















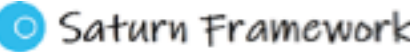

















LES FRAMEWORKS *EXEMPLES*

Langage	Tierless	Fullstack	Micro-Framework	CS Web UI	GUI
Java		 			
Scala					
Kotlin				Kotlin multiplatform	Kotlin multiplatform
C#				* 	 Include in .net6 MAUI
F#				 Elmish	
OCaml			Opium	ocaml-vdom	
Javascript					
Python			 Flask		
Rust				*  YEW	

* WASM

BACKEND API

PRATIQUES USUELLES

La programmation monothreadée asynchrone est la norme (sauf en Java) à base de Fiber* vs thread pool

=> à la base du langage : webAPI (js browser) / event-loop (node.js) ;

=> pleinement intégré : Coroutine (Kotlin); go-routine (Go); asyncio (python) ; projet Loom (WIP - Java)

=> fourni par lib tierce : FutureImpl Vert.x (Java); Lwt (OCaml) ; Cats-effect, ZIO (Scala) ; Tokio (Rust)

Montée en puissance des architectures « middleware » face aux « vieux » MVC ; dans certains cas d'autres patterns sont intéressants (machines de Moore, CQRS) ;

=> express (node.js) ; Vert.x (Java) ; Opium (OCaml), Flask (python); Ktor (Kotlin); Saturn (F#); actix (Rust)

Organisation du code en architecture hexagonale ou en oignon courante (structure de projet vu en ALOM)

*a.k.a green thread a.k.a light thread a.k.a co-routine a.k.a event-loop

MIAGE M2 - QUALITÉ DU SI - THOMAS HAESSLÉ & QUENTIN BURG

Feigning knowledge of a word you've heard a few times



Expert

Pretending to Know
About Stuff

O RLY?

@ThePracticalDev