Documento de Arquitectura de Software

Grupo [X]

Agustin Peraza

Mauricio Camacho

Damian Mesa

Santiago Sellanes

Contenido:

[1 Introducción. 3](#_Toc5092993)

[2 Vista de Casos de Uso 4](#_Toc5092994)

[2.1 Diagrama de Casos de Uso Críticos 4](#_Toc5092995)

[2.2 Actores 5](#_Toc5092996)

[2.3 Especificación de Casos de Uso Críticos 6](#_Toc5092997)

[3 Vista Lógica 7](#_Toc5092998)

[3.1 Estilo Arquitectónico. 7](#_Toc5092999)

[3.2 Subsistemas. 8](#_Toc5093000)

[3.3 Diagramas de Interacción 9](#_Toc5093001)

[4 Vista de Distribución (Deployment). 11](#_Toc5093002)

[4.1 [Escenario 1] 12](#_Toc5093003)

[4.2 [Escenario 2] 12](#_Toc5093004)

[5 Vista de Implementación. 13](#_Toc5093005)

[6 Referencias 14](#_Toc5093006)

# Introducción.

En el último año la industria de los videojuegos tuvo gran crecimiento, en parte por la

pandemia que obliga a las personas a permanecer en sus casas y por otro lado la gran

variedad de juegos disponibles. En particular en los últimos años han surgido muchos juegos

desarrollados por estudios pequeños y desconocidos o incluso desarrollados por una sola

persona.

En función de esto la empresa “SteamIndie” decidió desarrollar una plataforma web que

permita a los desarrolladores ofrecer sus juegos, a la vez que permite a los jugadores comprar

juegos y formar una comunidad con sus pares.

La plataforma ofrecerá a los usuarios la posibilidad de crear su perfil y poder interactuar con

otros usuarios. A su vez, estos podrán comprar juegos, de los cuales recibirán un enlace de

descarga, y mantener un registro (biblioteca) de los juegos comprados. Como parte del sentido

de comunidad, los jugadores podrán calificar los videojuegos que hayan comprado y dejar un

comentario sobre el juego que puede ser visualizado por otros jugadores.

A la hora de ofrecer los videojuegos, la plataforma ofrece un servicio de búsqueda que permite

buscar juegos por su nombre, por categoría o por tags personalizados de los usuarios. Los

usuarios contarán con su propia billetera en el sitio que utilizaran para comprar los videojuegos.

Esta podrán cargarla a través de tarjetas de crédito, Paypal o Google Wallet.

Para los creadores de videojuegos la empresa ofrece la posibilidad de publicar sus juegos. Las

publicaciones pueden incluir la descripción del juego, imágenes y videos promocionales.

También ofrecen la posibilidad de suscribirse a programas de ofertas o giveaways ofrecidos por

la plataforma.

La plataforma cobrará a los creadores de videojuegos un porcentaje de sus ventas, en principio

este porcentaje está estipulado en 10%.

El sistema deberá contar con dos módulos. Por un lado el backoffice para administración

general de la plataforma por parte de personal de SteamIndie. Y por otro lado un frontoffice

que permita a los usuarios (jugadores y creadores) acceder al contenido. A su vez interesa

que todas las funcionalidades del frontoffice se puedan acceder desde cualquier dispositivo.

A futuro está planeado el desarrollo de una aplicación móvil.

# Vista de Casos de Uso

[

Los elementos de las cuatro vistas desde las que se analiza la arquitectura son “ejercitados” por un pequeño subconjunto de casos de uso (o escenarios de esos casos de uso) que llamamos casos de uso críticos. Este subconjunto se debe elegir cuidadosamente, utilizando principalmente dos criterios:

* Que intervengan el mayor número de componentes arquitectónicos posibles
* Que intervengan los componentes que se vislumbran como críticos o más complejos.

]

## Diagrama de Casos de Uso Críticos

[

En esta sección se utilizan los Diagramas de Casos de Uso de UML para mostrar los casos de uso críticos para la arquitectura. Los elementos a utilizar en estos diagramas son:

|  |  |
| --- | --- |
| Caso de Uso | Secuencia de acciones con un fin específico (Ej. Enroll in University) |
| Actor | Persona, organización o sistema externo que interactúa con el sistema (Ej. Student) |
| Asociación | Relación entre un Caso de Uso y un Actor que participa en él. |
| Frontera del Sistema | La Frontera del Sistema se indica con un rectángulo que encierra los Casos de Uso. |
|  |  |

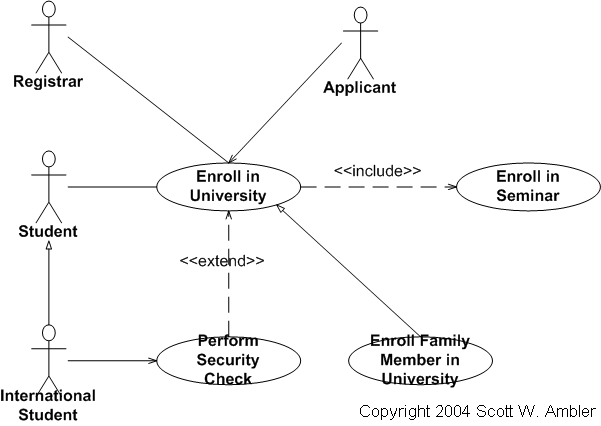


Diagrama de Ejemplo 1 (Ver [2])

]

## Actores

[

En esta sección se realiza una descripción de cada actor perteneciente al diagrama anterior.

]

### Visitante

Personas que ingresan a la página por primera vez y no están registradas.

### Jugador

Usuarios registrados a la página que ingresan principalmente para comprar juegos aparte de realizar otras acciones dentro del sistema.

2.2.3 **Creador**

Usuarios registrados a la página que ingresan principalmente para vender juegos aparte de realizar otras acciones dentro del sistema.

### Administrador

Se encarga de gestionar el sistema.

## Especificación de Casos de Uso Críticos

[

En esta sección se incluye un ítem por cada caso de uso mostrado en el diagrama anterior. De cada caso de uso debe brindarse su descripción, así como uno o más flujos de eventos que detallen la interacción actor-sistema para cada escenario relevante del caso de uso. Opcionalmente, pueden especificarse pre y postcondiciones para cada caso. Las precondiciones son las condiciones que deben cumplirse para que el flujo del caso de uso pueda realizarse. Las postcondiciones son las condiciones que se cumplen al terminar el flujo del caso de uso (modificaciones en el estado del sistema).

]

### [Caso de Uso Crítico 1]

##### Descripción

##### Pre-condiciones

##### Flujo de Eventos

|  |  |
| --- | --- |
| Acción de Actor | Respuesta del Sistema |
| 1 - |  |
|  | 2 - |
| 3 - |  |

##### Post-condiciones

### [Caso de Uso Crítico 2]

# Vista Lógica

[

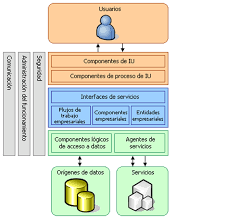
La vista lógica permite describir el sistema en base a abstracciones fundamentales del diseño orientado a objetos para dar soporte a los requerimientos funcionales. En un enfoque top-down, se comienza por descomponer el sistema en un conjunto de subsistemas “grandes”, como ser las “capas” (*layers*) si se utiliza una arquitectura en capas, y a partir de ellos se realizan sucesivos refinamientos hasta llegar a las unidades lógicas más pequeñas.

]

## Estilo Arquitectónico.

[

En esta sección se describe el estilo arquitectónico elegido para el sistema (capas estricto, capas no estricto, etc.). En el lenguaje de modelado UML 2, se utiliza el Diagrama de Componentes para representar componentes lógicos, creados en tiempo de diseño. Se aconseja utilizar diagramas de componentes para representar subsistemas (considerar el clasificador <<subsystem>> como alternativa a <<component>>.



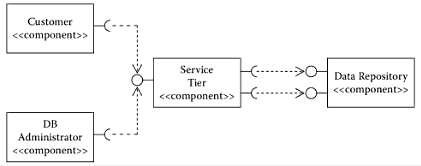


Diagrama de Ejemplo 2 (Ver [4])

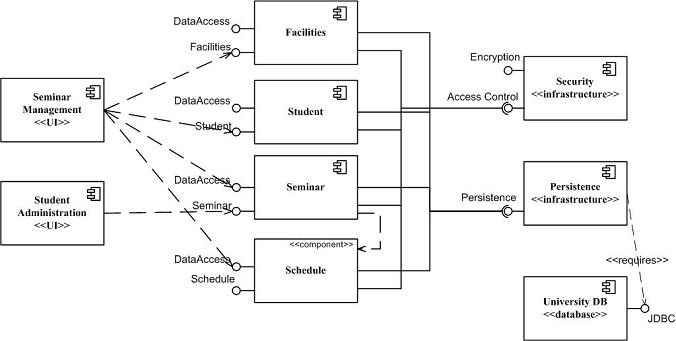


Diagrama de Ejemplo 3 (Ver [4])

]

# Vista de Distribución (Deployment).

[

En la vista de distribución o deployment se plantean uno o varios escenarios de distribución de los componentes en *tiers* o nodos. Se utiliza el Diagrama de Deployment de UML que permite mostrar la estructura del escenario en base a nodos, conexiones entre nodos y componentes de cada nodo. Un nodo puede ser un elemento de hardware (un host, un router, etc.) o un elemento de software (un servidor de aplicaciones JEE, por ejemplo). Para distinguir los nodos físicos (hardware) se utiliza el estereotipo <<device>>.

Opcionalmente, pueden describirse los nodos y conexiones presentes. De los nodos pueden especificarse requerimientos de software y hardware (sistema operativo, procesador, memoria, almacenamiento secundario, etc.) y de las conexiones los protocolos de comunicación, ancho de banda, etc.

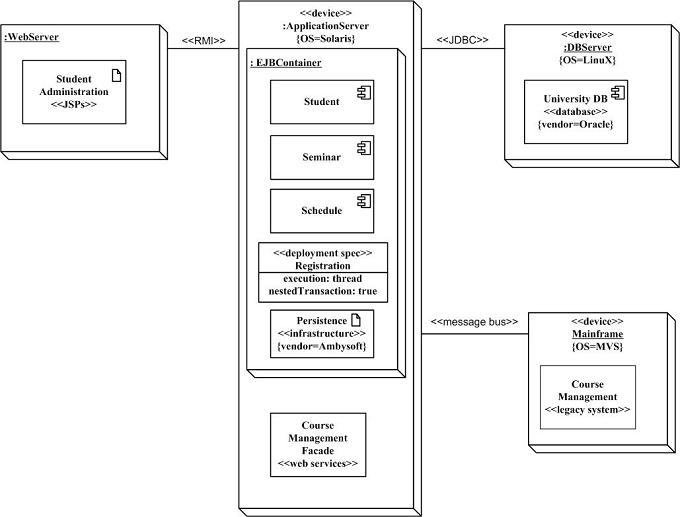


Diagrama de Ejemplo 7 (Diagrama de Deployment con cuatro nodos, ver [5])

]

# Vista de Implementación.

[

La vista de implementación se focaliza en los componentes en tiempo de ejecución que forman el sistema (ejecutables, archivos de clases, bibliotecas, frameworks, etc.), que son la implementación de los componentes lógicos (provenientes de Diseño).

Dentro de esta vista, interesa mostrar las dependencias entre componentes implementados (utilizando los “artefactos”de UML).

También pueden mostrarse la distribución de los artefactos en los nodos.

En ambos casos se utiliza el Diagrama de Deployment de UML.

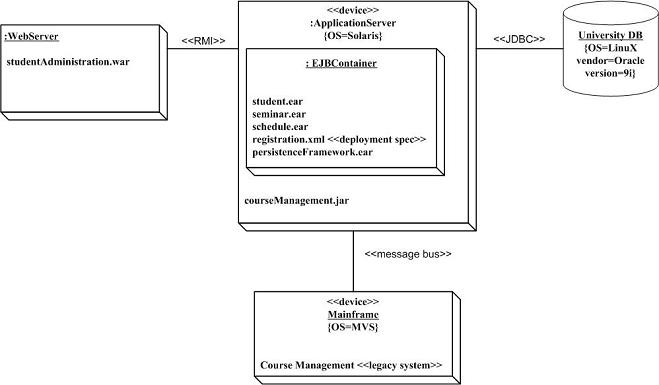


Diagrama de Ejemplo 10 (Ver [6])

]

# Referencias

1. Agile Models Distilled: Potential Artifacts for Agile Modeling  
   <http://www.agilemodeling.com/artifacts/>
2. Agile Modeling. UML 2 Use Case Diagrams  
   <http://www.agilemodeling.com/artifacts/useCaseDiagram.htm>
3. Agile Modeling. UML 2 Sequence Diagrams  
   <http://www.agilemodeling.com/artifacts/sequenceDiagram.htm>
4. Agile Modeling. UML 2 Component Diagrams  
   <http://www.agilemodeling.com/artifacts/componentDiagram.htm>
5. Agile Modeling. UML 2 Deployment Diagrams  
   <http://www.agilemodeling.com/artifacts/deploymentDiagram.htm>
6. Deployed Software: Artifacts  
   <http://codeidol.com/other/learnuml2/Modeling-Your-Deployed-System-Deployment-Diagrams/Deployed-Software-Artifacts/>
7. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. N. Rozanski, E. Woods. Addison-Wesley, 2005
8. Software Architecture in practice, Second Edition. L. Bass, P. Clemens, R. Kazman. Addison-Wesley, 2003
9. An Introduction to Software Architecture. D. Garlan, M. Shaw. 1994. <http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf>
10. Architectural Blueprints — The “4+1” View Model of Software Architecture. Kruchten, Philippe. 1995. <http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>
11. Software Architecture Links. Bredemeyer Consulting.  
    <http://www.bredemeyer.com/links.htm>