Se propone una arquitectura basada en microservicios. Esta arquitectura permite la creación de servicios independientes y escalables, lo que facilita la integración de nuevas funcionalidades y la adaptación a los cambios en los requerimientos.



La arquitectura se compone de los siguientes componentes:

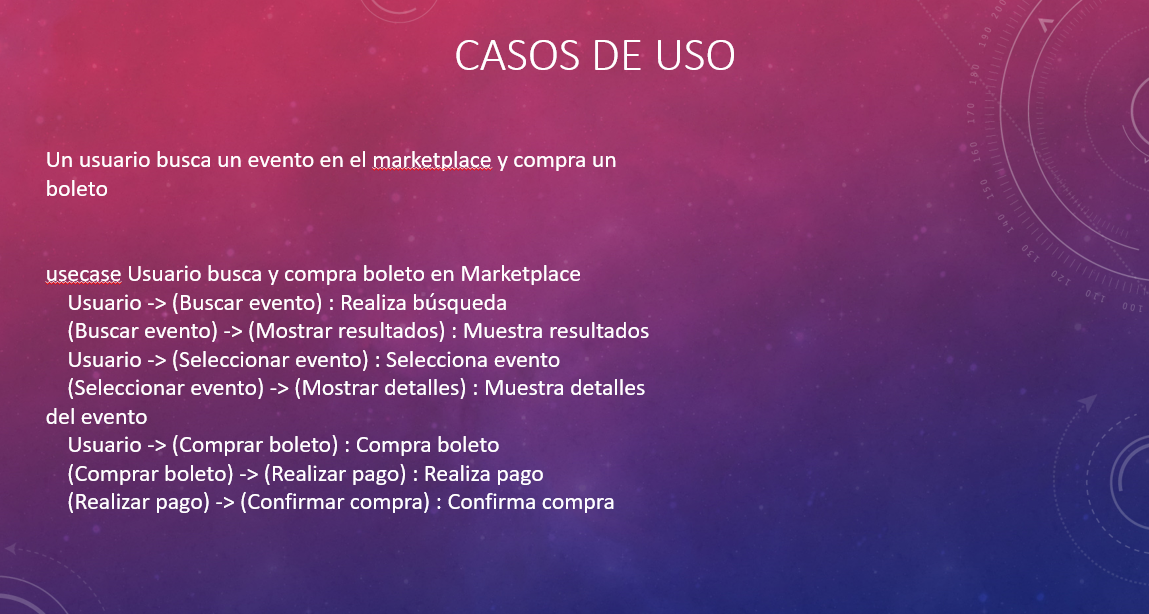
**Frontend**: es el componente que se encarga de la interfaz de usuario y la comunicación con los microservicios. Se puede Angular para su implementación.

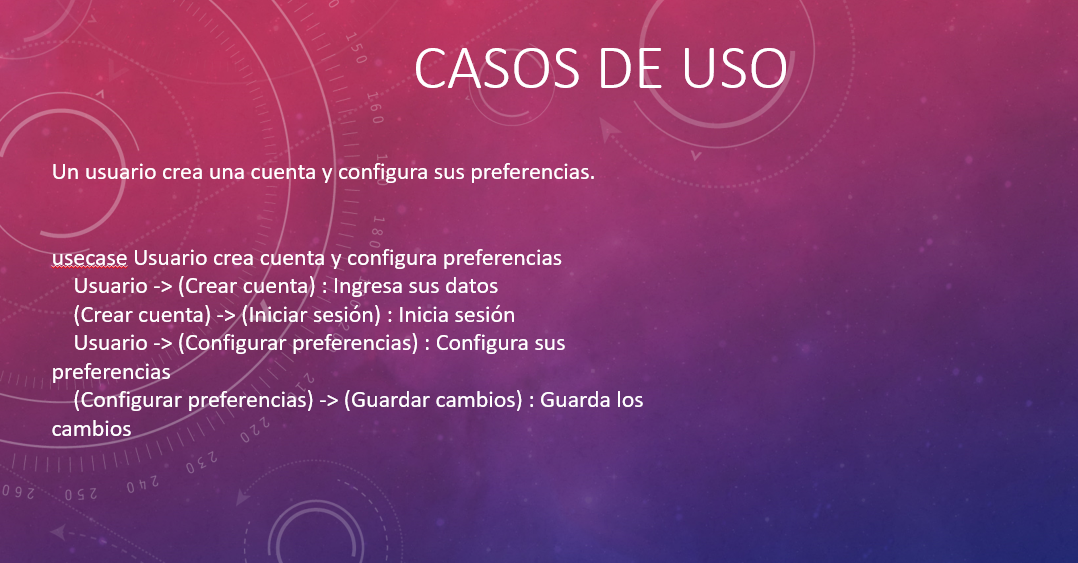
**API Gateway**: es el componente que se encarga de enrutar las solicitudes del cliente a los microservicios correspondientes. Se puede utilizar un servicio como Amazon API Gateway o Azure API Management para su implementación.

**Microservicios**: son los componentes que se encargan de proveer las funcionalidades del sistema. Se pueden implementar diferentes microservicios en NodeJ para cada funcionalidad, como por ejemplo: autenticación, gestión de eventos, gestión de usuarios, gestión de pagos, etc. Cada microservicio se comunica con una base de datos propia y se expone a través de una API RESTful.

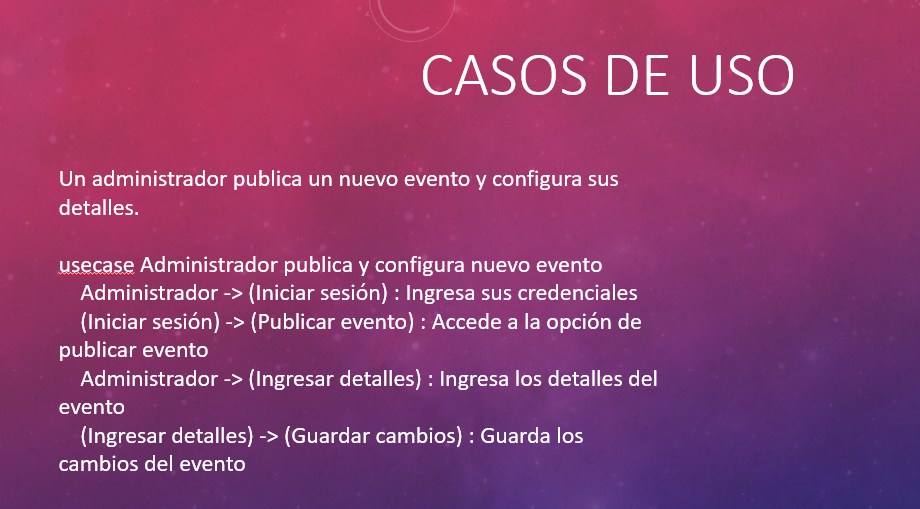
**Bases de datos**: cada microservicio utiliza su propia base de datos para almacenar la información correspondiente. Se pueden utilizar diferentes tipos de bases de datos, dependiendo de las necesidades específicas de cada microservicio.

**Balanceador de carga**: es el componente que se encarga de distribuir la carga entre los diferentes microservicios. Se puede utilizar un servicio como Amazon Elastic Load Balancer o Azure Load Balancer para su implementación.









Algunas características sistémicas que se pueden considerar son:

**Escalabilidad:** el sistema debe ser capaz de manejar un alto volumen de solicitudes y adaptarse a los cambios en la demanda.

**Disponibilidad**: el sistema debe estar disponible en todo momento, con una tasa mínima de errores y tiempos de respuesta rápidos.

**Seguridad:** el sistema debe proteger la información del usuario y garantizar la privacidad de los datos.

**Mantenibilidad**: el sistema debe ser fácilmente mantenible y actualizable, con una buena documentación y una arquitectura modular.

Algunos escenarios de calidad que se pueden considerar son:

**Rendimiento**: el sistema debe ser capaz de manejar un alto volumen de solicitudes sin afectar su rendimiento.

**Disponibilidad**: el sistema debe estar disponible en todo momento, con una tasa mínima de errores y tiempos de respuesta rápidos.

**Seguridad**: el sistema debe proteger la información del usuario y garantizar la privacidad de los datos.

**Usabilidad**: el sistema debe ser fácil de usar y accesible para todos los usuarios.