Documentación servicio de mensajes

Introducción:

El proyecto planteado consiste en un servicio de mensajería distribuida. Los clientes de la aplicación podrán acceder a ella a través de una interfaz por línea de comandos (que no forma parte de la arquitectura). La arquitectura decidida para implementar este servicio fue un cliente servidor distribuido. Esto implica que cliente y servidor se encuentran en nodos diferentes, así como el resto de componentes.

Requisitos funcionales:

Los requisitos funcionales del sistema son:

1. Registrarse con un nombre de usuario y una contraseña, nombre que ha de ser único.
2. Loguearse.
3. Enviar un mensaje a otro usuario que esté registrado.
4. Revisar los mensajes no leídos.
5. Revisar el histórico de mensajes.
6. Borrar mensajes leídos.
7. Listar los usuarios registrados en el sistema.

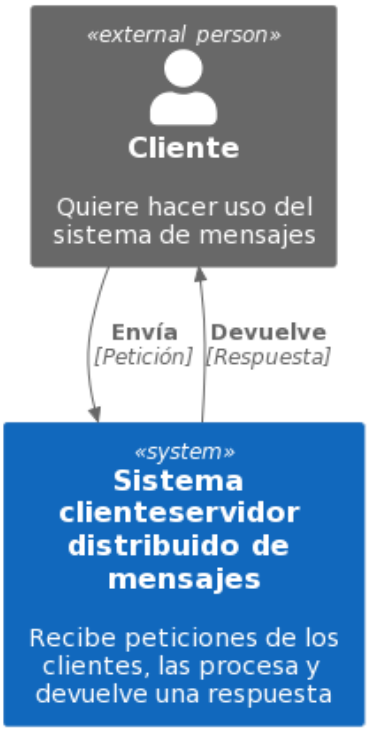
Requisitos no funcionales:

Nuestro servicio de mensajería se centra en el aspecto de la disponibilidad. Se duplican tanto balanceadores de carga como servicios. Con esto conseguimos que, en caso de uno de los nodos se caiga, el sistema pueda seguir funcionando correctamente. Es responsabilidad de un administrador de sistema volver a lanzar los nodos que se hayan caído o hayan sufrido problemas.

Por otro lado nuestro sistema es fácilmente escalable. Debido a que está configurado para llevar la lista de nodos a los que cada componente le realiza peticiones, solo es cuestión de añadir nuevos nodos a estas listas cuando sea necesario.

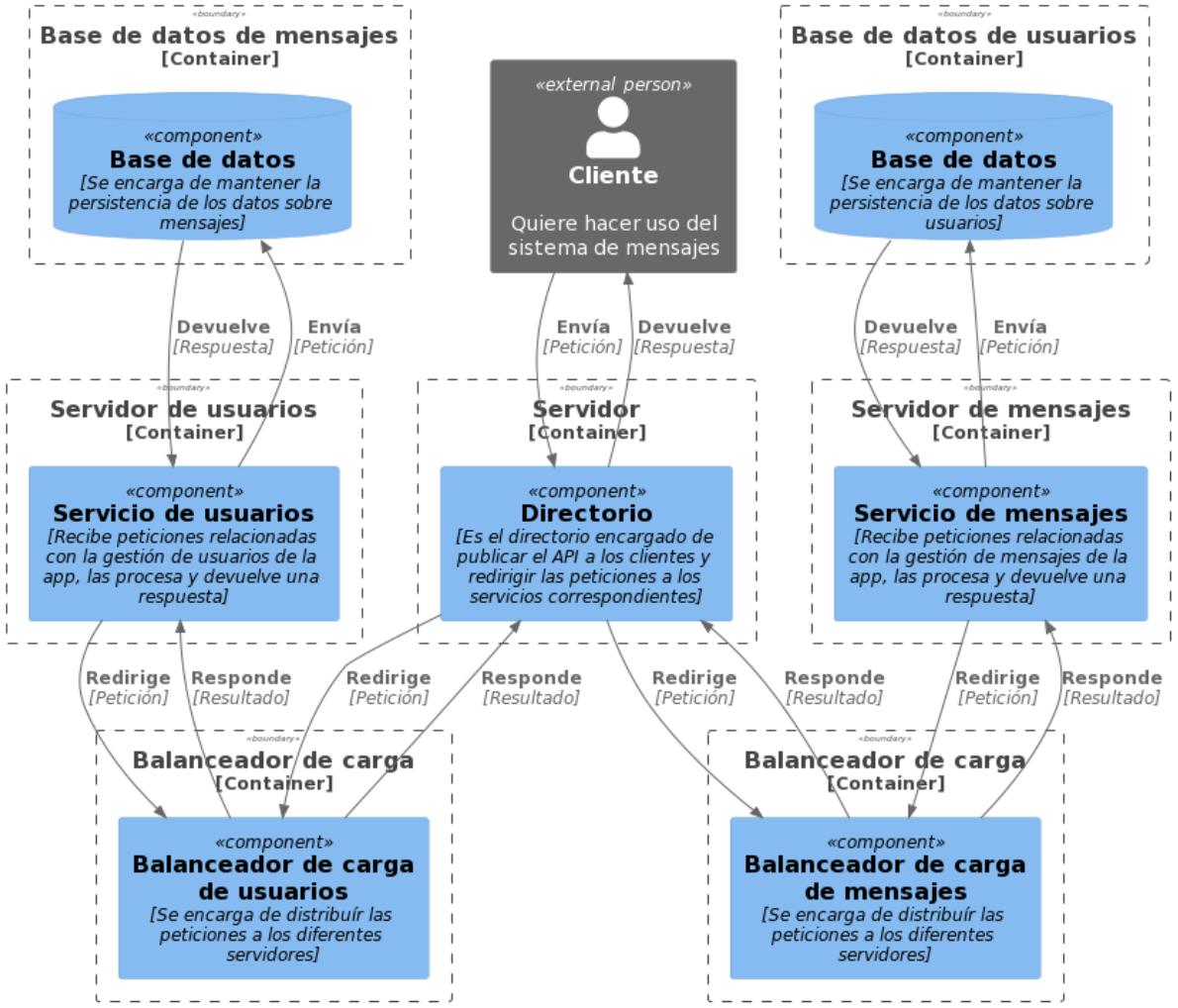
Diagramas:

- Diagrama de contexto



- En este caso el diagrama de contexto muestra de manera simplificada cómo el cliente le envía una petición al sistema, este la procesa y le devuelve una respuesta.

- Diagrama contenedor



- El diagrama contenedor pretende mostrar los diferentes contenedores que conforman nuestro sistema sin entrar en detalles sobre su estructura interna. También explica qué función realiza cada uno de estos.

- Diagrama de componente

https://github.com/UDC-FIC-AS/practica-final-davidgsalvado/blob/main/presentaci%C3%B3n/Documentacion/componente.png

- El diagrama de componente muestra los diferentes componentes que conforman nuestro sistema. En este caso sí se muestra la estructura interna de los mismos. Estos emplean módulos como ConnectionManager (para la comprobación de las conexiones con otros nodos) o NodeManager (para gestionar las listas de nodos de cada componente).

- Diagrama de nivel 4

https://github.com/UDC-FIC-AS/practica-final-davidgsalvado/blob/main/presentaci%C3%B3n/Documentacion/nivel4.png

  - Este diagrama muestra la serie de funcionalidades que se ejecutan en cada operación posible.

Tácticas:

Las tácticas que empleamos son, por un lado, **Ping/echo**. Cada vez que se va a realizar una petición a un nodo, se comprueba en el propio componente si la conexión con este sigue activa. En el caso de que así sea, se realiza la petición. En el caso de que no, se lanza una excepción. Esta es la segunda táctica empleada, las **Excepciones**. Estas permiten que en cualquier punto de fallo a lo largo del recorrido del sistema sea posible notificar al cliente y recuperarse de las mismas para que el servicio se mantenga activo.

Decisiones de diseño:

La arquitectura seleccionada para llevar a cabo este proyecto fue un cliente-servidor distribuido. Esto quiere decir que el cliente se encuentra en un nodo distinto al servidor y puede haber más componentes localizados en distintos nodos para evitar que los fallos o problemas generados en una máquina afecten a otras partes del sistema, favoreciendo a la disponibilidad de esta forma.

En un primer momento se ideó la arquitectura para que tuviera todos los servicios junto al directorio en el mismo nodo. Esta idea se descartó debido a que traía consigo muchos posibles problemas, como la tolerancia a fallos del sistema en conjunto.

Madurando la idea y pasando por diferentes posibilidades de descomposición de los elementos, terminamos con una arquitectura donde en una máquina se encuentra el directorio, que cuenta con dos estructuras auxiliares que se encargan de llevar una lista de los nodos a través de los cuales acceder a los balanceadores de carga. El sistema permite tener múltiples balanceadores de carga de cada tipo.

Los balanceadores de carga cuentan con la misma estructura auxiliar que el directorio, almacenan en listas los nodos de los servicios que cada uno oferta. Debido a que tenemos dos tipos de servicios (usuario y mensajes) consideramos

adecuado tener dos tipos de balanceadores de carga también. Cada balanceador de carga almacena múltiples nodos de servicios de su mismo tipo.
Los servicios son los encargados de la lógica de negocio. El servicio de mensajes gestiona los buzones de cada usuario y el de usuario guarda, para cada usuario, su contraseña y lleva la lista de todos los usuarios registrados. Estos servicios cuentan con la dirección al nodo de sus respectivas bases de datos (con la misma estructura auxiliar empleada en el directorio y los balanceadores de carga), a las que les realizan peticiones.
La estructura auxiliar de la que hacen uso el directorio, balanceadores de carga y servicios es un módulo llamado NodeManager. Este implementa operaciones para añadir, eliminar y conseguir nodos que están registrados internamente en una lista que mantiene un proceso Agent. La función NodeManager.get implementa la técnica “Round Robin” en los nodos que devuelve. De esta forma, cada nodo devuelto volverá a formar parte de la lista desde el final, dejando de primeros los que más tiempo llevan dentro. Es importante destacar que, a pesar de que los servicios añaden los nodos a sus bases de datos con NodeManager, al existir solo un nodo de cada base de datos, no surte efecto la técnica “Round Robin”, pues están registrados internamente en listas de un solo elemento (siempre se devolverá el único que existe). Esta decisión fue tomada con el fin de hacer uso de las funcionalidades ya implementadas. La adición o eliminación de nodos debe ser gestionada por un usuario administrador del sistema. Por otra parte, otro módulo crucial fue el ConnectionManager, este se encarga de verificar si existe conectividad con el nodo al que se le va a realizar una petición. Ayuda a implementar las tácticas de ping y excepciones, ya que si no hay conexión, lanza una excepción que es cacheada para posteriormente notificar al usuario.
Finalmente los dos únicos puntos de fallo posibles, a parte del ya obvio en la arquitectura, que sería el directorio, serían los accesos a las bases de datos.

Documentación del código:

El código se encuentra documentado en ExDoc, de tal manera que con el comando mix docs en la carpeta principal del repositorio, se generará una carpeta llamada doc en la cual aparecerá todo el html de la documentación.

Documentación de los tests:

En cuanto a las pruebas (que se encuentran explicadas con comentarios inline en el código de las mismas), se ha testeado de forma exhaustiva todo el sistema destacando los siguientes escenarios:

1. Registrarse de forma satisfactoria en el sistema, indicando un nombre de usuario y contraseña.
2. Registrarse con un usuario ya existente (**ERROR**).
3. Loguearse correctamente con un usuario ya existente.
4. Loguearse como un usuario no registrado (**ERROR**).
5. Loguearse con un usuario registrado y una contraseña incorrecta (**ERROR**).
6. Enviar un mensaje a un usuario registrado.
7. Enviar un mensaje a un usuario NO registrado (**ERROR**).
8. Revisar los mensajes no leídos del buzón de correo como usuario registrado.
9. Revisar los mensajes no leídos del buzón de correo sin estar registrado (**ERROR**).
10. Revisar todos los mensajes del buzón.
11. Revisar todos los mensajes del buzón sin estar registrado.
12. Borrar mensajes leídos.
13. Borrar mensajes leídos sin estar registrado (**ERROR**).
14. Listar los usuarios registrados a los que puede enviar un mensaje.