
from flask import render_template
```

▼ HTML5

SGML

SGML es un modelo que permite **estandarizar** un **documento electrónico**, independientemente del sistema en el que se utilice.

HTML

Es un lenguaje de marcado basado en SGML que **estructura** y **despliega** una página **web** y sus contenidos.

Características y Elementos

- **Marcado y Etiquetas:**

Utiliza etiquetas para marcar diferentes partes del contenido.

```
<p>Este es un párrafo</p>
```

- **Estructura Jerárquica:**

La estructura de un documento HTML es jerárquica. Los elementos se anidan unos dentro de otros para formar la estructura general de la página.

Un documento HTML típicamente comienza con un elemento `<html>` que contiene `<head>` y `<body>`, y dentro de ellos se encuentran otros elementos que representan el contenido.

- Tiene un **conjunto fijo de etiquetas** definido en su especificación.

- **Menos extensible** que XML.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Título de la Página</title>
  </head>
  <body>
    <h1>Encabezado Principal</h1>
    <p>Este es un párrafo de texto.</p>
```



```
<ul>
  <li>Elemento de lista 1</li>
  <li>Elemento de lista 2</li>
</ul>
<a href="https://www.ejemplo.com">Enlace a Ejemplo.com</a>
</body>
</html>
```

XML

XML es un **lenguaje de marcado**, que evoluciona de SGML. **Describe** y **organiza** datos de manera **estructurada** y se centra en la presentación de información.

Características

- **Marcado y Autodescriptivo:**

XML utiliza etiquetas para marcar los elementos de un documento.

```
<nombre>John</nombre>
```

Las etiquetas definen la estructura jerárquica de los datos, los cuáles están incluidos en el propio XML.

- **Extensibilidad:**

Se pueden **definir etiquetas** y reglas de marcado según las necesidades del usuario.

Ejemplo de documento XML:

```
<persona>
  <nombre>John</nombre>
  <edad>30</edad>
  <direccion>
    <calle>Main Street</calle>
    <ciudad>Cityville</ciudad>
  </direccion>
</persona>
```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



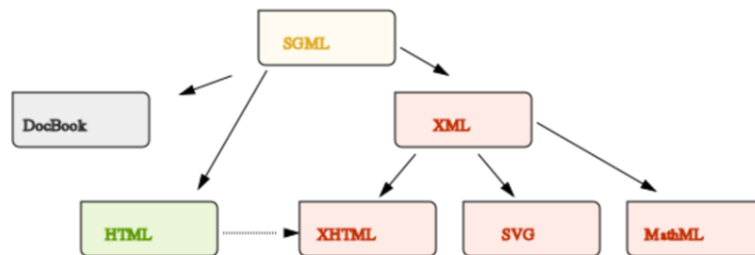
Necesito concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

XHTML

Es un **lenguaje de marcado** que **combina HTML y XML**. Su **estructura es más rigurosa** al estar basado en XML principalmente.



En resumen, **SGML** es el estándar, **HTML** es utilizado para la creación de páginas web, **XML** es utilizado para el intercambio de datos estructurados, y **XHTML** es una variante más estricta y basada en XML de HTML .

Diferencias HTML vs XHTML

HTML se centra en la presentación web con una sintaxis flexible. XHTML es una versión más estricta y basada en XML de HTML, con reglas de marcado más rigurosas. XHTML sigue las reglas de XML y requiere una sintaxis más precisa. HTML es más tolerante con errores, mientras que un documento XHTML inválido puede causar problemas. HTML5 combina características de ambas, ofreciendo flexibilidad sin comprometer la calidad del marcado.

CSS



CSS es un **lenguaje de estilo** utilizado para controlar la presentación y el diseño de páginas web, permitiendo la personalización de colores, fuentes, márgenes y más.

Bootstrap



Bootstrap es un **framework front-end** de código abierto que proporciona un **conjunto de estilos predefinidos** y **componentes** de diseño para **facilitar** la **creación** de sitios **web** responsivos y estéticamente agradables.

Tailwind CSS



Tailwind CSS es un **framework de utilidad** que se centra en clases de bajo nivel (p. ej., `bg-blue-500`, `p-4`, `text-center`) para aplicar **estilos directamente** en el marcado **HTML**.

▼ Formularios

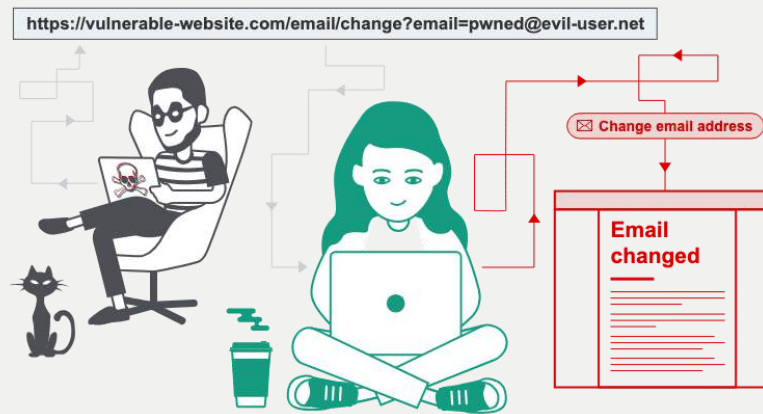
Django ofrece la clase `Forms` para facilitar la entrada de datos desde Formularios.

Características de la clase `Form`

- **Generación** de **HTML** en las plantillas
- **Conexión** con el **modelo**
- **Validación** en el servidor
- Tratamiento automático de los errores
- Seguridad (contra) **CSRF** (mediante el token CSRF)



El **CSRF** (Cross-site request forgery) es una vulnerabilidad web que permite al atacante influir en los usuarios para que lleven a cabo acciones que no quieren tomar realmente.



Para prevenir este tipo de ataques, los desarrolladores web implementan medidas de seguridad como **tokens CSRF** (únicos por sesión y verificados). [Más info](#)

▼ Ejemplo de formulario

```
# forms.py
from django import forms

class NameForm(forms.Form):
    your_name = forms.CharField(label="Your name", max_l
```

```
<!-- en formulario.html (templates) -->
<form action="/formulario" method="post">
    {% csrf_token %} # token de seguridad
    {{ form }}
</form>
```

```
# views.py
from django.shortcuts import render, redirect
from .forms import NameForm # Importa el formulario
```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

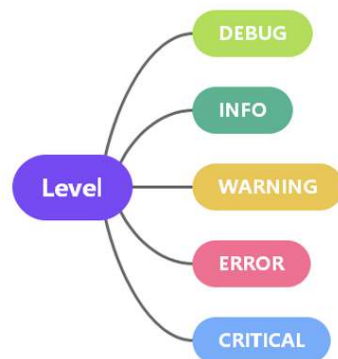
ali ali oohh
esto con 1 coin me
lo quito yo...

WUOLAH

```
def Add(request):  
    form = NameForm()  
  
    if request.method == "POST":  
        form = NameForm(data=request.POST)  
  
        if form.is_valid():  
            form.save()  
            return redirect('donde_sea') # Redirige a d  
  
    # Cuando GET o cuando no valida y rellena errores  
    context = {  
        'form': form  
    }  
    return render(request, 'template.html', context)
```

▼ Logging

El módulo `logging` proporciona un sistema flexible y configurable para registrar información acerca de lo que está sucediendo en tu aplicación.



Tiene distintos niveles de registro, que representan la severidad o importancia de un mensaje del log.

En entornos de desarrollo, es común utilizar niveles más bajos (como `DEBUG` y `INFO`) para tener una visión detallada del comportamiento de la aplicación.

En producción, es recomendable configurar niveles más altos (como `ERROR` o `CRITICAL`) para identificar y resolver problemas importantes sin inundar los

registros con información menos relevante.

Beneficios del logging en Django:

- **Depuración:** Ayuda a encontrar y solucionar errores.
- **Rastreo:** Permite rastrear eventos y actividades en la aplicación.
- **Monitorización:** Facilita la monitorización del rendimiento y comportamiento de la aplicación.

▼ Autenticación

El proceso de autenticación en el desarrollo de aplicaciones web aborda aspectos como el uso de **cookies**, **tokens** y **métodos de autenticación** web.

En **Django**, la autenticación se gestiona mediante **modelos de usuarios**, contraseñas, grupos y permisos. Las sesiones y middleware se emplean para **gestionar automáticamente** la autenticación de usuarios, **verificando credenciales** en la base de datos.

La ampliación o modificación del modelo de usuario se realiza mediante la **extensión del modelo existente**.

Si se quiere utilizar cuentas de redes sociales se puede usar el plugin `django-allauth`

Autenticación en APIs

Podemos usar dos métodos para autenticar a los usuarios de una API:

- Usando **cookies con sesiones** para sincronizar cliente/servidor,
- Usando **Tokens** en la cabecera HTTP

```
Authorization: Token asdf245adsfvasf2q45daf4123
# o con un token mas complejo
Authorization: Bearer asdf245asddfgdg.dfwawwtdsfvasf2.dsgdc
```

Json Web Token (JWT)

El JSON Web Token, es un estándar abierto (RFC 7519) utilizado para la creación de tokens de acceso que permiten la **transferencia segura de**

información. Están representados en formato JSON y constan de tres secciones, separadas por puntos (.):

1. **Header (Encabezado):** Contiene información sobre el tipo de token y el algoritmo de cifrado utilizado. Por ejemplo:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

2. **Payload (Carga útil):** Contiene los datos (claims) que se desean transmitir, como información de usuario, roles, permisos, etc. Por ejemplo:

```
{
  "usuario_id": 123,
  "nombre": "Usuario Ejemplo",
  "rol": "admin"
}
```

3. **Signature (Firma):** Es la firma digital generada con una clave secreta conocida solo por el servidor. Sirve para verificar la integridad del token y asegurar que no haya sido manipulado.

▼ APIs Restful



Una API RESTful (o API REST) es una **interfaz de programación de aplicaciones** que sigue los principios del **estilo arquitectónico REST** (Representational State Transfer).

Este estilo se basa en estándares y convenciones para facilitar la comunicación entre sistemas en la web. Son populares debido a su **simplicidad**, **escalabilidad**, y facilidad para entender y utilizar.

Características del estilo arquitectónico REST

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

- Ser **state-less** → Para que toda la información necesaria para obtener un recurso se realice en la misma llamada
- Usar los verbos HTTP (**GET**, **POST**, **PUT**, **PATCH**, **DELETE**) explícitamente
- Devolver XML o JSON
- Usar **URIs estilo path** → Es decir, que se puedan leer por personas, que muestren un orden. Es mejor para el SEO (Search Engine Optimization). Los paths de la API se denominan **endpoints**

```
// en lugar de:  
GET /product?name=jacket  
  
// mejor:  
GET /api/product/name/jacket
```

En una API RESTful, los verbos HTTP se utilizan para realizar operaciones específicas en los recursos:

Petición **GET**

Es utilizado para **solicitar datos** de un recurso específico en un servidor. Por lo general, se utiliza para obtener información y no para realizar cambios en el servidor.

```
GET https://api.ejemplo.com/users/1
```

Petición **POST**

Se emplea para **enviar datos al servidor para crear un recurso nuevo**. Puede enviar datos en el cuerpo de la solicitud HTTP y el servidor puede procesar esos datos para crear un nuevo recurso.

```
POST https://api.ejemplo.com/users
```

```
Body:  
{  
  "nombre": "Ejemplo Usuario",
```

```
"email": "ejemplo@email.com",  
"contrasena": "contrasena123"  
}
```

Petición **PUT**

Similar a POST, pero se usa para enviar datos al servidor para **crear o reemplazar** el recurso en su totalidad en la ubicación especificada. **Si el recurso ya existe, se reemplaza**; si no existe, se crea uno nuevo en la ubicación proporcionada.

```
PUT https://api.ejemplo.com/users/1  
  
Body:  
{  
  "nombre": "Usuario Actualizado",  
  "email": "nuevo@email.com",  
  "contrasena": "nuevacontrasena456"  
}
```

Petición **PATCH**

Se usa para realizar **modificaciones parciales** en un recurso. A diferencia del PUT, que reemplaza completamente el recurso, PATCH se usa para actualizar **solo una parte específica** del recurso.

```
PATCH https://api.ejemplo.com/users/1  
  
Body:  
{  
  "nombre": "Nombre Modificado"  
}
```

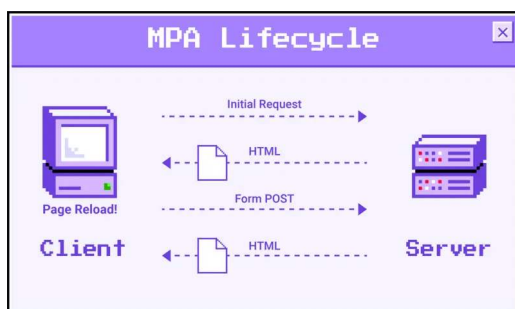
Petición **DELETE**

Se utiliza para **eliminar un recurso específico** en el servidor.

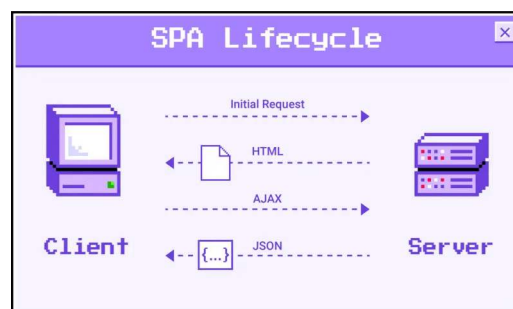
```
DELETE https://api.ejemplo.com/users/1
```

MPAs vs SPAs

Aspecto	Aplicaciones Web de Página Única (SPAs)	Aplicaciones Web de Múltiples Páginas (MPAs)
Arquitectura	Estructura de una sola página	Estructura con múltiples páginas
Navegación	Transiciones sin recarga de página	Recarga de página en cada transición
Interactividad	Respuestas rápidas, interacciones dinámicas	Puede parecer más lenta con recargas de página
Desarrollo	Basadas en frameworks JS (React, Angular, Vue.js)	Menos dependencia de frameworks de frontend
Mantenimiento	Estructura compleja que puede requerir más esfuerzo	Estructura más sencilla, mantenimiento más simple
SEO (Search Engine Optimization) e indexación	Difícil optimizar si no se implementan técnicas adecuadas	Más amigable para motores de búsqueda
Carga inicial	Puede ser más lenta debido a la carga inicial	Tiempo de carga inicial más rápido



Ciclo de vida de una MPA



Ciclo de vida de una SPA

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali oohh
esto con 1 coin me
lo quito yo...

WUOLAH



En resumen:

- **Complejidad vs. Velocidad:** SPAs ofrecen una experiencia más fluida, pero la complejidad puede requerir más recursos de desarrollo.
- **SEO y indexación:** MPAs son más amigables para motores de búsqueda, mientras que las SPAs requieren cuidados adicionales para la indexación.
- **Experiencia de usuario:** Las SPAs suelen ofrecer una experiencia más similar a las aplicaciones nativas.

▼ Document Object Model (DOM)



El Document Object Model (DOM) es esencial en el desarrollo web y permite la manipulación de la estructura HTML mediante JavaScript. Se pueden buscar y cambiar nodos usando métodos como `getElementById` y `querySelectorAll`.

Además, se exploran eventos, manejadores y se introduce la sintaxis de funciones de flecha de ES6. Se aborda AJAX con la biblioteca `axios`, que facilita llamadas asíncronas y se compara con la funcionalidad nativa `fetch` de ES6. Ambos métodos devuelven promesas para manejar respuestas asíncronas de manera efectiva en aplicaciones web.

Tipos de nodos

Root Node (Nodo Raíz):

Representa el nodo más alto en la jerarquía del DOM y generalmente coincide con el documento HTML completo.

Se accede a él con `document.documentElement` o simplemente `document`.

Element Node (Nodo de Elemento):

Representa elementos HTML, como `<div>`, `<p>`, `<a>`, etc.

Contiene otros nodos, como nodos de texto, atributos y otros elementos, formando la estructura del DOM.

```
<div id="miDiv">Este es un div</div>
```

Text Node (Nodo de Texto):

No se crea directamente como un nodo separado, ya que forma parte del contenido de los elementos HTML. Contiene texto dentro de un elemento y representa el contenido textual del mismo.

Puede ser accedido a través de la propiedad `textContent` o `innerText` del elemento.

Attribute Node (Nodo de Atributo):

Representa los atributos de un elemento, como `id`, `class`, `href`, etc.

Se accede a ellos a través de métodos específicos o propiedades del elemento.

```
<a href="https://www.ejemplo.com" id="miEnlace">Enlace de E
```

Buscar Nodos

getElementById

El `document.getElementById('id')` busca y devuelve el elemento con el ID especificado.

querySelector y querySelectorAll

El `document.querySelector('selector')` devuelve el **primer elemento** que coincide con el selector CSS especificado.

El `document.querySelectorAll('selector')` devuelve una NodeList de **todos los elementos que coinciden** con el selector. Útil para selecciones múltiples.

getElementsBy

Métodos como `getElementsByClassName` y `getElementsByTagName` seleccionan elementos por su clase o tipo, respectivamente. Devuelven una HTMLCollection.

Cambiar Nodos

innerHTML

El `element.innerHTML` permite obtener o **establecer el contenido HTML** de un elemento. Útil para cambiar completamente el contenido de un elemento.

Propiedades del Nodo

Diversas propiedades del nodo, como `element.textContent`, `element.value` (para elementos de formulario), permiten modificar el contenido de manera más específica.

Modificación de Estilos

El `element.style` proporciona acceso a las propiedades de **estilo del elemento**. Puedes cambiar el estilo directamente o agregar y quitar clases con `element.classList` para aplicar estilos predefinidos.

Atributos

Los atributos de los elementos se pueden modificar utilizando las propiedades como `element.setAttribute` y `element.removeAttribute`.

Cambios en el estado del documento que desencadenan respuestas específicas, como hacer clic, cargar una página o enviar un formulario.

Eventos, manejadores, funciones de flecha

Manejadores de Eventos:

- Funciones de JavaScript que responden a eventos específicos. Se asignan a elementos HTML y definen cómo la página debe reaccionar cuando ocurre un evento particular.

Funciones de Flecha (Arrow Functions):

- Las funciones de flecha son una forma concisa de escribir funciones en JavaScript. Tienen una sintaxis más corta y vinculan automáticamente el valor de `this`, lo que las hace útiles para los manejadores de eventos.

```
// Función tradicional
function suma(a, b) {
  return a + b;
}
```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

```
// Función de flecha equivalente
const sumaArrow = (a, b) => a + b;

// Uso de las funciones
console.log(suma(2, 3)); // Salida: 5
console.log(sumaArrow(2, 3)); // Salida: 5
```

Eventos:

▼ JQuery



JQuery es una librería que permite añadir una **capa de interacción AJAX** entre la web y las aplicaciones que desarrollemos controlando eventos, creando animaciones y otros efectos de forma que se genere menos código.

Ejemplos de sintaxis:

```
jQuery('h2').css('color', 'blue')
```

ó

```
$('#h2').css('color', 'blue')
```

```
$.each($('h2'), function() {
  $(this).css("color", "#333");
});
```

ó

```
$('#h2').css('color', '#333')
```

```
$('#h2 + p').css('color', '#946900');
```

Diferencia entre \$.each() y \$('h2')

En resumen, `each()` se utiliza para iterar sobre los elementos seleccionados para realizar **operaciones personalizadas** en cada uno, mientras que los métodos **sin** `each()` se aplican directamente a todos los elementos seleccionados de manera simultánea, realizando **una acción** específica en todos ellos.

▼ Preguntas de examen

▼ ¿Para qué es jQuery?

jQuery es una librería que permite añadir una **capa de interacción AJAX** entre la web y las aplicaciones que desarrollemos controlando eventos, creando animaciones y otros efectos de forma que se genere menos código.

▼ ¿Cómo es la sintaxis de una función de jQuery?

La sintaxis de una función de jQuery es `$(selector).accion()`.

▼ Pon un ejemplo de una función que afecte a todos los elementos de un HTML con atributo `class="a"` y otro ejemplo que afecte al elemento `<div id="2"></div>`

- Para afectar a todos los elementos HTML con atributo `class="a"`:

`$(".a").accion()` ó

```
$(".a").each(function() {  
  // Hacer algo con cada elemento, por ejemplo, cambi  
  $(this).css("background-color", "blue");  
});
```

- Para afectar al elemento `<div id="2"></div>`: `$("#2").accion()`

```
$("#2").hide()
```

▼ ES6

El ES6 (ECMAScript 6) es una versión de la especificación del lenguaje de programación JavaScript lanzada en 2015. Incluye las siguientes características nuevas y mejoras:

1. **Arrow Functions (Funciones de Flecha):** Sintaxis más corta y concisa para definir funciones, especialmente útil en funciones de callback.

```
// Sintaxis de función de flecha
const sumar = (a, b) => a + b;
```

2. **Let y Const:** Introducción de `let` y `const` para declarar variables, ofreciendo un alcance de bloque más preciso que `var`.

```
// let permite reasignación
let x = 10;
x = 20;

// const para variables inmutables
const PI = 3.1416;
```

3. **Plantillas de Cadena (Template Strings):** Uso de comillas graves (``...``) para crear cadenas de texto multilínea y la interpolación de variables.

```
const nombre = 'Juan';
const saludo = `Hola, ${nombre}!
Bienvenido a nuestro sitio.`;
```

4. **Desestructuración (Destructuring):** Extracción de datos de arreglos o objetos en variables individuales de forma más concisa.

```
const persona = { nombre: 'Ana', edad: 30 };
const { nombre, edad } = persona;
```

5. **Spread/Rest Operator (Operador de Propagación/Resto):** Uso del operador `...` para descomponer elementos de un arreglo o objeto o para agrupar elementos en un arreglo o parámetros de función.

```
// Spread para descomponer arreglos
const array1 = [1, 2, 3];
const array2 = [4, 5, 6];
const arrayConcatenado = [...array1, ...array2];
```


Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

```
// Rest para agrupar elementos
function sumar(...numeros) {
  return numeros.reduce((total, num) => total + num, 0)
}
```

6. **Clases:** Introducción de una sintaxis más simple para definir clases y trabajar con herencia.

```
class Persona {
  constructor(nombre) {
    this.nombre = nombre;
  }
  saludar() {
    return `Hola, soy ${this.nombre}`;
  }
}
const persona1 = new Persona('Carlos');
```

7. **Módulos:** Introducción de un sistema de módulos nativo para organizar y estructurar mejor el código.

```
// Archivo modulo.js
export const PI = 3.1416;
export function duplicar(numero) {
  return numero * 2;
}

// En otro archivo
import { PI, duplicar } from './modulo.js';
```

8. **Promesas (programación asíncrona):** Representación de operaciones asíncronas de una manera más clara y con un manejo más fácil de errores.

```
const miPromesa = new Promise((resolve, reject) => {
  // Lógica asíncrona
  setTimeout(() => {
    const exito = true;
    if (exito) {
```

```

        resolve('¡Éxito!');
    } else {
        reject('Error :(');
    }
}, 2000);
});

// Usar la promesa
miPromesa
    .then(resultado => console.log(resultado)) // En caso
    .catch(error => console.error(error)); // En caso de

```

9. **Funciones de Mapa y Conjunto (Map y Set):** Estructuras de datos adicionales que permiten manejar colecciones de datos únicos o asociados.

```

// Crear un mapa
const mapa = new Map();

// Agregar elementos al mapa
mapa.set('clave1', 'valor1');
mapa.set('clave2', 'valor2');

// Crear un conjunto
const conjunto = new Set();

// Agregar elementos al conjunto
conjunto.add(10);
conjunto.add(20);
conjunto.add(10); // No se añadirá, ya que los conjuntos

```

AJAX



AJAX es una **técnica** de desarrollo web que implica **enviar y recibir datos** desde el servidor de forma **asíncrona** sin tener que recargar toda la página.

ES6 introdujo algunas características (las promesas) que han hecho más fácil trabajar con AJAX y realizar solicitudes HTTP desde JavaScript.

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Error en la solicitud');
    }
    return response.json();
  })
  .then(data => {
    // Trabajar con los datos obtenidos
    console.log(data);
  })
  .catch(error => {
    // Manejar errores
    console.error('Error:', error);
  });
```

▼ React

React se utiliza para desarrollar aplicaciones web con interfaces de usuario (UI) complejas.

Parte de la aplicación se traslada al cliente mediante código ES6, que se empaqueta en un archivo bundle.js generado con Webpack.

Arquitectura Cliente-Servidor:

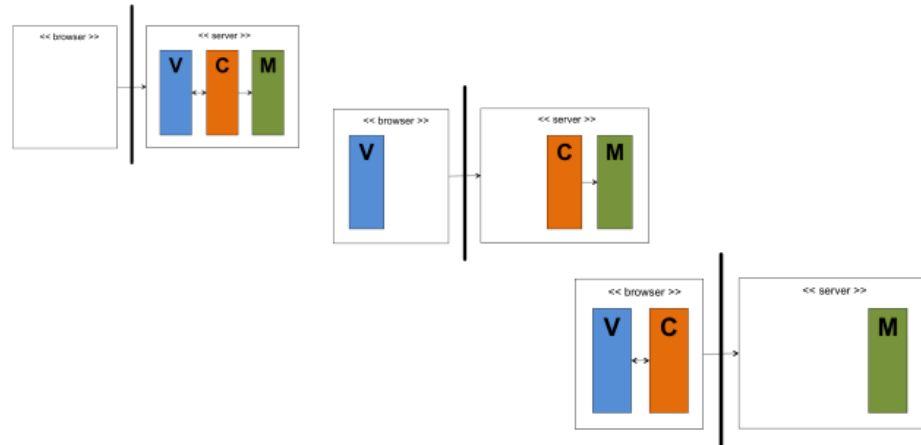
Se transita de la arquitectura MVC tradicional a un modelo en el que el servidor maneja el modelo y la autenticación/autorización, mientras que el cliente se encarga del controlador y la vista.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

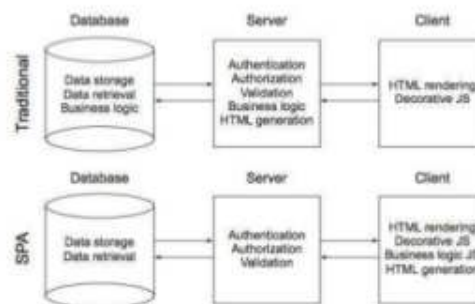
perdo
espacio



Características

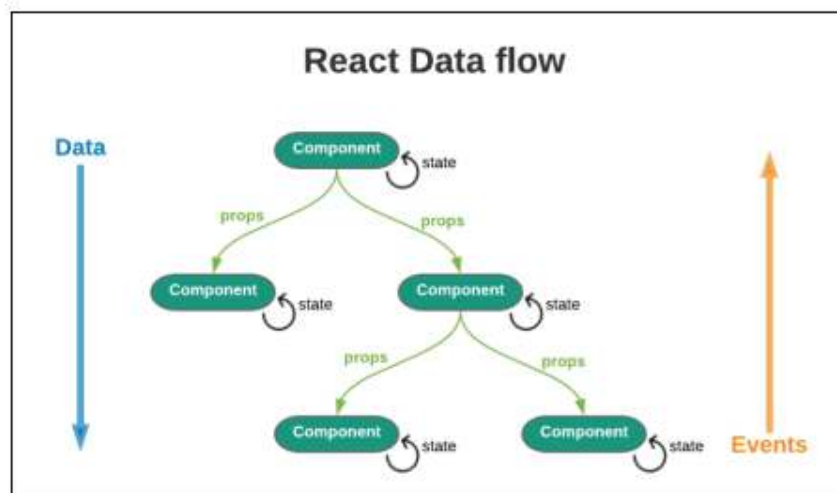
- Es **especialmente efectivo** para construir **Single Page Applications (SPA)**, donde solo el modelo y las operaciones de autenticación/autorización residen en el servidor.

Traditional vs SPA



- En el modelo MVC, React se posiciona como la **capa de vista (V)**, siendo una librería dedicada al desarrollo de interfaces de usuario en el front-end.
- Permite la creación de **componentes reutilizables**, lo que hace que la deuda técnica sea menor.

- Utiliza **JSX**, una extensión de JavaScript que facilita la creación de elementos de interfaz de usuario.
- Implementa un **DOM Virtual** para optimizar las actualizaciones y mejorar el rendimiento.
- React permite **gestionar eventos** y **estados** en los **componentes**, facilitando la interactividad.
- El sistema de props (propiedades) permite la **comunicación entre componentes**.



Frameworks Adicionales:

Para el desarrollo Frontend: NEXT.js o Astro.

Para Server Side Rendering (SSR): Vite.

Data Fetching con React:

Se usa `useState` y `useEffect` para realizar fetch de datos de manera eficiente.

▼ Despliegue

Tras la etapa de producción, llega la de despliegue. Para ello, en Django deberemos **deshabilitar el ambiente de depuración** (`DEBUG`) y **cambiar el servidor de desarrollo por el de producción** (el definitivo).

```
# settings.py

# Se pasa la variable DEBUG a False:
# DEBUG = True
DEBUG = False

# Se incluye el host de producción en los ALLOWED_HOSTS
# ALLOWED_HOSTS = []
ALLOWED_HOSTS = ['*'] # o el que sea en producción
```

Servidor web

El servidor web se encarga de:

- Servir archivos
- Redirigir `requests`
- Cifrar los datos (cifrado https)
- Lanzar procesos para atender a varios clientes a la vez
- Gestionar la caché
- Balancear la carga

Nginx

Nginx es un **servidor web** de código abierto y un servidor proxy inverso (que redirige `requests`). Está diseñado para ser **ligero, rápido, eficiente y escalable**, y es ampliamente utilizado para servir contenido web estático, como archivos

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

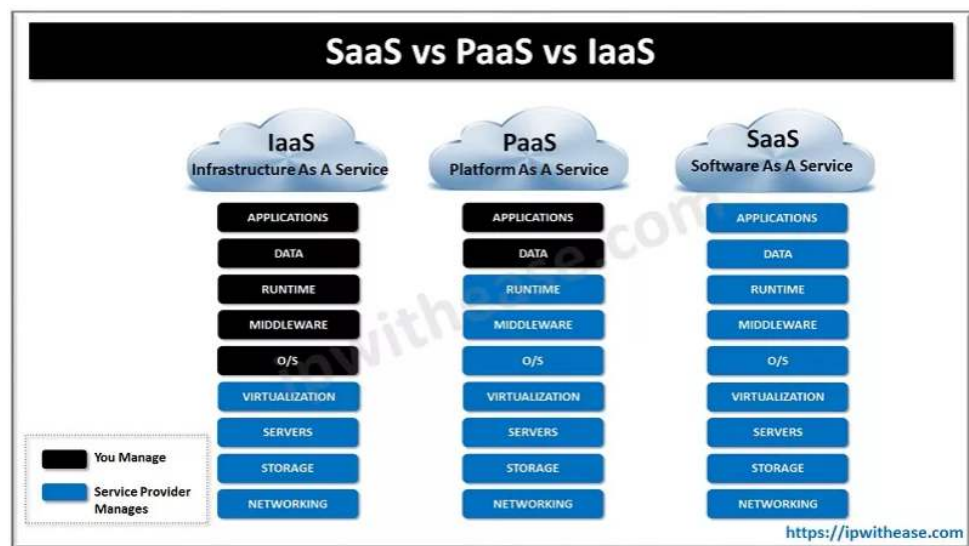
pierdo espacio



HTML, CSS, JavaScript, imágenes, y para actuar como un proxy para aplicaciones web.

SaaS vs PaaS vs IaaS

Estos tres términos se refieren a **modelos de servicios en la nube** que ofrecen diferentes niveles de infraestructura y servicios para satisfacer las necesidades de diferentes tipos de usuarios y empresas.



En resumen, **IaaS** proporciona **recursos básicos** de infraestructura, **PaaS** ofrece un **entorno de desarrollo y despliegue**, mientras que **SaaS** ofrece **aplicaciones completas** y listas para usar.