Desarrollo de Aplicaciones Distribuidas: Registrador de juegos

Rafael Gálvez-Cañero, Andreas Gerstmayr

Iteración 3 - 9 de Abril de 2015

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Datos generales

1.1. Miembros del grupo

Apellidos	Nombre	Correo-e	Grupo
Gálvez-Cañero	Rafael	galvesband@gmail.com	18
Gerstmayr	Andreas	andreas.gerstmayr@gmail.com	18

Cuadro 1.1: Miembros del grupo

1.2. Descripción del sistema

- **Tipo de sistema distribuido**: Sistema de información.
- **Nombre del proyecto**: Plataforma de juegos, Game Registry.
- **Breve descripción**: Sub-sistema para registrar sesiones de juego e información asociada.

1.2.1. Funcionalidad observable

- Registrar el inicio y el término de todas las sesiones de juego.
- Visualizar el historial de juegos.

1.2.2. Servicios ofrecidos

- Servicio de Registro: Capacidad de aceptar la información de una sesión de juego (juego ID, jugador ID y fecha de inicio y término).
- Servicio de Historial: Ofrece métodos para consultar el historial de sesiones.

1.2.3. Servicios demandados

- Servicio de Juego: avisar de término de una sesión de juego
- Servicio de Juego: recibir el título de un juego
- Servicio de Perfil: rebibir el nombre de un jugador

1.3. Direcciones de descarga y planificación

Código fuente	`https://repositorio.informatica.us.es/svn/lq3vqrtzfnh2nx9yhpk`
Planificación temporal	
Iteración 1	24/02/2015
Iteración 2	03/03/2015
Iteración 3	26/03/2015
Iteración 4	7/04/2015
Iteración 5	28/04/2015
Iteración 6	12/05/2015
Iteración 7	26/05/2015
Entrega Final	02/06/2015

Cuadro 1.2: Datos generales del trabajo en grupo

1.4. Seguimiento

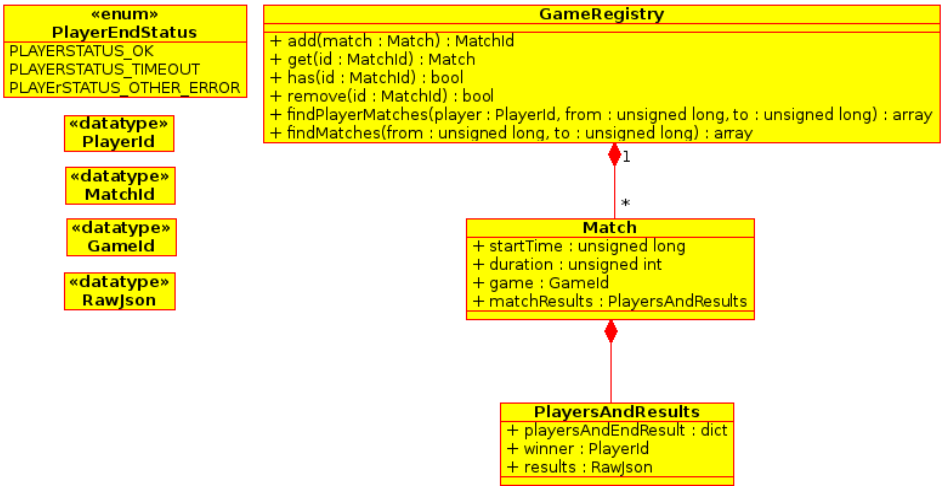
	Iteración									
Estudiante	1	2	3	4	5	6	7	Final	Total	Pond.
Rafael Gálvez-Cañero	5	5	5	5	-	-	-	-	**20**	1
Andreas Gerstmayr	5	5	5	5	-	-	-	-	**20**	1
Total	10	10	10	10	0	0	0			

Cuadro 1.3: Tabla de seguimiento

Capítulo 2

Modelado

2.1. Análisis del sistema



2.2. Arquitectura del sistema

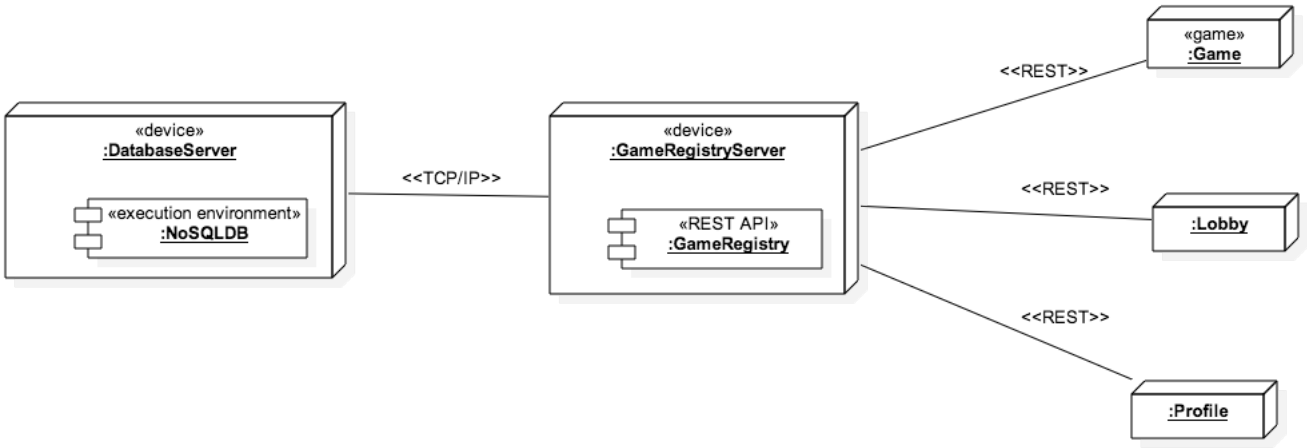


Figura 2.1: Modelo de despliegue del sistema

Capítulo 3

Iteración 3

3.1. Objetivos de iteración

- Integración de Gradle
- Servidor dockerizado
- Estructura inicial del Cliente vertx.

3.2. Gradle

Gradle es un gestor de construcción especialmente indicado para proyectos Java y con soporte para *Groovy*, *Vertx* y *Maven*.

Ha sido integrado mediante el wrapper `gradlew` que permite utilizar Gradle sin instalarlo de forma global en el sistema de desarrollo. La primera vez que se lanza descargará todas las bibliotecas necesarias.

En el archivo **Readme.md** hay información básica sobre como construir el proyecto. Algunos comandos útiles:

- Para **construir** el proyecto: `$ ./gradlew clean modZip`
- Para **lanzar** el servidor en la máquina local: `$ ./gradlew runMod -i`
- Para lanzar los **tests**: `$ ./gradlew clean test`
- Para preparar el proyecto para un **entorno de desarrollo**:
  - Eclipse: `$ ./gradlew eclipse`
  - IDEA: `$ ./gradlew idea`

3.3. Dockerización

Docker es una tecnología que permite utilizar contenedores sobre *Linux* para ejecutar procesos de forma aislada y con un runtime reproducible.

Nuestro proyecto proporciona un archivo **Dockerfile** con las instrucciones necesarias para construir el contenedor de la aplicación. Además, el archivo **Readme.md** contiene información sobre el procedimiento para lanzar el proyecto en *Docker*.

Capítulo 4

Iteración 4

4.1. Objetivos de iteración

- Integración de MongoDB (como contenedor docker)
- Cliente más avanzado. Servidor estructurado en Servicio / Controlador.

4.2. Integración de MongoDB

MongoDB es una base de datos no relacional (*NoSQL*) basada en documentos y que utiliza *JSon* como formato de intercambio de información.

La integración se ha realizado mediante un contenedor *Docker* de forma que tanto el desarrollo como el despliegue es fácilmente reproducible.

El servidor GameRegistry necesita una instancia de *MongoDB* funcionando para funcionar. La integración, realizada mediante el módulo *Vertx* llamado *MongoDB persistor*, depende de un archivo de configuración para obtener los datos relativos a la conexión con *MongoDB*. Cambiando los parámetros de este archivo de configuración es posible hacer que el servidor GameRegistry se conecte a una u otra instancia de *MongoDB*.

Hay que tener en cuenta que el contenedor de *MongoDB* utiliza un *Volumen* (un espacio de almacenamiento "externo" al contenedor que persiste de un lanzamiento a otro). A menudo es conveniente configurar el lanzamiento del contenedor de *MongoDB* de forma que dicho volumen corresponda a una carpeta del host:

```
$ docker run -v [local path]:/data/db/ --name mongo-server mongo:3.0.1
```

donde `local path` es la ruta a una carpeta del host que será utilizada por *Docker* para montar la carpeta `/data/db` en el contenedor de *Docker*. De este modo los archivos relativos a la base de datos de *MongoDB* quedan fuera del contenedor y es posible examinarlos de forma externa o hacer copias de seguridad fácilmente.

4.2.1. MongoDB y desarrollo

Para desarrollar el servidor GameRegistry o lanzarlo de forma local fuera de un entorno de contenedores *Docker* tan solo es necesario obtener una instancia de *MongoDB* a la que se pueda conectar y un archivo de configuración con la información necesaria para la conexión.

En el archivo **Readme.md** hay información básica sobre cómo conseguir una instancia de MongoDB utilizando *Docker*, que resulta a menudo más conveniente que instalar una instancia local del mismo.

Hay que tener en cuenta que los contenedores de *Docker* por defecto no exponen los puertos al host. Las instrucciones básicas para lanzar *MongoDB* como contenedor para desarrollo serían:

- Descargar el contenedor (sólo la primera vez): `$ docker pull mongo:3.0.1`
- Lanzar la instancia de *MongoDB* exponiendo el puerto al host: `$ docker run -P mongo:3.0.1`
- Modificar la configuración de GameRegistry si es necesario en el archivo *conf.json*
- Lanzar el servidor GameRegistry: `$ ./gradlew runMod -i`

4.2.2. MongoDB y despliegue

El *Dockerfile* que describe cómo construir el contenedor de nuestro servidor GameRegistry ha sido actualizado para integrar un archivo de configuración separado dedicado al contenedor llamado *conf-docker.json*. Este archivo contiene la configuración necesaria (mínima) para poder conectar con un servidor *MongoDB* que corre en una máquina llamada `mongo-server`. Esto resulta conveniente en un entorno *Docker* enlazando ambos contenedores. Por ejemplo:

- Descargar el contenedor de *MongoDB* (sólo la primera vez): `$ docker pull mongo:3.0.1`
- Lanzar la instancia de *MongoDB* sin exponer puertos al host y nombrando la instancia del contenedor: `$ docker run --name mongo-server mongo:3.0.1`
- Construir el contenedor para nuestro servidor: `$ docker build -t distributedsystems/gamere .`
- Lanzar el contenedor de GameRegistry enlazándolo con el de la instancia de *Docker* de forma que compartan la pila de red (y puedan así comunicarse), exponiendo el puerto de GameRegistry en el host: `$ docker run -p 8080:8080 --link mongo-server:mongo-server distributedsystems/gameregistry`

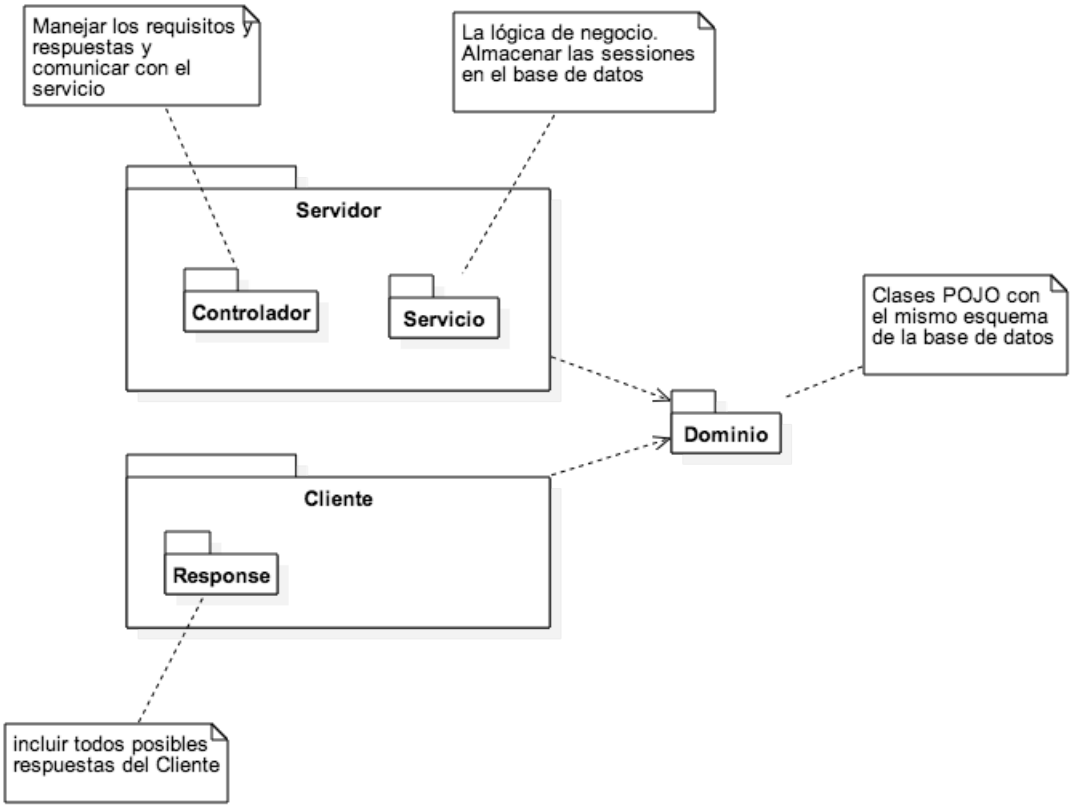
4.2.3. Distinción de dominio, controlador y servicio

Dominio
Clases POJO con el mismo esquema de la base de datos.

Controlador
Manejar los requisitos y respuestas y comunicar con el servicio.

Servicio
La lógica de negocio. Almacenar las sessiones en el base de datos.

4.3. Diagrama de paquetes



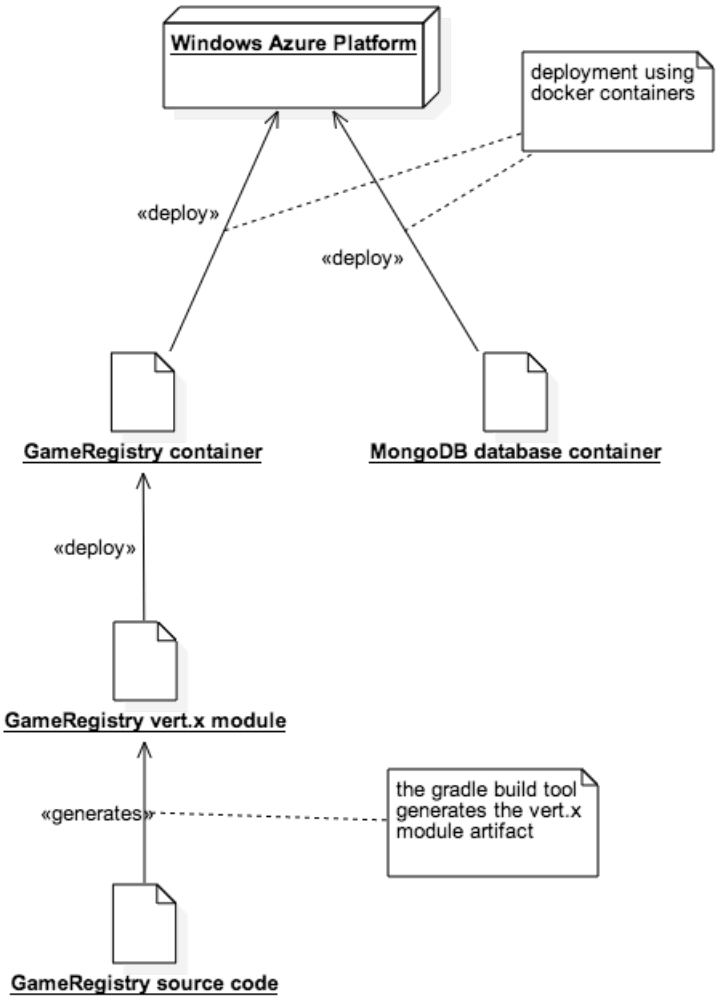
Capítulo 5

Iteracion 5

5.1. Objetivos de iteración

- Implementación inicial de API
- Primeros tests.
- Despliegue Azure plataforma
- ¿Integración contínua?

5.2. Diagrama de despliegue



Capítulo 6

Iteracion 6

6.1. Objetivos de iteración

Final testing.

Capítulo 7

Iteracion 7

7.1. Objetivos de iteración

Subir a repositorio Maven.