Proyecto final

Materia:	Sistemas Distribuidos
Término:	2018 II
Profesor:	Cristina Abad
Número de integrantes:	1 a 3 (restricción; estudiantes que repiten materia deben hacer proyecto de manera independiente)
Tema:	Arquitectura de micro-servicios para aplicación Web
Fecha entrega propuesta diseño:	Próximo martes (martes 18 de dic, 2am, vía SidWeb)
Fecha entrega proyecto:	Jueves 24 de enero, 8h00 (en mi oficina y vía SidWeb); a partir de las 8am puedo llamar a cualquier grupo a que presente su proyecto. Si un estudiante no está presente, tendrá automáticamente 0 en el proyecto. No atenderé a grupos que lleguen tarde.
Sistema operativo:	Linux
Lenguaje de prog.:	Java, C/C++, Clojure, Go, C#, Rust; **importante:** solamente acepto uso de otros lenguajes bajo pre-aprobación mía; la penalidad por usar un lenguaje no pre-aprobado es de -5 pts. javascript se permite solamente en el frontend; no en el backend.

Descripción

En el proyecto, deberán implementar un sistema web, con una arquitectura de microservicios, que sea capaz de cumplir los requisitos no funcionales planteados.

Meta

Diseñar, implementar parcialmente y evaluar un sistema de venta de boletos/tickets en línea.

Detalles

La empresa BoletosEcuador desea incursionar en el negocio de venta de tickets para eventos en línea. Cuando saque su producto al mercado ecuatoriano, desea desplazar a TicketShow, por lo que debe diseñar un sistema con mejor funcionalidad y escalabilidad que el sistema de TicketShow. Se ha contratado a un equipo de desarrollo Web y móvil quién desarrollará el frontend, y a Ud. lo han contratado para que diseñe y desarrolle el backend del sistema.

Requerimientos funcionales

IMPORTANTE: No es necesario que ustedes implementen al 100% los requerimientos funcionales del sistema; con un prototipo mínimo que permita probar la compra de boletos es suficiente. Se

supone que el frontend lo implementa otro equipo; sin embargo, si no me proporcionan un frontend sencillito (aunque esté feo e incompleto), no podríamos ver el backend funcionando.

- Un administrador debe poder crear “venues” (teatros, estadios, etc.), incluyendo información de los asientos clasificados por categoría (tribuna, palco, etc.). ← por ejemplo, esto es requerimiento del sistema completo y por lo tanto lo deben considerar en la BD, pero no necesitan implementar la interfaz para esta parte; pueden ingresar algunos registros manualmente y luego esos los usan para el resto del sistema.
- Un administrador debe poder crear eventos (ejemplo, concierto de X en teatro Y, el día Z) ← aplica el mismo comentario anterior
- Un usuario debe poder crear una cuenta. ← aplica el mismo comentario anterior
- Los usuarios deben poder consultar eventos disponibles, aún si no tienen cuenta o no se han autenticado con el sistema. → esto sí lo debo poder ver en el proyecto implementado
- Los usuarios deben poder comprar boletos. → también lo debo poder implementado
- No se debe permitir que dos usuarios compren el mismo boleto → también lo debo poder implementado.
- El sistema debe soportar clientes Web (con diferentes browsers) y clientes móviles (con diferentes sistemas operativos) → no lo validaré/revisaré.
- El sistema debe presentar varios reportes de ventas de tickets por evento/mes/fecha/localidad/etc. a los administradores → no lo validaré/revisaré.

Requerimientos no funcionales

- Capacidad:
    - El sistema debe poder soportar hasta 8 millones de usuarios (no todos activos simultáneamente).
    - El sistema debe ser capaz de expandirse para soportar hasta 300 millones de usuarios (no todos activos simultáneamente). ← Esto es, debido a que la empresa espera expandirse a otros países o regiones en un futuro.
- Latencia: El 99% de los requerimientos del frontend web debe tomar menos de 2 segundos.
- Throughput: El sistema debe poder atender hasta 1 millón de usuarios activos durante una hora. Sin embargo, se espera que en una hora pico promedio se atienda a máximo 500 mil usuarios y en usa hora baja promedio se atienda a máximo 10 mil usuarios.
- Costos: En caso de haber varias arquitecturas que cumplan con los requerimientos no funcionales del sistema, se debe preferir aquella que implique los costos más bajos para la empresa.
- Disponibilidad: Para cada componente/microservicio del sistema, este servicio debe continuar funcionando, aún si falla la máquina en donde está corriendo. El número de fallos que debe soportar cada servicio es 1 (es decir, si fallan dos máquinas del servicio, es posible que ya no esté disponible).
- Disponibilidad: El sistema debe poder recuperarse de fallos en los componentes.
- Seguridad:

    - No se debe permitir ningún uso no específicamente autorizado por los requerimientos funcionales.
    - No se debe poder modificar los queries a la BD con manipulaciones del lado de la interfaz del cliente.
- Mantenibilidad: Debe ser fácil implementar funcionalidades futuras en el sistema.

Otros (requerimientos de la materia)

- Deben utilizar Github. Enviarme el enlace al repositorio vía SidWeb, junto con la entrega del diseño del proyecto. **TODOS los miembros del grupo deben tener más de un commit en el repositorio; el primer commit de cada miembro debe tener una fecha máxima de Dic 20 de 2018. Necesito poder saber en qué trabajó cada uno.**
- Deberán entregar un documento final (2 a 6 páginas, no más) en el que documenten el problema y solución propuesta y decisiones de diseño, así como resultados de los experimentos de rendimiento (latencia y throughput).
    1. Para las pruebas de rendimiento, deben generar pedidos/requerimientos con alguna herramienta. Esto puede ser alguna herramienta que descarguen de Internet, o ustedes mismos programar algo en Python o Bash.
    2. La idea es que realicen pruebas que permitan observar cómo se cumple con los requerimientos no funcionales relacionados a rendimiento (latencia y throughput).
    3. Para cada uno de los dos tipos de experimentos, deben realizar al menos 10 pruebas, y mostrar los resultados usando un box plot o con un CDF.
- Deployment en la nube: Puede ser en Azure o AWS. AWS otorga créditos gratuitos para estudiantes. Alternativamente, búsquenme en mi oficina y les puedo dar credenciales de acceso a AWS, usando unos créditos que tengo yo. Restricción: Para la implementación solamente pueden usar recursos de cómputo del proveedor de la nube (ej.: AWS EC2, AWS Lambda, AWS Fargate, AWS ECS); no pueden usar bases de datos administradas por el proveedor como DynamoDB o Aurora. Sin embargo, sí es imposible incluir componentes de este tipo en el Diseño. **IMPORTANTE:** ESPOL tiene bloqueado el acceso al puerto 22, necesario para hacer SSH a instancias EC2. Esta parte la deben hacer desde su casa. El demo lo podrán mostrar sin problema, ya que ustedes se conectarán al servicio vía HTTP, lo cual no está bloqueado.

Detalle de las calificaciones

Será calificado sobre 35. La nota irá a calificación de proyecto final (2do parcial) **Y** del examen de mejoramiento. Para mejoramiento, es posible presentar un proyecto mejorado, en caso de que así lo deseen. La fecha máxima de entrega para el mejoramiento será el viernes de la semana de exámenes de mejoramiento, entre 8 y 9am, en mi oficina.

- Diseño: 10 puntos: Personal, NO en grupo.
    - Subir un PDF que describa la arquitectura propuesta. El documento debe tener las siguientes secciones (todas son obligatorias y todas serán calificadas).:
        - Diagrama de la arquitectura propuesta + descripción (1-2 párrafos) de la arquitectura propuesta.
        - Diagrama entidad-relación de la base de datos.
        - Descripción de las máquinas virtuales u otros componentes de cómputo que se usarán en el sistema (ej.: una VM con 4 cores y 32GB de memoria para el servidor web, una VM con 4 cores y 128GB de memoria para la BD, etc..).
        - Descripción de los productos de SW a ser usados (ej.: proxies, cachés, servidores web, BD, administradores de recursos, middlewares, librerías más importantes, etc.).
        - Descripción de cada uno de los componentes/microservicios, incluyendo detalle del lenguaje de programación a ser usado.
        - Descripción de los APIs que se usarán para comunicar dos componentes; esto debe proporcionarse para todos los pares de componentes que se comunican: Ejs.: El microservicio X se comunicará con la BD usando JDBC; el cliente web (browser) se comunicará con el proxy web (Nginx) usando un API REST; el componente X se comunicará con el componente Y usando un middleware pub-sub.
        - Detalle de costos: Se debe hacer un presupuesto anual de lo que costará ejecutar este sistema en un proveedor de nube pública.
        - Lista de recursos Web consultados para elaborar el diseño.
- Documento final: 0 puntos, pero no entregar este documento implica una penalidad de -5 puntos.
    - Sección de metodología describe y justifica correctamente las decisiones de diseño del proyecto (¿Estructuras de datos usados? ¿Librerías o middlewares utilizados? ¿Manejo de errores? ¿Mecanismos de sincronización usados? ¿Cómo realizaron las pruebas de rendimiento? ¿Qué nube usaron? ¿Otras cosas importantes?)
- Implementación: 25 puntos
    - Microservicios correctamente implementados: 5 puntos
    - Acceso correcto a datos (BD/cachés/etc): 5 puntos
    - Uso adecuado de mecanismos de comunicación entre procesos: 5 puntos
    - README que explique lo necesario para compilar/instalar/ejecutar el SW: 1 punto
    - Deployment correcto en la nube: 4 puntos.

    - Pruebas de rendimiento correctamente realizadas y correctamente documentadas (incluye el uso de una herramienta de benchmarking, o la implementación de scripts de prueba): 5 puntos
    - EXTRAS: Hasta 5 puntos, dependiendo de lo implementado.
- Participación en el proyecto: 10 puntos. Esta participación la evaluaré de dos maneras: (1) en la sustentación del proyecto, y (2) vía el log de commits del estudiante en el repositorio del proyecto. Si un estudiante no asiste a la sustentación o no tiene commits en el repositorio Git del proyecto immediatamente tendrá cero (0) en la nota del proyecto.
    - Estudiante demuestra haber participado activamente en el desarrollo del proyecto y demuestra entender su implementación y resultados: 10 puntos
    - Estudiante demuestra haber participado parcialmente el desarrollo del proyecto y demuestra entender parcialmente su implementación y resultados: 7.5 puntos
    - Estudiante demuestra participación limitada en el proyecto y/o no logra comprender los resultados obtenidos: 5 puntos
    - Estudiante demuestra una colaboración mínima en el desarrollo del proyecto: 2.5 puntos
    - Estudiante no colaboró en el desarrollo del proyecto: 0 puntos
- Nota Final: (Nota implementación) * (Nota participación en proyecto)/10 + Nota del diseño

IMPORTANTE: El diseño vale 10 puntos, por lo que se espera un diseño bien hecho. No se preocupen de que si proponen algo muy complejo luego les costará más implementarlo. Los diseños complejos que cumplan con todos los requerimientos del proyecto podrán ser simplificados al momento de implementarlos. Es decir, para la implementación del proyecto, yo les indicaré si algunos de los componentes no los deben implementar al 100%.