TALENTOFUTURO

Clase 1: Introducción al desarrollo con frameworks

Módulo 4: Desarrollo Backend en Entornos Python

Apoyado por:
CORFO

Clase de hoy

01

Frameworks de desarrollo de aplicaciones

02

Tipos de frameworks

03

Componentes de una arquitectura web

04

Stacks de desarrollo

1A

Frameworks Web

Bloque A

Qué veremos en bloque 1A

- **Desarrollo de aplicaciones con Frameworks**
  - ¿ Que es un Framework?
  - Framework vs Biblioteca
  - Patrón MVC vs MVT
  - Componentes Framework
- **Tipos de frameworks**
  - Frameworks Python

¿Qué es un Framework?

- Un **framework** es un conjunto de herramientas, bibliotecas y convenciones que proporcionan una estructura y un conjunto de funcionalidades predefinidas para el desarrollo de aplicaciones.

- Permiten manejar tareas comunes del desarrollo web:
  - Organización del código
  - Autenticación
  - Enrutamiento
  - Acceso a base de datos
  - Seguridad: inyección de SQL, Cross Site Scripting.
  - Testing

- Estas características permiten tener mayor **rapidez en el desarrollo**.

Framework vs Biblioteca

- **Biblioteca:**
  - Es un conjunto de funciones o clases que los desarrolladores pueden llamar cuando lo necesiten. La biblioteca es utilizada para resolver tareas específicas.
  - Los desarrolladores controlan cuándo y cómo usar la librería.
- **Framework:**
  - Impone una estructura y controla el flujo del programa.
  - Es el framework quien llama al código que el desarrollador escribe, no al revés.

El framework marca las reglas del juego y define cómo debe estructurarse la aplicación, mientras que, en el caso de una librería, el desarrollador tiene total libertad para usarla según lo requiera.

Patrón MVC

- **Modelo-Vista-Controlador (MVC)** Es un patrón común en frameworks de desarrollo web.
  - El **Modelo** es el objeto que representa la información del dominio e interactúa con la base de datos.
  - La **Vista** presenta la información del modelo al usuario. La vista trata **sólo** sobre presentar la información.
  - El **Controlador** gestiona las interacciones del usuario, es el pegamento entre vista y modelo: toma la entrada del usuario manipula el modelo y actualiza la vista.

MVC permite separar la presentación del modelo: UI / Lógica de negocio

Separar el controlador de la vista permite tener múltiples presentaciones de la información.

Patrón MVC/MVT

- Django sigue un paradigma llamado Model-View-Template (MVT):
  - El **Modelo** interactúa con la información de la base de datos
  - La **Vista** en Django se encarga de consultar e interactuar con los modelos.
  - El **Template** presenta la información

M	model	model	M
V	view	view	V
C	controler	template	T

https://docs.djangoproject.com/en/5.1/faq/general/#faq-mtv

Componentes Framework

- **ORM (Mapeo Objeto-Relacional):**
  - Facilita la interacción con bases de datos sin tener que escribir SQL manualmente.
- **Sistema de Rutas:**
  - Define cómo las URLs se mapean a las diferentes funciones o vistas de la aplicación.
- **Control de Estado y Sesiones:**
  - Proporcionan mecanismos para manejar sesiones de usuario, lo que permite saber si un usuario está autenticado.
- **Manejo de Plantillas (Templates):**
  - Para generar HTML dinámico
- **Autenticación y Autorización**
- **Validación de Formularios**

Tipos de Frameworks

- **Full Stack Framework**: Proporciona todas las herramientas necesarias para desarrollar aplicaciones completas (frontend y backend), ideal para proyectos grandes.

- **Micro Framework**: Ofrece lo esencial (enrutamiento y manejo de solicitudes HTTP), perfecto para proyectos pequeños y personalizados.

- **Framework Asíncrono**: Optimizado para alta concurrencia y aplicaciones en tiempo real (WebSockets, múltiples tareas simultáneas).

Tipos de Frameworks

Framework	Ventajas	Desventajas	Ejemplo
Full	Integración total Estandarización	Curva de aprendizaje Menor Flexibilidad	Django
Micro	Flexibilidad Ligero	Más trabajo manual Inadecuado para proyectos grandes	Flask
Asíncrono	Alto rendimiento Eficiencia y escalabilidad	Complejidad de concurrencia Soporte de librerías	FastAPI

Frameworks Python

- **Django**: Framework completo con todas las herramientas integradas, ideal para aplicaciones grandes con muchas funcionalidades.

- **Flask**: Ligero y flexible, ofrece solo lo esencial, perfecto para proyectos pequeños y personalizados.

- **FastAPI**: Rápido y eficiente, diseñado para APIs modernas con programación asíncrona y validación automática de datos.

Trabajo grupal – Bloque A

Paralelo 2

G1	G2	G3	G4
Nicolas Mardones	Víctor Meza Herrera	Daniela Méndez Gándara	Claudia Blanco
Manuel Denis	Estefania Manriquez	Álvaro Pérez	Ariel Inostroza
Bryan Castillo	Patricio Vera	Pedro Nahum	Héctor Aguayo
GERALDY SUAREZ	Oscar Torres	Javier Gajardo	Ruben Sanhueza Ramirez
Scarlett Espinoza	Braulio Quiroz	Angela Proboste Neira	Félix González
Ulises Campodónico	Yerko Gallardo	Nicolás Guzmán	Ariel Mora
Carol Leiva	Rodrigo Araya		

G5	G6	G7	G8
Fabian Díaz	Camila Oyarzún	Mayerlyn Rodriguez	Daniela Porto
Natalia Rivera	Stefanya Pulgar	Sebastian Vega	Cristian Chavez Jara
Juan Salinas	Carlos Emilio Azócar Riquelme	Efrain Duarte Campos	Juan Rodrigo Vega
Rodrigo Pastén Cortés	Nicolas Rojas	Bianel Bianchini	Rodolfo Cantillana
Flavio Jara R.	luis.paillan.cnc@gmail.com	Bastián Gamboa Labbé	Abraham Ruiz
Daniel García	Cristóbal Gajardo	Pablo Uribe	Rodrigo Álvarez

Trabajo grupal - Ejercicio #1

Para entender los beneficios de usar un framework y comprender los componentes de este vamos a crear nuestro propio framework desde cero usando python.

Utilizando el archivo python con la implementación del servidor *server.py* y el punto de entrada *app.py*:

- Implemente una vista de acuerdo con la filosofía de Django (views.py)
- La vista en este caso asumirá las funciones de presentación y modelo, que aún no hemos implementado.
- La vista debe ser un objeto que implemente el metodo `get_response`

Trabajo grupal - Ejercicio #1

```python
# server.py
from http.server import BaseHTTPRequestHandler, HTTPServer

class RequestHandler(BaseHTTPRequestHandler):
    def __init__(self, *args, route_handler=None, **kwargs):
        self.route_handler = route_handler
        super().__init__(*args, **kwargs)

    def do_GET(self):
        path = self.path
        if self.route_handler:
            response = self.route_handler(path)
        else:
            response = "404 Not Found"
        self.send_response(200 if response != "404 Not Found" else 404)
        self.send_header("Content-Type", "text/plain")
        self.end_headers()
        self.wfile.write(response.encode("utf-8"))

def start_server(port=8000, route_handler=None):
    def handler(*args, **kwargs):
        RequestHandler(*args, route_handler=route_handler, **kwargs)
    server_address = ('', port)
    httpd = HTTPServer(server_address, handler)
    print(f"Servidor corriendo en el puerto {port}...")
    httpd.serve_forever()
```

```python
# app.py
from views import View
from server import start_server

ROUTES = {
    "/": View(),
}

def route_request(path):
    view = ROUTES.get(path)
    if view:
        return view.get_response()
    else:
        return "404 Not Found"

if __name__ == "__main__":
    start_server(port=8000, route_handler=route_request)
```

Trabajo grupal - Ejercicio #2

Implemente una lógica de templates simples

- Cree un nuevo archivo llamado *templates.py*, este debe definir un método llamado *render_template*
- El método recibe un nombre de *template* y lee el contenido del archivo desde un directorio llamado templates, retornando su contenido.
- La vista ahora retornará el resultado de llamar a render_template sobre el template *home.html*
- Asegúrese de cambiar el tipo de contenido del server, ahora es:

```
self.send_header("Content-Type", "text/html")
```

Break!

1B

Frameworks Web

Bloque B

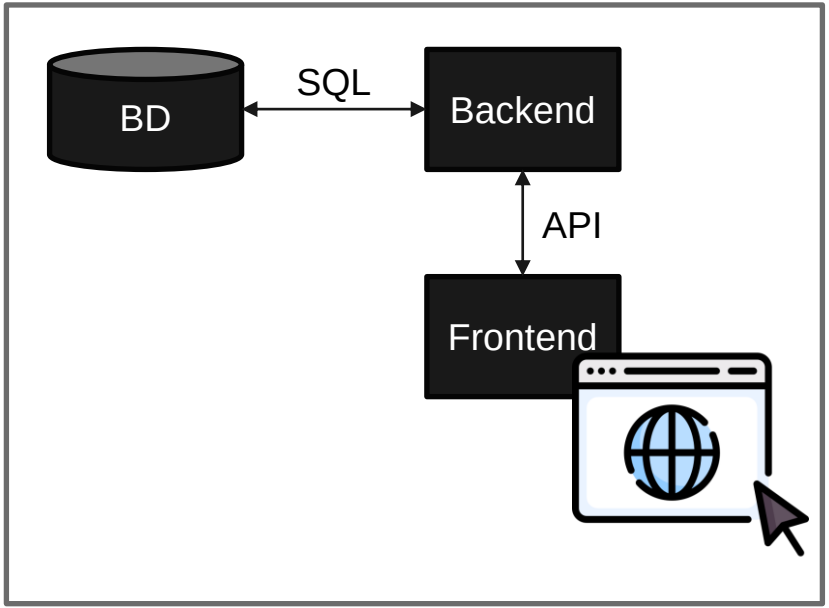
Qué veremos en bloque 1B

- **Componentes de una arquitectura web**
  - Roles en el Desarrollo Web
- **Stacks de desarrollo**
  - Popularidad de Stacks y Tecnología

Componentes Arquitectura Web

- Cliente (frontend)
- Servidor (backend)
- Base de Datos
- API

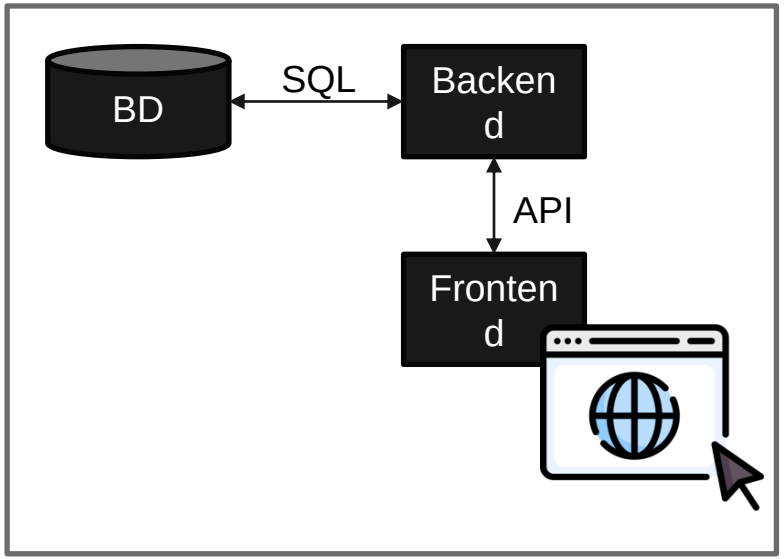
- Otros:
  - Balanceador de carga
  - Taras en 2do plano
  - Servidor de cache
  - Pub/Sub
  - Assets/CDN



Arquitectura Web

Roles en el Desarrollo Web

- **Desarrollador Frontend:**
  - Interfaz de usuario.
  - HTML, CSS y JavaScript
- **Desarrollador Backend:**
  - lógica del servidor, la interacción con bases de datos, APIs.
  - Python: Django, Flask
- **Desarrollador Fullstack:**
  - Combina habilidades de frontend y backend
- **Diseñador UX/UI:**
  - Experiencia de usuario e interfaz de usuario.
  - Figma/InVision
- **Arquitecto de Software:**
  - Arquitectura completa del sistema.
  - Elección de tecnologías y patrones, escalable y mantenible.
- **Ingeniero DevOps:**
  - Infraestructura y del flujo de trabajo entre desarrollo y operaciones. CI/CD
  - Terraform/CloudFormation

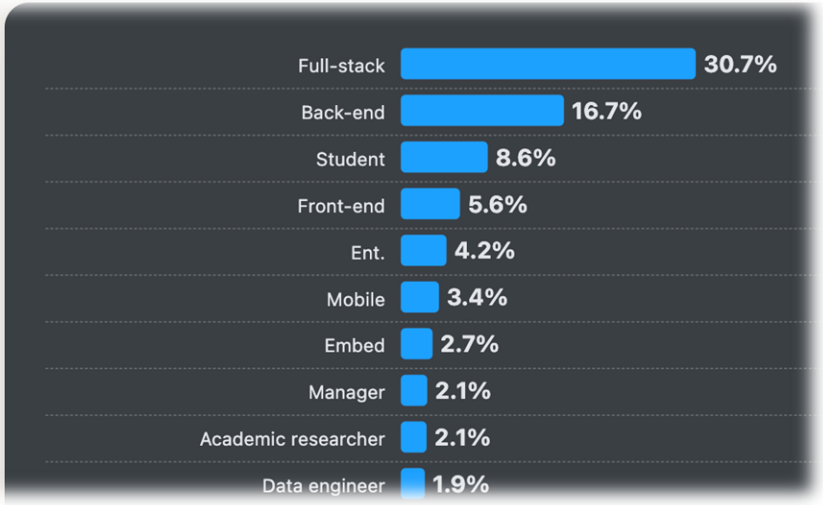


Arquitectura Web

Roles en el Desarrollo Web

Developer type

Full-stack, back-end, and front-end developers were the top three roles reported by developers for the last three years; however, front-end developers have decreased from 6.6% to 5.6% since last year. This year, more developers identify as students than as front-end employees and we saw a slight increase in developers who have an embedded applications role or are academic researchers.



2024 Developer Survey Stack Overflow - fuente

Stacks de desarrollo

- Un **stack de desarrollo** es el conjunto de herramientas, tecnologías y lenguajes de programación que se utilizan para desarrollar una aplicación.
- Es como la “caja de herramientas” de un desarrollador, donde se incluyen tanto las tecnologías de frontend como las de backend, además de bases de datos, servidores, etc.

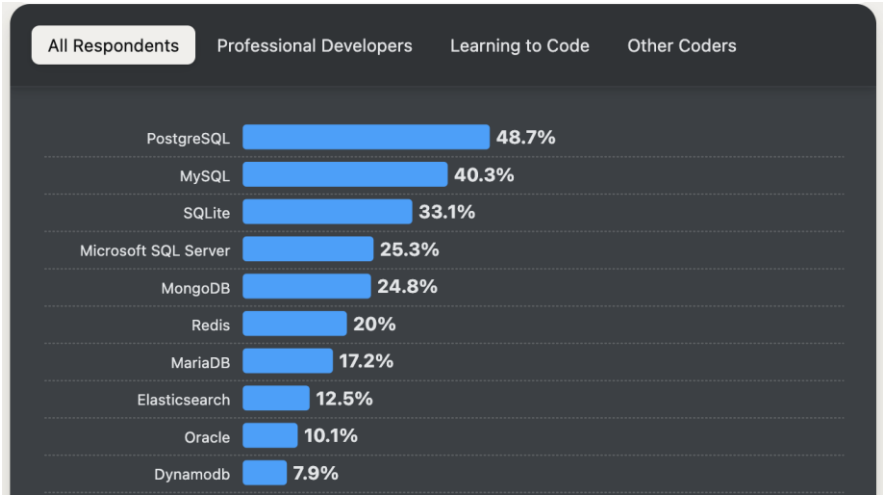
Stacks de desarrollo

- **Stacks Populares**
  - **LAMP:** Linux, Apache, MySQL, PHP/Python. Un stack clásico para desarrollo web.
  - **MERN:** MongoDB, Express, React, Node.js. Popular para aplicaciones JavaScript modernas

Stacks de desarrollo

Lenguajes

Bases de Datos



Source: Survey Stack Overflow 2024 Tech link

Stacks de desarrollo

Web Frameworks

Source: Survey Stack Overflow 2024 Tech link

Trabajo grupal – Bloque B

Paralelo 2

G1	G2	G3	G4
Nicolas Mardones	Víctor Meza Herrera	Daniela Méndez Gándara	Claudia Blanco
Manuel Denis	Estefania Manriquez	Álvaro Pérez	Ariel Inostroza
Bryan Castillo	Patricio Vera	Pedro Nahum	Héctor Aguayo
GERALDY SUAREZ	Oscar Torres	Javier Gajardo	Ruben Sanhueza Ramirez
Scarlett Espinoza	Braulio Quiroz	Angela Proboste Neira	Félix González
Ulises Campodónico	Yerko Gallardo	Nicolás Guzmán	Ariel Mora
Carol Leiva	Rodrigo Araya		

G5	G6	G7	G8
Fabian Díaz	Camila Oyarzún	Mayerlyn Rodriguez	Daniela Porto
Natalia Rivera	Stefanya Pulgar	Sebastian Vega	Cristian Chavez Jara
Juan Salinas	Carlos Emilio Azócar Riquelme	Efrain Duarte Campos	Juan Rodrigo Vega
Rodrigo Pastén Cortés	Nicolas Rojas	Bianel Bianchini	Rodolfo Cantillana
Flavio Jara R.	luis.paillan.cnc@gmail.com	Bastián Gamboa Labbé	Abraham Ruiz
Daniel García	Cristóbal Gajardo	Pablo Uribe	Rodrigo Álvarez

Trabajo grupal - Ejercicio #1

Terminemos la implementación de nuestro framework MVC, para esto crearemos el modelo.

- Cree un archivo llamado models.py este tendrá nuestros modelos y lógica de negocios, no interactuará con ninguna base de datos por simplificación, pero tendremos una lista de avisos.
- Implemente un método llamado all (de instancia por simplicidad) que retorne el listado como un diccionario.
- Desde la vista importe el modelo e imprima de momento por consola el resultado de invocar el método all.

Trabajo grupal - Ejercicio #2

Finalmente conectaremos el último componente, usemos los datos del modelo para renderizar.

- Modificaremos el archivo templates.py para que utilice jinja como lenguaje de templates. Se proporciona el nuevo archivo.
- Cambie la vista para entregar las variables a usar en el renderizado como contexto, se debe pasar un diccionario.
- Utilice estas variables en el template home.html para renderizar los avisos

Trabajo grupal - Ejercicio #2

```python
# templates.py
from jinja2 import Environment, FileSystemLoader

# Configurar el entorno Jinja2
TEMPLATES_DIR = "templates"
env = Environment(loader=FileSystemLoader(TEMPLATES_DIR))

def render_template(template_name, context=None):
    """Renderiza un template de Jinja2 con el contexto dado."""
    try:
        # Cargar el template desde el directorio de templates
        template = env.get_template(template_name)
        # Renderizar el template con el contexto
        return template.render(context or {})
    except FileNotFoundError:
        return "Template not found"
```