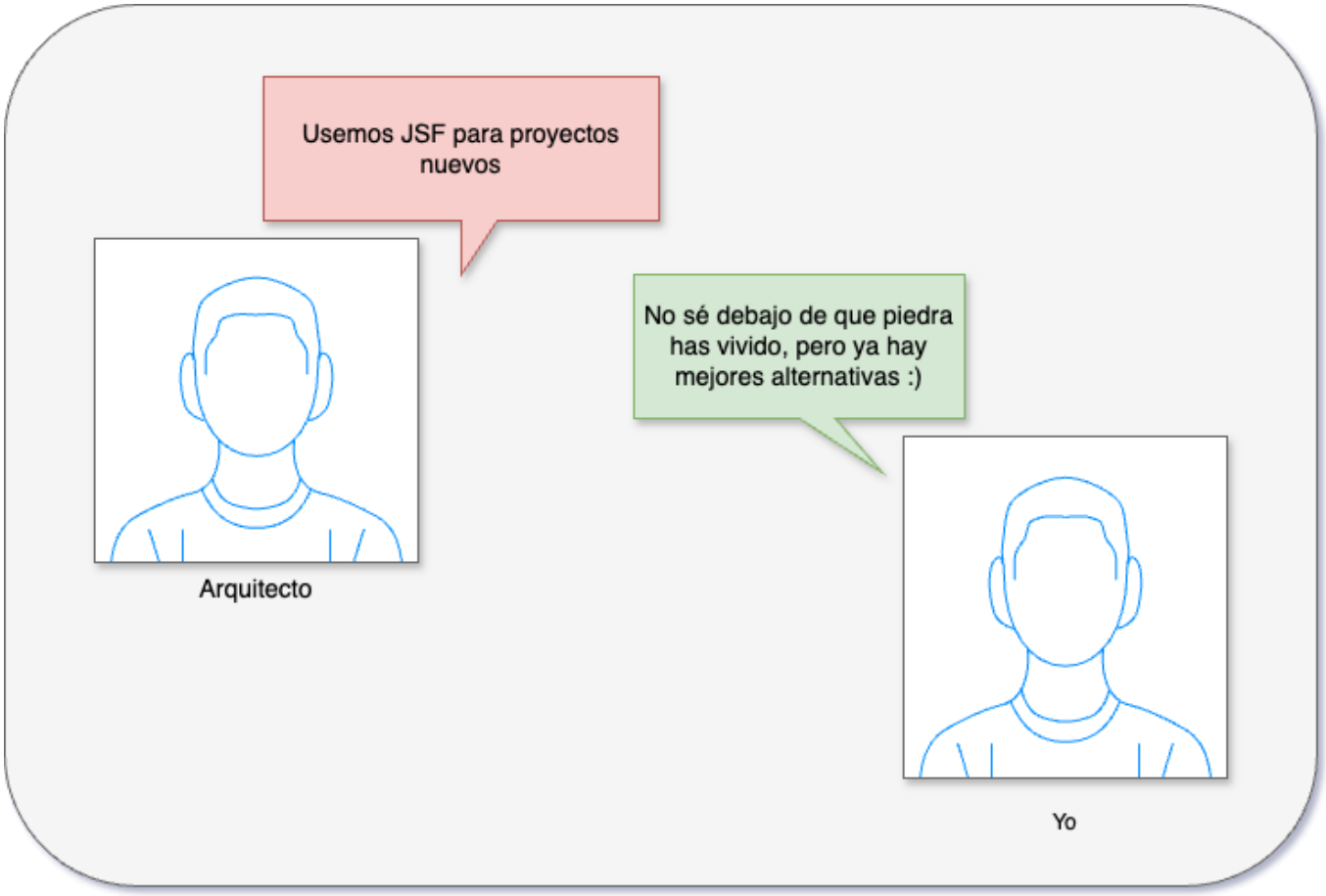

Hace poco estuve charlando con un amigo, no es cercano pero tenemos una relación de colegas grata, el me comentaba que en el lugar donde trabajar van a iniciar la modernización desde cero de 5 proyectos vitales en la organización y bueno me contó entre otras cosas que escogieron JSF para empezar estos nuevos proyectos.

Los motivos que dieron es que JSF a avanzado mucho en los últimos años y que quieren tener aplicaciones con un alto performance del lado del cliente, además de que su equipo tendría una menor curva de aprendizaje.

Honestamente entiendo la segunda razón, pero la primera la verdad es que no, no sé si es que su arquitecto vive debajo de una piedra pero la cantidad de tecnologías que son superiores a JSF en ofrecer una experiencia rica al usuario final no son pocas.

No entiendo y nunca entenderé porque muchas empresas se empecinan en no innovar, parece que no fuera esta una profesión basada en ciencias, parece que fuera un culto a la

rentabilidad y a los gráficos de desempeño.

Es francamente increíble.

Alternativas a JSF para Java Devs

Miren voy a ser muy honesto, cualquier framework/librería de JavaScript como Vue, React o Angular siempre que usen TypeScript van a entregar una experiencia familiar para los Java Devs, entonces cualquiera de esos es una buena alternativa, sobretodo Angular porque al ser TypeScript por defecto tiene bastantes adeptos que son Java devs.

Por otro lado están las propuestas SSR como Angular Universal, NextJs o Nuxt que son aún más superiores a JSF, hace un par de días creé un artículo sobre este tema y te lo dejo aquí ahí te enseño en que se diferencia JSF al Server Side Rendering sobretodo porque hay gente por ahí asegurando que JSF es igual que SSR.

JSF no es basura

Faltaría más que yo, hablara mal de JSF no es para nada una mala herramienta pero es que utilizarla ahora como la herramienta #1 para construir aplicaciones web enriquecidas la verdad es que no tiene mucho sentido, el ciclo de vida de los beans, la complejidad de la actualización de componentes, y otras cosas. Son temas que sabemos que podemos hacer mucho mejor con otras alternativas.

Si el problema es que usan una librería como PrimeFaces pues la misma gente de PrimerTek tiene una alternativa, han construido librerías para los frameworks que mencioné antes, así que las puedes usar y así pueden lograr reducir aún más la curva de aprendizaje, por ejemplo un buen punto de partida es usar TypeScript y alguna de las librerías de Prime, ahí empiezan a tener un proceso de innovación de aplicaciones bastante fuerte y moderno.

La curva de aprendizaje como excusa para no innovar

Yo creo que esto puede ser todo un artículo pero aquí rápidamente voy dar una opinión, no puede ser que el argumento para seguir trabajando siempre con lo mismo es que tiene una curva de aprendizaje larga, pero si de eso se trata esta carrera de aprender.

Es una profesión basada en ciencia, la base de la ciencia es la evolución e innovación con un propósito determinado, y no podemos simplemente seguir 10, 20 o 30 años con tecnologías

que ahora no son lo mejor solo por el hecho que migrar a una nueva tiene un costo de aprendizaje mayor.

Si la tecnología no es el top #1 no vale la pena sacrificar tantos atributos de calidad solo porque la curva de aprendizaje de otra tecnología superior es un poco alta.