Universidad Nacional de Colombia

Facultad de Ingenieria

Documento de descripion de arquitectura
ADD

SEVERE TRIP PROJECT

M A T E R I A:

Arquitectura de Software 2019-2

P R E S E N T A :

Daniel Alfonso Pinzon Chavarria
Alex Jose Alberto Barreto Cajicá
Katherine Viviana Sierra Briñez
Alberto Nicolai Romero Martinez
Julian David Rodriguez Ruiz
Verbo Sebastian Camacho Silva

TUTOR

Henry Roberto Umaña Acosta
Juan Camilo Rodriguez Puentes

Ciudad Universitaria, Bogota D.C, 2019

Índice

1. Introducción

1.1. Nombre del sistema de software

SevereTrip

1.2. Descripción

Plataforma web para la promoción y venta de servicios de hotelería,restaurantes, viajes y alquiler de autos.

2. Requisitos Funcionales (RF)

2.1. Definición y descripción de requisitos funcionales a partir de historias de usuario

En primera instancia en clase se subió la funcionalidad pedida por el product owner, en donde se encasillaron distintas funcionalidades tales como:

1. publicar/ver anuncios de:
   a) Hoteles.
   b) Alquiler de autos.
   c) Restaurantes.
   d) Vuelos.
2. Perfil de usuario cliente/proveedor.
3. Sistema de mensajería de usuarios.
4. Sistema de reservas.
5. Listas de favoritos.
6. filtros dinámicos para búsquedas de servicios.
7. sistema de facturación y cobros.

2.2. Creacion/modificacion del backlog del sprint

Luego de analizar la funcionalidad suministrada, planeamos una lista de historias de usuario en donde evaluaremos la importancia de estas para la creación y desarrollo de las tareas a programar. De esto sacamos el siguiente cuadro:

id	Responsable	Tipo de usuario	Solución	Descripción
t1	Julián	-	Escalamiento de bases de datos	escalamiento de las bases de datos como maestras o esclavas
t2	Katherine, Julián	-	Prueba de rendimiento **despues** de escalar	Pruebas de rendimiento con instancias multiples por componente, con escalamiento horizontal.
t3	Alex	-	Busqueda de solucion al bad request en reservation	Solucion a problema de conexion con el microservicio reservation
f1	Nicolai, Sebastián, Alex	Cliente Proveedor	Correcciones y adiciones visuales de la aplicación.	Adición de características, corrección de bugs, cambios de UI.
a1	Daniel	-	Documentación del proyecto y de la metodología de desarrollo.	Documento de Descripción de Arquitectura, Actividades de SCRUM.

En este cuadro también evidenciamos un código para cada una de las historias de usuario, el tipo de cliente, la funcionalidad y las tareas importantes.

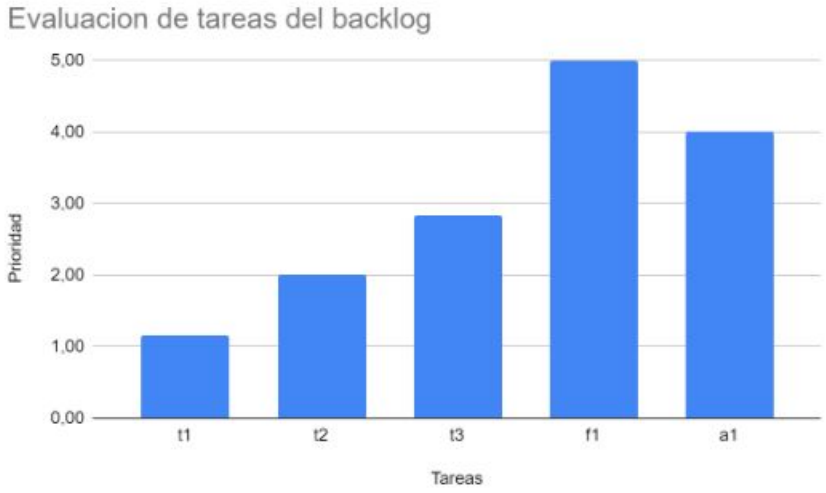
Luego vamos a evaluar la importancia y la dificultad de cada una de las tareas con el fin de tener claro cuál es la funcional mas optima a desarrollar. De esta manera y después de evaluar tenemos los siguientes datos:

1. Daniel (scrum master).
2. Sebastian.
3. Alex.
4. Nicolai.
5. Julian.
6. katherine (tester).

El resultado del analisis se miestra en la siguiente tabla:

id	1	2	3	4	5	6	total
t1	1	1	2	1	1	1	1,17
t2	2	2	1	3	2	2	2,00
t3	3	3	3	2	3	3	2,83
f1	5	5	5	5	5	5	5
a1	4	4	4	4	4	4	4

De este modo tenemos cada una de las historias de usuario evaluadas y así podemos atender a cual va a ser el odren a desarrollar. También cabe mencionar que la participación de cada uno de los estudiantes fue dispuesta de la siguiente manera:



A nivel visual aquí se muestran las prioridades que se van a tener con respecto a las historias de usuario.

Para finalizar se identificaron los microservicios a desarrollar y se asoció cada una de las funcionalidades descritas en las historias de usuario con cada uno de los microservicios. De esto obtuvimos el siguiente recuadro:

usuario			reservación
cliente + ˙proveedor + favorites	**post + comentarios**	**mensajes**	**reservación + pagos**
c1* - auth cliente	cp1 - Petición a publicaciones	c4 - Formulario de mensaje 1 a 1	cp3- Guardar reserva. (id de cliente, y de publicación)
c1- petición del perfil a cliente	cp2* - Petición a publicaciones (con filtros)		cp5 Obtener reservas (De proveedor)
c2- actualización de perfil cliente	p1- Formulario anuncio - (evaluar, retornar)		cp5- Contestar reservas. (proveedor)
cp2 - acceder a perfil de proveedor	c3- Formulario de comentario y calificación.		cp5- Guardar reservas.
p1* - Autenticación de proveedor	cp3- Petición a publicación a reservar.		cp4 Obtener reservas (De cliente)

Una salvedad de este recuadro es que las tareas en cuyo código aparece un asterisco, significa que esta tarea ha sido heredada de varias historias de usuario pero se le asignó el código de la historia más prioritario.

3. Requisitos No Funcionales (RNF)

3.1. Sprint 1

1. Lenguajes y frameworks:

   Para este primer Sprint vamos a plantear la información de los lenguajes y frameworks a utilizar dentro del desarrollo del Backend.
   En primera instancia vamos a desarrollar en Spring Boot, un Framework orientado a microservicios y ajustado para trabajar en metodologías ágiles, este está desarrollado en java y es heredero del antiguo framework Spring Framework pero con un repositorio muchisimo mas robusto y una detección de errores y autocompletado mas optimo.
   Este maneja un tomcat embebido para pruebas a nivel local en donde se puede probar tanto en un navegador como en un software de peticiones. Además se usará un visualizador de ApiRESTfull llamado Swagger para complementar la documentación tanto el dia de la presentación grupal como en la exposición de la documentación del Scrum Master el dia que sea pertinente.
   Para las bases de datos implementadas en los distintos microservicios, se pensó en incluir mongodb y mysql para cumplir con la funcionalidad planteada en la clase. Dependiendo la robustez del micro servicio se usará una o la otra, es decir, si un microservicio es muy robusto y necesita mayor interoperabilidad y almacenamiento de datos de diferentes entidades, se utilizará mysql; En otros casos se usará mongodb.
   Para la parte de probar peticiones tendremos en cuenta la herramienta conocida como postman, esta tiene un gran despliegue a nivel local y es muy intuitiva al momento de solicitar una consulta a los microservicios, entonces nos ayudara bastante en la revisión de las distintas consultas.

2. Estilo y/o patrones de arquitectura:

   Existen varios patrones de arquitectura identificables dentro del proyecto, pero ante la limitante de no tener un gateway o un front que haga las peticiones directamente, hay pocos estilos de diseño a implementar.
   Nosotros vamos a implementar varios patrones de diseño orientados a los microservicios que definimos dentro de nuestra funcionalidad. En primera instancia vamos a usar un patrón de capas el cual nos ayudará a acceder a los servicios más complejos de forma escalonada, es decir en un solo sentido, aunque este no muestra la funcionalidad interna ya que se encapsulan los microservicios dependiendo la capa a utilizar. Esta capa se puede ver mejor en el siguiente item de Diseño arquitectonico y descripcion de vista.
   Además de esto, para observar la funcionalidad entre microservicios, se podría

implementar un MVC en donde el postman funcionaria como la vista y los microservicios serían el controlador, pero como tal no existiría una vista aparte, por ello hemos decidido incluir un patrón de peer to peer relacionado con la comunicación entre microservicios ya que ambos actúan tanto como cliente y servidor.

3. Plataforma (contenedores)

   Para el despliegue desde el principio está estipulado la utilización de docker como un creador de imágenes en donde se pueda ejecutar tanto Spring boot como los distintos lenguajes orientados a bases de datos (mysql y mongodb). También esto irá ligado a la conexión con docker para tener un despliegue en cloud.
   Hay una salvedad al usar AWS y es que al momento de utilizar las herramientas de cloud en el proyecto, Spring boot tiene un repositorio considerable de varios servicios de Amazon para usarlos, estos incluyen cosas como bases de datos y eureka para control del balanceo de cargas de cada microservicio.

3.2. Sprint 2

1. Lenguajes y frameworks

   Adicionalmente ahora hacemos la inclusión de de un lenguaje para Front-End, este es React, este framework desarrollado desde facobook está enfocado para hacer más fácil el diseño y la conexión a los diferentes servicios, este tiene extensiones con conectores como fetch, axios (que fue el que usamos), apollo, etc.
   También hicimos un API Gateway enfocado a microservicios desarrollados en spring boot usando Spring Cloud, este podría decirse que es un complemento o dependencia de este framework y tiene su propio despliegue.

2. Estilos de arquitectura

   Para este sprint se insertó un modelo MVC ya que está claro que desde las vistas se va a acceder a los distintos servicios de los nodos desplegados, esta información será vista, modificada y hasta eliminada por el usuario (teniendo en cuenta que aún no hemos implementado seguridad).
   También es claro que hacia el Back-End se está implementando un estilo de peticiones tipo REST para solicitar o enviar información a los controladores.

3. Plataforma (contenedores)

   El despliegue del Front-End y el API Gateway, al igual que los demás nodos, se realizó en AWS, se puede visualizar su despliegue en la plataforma rancher, además como se mencionó anteriormente, se redujo la cantidad de microservicios desplegados.

3.3. Sprint 3

1. Lenguajes y frameworks

   Nuevamente, se realizaron los servicios de seguridad y manejo de sesión desde el framework Spring Boot para tener coherencia y fácil acoplamiento al microservicio que se está puntualizando (User), ya que este es el que tiene estas opciones de seguridad.
   Además de esto, desde ReactJS se realizaron las vistas del login desde el sprint pasado y por ende solo se modificaron los formularios para que se admitieran almacenamiento de información de usuario y credenciales incrustadas.

2. Estilos de arquitectura

   Prevalecen los estilos mencionados anteriormente y no hay adiciones a esto ya que la parte visual se está trabajando exactamente igual que en el sprint pasado, MVC.

3. Plataforma (contenedores)

   Aunque hubo una reducción importante de contenedores, el despliegue se sigue haciendo de la misma manera que en sprints pasados (AWS, Docker, Rancher).

3.4. Sprint 4

1. Lenguajes y frameworks

   Para el escalamiento horizontal se requiere trabajar con Docker y Rancher.

2. Estilos de arquitectura

   Prevalecen los estilos mencionados anteriormente y no hay adiciones a esto ya que la parte visual se está trabajando exactamente igual que en el sprint pasado, MVC.

3. Plataforma (contenedores)

   Como se mencionó anteriormente se trabaja con Docker y Rancher, no se tienen cambios sustanciales para esta entrega y se plantea no hacer más cambios en la arquitectura (al menos en el número de nodos)

3.5. Sprint 5

1. Lenguajes y frameworks

   Para el escalamiento horizontal se requiere trabajar con Docker y Rancher.

2. Estilos de arquitectura

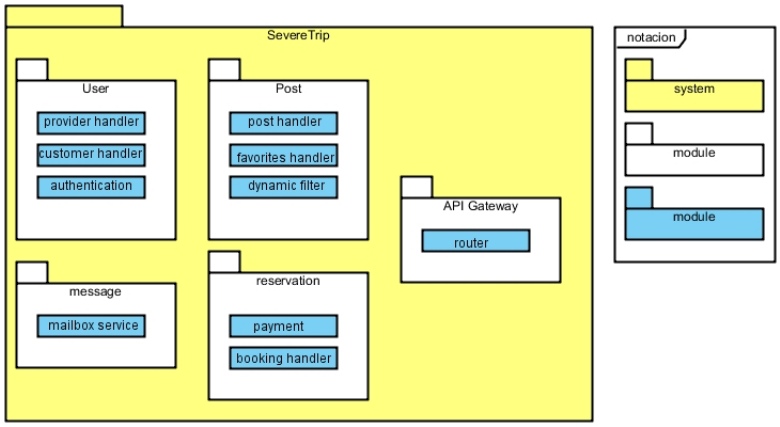
   Prevalecen los estilos mencionados anteriormente y no hay adiciones a esto ya que la parte visual se está trabajando exactamente igual que en el sprint pasado, MVC.

3. Plataforma (contenedores)

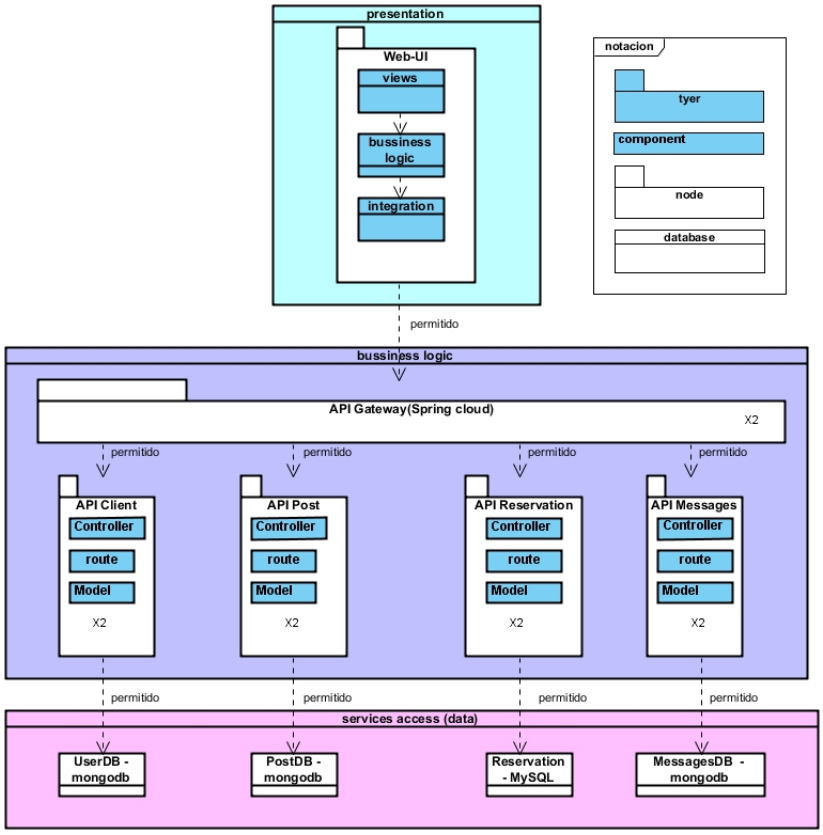
   Como se mencionó anteriormente se trabaja con Docker y Rancher, no se tienen cambios sustanciales para esta entrega y se plantea no hacer más cambios en la arquitectura (al menos en el número de nodos por el escalamiento).

4. Diseño Arquitectónico con descripción por vista

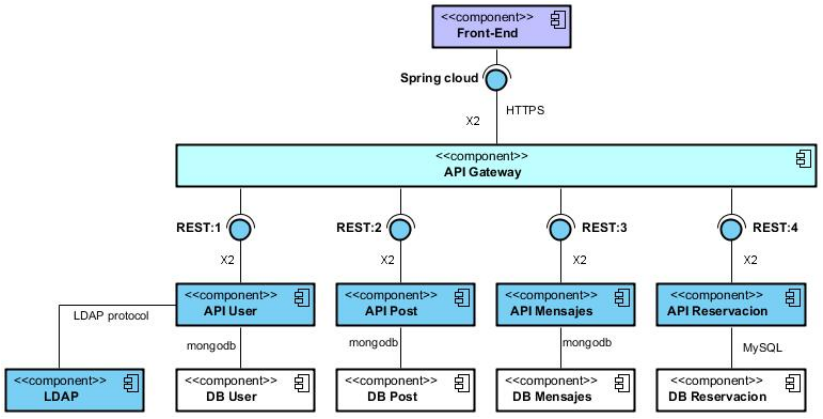
4.1. Vista de descomposición



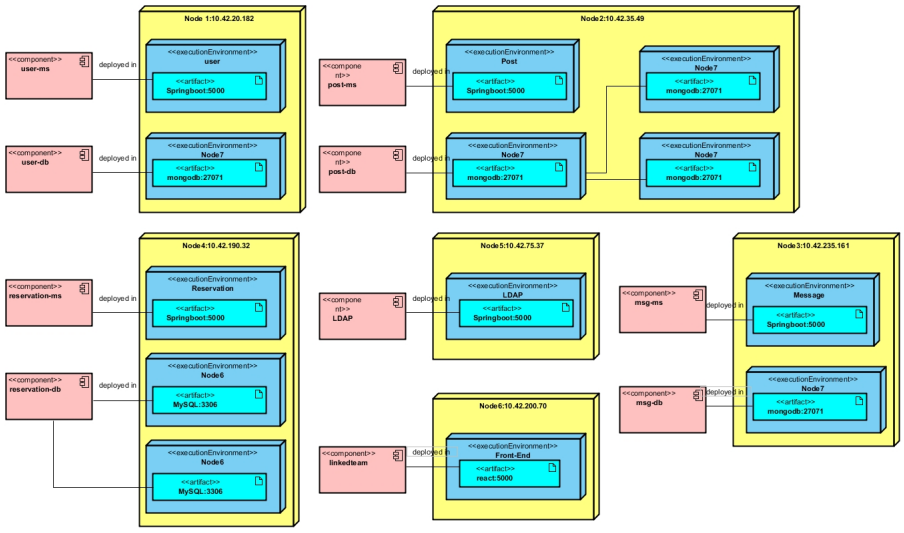
4.2. Vista de capas



4.3. Vista de componentes y conectores

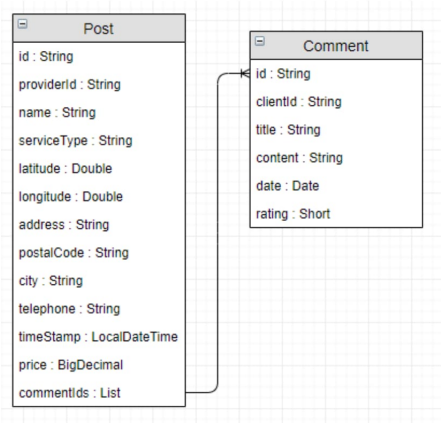


4.4. Vista de despliegue

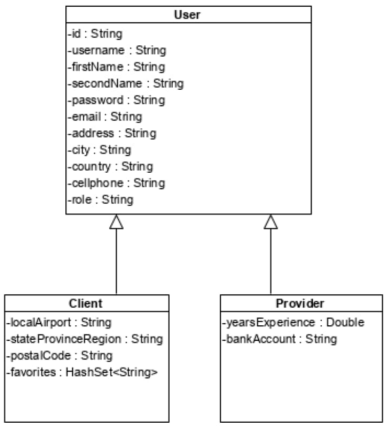


4.5. Vista de modelos de datos (un diagrama E-R por cada base de datos).

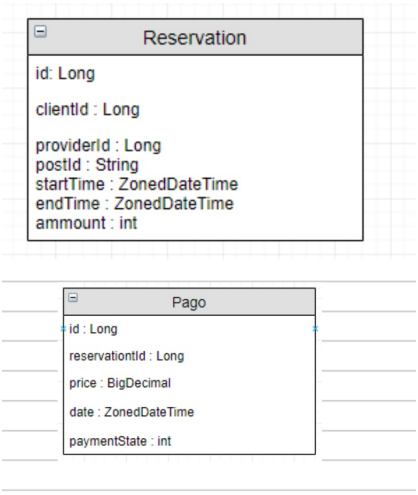
Para cada una de estas vistas se proporciona un diagrama teniendo en cuenta que cada microservicio tiene su base de datos independientemente si es sql o nosql, para esto se usarán herramientas como mongodb y MySQL dependiendo el nivel de complejidad de cada microservicio.
Las vistas son las siguientes:



MER post



MER user



MER reservation