**Sistemas Distribuidos**

# Monica Scheduler

Laura Victoria Riera Pérez
Marié del Valle Reyes

Cuarto año. Ciencias de la Computación.
Facultad de Matemática y Computación, Universidad de La Habana, Cuba

1 de julio de 2023

## Repositorio del proyecto

https://github.com/computer-science-crows/monica-scheduler

## I. Definición inicial del problema

El tiempo es un recurso de vital importancia. Una de las estrategias más comunes para su gestión es el uso de una agenda, para la cual existen diversas opciones que abarcan desde el formato tradicional en papel hasta las versiones electrónicas modernas. Sin embargo, debido a la naturaleza intrínseca de las interacciones humanas, una agenda personal a menudo resulta insuficiente. En numerosas ocasiones, es imprescindible coordinarla con otras personas para llevar a cabo actividades conjuntas. Este proceso implica la identificación de horarios compartidos y la detección de intervalos de tiempo libres. Además, estas planificaciones pueden verse alteradas por eventos imprevistos a los que se debe asistir, lo que conlleva la necesidad de modificar la agenda nuevamente.

Este proyecto consiste en crear una agenda distribuida como herramienta de gestíon del tiempo para eventos personales o grupales. Los requisitos clave para este sistema son:

1. **Arquitectura Distribuida**: El sistema debe ser diseñado e implementado como un sistema distribuido. Esto significa que el sistema debe ser capaz de funcionar en múltiples máquinas mientras se presenta a los usuarios como un sistema coherente único.

2. **Mecanismo de Autenticación e Identificación**: El sistema debe ser capaz de autenticar e identificar a cada usuario. Esto asegura que solo los usuarios autorizados puedan acceder al sistema y realizar ciertas acciones.

3. **Formación Dinámica de Grupos**: El sistema debe permitir la creación dinámica de grupos, ya sean jerárquicos o no jerárquicos. Esto significa que los usuarios deben poder crear y gestionar grupos dentro del sistema de manera flexible.

4. **Citas Grupales**: El sistema debe admitir la creación de citas grupales. Si se utiliza un grupo jerárquico, una cita creada por un superior debe aparecer automáticamente en las agendas de todos los miembros del grupo. Para grupos no jerárquicos, todos los miembros deben aceptar la cita para que se confirme.

5. **Actualizaciones Automáticas de la Agenda**: Cuando se crea, modifica o elimina una cita, los cambios deben reflejarse automáticamente en las agendas de los usuarios relevantes.

6. **Identificación de Conflictos**: El sistema debe ser capaz de identificar conflictos en las agendas locales. Por ejemplo, si un usuario tiene programadas dos citas al mismo tiempo, el sistema debe señalar esto como un conflicto.

7. **Visualización de la Agenda**: Los usuarios deben poder ver las agendas de un grupo de personas durante un período de tiempo determinado. El sistema debe respetar la privacidad de la información del usuario y los niveles jerárquicos al mostrar esta información.

## II. Modelación del problema

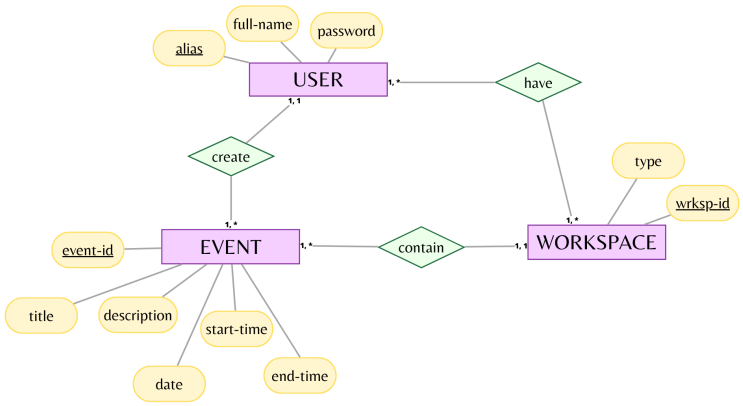


**Figura 1:** *text*

## III. Herramientas utilizadas

- docker
- MongoDB
- FastAPI
- React

## IV. Arquitectura: Microservicios

Microservices architecture, also known simply as microservices, is an architectural style that structures an application as a collection of small autonomous services, modeled around a business domain.

Services definition: Technology stack:

# V. DHT

## I. Kademlia

### i.1. Comunicación: RPCs

### i.2. Consistencia y Replicación: Active replication

Active replication, also known as state machine replication, is a strategy where each operation is performed on all replicas. This ensures that all replicas have the same state at any given time. Here's a basic outline of how it could work in your distributed agenda system:

1. **Operation Invocation**: When a client wants to perform an operation (like creating a new event in the agenda), it sends a request to a designated component in your system. This could be a load balancer, a leader node in a consensus protocol, or any other component that's responsible for coordinating requests.

2. **Broadcast to Replicas**: The coordinating component then broadcasts the operation to all replicas. This could be done through a variety of methods, such as multicast or a publish-subscribe system. The important part is that all replicas receive the operation.

3. **Perform Operation**: Each replica independently performs the operation. Since all replicas are identical and receive the same sequence of operations, they should all arrive at the same state. For example, if the operation was to create a new event, each replica would add that event to its copy of the agenda.

4. **Response to Client**: After performing the operation, each replica sends a response back to the coordinating component. Once the coordinator has received responses from all replicas (or a sufficient number of them), it sends a final response back to the client.

5. **Error Handling**: If any replica fails to perform the operation, the system needs to handle this error. This could involve retrying the operation, removing the faulty replica, or other error recovery strategies.

Remember, the key to active replication is ensuring that all replicas perform every operation in the same order. This often requires careful coordination and can introduce significant complexity into your system. But when done correctly, it can provide a high degree of reliability and fault tolerance.

### i.3. Tolerancia a Fallas

# VI. Seguridad

Security in distributed systems is concerned with the prevention and detection of unauthorized actions by users of a system. The security of a distributed system can be divided into two main parts: secure channels and secure processes. Secure channels are used to protect data from being intercepted or tampered with during transmission. Secure processes are used to ensure that only authorized users can access and manipulate data.

## I. Autenticación

## II. Autorización

# Referencias