Programación IV
Profesora Cynthia Estrada

Fecha: 28/08

"Frameworks 2"

Integrantes:

- "Díaz Rossini Juan José",
- "Gallo Genaro"
- "Navarro Victor Leandro"

---

1. ¿Cómo se pueden clasificar los frameworks?
2. Realice cuadro con frameworks más usados
3. ¿Cuáles son algunos recursos para aprender sobre framework en los lenguajes más usados?
4. Crear una lista con 5 consejos para usar framework en los lenguajes más usados
5. De todos los framework encontrados e identificados seleccione el que más le llamó la atención y crear un mapa conceptual sobre el mismo mostrando las cualidades que lo hacen mejor que otros
6. Nombra algunos frameworks de frontend
7. Nombra algunos frameworks del backend
8. Explique 2 problemas que se pueden producir al usar framework
9. ¿Cuál es la diferencia entre un framework de caja blanca contra uno de caja negra?
10. Cuáles son las normas o patrones que se recomiendan para los desarrolladores de framework explique cada una en detalle.

---

1. ¿Como se pueden clasificar los frameworks?

Según el tipo de desarrollo

Tipo	Descripción	Ejemplos
Frontend	Para el desarrollo de interfaces de usuario en aplicaciones web o móviles.	React, Angular, Vue.js
Backend	Para la lógica del servidor y acceso a base de datos.	Django, Laravel, Express.js
Fullstack	Soportan tanto frontend como backend.	Meteor, Next.js, Nuxt.js
Móviles	Especializados en el desarrollo de apps móviles.	Flutter, React Native, Ionic
Escritorio	Para aplicaciones que corren en sistemas operativos.	Electron, Tkinter, PyQt
Testing	Enfocados en testeo de aplicaciones.	Jest, Mocha, Selenium

Según su estructura

Arquitectura	Descripción
MVC (Modelo-Vista-Controlador)	Separa datos, lógica de negocio y presentación.
MVVM (Modelo-Vista-VistaModelo)	Muy usado en aplicaciones móviles.
SPA (Single Page Application)	Las páginas no se recargan, solo cambian vistas.
Microservicios	Divide la app en servicios pequeños y autónomos.

Según el lenguaje de programación

Lenguaje	Frameworks comunes
JavaScript	React, Angular, Express, Vue.js
Python	Django, Flask, FastAPI
Java	Spring, JSF, Hibernate
PHP	Laravel, Symfony, CodeIgniter
Ruby	Ruby on Rails
C# (.NET)	ASP.NET Core, Blazor
Dart	Flutter

---

2. Los frameworks más usados

Lenguaje	Frontend	Backend	Móvil	Fullstack / Otros
JavaScript	React, Vue.js, Angular	Express.js, Next.js	React Native	Meteor, Next.js
Python	(No aplica directamente)	Django, Flask, FastAPI	Kivy	Django + React
Java	(No aplica directamente)	Spring Boot, JSF	Android SDK	Spring Fullstack
PHP	(No aplica directamente)	Laravel, Symfony, CodeIgniter	-	Laravel Fullstack
Ruby	(No aplica directamente)	Ruby on Rails	-	Rails

C#	Blazor (web frontend)	ASP.NET Core	Xamarin	.NET MAUI
Dart	-	-	Flutter	Flutter Web + Móvil

---

3. Recursos para aprender sobre frameworks (por lenguaje)

Lenguaje	Frameworks	Recursos recomendados
JavaScript	React, Vue, Angular	- React Docs- Vue Mastery- Angular Docs
Python	Django, Flask	- Django Oficial- Flask Tutorial- Free Code Camp YouTube
PHP	Laravel, Symfony	- Laravel Docs- Symfony Casts
Ruby	Rails	- Rails Guides- The Odin Project (Ruby on Rails section)
Java	Spring Boot	- Spring Docs- Baeldung
C#	ASP.NET	- Microsoft Learn
Dart	Flutter	- Flutter Official - Academind Flutter Course

---

4. Lista con 5 consejos para usar frameworks en los lenguajes más usados:

- **Lee la documentación oficial:** Aunque los tutoriales ayudan, nada reemplaza conocer bien la documentación oficial del framework. Ahí están las buenas prácticas y las actualizaciones.

- **Mantén las dependencias actualizadas:** Usa herramientas como npm audit, pip, composer o maven para mantener todo al día. Los frameworks actualizados son más seguros.

- **No abuses de la "magia" del framework:** Muchos frameworks automatizan tareas. Aprende cómo funcionan internamente para no volverte dependiente o cometer errores invisibles.

- **Configura un entorno de desarrollo coherente:** Usa contenedores (Docker, por ejemplo) o entornos virtuales para evitar conflictos entre versiones.

- **Aprende el lenguaje antes del framework:** Muchos desarrolladores saltan al framework sin dominar el lenguaje base. Esto complica el aprendizaje y dificulta la resolución de errores.

---

5. Mapa conceptual

Concepto Principal	Subtema	Descripción / Detalles
Django	Seguridad	Protección contra XSS, CSRF, SQLi, clickjacking. Incluye middlewares de seguridad.
	Arquitectura	Basado en MVT (Modelo-Vista-Template), una variante de MVC.
	Automatización	Admin panel autogenerado, migraciones de base de datos, formularios automáticos.
	Documentación	Oficial muy completa y clara. Ejemplos, tutoriales y guías paso a paso.
	Comunidad	Comunidad grande, activa y colaborativa. Muchas librerías y soluciones listas.
	Escalabilidad	Soporta desde MVPs hasta grandes aplicaciones con alto tráfico.
	Ecosistema	Compatible con Django REST Framework, Celery, Channels (websockets), etc.
	Extensibilidad	Sistema robusto de apps reutilizables y configuración modular.
	Pruebas integradas	Soporte nativo para tests unitarios y de integración con unittest.

---

6. Nombra algunos frameworks de frontend

- **React.js**
- **Angular**
- **Vue.js**
- **Svelte**
- **Ember.js**
- **Backbone.js**
- **Preact**
- **Next.js** (aunque es fullstack, tiene un fuerte componente frontend)

---

7. Nombra algunos frameworks de backend

- **Express.js** (Node.js)
- **Django** (Python)
- **Flask** (Python)
- **Fast API** (Python)
- **Spring Boot** (Java)
- **Ruby on Rails** (Ruby)

- **Laravel** (PHP)
- **ASP.NET Core** (C#)
- **Nest JS** (Node.js

---

8. Explique 2 problemas que se pueden producir al usar framework

1) Dependencia tecnológica

Cuando una aplicación se desarrolla fuertemente con un framework, puede quedar "atada" a él. Si el framework deja de actualizarse o se vuelve impopular, la migración es costosa.

2) Curva de aprendizaje alta

Algunos frameworks tienen muchas convenciones, configuraciones y automatismos (como Angular o Spring), lo que implica que los desarrolladores deben invertir mucho tiempo en entender cómo funciona todo antes de ser productivos.

---

9. FRAMEWORK CAJA BLANCA VS CAJA NEGRA

Tipo de Framework	Descripción	Ejemplo
Caja Blanca	El desarrollador controla el flujo. El framework llama al código del desarrollador. Se puede modificar el comportamiento estándar.	Flask, Express, React
Caja Negra	El framework impone su propio flujo. El desarrollador adapta su código a las reglas del framework.	Django, Rails, Angular

---

10. ¿Cuáles son las normas o patrones que se recomiendan para los desarrolladores de frameworks? Explicación detallada

Cuando se desarrolla un **framework**, no basta con que funcione: debe ser mantenible, extensible, seguro y fácil de usar. Aquí tienes las **normas y patrones fundamentales** que se recomiendan:

1) Inversión de Control (IoC)

- **Qué es:** En lugar de que el desarrollador controle el flujo, el framework lo hace e invoca el código del usuario cuando lo necesita.
- **Por qué es importante:** Permite que los frameworks definan la estructura del programa, lo cual estandariza el desarrollo.
- **Ejemplo:** En Django, el framework decide cuándo ejecutar una vista; tú solo defines la función.

2) Principio de Convención sobre Configuración

- **Qué es:** Reduce la necesidad de configuración explícita si el desarrollador sigue convenciones preestablecidas.
- **Por qué es útil:** Acelera el desarrollo, ya que disminuye el tiempo de configuración inicial.

- **Ejemplo:** Ruby on Rails crea rutas automáticamente si nombras las vistas y controladores con las convenciones correctas.

3) Modularidad

- **Qué es:** Un buen framework debe estar dividido en módulos independientes y reutilizables.
- **Por qué es importante:** Facilita la extensión, el testeo y el mantenimiento del framework.
- **Ejemplo:** Flask permite usar solo los módulos que necesitas (ORM, autenticación, etc.), haciéndolo muy ligero.

4) Documentación clara y coherente

- **Qué es:** Explicar cómo usar cada componente del framework con ejemplos reales y completos.
- **Por qué es vital:** Un framework sin documentación clara se vuelve inútil, incluso si está bien programado.
- **Buenas prácticas:** Incluir tutoriales, API references, casos de uso, y ejemplos interactivos.

5) Compatibilidad y mantenimiento

- **Qué es:** Garantizar que el framework funcione con versiones actuales de lenguajes, navegadores y tecnologías.
- **Importancia:** Mantiene la relevancia y seguridad del framework con el tiempo.
- **Ejemplo:** React se actualiza frecuentemente para ser compatible con nuevas versiones de Node, Webpack y navegadores.

6) Separación de responsabilidades (SoC - Separation of Concerns)

- **Qué es:** Cada parte del framework debe encargarse de una sola responsabilidad.
- **Beneficio:** Facilita la comprensión del código, el testeo y el mantenimiento.
- **Ejemplo:** En Laravel, los controladores gestionan la lógica y las vistas se encargan de la presentación.

7) Patrones de diseño (como MVC, MVVM, etc.)

- **Qué son:** Estructuras arquitectónicas que organizan el código de forma mantenible y escalable.
- **Ejemplo:** MVC separa modelo, vista y controlador. Muy usado en frameworks como ASP.NET, Django, Rails.

8) Extensibilidad (Plug-in Architecture)

- **Qué es:** Permitir que otros desarrolladores creen módulos o extensiones que amplíen la funcionalidad.
- **Por qué es clave:** Fomenta una comunidad activa y da vida al ecosistema del framework.
- **Ejemplo:** Express.js tiene una gran cantidad de middlewares creados por terceros.

9) Pruebas integradas (Testing Built-in)

- **Qué es:** Incluir soporte directo para testing (unitario, de integración, etc.) dentro del framework.
- **Beneficio:** Facilita el desarrollo seguro, especialmente en grandes equipos.

- **Ejemplo:** Django incluye un sistema de testing con unittest, incluso para pruebas de base de datos.

10) Gestión de errores clara y trazable

- **Qué es:** Mensajes de error comprensibles, con trazas útiles y sugerencias.
- **Por qué importa:** Mejora la experiencia del desarrollador y acelera el debugging.
- **Ejemplo:** Angular muestra mensajes precisos sobre errores en los templates o dependencias.