**Linguagens de programação**

**Resumo**

**Questão**

É necessário escolher a linguagem de programação para o sistema de front-end adequada para web e uma linguagem de back-end adequada para servidor.

**Decisão**

Estamos escolhendo o CSS para o front-end.

Estamos escolhendo PHP para o back-end.

**Status**

Decidido. Estamos abertos a novas alternativas à medida que surgem.

**Detalhes**

**Suposições**

Queremos criar software para web que sejam modernos, rápidos, confiáveis, responsivos etc.

Os sistemas de front-end deverão se

* Usuários e interações típicas
* Navegadores e sistemas típicos
* Desenvolvimentos e implantações típicas

Os aplicativos de front-end provavelmente evoluirão rapidamente:

* Queremos garantir desenvolvimentos, implantações, iterações rápidas e fáceis, etc.
* Valorizamos a comprovação, como a segurança de tipo, e estamos bem trabalhando um pouco mais para alcançá-la.

Os aplicativos de back-end são superiores ao típico:

* Metas acima do normal para qualidade, especialmente provabilidade, confiabilidade, segurança, etc.
* Metas acima do normal para quase tempo real, ou seja, não queremos pausas devido à coleta de lixo da máquina virtual.
* Objetivos acima do normal para programação funcional, especialmente para paralelização, processamento de vários núcleos e segurança de memória.

Aceitamos velocidades de tempo de compilação mais baixas em favor da segurança de tempo de compilação e velocidades de tempo de execução.

**Restrições**

Temos uma forte restrição de idiomas que podem ser usados ​​com os principais serviços de provedores de nuvem para funções, como Amazon Lambda.

**Posições**

Nós consideramos estes idiomas:

* C
* C ++
* Clojure
* Elixir
* Erlang
* Olmo
* Fluxo
* Vai
* Haskell
* Java
* JavaScript
* Kotlin
* Pitão
* Rubi
* Ferrugem
* TypeScript

**Argumento**

Resumo por idioma:

* C: rejeitado devido à baixa segurança; A ferrugem pode fazer quase tudo melhor.
* C ++: rejeitado porque é uma bagunça; O Rush pode fazer quase tudo melhor.
* Clojure: excelente modelagem; melhor aproximação Lisp; ótimo tempo de execução no JVM.
* Elixir: excelente tempo de execução, incluindo capacidade de implantação e simultaneidade; excelente experiência de desenvolvedor; ecossistema relativamente pequeno.
* Erlang: excelente tempo de execução, incluindo capacidade de implantação e simultaneidade; experiência desafiadora do desenvolvedor; ecossistema relativamente pequeno.
* Elm: parece muito promissor; A IBM está publicando importantes estudos de caso com bons resultados; ecossistema menor.
* Fluxo: melhoria interessante em relação ao JavaScript; Contudo; os desenvolvedores estão se afastando disso.
* Go: excelente experiência de desenvolvedor; excelente concorrência; mas um histórico de más decisões que prejudicam a linguagem.
* Haskell: melhor linguagem funcional; comunidade de desenvolvedores menor; não alcançou sucessos de produção publicados o suficiente.
* Java: excelente tempo de execução; excelente ecossistema; experiência de desenvolvedor abaixo da média.
* JavaScript: a linguagem mais popular de todos os tempos; ecossistema mais difundido.
* Kotlin: corrige muito do Java; excelente respaldo da JetBrains; bons casos publicados de portabilidade de Java para Kotlin.
* Python: linguagem mais popular para administração de sistemas; ótimas ferramentas analíticas; boas estruturas da web; mas abandonado pelo Google em favor do Go.
* Ruby: a melhor experiência de desenvolvedor de todos os tempos; melhores estruturas da web; comunidade mais agradável; mas muito lento; um pouco difícil de embalar.
* Ferrugem: melhor nova linguagem; ênfase na abstração zero; ênfase de simultaneidade; no entanto, ecossistema relativamente pequeno; e tem limites deliberados em alguns tipos de acelerações do compilador, por exemplo, o acesso direto à memória precisa ser explicitamente inseguro.
* TypeScript: adiciona tipos ao JavaScript; ótimo transpiler; crescente ênfase do desenvolvedor em portar de JavaScript para TypeScript; forte apoio da Microsoft.

Decidimos que as VMs têm um conjunto de compensações de que não precisamos no momento, como complexidade adicional que fornece recursos de tempo de execução.

Acreditamos que nossa decisão central é impulsionada por duas questões transversais:

* Para maior velocidade de tempo de execução e acesso ao sistema mais restrito, escolheríamos JavaScript e C.
* Para velocidade de tempo de execução quase mais rápida e acesso ao sistema mais estreito, escolhemos TypeScript e Rust.

Menções honrosas vão para as linguagens VM e frameworks da web que escolheríamos se quiséssemos um VM lanauge:

* Fechamento e luminoso
* Java e Spring
* Elixir e Phoenix

**Implicações**

Os desenvolvedores de front-end precisarão aprender TypeScript. Esta é provavelmente uma curva de aprendizado fácil se a experiência principal do desenvolvedor for usar JavaScript.

Os desenvolvedores de back-end precisarão aprender o Rust. Esta é provavelmente uma curva de aprendizado moderada se a experiência principal do desenvolvedor estiver usando C / C ++, e uma curva de aprendizado difícil se a experiência principal do desenvolvedor estiver usando Java, Python, Ruby ou linguagens gerenciadas por memória semelhantes.

TypeScript e Rust são relativamente novos. Isso significa que muitas ferramentas ainda não possuem documentação para esses idiomas. Por exemplo, o pipeline devops precisará ser configurado para esses idiomas e, até agora, nenhuma das ferramentas devops que estamos avaliando tem exemplos padrão para esses idiomas.

Os tempos de compilação para TypeScript e Rust são bem lentos. Parte disso pode ser devido à novidade das línguas. Podemos querer ver como mitigar tempos de compilação lentos, como por compilação sob demanda, simultaneidade de compilação, etc.

O suporte IDE para essas linguagens ainda não é onipresente e ainda não é de primeira classe. Por exemplo, a JetBrains vende o PyCharm IDE para suporte de primeira classe para Python, mas não vende um IDE com suporte de primeira classe para Rust; em vez disso, o JetBrains pode usar um plug-in Rust que fornece talvez 80% do suporte à linguagem Rust em relação ao suporte à linguagem Python.

**Relacionados**

**Decisões relacionadas**

Nosso objetivo é fazer escolhas de ecossistemas que se alinhem a esses idiomas.

Por exemplo, queremos escolher um IDE que tenha boas capacidades para essas linguagens.

Por exemplo, para nosso framework web front-end, temos mais probabilidade de decidir sobre um framework que tende a apontar para o TypeScript (por exemplo, Vue) do que um framework que tende a apontar para JavaScript simples (por exemplo, React).

**Requisitos relacionados**

Todo o nosso conjunto de ferramentas deve oferecer suporte a esses idiomas.

**Artefatos relacionados**

Esperamos que possamos exportar alguns segredos para variáveis ​​de ambiente.

**Princípios relacionados**

Meça duas vezes, construa uma vez. Estamos priorizando alguma segurança em relação a alguma velocidade.

O tempo de execução é mais valioso do que o tempo de compilação. Estamos priorizando o uso do cliente sobre o uso do desenvolvedor.