códigofacilito

Cómo Elegir Un Framework Para El Backend

Uriel Hernandez CTO

Lees en 8 Min. 19437 visitas 01/08/18

Una de las decisiones más complicadas que enfrentan los nuevos desarrolladores web, es ¿con qué tecnologías voy a trabajar?

A diferencia del Frontend en donde existe al menos un estándar base de tecnologías, en el Backend, la variedad es muchísima entre lenguajes de programación, frameworks, gestores de base de datos, servidores, etc. Es por eso que una de las preguntas más populares que recibimos es ¿qué framework debería usar? ¿Laravel? ¿Rails? ¿Express? ¿Adonis? ¿Cuál es el mejor?

¿Express? ¿Adonis? ¿Cuál es el mejor?

Primero, aunque existen una serie de métricas que podemos usar para establecer una superioridad entre un framework u otro, como lo son la velocidad de respuesta, soporte a carga de peticiones alta, concurrencia, entre otros; sigue siendo imposible e innecesario definir qué framework es el mejor, en parte porque existen puntos subjetivos (que se prestan al debate) como la productividad, la expresividad, mejores prácticas, etc. puntos importantes a la hora de elegir un framework que no pueden ser medidos y que por lo tanto, hacen imposible la tarea de definir un mejor framework.

A continuación voy a entrar en detalle sobre algunos puntos de consideración a la hora de elegir un framework.

CONVENCIÓN VS CONFIGURACIÓN.

En algunas ocasiones, hemos podido escuchar frases como "Este

framework usa convención sobre configuración" o en otras menos populares "este framework es configuración sobre convención",

estas frases dicen muchísimo acerca de cómo es desarrollar en cada framework, sin embargo, cuando no tenemos mucha experiencia puede ser complicado razonar cuál es la diferencia entre un framework orientado a la convención y uno orientado a la configuración.

CONVENCIÓN

Según WordReference, la palabra convención significa

> *Una norma o práctica admitida por responder a precedentes o a la costumbre.*

En programación, una convención es una práctica recomendadas por una comunidad, luego de que la experiencia de miles de programadores dicte que seguir una convención en específico produce mejores resultados, dicho de otras palabras, es algo que hacemos porque a otros les ha funcionado muy bien antes.

Un ejemplo de convención es escribir las variables en camelCase, como se recomienda en algunos lenguajes:

```
let frameworkWeb = 'express';
```

Mientras que en algunos otros, la convención es usar guiones:

```
framework_web = 'Rails';
```

Las antes mencionadas son convenciones de sintaxis, y son sólo una categoría de las muchas convenciones que hay en programación, así como estas existen otro tipo de convenciones que pueden utilizarse para la organización del código, la arquitectura del proyecto, los nombres de los archivos, de los métodos, de las clases, etc.

CONFIGURACIÓN

Cuando hablamos de configuración, hablamos de una tecnología que no asume nada por ti y deja a tu decisión y libertad el que tú elijas cuáles serán las convenciones de tu proyecto.

Si bien, todos los proyectos de software, sin importar con qué framework sean desarrollados, siguen convenciones, algunos frameworks vienen pre configurados para seguir una serie de convenciones que la comunidad recomienda para dicho framework. En otros casos, los frameworks no asumen ninguna configuración y más bien permiten que tú tengas control absoluto sobre cómo será trabajar con dicho framework.

Los frameworks de configuración suelen incluir, como podrás imaginarte, múltiples archivos dedicados únicamente a la

configuración del framework y las partes que lo componen.

CONVENCIÓN SOBRE CONFIGURACIÓN Y VICEVERSA

Un framework de convención sobre configuración, como mencionaba antes, está compuesto de múltiples convenciones que han sido pensadas y perfeccionadas a lo largo de múltiples iteraciones del framework mismo.

Además de que sabemos que existen estas convenciones, el framework mismo está desarrollado para funcionar bajo estas convenciones, por lo que sería difícil querer que el framework funcionara lejos de dichas prácticas.

Los beneficios de establecer estas convenciones con el framework son muchos, entre ellos el que es posible obviar algunos puntos del desarrollo, como por ejemplo en Rails, donde no es necesario que definamos qué vista mostrará un método, ya que la convención es que la vista tendrá el mismo nombre que el método y se encontrará en una carpeta con el nombre de la clase del controlador, esto es un ejemplo de una convención:

```
class MainController < ApplicationController
    def index
        # No es necesario especificar una vista, se buscará un
```

```
    end
end
```

Un framework orientado a convención suele ser más productivo (porque no hay que configurar tantos elementos), además suele encaminarte por las buenas prácticas que la comunidad recomienda, de manera que además de ayudarte a poner en prácticas las convenciones definidas, también te las enseña, porque sin ellas no podrás usar el framework.

Por otro lado, los frameworks de convenciones también tienen desventajas, como por ejemplo, que algunas partes del framework se comporten como una *caja negra*.

En software nos referimos a una caja negra, cuando un elemento, un método o una operación produce resultados sin que sepamos o entendamos cómo los produce. Cabe aclarar que mientras vamos iniciando nos vamos a encontrar con muchos puntos como éste, no debemos afligirnos al respecto, es normal que al principio no nos quede claro como funcionan algunos componentes de las tecnologías que usamos.

Estas cajas negras suelen ser puntos de problema para encontrar errores, o modificar el comportamiento del framework para que se adapte a nuestras necesidades, por lo que a largo plazo es mejor evitarlos, esto puede ser evitando frameworks por convención o haciendo un esfuerzo adicional por entender a fondo el framework.

Los frameworks de configuración sobre convención, tienen otras ventajas y desventajas. Por un lado, son más fáciles de comprender

porque todo lo que hace funcionar el framework está expuesto para configurarse o adaptarse. Además, son más fáciles de adaptar a las necesidades del proyecto en caso de que esto sea necesario.

Podemos decir, en general, que es más difícil encontrarse con cajas negras, por lo que también son frameworks más fáciles de *debuggear* para encontrar problemas, bugs o puntos de bajo rendimiento.

Por otro lado, los frameworks de configuración sobre convención suelen ser verbosos, es decir, que realizar una modificación pequeña puede involucrar la modificación de varios archivos, o que agregar un nuevo componente requiera de más configuraciones que en un framework de convención. Por otro lado, desarrolladores nuevos pueden enfrentarse con problemas innecesarios como definir la arquitectura de trabajo, convenciones para nombres de archivos y el código mismo, esto puede traducirse en que a largo plazo el proyecto termina sin tener orden o forma alguna.

RENDIMIENTO Y VELOCIDAD.

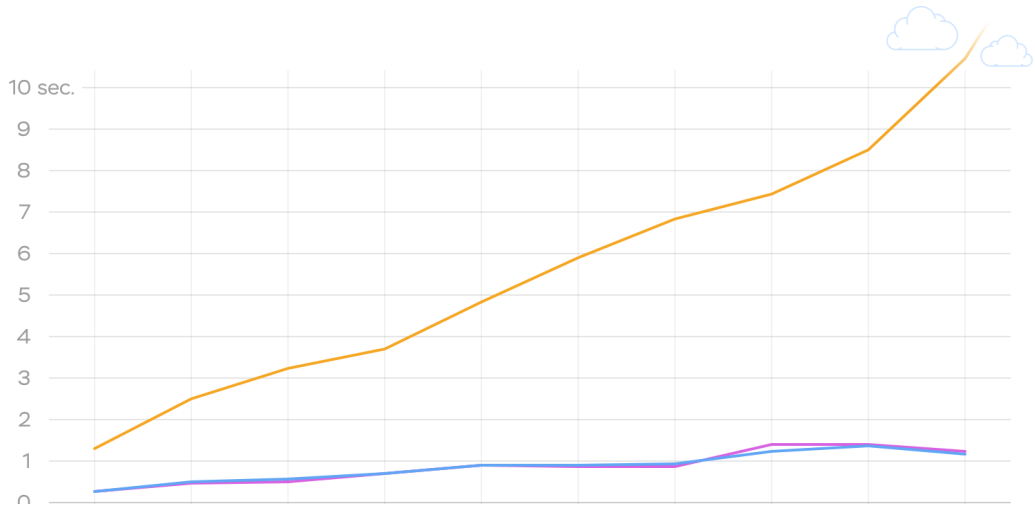
Uno de los aspectos más importantes al elegir un framework es qué tan rápido y eficiente es, este punto también se presta a muchos puntos de vista porque existen distintas métricas para medir el

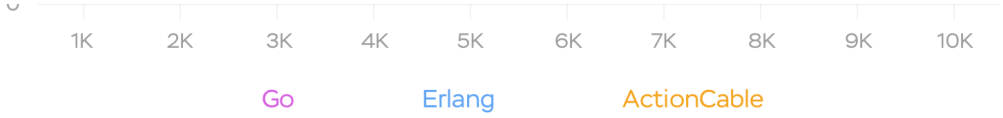
rendimiento de un framework.

Por un lado tenemos la velocidad de respuesta para archivos estáticos, la integración con la base de datos, uso del CPU, uso de la RAM y algo muy importante: Qué tanto se degradan estas métricas conforme más conexiones se realizan al framework.

Esta última consideración es importante porque ciertos frameworks pueden responder muy bien en las primeras peticiones, pero mientras más usuarios se conectan, las métricas se van degradando, es decir, entre más usuarios, más lento se vuelve. En ocasiones, esta relación no es lineal, y mientras más crecen las conexiones, los saltos en tiempo de respuesta o consumo de recursos son mucho más grandes.

Considera el siguiente ejemplo en el que se compara el rendimiento de ActionCable (WebSockets en Rails) con Go y Erlang, proveniente de la página https://evilmartians.com/chronicles/anycable-actioncable-on-steroids





Nota como la gráfica de tiempo de respuesta de ActionCable no es lineal, y conforme más usuarios se conectan el tiempo de respuesta crece más y más, muchas tecnologías tienen esta clase de problema de rendimiento, en algunas ocasiones en forma de un incremento en tiempo de respuesta, otros en más uso de memoria RAM y algunos más en consumo de CPU.

Dicho esto, hay que considerar que algunos de estos problemas se presentan con miles de conexiones concurrentes, y muchos proyectos nunca tienen esta clase de problemas de concurrencia, por lo que las medidas de rendimiento y velocidad deben considerarse de acuerdo al tipo de proyecto en el que estás trabajando, más adelante hablaré de eso.

LENGUAJE DE PROGRAMACIÓN.

Muchos de los frameworks se inspiran en otros para establecer convenciones, organización del proyecto, formas de configurarse, arquitecturas, etc. Eventualmente, esto se traduce en que algunos frameworks son muy parecidos entre sí, por lo que la diferencia se reduce al lenguaje de programación que se usa en cada uno.

Por ejemplo, solemos decir que para decidir si usar Django o Rails, tenemos que pensar solamente en si queremos usar Python o Ruby

como lenguajes de programación.

Hay factores importantes a considerar cuando definimos en qué lenguaje de programación queremos trabajar, algunos se prestan a la concurrencia, otros a la productividad, algunos más tienen el beneficio de ser expresivos o de tener una amplia biblioteca de librerías, una buena comunidad, buena documentación, etc.

Estos puntos deben considerarse de acuerdo al proyecto en el que estamos trabajando, por ejemplo, si se espera que se conecten muchos usuarios, quizás sea conveniente usar un framework que se preste a la programación asíncrona o concurrente, o bien que sea compilado, como Go, JAVA, Elixir, etc Por otro lado, si no se esperan millones de usuarios, quizás sea bueno darle prioridad a lenguajes expresivos y productivos como Ruby o Python.

Es muy importante recordar, que todos los lenguajes tiene casos de éxito en los que han sido utilizados por empresas enormes para manejar millones de peticiones, esto quiere decir que, sin importar la cantidad de usuarios que esperes, casi cualquier lenguaje puede dar soporte a dicha cantidad, si es usado apropiadamente.

Por otro lado, las empresas más grandes, aquellas con el alcance y uso de Facebook, Twitter, Google, Airbnb suelen tener stacks de tecnología que incluyen más de un lenguaje de programación, que se adapte a una tarea en específico, por lo que si tu aplicación algún día tendrá ese nivel de uso, debes esperar que en el camino hayan

modificaciones al lenguaje de programación usado.

RESUMEN

En mi experiencia podría decir que, en general, existen dos tipos de prioridades en un equipo de trabajo, rendimiento y productividad.

En una empresa grande como Amazon, Google o Facebook, la prioridad es el rendimiento y no la productividad, esto es porque mantienen aplicaciones que son usadas por miles de millones de usuarios, el desarrollo no tiene un presupuesto límite y los proyectos se trabajan en equipos de decenas de personas.

En una empresa pequeña como una Startup que inicia, una agencia de desarrollo, un proyecto personal, etc. La prioridad debe ser la productividad. Esto no quiere decir que el rendimiento no es importante, tal como la productividad no deja de ser importante en una empresa grande, pero, se pueden hacer sacrificios en rendimiento en pro de utilizar una tecnología orientada a convención, un lenguaje expresivo, etc.

Si consideramos los puntos que aquí toque, podría resumirlo de la siguiente manera:

Considera un lenguaje de programación expresivo y un framework de convención cuando tengas un equipo pequeño, deadlines

constantes o un presupuesto limitado. Mis recomendaciones favoritas son:

1. Ruby y Ruby on Rails.
2. Python y Django.
3. JavaScript y Algún framework de los existentes (Sails, Adonis, etc.)
4. PHP y Laravel.

Cuando el presupuesto no es límite, el equipo de trabajo puede ser amplio y especializado, o el rendimiento es crítico, considera usar un lenguaje pensado precisamente en rendimiento, concurrencia, tolerancia a fallos y quizás un framework orientado a configuración.

Quiero aclarar que todos los lenguajes suelen tener frameworks de convención y frameworks de configuración, por lo que algunas de mis recomendaciones a continuación son una combinación de lenguajes de programación eficientes y frameworks de convención:

1. Elixir y Phoenix.
2. Go y Algún framework web (Gorilla por ejemplo).
3. JAVA y Algún framework (usualmente son orientados a configuración).

CONCLUSIÓN

Estas son algunos puntos a considerar, sin embargo, existen muchos

otros puntos a considerar al elegir un framework, si esta guía no termina de resolver tus dudas, te recomiendo que comiences a explorar los frameworks por tu propia cuenta, considera realizar ejercicios simples y prácticos que te ayuden a definir si el framework es para ti o no.

Si consideras que en tu experiencia alguno de los puntos aquí mencionados pueden ser enriquecidos o mejorados, no dudes en colocarlo en los comentarios de este artículo, agradecería también leer tus sugerencias y opiniones sobre el post.

Otros artículos del blog

Qué es Inertia.js

3 Min.

Luis Fernando Garcia Perez

Realmente conoces los strings en Python?

4 Min.

Eduardo Ismael García Pérez

¿Cómo crear aplicaciones en tiempo real?

5 Min.

Eduardo Ismael García Pérez

Monitorear cambios en nuestros modelos con Ruby on Rails PT2

3 Min.

Eduardo Ismael García Pérez

Comentarios

Rigoberto Alcantar

10 Septiembre 18

Al leer esto es un gran alivio saber como piensan aquellos que ya han vivido en el mundo profesional, que saben de fallos y fortalezas de distintas tecnologias, como estudiante, siempre es necesario saber estos puntos cruciales, ya que con tanta informacion y distintas maneras de realizar un proyecto se vuelve algo muy tedioso y hasta estresante por la simple duda de: ¿Que tecnologia usare para este proyecto?

Responder

Diego Jesús Bolaños Suárez

01 Agosto 18

2

No sabía sobre estos conceptos sobre los frameworks, agradezco mucho te hayas tomado el tiempo de escribir este artículo.

Responder Ver respuestas (1)
