reservas.rs

server.rs

Teoría del Lenguaje {

[Programación en Rust]

< Integrantes: >

< Catarina Valdatta (110533),
Lucas Torres (97819),
Agustina Landi (107850) >

}

FIUBA - 1er cuatrimestre 2024.

reservas.rs

server.rs

Tabla de Contenidos {

}

reservas.rs

server.rs

01 {

[Origen]

< Motivación y sus usos >

}

reservas.rs

server.rs

¿Cuándo? < /1 > {

< Rust fue desarrollado por Mozilla Research, con su primera versión estable (1.0) lanzada en mayo de 2015. >

}

Motivación < /2 > {

< Resolver problemas de seguridad y gestión de memoria, sin sacrificar el rendimiento. >

}

reservas.rs

server.rs

¿Para qué sirve? < /1 > {

- Sistemas Embebidos y Programación de Sistemas.
- Aplicaciones Concurrentes.
- Aplicaciones de Alto Rendimiento.

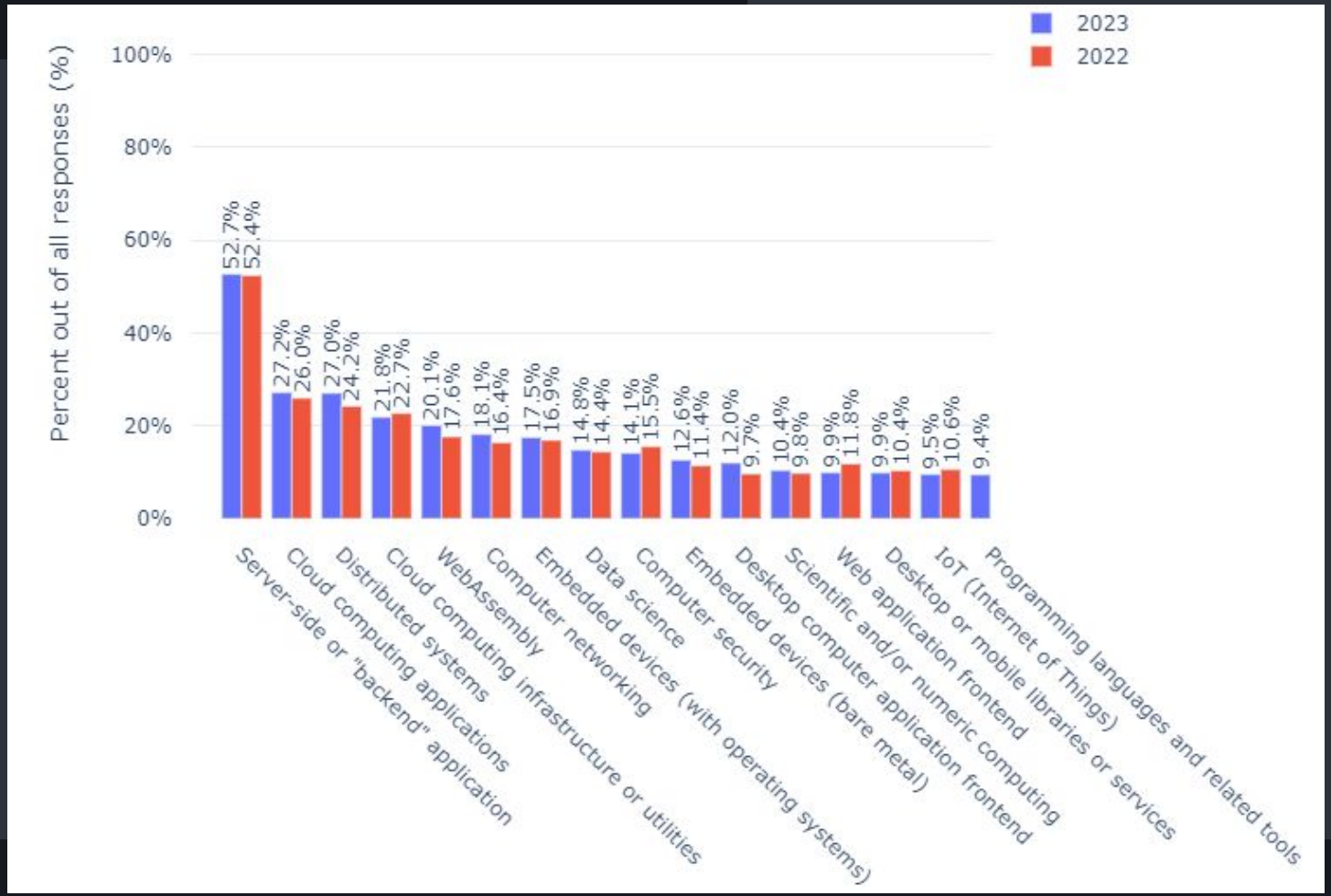
}

¿Para qué no? < /2 > {

- Desarrollo Rápido de Prototipos.
- Aplicaciones con Alta Interfaz Gráfica de Usuario (GUI).

}

< Encuesta anual realizada por Rust, acerca de sus usos >



reservas.rs

server.rs

02 {

[Características]

< Básicas y avanzadas >

}

reservas.rs

server.rs

Características Básicas {

01. Paradigmas

02. Compilado/ Interpretado

03. Tipado

04. Control de flujos

05. TDAs

06. Memoria

}

reservas.rs

server.rs

Paradigmas; {

Rust

< soporta programación imperativa, funcional y concurrente >

C/C++

< imperativo, orientado a objetos (en C++), funcional limitado y concurrente >

Oz

< combinación de paradigmas imperativo, declarativo (funcional y lógico) y concurrente >

}

reservas.rs

server.rs

Ejemplos; {

C/C++

```
std::thread t([](){
    while (running) {
        // Do something concurrently
    }
});
t.join();
```

Oz

```
thread
   {Wait Changed}
   ServerStopFlag := true
end
```

}

reservas.rs

server.rs

Compilado/Interpretado; {

Rust

< usa “rustc” para compilar el código >

C/C++

< C usa “gcc”, similar a C++ que utiliza gcc++ >

Oz

< es interpretado, utiliza el sistema Mozart >

}

reservas.rs

server.rs

Tipado; {

Rust

< estático y fuerte >

C/C++

< estático y fuerte >

Oz

< dinámico y fuerte >

```
declare
X = 10
{Browse X}
```

}

reservas.rs

server.rs

Control de flujos; {

Rust

< if, if let, else, match, loop, while y for >

C/C++

< if, else, switch, while, for, y do-while >

Oz

< if, case, for y while >

}

reservas.rs

server.rs

```
TDAs; {

}
```

Rust

< structs y enums >

C/C++

< C se maneja con structs, C++ con clases y enums >

Oz

< utiliza registros (record) >

reservas.rs server.rs

Manejo de memoria en Rust; {

‘Concepto de Ownership y Borrowing’

Ownership se refiere a la propiedad exclusiva de un valor. Solo un propietario puede modificar o liberar el recurso al que apunta.

Borrowing permite que varias referencias lean o una referencia modifique un valor sin tomar la propiedad. Mientras haya una referencia modificable, no puede haber referencias de solo lectura y viceversa.

}

reservas.rs

server.rs

Características Avanzadas {

01. Concurrencia

02. Errores

03. Lifetime

04. Generics

05. Threads

}

reservas.rs

server.rs

Manejo de concurrencia en Rust; {

‘Manejo de errores con Arc y Mutex’

Arc permite compartir datos entre múltiples threads de forma segura.

Mutex (Mutual Exclusion) permite proteger datos compartidos para garantizar que solo un thread acceda a los datos a la vez.

}

reservas.rs

server.rs

Manejo de concurrencia en Rust; {

‘Uso de Threads y Async’

Un thread es una secuencia de instrucciones que puede ser ejecutada de manera independiente dentro de un proceso.

La programación asíncrona es crucial para construir aplicaciones que pueden manejar muchas operaciones de entrada/salida (I/O) concurrentes sin bloquear el hilo de ejecución principal.

}

reservas.rs server.rs

Manejo de errores en Rust; {

‘Uso de Result y Option’

Result se utiliza para operaciones que pueden fallar. Tiene las variantes Ok(T) y Err(e).

Option se usa para valores que pueden estar o no presentes, sus variantes son Some(T) y None.

}

No incluidos en nuestro proyecto {

- **Lifetyme annotations** — Controlan cuánto tiempo vive una referencia para garantizar que no existan referencias colgantes.
- **Generics** — Permiten escribir funciones y tipos que pueden trabajar con cualquier tipo de datos.
- **Traits** — Proporcionan una forma de definir comportamiento compartido entre diferentes tipos.

}

reservas.rs

server.rs

```
03 {

    [Estadisticas y
     Casos de Estudio]

}
```

reservas.rs

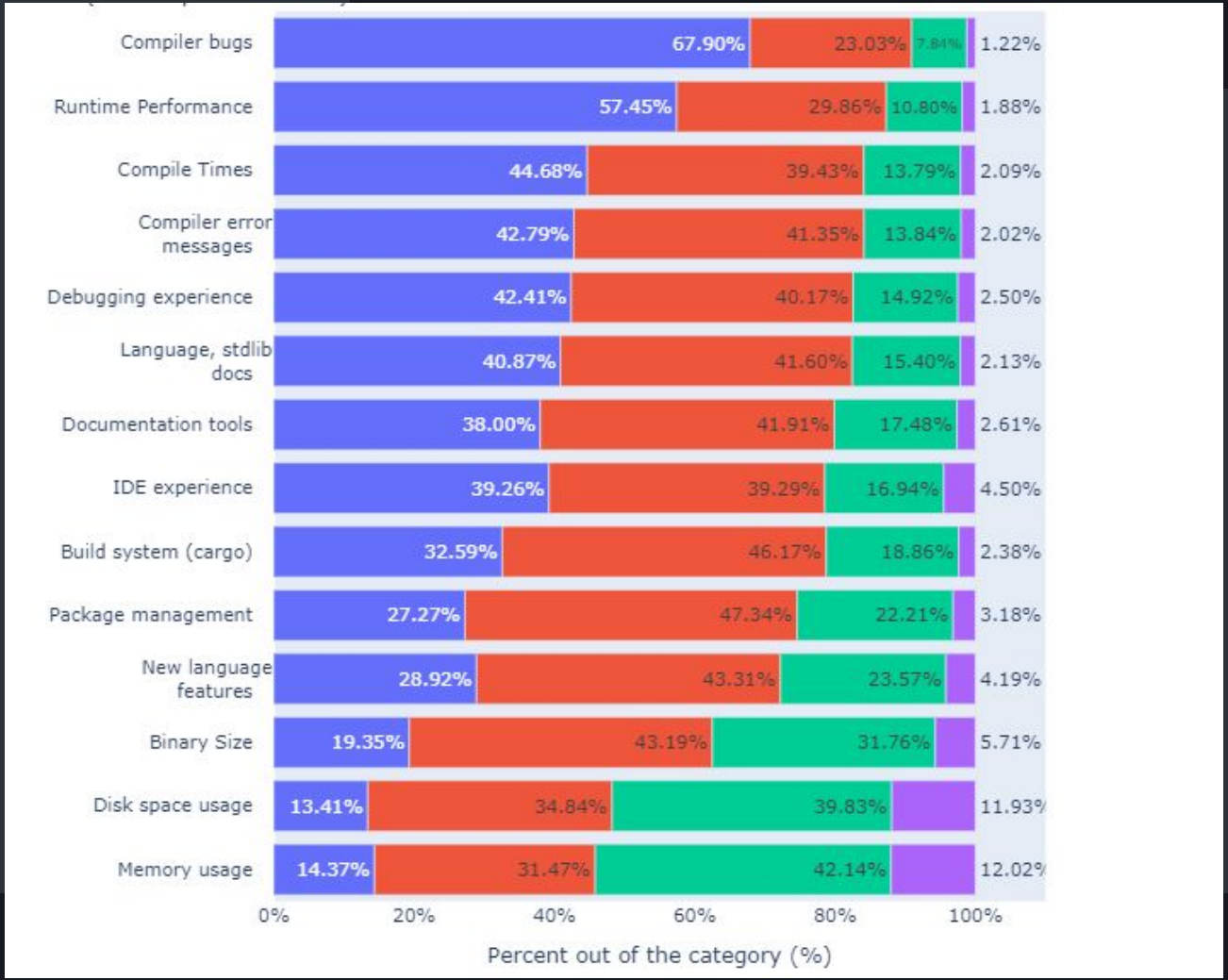
server.rs

Stack Overflow Survey; {

> ‘Lenguaje más admirado’

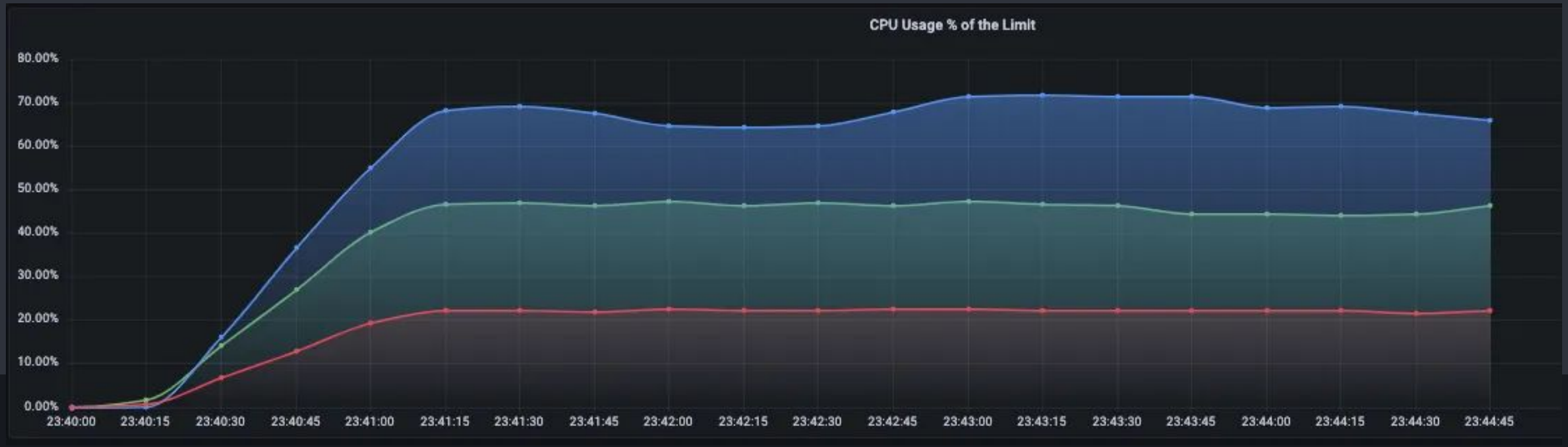
Rust Survey;

'Cosas a mejorar'

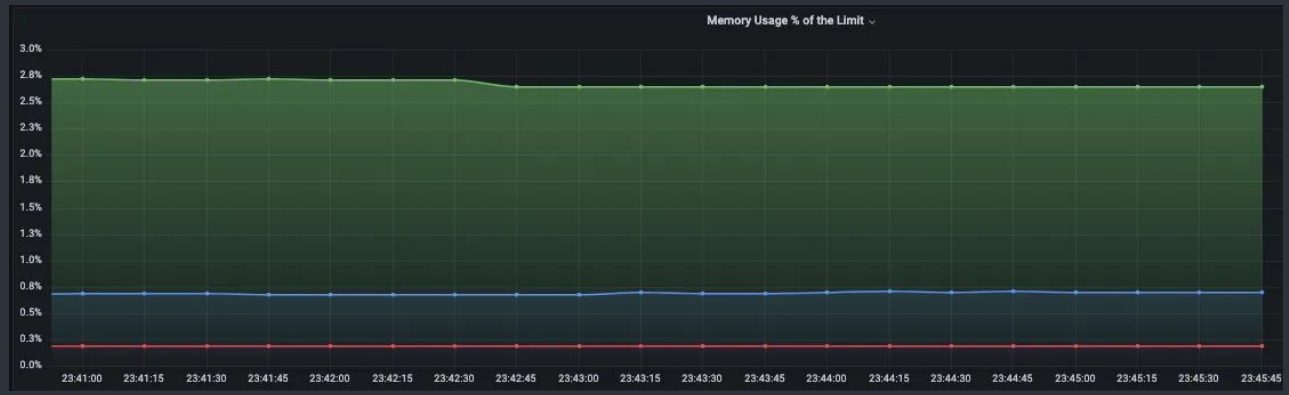


“Performance Comparison of Rust, C#, and Go for High-Performance Web APIs in Kubernetes Cluster”

< Go es el lenguaje que tiene más uso de CPU post load >



< C# es el lenguaje que tiene más uso de memoria post load >



< Rust es el lenguaje con menor latencia >

reservas.rs

server.rs

Casos de Estudio; {

> ‘Dropbox y Yelp’
>
> Varios componentes del sistema principal de almacenamiento de ficheros de Dropbox están escritos en Rust, como parte de un proyecto más amplio para aumentar la eficiencia de sus centros de datos.
>
> Yelp ha desarrollado un framework en Rust para tests A/B en tiempo real. Se usa en todos las páginas y apps de Yelp, para lanzar experimentos en áreas desde UX hasta infraestructura interna.

}

reservas.rs

server.rs

```
Fin {
  Gracias por
  escucharnos;
}
```