DOCUMENTO ARQUITECTURA

PAINT IT!

Angie Natalia Mojica Diaz
Daniel Antonio Santanilla Arias
Jefer Alexis González Romero

DECEMBER 10, 2023

GOLDEN3SURE

Contenido

1. Driver de arquitectura

1.1. Requerimientos funcionales

1.1.1. Configuración Paint It!

Página de Inicio

- **Como** usuario **quiero** entrar a una página de inicio **para poder** decidir si creo o me uno con un código a una partida en PaintIt!
- **Como** usuario **quiero** buscar una partida con un código que tengo **para poder** entrar a una sala con mis amigos.
- **Como** usuario **quiero** crear una partida de juego **para poder** jugar con mis amigos a PaintIt!

Personalizar Partida

- **Como** creador de una partida **quiero** ajustar el tamaño del tablero **para poder** jugar en un tablero más grande o pequeño.
- **Como** creador de una partida **quiero** tener la opción de establecer un límite de tiempo para mi partida de juego **para poder** ajustar la duración del juego de acuerdo con mis necesidades y prioridades.
- **Como** usuario **quiero** tener una zona donde pueda desistir de la personalización del tablero **para poder** ir pantalla principal.
- **Como** creador de una partida **quiero** tener una zona donde pueda aceptar mis cambios **para poder** empezar a jugar.
- **Como** creador de una partida **quiero** poder ver las configuraciones organizadas **para poder** decidir cuáles son las configuraciones que usaré en mi partida.

Personalizar el jugador

- **Como** usuario **quiero** poder ver las configuraciones organizadas **para poder** decidir cuales solas las configuraciones que usaré en mi jugador.
- **Como** usuario **quiero** agregar mi nombre en la personalización del jugador **para** que los demás jugadores vean mi nombre en la tabla de clasificación.
- **Como** usuario **quiero** tener una zona donde pueda aceptar mis cambios, o en caso contrario desistir de la personalización del jugador **para poder** empezar a jugar o ir a la pantalla principal.
- **Como** usuario **quiero** tener una zona donde pueda desistir de la personalización del jugador **para poder** ir pantalla principal.

Sala de espera

- **Como** creador de una partida **quiero** obtener un código de la sala que estoy haciendo **para poder** dárselo a mis amigos y que ellos entren a mi sala.
- **Como** usuario **quiero** ver configuraciones y jugadores en una sala de espera **para poder** ver contra quien me enfrento y los nombres que usarán, además de pensar una estrategia para poder ganar.
- **Como** anfitrión de una partida **quiero** poder iniciar la partida de juego **para poder** comenzar a jugar con mis amigos o los diferentes jugadores que se unieron a mi partida.

Comodines de juego

- **Como** creador de una partida **quiero** tener la opción de agregar comodines al tablero **para poder** mejorar mis posibilidades de ganar el juego y agregar desafíos.

1.1.2. Juego Paint It!

Zona de Juego

- **Como** usuario **quiero** ver cómo se va pintando el tablero de juego con mi color o el de los demás **para poder** pintar zonas que no son de mi color en el tablero y tener un mayor puntaje.

Tablas de Clasificación

- **Como** usuario **quiero** ver la tabla de clasificaciones de jugadores **para poder** quien es el que ha pintado el mayor número de casillas y así sentirme motivado y comprometido con el juego.

Temporizador de Partida

- **Como** usuario **quiero** tener la capacidad de monitorear el tiempo restante durante la partida **para poder** estar consciente del límite de tiempo y ajustar mi estrategia de juego en consecuencia.

Restricciones y supuestos arquitectónicos

1.2.1. Restricciones

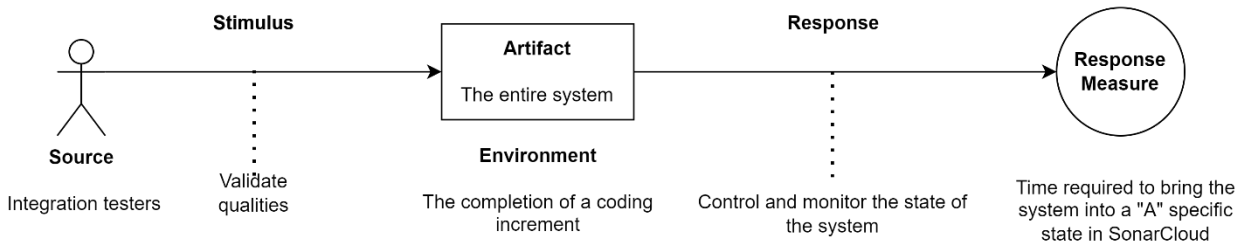
- El proyecto debe ser desplegado en la plataforma Azure.
- Se debe realizar en 2 sprints cada uno de 2 semanas.
- La planificación, el control de tiempos y la generación de Burndown Charts deben realizarse en Azure DevOps.
- La arquitectura planteada tiene que permitir escalar horizontalmente el Back-End.
- La aplicación debe ser sometida a una revisión de código estático como parte del ciclo de integración continua (Inspección Continua).
- Todas las métricas de SonarQube deben estar en estado “A” (Verde) y debe haber una cobertura de pruebas unitarias de al menos el 40%.
- Se deben cumplir con criterios de seguridad, mantenibilidad y disponibilidad.

1.2.2. Supuestos

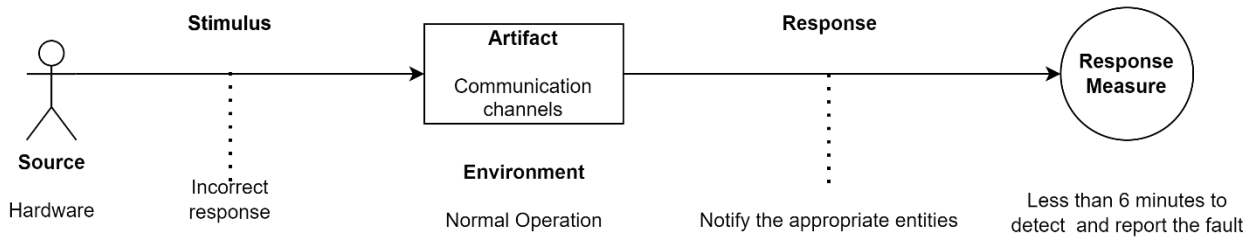
- Se asume que el equipo tiene las habilidades y conocimientos necesarios para trabajar con SCRUM, Azure DevOps, y la plataforma Azure.
- Se tendrá acceso a los créditos necesarios en Azure para implementar la arquitectura requerida.
- Se asume que la plataforma Azure es adecuada para las necesidades del proyecto.

1.3. Escenarios de calidad del sistema

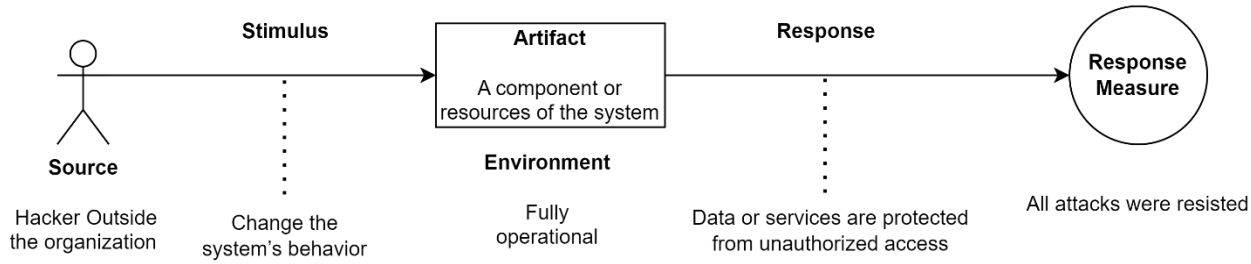
1.3.1. Mantenibilidad

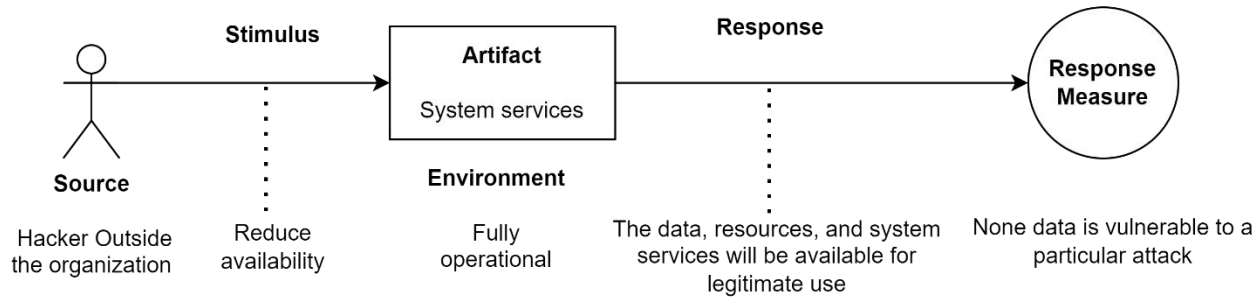


1.3.2. **Disponibilidad**



1.3.3. **Seguridad**





2. Decisiones arquitectónicas

Con el objetivo de satisfacer las expectativas de calidad establecidas para este proyecto, las cuales se detallaron previamente en la sección de Drivers Arquitectónicos, hemos adoptado una serie de decisiones arquitectónicas estratégicas. Estas decisiones, que se describen a continuación, han sido cuidadosamente consideradas y seleccionadas para garantizar que cumplimos con los estándares de calidad y rendimiento esperados. Además, estas decisiones nos permitirán abordar eficazmente cualquier desafío técnico que pueda surgir durante la implementación del proyecto.

- Considerando el tamaño de la aplicación, hemos decidido adoptar el estilo arquitectónico Cliente-Servidor como el principal enfoque para nuestro diseño. Esta decisión se basa en el hecho de que nuestra aplicación no requiere una gran cantidad de funcionalidades, pero sí necesita realizar transacciones con un sistema de caché. Por lo tanto, recomendamos una separación clara entre el front-end y el back-end. Esta estructura nos permitirá gestionar eficientemente las interacciones entre el cliente y el servidor, optimizando el rendimiento y la escalabilidad de nuestra aplicación.
- Teniendo en cuenta el estilo arquitectónico seleccionado y el número de funcionalidades que se van a implementar, hemos tomado la decisión de desplegar tanto los componentes de frontend como de backend utilizando el servicio App Services de Azure. Esta elección nos permite aprovechar las ventajas de la escalabilidad, la gestión simplificada y la integración continua que ofrece Azure, asegurando así un despliegue eficiente y robusto de nuestra aplicación. Además, al utilizar un servicio

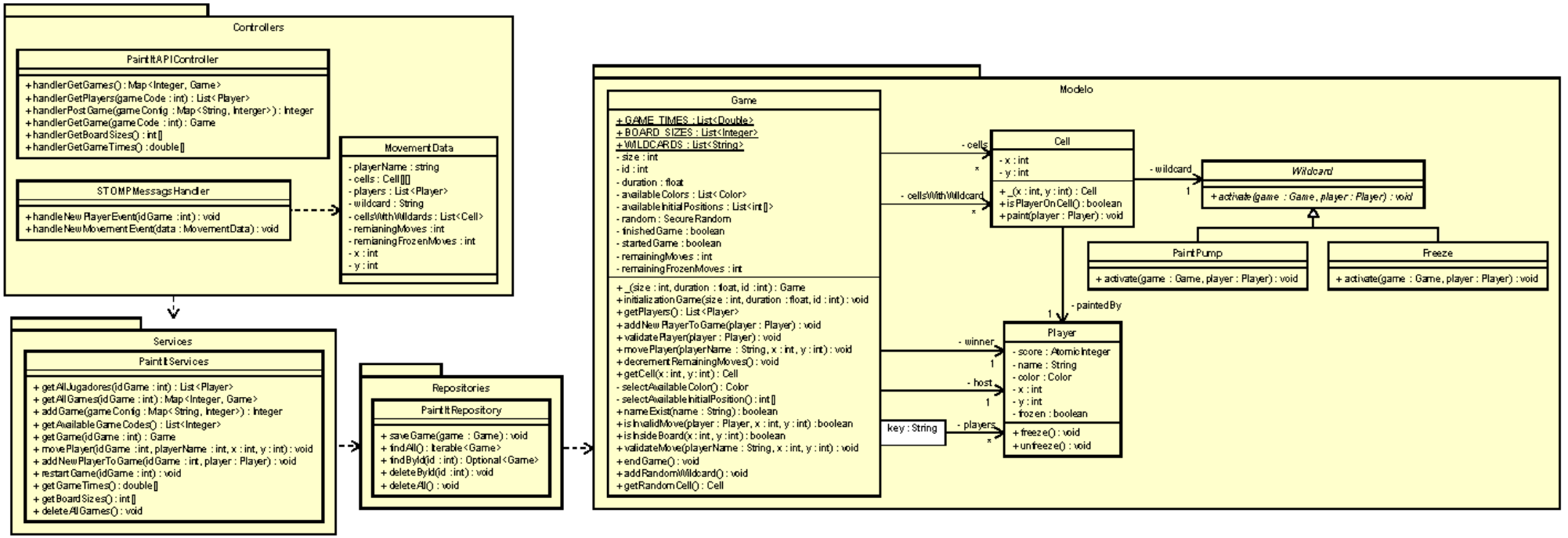
administrado, podemos centrarnos más en el desarrollo de las funcionalidades de la aplicación y menos en la gestión de la infraestructura.

- Con el objetivo de manejar eficientemente las solicitudes de los usuarios en nuestra aplicación, hemos decidido implementar un API Gateway. Este actuará como un único punto de entrada, simplificando así la administración del tráfico. El API Gateway será responsable de dirigir las solicitudes a los respectivos backend pools, asegurando una distribución de la carga y una respuesta rápida a las solicitudes de los usuarios.
- Decidimos utilizar un Firewall de Aplicaciones Web (WAF) en el Application Gateway esto debido a su capacidad para mejorar la seguridad de la aplicación al protegerla contra una variedad de amenazas, cumplir con las normativas de seguridad de la información, prevenir interrupciones del servicio al bloquear el tráfico malicioso, proporcionar un mayor control y visibilidad sobre el tráfico de tu aplicación.
- Realizamos la implementación de un sistema de caché ya que, por su capacidad para mejorar el rendimiento y la velocidad de la aplicación permite una recuperación rápida de los estados del juego, aumentar la resiliencia de la aplicación al prevenir la pérdida de datos en caso de interrupciones del servicio, y ayudar a manejar la carga adicional a medida que aumenta el número de usuarios y partidas, mejorando así la escalabilidad de la aplicación.
- Se manejarán las peticiones API Rest y los websockets en diferentes backend pools esto por la capacidad de cada uno para ser optimizado para su tipo específico de tráfico, mejorando así el rendimiento y la eficiencia. Además, permite que cada uno escale independientemente según sea necesario. También aumenta la resiliencia de la aplicación, ya que, si un backend pool experimenta problemas, el otro puede seguir funcionando normalmente. Por último, simplifica el mantenimiento de cada uno, mejorando la mantenibilidad de la aplicación.

3. Diagramas UML

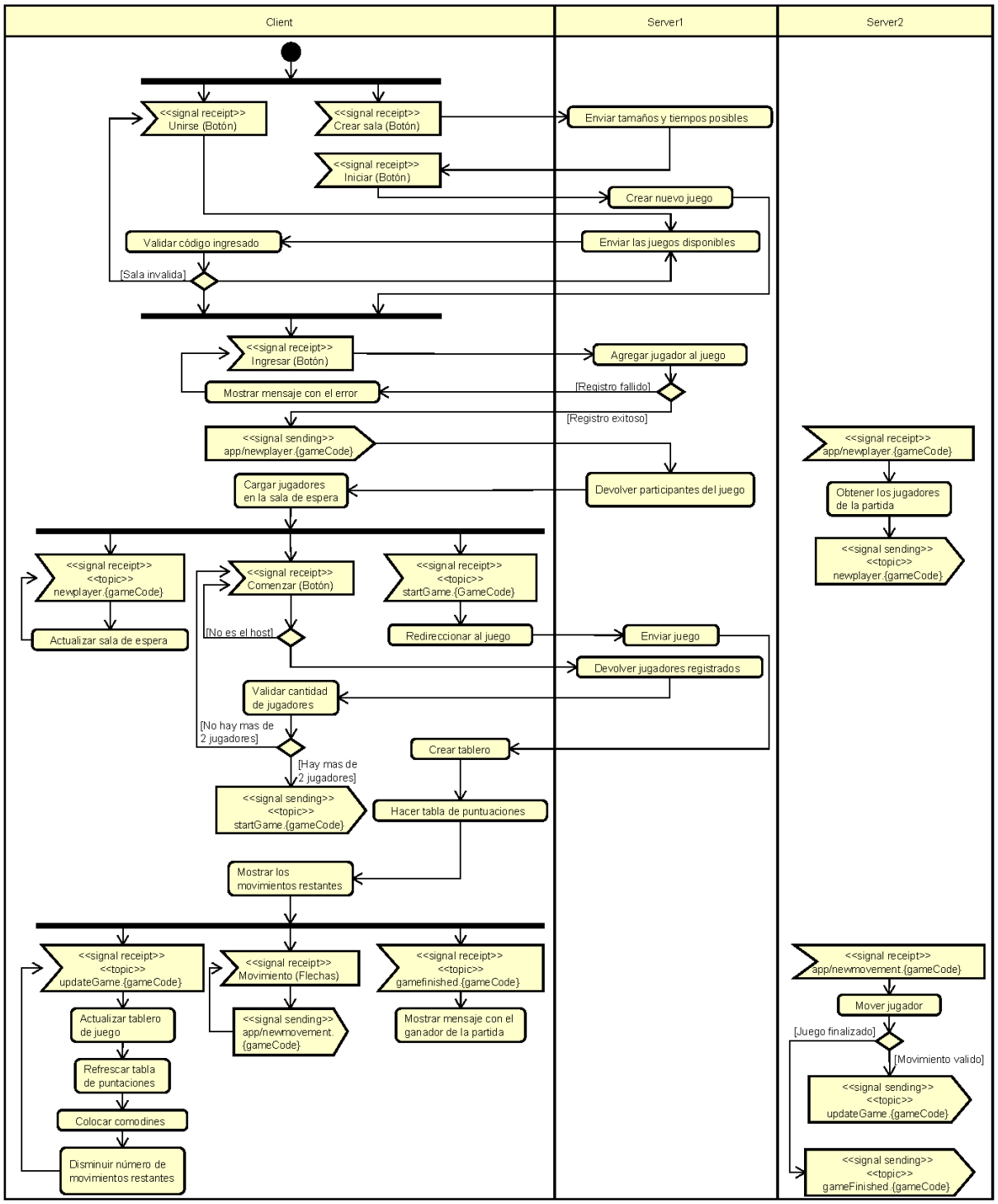
3.1. Vista lógica

3.1.1 Diagrama de Clase

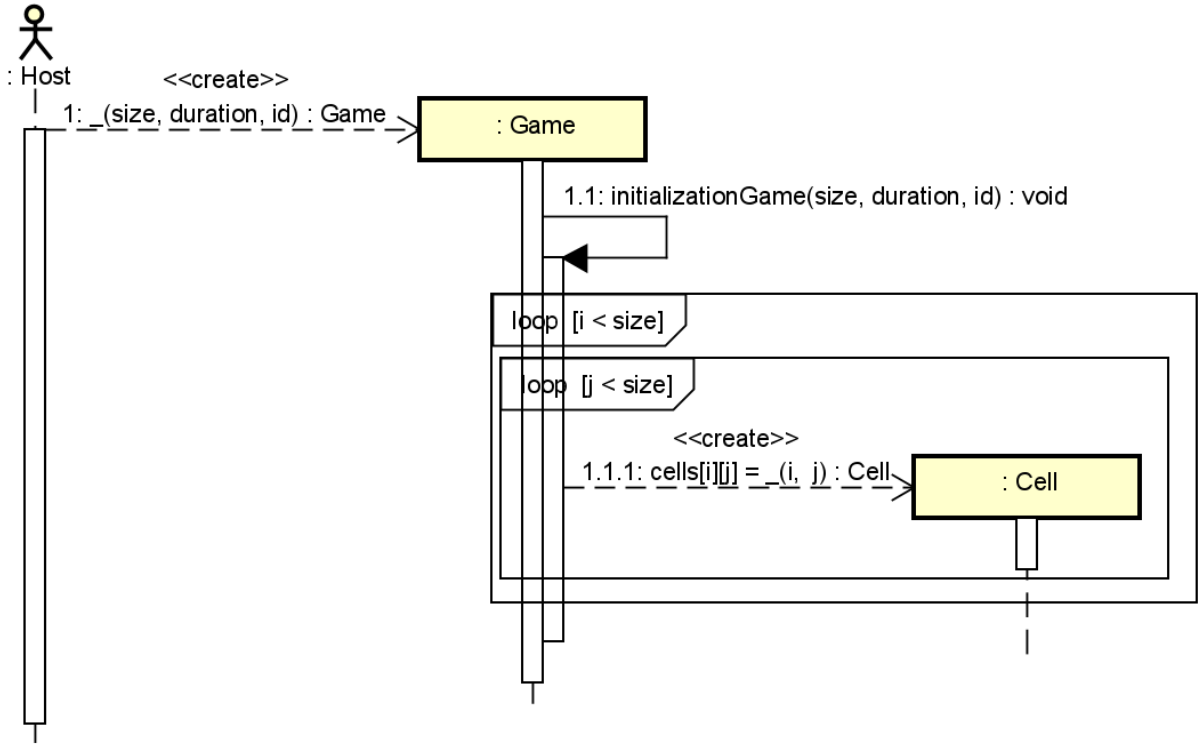

Controllers
PaintItAPIController
+handlerGetGames() : Map<Integer, Game>
+handlerGetPlayers(gameCode : int) : List<Player>
+handlerPostGame(gameConfig : Map<String, Integer>) : Integer
+handlerGetGame(gameCode : int) : Game
+handlerGetBoardSizes() : int[]
+handlerGetGameTimes() : double[]
STOMPMessagesHandler
+handleNewPlayerEvent(idGame :int) : void
+handleNewMovementEvent(data : MovementData) : void
MovementData
- playerName : string
- cells : Cell[][]
- players : List<Player>
- wildcard : String
- cellsWithWildards : List<Cell>
- remianingMoves :int
- remianingFrozenMoves : int
- x : int
- y : int
Services
PaintItServices
+ getAllJugadores(idGame : int) : List<Player>
+ getAllGames(idGame : int) : Map<Integer, Game>
+ addGame(gameConfig : Map<String, Integer>) : Integer
+ getAvailableGameCodes() : List<Integer>
+ getGame(idGame : int) : Game
+ movePlayer(idGame :int, playerName : int, x : int, y : int) : void
+ addNewPlayerToGame(idGame : int, player : Player) : void
+ restartGame(idGame : int) : void
+ getGameTimes() : double[]
+ getBoardSizes() : int[]
+ deleteAllGames() : void
Repositories
PaintItRepository
+ saveGame(game : Game) : void
+ findAll() : Iterable<Game>
+ findById(id : int) : Optional<Game>
+ deleteById(id :int) : void
+ deleteAll() : void
Modelo
Game
+ GAME_TIMES : List<Double>
+ BOARD_SIZES : List<Integer>
+ WILDCARDS : List<String>
- size : int
- id : int
- duration : float
- availableColors : List< Color>
- availableInitialPositions : List<int[]>
- random : SecureRandom
- finishedGame : boolean
- startedGame : boolean
- remainingMoves : int
- remainingFrozenMoves : int
+ _(size : int, duration : float, id :int) : Game
+initializationGame(size : int, duration :float, id : int) : void
+getPlayers() : List<Player>
+addNewPlayerToGame(player : Player) : void
+validatePlayer(player : Player) : void
+movePlayer(playerName : String, x : int, y : int) : void
+decrementRemainingMoves() : void
+getCell(x : int, y : int) : Cell
- selectAvailableColor() : Color
- selectAvailableInitialPosition() : int[]
+nameExist(name : String) : boolean
+isInvalidMove(player : Player, x : int, y : int) :boolean
+isInsideBoard(x : int, y :int) : boolean
+validateMove(playerName : String, x : int, y : int) : void
+endGame() : void
+addRandomWildcard() : void
+getRandomCell() : Cell
- cells
*
- cellsWithWildcard
*
Cell
- x : int
- y : int
+ _(x : int, y : int) : Cell
+isPlayerOnCell() : boolean
+paint(player : Player) : void
- wildcard
1
Wildcard
+activate(game : Game, player : Player) : void
PaintPump
+ activate(game : Game, player : Player) : void
Freeze
+ activate(game : Game, player : Player) : void
1
- paintedBy
- winner
1
- host
1
key : String
- players
*
Player
- score : AtomicInteger
- name : String
- color : Color
- x : int
- y : int
- frozen : boolean
+freeze() : void
+unfreeze() : void

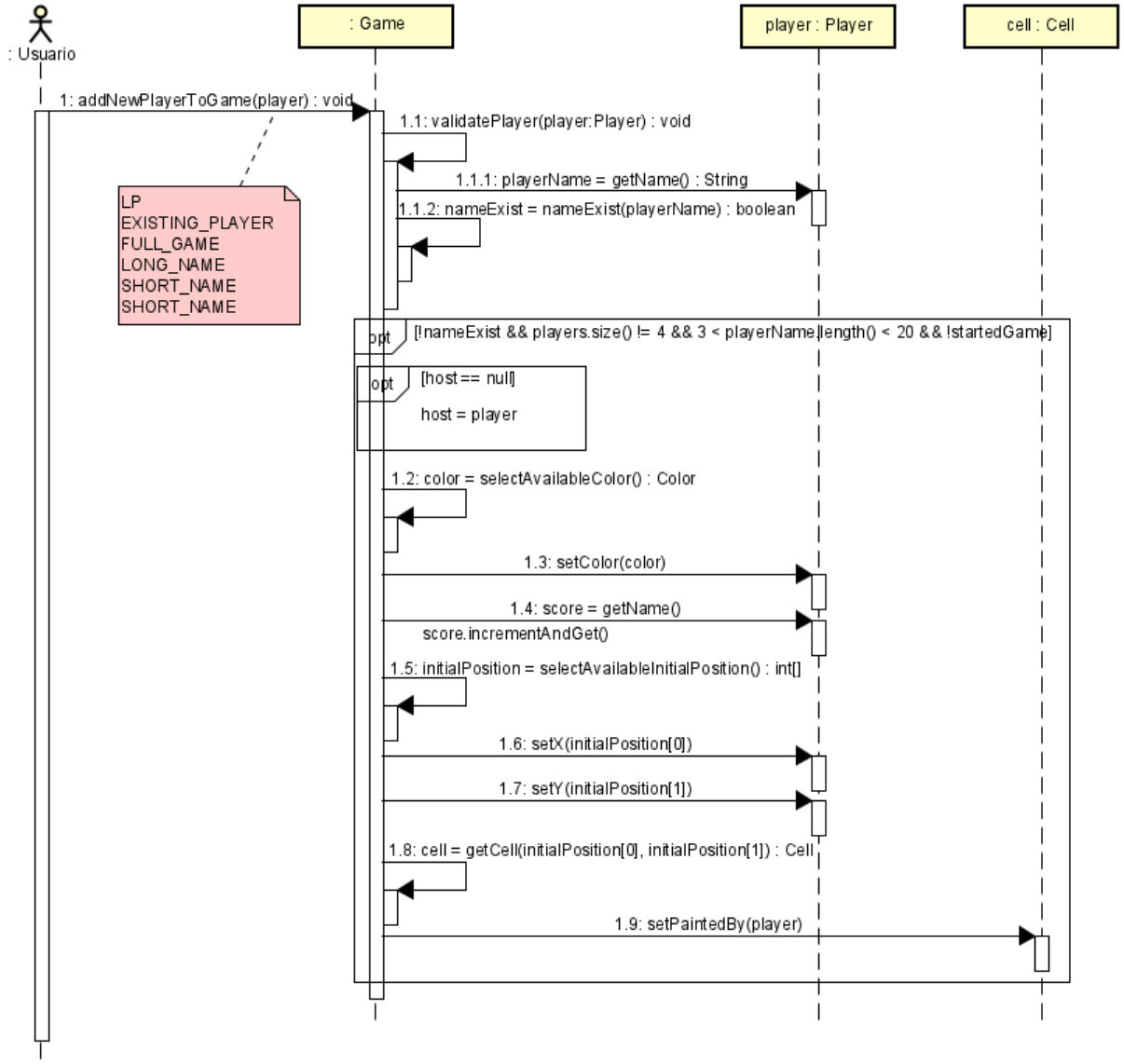
3.2. Vista de procesos

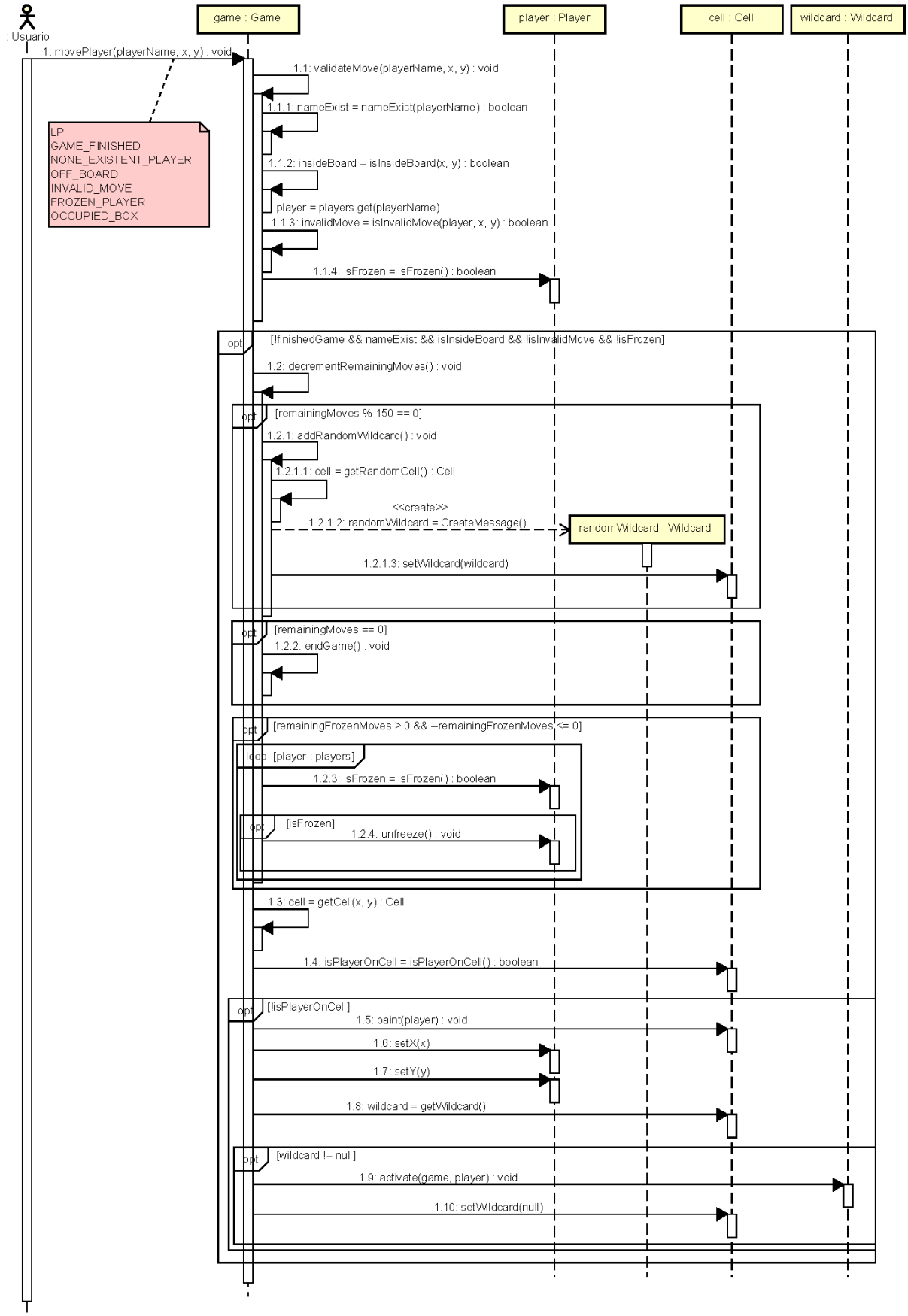
3.2.1 Diagrama de actividades



3.2.2 Diagrama de secuencia

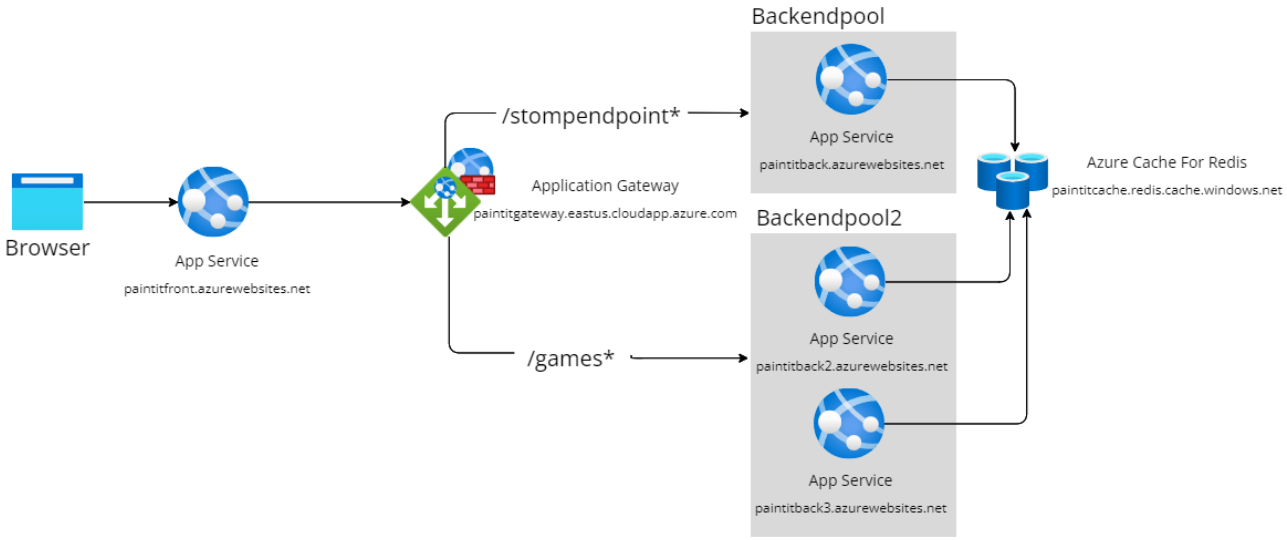



: Usuario
: Game
player : Player
cell : Cell
1: addNewPlayerToGame(player) : void
1.1: validatePlayer(player:Player) : void
1.1.1: playerName = getName() : String
1.1.2: nameExist = nameExist(playerName) : boolean
LP
EXISTING_PLAYER
FULL_GAME
LONG_NAME
SHORT_NAME
SHORT_NAME
opt
[!nameExist && players.size() != 4 && 3 < playerName.length() < 20 && !startedGame]
opt
[host == null]
host = player
1.2: color = selectAvailableColor() : Color
1.3: setColor(color)
1.4: score = getName()
score.incrementAndGet()
1.5: initialPosition = selectAvailableInitialPosition() : int[]
1.6: setX(initialPosition[0])
1.7: setY(initialPosition[1])
1.8: cell = getCell(initialPosition[0], initialPosition[1]) : Cell
1.9: setPaintedBy(player)


: Usuario
game : Game
player : Player
cell : Cell
wildcard : Wildcard
1: movePlayer(playerName, x, y) : void
LP
GAME_FINISHED
NONE_EXISTENT_PLAYER
OFF_BOARD
INVALID_MOVE
FROZEN_PLAYER
OCCUPIED_BOX
1.1: validateMove(playerName, x, y) : void
1.1.1: nameExist = nameExist(playerName) : boolean
1.1.2: insideBoard = isInsideBoard(x, y) : boolean
player = players.get(playerName)
1.1.3: invalidMove = isInvalidMove(player, x, y) : boolean
1.1.4: isFrozen = isFrozen() : boolean
opt
[!finishedGame && nameExist && isInsideBoard && !isInvalidMove && !isFrozen]
1.2: decrementRemainingMoves() : void
opt
[remainingMoves % 150 == 0]
1.2.1: addRandomWildcard() : void
1.2.1.1: cell = getRandomCell() : Cell
<<create>>
1.2.1.2: randomWildcard = CreateMessage()
randomWildcard : Wildcard
1.2.1.3: setWildcard(wildcard)
opt
[remainingMoves == 0]
1.2.2: endGame() : void
opt
[remainingFrozenMoves > 0 && --remainingFrozenMoves <= 0]
loop [player : players]
1.2.3: isFrozen = isFrozen() : boolean
opt
[isFrozen]
1.2.4: unfreeze() : void
1.3: cell = getCell(x, y) : Cell
1.4: isPlayerOnCell = isPlayerOnCell() : boolean
opt
[!isPlayerOnCell]
1.5: paint(player) : void
1.6: setX(x)
1.7: setY(y)
1.8: wildcard = getWildcard()
opt
[wildcard != null]
1.9: activate(game, player) : void
1.10: setWildcard(null)

3.3. Vista física

3.3.1 Diagrama de despliegue



3.4. Vista de desarrollo

3.4.1 Diagrama de Componentes