<Company Name>

Ultimate Tic Tac Toe
Software Architecture Document

Version 1.5

Revision History

Date	Version	Description	Author
03/03/15	1.0	Primera entrega del documento de arquitectura de software (1, 3, 4, 5, 10).	David Puga Mendivil A01332391 Dennis Omar Lugo A01214271 Cynthia Garcia Velasco A01332329
12/03/15	1.1	Segunda entrega del documento de arquitectura de software (6 y 8 (Overview)).	David Puga Mendivil A01332391 Dennis Omar Lugo A01214271 Cynthia Garcia Velasco A01332329
20/03/15	1.2	Tercera entrega del documento de arquitectura(11 y capa de presentación 8.1 y 8.2)	David Puga Mendivil A01332391 Dennis Omar Lugo A01214271 Cynthia Garcia Velasco A01332329
27/03/15	1.3	Cuarta entrega del documento de arquitectura (capa de negocio)	David Puga Mendivil A01332391 Dennis Omar Lugo A01214271 Cynthia Garcia Velasco A01332329
17/04/15	1.4	Quinta entrega del documento de arquitectura (PoC)	David Puga Mendivil A01332391 Dennis Omar Lugo A01214271 Cynthia Garcia Velasco A01332329
21/04/15	1.5	Sexta entrega del documento de arquitectura (capa integración acceso a datos, secciones 2, 7, 9)	David Puga Mendivil A01332391 Dennis Omar Lugo A01214271 Cynthia Garcia Velasco A01332329

Table of Contents

Software Architecture Document

1. Introduction

Este documento proporciona una visión general de la arquitectura de un sistema web desarrollado por los miembros del equipo. En este documento se describen las bases que los miembros usarán para la creación de una aplicación web, que estará basada en el clásico juego de gato, llamada Ultimate Tic Tac Toe. Este proyecto ayudará a los miembros a poner en práctica sus conocimientos para desarrollar el sistema previamente descrito.

El documento proporciona una descripción de alto nivel detallando los objetivos que tiene el desarrollo del proyecto, así como los casos de uso, capas, paquetes y subsistemas, que serán utilizados. De igual forma describe las restricciones y requerimientos necesarios para que la aplicación funcione de forma adecuada.

1.1 Purpose

La realización de nuestro proyecto tiene como objetivo demostrar los conocimientos adquiridos a lo largo del semestre en la materia de Diseño y Arquitectura de Software, aplicando los distintos tipos de patrones Gof en una aplicación web basada en el clásico juego de gato. A lo largo de este documento se describen las partes relacionadas con la estructura, creación e implementación del este sistema.

1.2 Scope

Este documento de arquitectura de software aplica al sistema web Ultimate Tic Tac Toe, el cuál será desarrollado a lo largo del semestre Enero-Mayo 2015. Esta aplicación estará basada en el clásico juego de gato para dos jugadores donde se tendrán nueve subtableros de gato que representarán cada casilla de un tablero de gato general. Para poder ganar la partida el jugador tendrá que ganar los respectivos subtableros del gato general que formen las líneas horizontales, verticales y diagonales, así como en el juego de gato clásico.

1.3 Definitions, Acronyms, and Abbreviations

Web	Conjunto de información que se encuentra en una dirección determinada de la web.
Servlets	Clase en el lenguaje de programación Java, utilizada para ampliar las capacidades de un servidor.
Java	Máquina virtual que usa Java Virtual Machine la cual contiene bibliotecas de la plataforma.
Base de datos	Programa capaz de almacenar gran cantidad de datos, relacionados y estructurados que pueden ser consultados rápidamente.
MySql	Sistema de gestión de base de datos relacional, multihilo y multiusuario.
Gof	Gang of Four. Se consideran los patrones de diseño fundamentales para orientación a objetos.
Servidor	Aplicación en ejecución capaz de atender las peticiones de un cliente y devolver una respuesta en concordancia.
Patrones de diseño	Son la base de las soluciones a problemas comunes en el desarrollo de software. Brindan una solución probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares.

JAR	Tipo de archivo que permite ejecutar aplicaciones escritas en el lenguaje Java.

1.4 References

- Diseño de Patrones- http://www.tutorialspoint.com/design_pattern/index.htm
- MySQL (Drivers) - http://dev.mysql.com/doc/connector-j/en/connector-j-usagenotes-connect-drivermanager.html
- JPA- http://www.tutorialspoint.com/jpa/
- JDBC- http://www.tutorialspoint.com/jdbc/
- Hibernate- http://www.tutorialspoint.com/hibernate/index.htm
- myBatis- http://www.tutorialspoint.com/ibatis/

1.5 Overview

Este documento de arquitectura de software se divide en once secciones dentro de las cuales se describen las partes que conforman la arquitectura de la aplicación Ultimate Tic Tac Toe.

- En la primera sección se describen los objetivos de la creación y desarrollo de este documento, así como de la aplicación misma. De igual forma se describen las definiciones y referencias que serán usadas y consultadas a lo largo del desarrollo de este sistema.
- En la segunda sección se describe la representación arquitectónica del sistema, mencionando las vistas necesarias para el sistema (Use-Case**,** Logical**,** Process**,** Deployment**,** and Implementation Views), así como el modelo utilizado por cada una de éstas.
- En la tercera sección se describen los objetivos que se pretenden alcanzar el sistema, así como sus correspondientes restricciones y requerimientos.
- En la cuarta sección se representará el modelo de casos de uso donde se describirá el comportamiento del sistema.
- En la quinta sección se describe la parte lógica del sistema, la funcionalidad que será proporcionada a los usuarios. Aquí se describen las partes más significantes de modelo, la forma en la que se encuentra compuesto y sus paquetes. Así mismo se definirán las clases que componen cada paquete con sus respectivas relaciones, operaciones y responsabilidades.
- En la sexta sección se describe el sistema en procesos, la forma en que estos se encontrarán agrupados y la forma en la que se comunicarán entre ellos.
- En la séptima sección se describen las características físicas necesarias para que el sistema pueda ser implementado.
- En la octava sección se describe la estructura del modelo de implementación en capas y subsistemas, incluyendo los componentes más significativos de éste.
- En la novena sección de describe forma en la cual se mantendrá la persistencia de datos, en caso de que el sistema lo requiera.
- En la décima sección se describen las características más importantes que tendrán un impacto dentro de la arquitectura del sistema, así como sus respectivas restricciones.
- En la décimo novena sección se describe como la arquitectura de software contribuye a las funcionalidades del sistema y la importancia de estas funcionalidades para el sistema.

2. Architectural Representation

Este documento representa la arquitectura de la aplicación Ultima Tic Tac Toe en una serie de vistas basados en el Lenguaje Unificado de Modelado (UML):

Vistas	Representación

Objetivos y restricciones de arquitectura	Los objetivos de arquitectura se definen como una seria de principios expresados en un formato estándar proporcionando su nombre, descripción, argumentos e implicaciones. Las restricciones de arquitectura se enlistan y describen a lo largo de este documento.
Vista de casos de uso	Se representa mediante los diagramas UML de casos de uso.
Vista Lógica	Se representa mediante el diagrama UML de componentes donde se describen los componentes principales, así como sus relaciones.
Vista de Despliegue	Se representa mediante el diagrama UML de despliegue donde se describen los nodos computacionales, su configuración y como se conectan sus elementos.
Vista de Implementación	Se representa mediante los diagramas UML de clases donde se describen las principales clases, componentes y la forma en que están asociados.
Tamaño y Desempeño	Describen los volúmenes de datos y cifras de rendimiento esperados.
Calidad	Métricas de calidad esperados.

3. Architectural Goals and Constraint

Los requerimientos de nuestra aplicación web Ultimate Tic Tac Toe:

1. Contar con un navegador web (Preferentemente Chrome y Firefox).
2. Contar con una conexión a internet.
3. Tener java instalado.

Objetivos:

4. Funcionamiento completo de la aplicación web Ultimate Tic Tac Toe.
5. Uso de patrones de diseño Gof aprendidos en clase en la implementación del sistema.
6. Mantener la aplicación portable a través de los navegadores web.
7. Mantener la conexión entre los jugadores en un mismo juego.

Restricciones importantes a considerar del sistema:

8. Sólo soporta dos jugadores simultáneamente.
9. Tener instalado plugins de java.
10. Conexión a internet.

Estrategia de diseño e implementación:

11. Uso de patrones de diseño Gof (Gang of Four).
12. Uso metodología cliente-servidor para conexión del juego.

Herramientas de desarrollo:

13. Lenguaje de programación Java.
14. Ambiente de desarrollo Netbeans.
15. Java servlets para conexión entre computadoras.
16. JavaFX para diseño de interfaz gráfica.

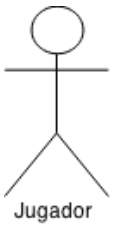
17. MySql para conexión a base de datos.

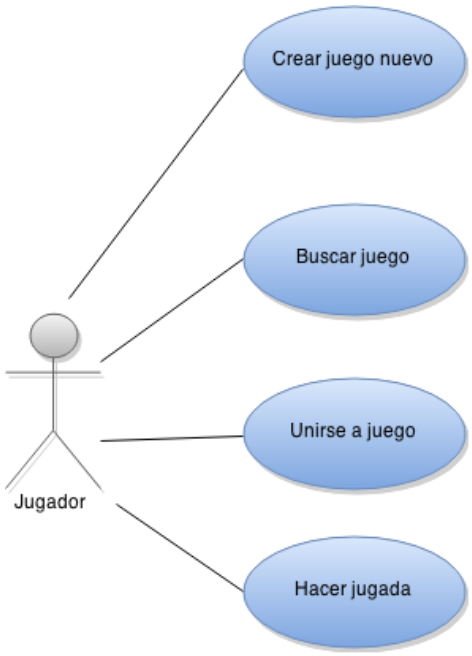
4. Use-Case View

Casos de uso:

18. Crear juego nuevo: este caso de uso describe como un usuario puede crear un nuevo juego de Ultimate Tic Tac Toe. El actor de este caso de uso es el jugador.
19. Buscar juego: este caso de uso describe como un usuario puede buscar un juego en el cuál puede participar. El actor de este caso de uso es el jugador.
20. Unirse a juego: este caso de uso describe como un usuario puede unirse a un juego existente, es decir, que haya sido iniciado por otro jugador, con la restricción de que sólo puede hacer dos jugadores en un solo juego. El actor de este caso de uso es el jugador.
21. Hacer jugada: este caso de uso describe como un usuario puede hacer una jugada, es decir, escoger entre X o O para colocar en algunas de las casillas del tablero. El actor de este caso de uso es el jugador.

Actor:





5. Logical View

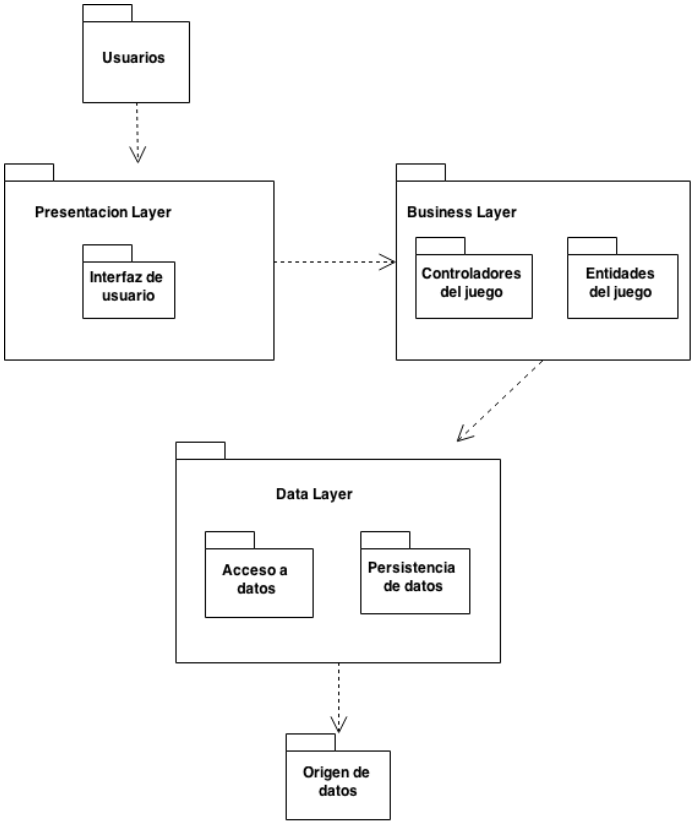
5.1 Overview

En esta sección se describen las clases más importantes del sistema, organizados en subsistemas y paquetes. De igual forma describe gráficamente los casos de uso más importantes del sistema. Se incluye también el

diagrama de las clases para exponer las relaciones más significativas entre las clases y los paquetes.

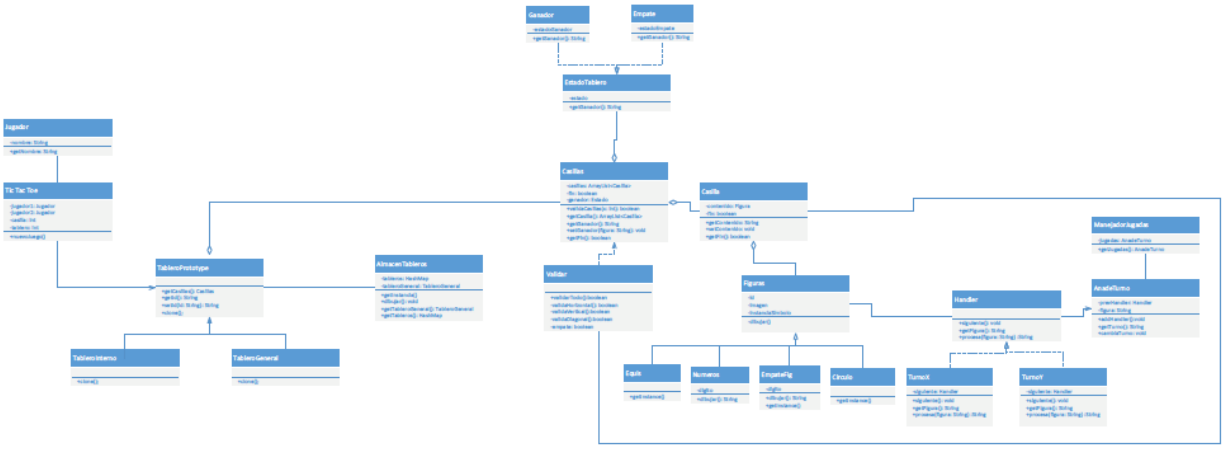
En esta vista lógica el sistema tiene tres capas principales: Presentación, Negocio y Datos.

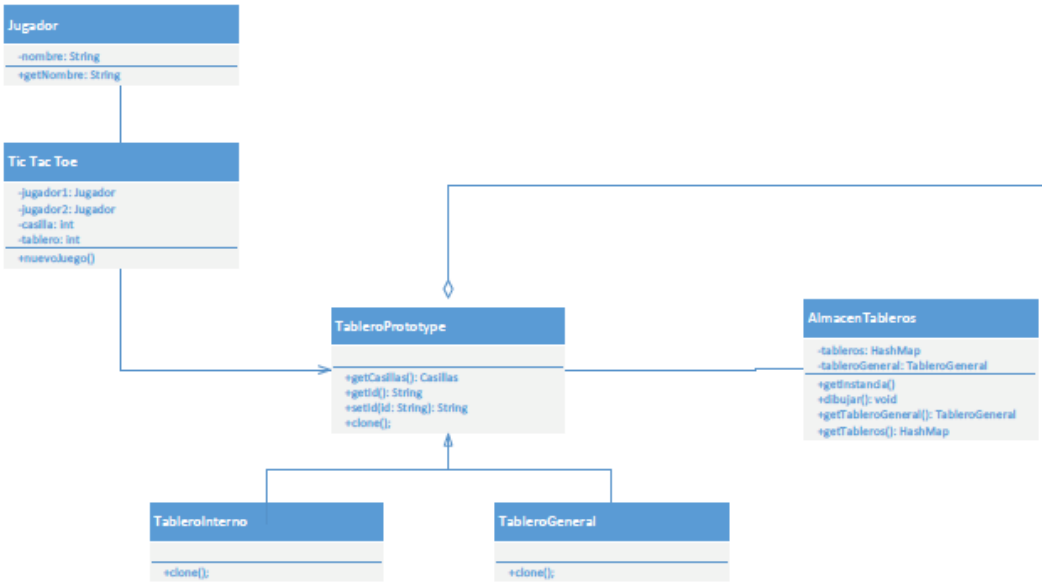
- Dentro de la capa de presentación se incluye el paquete de la interfaz de usuario, que representa la forma en la cual el actor, en este caso el jugador, interactuará con el sistema.
- Dentro de la capa de negocio se contienen las clases encargadas del control del sistema y las entidades que controlan.
- Dentro de la capa de negocio se incluyen los paquetes encargados de la interacción con la base de datos donde se almacenará la información necesaria para la persistencia de datos.



5.2 Architecturally Significant Design Packages

Diagrama de clases




Jugador
-nombre: String
+getNombre: String
Tic Tac Toe
-jugador1: Jugador
-jugador2: Jugador
-casilla: int
-tablero: int
+nuevoJuego()
TableroPrototype
+getCasillas(): Casillas
+getId(): String
+setId(id: String): String
+clone();
AlmacenTableros
-tableros: HashMap
-tableroGeneral: TableroGeneral
+getInstancia()
+dibujar(): void
+getTableroGeneral(): TableroGeneral
+getTableros(): HashMap
TableroInterno
+clone();
TableroGeneral
+clone();

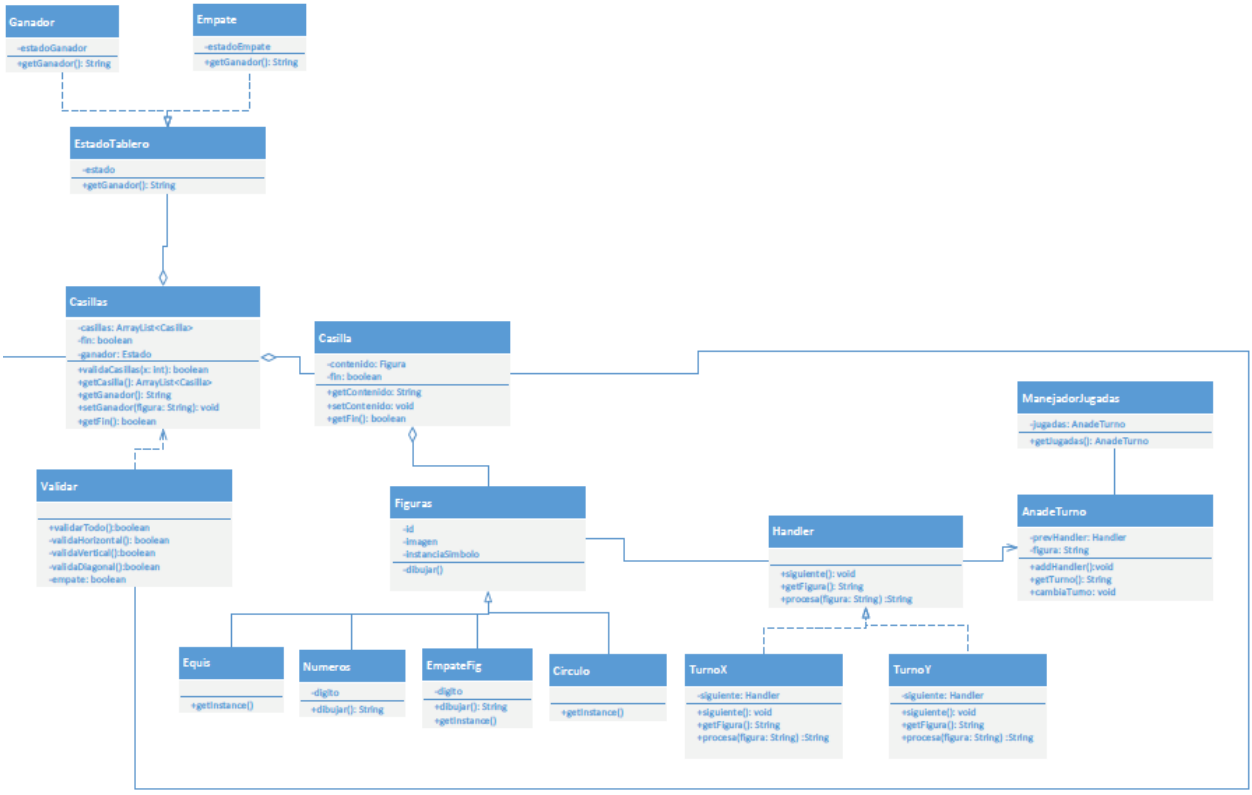
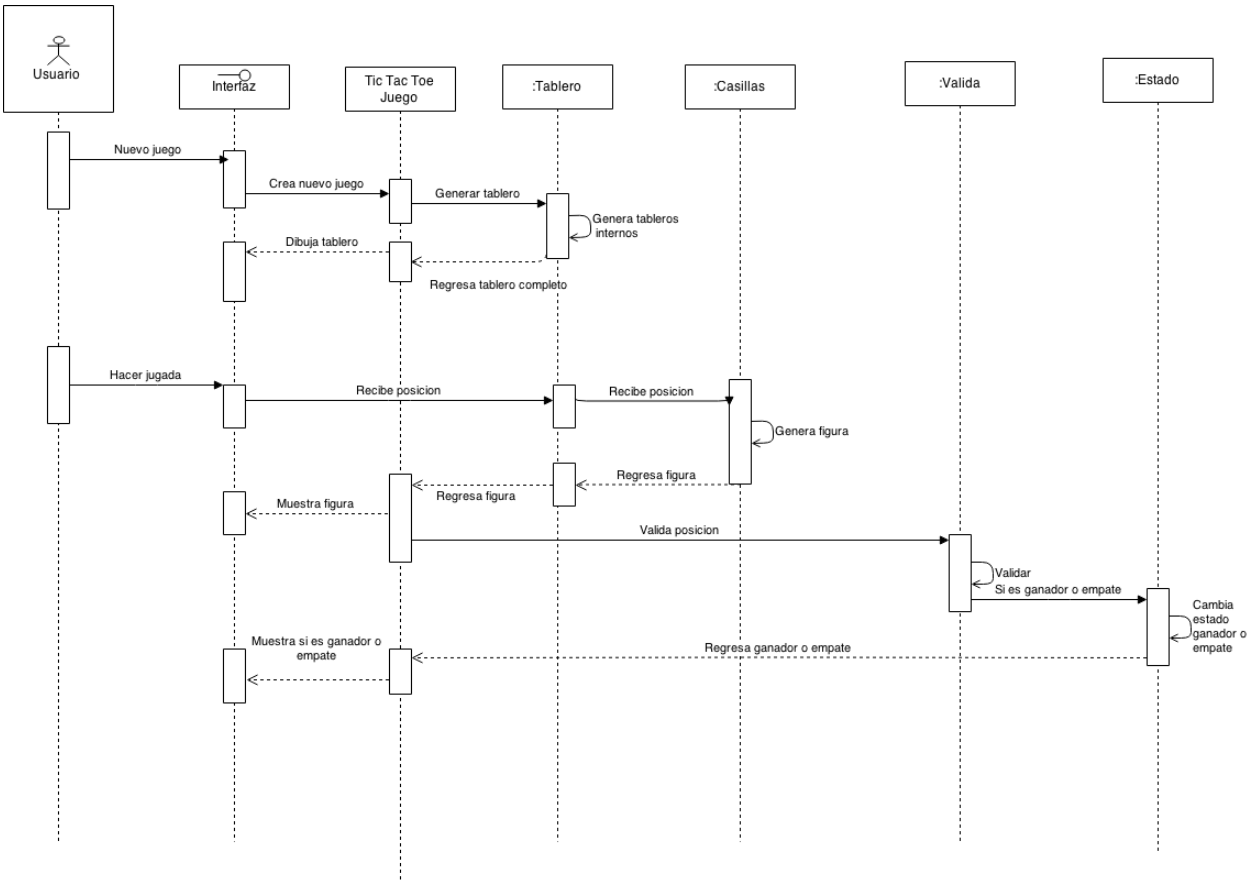
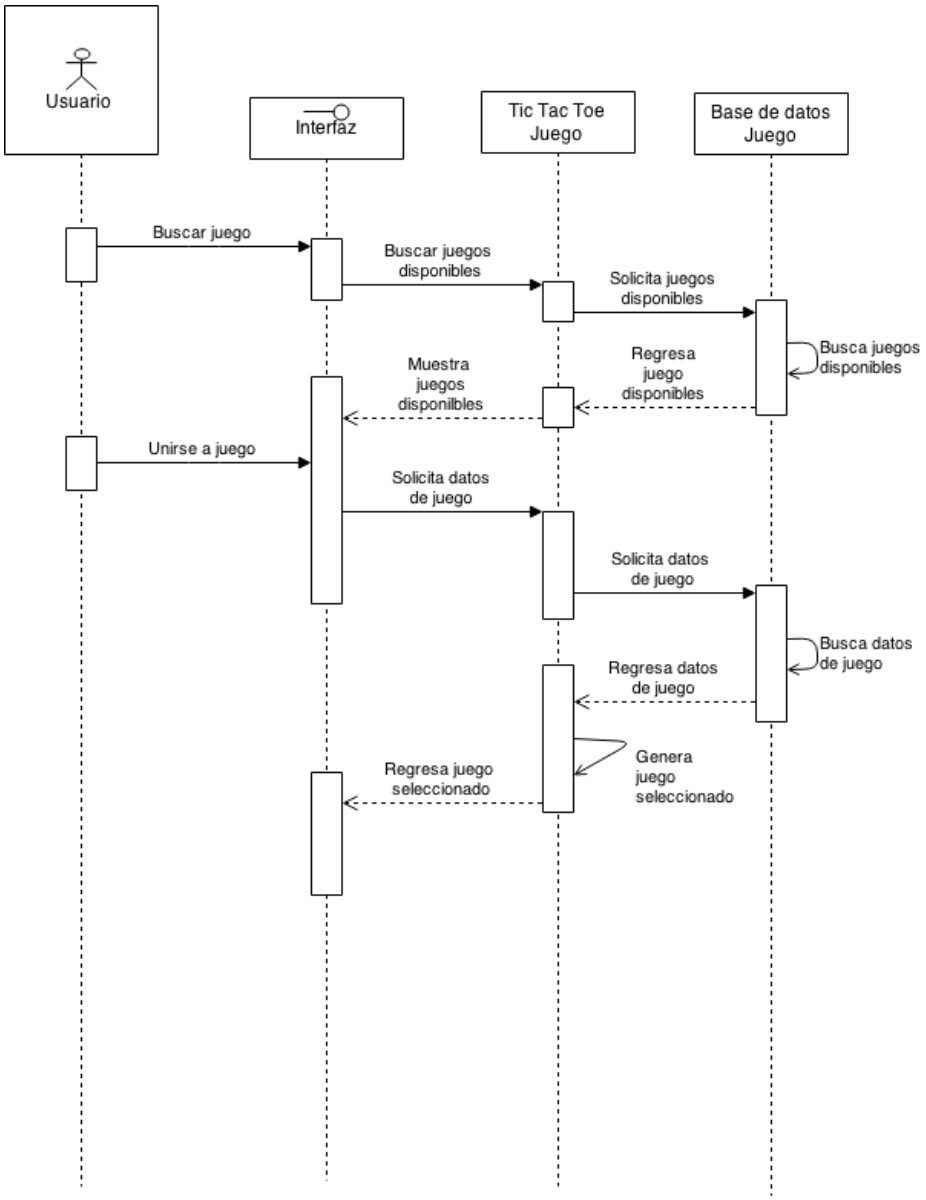


**Diagrama completo en anexo

5.3 Use-Case Realizations


Usuario
Interfaz
Tic Tac Toe Juego
:Tablero
:Casillas
:Valida
:Estado
Nuevo juego
Crea nuevo juego
Generar tablero
Genera tableros internos
Dibuja tablero
Regresa tablero completo
Hacer jugada
Recibe posicion
Recibe posicion
Genera figura
Regresa figura
Regresa figura
Muestra figura
Valida posicion
Validar
Si es ganador o empate
Cambia estado ganador o empate
Muestra si es ganador o empate
Regresa ganador o empate



6. Process View

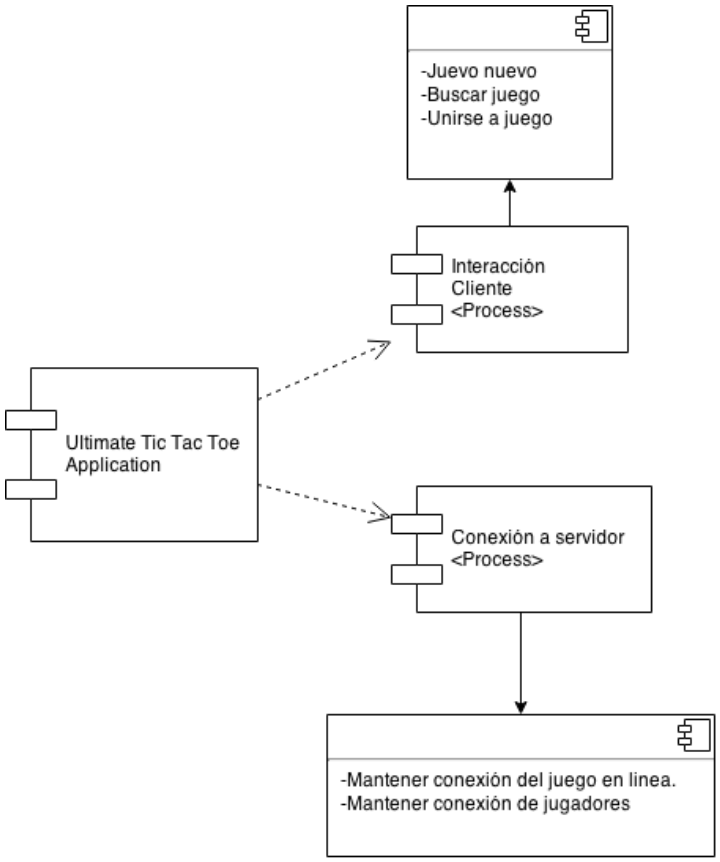
En esta sección se describen los procesos principales con los que contará la aplicación, así como la comunicación que se realizará entre cada uno de estos procesos.

-Interacción cliente: Este proceso se encargará de recibir toda interacción del jugador, es decir, recibirá toda información que sea ingresada por el usuario, incluyendo las acciones que desea realizar:

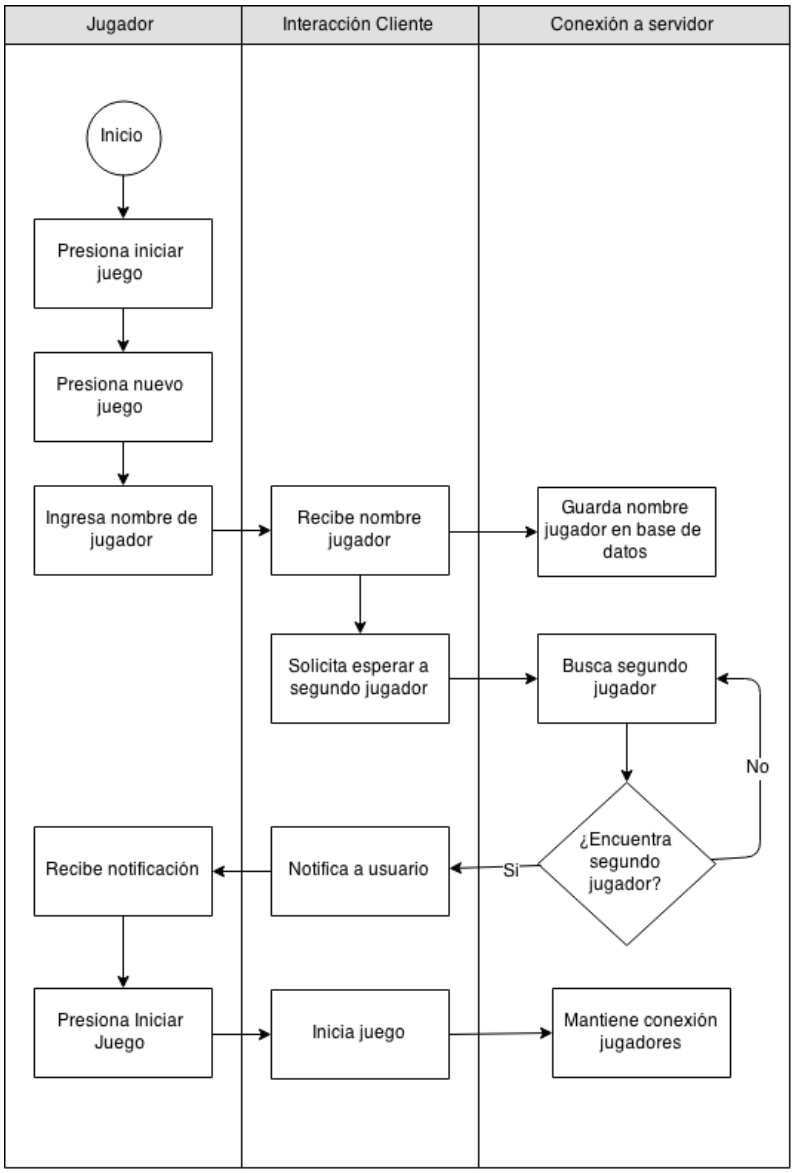
- Nuevo Juego: se crea un juego nuevo solicitando el nombre del jugador como referencia y será enviado a la base de datos.
- Buscar Juego: trae de la conexión al servidor cuales juegos se encuentran disponibles, es decir, cuales juegos solo cuentan con un jugador y están en espera del segundo.
- Unirse a Juego: conecta al jugador con el juego disponible que haya elegido el usuario.

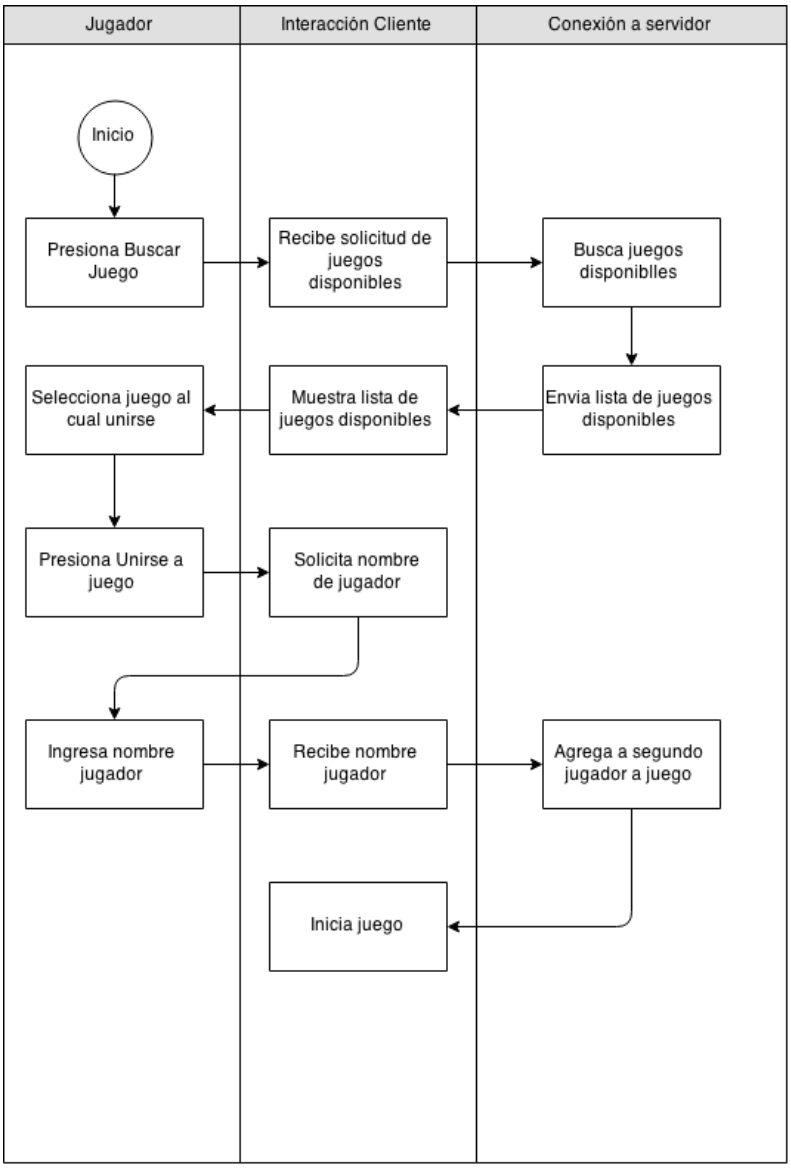
-Conexión a servidor: Este proceso se encargará de mantener la conexión del juego con los dos jugadores, y el servicio en línea del juego, de igual forma se conectará a la base de datos para obtener la información que requiera el juego.

Diagrama procesos:



Comunicación entre procesos:



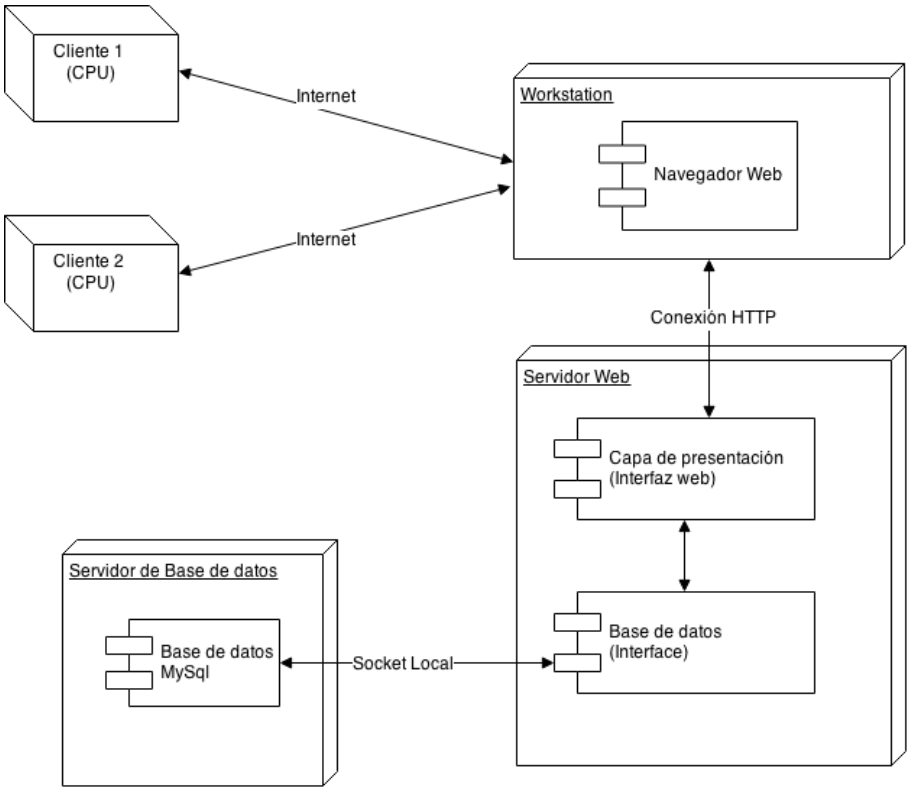


7. Deployment View

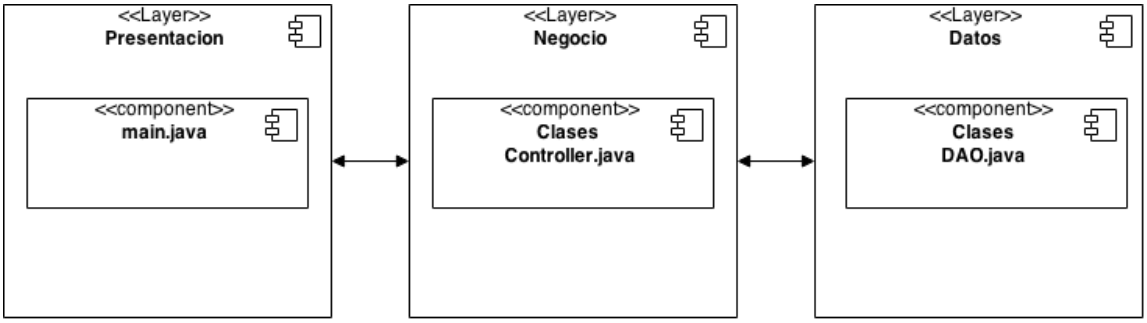
En esta sección se describe las configuraciones físicas en la que el software se implementa y ejecuta.

El servidor que será utilizado para montar la aplicación Ultimate Tic Tac Toe será en montado localmente en una computadora, la cual proporcionará la conexión a los clientes.

La máquina del cliente es cualquier dispositivo capaz de ejecutar un navegador Web (con compatibilidad con Java para los plugins) y la conexión se realizará a través de Internet.



8. Implementation View



Nuestro sistema estará dividido en 3 capas:

1. Presentación: esta capa contendrá a los componentes que tendrán una interacción directa con el usuario. En este caso, nuestro sistema tendrá al componente encargado de la interacción será nuestra interfaz gráfica descrita en el archivo main.java.
2. Negocio: esta capa tendrá los componentes encargados del control de la aplicación y de los elementos que serán presentados en la capa de presentación. En este caso, los componentes encargados del control de los elementos del juego serán las clases controller.java.
3. Datos: esta capa contendrá a los componentes encargados de almacenar la información obtenida de la base de datos. En este caso, nuestro sistema ocupará las clases DAO.java necesarias para acceder a la información almacenada en la base de datos.

8.1 Overview

Capa de presentación:

1. Tipo de cliente: El sistema será montado en un navegador web, a través del cual el jugador podrá interactuar con el sistema. Para que los jugadores utilicen la aplicación deberán tener los mínimos conocimiento en computación (manejo de mouse principalmente) para poder hacer uso de ésta.

2. Selección de tecnología: HTML, Java

3. Navegación e interfaz de usuario modular:
   El sistema contará con un total de 8 interfaces, las cuáles están divididas en las 3 secciones principales que conforman el sistema, inicio de juego, búsqueda de juego e instrucciones de juego. Cada una de estas interfaces será descrita a continuación:
   a. Menú principal: es la interfaz de inicio que contendrá la navegación a los demás componentes de la aplicación.
   b. Iniciar juego: es la interfaz que solicitará el nombre del jugador para crear un nuevo juego.
   c. Espera de jugador: esta interfaz servirá de transición al nuevo juego, servirá para indicarle al usuario que se está buscando un contrincante para la partida.
   d. Tablero gato: es la interfaz principal del sistema que contendrá el tablero de Ultimate Tic Tac Toe, en donde se desarrollará el juego.
   e. Búsqueda de jugador: en esta interfaz se mostrará el juego disponible al cual el jugador podrá unirse.
   f. Unirse a juego: en esta interfaz se solicitará el nombre del jugador que se unirá a la partida.
   g. Instrucciones del juego (parte I): esta interfaz mostrará las instrucciones para poder manejar el sistema.
   h. Instrucciones del juego (parte II): esta interfaz mostrará las instrucciones de juego para la aplicación Ultimate Tic Tac Toe.

   La navegación de las interfaces descritas anteriormente se podrá describir a través del diagrama de navegación en la sección 8.2.

4. Entidades del modelo de negocio:
   Las entidades de modelo de negocio usadas en este sistema se podrán ver expuestas en el diagrama en la sección 8.2.

5. Marco de arquitectura:
   a. Caching: El sistema no guardará datos no seguros dentro del cache del cliente. Los únicos datos que serán usados para realizar transacciones son los nombres de los jugadores, sin embargo esos datos tampoco serán guardados en caché.
   b. Validaciones: Se realizarán validaciones de campos para el nombre del jugador del lado cliente. Mientras que del lado del servidor se realizarán las validaciones de existencia de juego disponible en la base de datos.
   c. Restricción y control de accesos: El sistema no contará con control de accesos, únicamente será requerido el nombre del jugador para poder iniciar la aplicación. El administrador del sistema será el único usuario con acceso a los datos y componentes del sistema.
   d. Comunicación con capa de negocios: El sistema usará un tipo de comunicación Chatty que permitirá múltiples peticiones con la capa de negocio. La capa de presentación se diseñará de tal forma que evite múltiples transacciones que puedan comprometer el correcto funcionamiento del sistema.

e. Navegación: El usuario percibirá un tipo de navegación compuesta, ya que tendrá al inicio una navegación jerárquica (en la interfaz de inicio) y una navegación lineal en el resto de las interfaces (Se puede consultar el diagrama de navegación en la sección 8.2 para su mejor entendimiento).
f. Manejo de sesiones: El manejo de sesiones será independiente de la navegación del sistema y este será realizado del lado del servidor.
g. Desempeño: Se espera que los resultados de la comunicación con la capa de negocios permitan que la interface muestre los resultados de forma rápida y coordinada para los jugadores que se encuentren conectados a la misma partida.

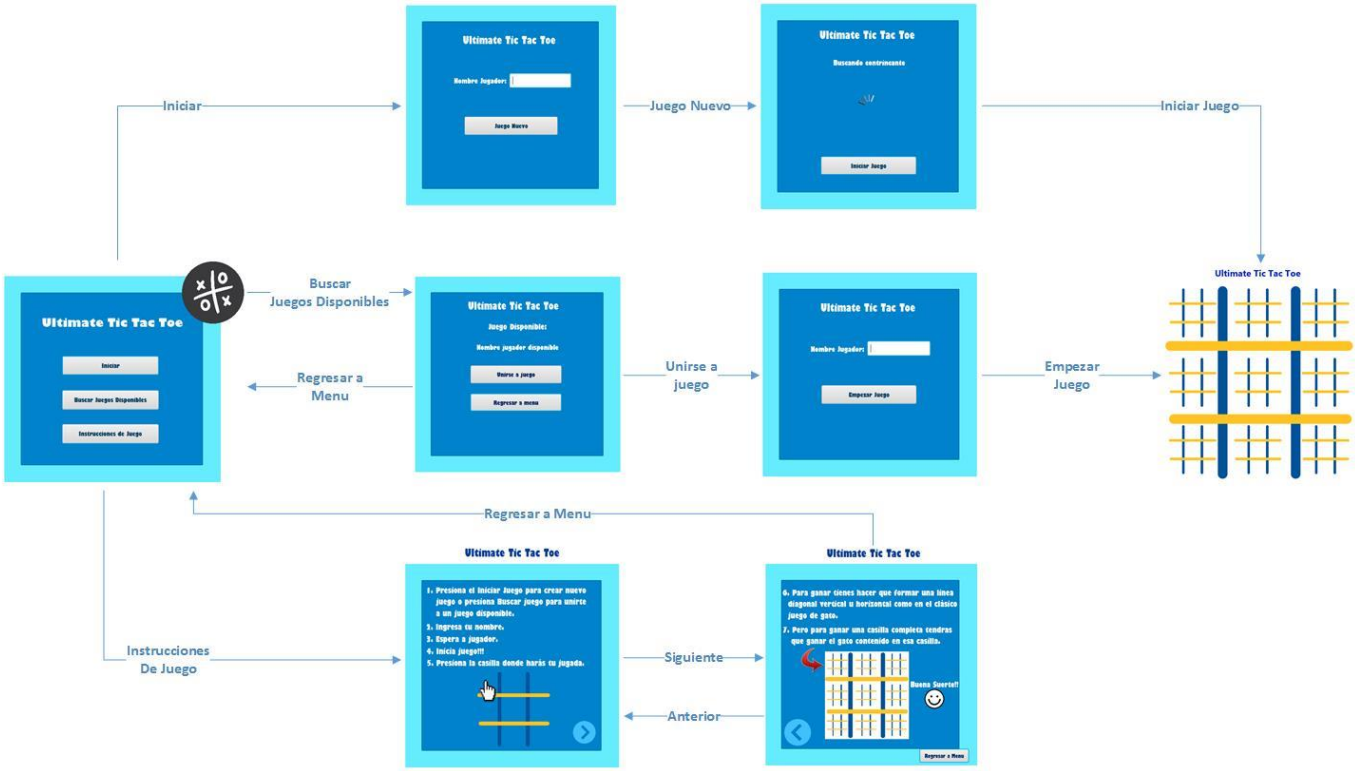
6. Patrones de diseño:
   Los patrones de diseño para la capa de presentación utilizados serán:
   - FrontController: manejará la interfaz a través de la cual el cliente interactuará con el sistema.
   - Helper: serán utilizados por las clases controladoras para tener acceso a la información de la base de datos que sea necesaria.

   La estructura de los patrones de diseño y sus relaciones con los componentes del sistema se podrán ver expuestos en el diagrama de clases en la sección 8.2.

El diseño propuesto para la capa de presentación considera las características de usabilidad necesarias para que el usuario pueda navegar con facilidad dentro del sistema, aún si es la primera vez que lo utiliza, de igual forma permitirá recordar con facilidad las funcionalidades de los componentes de la interface para cuando éste sea utilizado en el futuro. En cuanto a la organización de la información, está propuesta de tal forma que le permita al usuario identificar que componentes dentro de la interfaz corresponden a determinadas acciones y cómo estos componentes se relacionan en sus diferentes secciones.
**Se anexan las impresiones de pantalla de las interfaces del sistema.

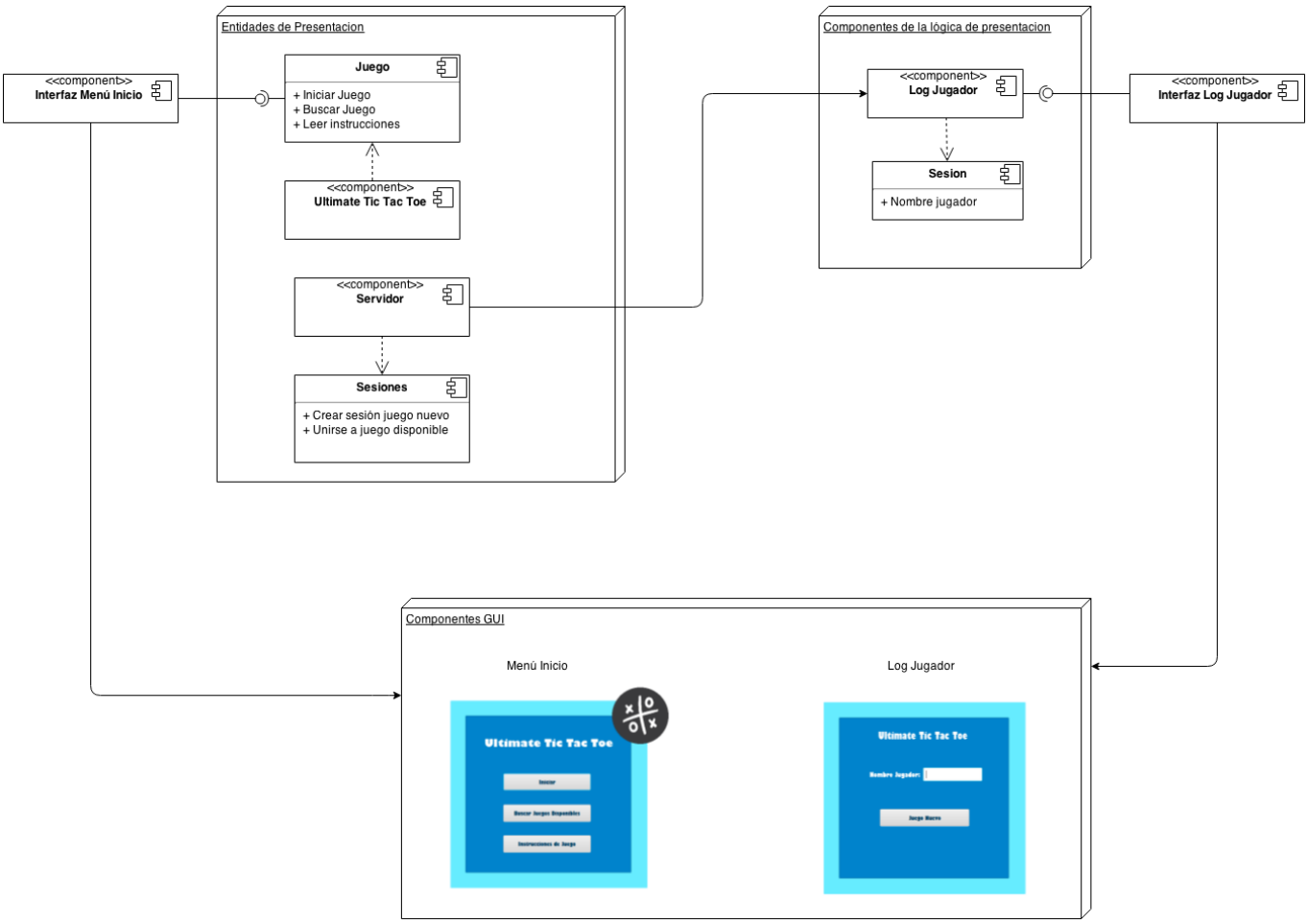
8.2 Layers

1. Diagrama de navegación:



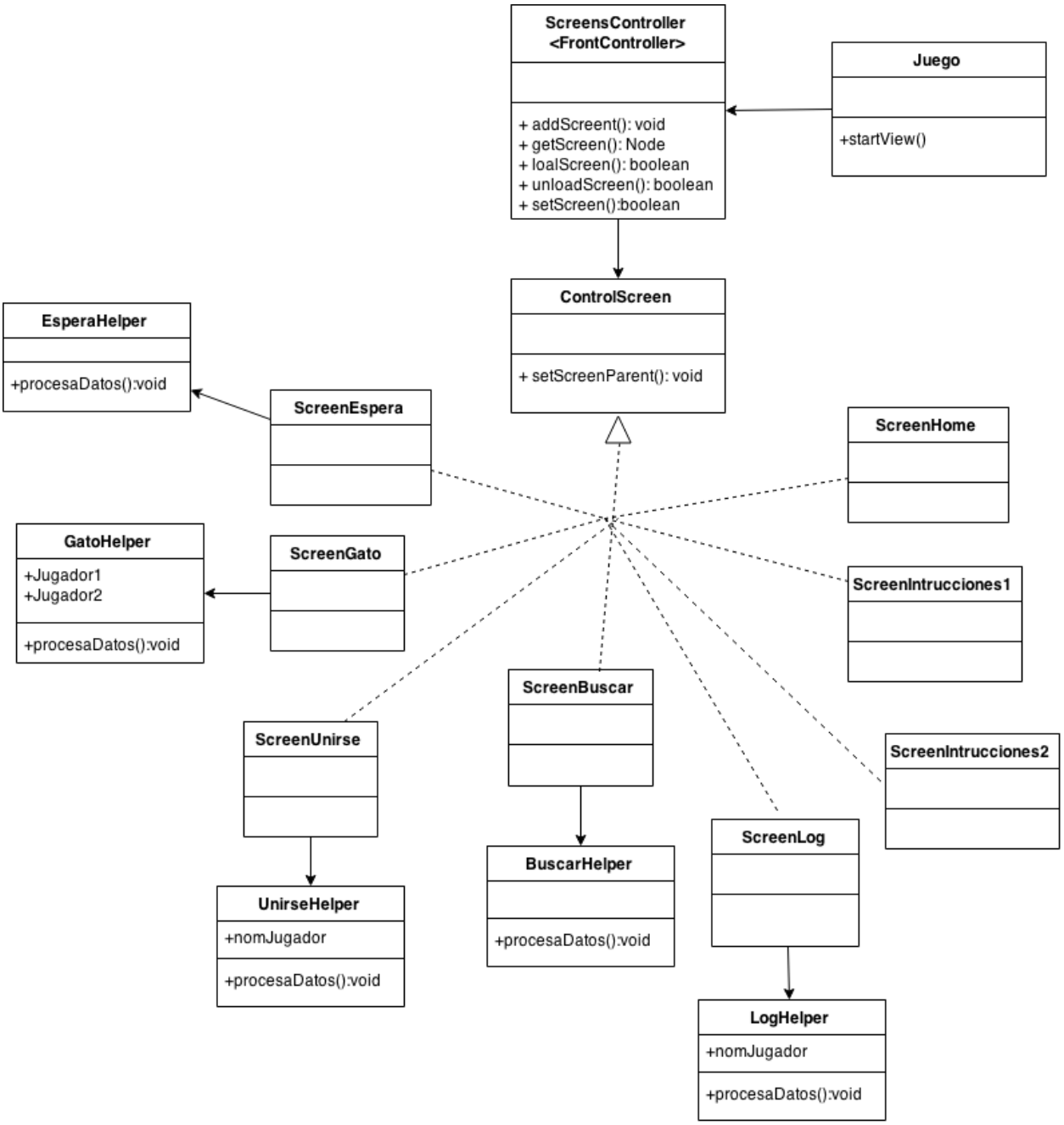
**Se anexa de diagrama completo para una mejor apreciación de los componentes.

2. Digrama de entidades de modelo de negocios:



** Se anexa de diagrama completo para una mejor apreciación de los componentes.

3. Patrones de diseño (Capa presentación):



Capa de Negocio:

Acontinuación se describen los componentes que conforman el negocio del sistema, así como las consideraciones del Marco de Arquitectura.

- Autentificación:

  El acceso al sistema se realizará únicamente en la capa de presentación, donde se solicitará el nombre del jugador. Los componentes de negocios serán utilizados únicamente por éste sistema, no formarán parte de otras aplicaciones.

- Autorización:

  Este diseño de negocio tendrá dos roles especificamente, el rol de usuario y el rol de administrador.

  El rol de usuario se le asignará a cualquier usuario que ingrese a la aplicación, éste tendrá los mismos privilegios, es decir, podrán realizar las mismas actividades, siendo éstas iniciar juego, buscar juego, unirse a juego y realizar jugadas. En cuanto al rol de administrador será asignado a las personas responsables del desarrollo y mantenimiento de la aplicación teniendo acceso a las capas de negocio y datos encargadas de la funcionalidad del sistema.

- Validación:

  Los parametros de entrada serán validados en primera instancia por la capa de presentación, en donde serán campos obligatorios los de nombre de jugador. Una vez que sean ingresados, se validará dentro de la capa de negocio que este nombre no exceda los 20 cáracteres y será interpretado como texto plano evitando así la inyección de sql.

- Mantenimiento:

  Los componentes de negocio serán diseñados en base a los patrones de diseño GoF (Gang of four) y los principios de GRASP que asignarán las responsabilidades respectivas a las clases y objetos garantizando un diseño con mayor cohesión y disminución de acoplamiento en la información del sistema.

- Caching:

  El sistema guardará únicamente la sesión de los jugadores a través del navegador web. Los procesos duplicados serán evitados mediante la capa de presentación que evitará multiples transacciones y la base de datos que tendrá una relación uno a uno en sus componentes.

- Instrumentación:

  El sistema llevará cuenta de los eventos de negocio relacionados con las jugadas realizadas, esto ayudará determinar el símbolo que contendrá la casilla, el estado (ganador o empate) de la casilla y al ganador del juego. El desempeño será monitoreado contantemente para determinar el correcto funcionamiento del sistema y garantizar que el sistema podrá mantener las jugadas en tiempo real.

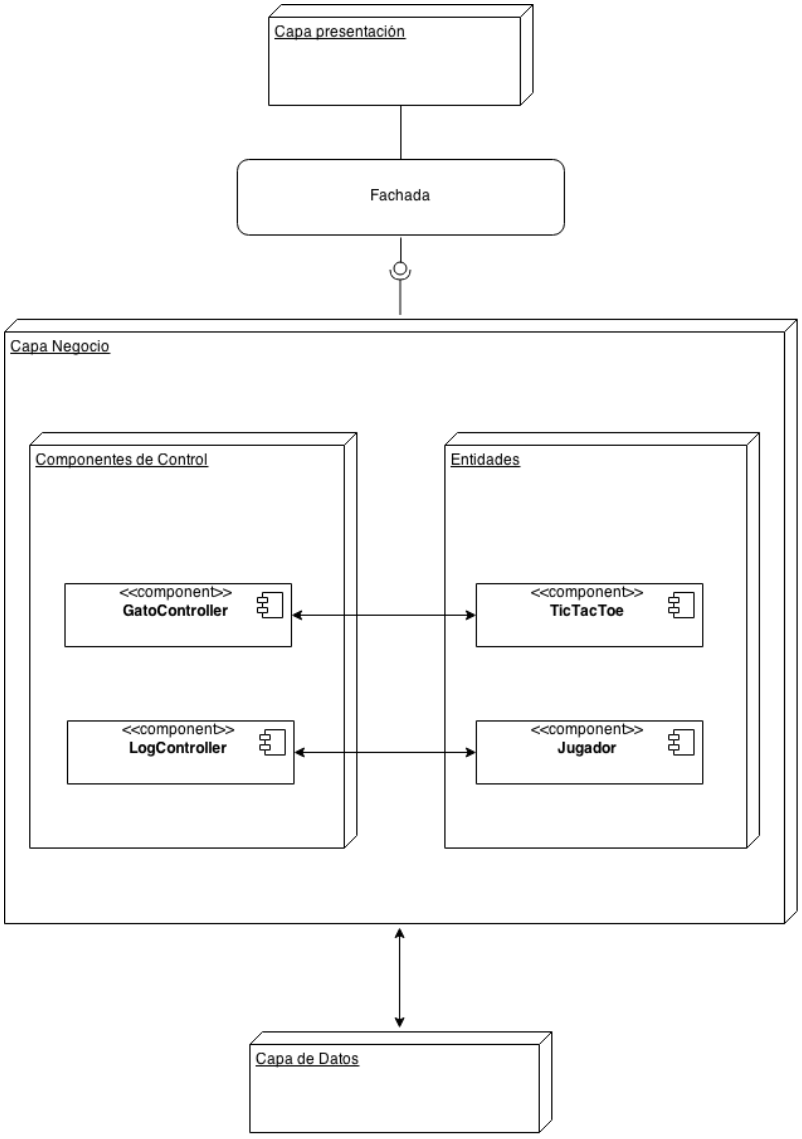
- Manejo de excepciones:

  En caso de encontrar excepciones durante los procesos de negocio, se regresará a un estado previo de la aplicación y se notificará a la capa de presentación que ha ocurrido una interrupción en la solicitud realizada por el usuario, esta notificación será mostrada al usuario para que éste pueda realizar nuevamente dicha solicitud.

- Logging y auditoría:

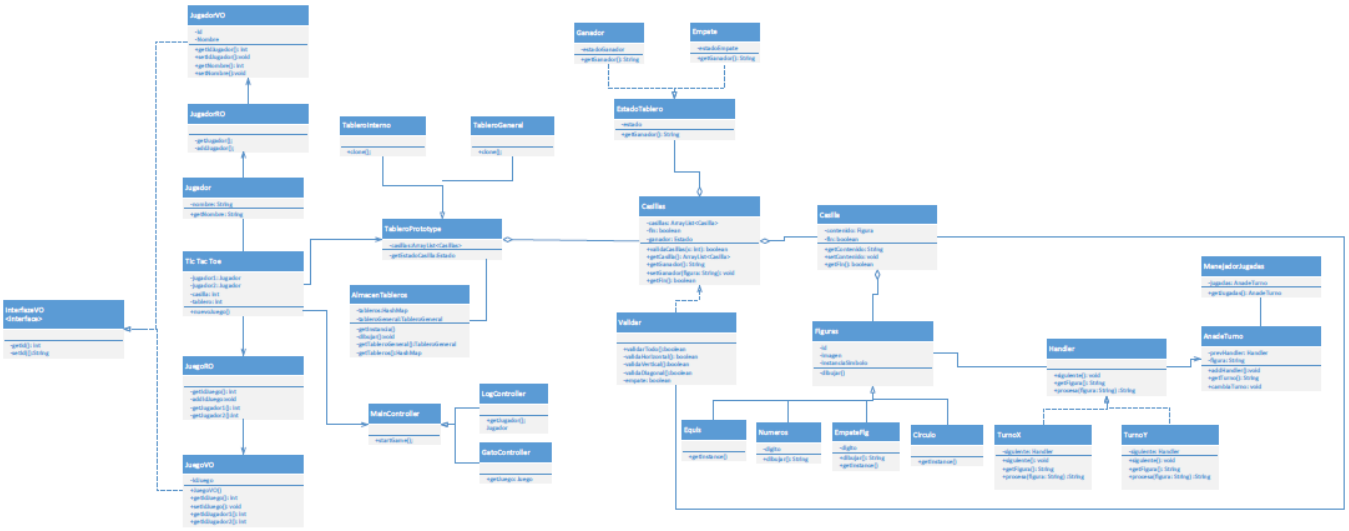
  No se requerirá llevar seguimiento de acceso, ya que no será requerido iniciar sesión dentro del sistema. Unicámente se realizará mantenimiendo de la base de datos, manteniendo así la integridad de los datos almacenados. Éste mantenimiento será realizado periódicamente, dentro de lapsos no mayores a una semana y únicamente podra ser realizado por los desarrolladores del sistema.

1. Diagrama de Componentes:



**Se anexa el diagrama completo para una mejor visualización.

2. Entidades de Negocio:



**Se anexa el diagrama completo para una mejor visualización.

Integración:

Frameworks

JPA	myBatis	Hibernate	JDBC
Este framework trata de conservar las ventajas de la orientación a objetos interactuando con la base de datos a través de objetos, permitiendo usar los "*Plain Old Java Object*" (POJOs).	Herramienta que permite mapear sentencias de SQL y procedimientos almacenados con objetos a partir de ficheros XML. De igual forma permite que el modelo de datos y el modelo de objetos se encuentren independientes.	Busca principalmente evitar problemas entre los modelos de datos que coexisten en la aplicación: el usado en la memoria de la computadora y el usado en las bases de datos.	Permite la ejecución de operaciones sobre bases de datos desde Java, independientemente del sistema operativo donde sea ejecutado o de la base de datos a la cual se accede.
Utiliza archivos XML para configurar el mappeo entre los objetos de Java y las tablas en la base de datos. Utiliza un lenguage similar a SQL, Java Persistence Query Language (JPQL).	Permite utilizar todas las funcionalidades de la base de datos (procedimientos almacenados, vistas). Es una herramienta indicada para bases de datos legadas o desnormalizadas.	Ofrece también el lenguaje HQL (Hibernate Query Language) que soporta características avanzadas como páginación y perfiles dinámicos, así como queries polimorficos.	Unicamente soporta SQL y se debe especificar en el código el mapeo para asignar la representación de datos de un modelo de objetos a un modelo de datos relacional y su esquema de base de datos correspondiente.

Pruebas de Concepto

PoC JPA:

En primera instancia se creó un nuevo proyecto en NetBeans de Java Application.

Se debe corroborar que exista el driver MySQL (Connector/J driver) dentro del proyecto.

Dentro de este proyecto se crearon las entidades a partir de la base de datos. Para esta creación se dio click derecho en el proyecto de NetBeans y se seleccionó la opción de “Entity Classes from Database”.

Después en la ventana emergente se creó un nuevo “Data Source” en el cuál se creó una nueva conexión donde seleccionamos en el Driver “MySQL (Connector/J driver)”. Luego proporcionamos los parametros para establecer dicha conexión (username, password y el nombre de la base de datos que vamos a utilizar, en nuestro caso fue ultimateGato) y presionamos “Finalizar”. Al establecerse la conexión aparecieron las tablas disponibles en la base de datos, las cuales seleccionamos y agregamos al proyecto. Finalmente presionamos “Finalizar” y se crearon dos clases, Jugador.java y Juegos.java que corresponden a nuestras tablas en base de datos (en nuestro caso únicamente contamos con 2 tablas Juegos y Juegador.

Jugador.java

```
@Entity
@Table(name = "Jugador")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Jugador.findAll", query = "SELECT j FROM Jugador j"),
    @NamedQuery(name = "Jugador.findByJugadorID", query = "SELECT j FROM Jugador j WHERE
j.jugadorID = :jugadorID"),
    @NamedQuery(name = "Jugador.findByNombre", query = "SELECT j FROM Jugador j WHERE
j.nombre = :nombre")})
public class Jugador implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "JugadorID")
    private Integer jugadorID;
    @Basic(optional = false)
    @Column(name = "Nombre")
    private String nombre;
    @OneToMany(mappedBy = "jugadorDos")
    private Collection<Juegos> juegosCollection;
    @OneToMany(mappedBy = "jugadorUno")
    private Collection<Juegos> juegosCollection1;

    public Jugador() {
    }

    public Jugador(Integer jugadorID) {
        this.jugadorID = jugadorID;
    }

    public Jugador(Integer jugadorID, String nombre) {
        this.jugadorID = jugadorID;
        this.nombre = nombre;
    }
```

```
    public Integer getJugadorID() {
        return jugadorID;
    }
    public void setJugadorID(Integer jugadorID) {
        this.jugadorID = jugadorID;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    @XmlTransient
    public Collection<Juegos> getJuegosCollection() {
        return juegosCollection;
    }
    public void setJuegosCollection(Collection<Juegos> juegosCollection) {
        this.juegosCollection = juegosCollection;
    }
    @XmlTransient
    public Collection<Juegos> getJuegosCollection1() {
        return juegosCollection1;
    }
    public void setJuegosCollection1(Collection<Juegos> juegosCollection1) {
        this.juegosCollection1 = juegosCollection1;
    }
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (jugadorID != null ? jugadorID.hashCode() : 0);
        return hash;
    }
    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Jugador)) {
            return false;
        }
        Jugador other = (Jugador) object;
        if ((this.jugadorID == null && other.jugadorID != null) || (this.jugadorID != null &&
!this.jugadorID.equals(other.jugadorID))) {
            return false;
        }
        return true;
    }
    @Override
    public String toString() {
        return "jpa.Jugador[ jugadorID=" + jugadorID + " ]";
    }

}
```

Juegos.java

```
@Entity
@Table(name = "Juegos")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Juegos.findAll", query = "SELECT j FROM Juegos j"),
    @NamedQuery(name = "Juegos.findByJuegoID", query = "SELECT j FROM Juegos j WHERE
j.juegoID = :juegoID"),
    @NamedQuery(name = "Juegos.findByGanador", query = "SELECT j FROM Juegos j WHERE
j.ganador = :ganador"),
    @NamedQuery(name = "Juegos.findByFecha", query = "SELECT j FROM Juegos j WHERE j.fecha =
:fecha")})
public class Juegos implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "JuegoID")
    private Integer juegoID;
    @Column(name = "Ganador")
    private Integer ganador;
    @Column(name = "Fecha")
    @Temporal(TemporalType.TIMESTAMP)
    private Date fecha;
    @Basic(optional = false)
    @Lob
    @Column(name = "Activo")
    private byte[] activo;
    @JoinColumn(name = "JugadorDos", referencedColumnName = "JugadorID")
    @ManyToOne
    private Jugador jugadorDos;
    @JoinColumn(name = "JugadorUno", referencedColumnName = "JugadorID")
    @ManyToOne
    private Jugador jugadorUno;

    public Juegos() {
    }
    public Juegos(Integer juegoID) {
        this.juegoID = juegoID;
    }
    public Juegos(Integer juegoID, byte[] activo) {
        this.juegoID = juegoID;
        this.activo = activo;
    }
    public Integer getJuegoID() {
        return juegoID;
    }
    public void setJuegoID(Integer juegoID) {
        this.juegoID = juegoID;
    }
```

```
    public Integer getGanador() {
        return ganador;
    }
    public void setGanador(Integer ganador) {
        this.ganador = ganador;
    }
    public Date getFecha() {
        return fecha;
    }
    public void setFecha(Date fecha) {
        this.fecha = fecha;
    }
    public byte[] getActivo() {
        return activo;
    }
    public void setActivo(byte[] activo) {
        this.activo = activo;
    }
    public Jugador getJugadorDos() {
        return jugadorDos;
    }
    public void setJugadorDos(Jugador jugadorDos) {
        this.jugadorDos = jugadorDos;
    }
    public Jugador getJugadorUno() {
        return jugadorUno;
    }
    public void setJugadorUno(Jugador jugadorUno) {
        this.jugadorUno = jugadorUno;
    }
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (juegoID != null ? juegoID.hashCode() : 0);
        return hash;
    }
    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Juegos)) {
            return false;
        }
        Juegos other = (Juegos) object;
        if ((this.juegoID == null && other.juegoID != null) || (this.juegoID != null &&
!this.juegoID.equals(other.juegoID))) {
            return false;
        }
        return true;
    }
```

```
    @Override
    public String toString() {
        return "jpa.Juegos[ juegoID=" + juegoID + " ]";
    }

}
```

De igual forma se creó el XML “Persistence.xml” en el cuál se especifican los datos de conexión a la base de datos.

Persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="JPAPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>jpa.Jugador</class>
    <class>jpa.Juegos</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/ultimateGato?zeroDateTimeBehavior=convertToNull"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.password" value=""/>
    </properties>
  </persistence-unit>
</persistence>
```

En la clase principal instanciamos los querys utilizados a través de un EntityManager, como se especifica a continuación:

JPA.java

```
public class JPA {
   public static void main(String[] args) {
      EntityManager entitymanager = Persistence.createEntityManagerFactory("JPAPU").createEntityManager();
      Query query = entitymanager.createNamedQuery("Jugador.findAll");
      List<Jugador> list = query.getResultList();
      for(Jugador j:list){
         System.out.println("jugadorID "+j.getJugadorID());
         System.out.println("Nombre "+j.getNombre());
      }
      query = entitymanager.createNamedQuery("Juegos.findByJuegoID");
      query.setParameter("juegoID", 1);
      List<Juegos> games = query.getResultList();
      for(Juegos js:games){
         System.out.println("Player1 "+js.getJugadorUno());
         System.out.println("Player2 "+js.getJugadorDos());
      }
   }
}
```

1. Con “entitymanager.createNamedQuery("Jugador.findAll");” ejecutamos el query con el identificador “Jugador.findAll” (que corresponde al “SELECT j FROM Juegos j” que especificamos en las entidades java).
2. Con getJugadorID y getNombre obtenemos el id y nombre de todos los jugadores que existen en la base de datos.
3. Con “entitymanager.createNamedQuery("Juegos.findByJuegoID");” ejecutamos el query con el identidicador "Juegos.findByJuegoID” (que corresponde al “SELECT j FROM Juegos j WHERE j.juegoID = :juegoID”, donde juegoID se los especificamos con el el query.setParameter("juegoID", 1) que nos trae el juego en ese especifico id.
4. Con getJugadorUno y getJugadorDos obtenemos los jugadores que existen en el juego que especificamos.

PoC myBatis:

En primera instancia se creó un nuevo proyecto en NetBeans de Java Application.

Se debe corroborar que exista el driver MySQL (Connector/J driver) dentro de las librerías del proyecto, así mismo se debe importar la libraría de myBatis (link de descarga para myBatis: https://github.com/mybatis/mybatis-3/releases).

Dentro de este proyecto se creó un nuevo paquete con el nombre de “XML”.Dentro de este paquete se definirán los siguientes XML que se encargarán de la conexión a la base de datos.

Se crea un nuevo archivo XML, llamado SQLConfig.xml. Dentro de este archivo se definirán los datos necesarios para la conexión con la base de datos.

SQLConfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sqlMapConfig>
  <transactionManager type="JDBC" commitRequired="false">
    <dataSource type="POOLED">
     <property name="JDBC.Driver" value="com.mysql.jdbc.Driver"/>
      <property name="JDBC.ConnectionURL"
value="jdbc:mysql://localhost:3306/ultimateGato?zeroDateTimeBehavior=convertToNull"/>
      <property name="JDBC.Username" value="root"/>
      <property name="JDBC.Password" value=""/>
    </dataSource>
  </transactionManager>
  <sqlMap resource="xml/Jugador.xml"/>
  <sqlMap resource="xml/Juegos.xml"/>
 </sqlMapConfig>
```

Aquí especificamos los información del driver para poder conectar la base de datos, así como el username y password para poder acceder a ella. De igual forma especificamos la localización del archivo XML Jugador y Juegos donde se mapearan los objetos a las sentencias de SQL, la códificación de estos se muestra acontinuación:

Jugador.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sqlMap namespace="Jugador">
  <typeAlias alias="Jugador " type="pojo.Jugador "/>
  <resultMap id="JugadorRest" class="Jugador">
    <result property="id" column="JugadorID"/>
    <result property="nombre" column="Nombre"/>
   </resultMap>
  <!—Aqui seleccionaremos todas los jugadores que existen dentro de la base de datos-->
  <select id="selectAllPlayers" resultMap="JugadorRest">
    select * from Jugador
  </select>
<!—Aqui seleccionamos al jugador correspondiente al id -->
<select id="selectPlayerById" parameterClass="int" resultClass="Jugador">
 <select JugadorID as id, Nombre as nombre from Jugador where JugadorID= #id#
</select>
<!--  Aquí insertaremos en la base de datos un nuevo jugador -->
<insert id="insertPlayer" parameterClass="Jugador">
    insert into Jugador (
       JugadorID,
       Nombre)
    values (
      #id#, #nombre#)
  </insert>
</sqlMap>
```

Juegos.xml

```
<sqlMap namespace="Juegos">
  <typeAlias alias="Juegos" type="pojo.Juegos "/>
  <resultMap id="JuegoRest" class="Juegos">
    <result property="idJuego" column="JuegoID"/>
    <result property="jugador1" column="JugadorUno"/>
    <result property="jugador2" column="JugadorDos"/>
    <result property="ganador" column="Ganador"/>
    <result property="fecha" column="Fecha"/>
    <result property="activo" column="Activo"/>
   </resultMap>
  <!—Aquí seleccionaremos todas los juegos que existen dentro de la base de datos-->
  <select id="selectAllGames" resultMap="JuegoRest">
    select * from Juegos
  </select>
<!—Aquí seleccionamos al juego correspondiente al id -->
<select id="selectGameById" parameterClass="int" resultClass="Juegos">
 <select JuegoID as idJuego, JugadorUno as jugador1, JugadorDos as jugador2 from Juegos where JuegoID = #
idJuego#
</select>
<!--  Aquí insertaremos un nuevo juego -->
<insert id="insertGame" parameterClass="Juegos">
    insert into Juegos (
       JuegoID,
       JugadorUno,
       JugadorDos,
       Ganador,
       Fecha,
       Activo)
    values (
      #idJuego#, #jugador1# , #jugador2#, #ganador#, #fecha#, #activo#)
  </insert>
</sqlMap>
```

Después creamos las clases objeto Java a las cuales esta mapeando el XML, Juego y Jugador:

Jugador.java

```
package pojo;

public class Jugador {
    private int id;
    private String nombre;
    //…
    //Getters y setters
}
```

Juegos.java

```
package pojo;

public class Juegos {
    private int id;
    private int jugador1;
    private int jugador2;
    private int ganador;
    private String fecha;
    private int activo;
    //….
    //Getters y setters
}
```

Finalmente cargamos nuestro mapeador SQLConfig.xml a nuestra aplicación a través de nuestra clase Servicio:

```
public class Servicio {
    private static SqlMapClient mapper;
    try {
      Reader reader = Resources.getResourceAsReader("xml/SQLConfig.xml");
      mapper = SqlMapClientBuilder.buildSqlMapClient(reader);
      reader.close();
    } catch (IOException e) {
      throw new RuntimeException("Hubo error en la conexion");
    }

  Jugador jugador1= new Jugador();
  jugador.setNombre(“David”);

  Juegos juego= new Juegos();
  juego.setId(1);
  juego.setJugador1(1);
  juego.setJugador2(2);
  juego.setGanador(1);
  juego.setFecha(“16-04-2015”);
  juego.setActivo(1);

   mapper.insert("insertGame", juego);
   mapper.insert("insertPlayer", jugador1);

   mapper.queryForList("selectAllPlayers");
   mapper.queryForList("selectAllGames");
}
```

A través de el mapper podemos acceder a los queries definidos en los XML, por ejemplo:

1. Para obtener todos los jugadores de la base de datos:

   mapper.queryForList("selectAllPlayers");

   donde especificamos el id del query que definimos en el XML para que nos mostrará toda la lista de jugadores.

2. Para poder insertar un jugador a la base de datos:

   mapper.insert("insertPlayer", jugador1);

   donde le especificamos el id del query que definimos en el XML para insertar un jugador a la base de datos y en el segundo parámetro le especifícamos el objeto POJO que creamos que hace referencia al jugador.

3. Para obtener todos los jugadores de la base de datos:

   mapper.queryForList("selectAllGames");

   donde especificamos el id del query que definimos en el XML para que nos mostrará toda la lista de juegos.

4. Para poder insertar un juego a la base de datos:

   mapper.insert("insertGame", juego);

donde le especificamos el id del query que definimos en el XML para insertar un juego a la base de datos y en el segundo parámetro le especifícamos el objeto POJO que creamos que hace referencia al juego.

PoC Hibernate

En primera instancia se creó un nuevo proyecto en NetBeans de Java Application.

Se importaron las librerias necesarias para el uso de Hibernate (link de descarga en caso de no tener las librerias de hibernate http://hibernate.org/orm/downloads/).

Dentro de este proyecto se creó un nuevo paquete con el nombre de "XML".Dentro de este paquete se definirán los siguientes XML que se encargarán de la conexión a la base de datos.

Se crea un nuevo archivo XML, llamado HibernateConfig. Dentro de este archivo se definirán los datos necesarios para la conexión con la base de datos.

HibernateConfig.xml

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-configuration>
  <session-factory>
  <property name="hibernate.dialect"> org.hibernate.dialect.MySQLDialect</property>
  <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/
ultimateGato?zeroDateTimeBehavior=convertToNull </property>
  <property name="hibernate.connection.username">root</property>
  <property name="hibernate.connection.password"></property>
  <mapping resource="Jugador.hbm.xml"/>
  <mapping resource="Juego.hbm.xml"/>
 </session-factory>
</hibernate-configuration>
```

Aquí especificamos los información del driver para poder conectar la base de datos, así como el username y password para poder acceder a ella. De igual forma especificamos la localización del archivo XML Jugador y Juego a las cuales mapearemos la base de datos. la códificación de estos se muestra acontinuación:

Jugador.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-mapping>
  <class name="Jugador" table="Jugador">
    <meta attribute="class-description">Descripcion de clase Jugador </meta>
    <id name="id" type="int" column="JugadorID"><generator class="native"/></id>
    <property name="nombre" column="Nombre" type="string"/>
  </class>
</hibernate-mapping>
```

Juego.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-mapping>
  <class name="Juego" table=" Juegos ">
    <meta attribute="class-description">Descripcion de clase Juego</meta>
    <id name="idJuego" type="int" column="JuegoID"><generator class="native"/></id>
    <property name="jugador1" column="JugadorUno" type="int"/>
    <property name="jugador2" column="JugadorDos" type="int"/>
    <property name="ganador" column="Ganador" type="int"/>
    <property name="fecha" column="Fecha" type="String"/>
    <property name="activo" column="Activo" type="int"/>
  </class>
</hibernate-mapping>
```

Aquí creamos los elementos correspondientes a las tablas y columnas que existen en la base de datos.

Después creamos las clases objeto Java a las cuales esta mapeando el XML, Juego y Jugador:

Jugador.java

```
public class Jugador {
   private int id;
   private String nombre;

   public Jugador (String nombre) {
     this.nombre = nombre;
  }
   //…
   //Getters y setters
}
```

Juego.java

```
package pojo;

public class Juego {
   private int id;
   private int jugador1;
   private int jugador2;
   private int ganador;
   private String fecha;
   private int activo;

   public Jugador (int jugador1, int jugador2, int ganador, String fecha, int activo) {
     this.jugador1 = jugador1;
     this.jugador2 = jugador2;
     this.ganador = ganador;
     this.fecha = fecha;
     this.activo = activo;
  }
   //….
   //Getters y setters
}
```

Finalmente cargamos nuestro SessionFactory que será el encargado de leer nuestro mapeo especificado dentro de nuestros XML:

Manejador.java

```
public class Manejador {

private static SessionFactory factory;
  public static void main(String[] args) {
     try{
        factory = new Configuration().configure().buildSessionFactory();
     }catch (Throwable ex) {
        throw new ExceptionInInitializerError(ex);
     }

     Manejador manager = new Manejador ();

     manager. addPlayer ("David");
     manager. addGame (1, 2, 1, “16/04/15”, 1);

     manager. getPlayers ();
     manager. getGames ();

}
```

Para agregar jugadores o juegos a nuestra base de datos, definimos un método addPlayer y addGame:

```
public Integer addPlayer(String nombre){
    Session session = factory.openSession();
    Transaction tx = null;
    Integer idJugador = null;
    try{
      tx = session.beginTransaction();
      Jugador jugador = new Jugador(nombre);
      idJugador = (Integer) session.save(jugador);
      tx.commit();
    }catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
    }finally {
      session.close();
    }
    return idJugador;
  }

public Integer addGame(int jugador1, int jugador2, int ganador, String fecha, int activo){
    Session session = factory.openSession();
    Transaction tx = null;
    Integer idJuego = null;
    try{
      tx = session.beginTransaction();
      Juego juego = new Juego(jugador1, jugador2, ganador, fecha, activo);
      idJuego = (Integer) session.save(juego);
      tx.commit();
    }catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
    }finally {
      session.close();
    }
    return idJuego;
  }
```

Para obtener datos de la base creamos un metodo getPlayers y getGames:

```
public void getPlayers( ){
    Session session = factory.openSession();
    Transaction tx = null;
    try{
      tx = session.beginTransaction();
      List players = session.createQuery("FROM Jugador").list();
      for (Iterator iterator = players.iterator(); iterator.hasNext();){
        Jugador jugador = (Jugador) iterator.next();
        System.out.print( jugador.getNombre() );
      }
      tx.commit();
    }catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
    }finally {
      session.close();
    }
  }

public void getGames( ){
    Session session = factory.openSession();
    Transaction tx = null;
    try{
      tx = session.beginTransaction();
      List games = session.createQuery("FROM Juegos").list();
      for (Iterator iterator = games.iterator(); iterator.hasNext();){
        Juego juego = (Juego) iterator.next();
        System.out.print( juego.getJugador1() );
        System.out.print( juego.getJugador2() );
      }
      tx.commit();
    }catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
    }finally {
      session.close();
    }
  }
```

PoC JDBC

En primera instancia se creó un nuevo proyecto en NetBeans de Java Application.

Se debe corroborar que exista el driver MySQL (Connector/J driver) dentro del proyecto.

Para el conectar la base de datos utilizamos una clase Conexión.java que manda los datos necesarios para establecer la conexión con sql.

```
public class Conexion {

    protected Connection con = null;
    protected CallableStatement consulta;
    private final String DB = "ultimateGato", User = "root", Password = "";

    public void conectar() {
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            this.con = DriverManager.getConnection("jdbc:mysql://localhost/" + DB, User, Password);
        } catch (SQLException | IllegalAccessException | InstantiationException | ClassNotFoundException ex) {
            ex.printStackTrace();
        }
    }

    public void desconectar() {
        try {
            this.con.close();
        } catch (SQLException ex) {
        }
    }

}
```

Después a través de la conexión utilizamos una clase Servicio.java, que se encarga obtener la información de las tablas de la base de datos a través de la ejecución de los querys especificados.

Servicio.java

```
public class Servicios extends Conexion {

    public int nuevoJugador(String nombre) {
        int id = 0;
        try {
            this.conectar();
            this.consulta = this.con.prepareCall("call nuevoJugador(?)");
            this.consulta.setString(1, nombre);
            ResultSet datos = this.consulta.executeQuery();
            datos.next();
            id = datos.getInt("ID");
            this.desconectar();
        } catch (SQLException ex) {
        }
        return id;
    }

    public ArrayList<Integer> buscarJuego() {
        ArrayList<Integer> al = new ArrayList<>();
        try {
            this.conectar();
            this.consulta = this.con.prepareCall("call buscarJuego()");
            ResultSet datos = this.consulta.executeQuery();
            datos.next();
            al.add(datos.getInt("JuegoID"));
            al.add(datos.getInt("JugadorUno"));
            this.desconectar();
        } catch (SQLException ex) {
        }
        return al;
    }

    public String getNombre(int id) {
        String nombre = null;
        try {
            this.conectar();
            this.consulta = this.con.prepareCall("call getNombre(?)");
            this.consulta.setInt(1, id);
            ResultSet datos = this.consulta.executeQuery();
            datos.next();
            nombre = datos.getString("Nombre");
            this.desconectar();
        } catch (SQLException ex) {
        }
        return nombre;
    }
```

```
public int insertarJugador(int juegoID, String jugador) {
        int id = 0;
        try {
            this.conectar();
            this.consulta = this.con.prepareCall("call insertarJugador(?, ?)");
            this.consulta.setInt(1, juegoID);
            this.consulta.setString(2, jugador);
            ResultSet datos = this.consulta.executeQuery();
            datos.next();
            id = datos.getInt("ID");
            this.desconectar();
        } catch (SQLException ex) {
        }
        return id;
    }

    public void ganarJuego(int juegoID, int jugadorID) {
        try {
            this.conectar();
            this.consulta = this.con.prepareCall("call ganarJuego(?, ?)");
            this.consulta.setInt(1, juegoID);
            this.consulta.setInt(2, jugadorID);
            this.consulta.executeQuery();
            this.desconectar();
        } catch (SQLException ex) {
        }
    }
}
```

**En este caso se utilizaron los procedures para llamar a los querys de select e insert.

Para mandar a llamar a los querys especificados en la clase de Servicio, en el main realizamos lo siguiente:

```
public class Main {

   public static void main(String[] args) {
      Servicios s = new Servicios();

      int david = s.nuevoJugador("David");
      System.out.println("david: " + david);

      int muchacho = s.nuevoJugador("Muchacho");
      System.out.println("muchacho: " + muchacho);

      ArrayList<Integer> al = s.buscarJuego();
      System.out.println("buscarJuego1: " + al.toString());

      int cyn = s.insertarJugador(al.get(0), "Cyn");
      System.out.println("cyn: " + cyn);

      ArrayList<Integer> al2 = s.buscarJuego();
      System.out.println("buscarJuego2: " + al2.toString());

      String nombre = s.getNombre(al.get(0));
      System.out.println("nombre: " + nombre);
      s.ganarJuego(al.get(0), cyn);
   }
}
```

A través del analisis previo de los distintos frameworks para conexión a base de datos, escogimos el última opción de JDBC para la aplicación ya que nos permite un fácil acceso a la base de datos, además de que cuenta con una alta compatibilidad en distintas plataformas. Mediante JDBC podemos realizar las llamadas a los queryes, que en nuestro caso serán llamadas a procedures que asegurarán la integridad de la base de datos.

Capa de Acceso a Datos

A continuación se describen las especificaciones de la capa de acceso a datos.

1. Marco de Arquitectura:

a) Escenario
Se tendrá un escenario greenfield ya que se tendrá control las decisiones que conforman la base de datos, así como la forma en la que se encuentra estructurada la base de datos.

b) Batch:
El sistema no contará con procesos que se ejecuten periódicamente, a excepción de los encargados de la actualización de las jugadas del tablero.

c) Cache:
El sistema no requerirá guardar datos en caché ya que únicamente guardará los nombres de los jugadores, mandando una transacción para agregar jugador por juego.

d) BLOBs (Binary Large Objects):
El sistema no contará con BLOBs, únicamente almacenará datos de tamaño pequeño como datos de tipo

String, Integer y Date, para los nombres de los jugadores, Ids y fecha respectivamente.

e) Manejo de Excepciones:
Las excepciones que surjan de la base de datos, serán mandadas a las capas superiores como error de conexión, de tal forma que el usuario no pueda ver código de excepciones sql.

f) Formato de Datos:
Los datos que conformarán la aplicación podrán ser usados para interoperabilidad, de igual forma podrán ser utilizados para diferentes escenarios con la posibilidad de escalabilidad.

g) Mapeo Objeto/Relacional:
Los datos utilizados por la base de datos serán referenciados a partir de las entidades de negocio respectivas utilizando el API de Java Database Connectivity (JDBC).

h) Transacciones:
Las transacciones tendrán una concurrencia optimista, ya que la estructura del sistema asegura que los datos que están siendo escritos son consistentes con los leídos en primera instancia, es decir que ningún otro elemento podrá modificar los datos después de que hayan sido leídos por la base de datos.

i) Conexiones:
Se utilizará una conexión por transacción de agregar jugador, buscar jugador, obtener el nombre del jugador y agregar el ganador del juego. Estás conexiones serán cerradas en el momento en que acaben de realizar sus respectivas operaciones.

j) Queries:
Los queries utilizados para la base de datos serán almacenados en procedures por lo que no serán necesarios objetos constructores de queries, las solicitudes a la base de datos sólo podrán realizar a través de los procedures con llamadas y ejecuciones de los queries ya predefinidos.

k) Store Procedures:
Las llamadas a la base de datos se realizarán a través de Store Procedures. Las operaciones que realizarán tendrán un gran peso ya que serán las responsables de crear jugadores y correlacionar a estos a su respectivo juego.

l) Validaciones:
Los datos ingresados por el usuario son únicamente los nombres de los jugadores, éstos serán interpretados como datos tipos String para su almacenamiento en la base de datos, la validación de éstos datos se realiza en las capas superiores. Los ids de los jugadores y juegos serán de tipo int, sin embargo la creación de éstos serán generados a partir de los queries, por lo que no requerirán validación. Sólo el dato del segundo jugadores podrá ser nulo, ya que el primer jugador creará el juego y estará en espera del ingreso del segundo jugador.

2. Agentes de Servicio:

- El sistema no requerirá agentes de servicio para su implementación e integración con la base de datos.

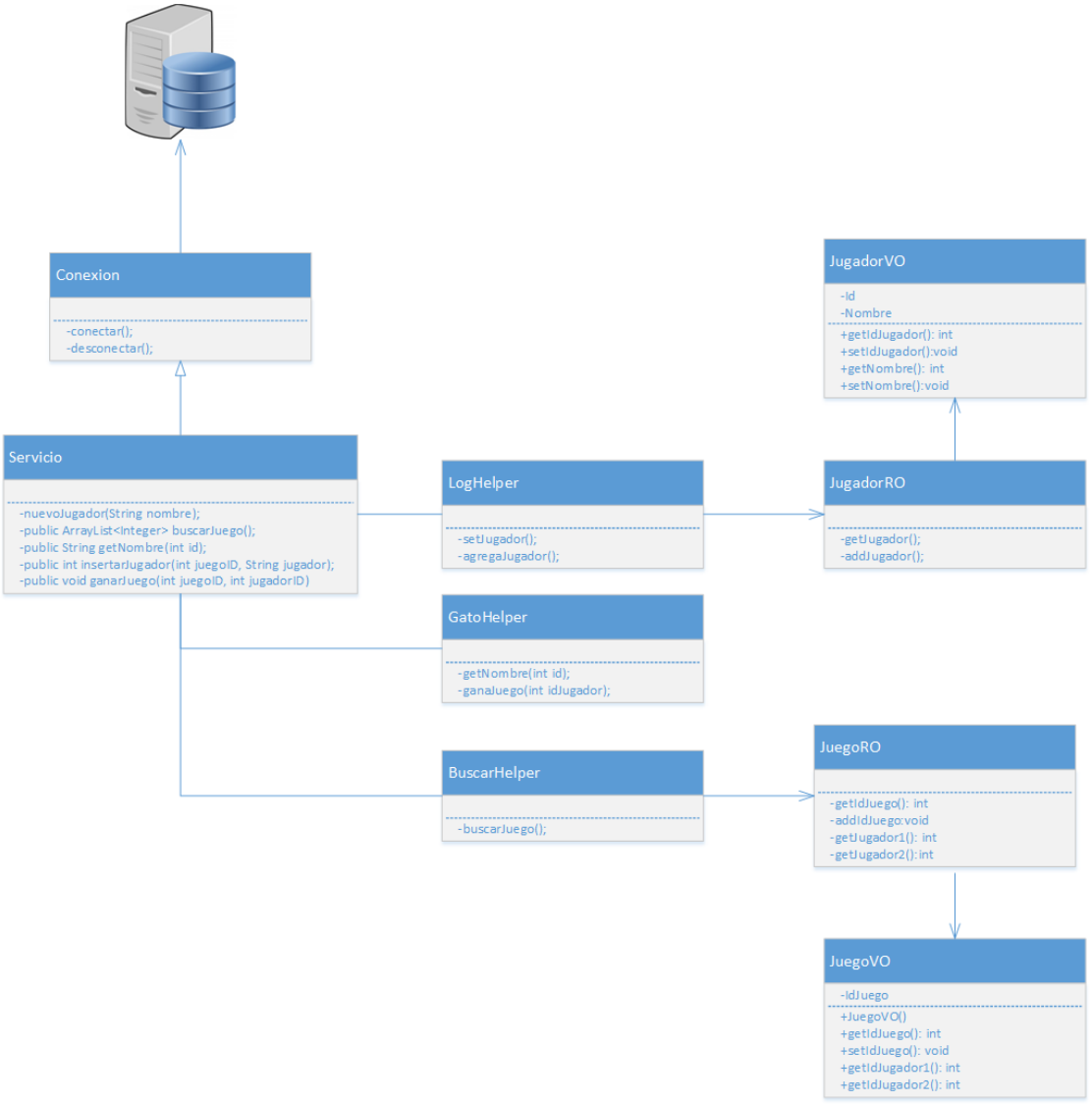
3. Formato de Entidades:

- Las entidades utilizadas en el modelo de acceso a datos serán en formato de clases a través de las cuáles serán manipuladas para integración con la estructura de la aplicación. El tipo de datos será correspondiente al tipo de dato de la entidad de negocio a la cual hace referencia.

4. Tecnología de acceso a datos:

- Java Database Connectivity (JDBC)

Componentes de acceso a datos:



**se anexa diagrama completo para una mejor visualización.

9. Data View

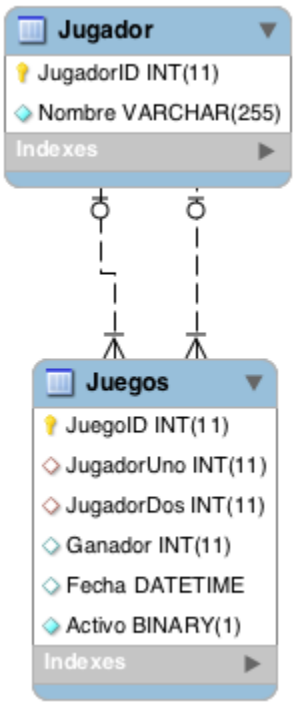
En esta sección se describe el modelo en el cuál se almacenarán los datos de la aplicación.

DAOs:

Para la conexión a base de datos se utilizará una clase DAO, nombrada como Servicios. Ésta DAO se encargará de hacer las llamadas a los queries encargados de agregar a los jugadores y correlacionar a los jugadores al juego respectivo.

De igual forma se tendrá una clase Conexión quien almacenará las credenciales necesarias para conectarse a la base de datos (nombre de usuario, contraseña, nombre de la base de datos).

Modelo de datos (Relacional):



Como podemos observar en el diagrama anterior, la base de datos contará con dos tablas Jugador y Juegos, donde cada una de ellas contendrá su respectivo Id que servirá como identificador para el manejo de los elementos a lo largo del sistema.

10. Size and Performance

Este diseño de arquitectura de software suporta los siguientes requerimientos:

- El sistema puede soportar como máximo a 2 usuarios conectados simultáneamente al servidor
- El sistema puede proporcionar la persistencia de datos mientras exista la conexión al servidor y a la base de datos.
- El sistema puede correr en cualquier computadora, mientras tenga instalado un navegador web y tenga conexión a internet.
- El cliente debe de tener instalados los plugins de java para su navegador web.
- Se requiere una conexión a internet mínima de 5 MB para que los usuarios puedan usar la aplicación.
- El cliente requiere un espacio en disco duro mínimo 600 MB libres, necesarios para la instalación de un navegador web y los plugins de java.

La arquitectura seleccionada es compatible con los requisitos de tamaño y de sincronización a través de la implementación de una arquitectura cliente-servidor. Los componentes han sido diseñados para garantizar que se necesitan requisitos mínimos de disco y de memoria por parte de la computadora del cliente.

11. Quality

- Calidad del Sistema
    - Compatibilidad:
      El sistema será compatibles con los siguientes navegadores web: Google Chrome, Mozilla Firefox, Internet Explorer. De igual forma el sistema requerirá tener instalado el plugin de java en los navegadores previamente mencionados para su correcto funcionamiento.
    - Capacidad de prueba:
      Para garantizar el funcionamiento correcto de los componentes del sistema se realizarán pruebas unitarias, así como pruebas de integración para verificar la interacción adecuada de los componentes.
- Calidad en tiempo de ejecución
    - Disponibilidad:
      El sistema estará disponible las 24 horas del día, los 7 días de la semana, teniendo un 5% de posibilidad en el que el sistema se encuentre "abajo". Dentro de este porcentaje se contempla errores en el sistema como sobrecarga de solicitudes o fallas en el servidor en el que haya sido montada la aplicación.
    - Interoperabilidad:
      El sistema tendrá la capacidad de comunicarse con MySql, mediante el cual el sistema podrá guardar la información de los jugadores que interactúen con el sistema.
    - Gestionable:
      El sistema proporcionará una interfaz que permitirá una fácil administración y manipulación de los componentes del sistema.
    - Desempeño:
      El sistema proporcionará una latencia de aproximadamente 250 ms.
    - Fiabilidad:
      Se espera que el tiempo entre fallas sea mayor a 300 horas.
    - Escalabilidad:
      El sistema proporcionará la habilidad de crecer en un sistema aún más grande, siendo extendido en más de un servidor y dejará la posibilidad de tener más de un juego por jugador.
    - Seguridad:
      El sistema garantizará la integridad de los datos, restringiendo el acceso a la base de datos únicamente a las clases DAO encargadas de la conexión. Toda clase que requiera información de la base de datos, se comunicará únicamente con las clases DAO, no directamente con la base de datos. Se usarán Callable Statements para evitar la inyección de código SQL en los campos de texto de la interfaz.
- Calidad de Diseño
    - Integridad Conceptual:
      Los componentes utilizados dentro del sistema, así como entidades, clases y métodos, tendrán nomenclaturas acordes la función que realizan y al tema de la aplicación.
    - Flexibilidad:
      El sistema proporcionará una fácil manipulación de sus componentes en caso de ser requerido.
    - Mantenimiento:
      El diseño de arquitectura de este sistema proporcionará la habilidad de agregar nuevas funcionalidades al sistema.
    - Reutilización:
      La estructura de las clases y métodos utilizados en el sistema permitirá su reusó en aplicaciones similares.
    - Portabilidad:
      El sistema tendrá la capacidad de correr en distintos tipos de sistemas operativos: Windows, Mac OSX, Linux.
- Calidad de Uso
    - Usabilidad/Experiencia de usuario:
      El sistema contará con una interfaz intuitiva que le permitirá a cualquier usuario, con mínimos

conocimientos en computación, como manejo de mouse y teclado, utilizarla.