# DOCUMENTO DE ARQUITECTURA

## DIEGO ALEJANDRO GONZALEZ GUALTEROS
## CRISTIAN ANDRES CASTELLANOS FINO
## EDUARDO OSPINA MEJIA

# ADVENTURE MAP

# 1. Introducción

Dentro del desarrollo de Adventure Map, se llegaron a plantear varios requisititos que se necesitarían para el desarrollo del proyecto. Al ser este un proyecto trabajo por un grupo, se requieren plantear correctamente todas las decisiones de arquitectura para permitir el desarrollo funcional de la aplicación.

## 1.1. Propósito

*Este documento proporciona un resumen general sobre la arquitectura de la aplicación pensada como proyecto final de la materia Arquitectura de Software. La idea del proyecto es poder crear un mapa con múltiples jugadores y monstruos donde estos puedan interactuar para peleas. Se tiene planteado la implementación con real time y batallas de manera concurrente. Se desea tener una implementación a partir de los servicios de Azure, y se debe tener en cuenta los atributos de calidad descritos más adelante para la arquitectura en su totalidad.*

## 1.2. Audiencia

*Diego Alejandro González Gualteros (Arquitecto-Estudiante)*
*Cristian Andrés Castellanos Fino (Arquitecto-Estudiante)*
*Eduardo Ospina Mejía (Arquitecto-Estudiante)*
*Diego Andrés Triviño Gonzalez (responsable de Aseguramiento de Calidad)*

## *1.3. Alcance*

*El Documento de Arquitectura abarca la definición del proyecto a través de las vistas de casos de uso, lógica (análisis y diseño) e implementación, atributos de calidad, requerimientos, restricciones y descripciones detalladas. De igual manera, se va a revisar el ambiente de despliegue al cual va a ser sometido el proyecto. Esto con la finalidad de que se pueda asegurar los atributos de calidad pedidos por el responsable de Aseguramiento de Calidad y se permita entender en su totalidad la arquitectura a desarrollar.*

# 2. Arquitectura del Producto/Sistema

## 2.1. Vista Funcional

### 2.1.1 Modelo de Contexto lógico

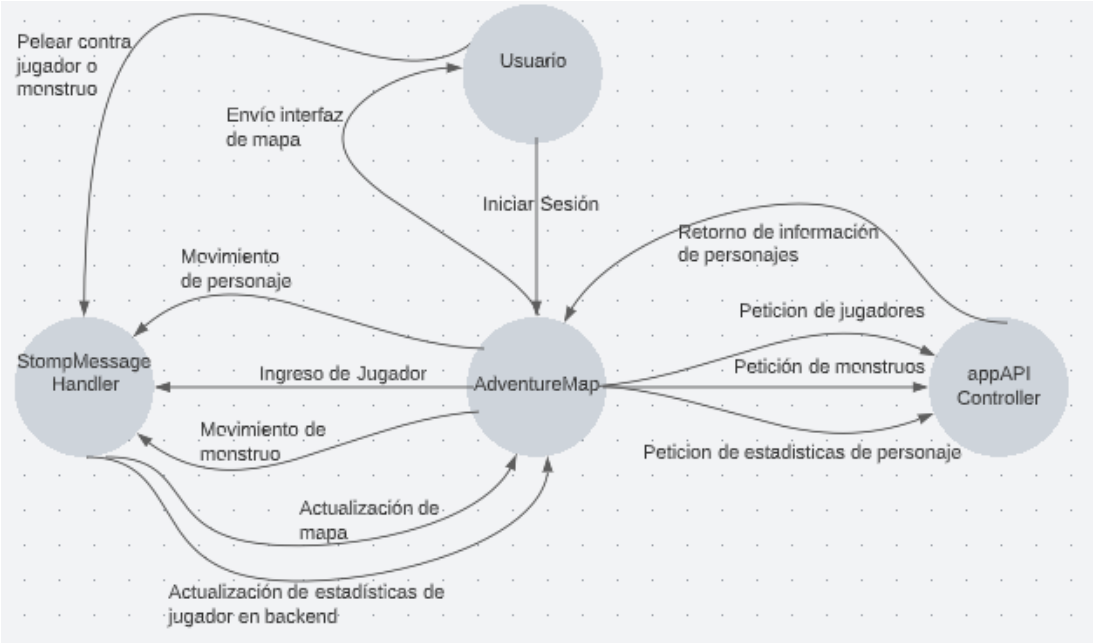


*Diagrama de Contexto lógico*

*En el diagrama del contexto lógico podemos ver cómo hacemos uso de los elementos principales a desarrollar para el funcionamiento de la aplicación. Para este proyecto manejaremos un Stompmessage handler, un api controller, al usuario que se conecta y a la página donde correrá el servicio. Tomamos el servicio de Adventure map que es de donde se centra el diagrama, en este se logra ver como funcionara la conexión con el usuario, que como planteamos más adelanta se realizara a través de Active directory de Azure, y le permite a este ingresar a la interfaz del mapa, el api controller es el que nos permite manejar todas las peticiones que se hagan durante el juego, ya sea relacionado con los jugadores o los monstruos. Y el Stomp Message Handler, que a partir de este se hace manejo de todo lo relacionado con los websockets que permiten manejar la concurrencia y el real time dentro de la aplicación mientras se encuentra corriendo.*

### 2.1.2. Modelo de contexto implementación Cloud (Azure)

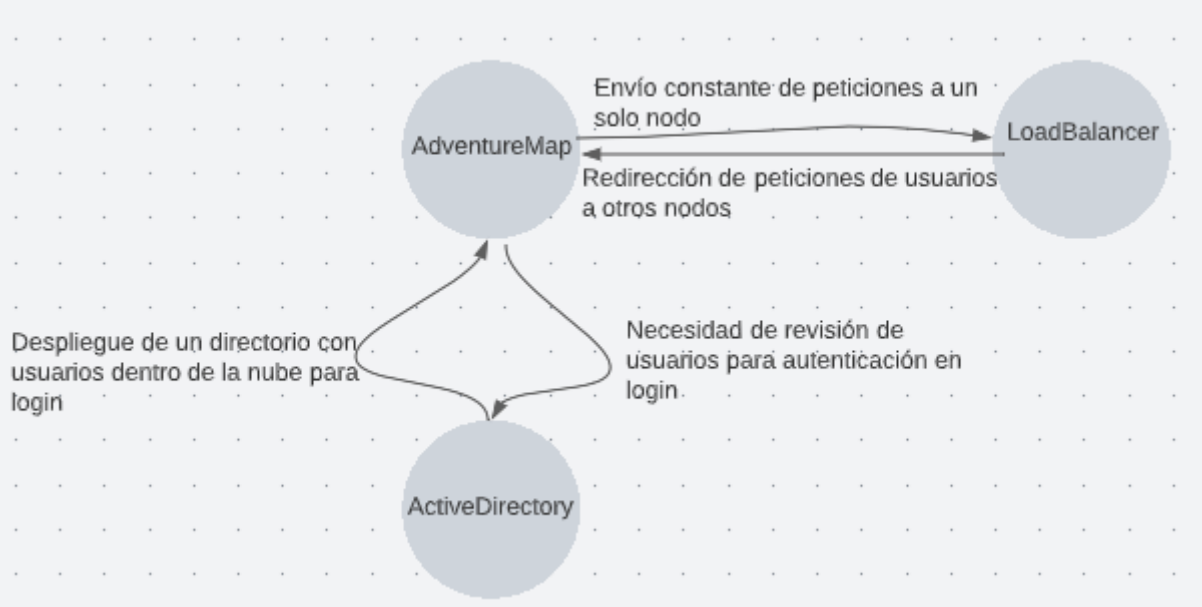


*Diagrama de Contexto Servicios Azure*

*Este diagrama nos permite ver de acuerdo con las necesidades del usuario y las necesidades de los requerimientos vemos los recursos que se utilizaron, en nuestro caso siendo estos los elementos externos que se utilizaron para el funcionamiento del proyecto, en nuestro caso siendo estos, como el active directory de Azure, los virtuales machines conectadas al balanceador de cargas y el sonar cube.*

### 2.1.3. Modelo de Casos de Uso

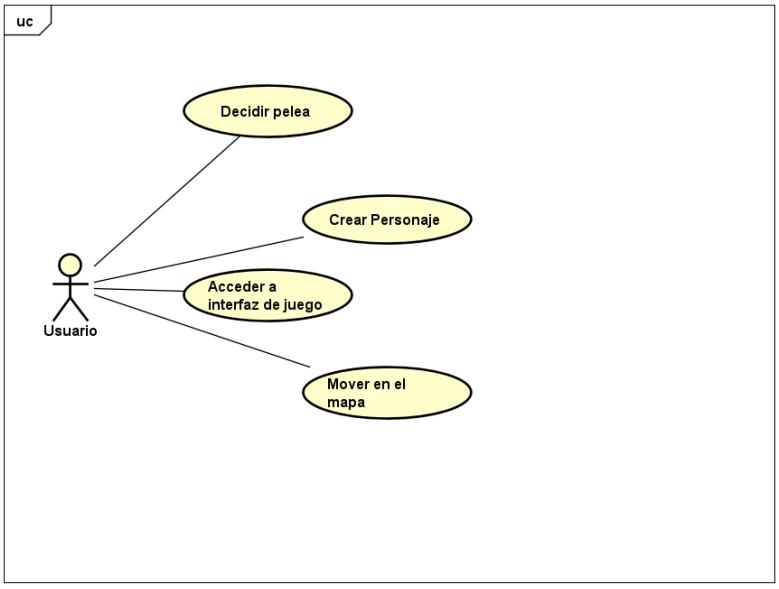


*Diagrama Casos de Uso*

En este diagrama planteamos todas las funcionalidades importantes para el cliente a partir de los casos de uso de la aplicación que planteamos para dar muestra de estos.

### *Contexto de casos de Uso*

- COMO usuario QUIERO decidir el punto de acción PARA poder revisar si se pelea o se huye de algún monstruo o usuario

- COMO usuario QUIERO crear personaje PARA poder entrar a la interfaz de juego

- COMO usuario QUIERO acceder a la interfaz de juego PARA poder interactuar con los demás usuarios y monstruos

- COMO usuario QUIERO acceder a la interfaz de juego PARA poder interactuar con los demás usuarios y monstruos

## 2.1.4. Modelos de Secuencia

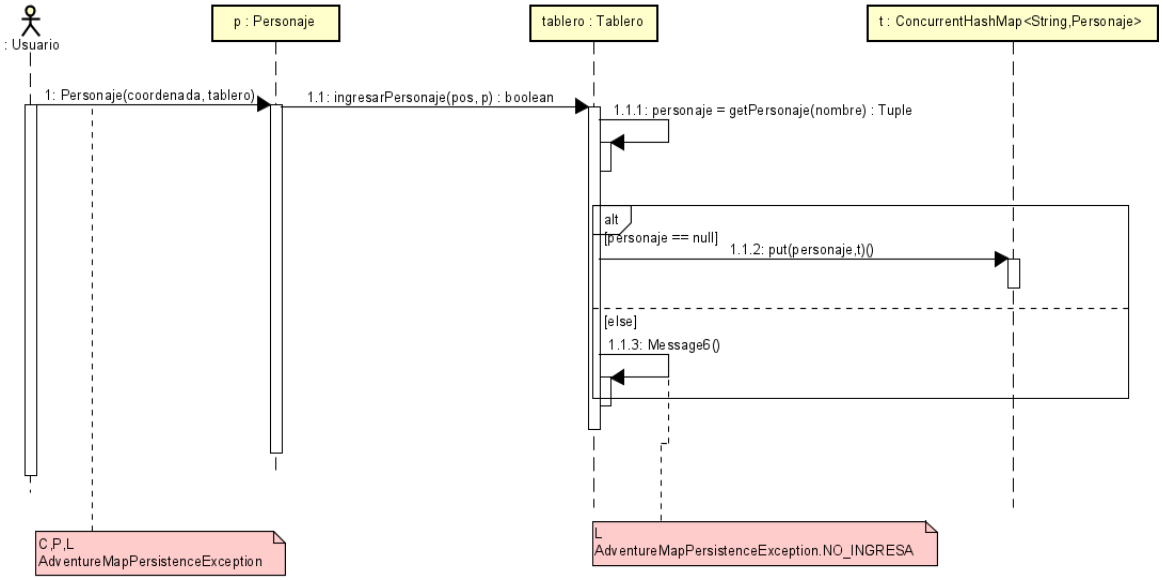


Diagrama de Secuencia Creación de Jugador

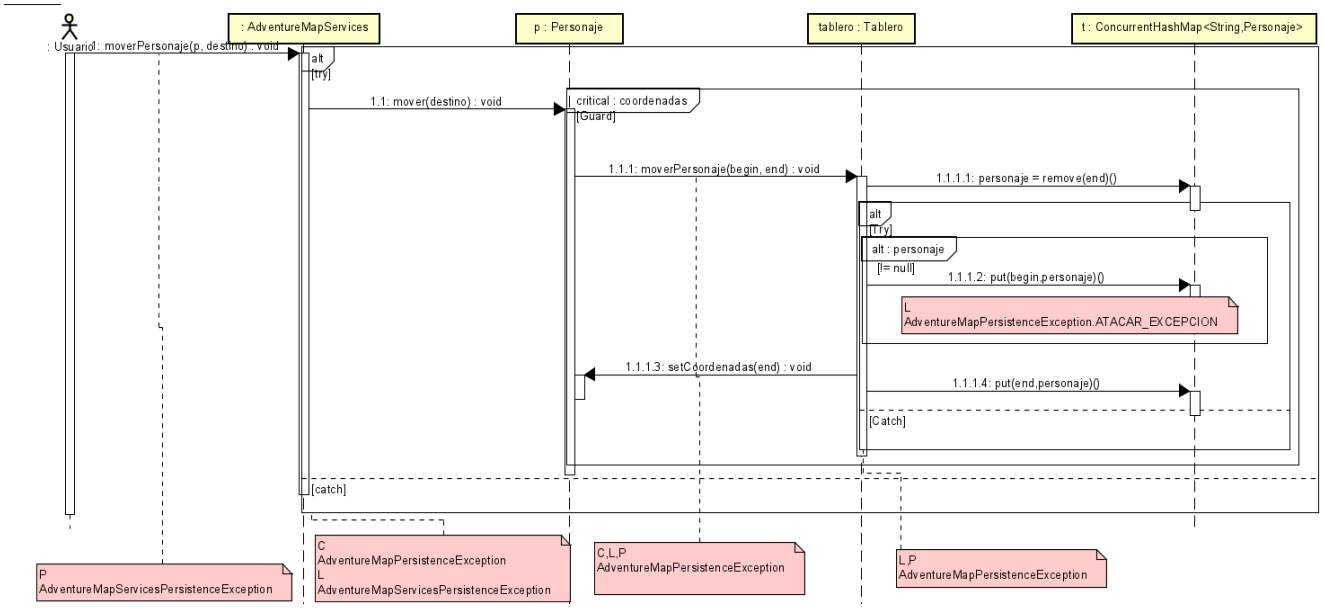


Diagrama de Secuencia Movimiento de Jugador

### 2.1.5. Modelo de actividades

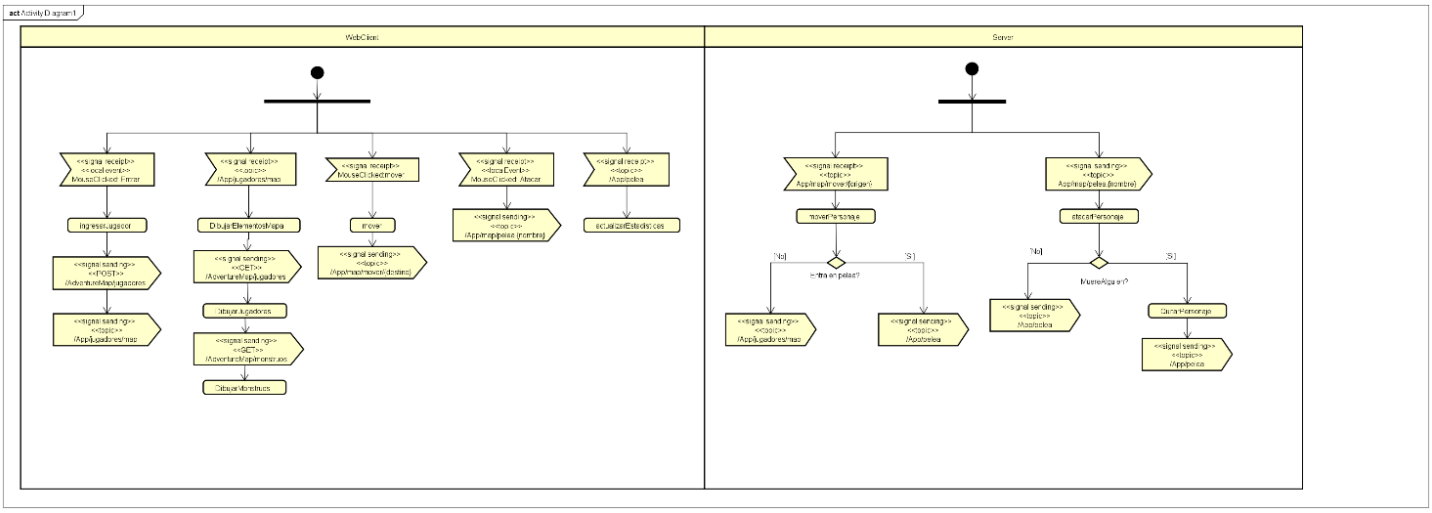


Diagrama de actividades

Se diseño el manejo de actividades a través de distintos tópicos, para el caso de esta arquitectura se utilizan 4 tópicos, dos dentro del cliente web y dos dentro del servidor. Dentro del cliente web el primer tópico es el de manejo de las posiciones de los jugadores y los monstruos, mandando a pedir los datos del servidor de estos componentes y se repintan, esto sucede cuando se suscriben a este tópico. El siguiente tópico dentro del cliente web es el de actualizar las estadísticas de los involucrados en la pelea. A parte de estos tópicos, dentro del cliente web también se encuentran las actividades que hacen funcionar al juego como la de ingreso del usuario, el movimiento del jugador dentro del tablero y la actividad de mandar el ataque que haga el jugador.

Por parte del servidor se encuentran dos tópicos, uno para el movimiento del jugador y otro para el ataque del jugador, en donde en cada uno de estos cuenta con condicionales como si el jugador muriese con el siguiente ataque y dependiendo de eso toma acciones diferentes.

## 2.2. Vista Lógica

### 2.2.1. Paquetes de Diseño Arquitectónicamente Significativos

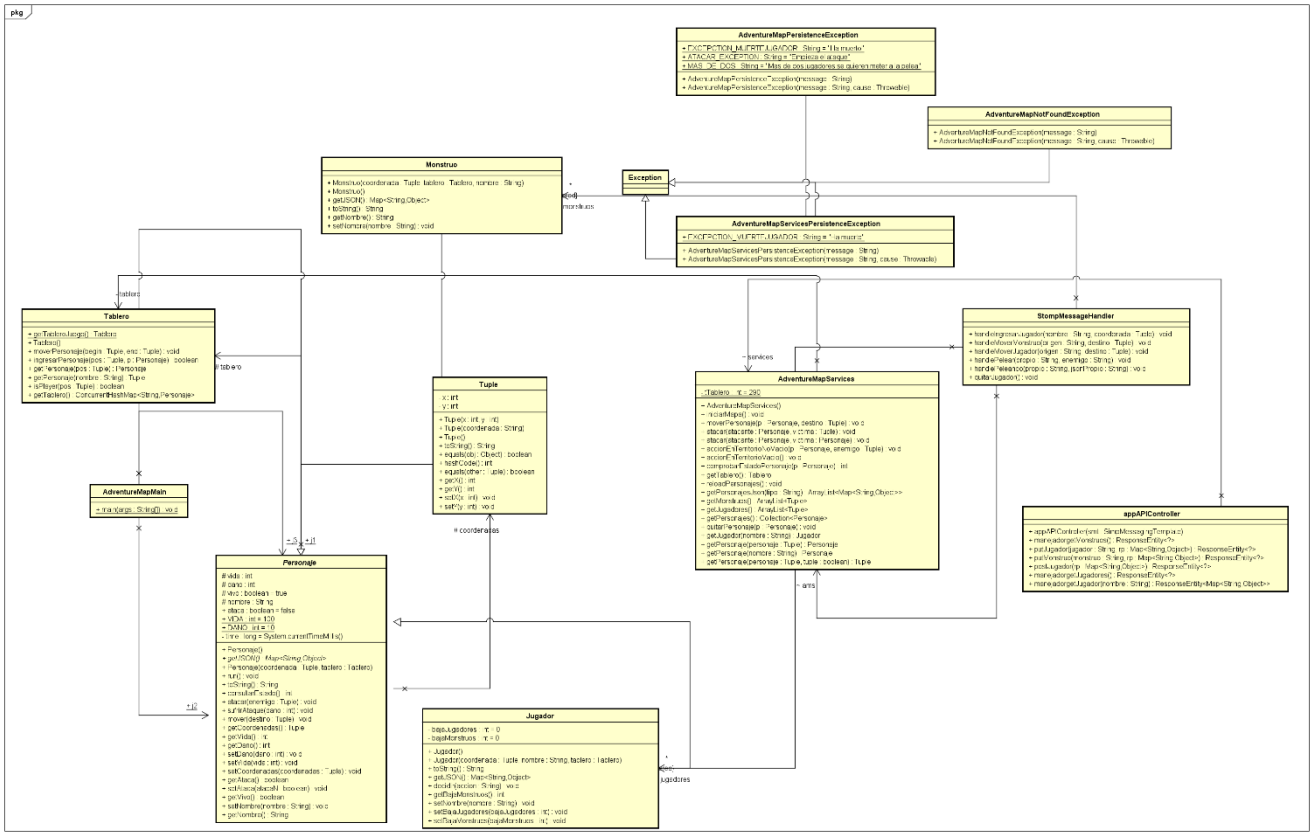


Diagrama de clases

**Se tienen 5 entidades principales:**

- **Jugador**: Personaje que crea el usuario al entrar al juego
- **Personaje**: Clase padre de Jugador y Monstruo, este contiene todas las funciones de retorno de estadísticas, recibimiento de daño y ataque del personaje
- **Monstruo**: Jugador generado el cuál deambula en el mapa hasta que alguien lo ataca. Ataca cada que reciba dos ataques del jugador
- **Tablero**: Clase que se usa para mantener los datos de los jugadores y monstruos dentro del tablero de juego. Esta clase también se encarga de retornar los elementos que tenga dentro, así como del movimiento de estos
- **Tuple**: Es un arreglo de posiciones de acuerdo con la capacidad del tablero en frontend. Sus funciones principales son las de retornar los valore de X, Y que este tenga guardados en sus variables

En cuestiones de **funcionalidad del juego**, se tiene una clase principal: AdventureMapServices. Esta clase se encarga de movimiento, ataque y pedidos de información tanto de monstruos como de jugadores. Se aprecia que tiene una relación con Tablero, y esto es porque tablero se encarga de la actualización de todos los datos y acciones de los objetos que tiene dentro.

Para el manejo de excepciones, se crean dos clases:

- **AdventureMapPersistenceException**: Se encarga de las excepciones que puede generar los servicios de AdventureMap, por ejemplo, la muerte del jugador, la excepción de ataque entre otras.
- **AdventureMapNotFoundException**: Se usa cuando algún recurso no existe dentro de los valores contenidos en el backend.

Se usan dos servicios fundamentales en el funcionamiento de los requerimientos principales del juego: **STOMP** y **REST.** El primero en cuestión se usa cuando hay modificaciones rápidas de datos o flujo de red constante. **REST** sirve para traer desde el backend mediante **peticiones (GET, PUT, POST, DELETE)** información contenida en este.

Se crean dos clases aparte para estos servicios. La primera de **STOMP** se encarga de los movimientos y peleas de los jugadores y monstruos. La segunda, **REST**, se encarga de las peticiones de información como estadísticas de jugadores, posiciones de los monstruos para repintarlos entre otros.

### *2.3.* Vista de Implementación - Componentes

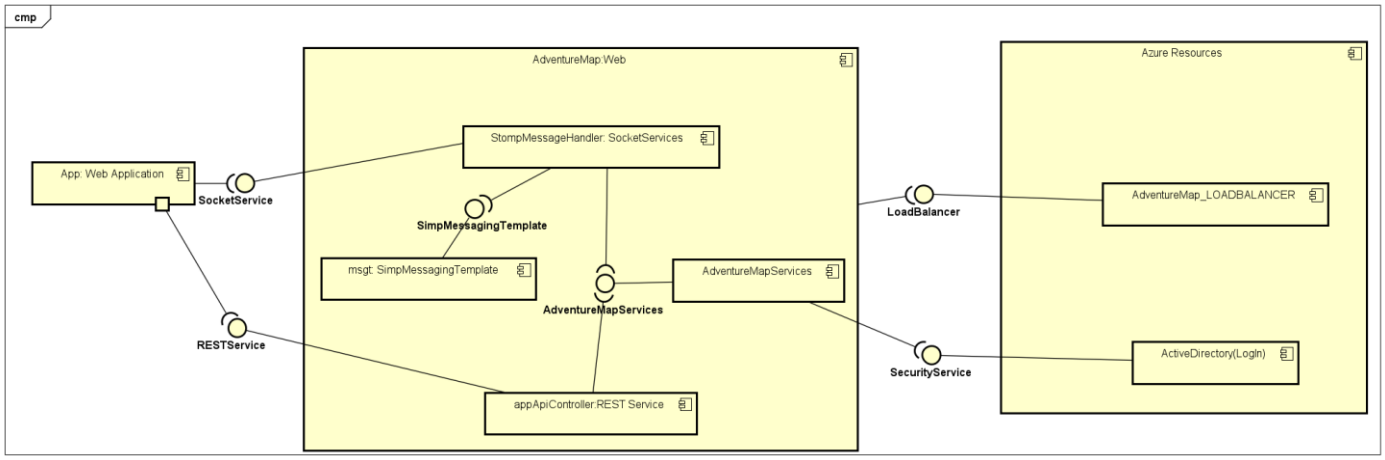


**Diagrama de componentes**

El primer nodo que se tiene es App, este necesita de los servicios de **REST** y **STOMP** debido a que se necesitan tópicos y peticiones tanto para el movimiento de los monstruos como para las estadísticas actualizadas de los monstruos y jugadores.

Dentro del nodo de **AdventureMap**:**Web** se tiene los servicios usados para la conexión entre jugadores para que todos puedan ver los mismos objetos dentro del mapa, así como su movimiento en tiempo real. En este caso tenemos el servicio de **StompMessage** para tópicos. Este necesita de los servicios de **SimpMessagingTemplate** para que se pueda conectar con suscripciones de **backend** mediante etiquetas. También se tiene por dentro el servicio **backend** de

REST donde este recibe del tablero la información de los objetos que se encuentran actualmente en el juego. Por último, se encuentra el **API** de servicios de **AdventureMap**, el cual se encarga de actualizar el juego en cuestión de estadísticas, peleas y movimientos en el **backend**.

En el tercer nodo, **Azure**, este nos presta sus servicios tanto de directorio activo como de **load balancer**, esto con el fin de manejar el tráfico de red de manera adecuada entre los dos nodos

(máquina virtual). De igual manera, nos presta para todo el nodo de **AdventureMap**:**Web** los servicios de **ActiveDirectory** para el **login** de las personas.

## *2.4.* Vista de Despliegue

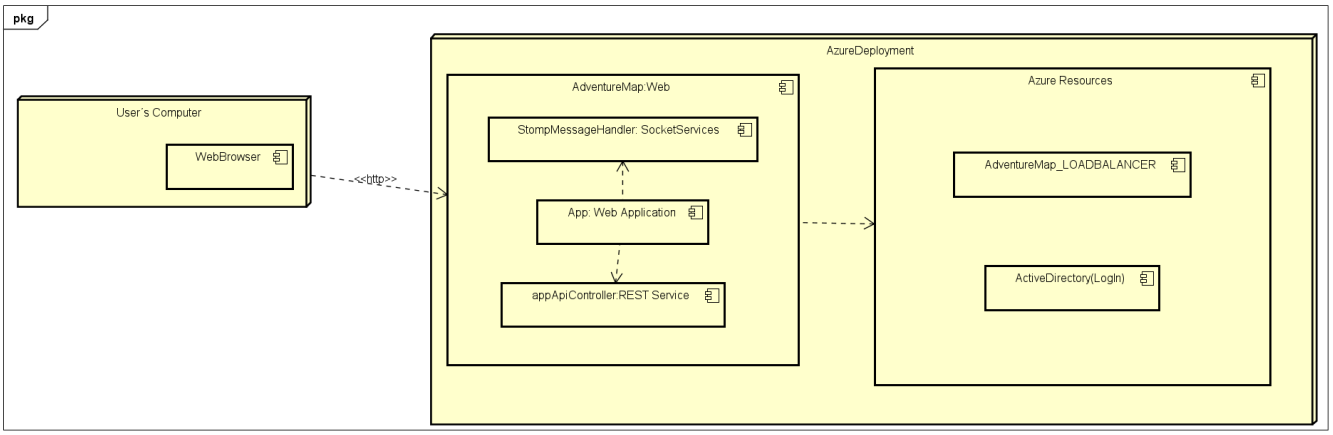


**Diagrama de Despliegue**

El diagrama de despliegue tiene un nodo llamado "User's Computer", este se encarga de la conexión por parte del usuario hacia el ambiente de despliegue (en este caso Azure), a través de peticiones HTTP, donde se le pide elementos tales como "index.html" para el ingreso del jugador y "Mapa.html", la cual es la interfaz del juego.
Revisando el nodo AzureDeployment, este contiene los siguientes componentes dentro del nodo de AdventureMap:

- **App: Web Application ->** Este componente es la conexión del frontend (Lenguaje JavaScript) hacia el usuario. Esto con el fin de conectar el ambiente de etiquetas de HTML con el ambiente de desarrollo de JavaScript.

- **StompMessageHandler** -> Este es un componente que se rige a partir de STOMP. Este se crea debido a que funciones tales como "Mover Personaje", "Pelear", entre otros, se manejan a través de suscripciones y envío de mensajes. Este componente se encarga de recibir los mensajes que se le envíen desde el frontend, y, a través de la etiqueta @MessageMapping, revisa a cuál tópico fue enviado el mensaje para empezar a correr las instrucciones de acuerdo a lo definido en el backend.

- **appAPIController** -> Este es un componente que se rige a partir de REST. Esto significa que a este componente le van a llegar solicitudes HTTP como "GET" y "PUT".

En donde todos estos componentes se conectan con los servicios de azure utilizado en la arquitectura, el adventuremap load balancer que es el que se ocupa de revisar el tráfico de red y dependiendo de si es mucho tráfico para el nodo que se está mandando y en caso de necesitarlo hacer la redirección a otro nodo para distribuir el tráfico entre el número de máquinas suscritas al balanceador. Y el elemento de Active directory, que es al que se realiza la validación para permitir el acceso a los usuarios escritos dentro del directorio creado.

## 2.5. Requisitos de Software/Hardware

*Equipamiento y Configuración:*
*Requisitos de dispositivo de usuario:*
- *Sistema Operativo* Windows, Linux o MAC

*Requisitos de componentes*
- *Modalidad de alojamiento de funcionalidad: Cloud (Azure)*

# 3. Atributos de Calidad

| Atributo de Calidad | Fuente | Estímulo | Entorno | Artefacto | Respuesta | Medida de Respuesta |
|---|---|---|---|---|---|---|
| Disponibilidad. | Como Administrador del sistema del juego de Adventure Map. | Mantener habilitado el acceso al sistema en cualquier momento bajo condiciones de fallos de sistema. Eso con el fin de poder brindar oportunidad de ingresar al juego y poder probar todas las funcionalidades que este ofrece | El intento de un jugador por acceder al juego cuando se produce la caída de uno de los servidores por factores externos o fallos internos y estos eventos no le permitan acceso al juego ni a ninguna función del servicio. | Balanceador de carga de capa que redirija de manera oportuna todo el tráfico de red al conectarse los jugadores. Esto ayuda a que, en caso de fallo de uno de los nodos servidor. Se pueda mandar el estado actual del juego (posiciones, información de jugadores) para que el otro nodo controle la totalidad del tráfico de red | Se espera poder prevenir el fallo, para tener acceso continuo a la aplicación y al juego. Este se aplicará a través de la escalabilidad horizontal que permitirá múltiples servidores. corran la aplicación y en caso de que alguno se caiga, el juego seguirá funcionando como se espera gracias a que todo el tráfico de red que manejaba la máquina caida, la manejará la que sigue activa | Restablecer la aplicación en un tiempo no superior a 10 seg. |
| Seguridad | Como Usuario de Adventure Map. | Ingresar a la aplicación con una cuenta verificada por un agente exterior, donde este sea capaz de autenticar las credenciales de estos y así poder evitar posibles entradas de personal no autorizado | Acceso a la aplicación en un entorno normal de funcionamiento. Gracias a agentes externos tales como Azure Active Directory, donde se genera una interfaz de logueo para la autenticación y posterior conexión de usuarios registrados en la base | Pantalla de inicio del juego, que conecta con el servicio de azure Active Directory para la posterior redirección hacia la interfaz de logueo que esta ofrece. Para, en caso de un logueo satisfactorio, poder habilitar el botón de entrar al juego. | Dependiendo si tiene acceso o no al juego, Se debe notificar al usuario que no se puede ingresar sin antes haberse registrado. O redirigirlo a la pantalla del juego en donde puede elegir un nombre y empezar a jugar. | Autenticación de los datos de jugador con aceso a las estadísticas y posición de otros jugadores y los monstruos para poder jugar una vez se haya autenticado. Se genera un filtro para no dejar pasar personal no registrado en la base del Active Directory |
| Mantenibilidad | Como Desarrollador de Adventure Map. | Modificar y realizar Cambios en el código dependiendo de lo que se necesite para la aplicación A partir del análisis del código fuente. | Entorno de desarrollo de la aplicación, siendo este en el lenguaje que se esté manejando y el acceso a los datos. Dentro del ambiente de cada desarrollador de la aplicación. | La aplicación de SonarQube. Para el análisis del código y regreso de la calificación del código fuente. | Debe retornar el análisis de código pertinente. Con una calificación en cada una de las áreas importantes qie se deben tener en cuenta para el desarrollo de código. | Valoración mínima de B. Para que el código cuente como pasable. |

Para este proyecto se van a manejar 3 atributos de calidad específicos:

***Implementación de atributos de calidad***

1. ***Seguridad***

   Se piensa en usar algún login externo que nos pueda brindar la aplicación Azure. Esto debido a que crear un login propio depende de varios factores como una base de datos donde guardar la información de los usuarios, sistemas de cifrado de contraseñas tales como SHA-256 entre otros para cumplir con atributos de seguridad en bases de datos, así como una interfaz de logueo. Esto consume demasiados recursos y tiempo de los desarrolladores.

Existe una herramienta que ofrece Azure: Active Directory. Es una base de datos y un conjunto de servicios que conectan a los usuarios con los recursos de red que

necesitan para realizar su trabajo. La base de datos (o el directorio) contiene información crítica sobre su entorno, incluidos los usuarios

***2. Disponibilidad***

En vista de que, gracias a las herramientas tanto de Publish/Subscriber como de REST vamos a tener un flujo de red mediano, se crea un balanceador de carga donde se pueda filtrar todas las peticiones del juego entre dos máquinas. De igual manera, se suben los tópicos a la nube para que ambas máquinas estén conectados a tópicos comunes. Esto ayuda al caso en que alguno de los nodos se caiga, la otra máquina sea capaz de mantener todo el flujo de red por si sola mientras se vuelve a subir la máquina que se cayó.

***3. Mantenibilidad***

Gracias al aplicativo SonarCloud, la cual es una plataforma de análisis de código continuo y online con la que se pueden analizar los proyectos (bugs, errores de seguridad, duplicación de código). Con esta herramienta se revisa que todos los puntos que revisa estén en una calificación mínima de B (Aceptable)

# 4. Requisitos:

Para el desarrollo de esta arquitectura se tienen requerimientos que permiten a la aplicación funcionar como se desea.

***1. Requerimiento: RealTime***

los jugadores puedan ver todos los movimientos que se hacen dentro del mapa de manera inmediata. Esto permite una experiencia dinámica y fluida cuando se esté jugando en el mapa, pues se sabrá las posiciones reales tanto de los enemigos como de los demás jugadores.

***2. Requerimiento: Concurrencia***

Se implementa concurrencia dirigida hacia todos los elementos que se encuentren dentro del mapa, con la finalidad de que no se generen condiciones de carrera ni deadlocks dentro del juego al momento de las batallas contra los monstruos y otros jugadores, lo cual podría afectar de manera negativa la jugabilidad de las personas al momento de sus ataques.

***3. Requerimientos funcionales***

***a. Monstruos***

- Movimiento de los monstruos de manera automática dentro del mapa
- Capacidad de ataque automático (Este ataca al jugador cada que recibe dos ataques)
- Los monstruos no deben tener la capacidad de atacar a otros monstruos o a jugadores. Estos para que empiecen a atacar debe ir un agente externo hacia ellos
- Capacidad de eliminar al monstruo del mapa en caso de que este muera

***b. Jugadores***

- Capacidad del jugador de poder moverse a libertad por los cuadros del mapa
- Capacidad de iniciar batallas con jugadores o monstruos
- Implementación de mitigación de condición de carrera de batalla de más de dos jugadores. En este caso el tercer jugador (quien se quiere meter a la pelea de los otros dos) no podrá atacar y se le avisará de que tiene que ir a otro lado

***c. Mapa***

- El mapa debe tener las estadísticas actualizadas de los jugadores que están peleando
- Implementación de botones de pelea y huida para jugadores
- Generación automática de monstruos al momento de iniciar la partida
- Envío al backend de datos actualizados de jugadores y monstruos (posiciones, estadísticas) por medio de herramientas como Publish/Subscriber o REST

# 5. Restricciones:

Las restricciones que aplican para este proyecto son:

- La mayoría de los componentes deben ser alojados en la nube.
- Componentes para ejecución del programa como Spring-Boot, Maven y Java fueron alojados dentro de máquinas virtuales Linux Ubuntu
- El juego no guarda información de anteriores partidas
- Se debe manejar persistencia únicamente al momento que el juego esta activo y una partida se encuentra en movimiento.

- Debe permitir más de 1 jugador entrar al tablero por cada juego.
- Deben generase únicamente 4 monstruos al inicio de la partida.
- Debe funcionar correctamente el realtime y la concurrencia.
- Se debe utilizar tópicos para el manejo de datos entre jugadores.
- Se usa REST para extracción de estadísticas de jugadores

# 6. Supuestos:

- Para poder hacer login con el active directory, el usuario necesita tener una cuenta registrada en el directorio activo generado.
- Solo se puede acceder con un jugador por pestaña de navegador.
- El jugador solo tiene permitido moverse cuando está fuera de pelea y con puntos de vida restantes.
- Los monstruos no pueden entrar en pelea con otros monstruos u algún jugador por su propia cuenta.

# 7. Conclusiones:

- Se logra completar la arquitectura que se desea utilizar para la creación de Adventure map, indicando su funcionamiento, como planeamos hacer uso de los recursos disponibles y las reglas que debemos seguir para su creación. Se busco generar un documento especializado que contenga toda la información del proyecto y su desarrollo.

- Dentro de esta arquitectura se definieron varias decisiones importantes para el desarrollo del proyecto, para empezar, se definieron los atributos de calidad con sus escenarios que deben ser incorporados dentro de la arquitectura. Realizamos los diagramas con sus explicaciones

para poder dar a entender cómo se busca que funcione la aplicación para cualquier persona que lea el documento. Todas estas decisiones se realizaron teniendo en cuenta las restricciones, los requerimientos y los supuestos planteados para el proyecto.