Aplicación Web multiusuario para la gamificación en el aula basada en Websocket

Autor: Alberto Jiménez Vázquez

Tutor: Francisco Javier Muñoz Calle

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Aplicación Web multiusuario para la gamificación en el aula basada en Websocket

Autor:
Alberto Jiménez Vázquez

Tutor:
Francisco Javier Muñoz Calle
Profesor Colaborador

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: Aplicación Web multiusuario para la gamificación en el aula basada en Websocket

Autor: Alberto Jiménez Vázquez

Tutor: Francisco Javier Muñoz Calle

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El secretario del Tribunal

A mi familia

A mis amigos

A mis maestros

Agradecimientos

Innumerables son las personas que directa o indirectamente me han sido de gran ayuda y apoyo durante la realización de este proyecto.

Me gustaría agradecer a mi familia, especialmente a mis padres Pasión Vázquez Martín y Luis Manuel Jiménez López, todo el apoyo y paciencia aportada, así como los consejos y constante interés en la realización de mi trabajo. A mis amigos con los cuales he compartido muchos de los mejores momentos de mi vida durante estos años de carrera, gracias por enseñarme que el mundo está lleno de grandes personas. Gracias a mi tutor Francisco Javier y a su hermano Jesús por la paciencia y confianza depositada en mí durante las extensas reuniones.

Mención especial para la persona que ha marcado un antes y un después en mi vida y que sin duda ha sido mi gran punto de apoyo durante la realización de este proyecto, mi pareja, Belén Rodríguez Santos.

Alberto Jiménez Vázquez

Sevilla, 2019

Resumen

El presente documento tiene por objeto recoger la especificación técnica y funcional del proyecto, cuyo objetivo es la implementación, despliegue y validación de una aplicación web basada en Websockets, que permita la gamificación multiusuario en las aulas mediante la participación en juegos educativos.

Más allá de este documento, se pretende lograr el desarrollo de un producto final que engloba las siguientes áreas funcionales:

- Acceso y registro de usuarios, creación, configuración y administración de partidas y resultados.
- Codificación y conversión en diferentes formatos de procesamiento de texto la estructura de los juegos.
- Interacción entre usuarios en los distintos juegos.

El proyecto DescartesJS, es una plataforma que permite el diseño de juegos por parte de los docentes y que da la posibilidad de participar en ellos a los alumnos dentro de las aulas, este proyecto recoge la integración realizada para permitir una interacción multiusuario con el motor de los juegos, recabado y almacenamiento de información en tiempo real, así como la comunicación con el resto de los servicios de las distintas áreas funcionales.

Abstract

The purpose of this document is to collect the technical and functional specification of the Project, whose objetive is the implementation, deployment and validation of a web application base don Websockets, which allow multi-user gamification in classrooms through participation in educational games.

Beyond this document, the aim is to achieve the development of a final product that includes the following functional areas:

- Access and registration of users, registration, configuration and administration of items and results.

- Coding and conversion in different formats of text processing the structure of the games.

- Interaction between users in the different games.

The DescartesJS project is a platform that allows the design of games by teachers and that gives the possibility to participate in them within the classrooms, this project includes the integration made to allow a multiuser interaction with the engine of the games, gathering and storage of information in real time, as well as communication with the rest of the services of the different functional areas.

Índice

Agradecimientos **9**

Resumen **11**

Abstract **13**

Índice **15**

Índice de Código **19**

Índice de Tablas **21**

Índice de Figuras **23**

1 Introducción **28**
1.1 La plataforma Descartes *28*
1.1.1 Herramienta DescartesJS 29
1.1.2 Aplicación de Juegos Didácticos en el Aula, AJDA 29
1.2 Alcance y objetivos *29*
1.3 Esquema general y contexto del proyecto *30*
1.4 Planificación temporal del proyecto *32*

2 Estado del Arte **35**
2.1 Comunicación Cliente-Servidor *36*
2.1.1 HTTP Polling 36
2.1.2 Long-Polling 37
2.2 Protocolo de comunicación Websockets *37*
2.2.1 Protocolo Websockets 37
2.2.2 API Websockets 42
2.2.3 Formato de texto JSON 45
2.3 AJDA y su integración con el proyecto *46*
2.4 Tecnología usada en el servidor *49*
2.4.1 Programación orientada a objetos, JAVA 49
2.4.2 Java Server Faces & Prime Faces, JSF/PF 54
2.4.3 Context Dependency Injection, CDI 57
2.4.4 Integración Weld y Tomcat 59
2.4.5 Apache Maven 60
2.4.6 Pruebas unitarias con JUnit 63
2.5 Tecnología usada en el cliente *64*
2.5.1 Hyper Text Markup Languaje, HTML 64
2.5.2 Cascading Style Sheets, CSS 64
2.5.3 JavaScript, JS 65
2.6 Tecnología servicio REST *66*
2.6.1 Servicio RESTful con Spring framework y Spring-Boot 66
2.6.2 Java Persistence API, JPA 68
2.6.3 Data Access Object, DAO 71
2.7 Gestión de la base de datos *72*

2.7.1 Modelo relacional en la gestión de bases de datos 72
2.7.2 PostgreSQL como sistema gestor 73
2.8 Servidor de aplicaciones *74*
2.8.1 Apache Tomcat 74

3 Herramientas utilizadas **76**
3.1 Spring Tool Suite, STS *77*
3.2 SoapUI *78*
3.3 PgModeler *80*
3.4 PgAdmin *83*
3.5 Diagramas con DrawIO *85*
3.6 Mozilla Firefox *86*

4 Análisis del Diseño **87**
4.1 Diagrama de componentes *87*
4.2 Diagramas de casos de uso *88*
4.3 Diagramas de flujo *91*
4.4 Diagramas de Secuencia *97*
4.5 Modelo E/R de la base de datos *102*

5 Interfaz de usuario **106**
5.1 Inicio de Partida *106*
5.2 Acceso de Jugadores *107*
5.3 Registro de Jugadores *109*
5.4 Formulario de respuesta *111*
5.5 Seguimiento de respuestas *112*
5.6 Estadísticas *113*
5.7 Fin de partida *113*

6 Validación **115**
6.1 Entorno de prueba *115*
6.1 Casos de pruebas unitarias *115*
6.1 Casos de pruebas integradas *118*
6.1 Conclusión *120*

7 Línea de mejora y conclusiones **121**
7.1 Mejoras *121*
7.2 Conclusiones *121*

Referencias **123**

Anexo A **126**
A.1 Sentencias SQL para la composición de la base de datos *126*

Anexo B **129**
B.1 Consumición del servicio REST *129*

Anexo C – JavaDoc del Proyecto **131**

ÍNDICE DE CÓDIGO

Código 2: Interfaz web Websockets. 42
Código 4: Inicialización objeto Websocket. 43
Código 3: Constructor Websocket. 43
Código 5: Estados de conexión. 43
Código 6: Estructura mensaje JSON. 45
Código 1: Variables necesarias DatosIniciales. 47
Código 7: Instalación Apache Maven. 60
Código 8: Estructura de un proyecto Maven. 62
Código 9: Empaquetado y compilación Maven. 62
Código 10: Instalación PostgreSQL. 73
Código 11: Gestión de usuario PostgreSQL. 73
Código 12: Acceso en host y reinicio PostgreSQL. 74
Código 13: Conexión con la base de datos. 74
Código 14: Instalación PgModeler. 80
Código 15: Instalación PgAdmin. 85
Código 16: Sentencias creación BBDD 01. 126
Código 17: Sentencias creación BBDD 02. 127
Código 18: Sentencias creación BBDD 03. 128
Código 19: Consumición del servicio REST 01. 129
Código 20: Consumición del servicio REST 02. 130

ÍNDICE DE TABLAS

Tabla 1: Estimación global. 33
Tabla 2: Técnicas utilizadas. 36
Tabla 3: Herramientas utilizadas. 76
Tabla 4: Navegadores que soportan Websocket. 86
Tabla 5: Definición de la información almacenada en la base de datos. 104
Tabla 6: Caso de prueba unitaria PU-1- 115
Tabla 7: Caso de prueba unitaria PU-2. 116
Tabla 8: Caso de prueba unitaria PU-3. 116
Tabla 9: Caso de prueba unitaria PU-4. 117
Tabla 10: Caso de prueba unitaria PU-5. 117
Tabla 11: Caso de prueba unitaria PU-6. 117
Tabla 12: Caso de prueba unitaria PU-7. 118
Tabla 13: Caso de prueba unitaria PU-8. 118
Tabla 14: Caso de prueba unitaria PU-9. 118
Tabla 15: Caso de prueba integrada PI-1. 119
Tabla 16: Caso de prueba integrada PI-2. 119
Tabla 17: Caso de prueba integrada PU-3. 120
Tabla 18: Caso de prueba integrada PU-4. 120

ÍNDICE DE FIGURAS

Ilustración 1: Diagrama de Gantt de la planificación del código del proyecto. 32
Ilustración 2: Diagrama de Gantt de la planificación de la memoria del proyecto. 32
Ilustración 3: Diagrama de Gantt de la planificación temporal de la presentación del proyecto. 33
Ilustración 4: Torre de protocolos HTTP [23]. 36
Ilustración 5: HTTP Polling [24]. 37
Ilustración 6: HTTP Long Polling [25]. 37
Ilustración 7: Diagrama de comunicación Websockets. 38
Ilustración 8: Request Cliente Websocket. 38
Ilustración 9: Datos de la cabecera HTTP. 39
Ilustración 10: Response del servidor al Handshake. 39
Ilustración 11: Estructura de una trama Websocket [26]. 40
Ilustración 12: Closing Handshake [27]. 41
Ilustración 13: Estructura cliente-servidor WebSocket. 44
Ilustración 14: Dependencia JSON. 46
Ilustración 15: Esquema de comunicación General. 46
Ilustración 16: Formato consultas AJDA. 48
Ilustración 17: Formato respuestas a consultas AJDA. 48
Ilustración 18: Formato resultados partida AJDA. 49
Ilustración 19: Java EE logo. 49
Ilustración 20: API Websocket en el servidor. 50
Ilustración 21: Evento OnOpen. 50
Ilustración 22: Evento OnMessage. 50
Ilustración 23: Evento OnClose. 51
Ilustración 24:Arquitectura basada en Microservicios. 52
Ilustración 25: Estructura proyecto Jugadores. 52
Ilustración 26: Estructura proyecto REST. 53
Ilustración 27: Estructura general proyecto Websocket. 53
Ilustración 28: Estructura WebApp Websocket. 54
Ilustración 29: Java Server Faces logo. 54
Ilustración 30: Dependencias JSF. 55
Ilustración 31:Configuración faces-config.xml. 55
Ilustración 32: Prime Faces logo. 56
Ilustración 33: Dependencias PrimeFaces. 56

Ilustración 34: Repositorio PrimeFaces en pom.xml. 56
Ilustración 35: Estructura web XHTML. 57
Ilustración 36: Logo CDI. 57
Ilustración 37: Dependencias CDI. 57
Ilustración 38: Context.xml para CDI. 58
Ilustración 39: Beans.xml para CDI. 58
Ilustración 40: Descriptor de despliegue Web.xml. 58
Ilustración 41: Anotaciones CDI. 59
Ilustración 42: Weld logo. 59
Ilustración 43: Dependencia para Weld. 60
Ilustración 44: Apache Maven logo. 60
Ilustración 45: Estructura pom.xml. 61
Ilustración 46: Repositorio Maven [28]. 62
Ilustración 47: JUnit logo. 63
Ilustración 48: Test de prueba unitaria para el servicio REST. 63
Ilustración 49: Test de prueba unitario para la interfaz DAO. 63
Ilustración 50: HTML logo. 64
Ilustración 51: CSS logo. 64
Ilustración 52: Media Query. 65
Ilustración 53: JavaScript logo. 65
Ilustración 54: Definición objeto Websocket en Javascript. 66
Ilustración 55: Definición eventos Websocket en Javascript. 66
Ilustración 56: RestController de la clase Jugador. 67
Ilustración 57: Clase Application para el servicio RESTful. 68
Ilustración 58: Dependencias Spring-Boot. 68
Ilustración 59: JPA logo. 69
Ilustración 60: Fichero de configuración JPA. 69
Ilustración 61: Dependencias JPA. 70
Ilustración 62: Entidades JPA. 70
Ilustración 63: Interfaz DAO. 71
Ilustración 64: Implementación DAO. 72
Ilustración 65: Esquema modelo relacional [29]. 73
Ilustración 66: PostgreSQL logo. 73
Ilustración 67: Apache Tomcat logo. 75
Ilustración 68: Dependencias Apache Tomcat. 75
Ilustración 69: Spring Tool Suite. 77
Ilustración 70: SoapUI logo. 78

Ilustración 71: Presentación SoapUI. 78
Ilustración 72: URI del servicio SoapUI. 79
Ilustración 73: Creación de la petición SoapUI. 79
Ilustración 74: Comprobación información obtenida SoapUI. 79
Ilustración 75: PgModeler logo. 80
Ilustración 76: Creación tabla PgModeler. 81
Ilustración 77: Creación columna PgModeler. 81
Ilustración 78: Definición columnas PgModeler. 82
Ilustración 79: Creación restricciones PgModeler. 82
Ilustración 80: Definición restricciones PgModeler. 82
Ilustración 81: Tabla PgModeler. 83
Ilustración 82: Ventana principal PgAdmin. 83
Ilustración 83: Estadísticas PgAdmin. 84
Ilustración 84: SQL Editor PgAdmin. 84
Ilustración 85: Histórico de consultas PgAdmin. 84
Ilustración 86: DrawIO logo. 85
Ilustración 87: Herramienta DrawIO. 85
Ilustración 88: Mozilla logo. 86
Ilustración 89: Diagarama de componentes. 87
Ilustración 90: Diagrama de casos de uso CU-001. 88
Ilustración 91: Diagrama de casos de uso CU-002. 89
Ilustración 92: Diagrama de casos de uso CU-003. 89
Ilustración 93: Diagrama de casos de uso CU-004. 90
Ilustración 94: Diagrama de casos de uso CU-005. 90
Ilustración 95: Diagrama de flujo DF-001. 91
Ilustración 96: Diagrama de flujo DF-002. 92
Ilustración 97: Diagrama de flujo DF-003. 93
Ilustración 98: Diagrama de flujo DF-004. 94
Ilustración 99: Diagrama de flujo DF-005. 95
Ilustración 100: Diagrama de flujo DF-006. 95
Ilustración 101: Diagrama de flujo DF-007. 96
Ilustración 102: Diagrama de flujo DF-008. 97
Ilustración 103: Diagrama de secuencia Registro Partida. 97
Ilustración 104: Diagrama de secuencia Seguimiento Respuestas. 98
Ilustración 105: Diagrama de secuencia para el Registro de Jugadores. 99
Ilustración 106: Diagrama de secuencia Juego de Preguntas. 100
Ilustración 107: Diagrama de secuencia Juego de Votaciones. 101

Ilustración 108: Diagrama modelo base de datos. 102
Ilustración 109: Vista comienzo de partida. 106
Ilustración 110: Vista solicitudes de acceso a partida. 107
Ilustración 111: Vista acceso a partida. 107
Ilustración 112: Vista formulario acceso jugadores. 108
Ilustración 113: Vista registro de jugador solicitado. 109
Ilustración 114: Vista administración registro jugadores. 109
Ilustración 115: Comienzo del juego. 110
Ilustración 116: Vista formulario de respuesta 01. 111
Ilustración 117: Vista formulario de respuesta 02. 111
Ilustración 118: Vista formulario de respuesta 03. 111
Ilustración 119: Iframe seguimiento de respuestas 01. 112
Ilustración 120: Iframe seguimiento de respuestas 02. 112
Ilustración 121: Vista estadísticas por pregunta 01. 113
Ilustración 122: Vista estadísticas por pregunta 02. 113
Ilustración 123: Vista juego finalizado. 114
Ilustración 124: Almacenamiento tras partida, entidad partida. 114
Ilustración 125: Almacenamiento tras partida, equipos. 114
Ilustración 126: Almacenamiento tras partida, grupos. 114
Ilustración 127: Almacenamiento tras partida, respuestas. 114

1 INTRODUCCIÓN

"Si quieres trabajadores creativos,
dales tiempo suficiente para jugar."
- John Cleese -

Hoy día vivimos la era de un mundo digital, donde todo está conectado. El desarrollo de la comunicación y de la transmisión de información han cambiado drásticamente la forma de vivir y de ver las cosas. Estamos ante lo que se conoce como la segunda gran revolución de nuestra sociedad moderna tras la revolución industrial del siglo XVIII.

Grandes, medianas y pequeñas empresas montan su servicio en la red con la finalidad de alcanzar un mercado mucho más amplio y variado, al igual, la administración pública utiliza plataformas y herramientas que permiten aumentar la eficiencia y la gestión de los distintos servicios gestionados, la telefonía, algo que hasta hace unos años parecía el futuro se encuentra hoy día en cada uno de nuestros bolsillos, o la banca, con aplicaciones que permiten mover cantidades de dinero en tan solo unos segundos y con un solo clic son algunos de los ejemplos que hoy día vivimos en la denominada era de Internet.

Hasta hace unas décadas el aprendizaje estaba fuertemente limitado, prácticamente para realizar una pequeña investigación se necesitaban horas de ardua lectura de algún libro o manual, en el mejor de los casos de tener acceso a ellos, hoy día las grandes cantidades de información a nuestro alcance permiten que con tan solo una búsqueda tengamos lo que necesitemos con un coste y esfuerzo relativamente bajos. Todo ello junto con la mejora de las técnicas de enseñanza y estudio han favorecido notablemente a la educación de los jóvenes, sector primordial para el cual este proyecto pretende aportar su pequeño granito de arena.

1.1 La plataforma Descartes

La Red Educativa Digital Descartes, RED Descartes [2], es una asociación no gubernamental sin ánimo de lucro que tiene como objetivo promover la renovación y cambio metodológico en los procesos de aprendizaje y enseñanza de las Matemáticas, y también en otras áreas de conocimiento en el aula, utilizando los recursos digitales interactivos generados en el Proyecto Descartes [1].

1.1.1 Herramienta DescartesJS

Descartes [3] es una herramienta que permite elaborar recursos didácticos interactivos que se embeben en páginas HTML, lo cual permite la interacción con dichos recursos por parte de cualquier dispositivo móvil, Tablet o PC que tenga acceso a internet y un navegador web.

Estos recursos no son imágenes animadas, sino que son escenas con las que el alumnado debe interactuar y en las cuales una reacción comienza con una acción por parte del alumno con la escena. Este concepto se describe muy bien con el símil que desde la web oficial [3] hacen con el teatro:

> "*No es lo mismo ser un espectador viendo una película que ser actor en una obra de teatro. Descartes aporta el escenario, el decorado, la infraestructura técnica, y es el usuario, nuestro alumnado y nosotros mismos, los que en cada escena han de abordar su papel de actor protagonista*".

A groso modo, el proyecto consiste en juegos (escenas) que el profesor crea y diseña para que sus alumnos puedan aprender y divertirse participando en ellos.

1.1.2 Aplicación de Juegos Didácticos en el Aula, AJDA

La herramienta AJDA [30] es un subproyecto del proyecto Descartes que tiene como objetivo el desarrollo de recursos interactivos basados en juegos y su aplicación didáctica en cualquier área o materia educacional. Con ellos, los contenidos educativos (preguntas, respuestas, palabras, cifras, frases, etc.) pueden generarse a través de formularios y guardarse en ficheros de texto que se catalogan y clasifican.

1.2 Alcance y objetivos

Hasta el momento, la forma de utilizar la herramienta era la siguiente:

- El profesor diseña un juego en forma de fichero de preguntas en formato de texto plano siguiendo una serie de directrices necesarias para su correcta interpretación por la herramienta. Aquí ya se puede identificar un problema, ya que la realización del fichero es tediosa y propensa a errores.
- Una vez que el juego es procesado, se muestra en pantalla a los alumnos, los cuales dan respuesta a la pregunta pertinente y es trabajo del profesor realizar una anotación de las respuestas de los jugadores para posteriormente cargarlas en el juego y que este procese los resultados, lo cual es una tarea ardua y engorrosa que lleva mucho tiempo.
- Finalmente, el juego muestra de forma básica los resultados de la partida.

Como objetivo global, se pretende desarrollar una aplicación web que permita una mejor gestión de las partidas realizadas, controlando el registro y almacenamiento de los resultados, así como integrar la interacción directa de los usuarios con el juego, facilitando notablemente las tareas de gestión del profesor, el cual únicamente deberá controlar el registro de jugadores y el control de las preguntas y el tiempo, el resto de las acciones serán automatizadas por la aplicación.

Esta integración multiusuario con la herramienta es el objeto de este proyecto, la cual se desarrollará mediante el uso de Websockets, además, se llevará a cabo un almacenamiento de datos de interés para análisis posteriores de los resultados obtenidos tras la participación de los jugadores en las partidas y serán publicados para su consulta por otros servicios a través de un servicio REST.

1.3 Esquema general y contexto del proyecto

En la siguiente ilustración se define el contexto general en el que se encuentra involucrado el proyecto (*Fig. 1*):

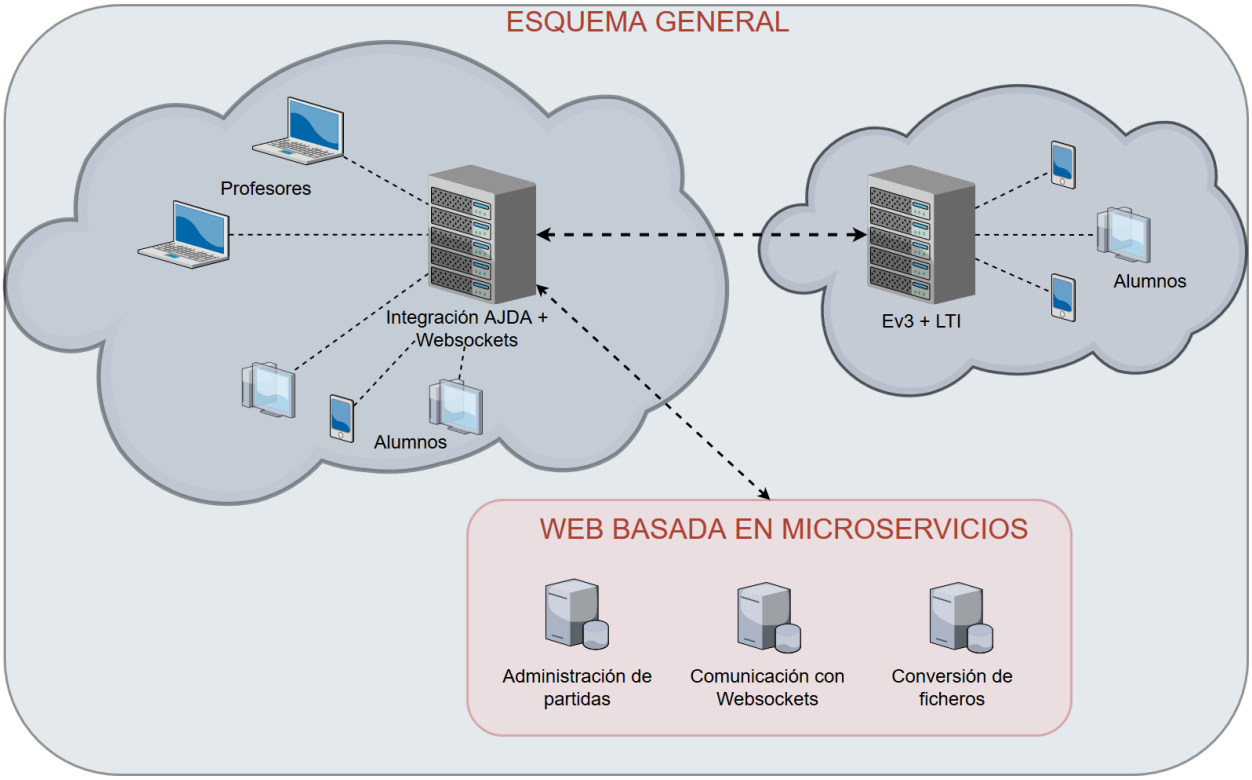


Ilustración 1: Esquema general del proyecto.

Por un lado, se ha realizado una integración de la aplicación AJDA con el servidor de Moodle de la enseñanza virtual de la universidad de Sevilla, cuyo objetivo es la integración de la herramienta con la web de la universidad para fines académicos, usando para ello LTI.

Por otro lado, se ha llevado a cabo el desarrollo de una aplicación web, en la que se integra la herramienta AJDA. El objetivo de esta aplicación es reducir la administración realizada por parte del profesor que crea las partidas, esta aplicación está basada en microservicios, los cuales se definen a continuación:

- **Microservicio de administración y gestión principal**: es el encargado de gestionar el registro y acceso de los usuarios del sistema, ofrecer un menú de creación y configuración de partidas, ofrecer la visualización de los resultados de las partidas, así como de almacenar información acerca de usuarios y configuraciones realizadas, que luego serán publicados a través de un servicio web basado en una API REST.
- **Microservicio de comunicación**: es el microservicio que se lleva a cabo en este proyecto, es el encargado de definir la interacción de los usuarios con la herramienta y otros usuarios. Define la comunicación entre profesores y el servidor de websockets para el registro de partidas en base a la información obtenida del microservicio de administración. Define la comunicación entre los profesores y jugadores a la hora de realizar el registro. Define la comunicación entre los jugadores con la herramienta AJDA a la hora de jugar. Además, almacena en su base de datos, información acerca de los jugadores, organización del juego (grupos, equipos) y estadísticas de respuesta por grupos que después serán publicados a través de un servicio web basado en una API REST.

- **Microservicio de conversión**: es el microservicio encargado de proveer a los profesores una herramienta de creación de ficheros de configuración. Los ficheros de configuración son los que recogen la información de cada tipo de juego y los que definen la estructura del juego. Actualmente los ficheros de configuración se recogían en texto plano propenso a errores, el objetivo de esta herramienta es proveer una conversión de los ficheros anteriores a ficheros basados en etiquetas XML. Este microservicio se nutre de la información almacenada en el microservicio de administración.

Un esquema resumen del proceso de uso de la aplicación web completa puede verse reflejado en la siguiente ilustración (*Fig. 2*):

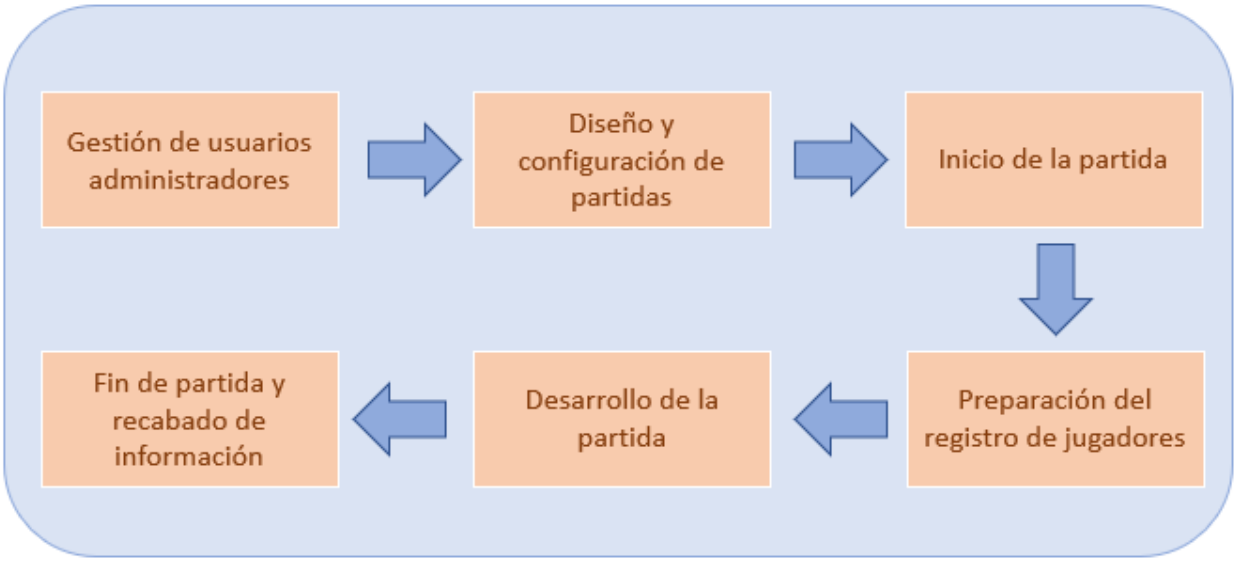


Ilustración 2: Proceso de uso de la aplicación web.

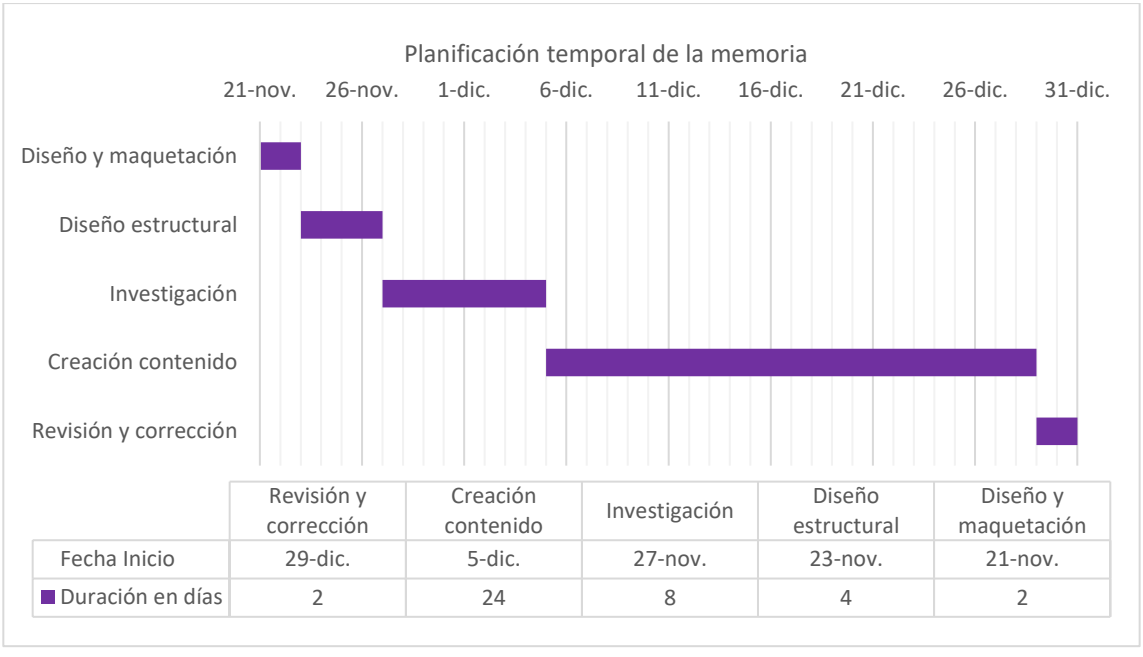
Inicialmente, la aplicación ofrecerá la posibilidad de gestionar quiénes serán los usuarios administradores del sistema, es decir, usuarios profesores, los cuales serán los encargados de determinar la segunda fase del proceso. El diseño y configuración de partidas es llevado a cabo por los profesores, para ello, definen las reglas y configuraciones que deben aplicarse a una partida en concreto, partiendo de un juego base previamente almacenado utilizando el servicio de conversión de ficheros. Una vez que la partida está definida puede ser almacenada para usarse más tarde o dar comienzo al inicio de la partida, este a su vez comienza con el registro de jugadores. El registro de jugadores es donde se produce por primera vez una comunicación entre jugadores y profesores, los jugadores comienzan a enviar solicitudes de acceso al servidor con sus nicks, grupos y equipos a los que quieren pertenecer y es el profesor el encargado de gestionar los grupos y equipos, distribuyendo a los jugadores a conveniencia del buen desarrollo de la partida. Una vez que el registro está organizado, el profesor da comienzo a la partida y es aquí donde tiene lugar la comunicación entre el juego y los jugadores. El juego va enviando preguntas y votaciones según configuración previa y los jugadores van respondiendo, todo ello supervisado por el profesor. La comunicación Jugador/Profesor y Juego/Jugador se sustenta del servicio de comunicación Websockets. Una vez finalizada la partida, el servicio de comunicación envía los resultados recabados de la partida al servicio de administración para ser almacenados, siendo los resultados individuales de cada jugador almacenados en el servicio de comunicación para la recogida de estadísticas individuales.

1.4 Planificación temporal del proyecto

La planificación temporal del proyecto puede verse recogida en los siguientes diagramas de Gantt:

	Depuración y errores	Validación y pruebas	Integración componentes	Desarrollo DAO	Desarrollo API REST	Desarrollo serv. cronómetro	Desarrollo serv. websocket	Desarrollo web servidor	Diseño web servidor	Desarrollo web cliente	Diseño web cliente	Instalación BBDD	Investigación	Entorno desarrollo
Fecha Inicio	1-nov.	1-nov.	8-ago.	6-ago.	2-ago.	30-jul.	23-abr.	15-mar.	13-mar.	2-mar.	28-feb.	25-feb.	15-ene.	15-ene.
Duración en días	20	20	100	25	40	20	150	13	1	10	1	2	40	3

Ilustración 1: Diagrama de Gantt de la planificación del código del proyecto.



	Revisión y corrección	Creación contenido	Investigación	Diseño estructural	Diseño y maquetación
Fecha Inicio	29-dic.	5-dic.	27-nov.	23-nov.	21-nov.
Duración en días	2	24	8	4	2

Ilustración 2: Diagrama de Gantt de la planificación de la memoria del proyecto.

Ilustración 3: Diagrama de Gantt de la planificación temporal de la presentación del proyecto.

El coste total de horas invertidas en la realización del proyecto se muestra en la siguiente tabla:

RESUMEN ESTIMACIÓN TEMPORAL (horas)				
REUNIONES	APLICACIÓN	MEMORIA	PRESENTACIÓN	**TOTAL**
16	450	130	26	**622**

Tabla 1: Estimación global.

2 ESTADO DEL ARTE

Internet facilita la información adecuada, en el momento adecuado, para el propósito adecuado.

- Bill Gates -

Dentro del marco tecnológico que nos encontramos hoy en día en las aplicaciones web y los diferentes servicios que se implementan, la arquitectura cliente-servidor es una de las más extendidas y utilizadas hasta el momento. Esto favorece el nacimiento de nuevas soluciones y la necesidad de disponer de una comunicación entre el cliente y el servidor consistente, libre de errores y lo más rápida posible.

En la siguiente tabla se recoge un resumen de las técnicas de desarrollo utilizadas en la realización del proyecto:

Técnica utilizada	Justificación de uso	Aplicación de la técnica
WEBSOCKET	Dada la necesidad de desarrollar una aplicación multiusuario en tiempo real, es necesaria la utilización de un protocolo de comunicación que permita la interacción entre usuarios.	Se ha utilizado en la comunicación realizada entre profesor/jugadores y juego/jugadores.
JSON	Definir una estructura generalizada en el formato de los mensajes, estandarizada y simple.	Formato de intercambio de mensajes realizados.
JAVA	Necesidad de un lenguaje de programación orientado a objetos que sea el núcleo de la lógica del servicio.	Desarrollo del Back-End de la aplicación.
JSF/PF	Necesidad de un lenguaje para definir la interfaz con la que interaccionarán los usuarios de la aplicación.	Interfaz de usuario web para la página de profesores.
CDI	Necesidad de un método de inyección de dependencias, tanto para manejar la información presentada en las vistas como para integrar los beans de respaldo con los servicios.	Integración de los componentes de la aplicación.
TOMCAT	Necesidad de un servidor de aplicaciones donde desplegar la aplicación.	Despliegue de la aplicación.

MAVEN	Facilitar la resolución de dependencias e integración de librerías y repositorios.	Resolución de dependencias, librerías y repositorios utilizados.
JUNIT	Definición de pruebas unitarias.	Definición de pruebas unitarias.
HTML	Necesidad de un lenguaje para definir la interfaz con la que interaccionarán los usuarios de la aplicación.	Interfaz de usuario web para la página de los jugadores.
CSS	Definir el estilo de las vistas.	Reglas de estilo para las páginas web de jugadores y profesores.
JAVASCRIPT	Definir la interacción del usuario con la vista.	Definir el comportamiento dinámico de las páginas ante interacciones con los usuarios.
SPRING-REST	Necesidad de un framework para el desarrollo de servicios web donde se publiquen para su consumo los datos de la aplicación.	Desarrollo de la API REST.
JPA	Unidad de persistencia que defina el mapeo de los objetos de la aplicación.	Almacenamiento de información en la base de datos.
DAO	Patrón de diseño para realizar buenas prácticas de desarrollo a la hora de aplicar métodos CRUD a la base de datos.	Mapeo entre objetos Java y la base de datos.
POSTGRESQL	Sistema gestor de base de datos que nos permita la creación y administración de una base de datos para el almacenamiento.	Gestor de la base de datos utilizada.

Tabla 2: Técnicas utilizadas.

2.1 Comunicación Cliente-Servidor

De estas necesidades de comunicación en internet surge el protocolo HTTP, creado por la *World Wide Web* en 1996. Este protocolo permite la comunicación de datos entre el lado cliente y el lado del servidor mediante el intercambio de mensajes sin estado.

Ubicación en la pila de protocolos	
Aplicación	HTTP
Transporte	TCP
Red	IP

Ilustración 4: Torre de protocolos HTTP [23].

2.1.1 HTTP Polling

Una de las variantes de comunicación consiste en la técnica de Polling, donde en el lado cliente se realiza una petición de datos al servidor, independientemente de si esos datos han sido actualizados o no, y de forma periódica.

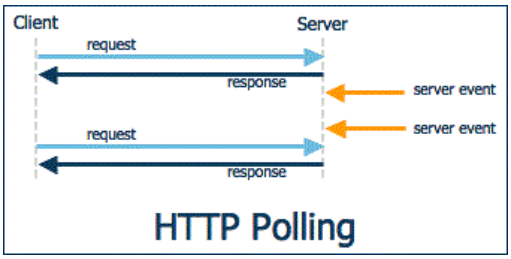


Ilustración 5: HTTP Polling [24].

Esta técnica provoca llamadas innecesarias al servidor, ya que pueden realizarse llamadas sin que en el servidor los datos hayan sido actualizados.

2.1.2 Long-Polling

La técnica de *Long-Polling* consiste en una variante de la técnica de *Polling*, donde se pretende mejorar la eficiencia de la comunicación. Para ello, a la hora de realizar la petición al servidor, este en lugar de devolver una respuesta vacía almacena la petición de forma que cuando se produzca una modificación en los datos este envíe la respuesta al cliente.

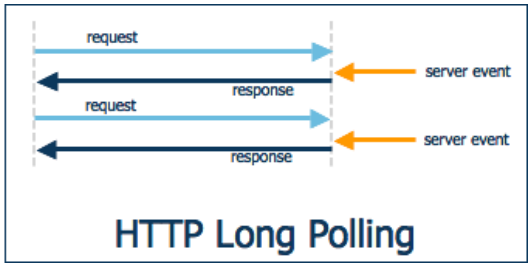


Ilustración 6: HTTP Long Polling [25].

Esto reduce el número de peticiones realizadas al servidor, sin embargo, obliga a mantener abiertas las conexiones con el servidor lo cual repercute en el rendimiento y en la latencia de los datos.

2.2 Protocolo de comunicación Websockets

Como alternativa a estas técnicas de comunicación nace el protocolo Websocket, ***ws***, que permite establecer una comunicación continua, bidireccional y full-duplex entre el cliente y el servidor sobre un canal TCP. Esta especificación está regulada por la *IETF*, *Internet Engineering Task Force* y se encuentra recogida en la *RFC 6455* [3].

2.2.1 Protocolo Websockets

El protocolo se divide en tres etapas: **apertura de conexión** o *Handshake*, **envío de datos** y **cierre de conexión.**

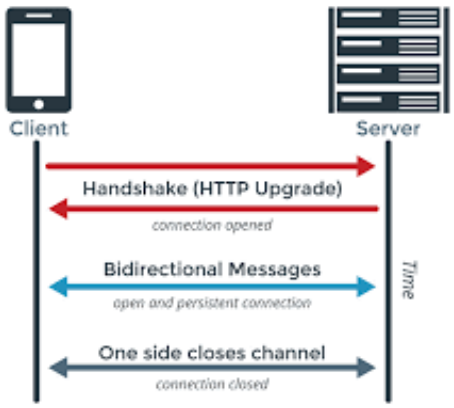


Ilustración 7: Diagrama de comunicación Websockets.

2.2.1.1 Apertura de conexión

La apertura de conexión se realiza mediante el uso del protocolo HTTP. En este caso, es el cliente el que solicita el inicio de la comunicación con el siguiente mensaje:

```
Accept: text/html,application/xhtml+xm…plication/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Cache-Control: no-cache
Connection: keep-alive, Upgrade
Cookie: JSESSIONID=19A88047566CCF56BE263AC9119BDFF4
Host: localhost:8081
Origin: http://localhost:8081
Pragma: no-cache
Sec-WebSocket-Extensions: permessage-deflate
Sec-WebSocket-Key: zKuB4CzK/Y7O+Q74TdMqkA==
Sec-WebSocket-Version: 13
Upgrade: websocket
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linu…) Gecko/20100101 Firefox/62.0
```

Ilustración 8: Request Cliente Websocket.

La definición de los campos más representativos de esta petición es la siguiente:

- **Connection:** incluye el token de la petición, puede ser un token de mantenimiento de conexión o un token de apertura de conexión.
- **Host:** autoridad del servidor.
- **Sec-WebSocket-Key:** clave necesaria para la validación de respuesta del servidor.
- **Sec-Websocket-Version:** versión actual del protocolo.
- **Upgrade:** protocolo, siempre websocket.
- **User-Agent: (opcional)** información del navegador.

- **Cookie: (opcional)** incluye el identificador único de sesión, *JSESSIONID*.
- **Origin: (opcional)** enviado por el navegador del cliente, contiene la URL raíz de la página a la que se accede.

El servidor se encarga de procesar la petición y enviar la respuesta al cliente:

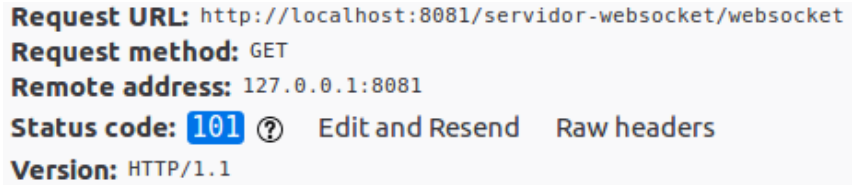


Ilustración 9: Datos de la cabecera HTTP.

Ilustración 10: Response del servidor al Handshake.

Los campos más representativos de la respuesta son los siguientes:

- **Connection**: token de la petición, en este caso de apertura de conexión.
- **Date**: fecha de apertura de conexión.
- **Server**: servidor web, en este caso *Apache-Coyote*, un conector para Tomcat que soporta el protocolo HTTP 1.1 como servidor web.
- **Upgrade:** protocolo, siempre websocket.
- **Sec-Websocket-Accept:** clave de verificación.
- **Sec-Websocket-Extensions:** extensiones de nivel de protocolo acordados entre el cliente y el servidor para el uso de la conexión.

Las cabeceras *Sec-Websocket-Key* y *Sec-Websocket-Accept* juegan un papel fundamental en la apertura de conexión, ya que para que dicha conexión se produzca el servidor debe utilizar el código de la Key para crear un código de validación (Accept). Esto se realiza concatenando la Key con el GUID, un código alfanumérico de identificación único con formato: *AAAAAAAA-BBBB-CCCC-DDDD-EEEEEEEEEEEE*. Posteriormente se calcula el hash SHA-1 de la concatenación y se envía al cliente codificado en base64 en la cabecera *Sec-Websocket-Accept*. Esto permite validar que se ha recibido la respuesta correcta por parte del servidor.

Si el *Handshake* se realiza correctamente y los campos en la cabecera de petición del cliente son correctos, el servidor devolverá una respuesta con código de estado *101 OK*, indicando que la conexión queda establecida.

2.2.1.2 Envío de datos

La estructura de una trama de datos Websocket se corresponde con la siguiente imagen:

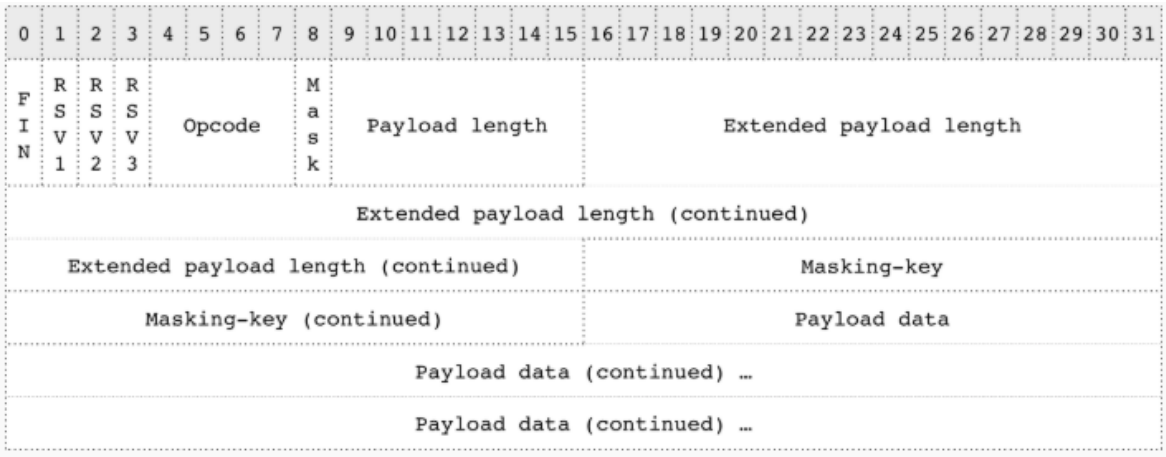


Ilustración 11: Estructura de una trama Websocket [26].

La definición de los campos que componen la trama es la siguiente:

- **FIN:** indica que es el fragmento final de un mensaje.
- **RSV:** utilizados en la negociación de las extensiones del protocolo.
- **Opcode:** define la interpretación de los datos de la carga útil (texto, binario, etc).
- **Mask:** vale 1 en caso de que los datos estén enmascarados.
- **Payload length:** longitud de los datos de la carga útil.
- **Extended payload length:** longitud de los datos de extensión.
- **Masking-key:** clave de enmascaramiento de datos de 32 bits elegida arbitrariamente por el cliente.
- **Payload-data:** incluye los datos de extensión y los datos de aplicación (carga útil).

Para que pueda producirse el envío de datos, deben de cumplirse los siguientes requisitos:

- El punto final de comunicación debe asegurarse tener una conexión Websocket en estado *OPEN*.
- Los datos deben ser encapsulados según el marco de trama definida anteriormente.
- El código de operación de la trama, *Opcode*, debe establecerse correctamente en función de la tipología de datos que se transmitan.
- El bit *FIN* del último segmento del mensaje debe establecerse a 1.
- Los datos transmitidos por el cliente deben viajar enmascarados utilizando la *Masking-Key*.

2.2.1.3 Cierre de conexión

Cualquier punto final de comunicación puede iniciar el cierre de conexión, ya sea cliente o servidor. El punto final que inicia el cierre envía una petición de *Closing Handshake* al punto extremo, una vez recibido, este responde con otro mensaje de cierre. A partir de este momento se inicia el fin de la session TCP y la conexión queda cerrada.

El diagrama de comunicación puede verse representado en el siguiente diagrama:

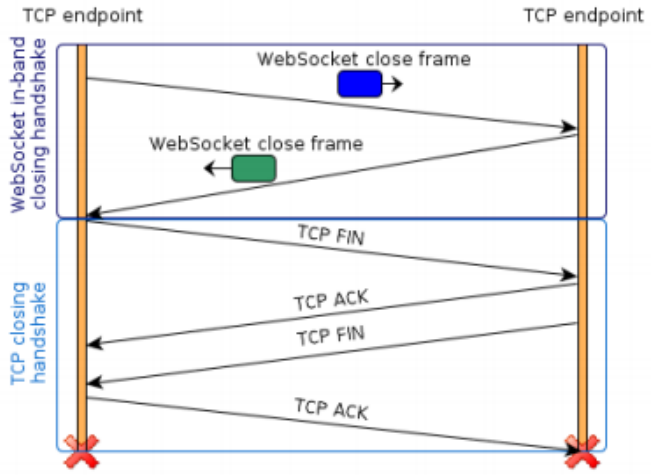


Ilustración 12: Closing Handshake [27].

En las tramas de cierre de conexión debe indicarse:

- Código de estado de cierre:
  - 1000: cierre normal.
  - 1002: error de protocolo.
  - 1007: mensaje incoherente.
  - 1009: trama demasiado grande.
  - 1010: fallo en negociación de extensiones.
- Razón de cierre (**opcional**).

Cualquier trama recibida posteriormente al cierre de conexión será desechada.

2.2.1.4 Consideraciones de seguridad

- **Clientes que no son navegadores**: protección frente a la ejecución de JavaScript malicioso y ataques de inyección SQL procedentes de falsos clientes. Para ello el servidor debe verificar que el sitio web que solicita conexión es un sitio web conocido mediante la validación del campo origen. En caso de detectar un falso origen el servidor debe dar respuesta con ***HTTP 403 Forbidden.***
- **Ataques de envenenamiento a proxies**: sistemas intermediarios como proxies, enrutadores y otros sistemas como ISPs reciben los mensajes Websockets. Estos sistemas en ocasiones procesan y corrigen estos mensajes al tratarse de mensajes muy similares a HTTP y recomponen el mensaje a veces respondiendo en base a la propia caché. Si por ejemplo un atacante enviara un mensaje parcialmente erróneo similar a una trama HTTP podría envenenar la caché de estos sistemas. Por tanto, como solución se propone el enmascaramiento de las tramas de información de los mensajes Websockets con la finalidad de que no sean reconocidas por estos sistemas intermedios y estos a su vez actúen de enrutadores en vez de proceder al procesamiento.

- **Denegación de servicio por trama gigante**: implementación de un límite en el tamaño de los mensajes con el fin de evitar el desbordamiento de la memoria debido a ataques de denegación del servicio.
- **Autenticación**: uso de mecanismos de autenticación del cliente, como por ejemplo cookies, autenticación HTTP genérica o TLS.
- **Datos inválidos**: comprobación de los datos enviados por los puntos finales. Estos siempre deben ser datos correctos y esperados, en caso contrario se procede al cierre de la conexión.

2.2.2 API Websockets

La API Websocket [4] para desarrollar aplicaciones que permitan la conexión entre navegador y servidor está definida por el estándar de la W3Schools y presenta la siguiente interfaz:

```
[Constructor(DOMString url, optional (DOMString or DOMString[]) protocols)]
interface WebSocket : EventTarget {

  readonly attribute DOMString url;

  // READY STATE
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSING = 2;
  const unsigned short CLOSED = 3;

  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;

  // NETWORKING
  attribute EventHandler onopen;
  attribute EventHandler onerror;
  attribute EventHandler onclose;
  readonly attribute DOMString extensions;
  readonly attribute DOMString protocol;

  void close([Clamp] optional unsigned short code, optional DOMString
reason);

  // MESSAGING
  attribute EventHandler onmessage;
  attribute DOMString binaryType;

  void send(DOMString data);
  void send(Blob data);
  void send(ArrayBuffer data);
  void send(ArrayBufferView data);
};
```

Código 1: Interfaz web Websockets.

El constructor del objeto Websocket recibe dos argumentos, una URL que define cómo conectarse con el servidor y un argumento opcional que es un array de sub-protocolos, de forma que puedes implementar un servidor que soporte protocolos diferentes en función de las necesidades:

```
WebSocket WebSocket(
  in DOMString url,
  in optional DOMString protocols
);
```

Código 3: Constructor Websocket.

```
var exampleSocket = new WebSocket("ws://www.example.com/socketserver",
["protocolOne", "protocolTwo"]);
```

Código 2: Inicialización objeto Websocket.

Un ejemplo de creación de un objeto Websocket sería la siguiente:

La URL está formada por tres partes:

- Método de seguridad: ***ws*** para HTTP, ***wss*** para HTTPS.
- Dirección del servidor.
- Ruta local/alias.

El array de protocolos se define mediante un array de strings especificando el nombre de cada protocolo, en el caso de este proyecto no se han utilizado sub-protocolos.

El atributo ***readyState*** define el estado de la conexión y puede tener uno de los siguientes valores:

```
CONNECTING (valor numérico 0)
      La conexión aún no ha sido establecida.
OPEN (valor numérico 1)
      Comunicación establecida.
CLOSING (valor numérico 2)
      La conexión está en estado de cierre, se ha recibido el HandShake
      de cierre o se ha invocado al método close().
CLOSED (valor numérico 3)
      Conexión cerrada.
```

Código 4: Estados de conexión.

El atributo ***extensions*** se inicializa con una cadena vacía en el momento de establecer la conexión. Una vez que la conexión ha sido establecida se utiliza para almacenar las extensiones del protocolo definidas por el servidor, en caso de que se hayan negociado previamente.

El atributo ***protocol*** se inicializa con una cadena vacía en el momento de establecer la conexión. Una vez que la conexión ha sido establecida se utiliza para almacenar el sub-protocolo definido por el servidor, en caso de que se hayan definido sub-protocolos.

El método ***close([****código cierre****],[****razón cierre****])*** es utilizado para solicitar el cierre de la conexión,

este método lo podemos invocar con dos parámetros opcionales:

- Código de cierre: puede ser un entero igual a 1000 o en el rengo de 3000-4999, en caso contrario lanza *InvalidAccesError* y se aborta el cierre.

- Razón de cierre: no puede ser superior a 123 bytes, en caso contrario se lanza *SyntaxError* y se aborta el cierre.

El método ***send([datos])*** es utilizado para enviar datos al servidor, estos pueden ser enviados con distinto formato (*DOMString*, *Blob*, *ArrayBuffer* y *ArrayBufferView*). En este proyecto los datos enviados al servidor serán objetos JSON contenidos en cadenas de *DOMString*.

El atributo ***bufferedAmount*** indica el tamaño en bytes que se han encolado tras la ejecución de método ***send([datos])***.

El atributo ***binaryType*** permite definir la forma en la que vamos a enviar información al servidor en forma de datos binarios. Estos datos pueden enviarse en dos formatos, ***Blob*** (definido por defecto) o ***ArrayBuffer***.

Los manejadores de evento [6], ***EventHandler***, son funciones de callback que se ejecutan cuando un evento ha sido lanzado por el servidor. Dichas funciones reciben el evento como parámetro, este evento puede ser:

- ***Onopen:*** se ejecuta cuando la conexión ha sido establecida.
- ***Onmessage:*** se ejecuta cuando recibimos un mensaje del servidor, de este objeto obtenemos los datos recibido en el formato establecido.
- ***Onerror:*** se ejecuta cuando se ha producido un error, del evento podemos obtener la información del error producido.
- ***Onclose:*** se ejecuta cuando se cierra la conexión.

Siguiendo las pautas definidas por esta especificación, podemos desarrollar un cliente Websocket mediante el uso de JavaScript, que nos permita la comunicación con un servidor WebSocket.

La estructura cliente servidor implementada sigue el siguiente esquema:

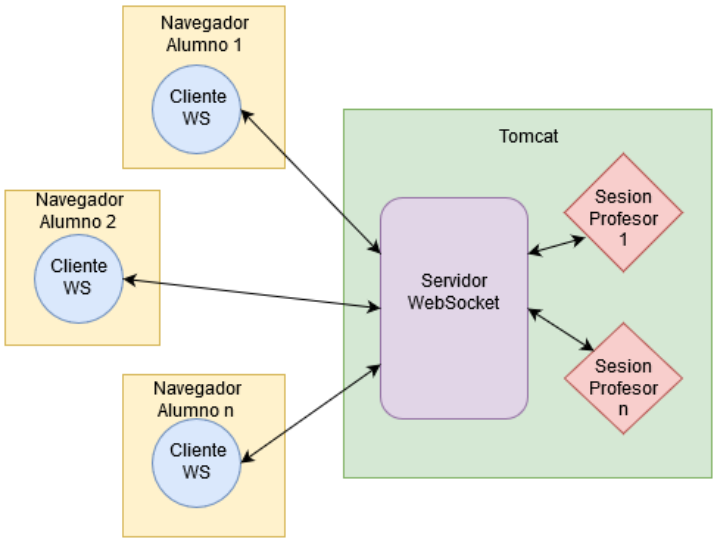


Ilustración 13: Estructura cliente-servidor WebSocket.

Un servidor Websocket permitirá la interacción de profesores y alumnos con la interfaz del juego. Cada alumno tendrá una sesión Websocket volátil (no se mantiene la sesión de usuario) con el servidor que le permitirá comunicarse con el juego para el registro y la respuesta a preguntas y votaciones. Y cada profesor tendrá una sesión Websocket no volátil (mantiene la sesión del usuario en la aplicación) con el servidor que le permitirá organizar el registro de jugadores y la administración de la partida.

2.2.3 Formato de texto JSON

JSON, JavaScript Object Notation, es un formato ligero y simple de intercambio de datos. Es sencillo de leer y escribir para las personas y poco costoso de interpretar y procesar por las aplicaciones.

Los tipos de datos disponibles con JSON son:

- **Números**: Se permiten números negativos y opcionalmente decimales separados por punto.
- **Cadenas**: Representan secuencias de cero o más caracteres. Se ponen entre doble comilla y se permiten cadenas de escape.
- **Booleanos**: pueden tener dos valores: true y false.
- **null**: Representan el valor nulo.
- **Array**: Representa una lista ordenada de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes.
- **Objetos**: Son colecciones no ordenadas de pares nombre-valor separados por comas y puestas entre llaves. El nombre tiene que ser una cadena entre comillas. El valor puede ser de cualquier tipo.

La estructura de los mensajes definidos en este proyecto cumple con la siguiente especificación:

```
{
    "evento": "nombre_evento"  // Nombre del evento/mensaje

    "parámetro_1": "valor",
                ...            // Listado de parámetros necesarios.
    "parámetro_1": "valor",

    "objeto_dato_1": {
        "valor_1": "valor",
        "valor_2": "valor",
                ...            // Características del objeto.
        "valor_n": "valor"
    }
                ...            // Listado de objetos necesarios.
    "objeto_dato_2": {
        "valor_1": "valor",
        "valor_2": "valor",
                ...            // Características del objeto
        "valor_n": "valor"
    }
}
```

Código 5: Estructura mensaje JSON.

Todo mensaje intercambiado entre cliente y servidor debe tener un campo ***"evento"*** de tipo obligatorio, el cual determina la tipología del mensaje enviado, y un conjunto de parámetros y objetos variables que contienen los datos a procesar.

Para integrar las librerías JSON a nuestro proyecto, debemos incluir en nuestro pom.xml la siguiente dependencia:

```
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20180130</version>
</dependency>
```

Ilustración 14: Dependencia JSON.

La utilización de JSON para el proyecto viene incentivada por el uso de una estructura estandarizada y simple para la composición de mensajes entre aplicaciones. No obstante, hay otros estándares en el mercado como XML o YAML que también ofrecen características similares.

2.3 AJDA y su integración con el proyecto

A continuación, se muestra el esquema general de comunicación entre la aplicación Websocket y la herramienta *AJDA* [30] *(Fig. 15):*

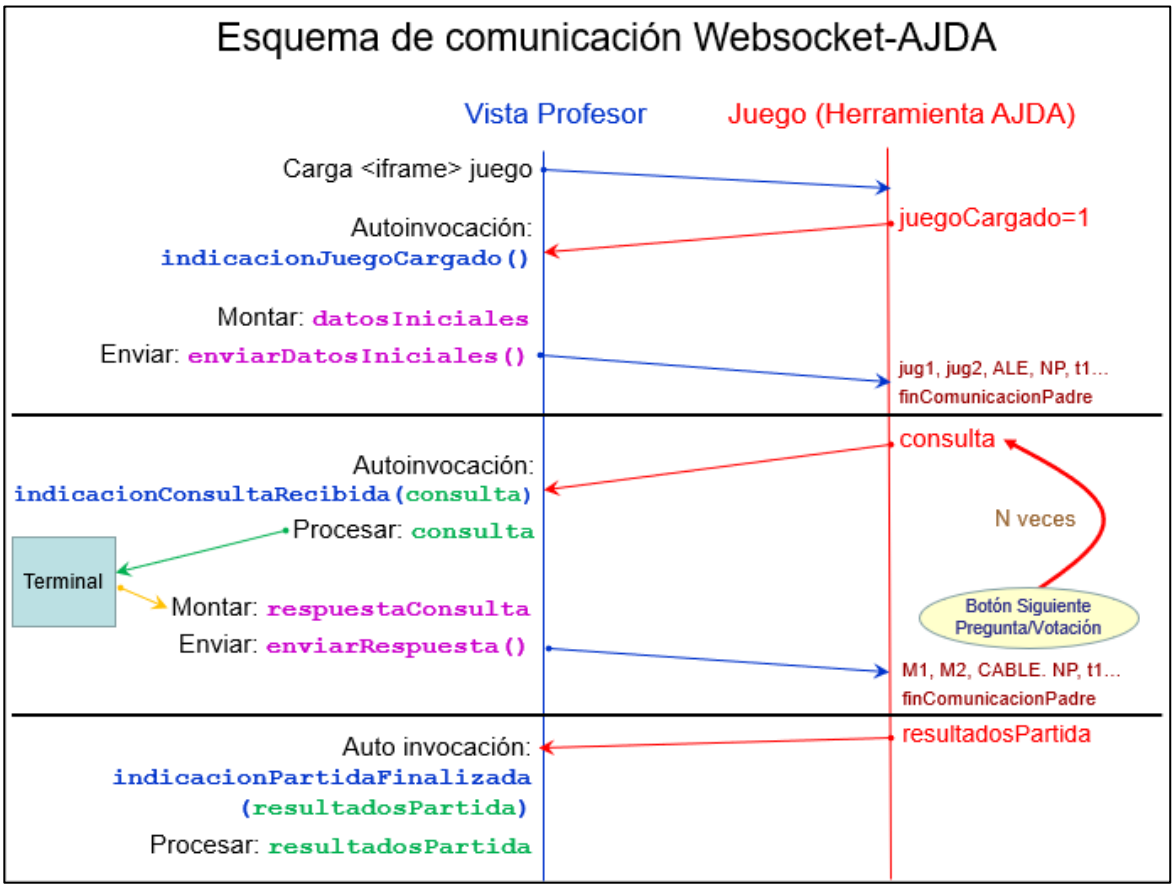


Ilustración 15: Esquema de comunicación General.

La comunicación se divide en tres etapas:

- Una primera etapa en la que se carga la aplicación *AJDA* en un *iframe* en la pantalla del profesor. La aplicación necesita del objeto *DatosIniciales* para la configuración del juego, este objeto es obtenido por el servidor Websocket de un servicio externo y enviado al juego a través de una API Javascript externa de comunicación.

- Una vez que comienza la partida, el juego va enviado las consultas (preguntas o votaciones) a la vista del profesor para que sean procesadas y retransmitidas a los alumnos. A su vez, la respuesta de los jugadores es enviada al juego para su procesamiento.

- Al finalizar la partida, con la información recabada de las respuestas de los jugadores, el juego enviará un resultado final al servidor para que sea procesado y almacenado en la base de datos.

Las variables que necesita el juego y que se envían en el objeto *DatosIniciales* se muestran en el siguiente fragmento:

```
-- Variables necesárias en el objeto DatosIniciales

Jug                 // Número de grupos
jug[1-n]            // Nombre del grupo
pun[1-n]            // Puntos iniciales del grupo
ALE                 // Preguntas aleatorias (0-SI, 1-NO)
NP                  // Número de preguntas
MAT                 // Opción de Jaque Mate (1-SI, 0-NO)
BOTE                // Puntos del bote del juego
TOTALENVI           // contiene los datos del fichero de preguntas
CONTIPAR            // 0 (partida nueva), 1 (continuación)
CONTINUARPARTIDA    // Datos del fichero de continuación de partida
TIPORESPUESTA       // 0 (AUTOMÁTICA Y MANUAL), 1 (SOLO AUTOMÁTICA)
TIPOCORRECCION      // 0 (AUTOMÁTICA), 1 (MANUAL)

-- Variables de uso exclusivo en el servidor

PORCENCORRECCION  // Porcentaje de corrección aplicado a respuestas
TIEMPOPREGUNTA    // Tiempo para contestar a cada pregunta
TIEMPOVOTACION    // Tiempo para contestar a cada consulta o votación
```

Código 6: Variables necesarias DatosIniciales.

El formato de las consultas enviadas por el juego al servidor sigue un formato de texto JSON tal y como se muestra en la siguiente ilustración *(Fig. 16):*

```
consulta =
{
     "pregunta":
     {  "tipo":                  0,    // 0 (Opciones),  1 (Cifra),  2 (Texto)
        "numeroPregunta":        7,    // ID de la pregunta
        "numeroOpciones":        4,    // nº de opciones de respuesta a la pregunta
        "correcta":              "B", // Respuesta correcta y notación de la misma
        "gruposVarRespuesta":    [ ["jug1", // Variable con la que el juego
                                            identifica al grupo
                                   "M1"],  // Variable en la que juego espera la
                                             respuesta del grupo
                                  ["jug2", "M2"] ],
         "repeticionPregunta":   0 // 0 () 1 (indica que la pregunta se formula de nuevo)
         "tiempoRespuesta":      20 // Si se indica 0 se toma el tiempo por omisión

     },

     "votación":
     {
        "tipo":                  0,    // 0 (Opciones),  1 (Cifra)
        "opciones":              ["Rojo","Amarillo","Azul","Verde"], //opciones
        "gruposVarVotacion":     [ ["jug1",  // Variable con la que el juego
                                            identifica al grupo
                                   "CABLE"], // Variable en la que juego espera la
                                               respuesta del grupo
                                  ["jug3", "CABLE"] ],
         "repeticionVotacion":   0 // 0 () 1 (la votación se formula de nuevo)
         "tiempoVotacion":       0 // 0 se toma el tiempo por omisión
     }
}
```

Ilustración 16: Formato consultas AJDA.

El formato de respuesta por parte del servidor, una vez procesada la respuesta de los jugadores, sigue el siguiente formato *(Fig. 17):*

```
respuestaConsulta =
{
     "pregunta":
         {
           "M1": 3,    // Variable que recibe la respuesta de la pregunta.
                          La respuesta puede ser opciones, Cifra o Texto (ver
                          datos recibidos del juego para la consulta).
           "M2": ...  //Dependiendo el juego puede haber una o más variables
         },

     "votación":
         {
           "CABLE1": 3,  // Variable que recibe la respuesta de la votación.
                          La respuesta puede ser opciones o Cifra o Texto (ver
                          datos recibidos del juego para la consulta).
           "CABLE2": ...  //Dependiendo el juego puede haber una o más variables
         }
}
```

Ilustración 17: Formato respuestas a consultas AJDA.

Finalmente, los resultados de la partida que el juego envía al servidor Websocket, siguen el siguiente formato *(Fig. 18):*

```
resultadosPartida=
{
        "Globales":     [14,    // Nº Preguntas Formuladas
                        100],   // Puntos máximos que se pueden alcanzar

        "grupo1":       ["jug1", // Variable que identifica al grupo
                        4,      // Nº Respuestas correctas
                        3,      // Nº Respuestas incorrectas (o no contestadas)
                        100,    // Puntos del grupo
                        0,      // Grupo Ganador (1-SI, 0-NO)
                                // CAMPOS Opcionales:
                        1],     // Grupo Eliminado (1-SI, 0-NO)

        "grupo2":       ["jug2", // Variable que identifica al grupo
                        4,      // Nº Respuestas correctas
                        3,      // Nº Respuestas incorrectas (o no contestadas)
                        100,    // Puntos del grupo
                        0,      // Grupo Ganador (1-SI, 0-NO)
                                // CAMPOS Opcionales:
                        1]      // Grupo Eliminado (1-SI, 0-NO)
        ...
}
```

Ilustración 18: Formato resultados partida AJDA.

2.4 Tecnología usada en el servidor

2.4.1 Programación orientada a objetos, JAVA

Java Platform Enterprise Edition es una plataforma para el desarrollo de aplicaciones software en el lenguaje de programación Java que se ejecutan sobre un servidor de aplicaciones. Java EE tiene infinidad de especificaciones que permiten desarrollar aplicaciones y servicios web multiplataformas.

Ilustración 19: Java EE logo.

La implementación del servidor de Websocket se ha desarrollado siguiendo la definición del estándar Java EE 7 para la API Websocket.

La definición de la clase principal que actuará de servidor y desde la cual se aceptarán las conexiones y el procesamiento de mensajes debe estar anotada con las siguientes anotaciones [10]:

- **@ApplicationScope**: es necesario definir un ámbito de aplicación para el servidor, ya que será necesario durante todo el periodo de vida de la aplicación.
- **@ServerEndpoint**: con esta anotación estaremos indicando al Application Server que esta será nuestra clase servidor y que escuchará peticiones Websockets en la URL ***http://{HOST:PORT}/application/alias***, donde *HOST* es la dirección de la máquina donde se hospede el servidor, *PORT* un puerto libre de escucha, *application* es el *artifactId* del proyecto y alias la ruta que denotemos en la etiqueta.

```java
@ServerEndpoint("/websocket")
@ApplicationScoped
public class WebsocketController implements Serializable {

    private static final long serialVersionUID = 1L;

    // Listado de conexiones
    static final List<Session> conexiones = new ArrayList<>();

    // Listado de partidas en curso
    static final List<PartidaGlobal> partidas_globales = new ArrayList<>();

    // Mapeo [partida - SessionID de profesor]
    static final Map<Integer, String> mapaCookies = new HashMap<Integer, String>();
```

Ilustración 20: API Websocket en el servidor.

Para una correcta implementación del servicio:

- La clase servidor debe implementar la interfaz serializable y definir un *serialVersionUID*.
- Almacenar el listado de conexiones con el servidor actualmente abiertas.
- Definir un mapeo de las sesiones de los profesores con su ID de sesión para la reapertura de conexiones.

Para la gestión de la comunicación debemos definir los métodos con sus respectivas anotaciones, previamente definidas en la sección del protocolo Websocket, que permitan el procesamiento de eventos:

```java
@OnOpen
public void iniciaSesion(Session session) {
    // En caso de ser una estadistica la agregamos por separado para poder enviarles el tiempo y refresco
    if(session.getQueryString() != null && session.getQueryString().equals("estadisticas")) {
        if(mapaCookies.containsValue(((WsSession) session).getHttpSessionId())) {
            // Recorremos el mapa para localizar la partida
            for(Map.Entry<Integer, String> entry : mapaCookies.entrySet()) {
                if (entry.getValue().equals(((WsSession) session).getHttpSessionId())) {
                    buscaPartidaGlobalPorIdPartida(entry.getKey()).añadirConEstadistica(session);
                }
            }
        }
    }
    conexiones.add(session);
}
```

Ilustración 21: Evento OnOpen.

El método *inciaSesion* está marcado con la anotación ***@OnOpen*** y se ejecutará cada vez que una nueva conexión se establezca con el servidor, el parámetro *sesion* representa la conexión y la almacenamos en la lista de conexiones.

```java
@OnMessage
public void onMessage(String mensaje, Session sesion) {

    System.out.println("Mensaje recibido: " + mensaje);

    // Procesamos el evento que nos llega
    try {
        procesarEvento(sesion, mensaje);
    } catch (JSONException e) {
        e.printStackTrace();
    }

}
```

Ilustración 22: Evento OnMessage.

El método *onMessage* está marcado con la anotación ***@OnMessage***, será ejecutado cada vez que

llegue un nuevo mensaje de alguno de los navegadores de los alumnos. Este recibe como parámetro el mensaje enviado como cadena JSON y el objeto *sesion* que mandó el mensaje. Cada vez que se reciba un mensaje será procesado por el método *procesarEvento.*

```
@OnClose
public void finConexion(Session session) {
    try {
        // Cerramos la conexión
        session.close();
        // Se retira de la lista
        conexiones.remove(session);
        // Tenemos que eliminar al jugador de la partida (excepto profesores que pueden reconectar)
        try {
            if (!esSesionProfesor(session) && session.getQueryString() != null && !session.getQueryString().equals("estadisticas")) {
                borrarJugadorPorDesconexion(session);
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    } catch (IOException ex) {
    }
    System.out.println("Conexion cerrada.");
}
```

Ilustración 23: Evento OnClose.

El método *finConexion* está marcado con la anotación ***@OnClose*** y se ejecutara cada vez que una sesión se desconecte. Recibe el objeto *sesion* que se desconectó para poder identificarla. En función de la sesión que se desconecte (profesor o alumno) se realizará una acción u otra.

2.4.1.1 Arquitectura MVC basada en microservicios

El proyecto se rige por el patrón de diseño MVC, un patrón de diseño software que propone una separación de funcionalidades en tres componentes: el modelo de datos, la lógica de negocio y la lógica de presentación.

La ***vista*** se implementa mediante los ficheros XHTML donde se definen los componentes JSF que finalmente se convierten a páginas HTML, estas son mostradas al usuario por el navegador. Un aspecto fundamental es la conexión de las vistas con la aplicación mediante los *Beans* gestionados. Esta conexión se realiza mediante el lenguaje de expresiones JSF EL, con este lenguaje se definen los *bindings* ente las propiedades del *bean* y los componentes de la vista.

El ***controlador*** se implementa mediante el código Java del servidor Websocket, es el encargado de responder ante los eventos del usuario e invocar las peticiones de información al modelo. Es el intermediario entre la vista y el modelo.

El ***modelo*** es el encargado de la representación y almacenado de la información, gestiona los accesos y actualizaciones de esta a través de las peticiones de acceso que el controlador le transmite por parte de la vista.

La arquitectura del proyecto está basada en Microservicios, un enfoque para desarrollar una única aplicación como un conjunto de pequeños servicios, ejecutados de forma independiente, con un lenguaje independiente y comunicándose, por lo general, a través de una API HTTP.

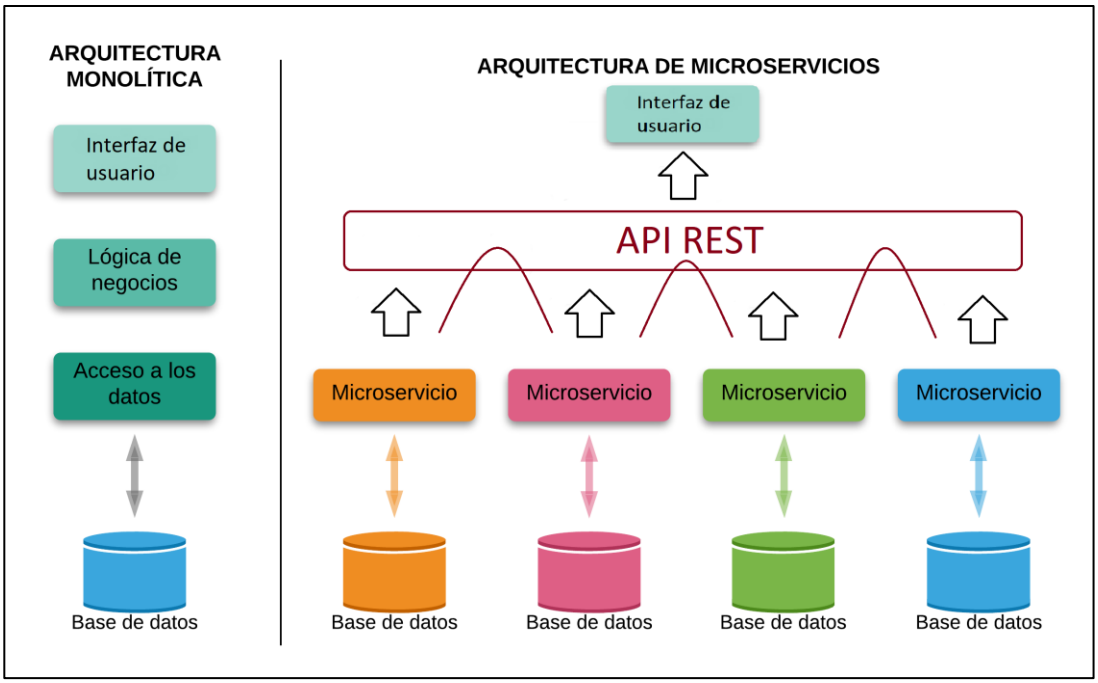


Ilustración 24:Arquitectura basada en Microservicios.

Cada microservicio es un sistema independiente, con una base de datos independiente, en concreto el microservicio del proyecto nanjea información sobre:

- Jugadores, grupos, equipos y partidas registradas.
- Estadísticas de respuestas a votaciones y preguntas.

2.4.1.2 Estructura de los proyectos

A continuación, se detalla la estructura de carpetas y ficheros de los proyectos eclipse que componen la aplicación desarrollada.

El proyecto *acceso-jugadores* es el más simple en estructura, se trata de un proyecto web basado en páginas con etiquetado HTML dentro de la ruta */WebContent* del proyecto, definición de estilos CSS en */WebContent/estilos* y la interacción del usuario con el DOM de la página en */WebContent/javascript*. Además, dispone de una carpeta de recursos de imagen en */WebContent/imágenes*.

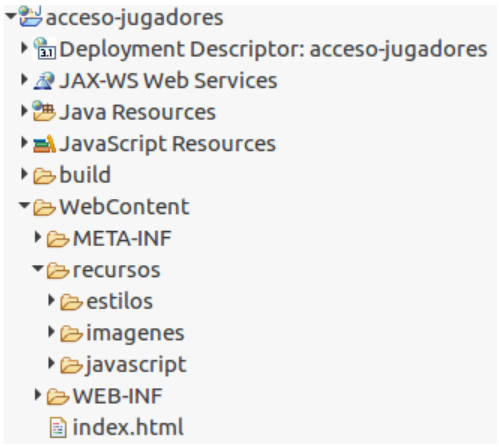


Ilustración 25: Estructura proyecto Jugadores.

En el proyecto *servicio-rest* se define el modelo de la aplicación. Se compone de las entidades, que son las clases que se mapean con las tablas en la base de datos, las interfaces de acceso a objetos DAO y los controladores del servicio REST para la publicación de una API REST. Además, se incluyen una serie de pruebas unitarias basadas en Junit.

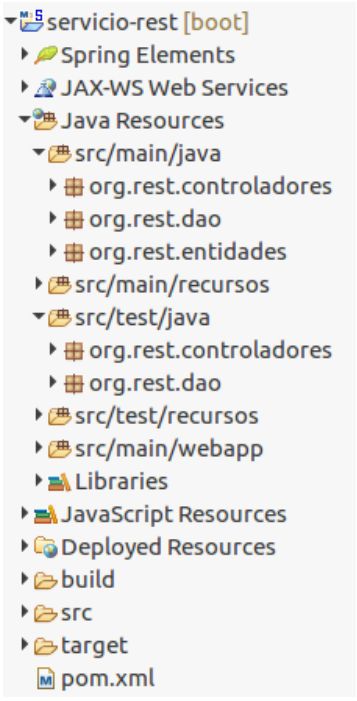


Ilustración 26: Estructura proyecto REST.

El proyecto *servidor-websocket* contiene la lógica de negocio (controlador) y las vistas de la aplicación (vista):

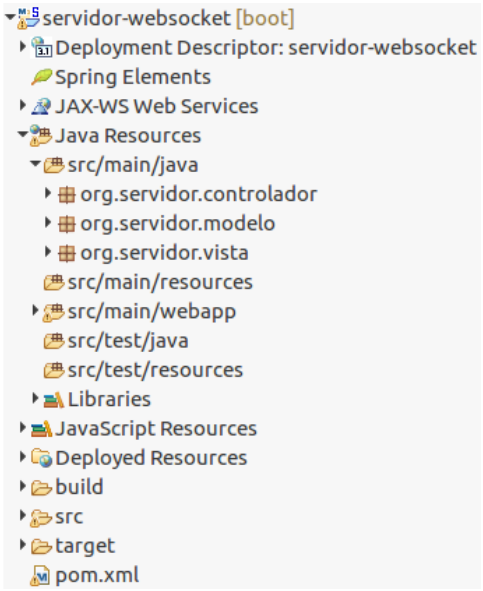


Ilustración 27: Estructura general proyecto Websocket.

En la estructura web del proyecto se encuentran los ficheros XHTML con los componentes JSF/PF, los recursos CSS y una serie de librerías JavaScript para la interacción con la plataforma Descartes: común.js, descartes-min.js y comunicación.js ficheros propios de la herramienta *AJDA*, estadísticas.js y envio-variables.js contienen la lógica JavaScript para la interacción con el DOM de

la página y la definición del cliente Websocket de la parte del profesor.

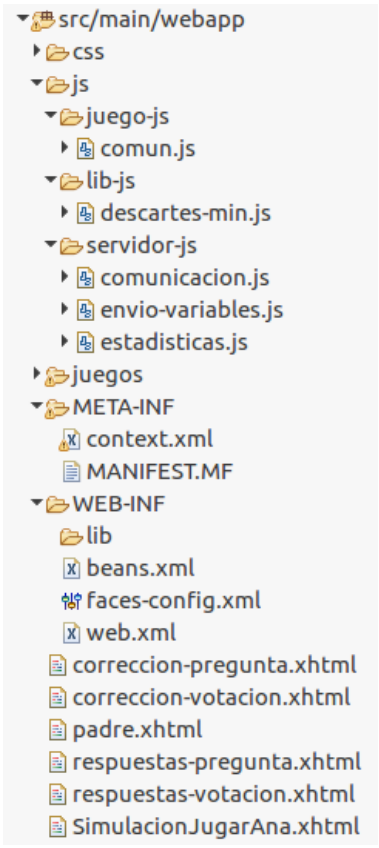


Ilustración 28: Estructura WebApp Websocket.

El lenguaje de programación orientado a objetos JAVA es el lenguaje más extendido a nivel mundial, su estandarización para la programación de la lógica de negocio de las aplicaciones y la cantidad de funcionalidades aportadas la han convertido en el lenguaje principal seleccionado para el desarrollo del proyecto. Otras alternativas podrían ser C++, Ruby o Phyton.

2.4.2 Java Server Faces & Prime Faces, JSF/PF

JSF [9] es un framework MVC que ofrece un conjunto de componentes en forma de etiquetas, las cuales se definen en páginas XHTML. Esta definición de etiquetas está basada en el framework de *Facelets* que permite la creación de componentes compuestos mediante el uso de plantillas predefinidas. JSF no requiere de servicios adicionales, simplemente un contenedor de *Servlets*. En este proyecto se ha usado Tomcat. Como contenedor.

Ilustración 29: Java Server Faces logo.

JSF proporciona las siguientes características:

- Definición de las interfaces de usuario mediante vistas que contienen los componentes gráficos.

- Beans gestionados para la conexión de los componentes gráficos.
- Navegación entre vistas.
- Conversión de datos y validación automática de la entrada del usuario.
- Internacionalización.
- Modelo estándar de comunicación Ajax entre la vista y el servidor (a partir de JSF 2.0).

Como implementación de JSF a utilizar, se ha optado por usar *Mojarra*, especificación de referencia desarrollada por Sun y que se mantiene activa y estable hasta la actualidad.

Para el uso de JSF en nuestro proyecto, son necesarias las siguientes dependencias en nuestro pom.xml:

```
<!-- Dependencias para usar JSF -->
<dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>2.2.0</version>
</dependency>
<dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-impl</artifactId>
    <version>2.2.0</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.0</version>
    <scope>provided</scope>
</dependency>
```

Ilustración 30: Dependencias JSF.

Además, será necesario crear y definir el fichero faces-config.xml en nuestra carpeta de proyecto */WEB-INF* de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
    version="2.2">

</faces-config>
```

Ilustración 31:Configuración faces-config.xml.

En este fichero se define la configuración de:

- *Beans* de la aplicación, sus nombres y propiedades iniciales.
- Reglas de validación de los componentes de entrada.
- Reglas de navegación entre distintas páginas de la aplicación.
- Ficheros de recursos para la internacionalización de la aplicación.

Sin embargo, lo dejaremos vacío ya que nos aprovecharemos de las anotaciones que nos provee el framework.

Adicionalmente a la implementación JSF, diferentes autores definen sus librerías de componentes, como por ejemplo RichFaces, IceFaces, etc. En este proyecto se utiliza PrimeFaces [8], una librería de componentes visuales que ofrecen una mayor versatilidad, la cual está desarrollada y mantenida por *Prime Technology*.

Ilustración 32: Prime Faces logo.

Las características que ofrece PrimeFaces son las siguientes:

- Un amplio conjunto de componentes.
- Soporte AJAX para una mejor interacción del usuario con la página.
- Amplia documentación y difusión de la librería.
- Uso de Javascript no intrusivo.
- Proyecto open source.
- Compatibilidad con otras librerías de componentes.

Para el uso de PrimeFaces en nuestro proyecto, son necesarias las siguientes dependencias en nuestro pom.xml:

```
<!-- Dependencias para usar PrimeFaces -->
<dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>6.2</version>
</dependency>
```

Ilustración 33: Dependencias PrimeFaces.

Para incluir el repositorio de Primefaces, incluimos el siguiente fragmento en nuestro pom.xml:

```
<repositories>
    <repository>
        <id>prime-repo</id>
        <name>PrimeFaces Maven Repository</name>
        <url>http://repository.primefaces.org</url>
    </repository>
</repositories>
```

Ilustración 34: Repositorio PrimeFaces en pom.xml.

Además, nuestro fichero debe tener extensión XHTML y seguir la siguiente estructura:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui">

     <h:head>
        <!-- Cabeceras/Librería/Scripts -->
     </h:head>

     <h:body>
         <!-- Componentes JSF/PrimeFaces -->
     </h:body>
</html>
```

Ilustración 35: Estructura web XHTML.

El uso de Java Server Faces y Prime Faces en el proyecto viene determinado por las características, anteriormente comentadas, que ofrecen. Hay otras alternativas totalmente lícitas para el desarrollo del proyecto, como podrían ser RichFaces o Thymeleaf.

2.4.3 Context Dependency Injection, CDI

CDI o Context Dependency Injection [11], define un mecanismo para resolución de dependencias entre servicios dentro del estándar JavaEE para versiones superiores a la versión 6. CDI nos permite administrar el ciclo de vida de los componentes e inyectar servicios en los beans controlados.

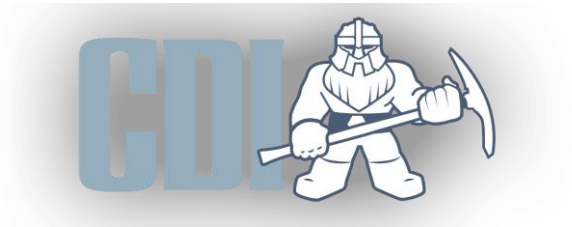


Ilustración 36: Logo CDI.

Para el uso de CDI en nuestro proyecto, son necesarias las siguientes dependencias en nuestro pom.xml:

```
<dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-web-api</artifactId>
    <version>7.0</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.enterprise</groupId>
    <artifactId>cdi-api</artifactId>
    <version>2.0</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.jboss.weld.servlet</groupId>
    <artifactId>weld-servlet-shaded</artifactId>
    <version>3.0.3.Final</version>
</dependency>
```

Ilustración 37: Dependencias CDI.

Además, es necesaria la creación de los siguientes ficheros de configuración:

- Fichero context.xml en la ruta */META-INF* de nuestro proyecto, en el definiremos el *BeanManager* que nos permite realizar las operaciones necesarias para tratar con el contexto:

```
<Context>
    <Resource name="BeanManager"
        auth="Container"
        type="javax.enterprise.inject.spi.BeanManager"
        factory="org.jboss.weld.resources.ManagerObjectFactory" />
</Context>
```

Ilustración 38: Context.xml para CDI.

- Fichero beans.xml en el directorio */WEB-INF*, donde se definen los beans para las inyecciones y que en el desarrollo de este proyecto se define vacío ya que se aprovechará la facilidad en el uso de anotaciones:

```
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
       http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
       version="1.2" bean-discovery-mode="annotated">

    <!-- some content -->
</beans>
```

Ilustración 39: Beans.xml para CDI.

- Fichero Web.xml en la ruta */WEB-INF* como descriptor de despliegue, donde se define como desplegar la aplicación web. Una vez definido el uso de JSF y CDI debe quedar con la siguiente configuración:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    id="WebApp_ID" version="3.1">
    <display-name>servidor-websocket</display-name>

    <!-- Welcome page -->
    <welcome-file-list>
        <welcome-file>index.xhtml</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>

    <resource-env-ref>
        <!-- Enable Weld CDI, also needs META-INF/context.xml entry -->
        <resource-env-ref-name>BeanManager</resource-env-ref-name>
        <resource-env-ref-type>javax.enterprise.inject.spi.BeanManager</resource-env-ref-type>
    </resource-env-ref>

    <context-param>
        <param-name>primefaces.FONT_AWESOME</param-name>
        <param-value>true</param-value>
    </context-param>

</web-app>
```

Ilustración 40: Descriptor de despliegue Web.xml.

Entre muchas de las funcionalidades que nos ofrece CDI, una de las más interesantes es la gestión del ámbito de los beans de respaldo de la vista, lo cual se realiza mediante el uso de anotaciones:

- **@RequestScoped**: el periodo del bean es el mismo que el de la petición HTTP. Una vez finalizada la petición, el bean es destruido.
- **@SessionScoped**: el bean es compartido por todas las peticiones HTTP que pertenezcan a la misma sesión.
- **@ApplicationScoped**: el bean es creado en el arranque de la aplicación y su periodo de vida no termina hasta que no finalice la aplicación.
- **@ConversationScoped**: permite mantener una instancia de un mismo bean durante varias peticiones HTTP, sin tener que ser compartido por todas las peticiones de la misma sesión. Es posible demarcar el ámbito de inicio y fin de la transacción.
- **@Singleton**: se mantiene una instancia única del bean.
- **@Dependent**: anotación por defecto. El ámbito es heredado del bean en el cual se inyecta.

Las principales anotaciones usadas en el proyecto son las siguientes:

```
@Named("someBean")
@RequestScoped
public class SomeBean {

@Inject
private SomeService someService;
}
```

Ilustración 41: Anotaciones CDI.

- **@Named(*"nombre_bean"*)**: anotación que nos permite connotar un nombre a nuestro bean.
- **@{*Ambito*}Scoped**: anotación para la definición del ámbito del bean.
- **@Inject**: anotación que permite la inyección entre clases.

El manejo de los Beans que respaldan las vistas de nuestra aplicación puede realizarse tanto usando el framework ofrecido por JSF como con CDI. Sin embargo, se ha optado por usar la definición CDI ya que ofrece una mayor flexibilidad a la hora de definir los ámbitos de las clases y sobre todo por su aplicación en la inyección de dependencias entre diferentes especificaciones. Simplemente usando la especificación de JSF no era posible integrar las implementaciones de JSF con la API Websocket. CDI por el contrario permite la integración entre ambas tecnologías lo cual da una mayor flexibilidad a la hora del desarrollo del proyecto.

2.4.4 Integración Weld y Tomcat

Weld es la implementación de referencia de CDI, un estándar para la inyección de dependencias y la gestión del ciclo de vida de las clases de nuestro proyecto.

Ilustración 42: Weld logo.

Comúnmente, Weld viene integrado en servidores de aplicaciones puramente Java EE como por ejemplo TomEE, Tomcat, al ser simplemente un contenedor de Servlet no puramente Java EE es necesaria la integración de Weld con la siguiente dependencia en nuestro pom.xml:

```
<dependency>
    <groupId>org.jboss.weld.servlet</groupId>
    <artifactId>weld-servlet-shaded</artifactId>
    <version>3.0.3.Final</version>
</dependency>
```

Ilustración 43: Dependencia para Weld.

Inicialmente, se planteó la aplicación de resolución de dependencias mediante el uso que ofrecía el framework de Java Server Faces, sin embargo, finalmente se optó por usar Context Dependency Injection ya que permite la integración de componentes dentro del estándar JavaEE, mientras que JSF ofrecía una API totalmente independiente.

2.4.5 Apache Maven

Apache Maven [7] es una herramienta para la generación y gestión de proyectos software, entre sus funciones más destacadas se encuentran:

- Gestión de dependencias.
- Gestión de repositorios.
- Compilación.
- Empaquetado.
- Generación de documentación.
- Realización de tests.

Ilustración 44: Apache Maven logo.

La instalación de Maven es rápida y sencilla, solo basta con descargar la versión elegida desde la web oficial de Apache Maven y ejecutar los siguientes comandos en nuestro terminal Linux para su correcta instalación:

La descripción del proyecto se define en el fichero de configuración de Maven, pom.xml, en él se definen todas las características que debe cumplir un proyecto Maven:

```
~ cd /opt
~ tar -xzvf /Descargas/Maven-x.x.x-bin.tar.gz
~ export PATH=$PATH:/opt/Maven-x.x.x/bin
~ mvn --version
```

Código 7: Instalación Apache Maven.

```
<project ...>
    <modelVersion>version_modelo</modelVersion>
    <groupId>org.etsi.demo</groupId>
    <artifactId>artefacto</artifactId>
    <packaging>jar</packaging>
    <version>version</version>
    <name>nombre</name>
    <dependencies>
        ...
        <dependency>
            <groupId>---</groupId>
            <artifactId>---</artifactId>
            <version>---</version>
            <scope>---</scope>
        </dependency>
        ...
    </dependencies>
    ...
</project>
```

Ilustración 45: Estructura pom.xml.

- **modelVersion**: versión para Maven.
- **groupId**: define la estructura de paquetes del proyecto.
- **artifactId**: nombre del proyecto.
- **packaging**: indica el tipo de empaquetado del proyecto, jar, war, ear, etc.
- **version**: indica la versión del proyecto.
- **name**: suele emplearse el nombre del proyecto.
- **dependencies**: dentro de esta etiqueta se definen las etiquetas de tipo *dependecy* las cuales son necesarias para el uso de librerías necesarias en el proyecto.

Una configuración del fichero pom.xml anterior nos generará un proyecto Maven con la siguiente estructura predefinida para todos los proyectos Maven:

```
artefacto
|-- pom.xml
|-- src
    |-- main
    |    |-- java
    |        |-- org
    |            |-- etsi
    |                |-- demo
    |                    |-- Class.java
    |-- test
        |-- java
            |-- org
                |-- etsi
                    |-- demo
                        |-- ClassTest.java
```

Código 8: Estructura de un proyecto Maven.

Una vez definida la configuración del pom.xml, la compilación y empaquetado del proyecto es muy sencilla (estas funciones también se encuentran integradas dentro del IDE STS):

```
~ mvn compile      // compila el proyecto
~ mvn test         // compilación y ejecución de tests
~ mvn package      // empaqueta el proyecto
~ mvn install      // guarda el proyecto en el repositorio
~ mvn clean       // borra el directorio de salida
```

Código 9: Empaquetado y compilación Maven.

Para finalizar, Apache Maven cuenta con un amplio repositorio, sencillo e intuitivo de utilizar en el que con solo realizar una búsqueda te permite obtener las diferentes dependencias para tu proyecto:

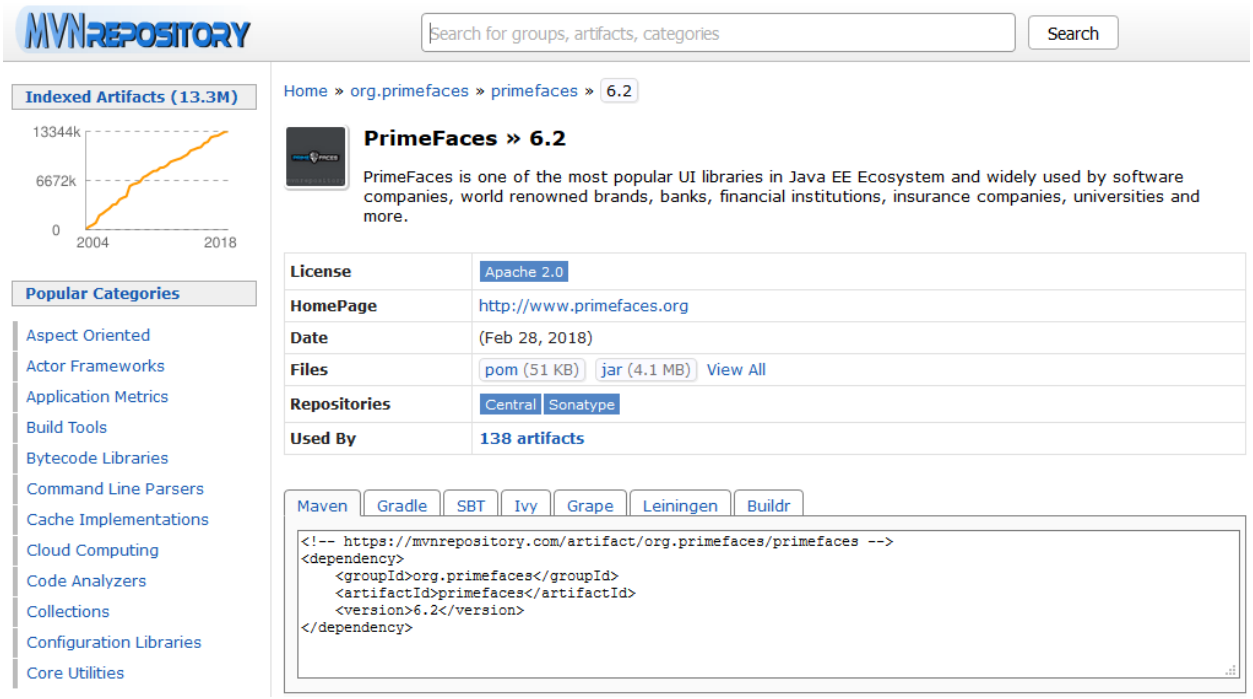


Ilustración 46: Repositorio Maven [28].

La utilización de Maven en el proyecto se reduce a un sistema que permita reducir el costo de integración de las distintas librerías para el proyecto. Otra posible alternativa es Gradle con una gestión mucho más simplificada y un mejor rendimiento mediante una caché de descarga de dependencias y un daemon interno a la hora de construcción de los proyectos.

2.4.6 Pruebas unitarias con JUnit

Junit es un framework que permite la ejecución de clases de forma controlada para la realización de pruebas unitarias sobre nuestros componentes software.

Ilustración 47: JUnit logo.

Simplemente debemos definir una clase de pruebas y anotar los métodos de prueba con la anotación ***@Test*** para que el módulo de JUnit realice la gestión. Con la clase ***Assert*** definimos los resultados esperados. Para la ejecución basta con añadir el plugin de Junit a nuestro STS.

Los tests de prueba han sido implementados para los controladores del servicio REST y la interfaz de acceso DAO *(Fig. 48, 49),* con la finalidad de evaluar la conectividad con el servicio y la correcta lectura y escritura en la base de datos:

```
@Test
public void getPartidasTest() throws ClientProtocolException, IOException {

    // URL a probar
    HttpUriRequest request = new HttpGet("http://localhost:8080/getPartidas");

    // Lanzamos la peticion GET
    HttpResponse httpResponse = HttpClientBuilder.create().build().execute(request);

    // Comprobamos que devuelve 200 OK
    Assert.assertEquals(httpResponse.getStatusLine().getStatusCode(), 200);
}
```

Ilustración 48: Test de prueba unitaria para el servicio REST.

```
@Test
public void createRespuestaPreguntaTest() {

    // Creamos el objeto RespuestaPregunta
    RespuestaPregunta rp_persistir = new RespuestaPregunta();
    rp_persistir.setIdJugador(2);
    rp_persistir.setIdPartida(12345);
    rp_persistir.setOpcionRespuesta("A");
    rp_persistir.setNumeroPregunta(2);
    rp_persistir.setCorrecta(true);

    // Persistimos
    respuestaPreguntaDAO.createRespuestaPregunta(rp_persistir);

    // Buscamos la RespuestaPregunta
    RespuestaPregunta rp_persistido = respuestaPreguntaDAO.getRespuestaPreguntaByPartidaPreguntaJugador(12345, 2, 7)
            .get(0);

    // Pasamos el test
    Assert.assertEquals(rp_persistido.getOpcionRespuesta(), rp_persistir.getOpcionRespuesta());

}
```

Ilustración 49: Test de prueba unitario para la interfaz DAO.

Se ha escogido JUnit para la realización de las pruebas unitarios para el proyecto ya que es el estándar de JEE. Las pruebas realizadas no son costosas y su ejecución se realiza en cuestión de segundos. Sin embargo, existen otras alternativas en el mercado que pueden mejorar el rendimiento

de ejecución y la facilidad de desarrollo de los tests como por ejemplo Selenium o Mockito.

2.5 Tecnología usada en el cliente

2.5.1 Hyper Text Markup Languaje, HTML

Hyper Text Markup Languaje o HTML es un lenguaje basado en etiquetas que compone el elemento más básico de una página web. Es utilizado para la creación y representación del contenido de la página (texto, imágenes, vídeos, etc.) pero no determina su funcionalidad. Fue lanzado inicialmente en 1993 y actualmente se encuentra en su versión HTML5.2 regulado por la World Wide Web Consortium, W3C.

Ilustración 50: HTML logo.

Su uso en el proyecto ha sido aplicado para definir la interfaz gráfica de los jugadores que participen en los juegos de la aplicación. Se ha utilizado para definir los formularios de acceso a partida, registro del jugador, respuesta a preguntas y votaciones y finalmente la publicación de resultados.

2.5.2 Cascading Style Sheets, CSS

Cascading Style Sheets o CSS es un lenguaje para la definición de la presentación de un documento escrito usando lenguajes de marcado como HMTL, XML y sus derivados. Es utilizado para realizar el diseño visual de los elementos de la página web. Fue lanzado inicialmente en 1996 y actualmente se encuentra en su versión CSS3 regulado por la World Wide Web Consortium, W3C.

Ilustración 51: CSS logo.

Su uso en el proyecto ha sido aplicado para la definición del estilo y maquetación de la página y, mezclado con JavaScript, para los efectos de las transiciones en el tiempo restante de respuesta a pregunta o votación. Un aspecto particular de CSS son las **media query**, un módulo CSS3 que permite adaptar la presentación del contenido a las características de los dispositivos. Dentro de las

media query se pueden usar operadores lógicos y condiciones, en función de los cuales el diseño de la página se verá alterado.

```
@media (max-width: 600px) {
  .formulario_respuesta {
    width: 200px;
  }
}
```

Ilustración 52: Media Query.

En esta media query de ejemplo hemos definido la condición de que si la pantalla del dispositivo es inferior a 600px el valor del ancho del formulario de respuesta de los jugadores sea de 200px. Esto nos permite ofrecer un diseño responsable en función del dispositivo que accede a la web. La finalidad es conseguir un buen diseño para el uso en terminales móviles.

2.5.3 JavaScript, JS

JavaScript o JS es un lenguaje de programación interpretado, utilizado principalmente en el lado del cliente permitiendo una mejora de la interfaz de usuario, dando un mayor dinamismo a la web. Con JavaScript se puede realizar un control de archivos multimedia, la creación de imágenes animadas y otros efectos sobre la actualización de la interfaz del usuario gracias al manejo del DOM de la página. Es ejecutado por el motor JavaScript del navegador una vez que el código HTML y CSS ha sido cargado y congregado dentro de la página. Fue lanzado en 1995.

Ilustración 53: JavaScript logo.

Su uso dentro del proyecto ha sido aplicado para la modificación del DOM de la página ante interacciones del usuario y ante los eventos procedentes de la comunicación Websocket establecida con el servidor (modificación en los grupos y equipos, gestión del jugador, barra de tiempo restante, entre otros).

Para establecer un punto final de comunicación o cliente Websocket es necesaria la creación de un nuevo objeto Websocket siguiendo la especificación de la API Websocket, puede verse en la siguiente ilustración *(Fig. 54):*

```
/**
 * URI de conexion con el servidor, cada vez que la pagina se recargue
 * se creará una sesion TCP bi-direccional entre el jugador y el servidor,
 * si la página se recarga la sesión se rompe y se inicia una nueva sesion
 * por lo que el jugador no puede recargar pagina durante el juego o será
 * expulsado de la partida.
 * */
var wsUri = "ws://localhost:8081/servidor-websocket/websocket";
var websocket = new WebSocket(wsUri);
```

Ilustración 54: Definición objeto Websocket en Javascript.

Y además definir los eventos de comunicación para el procesamiento de los mensajes recibidos, puede verse en la siguiente ilustración *(Fig. 55):*

```
/**
 * Evento: se carga la pagina y se crea el WS
 */
websocket.onopen = function(evt) {
    alert("Conectado");
};

/**
 * Evento: recibimos un mensaje a nuestra sesion.
 */
websocket.onmessage = function(evt) {
    procesarEvento(evt);
};

/**
 * Evento: ha ocurrido un error de conexion.
 */
websocket.onerror = function(evt) {
    alert("Ha habido errores");
};
```

Ilustración 55: Definición eventos Websocket en Javascript.

2.6 Tecnología servicio REST

2.6.1 Servicio RESTful con Spring framework y Spring-Boot

Spring [20] es uno de los frameworks más populares de la actualidad, facilita la creación y reutilización de código a los desarrolladores ofreciendo un soporte de infraestructura a nivel de aplicación. A su vez, Spring ofrece soporte para la integración de otros framework como pueden ser Hibernte, EJB, JSF, entre otros.

Spring-Boot es un módulo de Spring que sirve como herramienta para la simplificación de la capa de configuración que el núcleo de Spring supone, permitiendo el enfoque en el desarrollo de las aplicaciones. Las características que Spring-Boot ofrece son:

- Configuración, Spring-Boot cuenta con un módulo de autoconfiguración, el cual permite abstraerse de los aspectos configurables de nuestra aplicación y ejecutar la aplicación con unos simples pasos.
- Resolución de dependencias, con solo determinar el tipo de proyecto, Spring-Boot se encarga de resolver las dependencias de forma automática.
- Despliegue de aplicaciones, dispone de un servidor web integrado que permite el despliegue

de aplicaciones sin la necesidad de usar un servidor en concreto.

Para la creación de nuestro servicio *RESTful* [19], las peticiones HTTP deben ser manejadas por una clase que actúe a modo de controlador, para definir una clase controladora se utiliza la anotación ***@RestController***, esta anotación le indicará a Spring que esta clase debe ser la encargada de controlar las peticiones HTTP que se reciban.

```
 * Clase encargada de publicar las URL para obtener los jugadores.
 *
 * @author Alberto Jiménez Vázquez
 * @version 1.0
 *
 * */

@RestController
public class JugadorController {

    JugadorDAOImpl JugadorDAO = new JugadorDAOImpl();

    @RequestMapping(value = "/getJugadores", method = { RequestMethod.GET })
    public List<Jugador> getJugadores() {
        return JugadorDAO.getJugadorList();
    }

    @RequestMapping(value = "/getJugadoresPorPartida", method = { RequestMethod.GET })
    public List<Jugador> getJugadoresPorPartida(@RequestParam(value = "partida") int partida) {
        return JugadorDAO.getJugadoresByPartida(partida);
    }

    @RequestMapping(value = "/postJugador", method = { RequestMethod.POST })
    public Jugador createJugador(@RequestBody Jugador jug) {
        JugadorDAO.createJugador(jug);
        return jug;
    }

    @RequestMapping(value = "/putJugador", method = { RequestMethod.PUT })
    public Jugador updateJugador(@RequestBody Jugador jug) {
        JugadorDAO.updateJugador(jug);
        return jug;
    }

    @RequestMapping(value = "/delJugador/{id}", method = { RequestMethod.DELETE })
    public ResponseEntity<String> deleteJugador(@PathVariable("id") int id) {
        JugadorDAO.deleteJugador(id);
        return new ResponseEntity<String>(HttpStatus.OK);
    }
}
```

Ilustración 56: RestController de la clase Jugador.

Para definir acciones ante las peticiones HTTP, necesitamos mapear dichas peticiones a los métodos que se encarguen de procesarlas, esto se consigue con la anotación ***@RequestMapping***, esta anotación mapea todas las operaciones posibles (*GET, PUT, POST* y *DELETE*) al método en cuestión, por lo que si queremos diferenciar debemos indicarle a la anotación el tipo de operación que queremos procesar. La definición del recurso de la URL también debemos indicarla. Si queremos definir un mapeo de los parámetros pasado a la URL con los parámetros del método Java utilizamos las anotaciones ***@RequestParam*** y ***@PathVariable***.

En la comunicación del servicio, los objetos son convertidos en cadenas de texto con formato JSON, esto lo realiza Spring de forma interna mediante el uso de la librería *MappingJackson2HttpMessageConverter.*

Para la ejecución del servicio, es necesario definir la clase *Application (Fig. 57):*

```java
/**
 * Clase necesaria para que Spring-boot arranque el servicio REST.
 *
 * @author Alberto Jiménez Vázquez
 * @version 1.0
 *
 * */

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Ilustración 57: Clase Application para el servicio RESTful.

Con la anotación ***@SpringBootApplication*** autoconfiguramos nuestra aplicación para que Spring despliegue de forma automática.

Una vez definido el servicio web, simplemente basta implementar un cliente REST que consuma el servicio, siguiendo las directrices indicadas en el *Anexo B – Consumición del servicio REST*.

Para la gestión de las dependencias es necesario incluir en nuestro fichero de configuración Pom.xml las siguientes líneas *(Fig. 58):*

```xml
<!-- Ejecución, configuración y arranque con Spring-Boot -->
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.3.RELEASE</version>
</parent>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</dependency>
<!-- Necesario para los Test de prueba del servicio -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.6</version>
</dependency>
```

Ilustración 58: Dependencias Spring-Boot.

2.6.2 Java Persistence API, JPA

Java Persistence API o JPA [21] es una especificación que ofrece la plataforma de desarrollo Java para la implementación de un Framework que permita la interacción con la base de datos por medio de objetos. JPA convierte los objetos Java en instrucciones para el manejador de la base de datos. JPA es una especificación y no un framework, como son Hibernate, ObjectDB y OpenJPA entre otros.

Ilustración 59: JPA logo.

Las ventajas y desventajas [22] que ofrece JPA son las siguientes:

Ventajas JPA:

- Desarrollo más rápido y eficiente.
- Paradigma completamente orientado a objetos.
- Reducción de errores en tiempo de ejecución.
- Mejora del mantenimiento del software.

Desventajas JPA:

- Sin soporte para la realización de consultas nativas.
- Una mayor curva de aprendizaje.

Para la integración de JPA en nuestra aplicación, lo primero de todo es establecer la configuración y conexión con la base de datos, para ello necesitamos definir el fichero de configuración JPA persistence.xml el cual debemos alojar en nuestra carpeta de proyecto */META-INF*, la definición del fichero se recoge en la siguiente ilustración *(Fig. 60)*:

```
<?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">

    <!-- Unidad de persistencia -->
    <persistence-unit name="JPA_PU" transaction-type="RESOURCE_LOCAL">

        <!-- Proveedor de persistencia -->
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

        <!-- Clases de nuestro modelo -->
        <class>modelo.entidades.Jugador</class>
        <class>modelo.entidades.Equipo</class>
        <class>modelo.entidades.Grupo</class>
        <class>modelo.entidades.Partida</class>
        <class>modelo.entidades.EstadisticasEquipo</class>
        <class>modelo.entidades.EstadisticasGrupo</class>
        <class>modelo.entidades.EstadisticasJugador</class>
        <class>modelo.entidades.RespuestaPregunta</class>
        <class>modelo.entidades.RespuestaVotacion</class>

        <!-- No excluimos ninguna clase -->
        <exclude-unlisted-classes>false</exclude-unlisted-classes>

        <!-- Configuracion de conexion a base de datos -->
        <properties>
            <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost/mibbdd" />
            <property name="javax.persistence.jdbc.user" value="ajiva" />
            <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
            <property name="javax.persistence.jdbc.password" value="prueba01" />
            <property name="javax.persistence.schema-generation.database.action"
                value="create" />
        </properties>
    </persistence-unit>
</persistence>
```

Ilustración 60: Fichero de configuración JPA.

En primer lugar, debemos de crear la unidad de persistencia a la que nombraremos ***JPA_PU***, este nombre debe ser único. Como tipo de transacción podemos indicar dos, ***JTA*** para entornos Java EE que corran sobre servidores de aplicaciones como TomEE (no Apache Tomcat) y que gestionarán la clase principal que es el *EntityManager* encargada de gestionar todas las clases que intervienen en la persistencia. La otra opción es ***RESOURCE_LOCAL*** para entornos Java SE y el uso de contenedores de Servlet tipo Tomcat.

En segundo lugar, debemos indicar el proveedor de la implementación de JPA, dado que JPA es una especificación no incluye las librerías necesarias para llevar a cabo el desarrollo. En nuestro caso usaremos la implementación de EclipseLinc, también podríamos usar otras implementaciones como Hibernate. Las dependencias necesarias para nuestro pom.xml son las siguientes:

```
<!-- Necesario para el uso de JPA -->
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>javax.persistence</artifactId>
    <version>2.2.0</version>
</dependency>
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa</artifactId>
    <version>2.7.1</version>
</dependency>
```

Ilustración 61: Dependencias JPA.

El siguiente paso es la definición de las clases que representan las entidades de persistencia para que sean gestionadas por el EntityManager, opcionalmente podemos definir el atributo de exclusión de clases para que automáticamente se detecten todas las clases anotadas, pero dado que no es soportado por todas las implementaciones se recomienda la definición de todas las clases.

Las entidades se definen mediante el uso de anotaciones, véase la siguiente ilustración *(Fig. 62):*

```
/**
 * Entidad Grupo
 *
 * @author Alberto Jiménez Vázquez
 * @version 1.0
 *
 * */

@Entity
@Table(name = "grupo")
public class Grupo {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "descripcion")
    private String descripcion;

    @Column(name = "equipo")
    private int equipo;

    @Column(name = "partida")
    private int partida;

    @Column(name = "numjugadores")
    private int numjugadores;
```

Ilustración 62: Entidades JPA.

La anotación ***@Entity*** se utiliza para definir que la clase es una entidad que necesita ser gestionada por el EntityManager, la anotación ***@Table*** define el mapeo de la entidad con la tabla correspondiente en la base de datos, ***@Id*** se utiliza para definir un atributo como clave primaria, ***@Column*** indica que el atributo es una columna de la tabla y ***@GeneratedValue*** se ha utilizado para la generación de un índice numérico incremental automático a la hora de la inserción de objetos en la base de datos.

Por último, definimos la conexión con nuestra base de datos, indicando los siguientes parámetros:

- Ruta de conexión con la base de datos.
- Usuario y contraseña de conexión.
- Driver de la base de datos, en este caso PostgreSQL.
- Definir el comportamiento del atributo esquema de generación para nuestra unidad de persistencia, en este caso, EclipseLinc es capaz de crear las tablas en función de las entidades anotadas en caso de no existir.

El EntityManager es el componente de JPA que se encarga del ciclo de vida de todas las entidades definidas en la unidad de persistencia y ofrece la interfaz para realizar las operaciones básicas sobre la base de datos (consultar, actualizar, borrar e insertar) llevando el control de todas las transacciones. Todas estas acciones serán implementadas utilizando el patrón de diseño DAO, Data Access Object.

2.6.3 Data Access Object, DAO

Data Access Object [17] [18] es un patrón de diseño software que permite la encapsulación del acceso a la base de datos, separando la capa de la lógica de negocio de la capa de persistencia. Siguiendo estas directrices se ha definido una API dentro del servicio REST que ofrece el uso de métodos *CRUD* (*Create, Read, Update* y *Delete*) sobre la base de datos.

Para ello simplemente debemos definir una interfaz y una implementación que será la que gestione el acceso a base de datos *(Fig. 63, 64):*

```
/**
 * Interfaz para el acceso de objetos Equipo a base de datos.
 *
 * @author Alberto Jiménez Vázquez
 * @version 1.0
 *
 * */

public interface EquipoDAO {

    public List<Equipo> getEquipoList();

    public Equipo getEquipoById(int idEquipo);

    public void createEquipo(Equipo equ);

}
```

Ilustración 63: Interfaz DAO.

Para ello debemos de crear un objeto que implemente la clase EntityManager a partir de la cual se iniciaran las transacciones sobre la base de datos *(Fig. 64).*

De esta interfaz de acceso hará uso el servicio REST a la hora de atender peticiones que implique lectura y escritura en la base de datos.

```java
/**
 * Implementación de la interfaz de acceso a objetos Equipo en base de datos.
 *
 * @author Alberto Jiménez Vázquez
 * @version 1.0
 *
 * */

public class EquipoDAOImpl implements EquipoDAO {

    static EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPA_PU");
    private static EntityManager em = emf.createEntityManager();

    @SuppressWarnings("unchecked")
    public List<Equipo> getEquipoList() {
        return em.createQuery("select e from Equipo e").getResultList();
    }

    public Equipo getEquipoById(int idEquipo) {
        return (Equipo) em.createQuery("select e from Equipo e where e.id =" + idEquipo).getSingleResult();
    }

    public void createEquipo(Equipo equ) {
        em.getTransaction().begin();
        em.persist(equ);
        em.flush();
        em.getTransaction().commit();
    }

}
```

Ilustración 64: Implementación DAO.

Este patrón de diseño ha sido utilizando ya que aporta una gran flexibilidad a la capa de persistencia de nuestra aplicación. Otra alternativa es ActiveRecord, utilizado cuando hay mayor aislamiento entre nuestra aplicación y la base de datos, ya que son los componentes de ActiveRecord los encargados de realizar todas las operaciones.

2.7 Gestión de la base de datos

2.7.1 Modelo relacional en la gestión de bases de datos

El modelo relacional es un modelo de organización y gestión de bases de datos que consiste en el almacenamiento de información en relaciones o tablas, compuestas por tuplas o filas y columnas o campos. Está basado en el almacenamiento de datos en relaciones, que a su vez son conjuntos de datos, es por ello por lo que el orden de almacenamiento no tiene relevancia.

La información puede ser recuperada o almacenada mediante consultas que ofrecen una amplia flexibilidad para la administración de información. Este lenguaje de manipulación de datos se denomina lenguaje relacional. En este proyecto se usará SQL como lenguaje relacional.

La base de datos es el conjunto de datos almacenados como tal, mientras que el sistema gestor de base de datos es el software encargado de la gestión y administración de dicha base de datos.

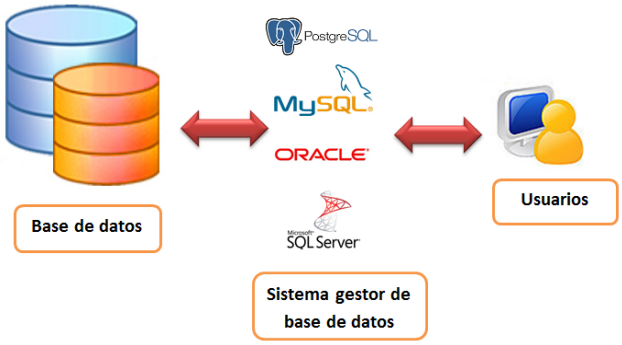


Ilustración 65: Esquema modelo relacional [29].

Para el almacenamiento de datos de interés, en el proyecto usaremos una base de datos relacional gestionada por un sistema gestor de base de datos PostgreSQL.

Dado la necesidad de un almacenamiento consistente de datos en una única máquina en la que las operaciones de lectura sobre la base de datos no son muy elevadas se ha determinado por el uso de un sistema SQL en vez de uno NOSQL.

2.7.2 PostgreSQL como sistema gestor

PostgreSQL [13] es un sistema gestor de base de datos relacionales, de código abierto y publicado en 1995. Es un sistema que destaca por ofrecer una alta concurrencia mediante el sistema *MVCC* (acceso concurrente multiversión), lo cual permite que mientras un proceso escribe en una tabla, otros procesos puedan acceder a esa misma tabla sin reportar bloqueos.

Ilustración 66: PostgreSQL logo.

Para la instalación de PostgreSQL en nuestra máquina Linux *Ubuntu 16.04*, basta con ejecutar los siguientes comandos:

```
~ sudo apt-get update
~ sudo apt-get install postgresql
```

Código 10: Instalación PostgreSQL.

Al instalar el SGBD postgresql, se crea dentro del S.O. Linux el usuario postgres, que además es superusuario del servidor PostgreSQL. Para cambiar la contraseña del superusuario del servidor PostgreSQL (postgres) dentro del SGBD y además crear un nuevo usuario propio (*ajiva*) y una nueva base de datos, es necesario ejecutar las siguientes órdenes:

```
~ passwd postgres
~ su - postgres
~ psql
~ CREATE USER ajiva WITH ENCRYPTED PASSWORD '---' CREATEDB;
~ createdb mibbdd
```

Código 11: Gestión de usuario PostgreSQL.

Por último, sólo hay que cambiar la configuración correspondiente para permitir la autenticación por contraseña de los usuarios del sistema gestor de bases de datos PostgreSQL. Para ello sólo hay que añadir en el fichero de configuración de acceso basado en host de postgresql, ***/etc/postgresql/9.1/main/pg_hba.conf*** la siguiente línea y posteriormente reiniciar el servidor para que se aplique la nueva configuración:

Para cambiar el puerto de escucha del servidor, cuyo valor por defecto es 5432, es necesario acceder al fichero de configuración principal del servidor ***/etc/postgresql/9.2/main/postgresql.conf***, y cambiar el valor de la linea *'port = 5432'* por *'port = 8001'* por ejemplo.

Para habilitar el acceso desde el exterior al servidor PostgreSQL, el primer paso es modificar el fichero **/etc/postgresql/9.2/main/postgresql.conf**, modificando el atributo *listen_addresses ='*'*.

El siguiente paso es modificar el fichero **/etc/postgresql/9.2/main/pg_hba.conf** añadiendo una

```
// Incluir: local all ajiva md5

~ /etc/init.d/postgresql restart o service postgresql restart
```

Código 12: Acceso en host y reinicio PostgreSQL.

nueva línea tras la línea anteriormente añadida, con contenido *'host all ajiva 0.0.0.0/0 md5'*.

Tras realizar estos cambios, es necesario reiniciar el servicio.

Con los cambios introducidos, ahora el usuario ajiva puede acceder desde cualquier máquina a la base de datos creada mibbdd mediante la siguiente sentencia:

```
~  psql -h <IP_servidor> -U ajiva -d mibbdd -p 8001
```

Código 13: Conexión con la base de datos.

Siguiendo estos pasos disponemos de un sistema gestor de base de datos PostgreSQL instalado y una mase de datos creada donde se ejecutarán las sentencias para la creación de relaciones y donde posteriormente serán almacenados los datos de la aplicación.

El uso de PostgreSQL viene determinado por la familiarización con este sistema gestor. En el mercado hay innumerables sistemas para la gestión de base de datos como pueden ser MySQL, Oracle o Microsoft SQL Server.

2.8 Servidor de aplicaciones

2.8.1 Apache Tomcat

Como servidor para el despliegue de la aplicación se ha optado por usar Apache Tomcat [12]. Tomcat no es un servidor de aplicaciones como tal, sino un contenedor de Servlet y soporte JSP. El motor de Servlet de Tomcat a menudo se presenta en combinación con el servidor web Apache.

Ilustración 67: Apache Tomcat logo.

Tomcat puede funcionar como un servidor web por si mismo y dado que fue escrito en Java es compatible con cualquier sistema operativo que disponga de la máquina virtual de Java. Además, su integración con Eclipse o Spring Tools Suite es muy sencilla desde la gestión de servidores del IDE.

Para disponer de una correcta integración de Apache Tomcat con nuestro IDE, necesitamos la siguiente dependencia en nuestro pom.xml:

```
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-catalina</artifactId>
    <version>8.0.32</version>
</dependency>
```

Ilustración 68: Dependencias Apache Tomcat.

El uso de Tomcat en el proyecto viene incentivado por la necesidad de un servidor de aplicaciones para desplegar la aplicación. Tomcat ofrece la funcionalidad necesaria, aunque podrían ser utilizados otros servidores como TomEE que ofrece el soporte para CDI integrado en el servidor.

3 HERRAMIENTAS UTILIZADAS

"Si tu única herramienta es un martillo, tiendes a tratar cada problema como si fuera un clavo".

- Abraham Maslow -

La finalidad de este apartado es describir las herramientas sotfware que han sido utilizadas durante la realización del proyecto. Herramientas como el entorno de desarrollo, administración y modelado de la base de datos, realización del diseño de diagramas, testing de las URL definidas en el servicio REST o el navegador web utilizado son fundamentales para la correcta ejecución de las tareas llevadas a cabo.

En la siguiente tabla se recoge un resumen de las herramientas utilizadas en el proyecto (*Tabla 3*):

Herramienta necesaria	Justificación de uso	Aplicación de la herramienta
SPRING TOOL SUITE	Necesidad de una herramienta que sirva de entorno de desarrollo de la aplicación.	Entorno de desarrollo.
SOAPUI	Necesidad de una herramienta para la realización de tests sobre el servicio REST.	Test de petición y respuesta de la API REST.
PGMODELER	Necesidad de una herramienta para el diseño de la base de datos.	Diseño de la base de datos.
PGADMIN	Necesidad de una herramienta que permita la administración de la base de datos.	Administración de la base de datos.
DRAWIO	Necesidad de una herramienta para la elaboración de diagramas y esquemas.	Elaboración de diagramas y esquemas.
FIREFOX	Necesidad de un navegador web con el que realizar las pruebas de la aplicación.	Realización de pruebas.

Tabla 3: Herramientas utilizadas.

3.1 Spring Tool Suite, STS

Spring Tool Suite o STS es un entorno de desarrollo software basado en las últimas versiones de Eclipse y personalizado para el desarrollo de aplicaciones Spring. STS proporciona un entorno para implementar, depurar, ejecutar y desplegar aplicaciones Spring. Incluye también integraciones para Pivotal tc Server, una versión de Apache Tomcat, así como Git, Maven y otras herramientas.

Analizando la herramienta, STS prácticamente es Eclipse con ciertas configuraciones diferentes, la barra de herramientas, el explorador de proyectos, la consola y la importación de servidor es exactamente igual que los de Eclipse. Algunas de estas configuraciones integradas en STS que ofrecen una mejora respecto a Eclipse son:

- Dispone de la distribución JavaEE completa de Eclipse integrada.
- Incluye una integración con Maven, Spring Roo y Pivotal tc Server.
- Incluye una guía de inicio muy completa sobre el IDE y Spring Framework.

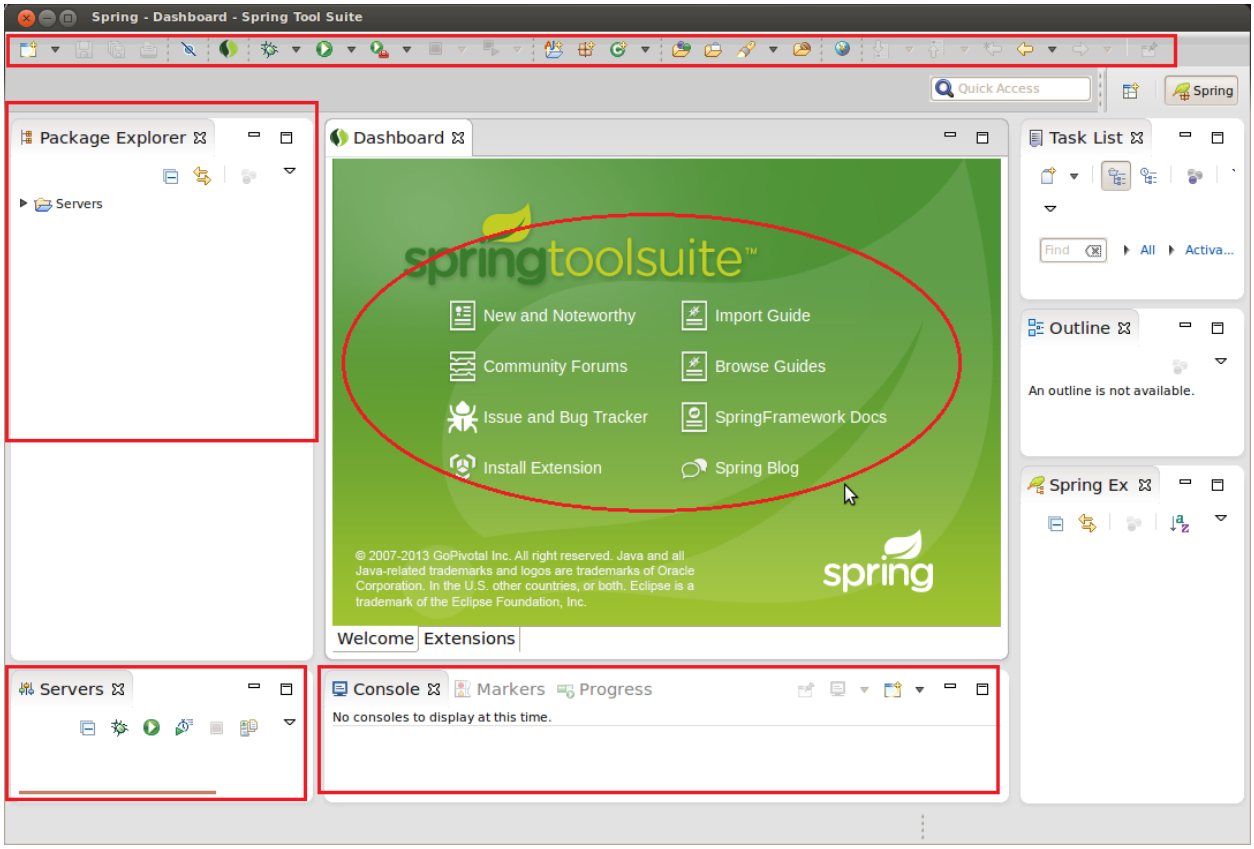


Ilustración 69: Spring Tool Suite.

Spring Tool Suite es un entorno de desarrollo de libre acceso para el desarrollador, de código abierto y bajo licencia pública Eclipse, por lo que se convierte en una muy buena herramienta para desarrollar nuestras aplicaciones.

La personalización de STS para su uso en aplicaciones con Spring Framework lo hacen ideal como entorno de desarrollo, aunque son totalmente lícitos otras alternativas como las últimas versiones de Eclipse, IntelliJ o Netbeans.

3.2 SoapUI

SoapUI [14] es una herramienta para la realización de pruebas a aplicaciones con arquitectura orientada a servicios (SOA) y transferencia de estado representacional (REST). Soporta múltiples protocolos: SOAP, REST, HTTP y JDBC entre otros.

Ilustración 70: SoapUI logo.

SoapUI posee una interfaz gráfica bastante intuitiva y sencilla de usar. Una barra de herramientas donde seleccionar los proyectos a crear y su importación, un explorar de proyectos similar al que ofrece Eclipse y un manual de iniciación bastante útil.

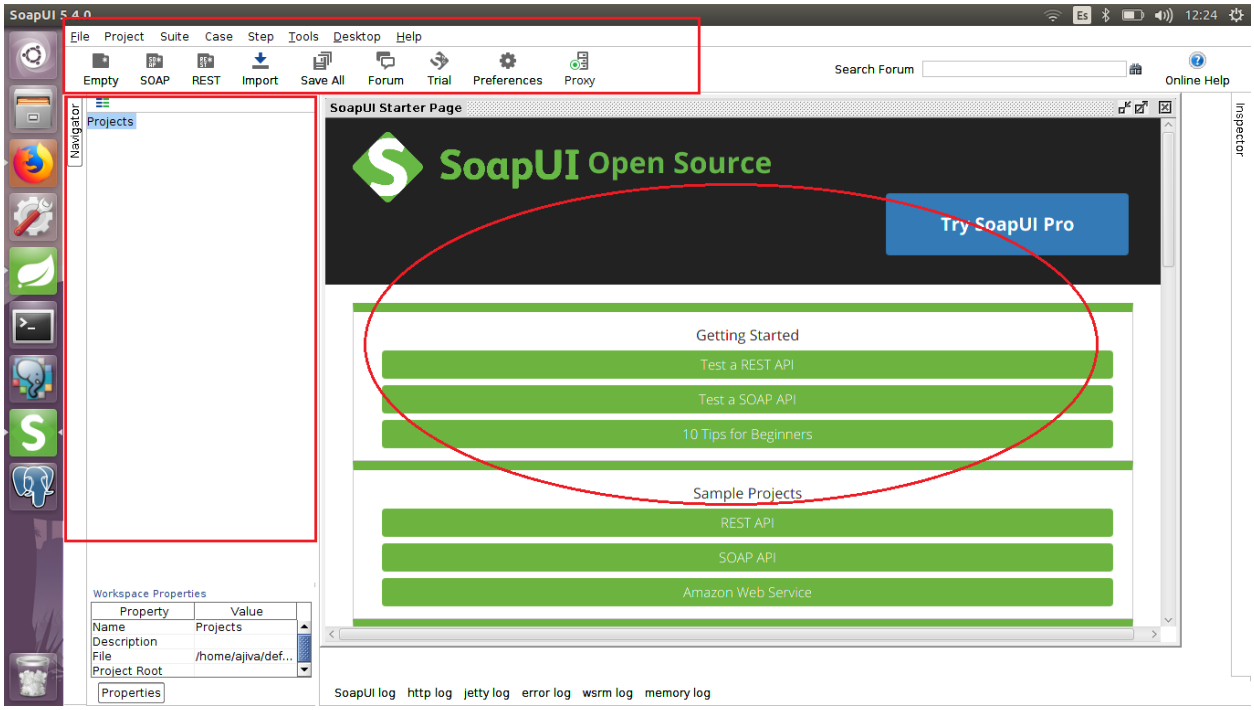


Ilustración 71: Presentación SoapUI.

Para crear un proyecto SoapUI y probar nuestro servicio, simplemente pulsamos sobre la opción Menú -> REST y se nos abrirá una pestaña donde indiquemos la URI del servicio:

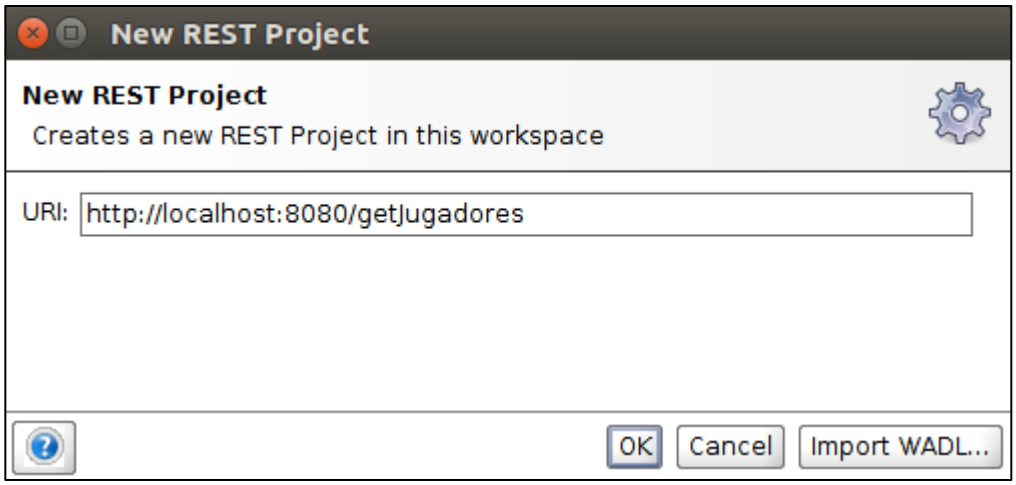


Ilustración 72: URI del servicio SoapUI.

Una vez iniciado, podemos ver a la izquierda la información del proyecto y modificar la petición en la barra de peticiones, donde se puede definir el punto final, el recurso y los parámetros de la petición, así como el método de la petición que queremos realizar. Una vez definida la petición, simplemente tenemos que lanzarla y en la consola podremos ver: información de la petición y de la respuesta, los resultados obtenidos por la petición. En este caso estamos solicitando obtener la información de todos los jugadores almacenados en la base de datos, la cual nos será brindada en formato de texto JSON.

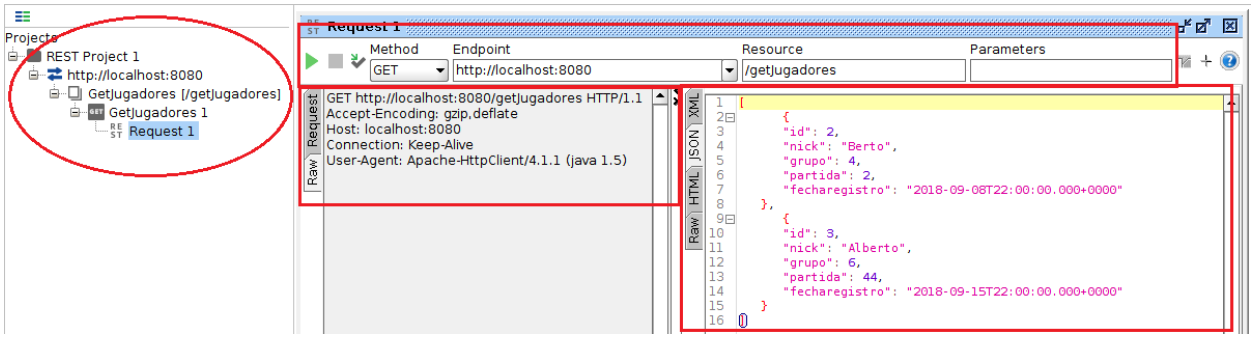


Ilustración 73: Creación de la petición SoapUI.

Si realizamos esa misma consulta traducida a SQL en el terminal de nuestra base de datos, obtenemos la misma información:

```
mibbdd=> select * from jugador;
 id |  nick   | grupo | fecharegistro | partida
----+---------+-------+---------------+---------
  2 | Berto   |     4 | 2018-09-09    |       2
  3 | Alberto |     6 | 2018-09-16    |      44
(2 rows)
```

Ilustración 74: Comprobación información obtenida SoapUI.

La elección de SoapUI viene incentiva por la familiarización con el software y su facilidad e intuición de uso. Otras herramientas que ofrecen características similares para las pruebas sobre los servicios REST pueden ser Postman, Insomnia REST Client o HttpMaster.

3.3 PgModeler

PgModeler [15] es una herramienta para el modelado y administración de bases de datos PostgreSQL, las funcionalidades que ofrece son:

- Creación y edición de forma rápida y sencilla de modelos de datos.
- Exportación de las estructuras de datos una vez realizadas.
- Ofrece un módulo de administración de base de datos.
- Generación de Scripts SQL.
- Open source y multiplataforma.

Ilustración 75: PgModeler logo.

La instalación de PgModeler es muy sencilla, basta con ejecutar los siguientes comandos:

```
~ sudo apt-get update
~ sudo apt-get install pgmodeler
```

Código 14: Instalación PgModeler.

El diseño de modelos con esta herramienta es muy intuitivo, simplemente tenemos que hacer click derecho sobre el panel de inicio de la herramienta y crear nueva tabla. Dentro del menú de creación de la tabla podemos definir sus características principales (nombre, esquema al que pertenece, dueño de la tabla) y definir la estructura de la tabla (columnas, restricciones, índices, etc.):

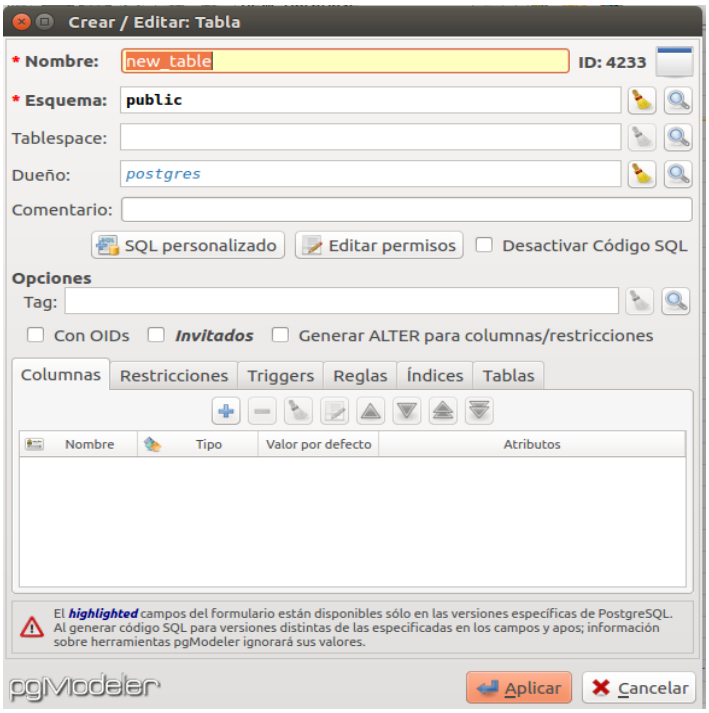


Ilustración 76: Creación tabla PgModeler.

Para la creación de las columnas simplemente pulsamos sobre el botón de añadir columna y definimos el nuevo campo de la tabla. Podemos indicar el tipo y formato de la celda e incluir valores por defecto y secuencias:

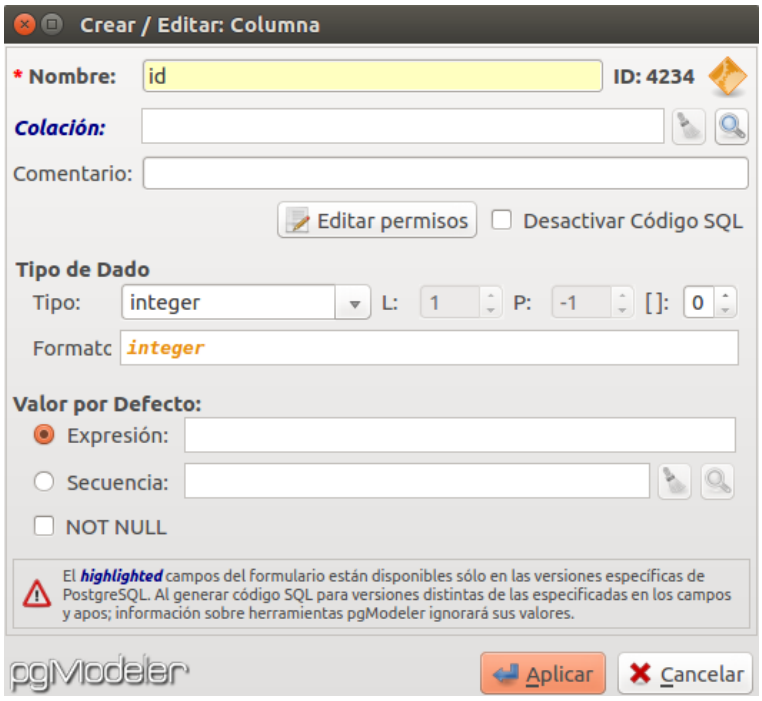


Ilustración 77: Creación columna PgModeler.

En la siguiente ilustración *(Fig.78)* podemos ver cómo quedaría la definición de las columnas de la tabla *respuesta_votacion* de nuestro modelo:

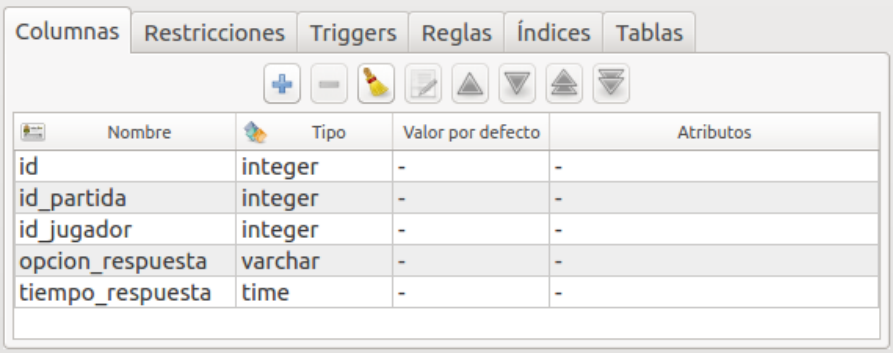


Ilustración 78: Definición columnas PgModeler.

Una vez que tenemos definidas las columnas de nuestra relación pasamos a definir las restricciones que deben cumplir dichas columnas, para ello relacionamos mediante la tipología de la restricción las columnas de la tabla con columnas de otras tablas en caso de ser clave externa o de la misma tabla en caso de ser claves primarias, también es necesario definir la regla a aplicar en caso de borrado o actualización:

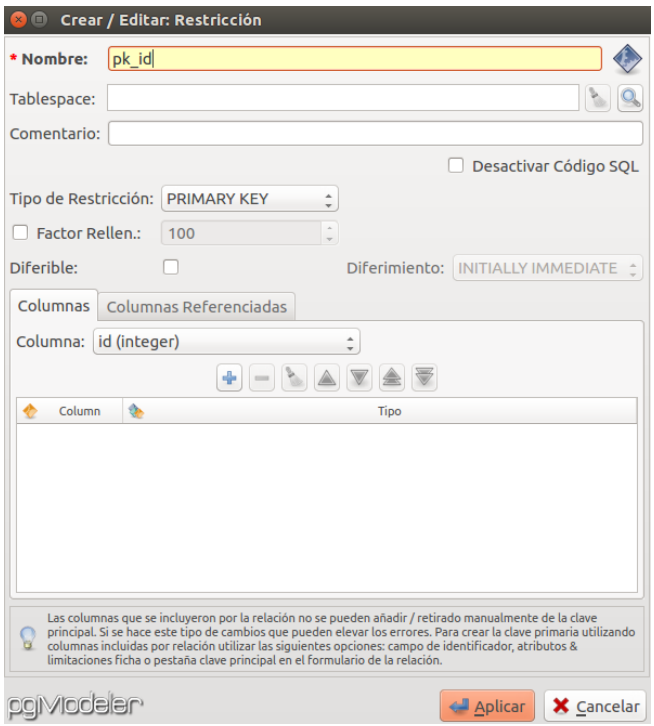


Ilustración 79: Creación restricciones PgModeler.

Como resultado podemos ver las restricciones creadas en el panel de restricciones del menú de creación de la tabla:

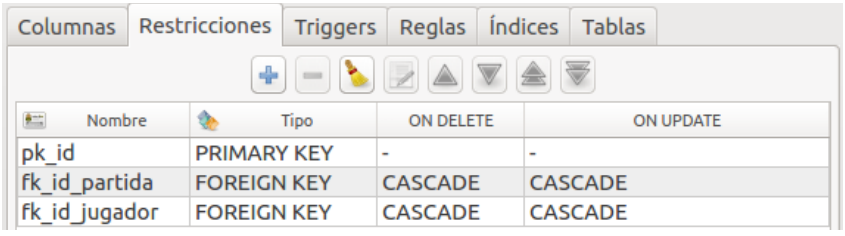


Ilustración 80: Definición restricciones PgModeler.

Finalmente, una vez definida nuestra tabla al completo *(Fig. 81)* podríamos usarla para la creación de un modelo completo y la agregación de relaciones:

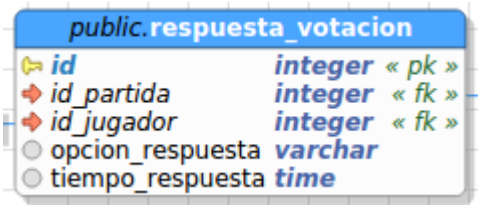


Ilustración 81: Tabla PgModeler.

La elección de pgModeler se ha realizado por su uso intuitivo y sencillo, no obstante, existen otras alternativas en el mercado como pueden ser Vertabelo, Lucidchart o SQL Server modeler.

3.4 PgAdmin

PgAdmin [16] es una herramienta para la administración de bases de datos PostgreSQL, es de código abierto y se encuentra disponible para mayoría de sistemas operativos. Entre sus características destacan:

- Interfaz de administración gráfica.
- Realización de consultas SQL.
- Editor de código procedural.

En la pantalla de inicio disponemos un panel de herramientas desde donde nos permite agregar conexiones a nuestras bases de datos. Una vez añadida en el panel izquierdo de exploración podemos gestionar todo lo que corresponde con la base de datos (extensiones, esquemas, tablas, usuarios, etc.).

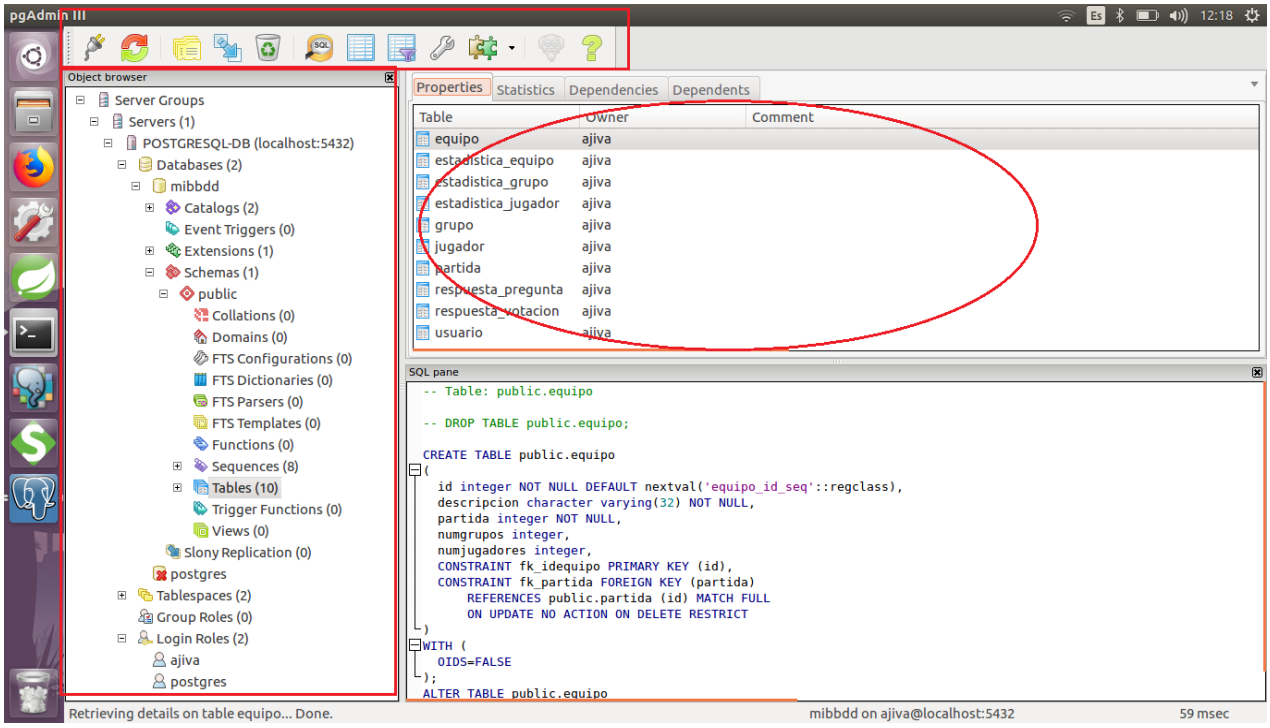


Ilustración 82: Ventana principal PgAdmin.

Una herramienta muy interesante de la que dispone PgAdmin es una recolección estadística del uso de nuestras tablas:

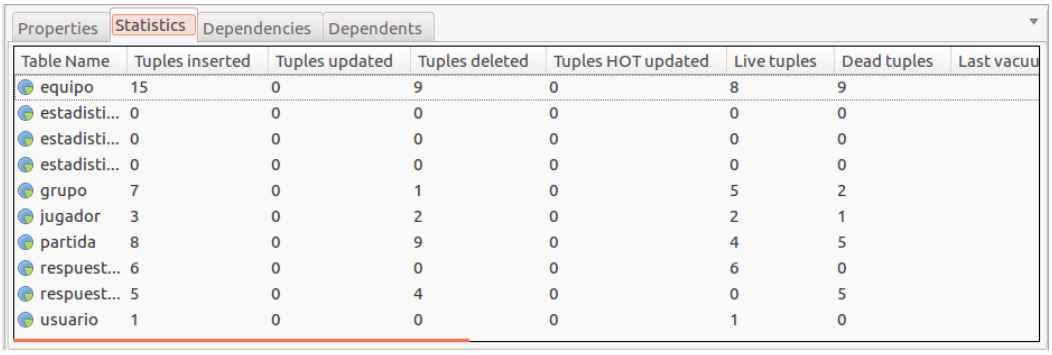
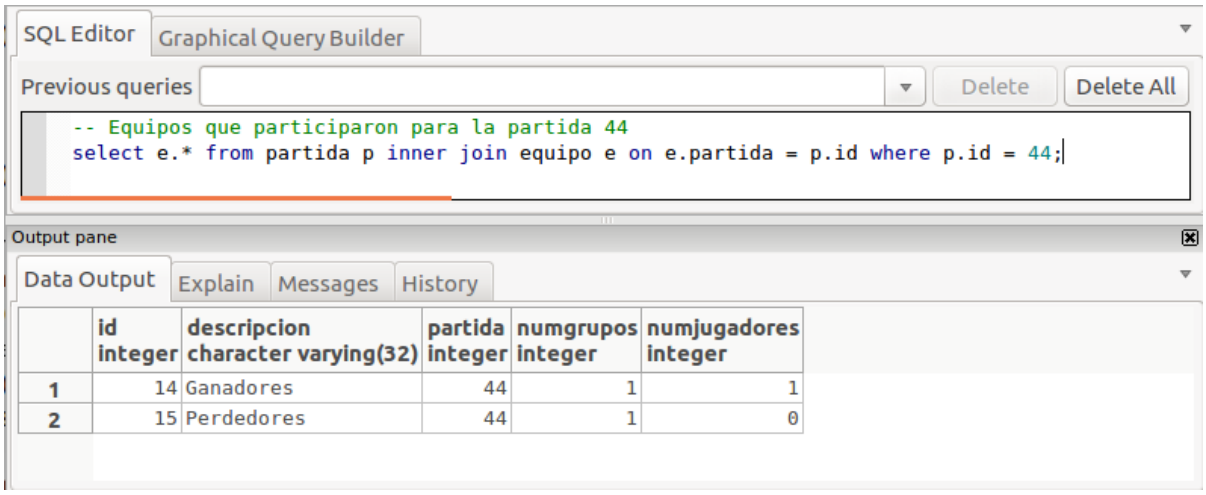


Table Name	Tuples inserted	Tuples updated	Tuples deleted	Tuples HOT updated	Live tuples	Dead tuples	Last vacuu
equipo	15	0	9	0	8	9	
estadisti...	0	0	0	0	0	0	
estadisti...	0	0	0	0	0	0	
estadisti...	0	0	0	0	0	0	
grupo	7	0	1	0	5	2	
jugador	3	0	2	0	2	1	
partida	8	0	9	0	4	5	
respuest...	6	0	0	0	6	0	
respuest...	5	0	4	0	0	5	
usuario	1	0	0	0	1	0	

Ilustración 83: Estadísticas PgAdmin.

Una de sus mayores utilidades es el editor SQL para la realización de consultas, en la siguiente ilustración (Fig. 84) podemos ver una consulta realizada de los equipos que participaron en la partida con identificador 44:



	id integer	descripcion character varying(32)	partida integer	numgrupos integer	numjugadores integer
1	14	Ganadores	44	1	1
2	15	Perdedores	44	1	0

Ilustración 84: SQL Editor PgAdmin.

También dispone de un histórico de consultas realizadas:

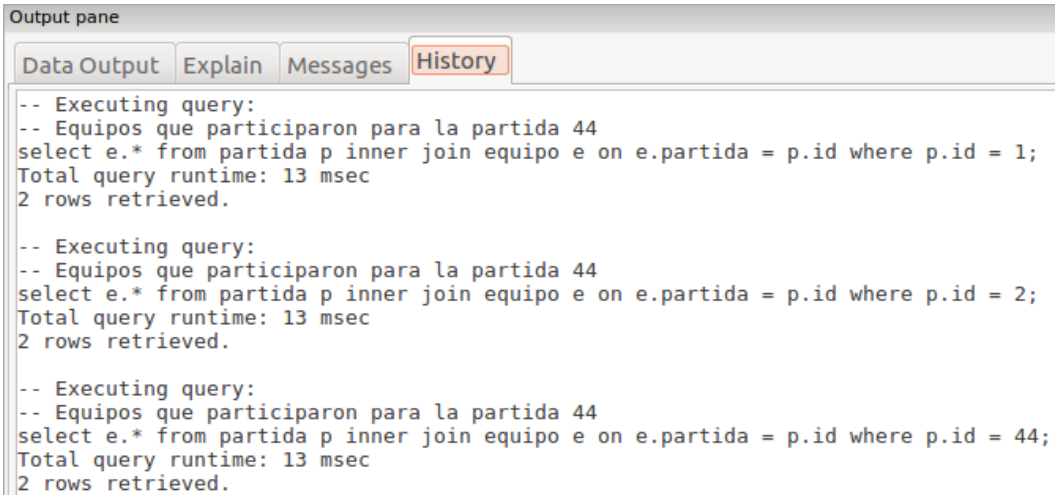


Ilustración 85: Histórico de consultas PgAdmin.

La instalación de PgAdmin es muy sencilla, basta con ejecutar los siguientes comandos:

```
~ sudo apt-get update
~ sudo apt-get install pgadmin3
```

Código 15: Instalación PgAdmin.

PgAdmin ofrece numerosas funcionalidades para la administración de nuestra base de datos PostgreSQL, así como una interfaz de usuario amigable y simple de usar. Otras posibles herramientas son SEQUEL, Navicat o la gestión mediante línea de comandos de PostgreSQL en Linux para los más tradicionales.

3.5 Diagramas con DrawIO

DrawIO [31] es una aplicación web que nos permite la creación de diagramas de múltiples tipos, diagramas UML, diagramas de red, diagramas de infraestructuras e incluso diseñar tus propios diagramas.

Ilustración 86: DrawIO logo.

Es totalmente gratuita y libre de publicidad y licencia, permite exportar tus diagramas de forma cómoda y sencilla y permite la exportación de tus proyectos en formato imagen.

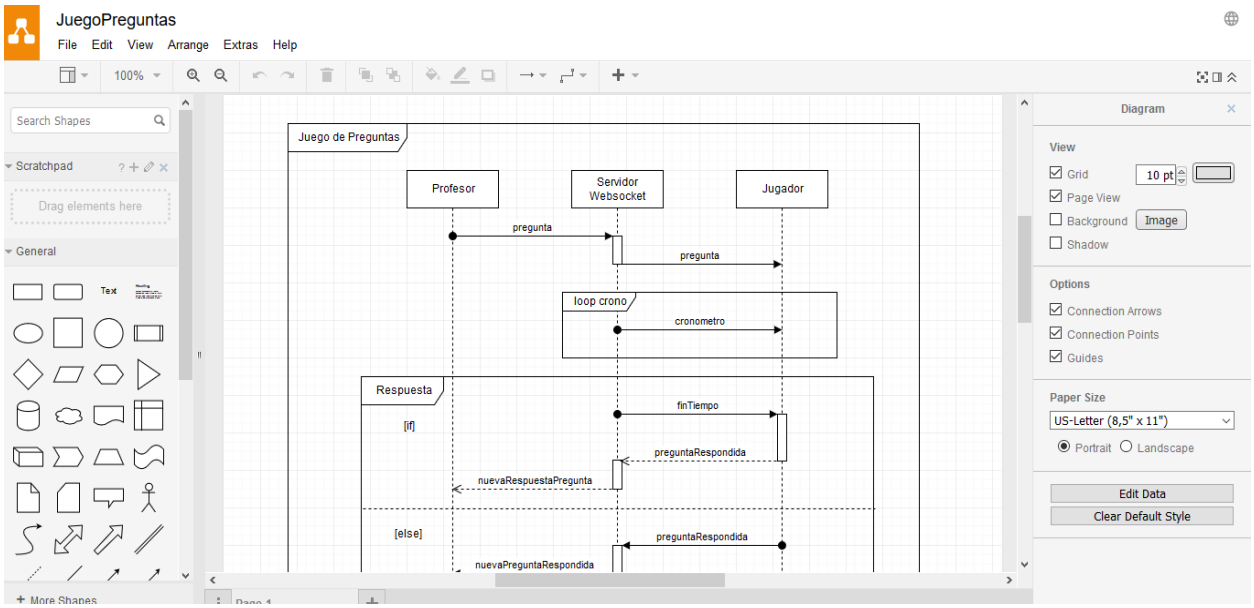


Ilustración 87: Herramienta DrawIO.

La herramienta DrawIO se ha utilizado en este proyecto para el diseño de los diagramas de secuencia en el flujo de mensajes entre los módulos cliente-servidor de la aplicación. DrawIO es una de las mejores herramientas para la elaboración de diagramas, ya que ofrece numerosas funcionalidades, una interfaz onlinea sin tener que realizar descargas y libre de publicidad. Otra alternativa puede ser Lucidchart.

3.6 Mozilla Firefox

Mozilla Firefox es un navegador web libre y de código abierto desarrollado por la Fundación Mozilla. Ha sido el navegador web utilizado durante la realización de pruebas para el proyecto, su elección ha sido simplemente porque es uno de los navegadores que ofrece soporte para Websocket en su versión más reciente.

Ilustración 88: Mozilla logo.

En la siguiente tabla se ofrece una comparativa de las versiones del protocolo Websocket y los navegadores web que las soportan:

Versión WS	*Chrome*	*Firefox*	*Explorer*	*Opera*	*Safari*
Versión Inicial	6	4.0	Sin Soporte	11.0	5.01
7	14	6.0	Sin Soporte	Sin Soporte	Sin Soporte
10	14	7.0	Sin Soporte	Sin Soporte	Sin Soporte
17	16	11.0	10	12.10	Sin Soporte

Tabla 4: Navegadores que soportan Websocket.

4 ANÁLISIS DEL DISEÑO

"Si no lo puedes explicar con simplicidad, es que no lo entiendes bien".

- Albert Einstein -

En esta sección se especificará el diseño de la comunicación entre los módulos de la aplicación. El flujo de mensajes que tienen lugar entre la vista del profesor, la vista del alumno y el servidor de websockets que será el centro de la comunicación ya que por él debe transcurrir todo el intercambio de mensajes. También se detallará el esquema de modelado de la base de dato, definiendo todas las tablas que intervienen para el almacenamiento de datos y las relaciones entre sí.

4.1 Diagrama de componentes

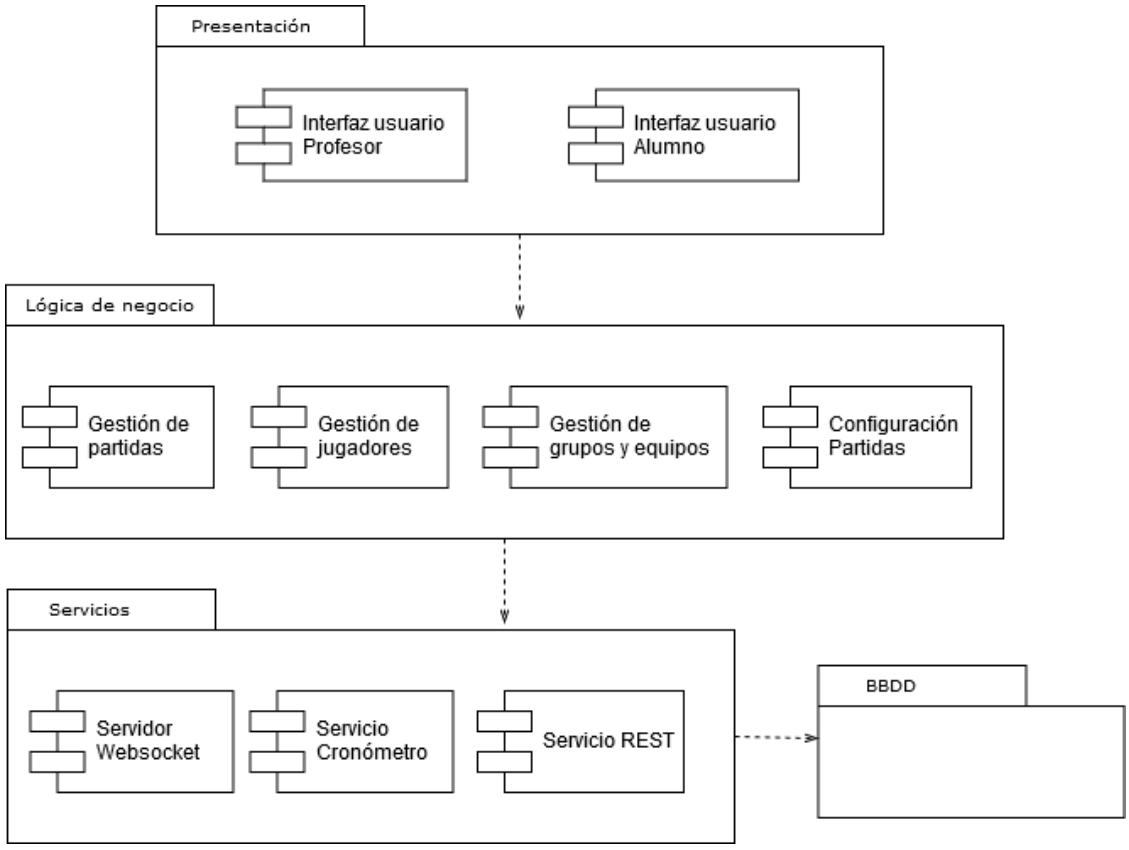


Ilustración 89: Diagrama de componentes.

Componentes pertenecientes a la presentación de la interfaz de usuario:

- Interfaz de usuario profesor: encargada de mostrar los datos referentes al registro de jugadores y evolución del juego al profesor.
- Interfaz de usuario alumno: encargada de mostrar los datos referentes a los formularios de registros y de respuesta a los jugadores.

Componentes pertenecientes a la lógica de negocio de la aplicación:

- Gestión de partidas: gestiona el ciclo de vida de las partidas.
- Gestión de jugadores: gestiona las peticiones de participación por parte de los jugadores.
- Gestión de grupos y equipos: gestiona la estructura de grupos y equipos y la asignación de jugadores a los mismo.
- Configuración de partidas: almacena la información de configuración de las partidas, previamente solicitada al servicio de administración.

Componentes pertenecientes a los servicios de la aplicación:

- Servidor websocket: componente principal del proyecto, se encarga de la redistribución de mensajes entre las distintas partes (juego, profesor y jugador) así como de la gestión del resto de componentes.
- Servicio cronómetro: encargado de establecer un proceso en segundo plano que defina el tiempo restante de cada pregunta.
- Servicio REST: API para la comunicación con el resto de los servicios, en este caso, para solicitar información al servicio de administración y para ofrecer la información almacenada en la base de datos.

Finalmente, los módulos se encuentran respaldados por una base de datos donde almacenar la información referente al contexto de la partida.

4.2 Diagramas de casos de uso

En esta sección se definen los diagramas de caso de uso de la aplicación.

El siguiente diagrama (*CU-001*) define la funcionalidad disponible para el profesor de iniciar partida, es la funcionalidad que enlaza la web de administración con la de comunicación, ya que desde el botón de iniciar partida (administración) se da comienzo al inicio del juego (comunicación).

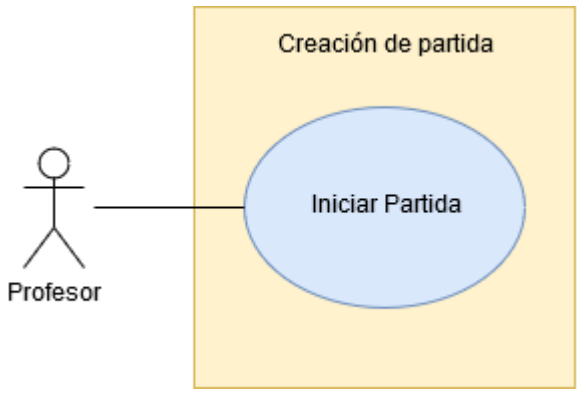


Ilustración 90: Diagrama de casos de uso CU-001.

El siguiente diagrama (*CU-002*) define las dos funcionalidades iniciales de la que disponen de los jugadores antes de iniciar le juego. Primeramente, deben solicitar el acceso a la partida mediante la introducción del identificador de partida y posteriormente, una vez seleccionado grupos y equipos solicitar el registro.

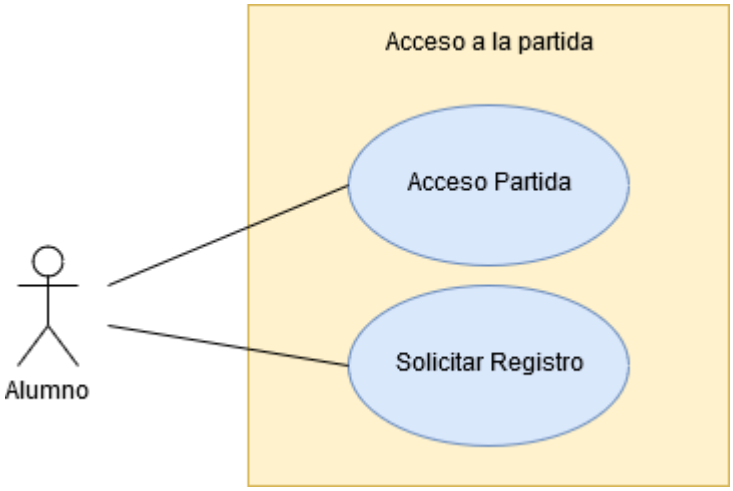


Ilustración 91: Diagrama de casos de uso CU-002.

El siguiente diagrama (*CU-003*) las cuatro funcionalidades de las que dispone el profesor a la hora de gestionar los jugadores que van solicitando participar en la partida. Estas funcionalidades son de eliminación de jugadores y redistribución de jugadores en otros grupos o equipos.

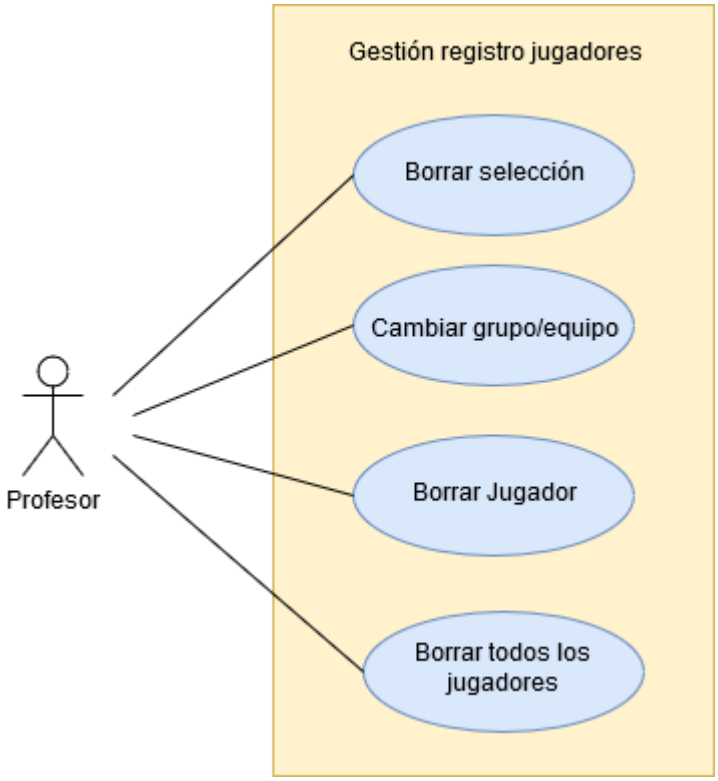


Ilustración 92: Diagrama de casos de uso CU-003.

El siguiente diagrama (*CU-004*) define la única funcionalidad de la que disponen los jugadores durante el juego, que es la de responder las preguntas y votaciones que vaya lanzando el juego.

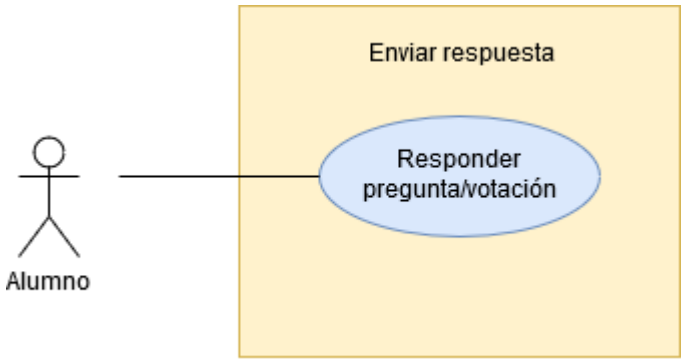


Ilustración 93: Diagrama de casos de uso CU-004.

El siguiente diagrama (*CU-005*) se definen las cuatro funcionalidades de las que dispone el profesor para gestionar la partida en curso. Para dar comienzo a la partida, es necesario avisar al servidor que el proceso de registro ha terminado y enviar al juego la información con los jugadores para que comiencen las preguntas, para ello el profesor debe comenzar la partida. Durante el juego, el profesor puede decidir en qué momento detener y activar el cronómetro que marca el tiempo restante de respuesta. Finalmente, el profesor puede indicar que quiere jugar otra partida, lo que volvería a la página de administración del juego.

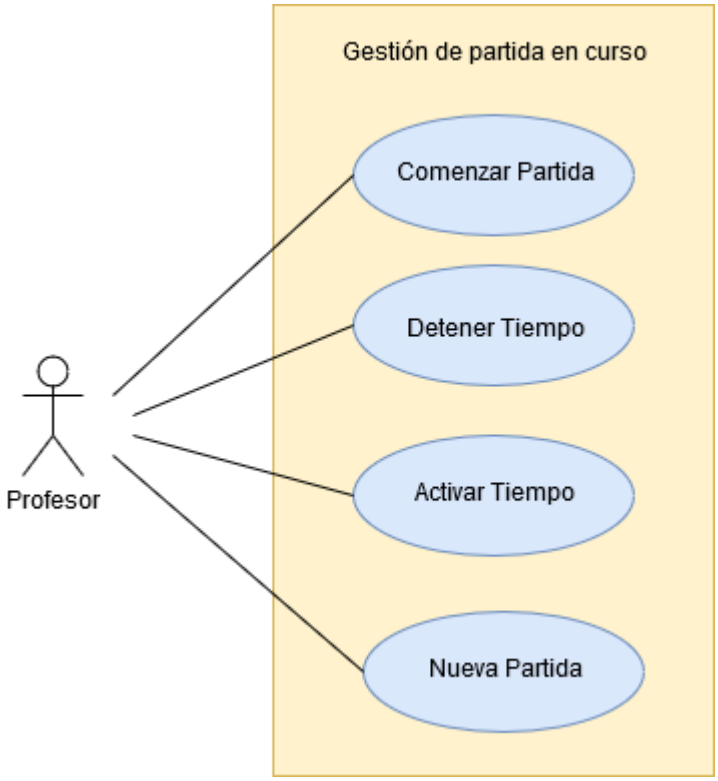


Ilustración 94: Diagrama de casos de uso CU-005.

4.3 Diagramas de flujo

En esta sección se definen los diagramas de flujo de la aplicación, donde se recogen los principales procesos internos que tienen lugar en el código de la aplicación.

En el siguiente diagrama (*DF-001*) se recogen los procesos internos en el momento en que un profesor decide iniciar una partida, la cual comienza con el registro de jugadores.

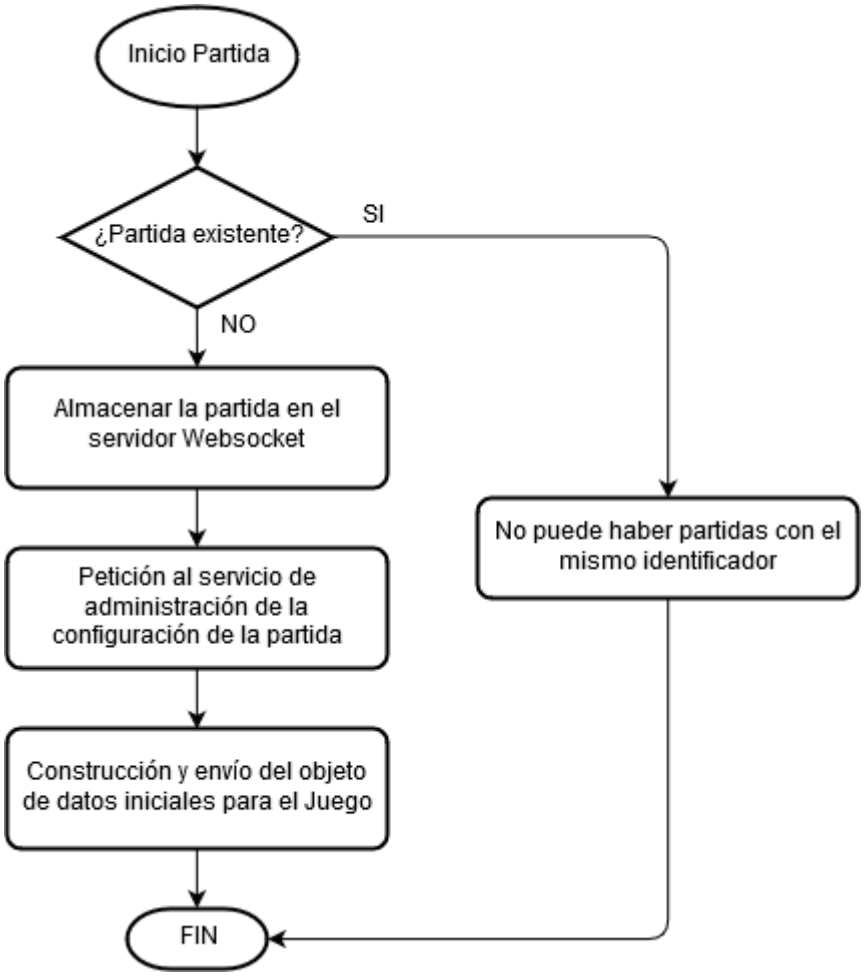


Ilustración 95: Diagrama de flujo DF-001.

En el siguiente diagrama (*DF-002*) se recogen los procesos internos en el momento que un jugador introduce el identificador de partida y solicita el acceso a la misma.

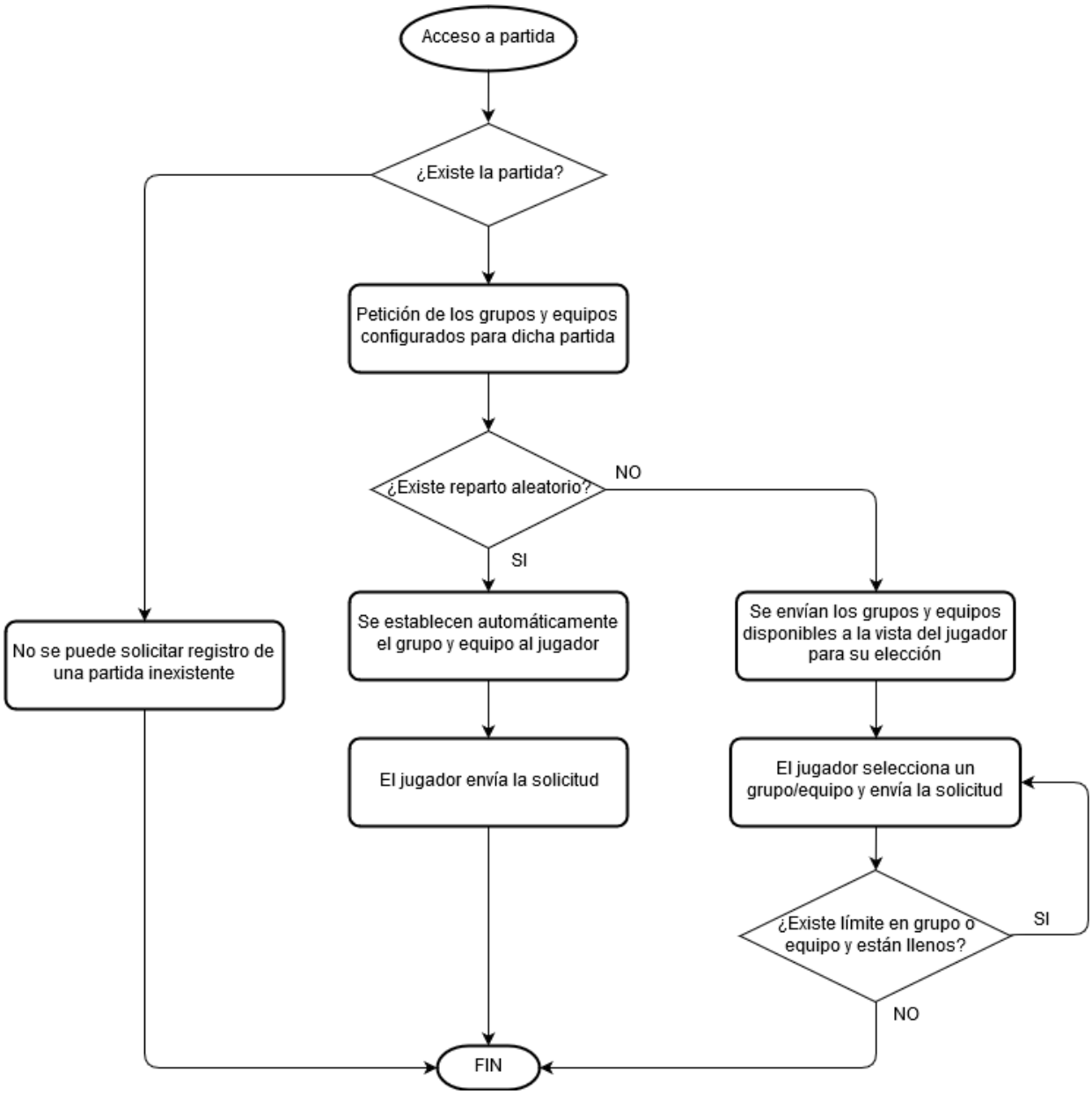


Ilustración 96: Diagrama de flujo DF-002.

En el siguiente diagrama (*DF-003*) se recogen los procesos internos que tienen lugar en el momento que un jugador rellena el formulario de acceso y solicita participar en la partida.

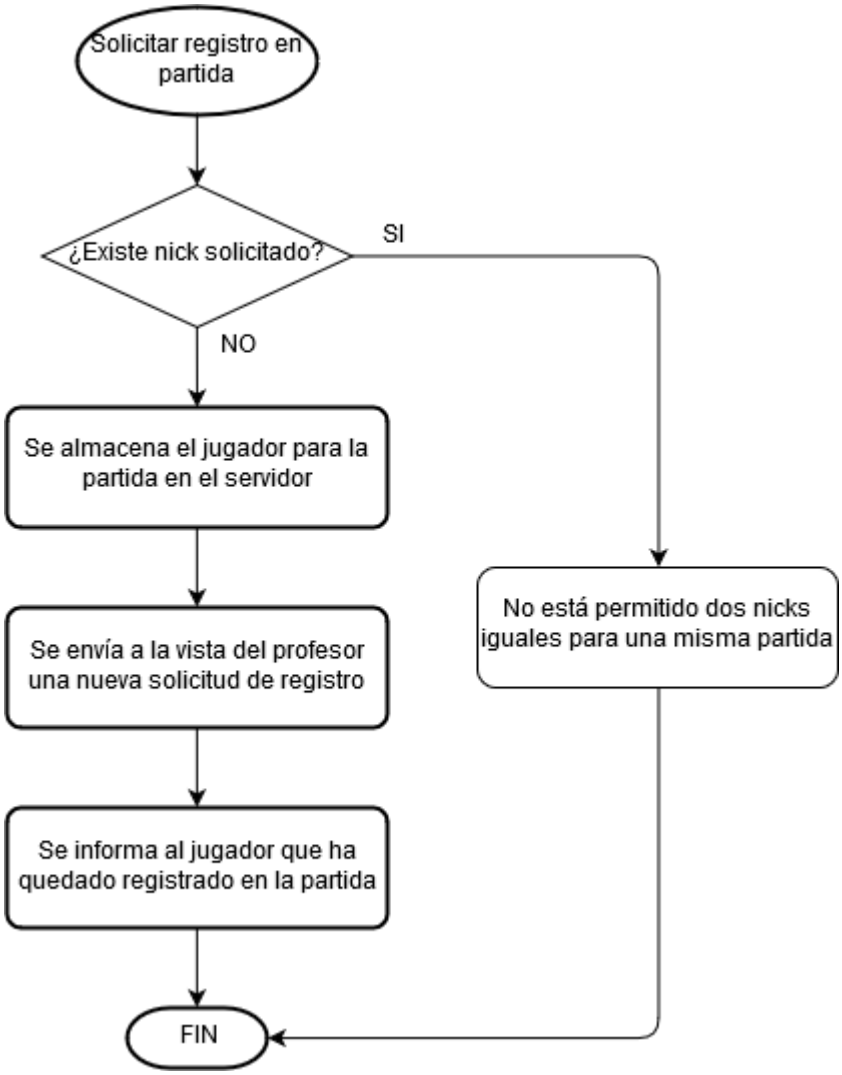


Ilustración 97: Diagrama de flujo DF-003.

En el siguiente diagrama (*DF-004*) se recogen los procesos internos que tienen lugar en el momento que un profesor gestiona el registro de un jugador, realizando modificaciones sobre el mismo o eliminandolo del registro.

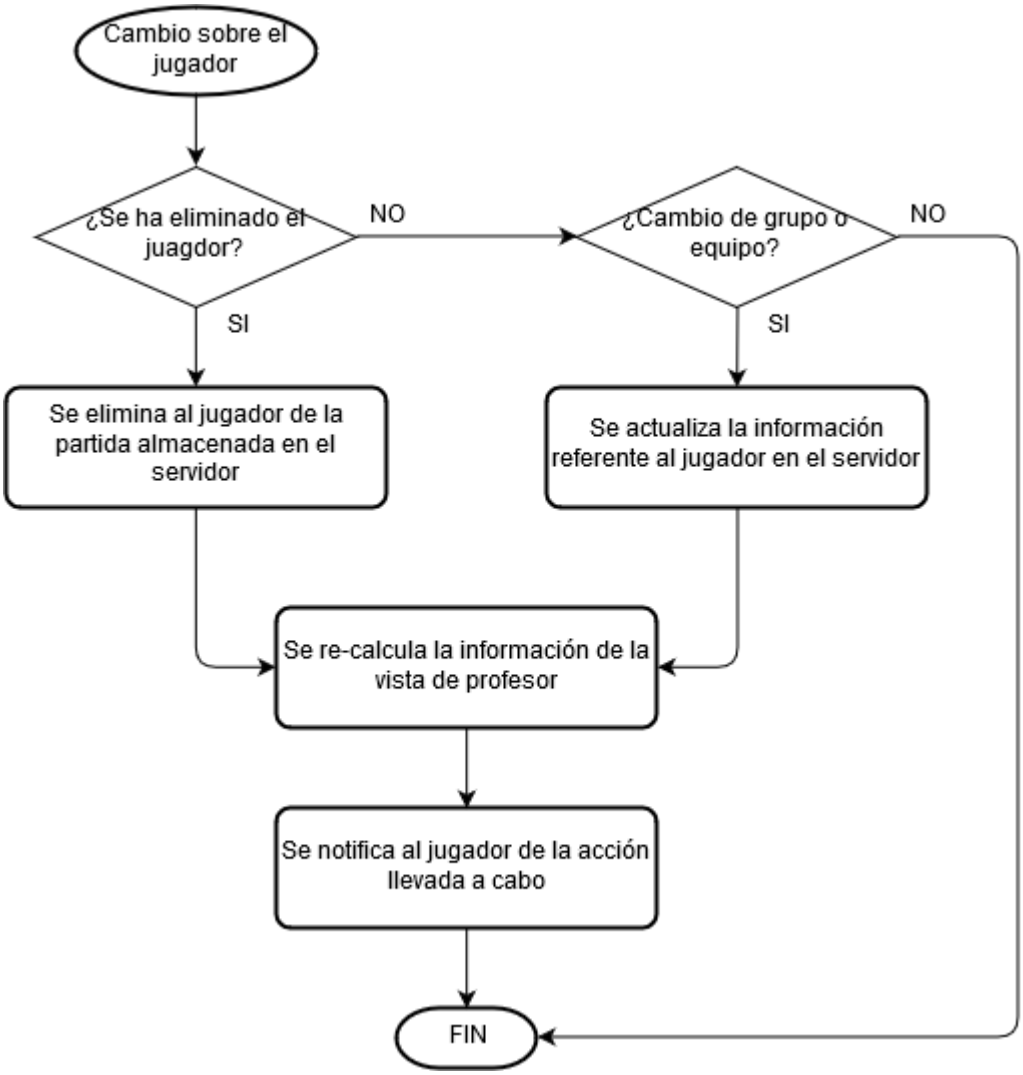


Ilustración 98: Diagrama de flujo DF-004.

En el siguiente diagrama (*DF-005*) se recogen los procesos internos que tienen lugar en el momento en que el juego lanza una nueva consulta, es decir, una votación o una pregunta.

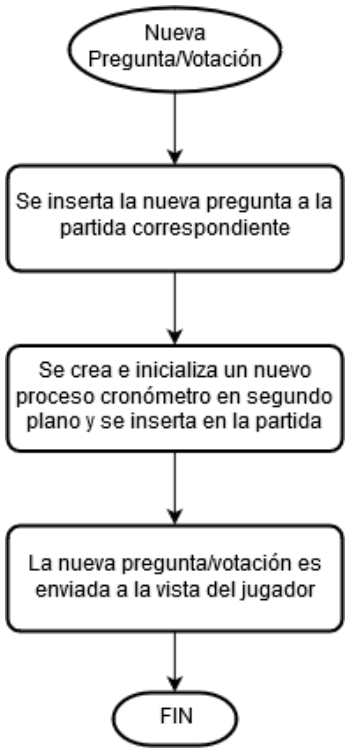


Ilustración 99: Diagrama de flujo DF-005.

En el siguiente diagrama (*DF-006*) se recogen los procesos internos en el momento en que un jugador se desconecta de la partida (cierre de la ventana de juego).

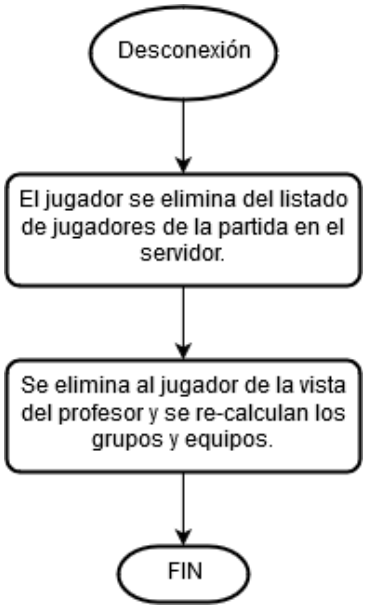


Ilustración 100: Diagrama de flujo DF-006.

En el siguiente diagrama (*DF-007*) se recogen los procesos internos que tienen lugar en el momento en que un jugador responde a una consulta.

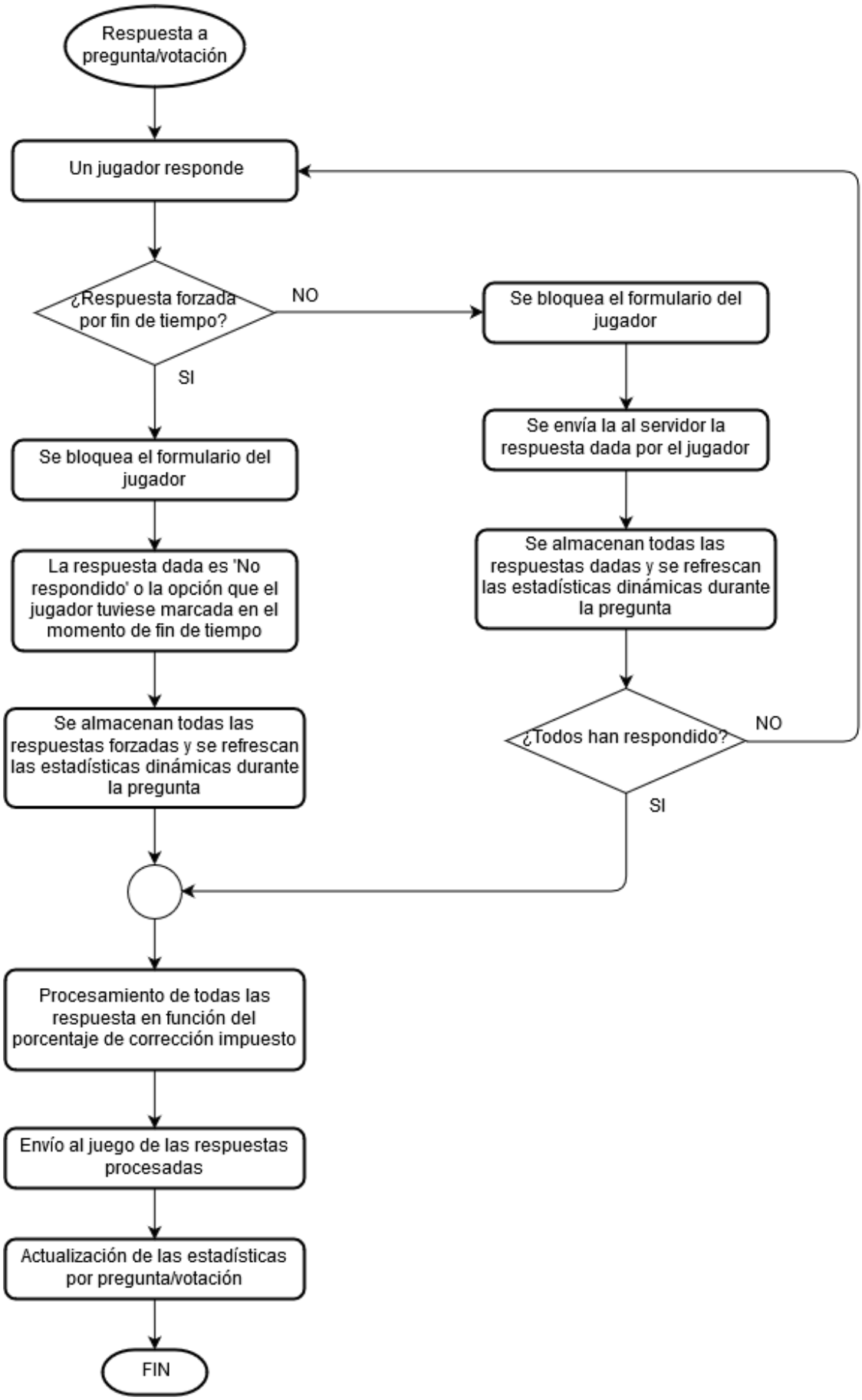


Ilustración 101: Diagrama de flujo DF-007.

En el siguiente diagrama (*DF-008*) se recogen los procesos internos que tienen lugar en el momento en que la partida finaliza.

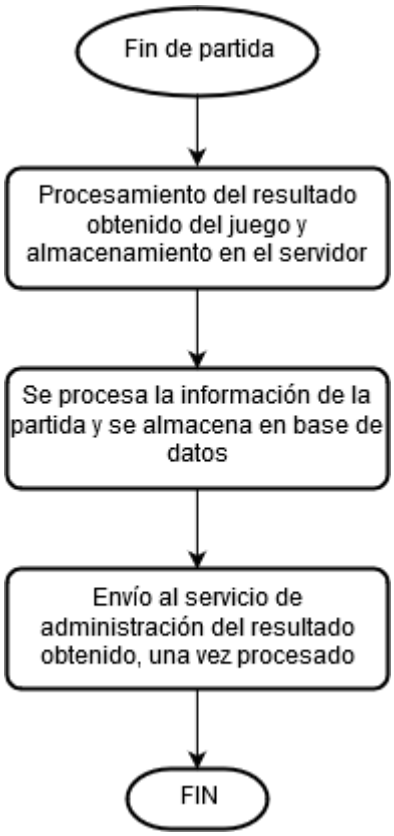


Ilustración 102: Diagrama de flujo DF-008.

4.4 Diagramas de Secuencia

En el siguiente diagrama *(Fig. 103)* se define el intercambio de mensajes para el registro de la partida. El profesor solicita el registro de una nueva partida y será el servidor el encargado de construir y enviar a la vista del profesor el objeto *DatosIniciales* con el que la aplicación de Descartes generará la configuración del juego.

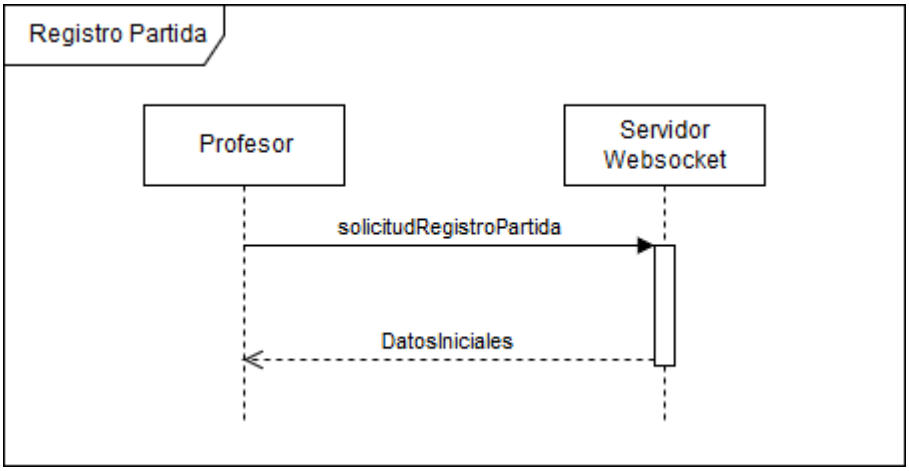


Ilustración 103: Diagrama de secuencia Registro Partida.

En el siguiente diagrama *(Fig. 104)* se define el intercambio de mensajes para el seguimiento de respuestas. El servidor envía una marca de tiempo cada segundo a la página de seguimiento para que se refresque el tiempo restante y las preguntas respondidas por los jugadores.

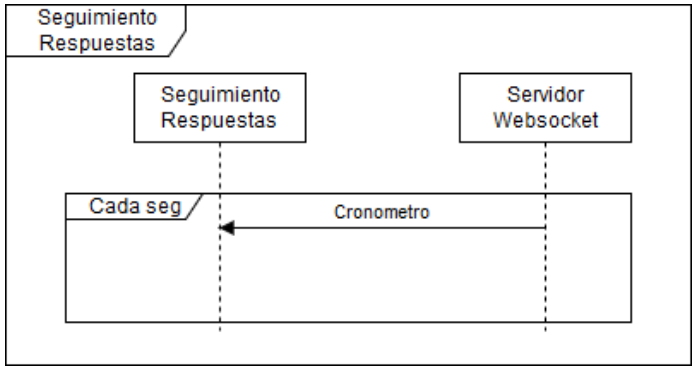


Ilustración 104: Diagrama de secuencia Seguimiento Respuestas.

El siguiente diagrama *(Fig. 105)* define la comunicación para el registro de jugadores, se divide en tres etapas: solicitud de acceso, validación de acceso y acciones de administración por parte del profesor.

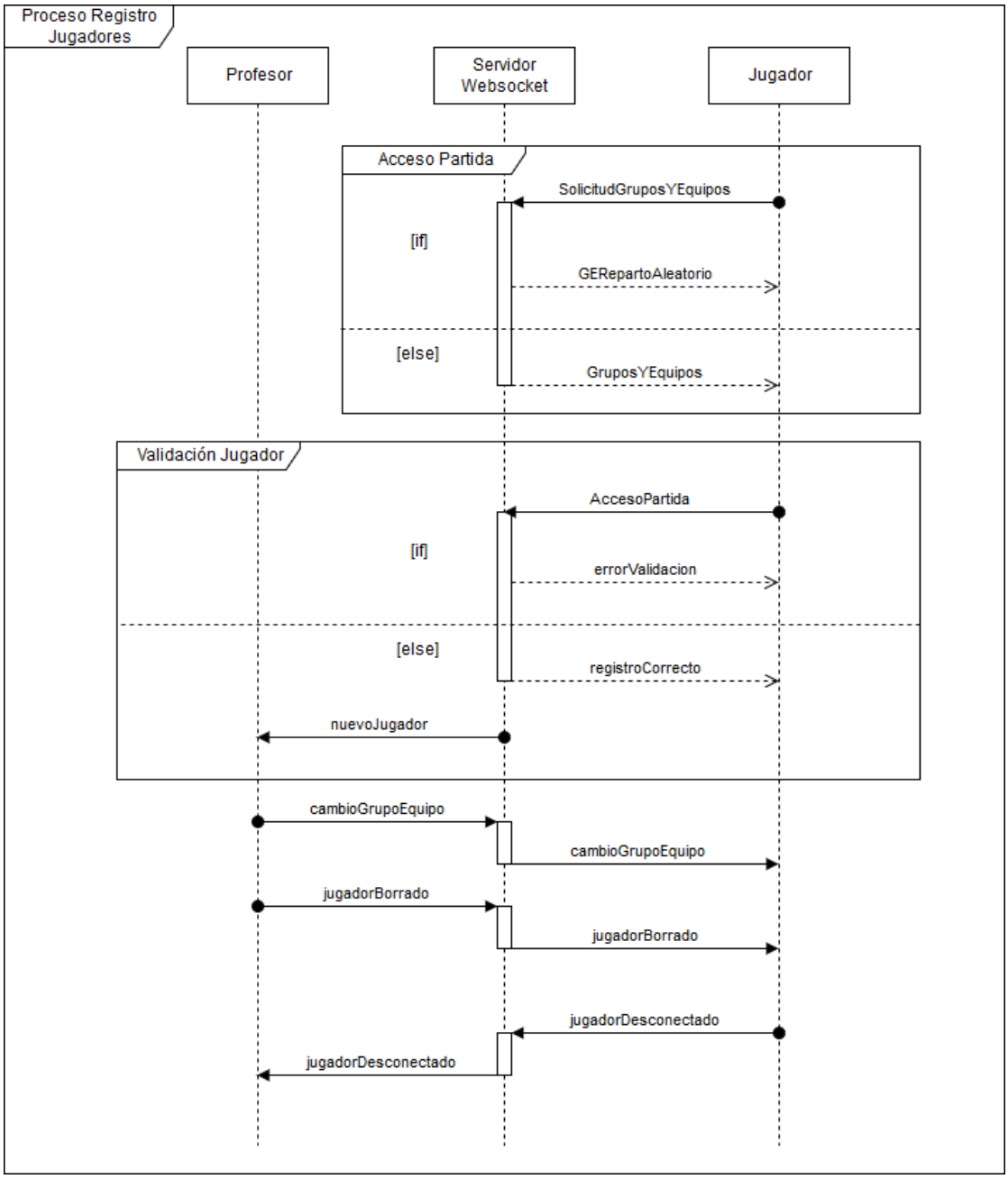


Ilustración 105: Diagrama de secuencia para el Registro de Jugadores.

El siguiente diagrama *(Fig. 106)* define la comunicación que tiene lugar durante el proceso de juego de una ronda de preguntas, se divide en cuatro etapas y un proceso en segundo plano: el envío de una nueva pregunta, la respuesta a la pregunta, la finalización de una pregunta, la finalización de la partida y un evento constante de envío de tiempo por parte del servidor al navegador de los jugadores para marcar el tiempo de respuesta.

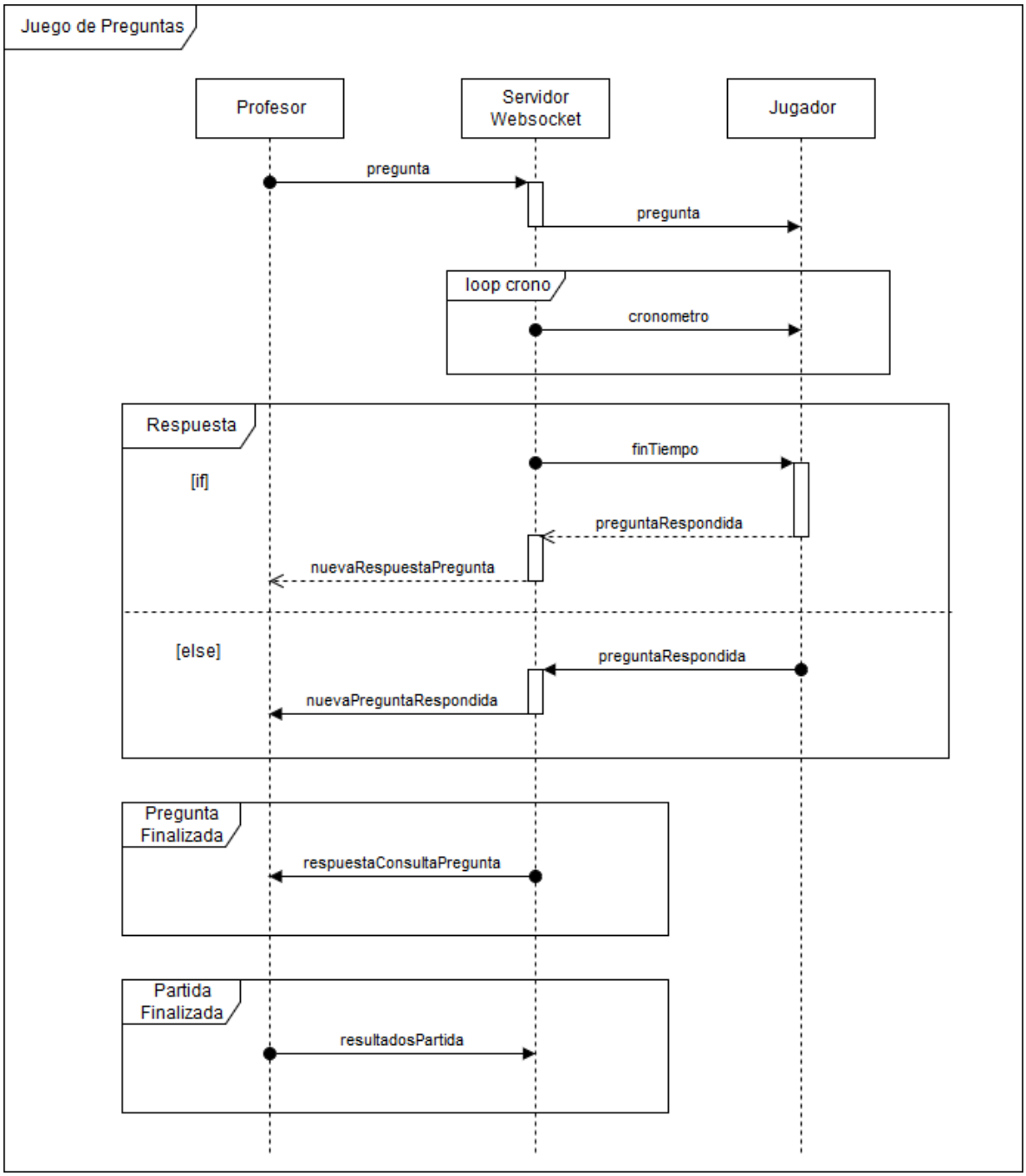


Ilustración 106: Diagrama de secuencia Juego de Preguntas.

El siguiente diagrama *(Fig. 107)* define la comunicación que tiene lugar durante el proceso de juego de una ronda de votaciones. El comportamiento es exactamente igual que para una ronda de preguntas.

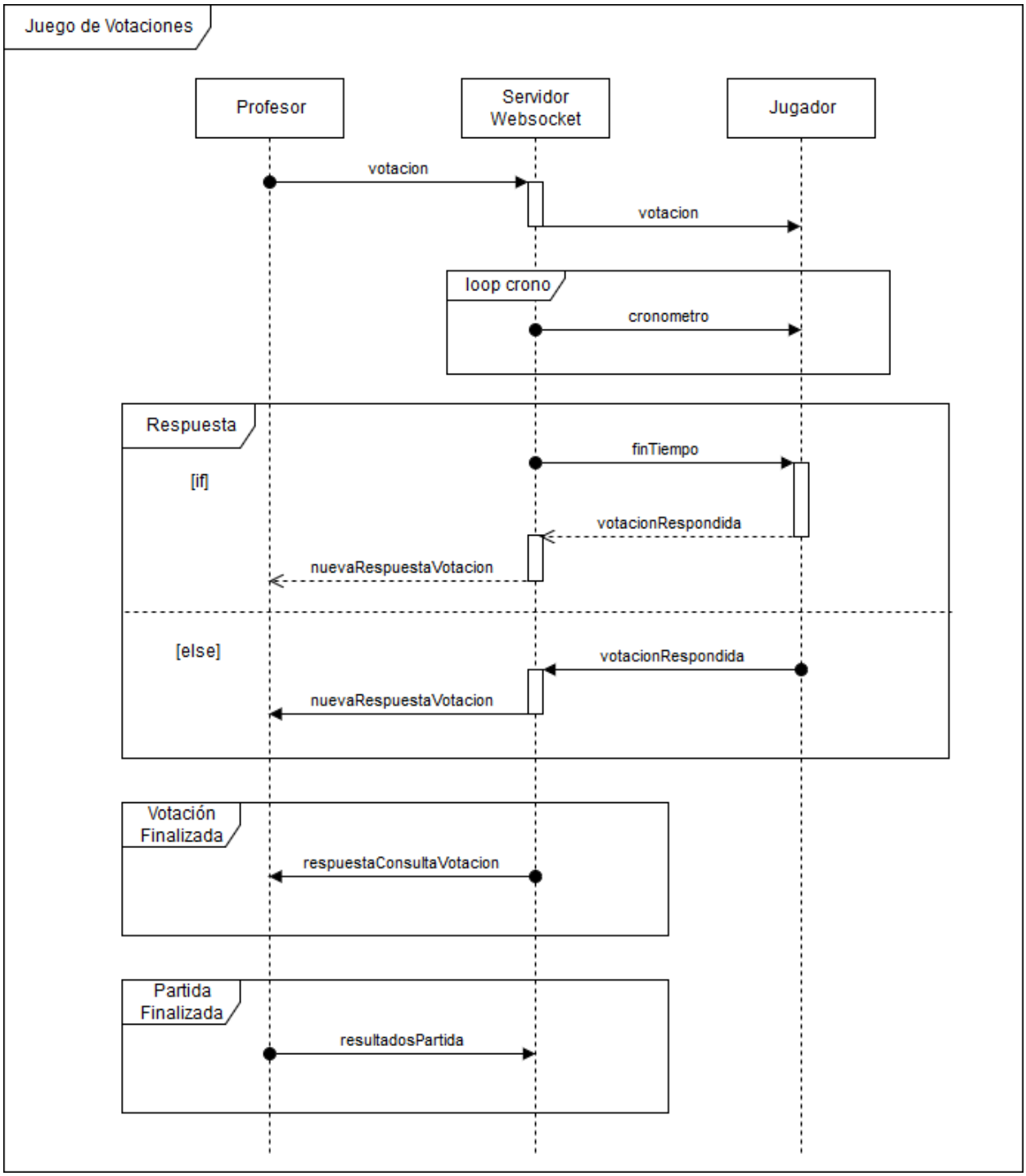


Ilustración 107: Diagrama de secuencia Juego de Votaciones.

4.5 Modelo E/R de la base de datos

En la siguiente ilustración *(Fig. 108)* se recoge el modelo de la base de datos.

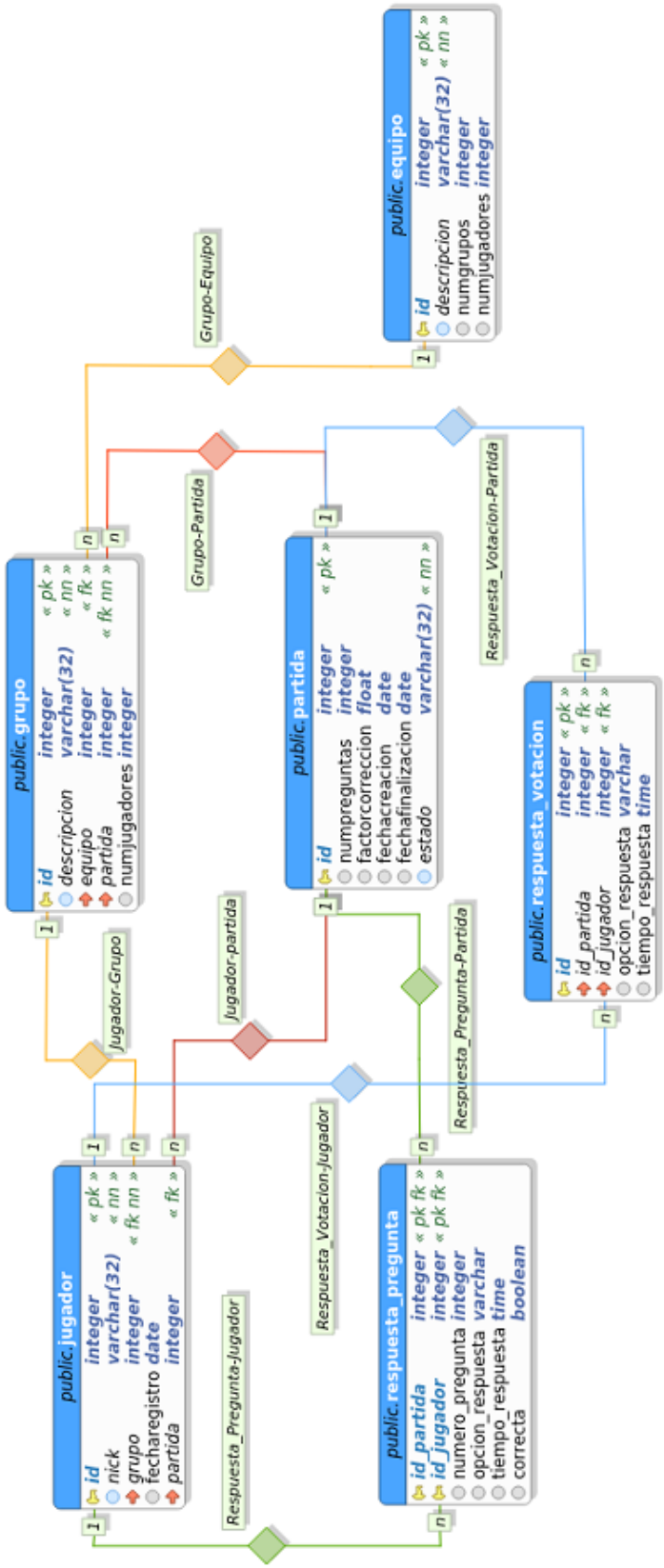


Ilustración 108: Diagrama modelo base de datos.

El diseño de la base de datos está basado en la creación automática de índices para las tablas, cada elemento de la tabla es único respecto al índice, el objetivo es la eficiencia de las consultas mediante el indexado.

La especificación detalla de las tablas del modelo se recoge en la siguiente tabla (*Tabla 3*):

Tabla	Función de la tabla	Columnas	Función de la columna
Partida	Guarda la información relevante sobre las partidas registradas.	*id*	ID generado automática mente para identificar de forma única la partida.
		numpreguntas	Número de preguntas que conforman la partida.
		factorcorrección	Porcentaje de corrección aplicado, si el % es del 60% y el grupo se compone de 10 jugadores, al menos 6 deberán acertar su pregunta para que el grupo la supere.
		fechacreación	Fecha de creación de la partida en el microservicio de websockets.
		fechafinalización	Fecha en la que se finaliza la partida jugada.
		estado	Estado en el que se encuentra la partida.
Equipo	Guarda la información relevante sobre los equipos registrados.	*id*	ID generado automática mente para identificar de forma única el equipo.
		descripción	Identificación del nombre del equipo.
		numgrupos	Número de grupos que conforman el equipo.
		numjugadores	Numero de jugadores que conforman el equipo.
Grupo	Guarda la información relevante sobre los grupos registrados.	*id*	ID generado automática mente para identificar de forma única el grupo.
		descripción	Identificación del nombre del grupo.
		equipo	ID del equipo al que pertenece.
		partida	ID de la partida a la que pertenece.
		numjugadores	Número de jugadores que componen el grupo.

Jugador	Guarda la información relevante sobre los jugadores registrados.	*id*	ID generado automática mente para identificar de forma única el jugador.
		nick	Nick usado por el jugador.
		grupo	ID del grupo al que pertenece.
		fecharegistro	Fecha en la que se ha registrado el jugador.
		partida	ID de la partida a la que pertenece.
Respuesta pregunta	Almacena información estadística sobre la respuesta por parte de los jugadores a las preguntas realizadas.	*id_partida*	ID de la partida a la que pertenece, forma parte de la clave primaria.
		id_jugador	ID del jugador que ha formulado la respuesta, forma parte de la clave primaria.
		num_pregunta	Posición de la pregunta en el listado de preguntas de la partida.
		opción_respuesta	Respuesta ofrecida por el jugador.
		tiempo_respuesta	Tiempo invertido por el jugador para responder.
		correcta	Indica si la respuesta del jugador ha sido correcta o no.
Respuesta votación	Almacena información estadística sobre la respuesta por parte de los jugadores a las votaciones realizadas.	*id*	ID generado automática mente para identificar de forma única la respuesta, ya que no se dispone de número de votación para construir una clave primaria similar a la de la respuesta pregunta.
		id_partida	ID de la partida a la que pertenece.
		id_jugador	ID del jugador que formula la respuesta.
		opción_respuesta	Opción ofrecida por el jugador.
		tiempo_respuesta	Tiempo invertido por el jugador en responder.

Tabla 5: Definición de la información almacenada en la base de datos.

5 INTERFAZ DE USUARIO

"Uno se sienta más cómodo en un color que le gusta".

- Verner Panton -

En esta sección se describe la interacción de los usuarios con la interfaz gráfica, tanto de los jugadores para la partición en los juegos, como de los profesores a la hora de administrar el juego y el registro de alumnos a la partida. Se especificarán también los requisitos funcionales que se han tenido en cuenta.

5.1 Inicio de Partida

En el momento que el profesor ha registrado la partida y quiere dar comienzo al inicio del juego (*CU-005*) se accede a la vista de registro de jugadores *(Fig. 109)*.

Ilustración 109: Vista comienzo de partida.

En esta pantalla aparece el identificador único de partida, que los jugadores necesitan para solicitar el registro a esta partida en concreto.

El profesor cuanta con un cuadro de control donde puede realizar las siguientes acciones:

- Dar comienzo a la partida.
- Controlar el tiempo de respuesta, deteniéndolo y activándolo en el momento que considere preciso.
- Una vez finalizada una partida dar comienzo a una nueva partida.

En el momento en que los jugadores vayan accediendo se notificará en pantalla que un nuevo jugador ha solicitado acceso, y aparecerá en el listado de jugadores ordenados por grupo y equipo en caso de ser partida por equipos, en caso contrario pertenecerá al equipo por defecto (sin equipos), la pantalla puede verse en la siguiente ilustración *(Fig. 110)*.

Ilustración 110: Vista solicitudes de acceso a partida.

Los jugadores son agrupados por grupos y los grupos son agrupados por equipos (en caso de haberlos) para una menor administración por parte del profesor.

5.2 Acceso de Jugadores

En el lado del jugador, la primera acción que debe realizar es el acceso a la partida (*CU-002*), para ello deberá indicar el identificador de la partida, con un valor numérico, en caso contrario no dejará la escritura en el formulario de acceso (Fig. 111).

Ilustración 111: Vista acceso a partida.

Todos los botones de la interfaz del jugador disponen de estilo para que cuando el ratón pasa por encima o se seleccione cambie el sombreado del botón.

Una vez solicitado el acceso, la información de grupos y equipos disponible será enviada al navegador (*CU-002*) del alumno para su elección, esto forma parte del formulario de registro del jugador *(Fig. 112)*.

Ilustración 112: Vista formulario acceso jugadores.

En ese formulario, el jugador debe introducir el identificador de la partida que aparece por defecto, su Nick de jugador y seleccionar el equipo y el grupo, cumpliéndose las siguientes condiciones:

- El seleccionable de equipos y grupos será rellenado automáticamente al cargarse la pantalla.
- El seleccionable de equipos al ser opcional no aparecerá en caso de no ser una partida por equipos.
- En caso de ser partida por equipos, primero se seleccionará el equipo, una vez seleccionado el desplegable de grupo será recalculado automáticamente mostrando únicamente los grupos pertenecientes al equipo seleccionado.
- El jugador podrá cambiar de decisión, siempre y cuando no pulse sobre el botón de comienzo.
- Una vez pulsado el botón de comienzo se enviará al servidor la petición de registro del jugador y esta será transmitida al profesor.

En el momento que se realice la petición de registro, el jugador no podrá realizar cambios en el formulario y este quedará bloqueado *(Fig. 113)* hasta que se reciba un nuevo evento:

- Un cambio de equipo o grupo, lo cual dejaría el formulario en el mismo estado, pero con el nuevo equipo y grupo.
- La eliminación del jugador, lo cual le devolvería al formulario de registro.
- Un error de validación y de grupo completo, lo cual le devolvería al formulario de registro.

Ilustración 113: Vista registro de jugador solicitado.

5.3 Registro de Jugadores

Mientras los jugadores están en proceso de registro el profesor podrá ir realizando modificaciones en los jugadores (*CU-003*), grupos y equipos para la correcta administración de la partida *(Fig. 114)*.

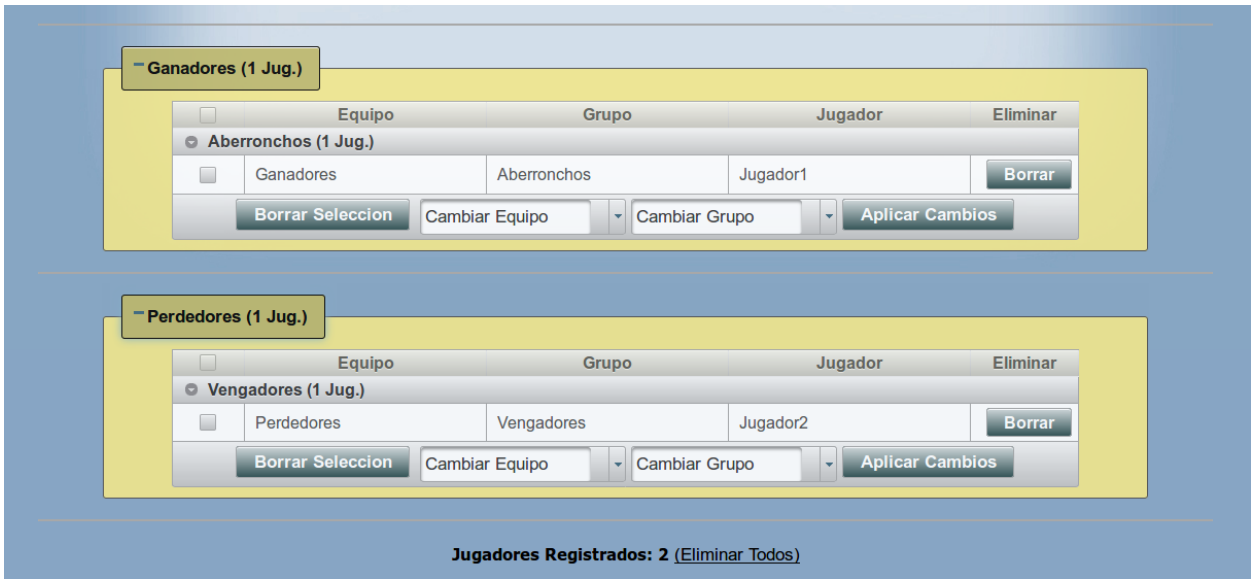


Ilustración 114: Vista administración registro jugadores.

Los jugadores serán agrupados en grupos en un formato de tabla, la tabla contendrá información sobre los jugadores (Nick, equipo y grupo) y un contador total del número de jugadores que tiene el grupo.

A su vez, los grupos estarán agrupados en equipos, en caso de ser partida por equipos, los equipos serán desplegables con el nombre del equipo y el número total de jugadores que tiene el equipo.

Así mismo, las funciones que el profesor puede realizar como administrador de la partida son:

- Cada jugador podrá ser eliminado de forma individual mediante el botón "Borrar".
- Es posible realizar la selección de varios jugadores dentro de un mismo grupo y eliminarlos conjuntamente mediante el botón "Borrar selección".
- Cada jugador podrá ser editado individualmente, puede realizarse pulsando sobre la celda del jugador que queremos modificar, esta pasará a modo editable mediante un desplegable precargado con la información disponible para modificar, en el momento que se pulse el botón de intro el jugador quedará modificado y una notificación será enviada por pantalla. Cambios sobre un jugador en la vista del profesor repercute sobre la información almacenada en el servidor y un aviso será enviado al jugador, quedando constancia de los cambios en su pantalla de registro.
- También es posible la opción de modificar un conjunto de jugadores, simplemente basta con seleccionarlos y establecer en los desplegables de cambio el equipo y grupo al que se desea mover a los jugadores. Estos desplegables se recalculan automáticamente, es decir, si en el desplegable de equipo se establece el equipo A, en el desplegable de grupo solo estarán disponibles los grupos pertenecientes al equipo A. Si se quiere que los cambios se hagan efectivos se debe de pulsar el botón de "Aplicar Cambios".
- En el pie de la página aparece un contador de todos los jugadores que se encuentran registrados en la partida. Además, se incluye la opción de eliminación de todos los jugadores, volviendo al inicio del registro.

Una vez finalizado el registro, el profesor deberá pulsar el botón "Comenzar partida", esto dará por finalizado y validado el proceso de registro. A partir de este momento nadie podrá acceder a la partida y los jugadores que se desconecten aparecerán con estadísticas nulas en la partida. En este momento será cargada la interfaz de la herramienta *AJDA* en pantalla para el inicio del juego *(Fig. 115)*.

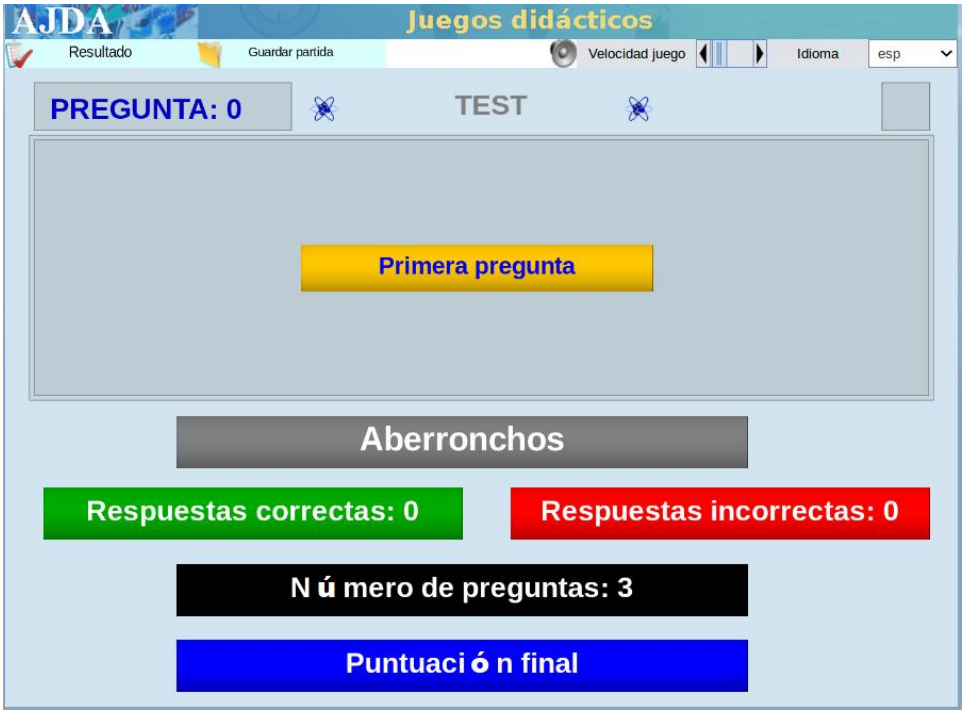


Ilustración 115: Comienzo del juego.

En el momento que se pulse sobre primera pregunta, el juego dará comienzo.

5.4 Formulario de respuesta

En el momento en que da comienzo una pregunta, el enunciado y las opciones a marcar se encuentran proyectadas en la pantalla del profesor. A los alumnos en su terminal le aparecerá un formulario de respuesta (*CU-004*) con las opciones disponibles *(Fig. 116, 117, 118):*

Ilustración 116: Vista formulario de respuesta 01.

Ilustración 117: Vista formulario de respuesta 02.

Ilustración 118: Vista formulario de respuesta 03.

El formulario de respuesta se compone de:

- Una barra de tiempo decremental, del cual dispone para responder a la pregunta, cada actualización de tiempo es un mensaje del servicio cronómetro en el servidor ejecutado en segundo plano. La barra comienza en color verde, cuando falte el 60% del tiempo pasará a tono amarillo de aviso y cuando falte el 30% del tiempo se verá en tono crítico de color rojo.
- Un menú de botones con las opciones de respuesta, 10 botones como máximo, en caso de haber más de 10 opciones posibles se mostrará un desplegable con las opciones. En caso de ser respuesta escrita aparecerá una entrada de texto. En caso de marcar un botón, este cambiará su estilo para determinar que se encuentra seleccionada esa opción. Se podrá cambiar de opción siempre y cuando no se agote el tiempo.
- Un botón para el envío de la respuesta, una vez marcado este botón no habrá vuelta atrás y la respuesta será enviada.

En el momento que la respuesta es enviada, el formulario quedará bloqueado en espera de que finalice el tiempo. Una vez finalizado el tiempo se mostrará un formulario de resultados a la pregunta con: la opción correcta, la opción marcada y el tiempo de respuesta.

En el momento en que una nueva pregunta sea lanzada, el proceso se repite.

5.5 Seguimiento de respuestas

Con el objetivo de llevar un seguimiento de las respuestas que van dando los jugadores, en la vista del profesor aparece un pequeño *iframe* con el seguimiento de las respuestas *(Fig. 119):*

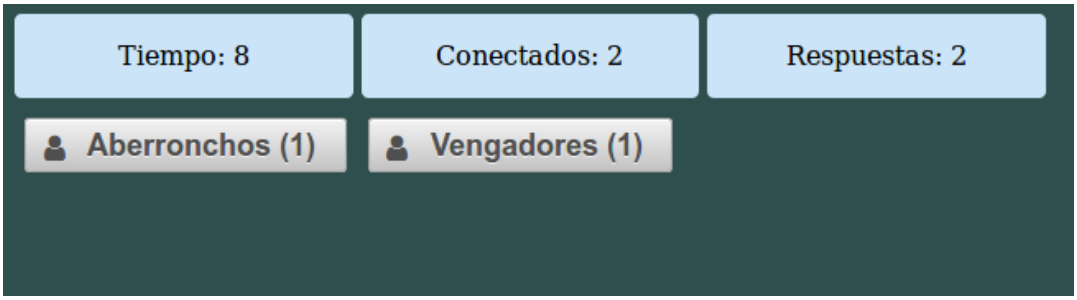


Ilustración 119: Iframe seguimiento de respuestas 01.

En este *Iframe* se muestra la siguiente información:

- Tiempo restante de pregunta, cada segundo el servicio cronómetro enviará la marca de tiempo a la vista del profesor.
- El número de jugadores conectados a la partida.
- El número de respuestas dadas.
- Un botón por grupo, con el número de respuestas dadas por el grupo, el cual permite pulsar sobre el para ver el listado de jugadores que ya han dado respuesta *(Fig. 120).*

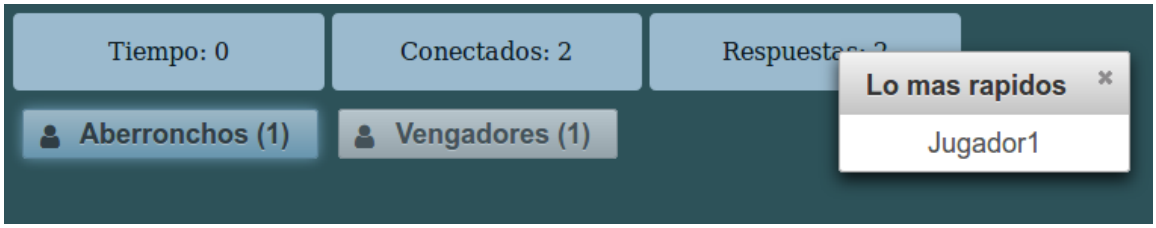


Ilustración 120: Iframe seguimiento de respuestas 02.

5.6 Estadísticas

En el momento en que finaliza la pregunta, se muestra en un pequeño *iframe* en la vista del profesor con las estadísticas de la pregunta *(Fig. 121):*

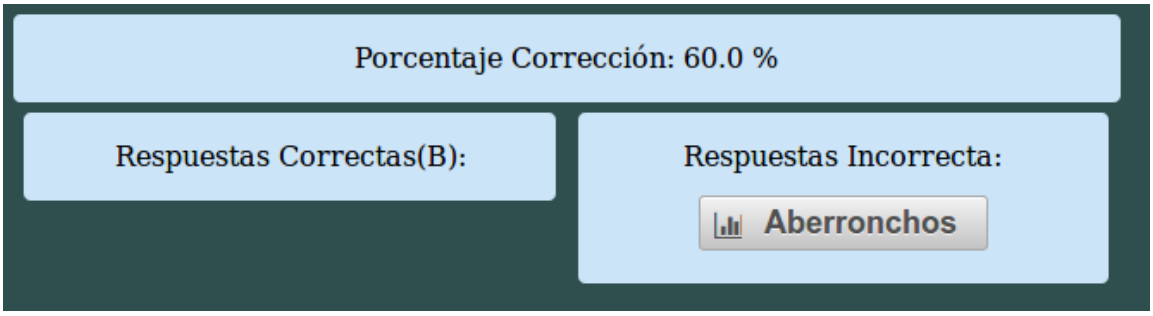


Ilustración 121: Vista estadísticas por pregunta 01.

En esta vista se muestra la siguiente información:

- El porcentaje de corrección aplicado a la pregunta.
- Un bloque de respuestas correctas, entre paréntesis la respuesta correcta, donde se posicionan los grupos cuya respuesta final ponderada haya sido correcta.
- Un bloque de respuestas incorrectas con los grupos que no hayan superado la pregunta.

Cada botón contiene la estadística de cada grupo, recogida en una gráfica de pastel con las opciones disponibles y las respuestas dadas por los jugadores pertenecientes al grupo *(Fig. 122):*

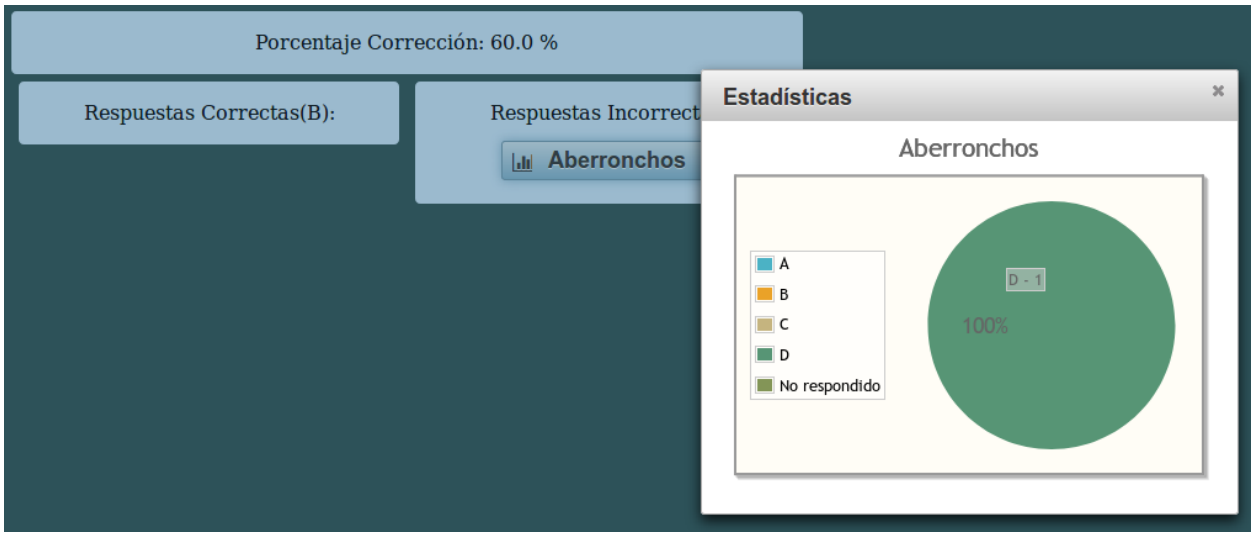


Ilustración 122: Vista estadísticas por pregunta 02.

5.7 Fin de partida

Una vez finalizada la partida, se pulsa sobre el botón “Juego finalizado” de la herramienta AJDA *(Fig. 123),* esto enviará los resultados de la partida al servidor para ser almacenados.

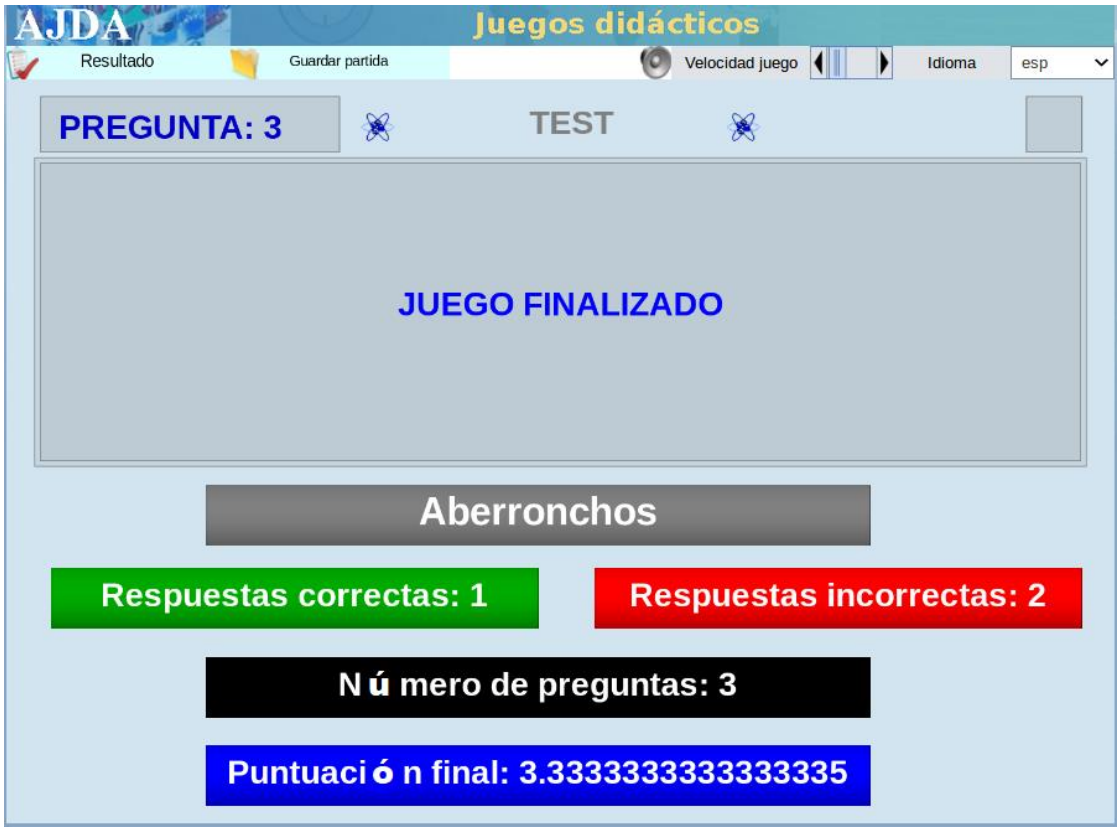


Ilustración 123: Vista juego finalizado.

En las siguientes imágenes se recoge como los datos son almacenados tras la realización de la partida número 500:

```
mibbdd=> select * from partida where id = 500;
 id  | numpreguntas | factorcorreccion | fechacreacion | fechafinalizacion |   estado
-----+--------------+------------------+---------------+-------------------+------------
 500 |            3 |               60 | 2018-12-27    | 2018-12-27        | REGISTRADA
(1 row)
```

Ilustración 124: Almacenamiento tras partida, entidad partida.

```
mibbdd=> select * from equipo where partida = 500;
 id | descripcion | partida | numgrupos | numjugadores
----+-------------+---------+-----------+--------------
 16 | Ganadores   |     500 |         1 |            1
 17 | Perdedores  |     500 |         1 |            0
(2 rows)
```

Ilustración 125: Almacenamiento tras partida, equipos.

```
mibbdd=> select * from grupo where partida = 500;
 id | descripcion | equipo | partida | numjugadores
----+-------------+--------+---------+--------------
  8 | Aberronchos |     16 |     500 |            1
  9 | Vengadores  |     17 |     500 |            0
(2 rows)
```

Ilustración 126: Almacenamiento tras partida, grupos.

```
mibbdd=> select * from respuesta_pregunta where id_partida = 500;
 id_partida | id_jugador | numero_pregunta | opcion_respuesta | tiempo_respuesta | correcta
------------+------------+-----------------+------------------+------------------+----------
        500 |          4 |               2 | D                | 01:00:07         | f
        500 |          4 |               3 | A                | 01:00:16         | f
        500 |          4 |               4 | D                | 01:00:14         | t
(3 rows)
```

Ilustración 127: Almacenamiento tras partida, respuestas.

6 VALIDACIÓN

"Si buscas resultados distintos, no hagas siempre lo mismo".

- Albert Einstein -

En esta sección se recoge la validación de la aplicación mediante la definición de una serie de casos de pruebas unitarias e integradas donde se contemplan los objetivos marcados y el grado de cumplimiento alcanzado.

6.1 Entorno de prueba

El entorno utilizado para la realización de las pruebas se compone por:

- Portátil MSI CX62 6QD
- Sistema operativo Linux Ubuntu 16.02
- Navegador Mozilla Firefox 10.0
- Versión Websocket 13.0

6.1 Casos de pruebas unitarias

Nombre	**PU-1**
Objetivo	Borrar un jugador del registro de partida de forma individual.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de uno o más alumnos que soliciten acceso.
Agentes	Usuario profesor, usuario alumno.
Definición	Sobre el listado de alumnos registrados, con el profesor pulsar en el botón borrar de la fila del jugador.
Resultados esperados	El jugador es borrado del registro de la partida.
Resultlados obtenidos	El jugador es borrado del registro de la partida.
Comentarios	Este jugador debe borrarse tanto del listado de jugadores de la vista, como el listado en la caché del servidor.

Tabla 6: Caso de prueba unitaria PU-1-

Nombre	PU-2
Objetivo	Borrar una selección de más de un jugador.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de varios alumnos que hayan solicitado acceso.
Agentes	Usuario profesor, dos usuarios alumnos.
Definición	Sobre la lista de alumnos registrados, con el profesor seleccionar con los check dos jugadores y pulsar sobre el botón de borrar selección.
Resultados esperados	Los jugadores son borrados del registro de la partida.
Resultlados obtenidos	Los jugadores son borrados del registro de la partida.
Comentarios	Los jugadores deben borrarse tanto del listado de jugadores de la vista, como del listado en la caché del servidor.

Tabla 7: Caso de prueba unitaria PU-2.

Nombre	PU-3
Objetivo	Aplicar un cambio de grupo o equipo a un jugador de forma individual.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de uno o más alumnos que soliciten acceso.
Agentes	Usuario profesor, usuario alumno.
Definición	Sobre el listado de alumnos registrados, con el profesor hacer click sobre el grupo o equipo del jugador y seleccionar un nuevo grupo o equipo en el desplegable.
Resultados esperados	En el momento de pulsar sobre la casilla de grupo o equipo debe abrirse un desplegable con los grupos o equipos disponibles, en el momento de seleccionar un nuevo grupo o equipo el jugador debe cambiar de grupo o equipo y situarse en el listado de grupo o equipo correspondiente.
Resultlados obtenidos	Se muestra el desplegable con los grupos y equipos. Al seleccionar un nuevo grupo o equipo el jugador se situa en el listado de grupo o equipo correspondiente.
Comentarios	La información debe actualizarse tanto en los datos de la página como en la caché del servidor.

Tabla 8: Caso de prueba unitaria PU-3.

Nombre	PU-4
Objetivo	Aplicar un cambio de grupo o equipo a varios jugadores.
Requisitos	Disponer de un usuario profesor que haya creado una partida y varios alumnos que soliciten acceso.
Agentes	Usuario profesor, dos usuarios alumno.
Definición	Sobre el listado de alumnos registrados, con el profesor seleccionar dos alumnos con los checks, seleccionar un equipo o grupo y pulsar sobre el botón de aplicar cambios.
Resultados esperados	Al pulsar sobre los checks los jugadores deben quedar seleccionados. Tras seleccionar un equipo en el desplegable de equipos, el desplegable de grupos debe recalcularse para mostrar los grupos pertenecientes a dicho equipo. Una vez aplicados los cambios, los jugadores deben redistribuirse al

	listado de grupo o equipo correspondiente.
Resultlados obtenidos	Jugadores marcados correctamente. Recálculo del desplegable de grupos en función del equipo seleccionado. Los jugadores se redistribuyen correctamente.
Comentarios	La información debe actualizarse tanto en los datos de la página como en la caché del servidor.

Tabla 9: Caso de prueba unitaria PU-4.

Nombre	**PU-5**
Objetivo	Borrar todos los juagdores del registro.
Requisitos	Disponer de un usuario profesor que haya creado una partida y varios alumnos que soliciten acceso.
Agentes	Usuario profesor, tres usuarios jugadores.
Definición	Con el profesor pulsar sobre el link "*Eliminar todos*".
Resultados esperados	Todos los jugadores del registro son eliminados.
Resultlados obtenidos	Todos los jugadores del registro son eliminados.
Comentarios	La información debe actualizarse tanto en los datos de la página como en la caché del servidor.

Tabla 10: Caso de prueba unitaria PU-5.

Nombre	**PU-6**
Objetivo	Comprobar que la solicitud de acceso a partida se realiza correctamente.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de uno o varios alumnos que soliciten acceso.
Agentes	Usuario profesor, usuario alumno.
Definición	El usuario jugador debe introducir el identificador de la partida en el terminal del jugador y solicitar acceso.
Resultados esperados	La petición de acceso debe lanzarse al servidor, si este la acepta el formulario de entrada del jugador cambiará para poder introducir sus datos y tendrá precargados los grupos y equipos.
Resultlados obtenidos	La petición se envia y el formulario de acceso se habilita precargado con los grupos y equipos seleccionables.
Comentarios	

Tabla 11: Caso de prueba unitaria PU-6.

Nombre	**PU-7**
Objetivo	Comprobar el recálculo de grupos en función de equipo en el formulario de acceso del jugador.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de uno o varios alumnos que soliciten acceso.
Agentes	Usuario profesor, usuario alumno.
Definición	El usuario jugador debe seleccionar un equipo del desplegable de equipos disponibles.
Resultados esperados	Al seleccionar el equipo, el desplegable de grupos debe recalcularse con los grupos pertenecientes a dicho equipo.
Resultlados obtenidos	Al seleccionar el equipo, el desplegable de grupos se recalcula con los grupos pertenecientes a dicho equipo.

Comentarios	

Tabla 12: Caso de prueba unitaria PU-7.

Nombre	**PU-8**
Objetivo	Comprobar que la solicitud de registro a partida se realiza correctamente.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de uno o varios alumnos que soliciten acceso.
Agentes	Usuario profesor, usuario alumno.
Definición	El usuario jugador una vez rellenado el formulario de acceso debe pulsar sobre el botón de enviar.
Resultados esperados	Si el servidor acepta su unión a la partida, el formulario debe quedar bloqueado a la espera del comienzo de la partida.
Resultlados obtenidos	El formulario queda bloqueado en espera del inicio de partida.
Comentarios	

Tabla 13: Caso de prueba unitaria PU-8.

Nombre	**PU-9**
Objetivo	Comprobar que la solicitud de respuesta a pregunta se envía correctamente.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de uno o varios alumnos que soliciten acceso. El juedo debe estar iniciado.
Agentes	Usuario profesor, usuario alumno.
Definición	El usuario jugador selecciona y envía una respuesta a la pregunta.
Resultados esperados	El jugador debe visualizar las opciones de respuesta, así como el tiempo restante para responder. El jugador puede cambiar en todo momento, siempre y cuando el tiempo no esté agotado, su decisión de respuesta. La respuesta seleccionada debe quedar premarcada. Al enviar la respuesta el formulario debe quedar bloquedado hasta la siguiente pregunta.
Resultlados obtenidos	El jugador puede visualizar todas las opciones y el tiempo restante correctamente. El jugador puede cambair su decisión siempre y cuando el tiempo no se agote. La respuesta seleccionada queda premarcada. Al enviar la respuesta el formulario queda bloqueado.
Comentarios	

Tabla 14: Caso de prueba unitaria PU-9.

6.1 Casos de pruebas integradas

Nombre	**PI-1**
Objetivo	Validar la comunicación profesor/alumno ante acciones de administración del profesor durante el registro.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de uno o más alumnos que soliciten acceso.
Agentes	Usuario profesor, usuario alumno.
Definición	Realizar con el profesor una acción de admnistración durante el registro, por ejemplo, borrar un jugador,

	cambiar de grupo o equipo un jugador, borrar varios jugadores.
Resultados esperados	Los cambios realizados por el profesor deben ser notificados al jugador. Si el jugador es borrado debe recibir un mensaje de que ha sido borrado y disponer nuevamente del formulario de registro. Si ha sido reasignado de grupo o equipo debe ser avisado y el formulario permanecer bloqueado, pero con el cambio aplicado.
Resultlados obtenidos	Las acciones del profesor son notifadas al usuario. Si el usuario es borrado, dispone nuevamente del formulario de registro. Si el juador es reasignado a un neuvo grupo o equipo el formulario continúa bloqueado con el cambio aplicado.
Comentarios	Las acciones realizadas por el profesor suponen una comunicación profesor-servidor y servidor-alumno.

Tabla 15: Caso de prueba integrada PI-1.

Nombre	**PI-2**
Objetivo	Validar la comunicación profesor/alumno y el control del servicio cronómetro de la partida.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de uno o más alumnos que soliciten acceso.
Agentes	Usuario profesor, usuario alumno.
Definición	Realizar con el profesor una acción de administración de tiempo de la partida, por ejemplo, detener el tiempo o activar el tiempo.
Resultados esperados	Si el profesor detiene el tiempo, en la pantalla del alumno el cronómetro decremental debe detenerse hasta que el profesor vuelva a activar el tiempo, en este caso, el cronómetro del jugador continúa decrementando hasta llegar a cero.
Resultlados obtenidos	El tiempo queda detenido cuando el profesor pulsa en detener tiempo, y se activa cuando el profesor pulsa sobre activar tiempo.
Comentarios	Las acciones sobre el tiempo requieren de una comunicación profesor-servidor y servidor-jugador.

Tabla 16: Caso de prueba integrada PI-2.

Nombre	**PI-3**
Objetivo	Validar la comunicación juego/alumno.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de uno o más alumnos que soliciten acceso.
Agentes	Usuario profesor, usuario alumno.
Definición	Responder una pregunta recibida por el juego con el usuario alumno.
Resultados esperados	La respuesta dada por el alumno deberá almacenarse en la caché del servidor hasta que todos los jugadores hayan respondido, entonces, el servidor enviará procesadas todas las preguntas al juego.

Resultlados obtenidos	La respuesta del jugador es almacenada en la caché del servidor. El servidor espera la recepción de todas las respuestas, las procesa y las envía al juego. El juego recibe las respuestas procesadas correctamente.
Comentarios	Las acciones de respuesta requieren una comunicación jugador-servidor y servidor-juego.

Tabla 17: Caso de prueba integrada PU-3.

Nombre	**PI-4**
Objetivo	Validar el almacenamiento de los datos de la partida en base de datos.
Requisitos	Disponer de un usuario profesor que haya creado una partida y de uno o más alumnos que soliciten acceso.
Agentes	Usuario profesor, usuario alumno.
Definición	Realizar una partida completa: inicio de partida, registro de jugadores, comienzo de partida, evío de preguntas y respuestas, finalización de partida.
Resultados esperados	Todos los datos ya cciones que intervienen en el proceso de la partida deben quedar almacenados: usuarios que intervienen, partida jugada, grupos y equipos configurados, respuestas dadas por los jugadores. El resultado de la partida queda en la caché del servidor para ser enviado al servicio de administración.
Resultlados obtenidos	Toda la información relativa a jugadores, partidas, grupo equipos y respuestas dadas por los jugadores quedan almacenados en la base de datos. Los resultados de la partida permanecen en la cache del servidor para ser enviados al servicio de administración.
Comentarios	Estas acciones suponen una comunicación de todos los componentes del proyecto, así como una comunicación con la base de datos y el servicio de administración.

Tabla 18: Caso de prueba integrada PU-4.

6.1 Conclusión

Como objetivo del proyecto se pretende el desarrollo de una aplicación multiusuario para el desarrollo de juegos preparados por el profesor en el aula, que facilite las tareas de gestión al profesor y que ofrezca un mayor dinamismo al desarrollo de las partidas, así como el almacenamiento de información referente a las partidas llevadas a cabo. Tal y como se establece en los casos de prueba, todos los resultados obtenidos cumplen con los objetivos del proyecto.

7 LÍNEA DE MEJORA Y CONCLUSIONES

"En toda cosa hay que considerar el fin".

- Jean de la Fontaine -

En esta sección se definen las líneas de mejora que se dejan pendiente para futuras actualizaciones del aplicativo. Además, se realiza una retrospectiva hacia los conocimientos adquiridos y una valoración de las tecnologías utilizadas.

7.1 Mejoras

Las mejoras propuestas para futuras versiones son las siguientes:

- Configuración dinámica del intervalo de tiempo de refresco para el tiempo de respuesta a una pregunta o votación. Actualmente cada segundo el servidor manda un mensaje al navegador del alumno con el tiempo restante. Para no sobrecargar al servidor en caso de haber un número considerable de jugadores en línea, la mejora consistiría en no enviar el refresco cada segundo, sino mandarlo cada X tiempo, siendo ese tiempo un parámetro configurable.
- Configuración dinámica en el registro aleatorio de grupos. Por ejemplo, si una partida se inicia en modo estándar de registro (un jugador accede seleccionando el grupo y equipo deseado), el profesor podrá establecer el registro automático en un momento determinado del registro, pasando de modo estándar a modo de registro aleatorio (el equipo y el grupo se establecen de forma aleatoria para el jugador).
- Reestructuración total de jugadores una vez completado el registro. Una vez finalizado el registro, en caso de no estar conforme con la distribución, el profesor podrá solicitar una reestructuración aleatoria automática de todos los jugadores en todos los grupos y equipos.

7.2 Conclusiones

Los Websockets es una de las características más interesantes introducidas con el nuevo estándar de HTML5, permiten establecer una comunicación bidireccional de baja latencia entre múltiples puntos de comunicación. Lo cual favorece bastante el desarrollo de aplicaciones en tiempo real y multiusuario. Además, ofrece una abstracción de la parte de comunicación, permitiendo la comunicación entre aplicaciones independientemente del lenguaje en el que se hayan escrito, esto abre un amplio abanico de posibilidades en la integración de aplicaciones y el intercambio de mensajes favoreciendo la estandarización de los mecanismos de comunicación.

REFERENCIAS

[1] El proyecto Descartes - http://proyectodescartes.org/descartescms/red-descartes

[2] Herramienta Descartes - http://reddescartes.org/index.html

[3] Protocolo Websocket - https://tools.ietf.org/html/rfc6455

[4] Escribiendo aplicaciones Websocket - https://developer.mozilla.org/es/docs/WebSockets-840092-dup/Escribiendo_servidores_con_WebSocket

[5] API Websocket - https://www.w3.org/TR/websockets

[6] Usando la API Websocket - https://developer.mozilla.org/es/docs/WebSockets-840092-dup/Writing_WebSocket_client_applications

[7] Apache Maven - http://maven.apache.org

[8] Primefaces - https://www.primefaces.org/

[9] Java Server Faces - http://www.javaserverfaces.org/

[10] Java Server Faces - http://www.jtech.ua.es/j2ee/publico/jsf-2012-13/sesion01-apuntes.html

[11] Especificación CDI - http://cdi-spec.org/

[12] Apache Tomcat - http://tomcat.apache.org/

[13] PostgreSQL - https://www.postgresql.org/

[14] SoapUI - https://www.soapui.org/

[15] PgModeler - https://pgmodeler.io/

[16] PgAdmin - https://www.pgadmin.org/

[17] Data Access Object - http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/174

[18] Data Access Object - https://www.oracle.com/technetwork/java/dataaccessobject-138824.html

[19] Doc Spring REST - https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html

[20] Spring REST - https://spring.io/guides/gs/rest-service/

[21] JPA - https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html

[22] JPA - https://www.oscarblancarteblog.com/tutoriales/java-persistence-api-jpa/

[23] https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto

[24] https://shyamapadabatabyal.wordpress.com/tag/long-polling/

[25] https://www.pubnub.com/blog/2014-10-01-websockets-and-long-polling-in-javascript-ruby-and-python/

[26] https://victordiaz.me/websocket

[27] https://dl.packetstormsecurity.net/papers/attack/AC07815487.pdf

[28] https://mvnrepository.com/

[29] https://mind42.com/public/8779ae1d-6ad1-45a0-a26d-86c1c5aca402

[30]http://newton.proyectodescartes.org/juegosdidacticos/index.php?option=com_content&view=article&id=71&Itemid=469&lang=es#

[31] https://www.draw.io/

ANEXO A

A.1 Sentencias SQL para la composición de la base de datos

Para la composición de la base de datos, simplemente basta con ejecutar las siguientes sentencias que crearán las tablas, relaciones y restricciones de nuestra base de datos:

```
-- Database generated with pgModeler (PostgreSQL Database Modeler).
-- pgModeler  version: 0.8.1
-- PostgreSQL version: 9.4
-- Project Site: pgmodeler.com.br
-- Model Author: ---

-- object: public.jugador | type: TABLE --
-- DROP TABLE IF EXISTS public.jugador CASCADE;
CREATE TABLE public.jugador(
	id SERIAL,
	nick varchar(32) NOT NULL,
	grupo integer NOT NULL,
	fecharegistro date DEFAULT null,
	partida integer,
	CONSTRAINT pk_idjugador PRIMARY KEY (id)
);

-- object: public.grupo | type: TABLE --
-- DROP TABLE IF EXISTS public.grupo CASCADE;
CREATE TABLE public.grupo(
	id SERIAL,
	descripcion varchar(32) NOT NULL,
	equipo integer DEFAULT null,
	partida integer NOT NULL,
	numjugadores integer DEFAULT null,
	CONSTRAINT pk_idgrupo PRIMARY KEY (id)
);

-- object: public.equipo | type: TABLE --
-- DROP TABLE IF EXISTS public.equipo CASCADE;
CREATE TABLE public.equipo(
	id SERIAL,
	descripcion varchar(32) NOT NULL,
	partida integer NOT NULL,
	numgrupos integer DEFAULT null,
	numjugadores integer DEFAULT null,
	CONSTRAINT fk_idequipo PRIMARY KEY (id)
);
```

Código 16: Sentencias creación BBDD 01.

```
-- object: public.partida | type: TABLE --
-- DROP TABLE IF EXISTS public.partida CASCADE;
CREATE TABLE public.partida(
	id integer NOT NULL,
	numpreguntas integer DEFAULT null,
	factorcorreccion float DEFAULT null,
	fechacreacion date DEFAULT null,
	fechafinalizacion date DEFAULT null,
	estado varchar(32) NOT NULL DEFAULT null,
	CONSTRAINT pk_idpartida PRIMARY KEY (id)
);

-- object: public.respuesta_pregunta | type: TABLE --
-- DROP TABLE IF EXISTS public.respuesta_pregunta CASCADE;
CREATE TABLE public.respuesta_pregunta(
	id_partida integer NOT NULL,
	id_jugador integer NOT NULL,
	numero_pregunta integer NOT NULL,
	opcion_respuesta varchar(32),
	tiempo_respuesta time,
	correcta boolean,
	CONSTRAINT pk_idrespuesta_pregunta PRIMARY KEY (id_partida,
id_jugador, numero_pregunta)
);

-- object: public.respuesta_votacion | type: TABLE --
-- DROP TABLE IF EXISTS public.respuesta_votacion CASCADE;
CREATE TABLE public.respuesta_votacion(
	id SERIAL,
	id_partida integer NOT NULL,
	id_jugador integer NOT NULL,
	opcion_respuesta varchar(32),
	tiempo_respuesta time,
	CONSTRAINT pk_idrespuesta_votacion PRIMARY KEY (id)
);


-- object: fk_grupo | type: CONSTRAINT --
-- ALTER TABLE public.jugador DROP CONSTRAINT IF EXISTS fk_grupo
CASCADE;
ALTER TABLE public.jugador ADD CONSTRAINT fk_grupo FOREIGN KEY (grupo)
REFERENCES public.grupo (id) MATCH FULL
ON DELETE RESTRICT ON UPDATE NO ACTION;

-- object: fk_idpartida | type: CONSTRAINT --
-- ALTER TABLE public.jugador DROP CONSTRAINT IF EXISTS fk_idpartida
CASCADE;
ALTER TABLE public.jugador ADD CONSTRAINT fk_idpartida FOREIGN KEY
(partida)
REFERENCES public.partida (id) MATCH FULL
ON DELETE RESTRICT ON UPDATE NO ACTION;
```

Código 17: Sentencias creación BBDD 02.

```
-- object: fk_partida | type: CONSTRAINT --
-- ALTER TABLE public.grupo DROP CONSTRAINT IF EXISTS fk_partida
CASCADE;
ALTER TABLE public.grupo ADD CONSTRAINT fk_partida FOREIGN KEY (partida)
REFERENCES public.partida (id) MATCH FULL
ON DELETE RESTRICT ON UPDATE NO ACTION;

-- object: fk_equipo | type: CONSTRAINT --
-- ALTER TABLE public.grupo DROP CONSTRAINT IF EXISTS fk_equipo CASCADE;
ALTER TABLE public.grupo ADD CONSTRAINT fk_equipo FOREIGN KEY (equipo)
REFERENCES public.equipo (id) MATCH FULL
ON DELETE RESTRICT ON UPDATE NO ACTION;

-- object: fk_partida | type: CONSTRAINT --
-- ALTER TABLE public.grupo DROP CONSTRAINT IF EXISTS fk_partida
CASCADE;
ALTER TABLE public.equipo ADD CONSTRAINT fk_partida FOREIGN KEY
(partida)
REFERENCES public.partida (id) MATCH FULL
ON DELETE RESTRICT ON UPDATE NO ACTION;

-- object: fk_partida | type: CONSTRAINT --
-- ALTER TABLE public.respuesta_pregunta DROP CONSTRAINT IF EXISTS
fk_partida CASCADE;
ALTER TABLE public.respuesta_pregunta ADD CONSTRAINT fk_partida FOREIGN
KEY (id_partida)
REFERENCES public.partida (id) MATCH FULL
ON DELETE RESTRICT ON UPDATE NO ACTION;

-- object: fk_jugador | type: CONSTRAINT --
-- ALTER TABLE public.respuesta_pregunta DROP CONSTRAINT IF EXISTS
fk_jugador CASCADE;
ALTER TABLE public.respuesta_pregunta ADD CONSTRAINT fk_jugador FOREIGN
KEY (id_jugador)
REFERENCES public.jugador (id) MATCH FULL
ON DELETE RESTRICT ON UPDATE NO ACTION;


-- object: fk_partida | type: CONSTRAINT --
-- ALTER TABLE public.respuesta_pregunta DROP CONSTRAINT IF EXISTS
fk_partida CASCADE;
ALTER TABLE public.respuesta_votacion ADD CONSTRAINT fk_partida FOREIGN
KEY (id_partida)
REFERENCES public.partida (id) MATCH FULL
ON DELETE RESTRICT ON UPDATE NO ACTION;

-- object: fk_jugador | type: CONSTRAINT --
-- ALTER TABLE public.respuesta_pregunta DROP CONSTRAINT IF EXISTS
fk_jugador CASCADE;
ALTER TABLE public.respuesta_votacion ADD CONSTRAINT fk_jugador FOREIGN
KEY (id_jugador)
REFERENCES public.jugador (id) MATCH FULL
ON DELETE RESTRICT ON UPDATE NO ACTION;
```

Código 18: Sentencias creación BBDD 03.

ANEXO B

B.1 Consumición del servicio REST

En el siguiente fragmento se ofrecen las URL para la consumición del servicio REST que envuelve a la base de datos del proyecto.

```
-- EQUIPO
-- Devuelve todos los equipos
http://<IP>:<PUERTO>/getEquipos
-- Inserta un Equipo
http://<IP>:<PUERTO>/postEquipo

-- GRUPO
-- Devuelve todos los grupos
http://<IP>:<PUERTO>/getGrupos
-- Inserta un Grupo
http://<IP>:<PUERTO>/postGrupo

-- JUGADOR
-- Devuelve todos los jugadores
http://<IP>:<PUERTO>/getJugadores
-- Devuelve los jugadores que han jugado una partida
http://<IP>:<PUERTO>/getJugadoresPorPartida?{partida=id_partida}
-- Inserta un jugador
http://<IP>:<PUERTO>/postJugador
-- Actualiza un jugador
http://<IP>:<PUERTO>/putJugador
-- Elimina un Jugador
http://<IP>:<PUERTO>/delJugador?id=id_jugador

-- PARTIDA
-- Devuelve todas las partidas
http://<IP>:<PUERTO>/getPartidas
-- Inserta una Partida
http://<IP>:<PUERTO>/postPartida
```

Código 19: Consumición del servicio REST 01.

```
-- RESPUESTAPREGUNTA

-- Devuelve la respuesta a una pregunta por un jugador para una partida en
concreto

http://<IP>:<PUERTO>/getRespuestaPreguntas?partida=id_partida&jugador=id_ju
gador&pregunta=num_pregunta


-- Inserta una respuesta a pregunta

http://<IP>:<PUERTO>/postRespuestaPregunta


-- RESPUESTAVOTACION

- Devuelve las respuestas a una votación por un jugador para una partida en
concreto

http://<IP>:<PUERTO>/getRespuestaVotaciones?partida=id_partida&jugador=id_j
ugador


-- Inserta una respuesta a votación

http://<IP>:<PUERTO>/postRespuestaVotacion
```

Código 20: Consumición del servicio REST 02.

ANEXO C – JAVADOC DEL PROYECTO

org.rest.controladores

Class Summary

Class	Description
Application	Clase necesaria para que Spring-boot arranque el servicio REST.
EquipoController	Clase encargada de publicar las URL para obtener los equipos.
EquipoControllerTest	Test EquipoController
EstadisticaEquipoControllerTest	Test EstadisticaEquipoController
EstadisticaGrupoControllerTest	Test EstadisticaGrupoController
EstadisticaJugadorControllerTest	Test EstadisticaJugadorController
EstadisticasEquipoController	No utilizada.
EstadisticasGrupoController	No utilizada.
EstadisticasJugadorController	No utilizada.
GrupoController	Clase encargada de publicar las URL para obtener los grupos.
GrupoControllerTest	Test GrupoController
JugadorController	Clase encargada de publicar las URL para obtener los jugadores.
JugadorControllerTest	Test JugadorController
PartidaController	Clase encargada de publicar las URL para obtener las partidas.
PartidaControllerTest	Test PartidaController
RespuestaPreguntaController	Clase encargada de publicar las URL para obtener las respuestas a pregunta.
RespuestaPreguntaControllerTest	Test RespuestaPreguntaController
RespuestaVotacionController	Clase encargada de publicar las URL para obtener datos acerca de las respuesta a votación.
RespuestaVotacionControllerTest	Test RespuestaVotacionController

Class PartidaController

- java.lang.Object
  - org.rest.controladores.PartidaController

```
@RestController
public class PartidaController
extends java.lang.Object
```

Clase encargada de publicar las URL para obtener las partidas.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Detail

- **PartidaController**
  ```
  public PartidaController()
  ```

Method Detail

- **getPartidas**
  ```
  @RequestMapping(value="/getPartidas", method=GET)
  public java.util.List<Partida> getPartidas()
  ```

- **createPartida**
  ```
  @RequestMapping(value="/postPartida", method=POST)
  public Partida createPartida(@RequestBody Partida partida)
  ```

Class RespuestaPreguntaController

- java.lang.Object
  - org.rest.controladores.RespuestaPreguntaController

```
@RestController
public class RespuestaPreguntaController
extends java.lang.Object
```

Clase encargada de publicar las URL para obtener las respuestas a pregunta.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Detail

- **RespuestaPreguntaController**
  `public RespuestaPreguntaController()`

Method Detail

- **getRespuestaPreguntas**

```
@RequestMapping(value="/getRespuestaPreguntas", method=GET)
public java.util.List<RespuestaPregunta> getRespuestaPreguntas(@RequestParam(valu
e="partida") int partida,  @RequestParam(value="jugador") int jugador,
@RequestParam(value="pregunta") int pregunta)
```

- **createRespuestaPregunta**

```
@RequestMapping(value="/postRespuestaPregunta", method=POST)
public RespuestaPregunta createRespuestaPregunta(@RequestBody
RespuestaPregunta respuestapregunta)
```

Class RespuestaVotacionController

- java.lang.Object
  - org.rest.controladores.RespuestaVotacionController

```
@RestController
public class RespuestaVotacionController
extends java.lang.Object
```

Clase encargada de publicar las URL para obtener datos acerca de las respuesta a votación.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Detail

- **RespuestaVotacionController**
  `public RespuestaVotacionController()`

Method Detail

- **getRespuestaVotaciones**
  ```
  @RequestMapping(value="/getRespuestaVotaciones", method=GET)
  public java.util.List<RespuestaVotacion> getRespuestaVotaciones(@RequestParam(valu
  e="partida") int partida,  @RequestParam(value="jugador") int jugador)
  ```

- **createRespuestaVotacion**
  ```
  @RequestMapping(value="/postRespuestaVotacion", method=POST)
  public RespuestaVotacion createRespuestaVotacion(@RequestBody
  RespuestaVotacion respuestavotacion)
  ```

Class Application

- java.lang.Object
  - org.rest.controladores.Application

```
@SpringBootApplication
public class Application
extends java.lang.Object
```

Clase necesaria para que Spring-boot arranque el servicio REST.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Detail

- **Application**
  ```
  public Application()
  ```

Method Detail

- **main**
  ```
  public static void main(java.lang.String[] args)
  ```

Class EquipoController

- java.lang.Object
  - org.rest.controladores.EquipoController

```
@RestController
public class EquipoController
extends java.lang.Object
```

Clase encargada de publicar las URL para obtener los equipos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Detail

- **EquipoController**
  ```
  public EquipoController()
  ```

Method Detail

- **getEquipos**
  ```
  @RequestMapping(value="/getEquipos", method=GET)
  public java.util.List<Equipo> getEquipos()
  ```

- **createEquipo**
  ```
  @RequestMapping(value="/postEquipo", method=POST)
  public Equipo createEquipo(@RequestBody Equipo equipo)
  ```

Class GrupoController

- java.lang.Object
  - org.rest.controladores.GrupoController

```
@RestController
public class GrupoController
extends java.lang.Object
```

Clase encargada de publicar las URL para obtener los grupos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Detail

- **GrupoController**
  `public GrupoController()`

Method Detail

- **getGrupos**
  ```
  @RequestMapping(value="/getGrupos", method=GET)
  public java.util.List<Grupo> getGrupos()
  ```

- **createGrupo**
  ```
  @RequestMapping(value="/postGrupo", method=POST)
  public Grupo createGrupo(@RequestBody Grupo grupo)
  ```

Class JugadorController

- java.lang.Object
  - org.rest.controladores.JugadorController

```
@RestController
public class JugadorController
extends java.lang.Object
```

Clase encargada de publicar las URL para obtener los jugadores.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Detail

- **JugadorController**
  `public JugadorController()`

Method Detail

- **getJugadores**

```
@RequestMapping(value="/getJugadores", method=GET)
public java.util.List<Jugador> getJugadores()
```

- **getJugadoresPorPartida**

```
@RequestMapping(value="/getJugadoresPorPartida", method=GET)
public java.util.List<Jugador> getJugadoresPorPartida(@RequestParam(value="par
tida") int partida)
```

- **createJugador**

```
@RequestMapping(value="/postJugador", method=POST)
public Jugador createJugador(@RequestBody Jugador jug)
```

- **updateJugador**

```
@RequestMapping(value="/putJugador", method=PUT)
public Jugador updateJugador(@RequestBody Jugador jug)
```

- **deleteJugador**

```
@RequestMapping(value="/delJugador/{id}", method=DELETE)
public org.springframework.http.ResponseEntity<java.lang.String> deleteJugador
(@PathVariable(value="id") int id)
```

org.rest.dao

Interface Summary

Interface	Description
EquipoDAO	Interfaz para el acceso de objetos Equipo a base de datos.
EstadisticasEquipoDAO	No utilizada.
EstadisticasGrupoDAO	No utilizada.
EstadisticasJugadorDAO	No utilizada.
GrupoDAO	
JugadorDAO	Interfaz para el acceso de objetos Jugador a base de datos.
PartidaDAO	Interfaz para el acceso de objetos Partida a base de datos.
RespuestaPreguntaDAO	Interfaz para el acceso de objetos Respuesta a Pregunta a base de datos.
RespuestaVotacionDAO	Interfaz para el acceso de objetos Respuesta a Votacion a base de datos.

Class Summary

Class	Description
EquipoDAOImpl	Implementación de la interfaz de acceso a objetos Equipo en base de datos.
EquipoDAOTest	Test EquipoDAO
EstadisticasEquipoDAOImpl	No utilizada.
EstadisticasEquipoDAOTest	No utilizada
EstadisticasGrupoDAOImpl	No utilizada.
EstadisticasGrupoDAOTest	No utilizada
EstadisticasJugadorDAOImpl	No utilizada.
EstadisticasJugadorDAOTest	No utilizada
GrupoDAOImpl	Implementación de la interfaz de acceso a objetos Grupo en base de datos.
GrupoDAOTest	Test GrupoDAO
JugadorDAOImpl	Implementación de la interfaz de acceso a objetos Jugador en base de datos.
JugadorDAOTest	Test JugadorDAO
PartidaDAOImpl	Implementación de la interfaz de acceso a objetos Partida en base de datos.
PartidaDAOTest	Test PartidaDAO
RespuestaPreguntaDAOImpl	Implementación de la interfaz de acceso a objetos Resouesta a Pregunta en base de datos.
RespuestaPreguntaDAOTest	Test RespuestaPreguntaDAO
RespuestaVotacionDAOImpl	Implementación de la interfaz de acceso a objetos Resouesta a Votacion en base de datos.
RespuestaVotacionDAOTest	Test RespuestaVotacionDAO

Interface EquipoDAO

```
public interface EquipoDAO
```

Interfaz para el acceso de objetos Equipo a base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Method Summary

Modifier and Type	Method and Description
void	createEquipo(Equipo equ)
Equipo	getEquipoById(int idEquipo)
java.util.List<Equipo>	getEquipoList()

Class EquipoDAOImpl

- java.lang.Object
  - org.rest.dao.EquipoDAOImpl

```
public class EquipoDAOImpl
extends java.lang.Object
implements EquipoDAO
```

Implementación de la interfaz de acceso a objetos Equipo en base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
EquipoDAOImpl()

Method Detail

- **getEquipoList**
  `public java.util.List<Equipo> getEquipoList()`

  Specified by:
  `getEquipoList` in interface `EquipoDAO`

- **getEquipoById**
  `public Equipo getEquipoById(int idEquipo)`

Specified by:
`getEquipoById` in interface `EquipoDAO`

- **createEquipo**
  `public void createEquipo(Equipo equ)`

  Specified by:
  `createEquipo` in interface `EquipoDAO`

Interface GrupoDAO

`public interface EquipoDAO`

Interfaz para el acceso de objetos Grupo a base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Method Summary

Modifier and Type	Method and Description
`void`	`createGrupo(Grupo gru)`
`Grupo`	`getGrupoById(int idGrupo)`
`java.util.List<Grupo>`	`getGrupoList()`

Class GrupoDAOImpl

- java.lang.Object
  - org.rest.dao.GrupoDAOImpl

```
public class GrupoDAOImpl
extends java.lang.Object
implements GrupoDAO
```

Implementación de la interfaz de acceso a objetos Grupo en base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`GrupoDAOImpl()`

Method Detail

- **getGrupoList**
  `public java.util.List<Grupo> getGrupoList()`

  Specified by:
  `getGrupoList` in interface `GrupoDAO`

- **getGrupoById**
  `public Grupo getGrupoById(int idGrupo)`

  Specified by:
  `getGrupoById` in interface `GrupoDAO`

- **createGrupo**
  `public void createGrupo(Grupo gru)`

  Specified by:
  `createGrupo` in interface `GrupoDAO`

Interface JugadorDAO

`public interface JugadorDAO`

Interfaz para el acceso de objetos Jugador a base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Method Summary

Modifier and Type	Method and Description
void	createJugador(Jugador jug)
void	deleteJugador(int id)
Jugador	getJugadorById(int idJugador)
java.util.List<Jugador>	getJugadoresByGrupo(int idGrupo)
java.util.List<Jugador>	getJugadoresByPartida(int idPartida)
java.util.List<Jugador>	getJugadorList()
void	updateJugador(Jugador jug)

Class JugadorDAOImpl

- java.lang.Object
  - org.rest.dao.JugadorDAOImpl

```
public class JugadorDAOImpl
extends java.lang.Object
implements JugadorDAO
```

Implementación de la interfaz de acceso a objetos Jugador en base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
JugadorDAOImpl()

Method Detail

- **getJugadorById**
  `public Jugador getJugadorById(int idJugador)`

  Specified by:
  `getJugadorById` in interface `JugadorDAO`

- **getJugadoresByGrupo**
  `public java.util.List<Jugador> getJugadoresByGrupo(int idGrupo)`

  Specified by:
  `getJugadoresByGrupo` in interface `JugadorDAO`

- **getJugadoresByPartida**
  `public java.util.List<Jugador> getJugadoresByPartida(int idPartida)`

  Specified by:
  `getJugadoresByPartida` in interface `JugadorDAO`

- getJugadorList
  `public java.util.List<Jugador> getJugadorList()`

  Specified by:
  `getJugadorList` in interface `JugadorDAO`

- **createJugador**
  `public void createJugador(Jugador jug)`

  Specified by:
  `createJugador` in interface `JugadorDAO`

- **updateJugador**
  `public void updateJugador(Jugador jug)`

  Specified by:
  `updateJugador` in interface `JugadorDAO`

- **deleteJugador**
  `public void deleteJugador(int id)`

  Specified by:
  `deleteJugador` in interface `JugadorDAO`

Interface PartidaDAO

```
public interface PartidaDAO
```

Interfaz para el acceso de objetos Partida a base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Method Summary

Modifier and Type	Method and Description
`void`	`createPartida(Partida par)`
`Partida`	`getPartidaById(int idPartida)`
`java.util.List<Partida>`	`getPartidaList()`

Class PartidaDAOImpl

- java.lang.Object
  - org.rest.dao.PartidaDAOImpl

```
public class PartidaDAOImpl
extends java.lang.Object
implements PartidaDAO
```

Implementación de la interfaz de acceso a objetos Partida en base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`PartidaDAOImpl()`

Method Detail

- **getPartidaList**
  `public java.util.List<Partida> getPartidaList()`

  Specified by:
  `getPartidaList` in interface `PartidaDAO`

- **getPartidaById**
  `public Partida getPartidaById(int idPartida)`

  Specified by:
  `getPartidaById` in interface `PartidaDAO`

- **createPartida**
  `public void createPartida(Partida par)`

  Specified by:
  `createPartida` in interface `PartidaDAO`

Interface RespuestaPreguntaDAO

`public interface RespuestaPreguntaDAO`

Interfaz para el acceso de objetos Respuesta a Pregunta a base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Method Summary

Modifier and Type	Method and Description
`void`	`createRespuestaPregunta(RespuestaPregunta res)`
`java.util.List<RespuestaPregunta>`	`getRespuestaPreguntaById(int idRespuestaPregunta)`
`java.util.List<RespuestaPregunta>`	`getRespuestaPreguntaByPartidaPregunta(int idPartida, int numPregunta)`
`java.util.List<RespuestaPregunta>`	`getRespuestaPreguntaByPartidaPreguntaJugador(int idPartida, int idJugador, int numPregunta)`

Class RespuestaPreguntaDAOImpl

- java.lang.Object
  - org.rest.dao.RespuestaPreguntaDAOImpl

```
public class RespuestaPreguntaDAOImpl
extends java.lang.Object
implements RespuestaPreguntaDAO
```

Implementación de la interfaz de acceso a objetos Resouesta a Pregunta en base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`RespuestaPreguntaDAOImpl()`

Method Detail

- **getRespuestaPreguntaByPartidaPregunta**
  `public java.util.List<RespuestaPregunta> getRespuestaPreguntaByPartidaPregunta (int idPartida, int numPregunta)`

  Specified by:
  `getRespuestaPreguntaByPartidaPregunta` in interface `RespuestaPreguntaDAO`

- **getRespuestaPreguntaByPartidaPreguntaJugador**
  `public java.util.List<RespuestaPregunta> getRespuestaPreguntaByPartidaPregunta Jugador(int idPartida, int idJugador, int numPregunta)`

  Specified by:
  `getRespuestaPreguntaByPartidaPreguntaJugador` in interface `RespuestaPreguntaDAO`

- **getRespuestaPreguntaById**
  `public java.util.List<RespuestaPregunta> getRespuestaPreguntaById(int idRespue staPregunta)`

  Specified by:
  `getRespuestaPreguntaById` in interface `RespuestaPreguntaDAO`

- **createRespuestaPregunta**
  `public void createRespuestaPregunta(RespuestaPregunta respuestaPregunta)`

  Specified by:
  `createRespuestaPregunta` in interface `RespuestaPreguntaDAO`

Interface RespuestaVotacionDAO

`public interface RespuestaVotacionDAO`

Interfaz para el acceso de objetos Respuesta a Votacion a base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Method Summary

Modifier and Type	Method and Description
`void`	`createRespuestaVotacion(RespuestaVotacion res)`
`java.util.List<RespuestaVotacion>`	`getRespuestaPreguntaById(int idRespuestaPregunta)`
`java.util.List<RespuestaVotacion>`	`getRespuestaVotacionByPartidaJugador(int idPartida, int idJugador)`

Class RespuestaVotacionDAOImpl

- java.lang.Object
  - org.rest.dao.RespuestaVotacionDAOImpl

```
public class RespuestaVotacionDAOImpl
extends java.lang.Object
```

Implementación de la interfaz de acceso a objetos Resouesta a Votacion en base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
RespuestaVotacionDAOImpl()

Method Detail

- **getRespuestaVotacionByPartidaJugador**
  `public java.util.List<RespuestaVotacion> getRespuestaVotacionByPartidaJugador(int idPartida, int idJugador)`

- **getRespuestaVotacionById**
  `public java.util.List<RespuestaVotacion> getRespuestaVotacionById(int idRespuestaVotacion)`

- **createRespuestaVotacion**
  `public void createRespuestaVotacion(RespuestaVotacion respuestaVotacion)`

org.rest.entidades

Class Summary

Class	Description
Equipo	Entidad Equipo
EstadisticasEquipo	No utilizada
EstadisticasGrupo	No utilizada
EstadisticasJugador	No utilizada
Grupo	Entidad Grupo
Jugador	Entidad Jugador
Partida	Entidad Partida
RespuestaPregunta	Entidad RespuestaPregunta
RespuestaVotacion	Entidad RespuestaVotacion

Class Partida

- java.lang.Object
  - org.rest.entidades.Partida

```
@Entity
public class Partida
extends java.lang.Object
```

Entidad Partida

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`Partida()`

Method Summary

Modifier and Type	Method and Description
`java.lang.String`	`getEstado()`
`double`	`getFactorcorreccion()`
`java.util.Date`	`getFechacreacion()`
`java.util.Date`	`getFechafinalizacion()`
`int`	`getId()`
`int`	`getNumpreguntas()`
`void`	`setEstado(java.lang.String estado)`
`void`	`setFactorcorreccion(double factorcorreccion)`
`void`	`setFechacreacion(java.util.Date fechacreacion)`
`void`	`setFechafinalizacion(java.util.Date fechafinalizacion)`
`void`	`setId(int id)`
`void`	`setNumpreguntas(int numpreguntas)`

Class Equipo

- java.lang.Object
  - org.rest.entidades.Equipo

```
@Entity
public class Equipo
extends java.lang.Object
```

Entidad Equipo

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`Equipo()`
`Equipo(java.lang.String descripcion, int numgrupos, int numjugadores)`

Method Summary

Modifier and Type	Method and Description
`java.lang.String`	`getDescripcion()`
`int`	`getId()`
`int`	`getNumgrupos()`
`int`	`getNumjugadores()`
`int`	`getPartida()`
`void`	`setDescripcion(java.lang.String descripcion)`
`void`	`setId(int id)`
`void`	`setNumgrupos(int numgrupos)`
`void`	`setNumjugadores(int numjugadores)`
`void`	`setPartida(int partida)`

Class Grupo

- java.lang.Object
  - org.rest.entidades.Grupo

```
@Entity
public class Grupo
extends java.lang.Object
```

Entidad Grupo

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
Grupo()

Method Summary

Modifier and Type	Method and Description
java.lang.String	getDescripcion()
int	getEquipo()
int	getId()
int	getNumjugadores()
int	getPartida()
void	setDescripcion(java.lang.String descripcion)
void	setEquipo(int equipo)
void	setId(int id)
void	setNumjugadores(int numjugadores)
void	setPartida(int partida)

Class Jugador

- java.lang.Object
  - org.rest.entidades.Jugador

```
@Entity
public class Jugador
extends java.lang.Object
```

Entidad Jugador

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
Jugador()

Method Summary

Modifier and Type	Method and Description
java.util.Date	getFecharegistro()
int	getGrupo()
int	getId()
java.lang.String	getNick()
int	getPartida()
void	setFecharegistro(java.util.Date fecharegistro)
void	setGrupo(int grupo)
void	setId(int id)
void	setNick(java.lang.String nick)
void	setPartida(int partida)

Class RespuestaPregunta

- java.lang.Object
  - org.rest.entidades.RespuestaPregunta

```
@Entity
public class RespuestaPregunta
extends java.lang.Object
```

Entidad RespuestaPregunta

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
RespuestaPregunta()

Method Summary

Modifier and Type	Method and Description
int	getIdJugador()
int	getIdPartida()
int	getNumeroPregunta()
java.lang.String	getOpcionRespuesta()
java.sql.Time	getTiempoRespuesta()
boolean	isCorrecta()
void	setCorrecta(boolean correcta)
void	setIdJugador(int idJugador)
void	setIdPartida(int idPartida)
void	setNumeroPregunta(int numeroPregunta)
void	setOpcionRespuesta(java.lang.String opcionRespuesta)
void	setTiempoRespuesta(java.sql.Time tiempoRespuesta)

Class RespuestaVotacion

- java.lang.Object
  - org.rest.entidades.RespuestaVotacion

```
@Entity
public class RespuestaVotacion
extends java.lang.Object
```

Entidad ResouestaVotacion

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
RespuestaVotacion()

Method Summary

Modifier and Type	Method and Description
int	getId_jugador()
int	getId_partida()
int	getId()
java.lang.String	getOpcion_respuesta()

java.sql.Time	getTiempo_respuesta()
void	setId_jugador(int id_jugador)
void	setId_partida(int id_partida)
void	setId(int id)
void	setOpcion_respuesta(java.lang.String opcion_respuesta)
void	setTiempo_respuesta(java.sql.Time tiempo_respuesta)

org.servidor.controlador

Class Summary

Class	Description
Cronometro	Esta clase controla el tiempo restante para cada pregunta o votacion, envía un timestamp a los jugadores y al rofesor con dicho tiempo, además de controlar la parada o activación del tiempo.
WebsocketController	Esta clase es el core del servidor Websocket, en dicha clase se almacena toda la información de sesiones y es la controladora del flujo de mensajes entre los jugadores y el profesor.

Class WebsocketController

- java.lang.Object
  - org.servidor.controlador.WebsocketController
- **All Implemented Interfaces:**
- java.io.Serializable

```
@ApplicationScoped
public class WebsocketController
extends java.lang.Object
implements java.io.Serializable
```

Esta clase es el core del servidor Websocket, en dicha clase se almacena toda la información de sesiones y es la controladora del flujo de mensajes entre los jugadores y el profesor. También es la encargada de solicitar la llama al servicio REST para almacenar los resultados en la base de datos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`WebsocketController()`

Method Detail

- **iniciaSesion**
  `public void iniciaSesion(javax.websocket.Session session)`

  Función que se ejecuta cuando un cliente se conecta al servidor denotado por la ruta de la anotación @ServerEndPoint

  Parameters:
  `session` - sesion del cliente que se conecta al servidor

- **onMessage**
  ```
  public void onMessage(java.lang.String mensaje,
                        javax.websocket.Session sesion)
  ```

  Función que se ejecuta cuando un cliente lanza un mensaje a través del canal bidireccional establecido.

  Parameters:
  `mensaje` - texto recibido en formato JSON

  `sesion` - cliente que envia el mensaje

- **finConexion**
  `public void finConexion(javax.websocket.Session session)`

  Funcion que se ejecuta cuando un cliente cierra su sesión, voluntariamente o no, con el servidor.

  Parameters:
  `session` - cliente que aborta conexión

- **procesarEvento**
  ```
  public void procesarEvento(javax.websocket.Session sesion,
  java.lang.String mensaje) throws org.json.JSONException
  ```

  Función que procesará los eventos que llegan al servidor, y en función del tipo de evento se llamarán a unas funciones u otras.

  Parameters:
  `sesion` - cliente que envía el mensaje

`mensaje` - texto enviado en formato JSON

Throws:
`org.json.JSONException`

- **borrarJugadorPorDesconexion**
  `public void borrarJugadorPorDesconexion(javax.websocket.Session sesion) throws org.json.JSONException`

  Funcion encargada de localizar el jugador que se acaba de desconectar y eliminarlo de la partida.

  Parameters:
  `sesion` - sesion del jugador

  Throws:
  `org.json.JSONException`

- **esSesionProfesor**
  `public boolean esSesionProfesor(javax.websocket.Session sesion)`

  Me indica si la sesion que se ha finalizado es de un profesor.

  Parameters:
  `sesion` - sesion que ha abortado conexion

- **registrarPartida**
  `public void registrarPartida(javax.websocket.Session sesion, int idPartida) throws org.json.JSONException`

  Función encargada de dejar registrada la partida y asociarla a la sesión del profesor. Es aquí donde se hace la llamada al servicio de Ana para obtener todos los datos de configuración creados por el profesor y que son necesarios de pasar al juego.
  IMPORTANTE: como aun no está implementado, se hará de forma tosca un ejemplo para poder probar el resto del programa.

  Parameters:
  `sesion` - sesion del profesor que registra la partida

  `idPartida` - identificador de la partida

  Throws:
  `org.json.JSONException`

- **enviarGruposEquipos**
  `public void enviarGruposEquipos(javax.websocket.Session sesion, int idPartida) throws org.json.JSONException`

Funcion que responde ante la necesidad de un jugador de conocer cuales son los grupos y equipos para su partida solicitada.

Parameters:

`sesion` - sesion del profesor que registra la partida

`idPartida` - identificador de la partida

Throws:

`org.json.JSONException`

- **buscaPartidaGlobalPorIdPartida**

```
public PartidaGlobal buscaPartidaGlobalPorIdPartida(int idPartida)
```

Funcion que devuelve una partida global por el id de partida.

Parameters:

`idPartida` - identificador de la partida

Returns:

objeto partida global con toda la informacion de la partida

- **obtenerListaDescripcionGrupos**

```
public java.util.List<java.lang.String> obtenerListaDescripcionGrupos(java.util.List<Grupo> listaGrupos)
```

Funcion que recibe una lista de objetos Grupo y me devuelve una lista con las descripciones.

Parameters:

`listaGrupos` - listado de grupos

Returns:

listado de nombre de grupos

- **obtenerListaDescripcionEquipos**

```
public java.util.List<java.lang.String> obtenerListaDescripcionEquipos(java.util.List<Equipo> listaEquipos)
```

Funcion que recibe una lista de objetos Equipo y me devuelve una lista con las descripciones.

Parameters:

`listaGrupos` - listado de equipos

Returns:

listado de nombres de equipo

- **validarJugador**
  `public boolean validarJugador(javax.websocket.Session sesion, java.lang.String nick, int idPartida) throws org.json.JSONException`

  Funcion que se encarga de comprobar que el jugador que accede a la partida use un nick que no esté repetido, en caso contrario se le avisa de que no puede acceder con dicho nick.

  Parameters:
  `sesion` - sesion del jugador

  `nick` - nombre del jugador

  `idPartida` - partida a la que pertenece

  Returns:
  nick validado si/no

  Throws:
  `org.json.JSONException`

- **registrarJugador**
  `public void registrarJugador(javax.websocket.Session sesion, org.json.JSONObject objJSON) throws org.json.JSONException`

  Funcion que se encarga de registrar un jugador en una partida, para ello lo incluye en la lista de jugadores de una partida global.

  Parameters:
  `sesion` - sesion del jugador

  `objJSON` - json con los datos del jugador

  Throws:
  `org.json.JSONException`

- **comprobarGrupoLleno**
  `public boolean comprobarGrupoLleno(int idPartida, java.lang.String grupo)`

  Comprueba que el grupo al que se accede no se encuentra ya lleno.

  Parameters:
  `idPartida` - identificador de partida

  `grupo` - grupo de acceso

  Returns:
  si/no

- **accionJugadorModificado**
  `public void accionJugadorModificado(java.lang.String nick,  int idPartida, java.lang.String grupo, java.lang.String equipo) throws org.json.JSONException`

  Esta funcion, ejecutada desde la vista del profesor, se encarga de actualizar un jugador previamente modificado por el profesor y enviar al navegador del jugador la orden de establecer el nuevo valor impuesto por el profesor en el terminal del jugador.

  Parameters:
  `nick` - jugador

  `idPartida` - identificador partida

  `grupo` - grupo del jugador

  `equipo` - equipo del jugador

  Throws:
  `org.json.JSONException`

- **eliminarJugador**
  `public void eliminarJugador(java.lang.String nick, int idPartida) throws org.json.JSONException`

  Funcion que se ejecuta cuando un jugador ha sido borrado de la lista de registro por el profesor. Esta función elimina el jugador del servidor y envía un mensaje al jugador para que se vuelva a registrar.

  Parameters:
  `nick` - jugador

  `idPartida` - idetificador de partida

  Throws:
  `org.json.JSONException`

- **confirmarPartida**
  `public void confirmarPartida(int idPartida) throws org.json.JSONException`

  Funcion que rellena de jugadores y grupos el objeto de datos iniciales y lo envia al javascript para poder ser enviado al juego.

  Parameters:
  `idPartida` - idetificador de partida

  Throws:
  `org.json.JSONException`

- **detenerTiempo**
  `public void detenerTiempo(int idPartida)`

Funcion encargada de obtener el cronometro de la partida y detenerlo.

Parameters:

`idPartida` - identificador de partida

- **activarTiempo**

```
public void activarTiempo(int idPartida)
```

Funcion encargada de obtener el cronometro de la partida y detenerlo.

Parameters:

`idPartida` - identificador de partida

- **procesarPregunta**

```
public void procesarPregunta(org.json.JSONObject objJSON) throws
org.json.JSONException
```

Funcion que se encarga de reenviar la pregunta a todos los jugadores que participan en la partida, además, deja preparado una serie de datos necesarios para la correccion.

Parameters:

`objJSON` - objeto con los datos de la pregunta

Throws:

`org.json.JSONException`

- **procesarVotacion**

```
public void procesarVotacion(org.json.JSONObject objJSON) throws
org.json.JSONException
```

Funcion que se encarga de reenviar la votacion a todos los jugadores que participan en la partida, además, deja preparado una serie de datos necesarios para la correccion.

Parameters:

`objJSON` - objeto con los datos de la votacion

Throws:

`org.json.JSONException`

- **procesarRespuestaPregunta**

```
public void procesarRespuestaPregunta(org.json.JSONObject objJSON) throws
org.json.JSONException
```

Este metodo se encarga de procesar la respuesta a una pregunta por parte de los jugadores. Almacena el resultado de la respuesta en el listado de participaciones y comprueba que todos hayan respondido.

Parameters:

`objJSON` - objeto con los datos de la respuesta a pregunta

Throws:

`org.json.JSONException`

- **procesarRespuestaVotacion**

```
public void procesarRespuestaVotacion(org.json.JSONObject objJSON) throws
org.json.JSONException
```

Este metodo se encarga de procesar la respuesta a una votacion por parte de los jugadores. Almacena el resultado de la respuesta en el listado de participaciones y comprueba que todos hayan respondido.

Parameters:

`objJSON` - objeto con los datos de la respuesta a votacion

Throws:

`org.json.JSONException`

- **tiempoAgotado**

```
public void tiempoAgotado(int idPartida, boolean esPregunta)
```

Funcion que es ejecutada por el Cronometro cuando este finaliza el tiempo y construye la respuesta para el juego.

Parameters:

`idPartida` - identificador partida

`esPregunta` - definir si es pregunta o votacion

- **construirObjetoRespuestaPreguntaAlJuego**

```
public void construirObjetoRespuestaPreguntaAlJuego(PartidaGlobal partidaglobal,
Pregunta pregunta_actual) throws org.json.JSONException
```

Una vez que todos los jugadores hayan dado su respuesta, bien sea por ellos mismos, como de forma automatica tras la finalización del tiempo, esta funcion se encarga de construir el objeto de respuesta a la pregunta que debe ser enviado el juego.

Parameters:

`partidaglobal` - partida

`pregunta_actual` - pregunta

Throws:

`org.json.JSONException`

- **construirObjetoRespuestaVotacionAlJuego**

```
public void construirObjetoRespuestaVotacionAlJuego(PartidaGlobal partidaglobal,
Votacion votacion_actual) throws org.json.JSONException
```

Una vez que todos los jugadores hayan dado su respuesta, bien sea por ellos mismos, como de forma automatica tras la finalización del tiempo, esta funcion se encarga de construir el objeto de respuesta a la votacion que debe ser enviado el juego.

Parameters:

`partidaglobal` - partida

`votacion_actual` - votacion

Throws:

`org.json.JSONException`

- **almacenarResultadoPartida**

```
public void almacenarResultadoPartida(org.json.JSONObject objJSON) throws
org.json.JSONException
```

Esta función se encarga de procesar el JSON recibido por parte del juego con los resultados de la partida y debe almacenar los datos en un objeto de resultados para posteriormente enviarlo a ANA.

Parameters:

`objJSON` - objeto con los resultados de la aprtida

Throws:

`org.json.JSONException`

- **procesarAlmacenamientoDatosPartidaBaseDatos**

```
public void procesarAlmacenamientoDatosPartidaBaseDatos(PartidaGlobal partidagloba
l)
```

Esta función se encarga de obtener los datos en caliente de la partida finalizada y hacer llamadas al servicio REST para almacenar los datos en la base de datos del servicio de comunicacion WebSockets.

Parameters:

`partidaglobal` - partida

- **persistirPartida**

```
public Partida persistirPartida(Partida par)
```

Funcion que crea el objeto RestTemplate correspondiente y lanza la peticion de persistir la partida.

Parameters:

`par` - partida a persistir

Returns:

partida persistida

- **persistirEquipo**
  `public Equipo persistirEquipo(Equipo equ)`

  Funcion que crea el objeto RestTemplate correspondiente y lanza la peticion de persistir el equipo.

  Parameters:
  `equ` - equipo a persistir

  Returns:
  equipo persistido

- **persistirGrupo**
  `public Grupo persistirGrupo(Grupo gru)`

  Funcion que crea el objeto RestTemplate correspondiente y lanza la peticion de persistir el grupo.

  Parameters:
  `gru` - grupo a persistir

  Returns:
  grupo persistido

- **persistirJugador**
  `public Jugador persistirJugador(Jugador jug)`

  Funcion que crea el objeto RestTemplate correspondiente y lanza la peticion de persistir el jugador.

  Parameters:
  `jug` - jugador a persistir

  Returns:
  jugador persistido

- **persistirRespuestaPregunta**
  `public RespuestaPregunta persistirRespuestaPregunta(RespuestaPregunta res)`

  Funcion que crea el objeto RestTemplate correspondiente y lanza la peticion de persistir la RespuestaPregunta.

  Parameters:
  `res` - respuesta a pregunta a persistir

  Returns:
  respuesta a pregunta persistida

- **persistirRespuestaVotacion**
  `public RespuestaVotacion persistirRespuestaVotacion(RespuestaVotacion res)`

Funcion que crea el objeto RestTemplate correspondiente y lanza la peticion de persistir la RespuestaPregunta.

Parameters:
`res` - respuesta a votacion a persistir

Returns:
respuesta a votacion persistida

- **limpiaAcentos**
  `public static java.lang.String limpiaAcentos(java.lang.String texto)`

  Herramienta para quitar acentos de las palabras.

  Parameters:
  `texto` - a quitar acentos

  Returns:
  texto sin acentos

Class Cronometro

- java.lang.Object
  - java.lang.Thread
    - org.servidor.controlador.Cronometro
- **All Implemented Interfaces:**
- java.lang.Runnable

```
public class Cronometro
extends java.lang.Thread
```

Esta clase controla el tiempo restante para cada pregunta o votacion, envía un timestamp a los jugadores y al rofesor con dicho tiempo, además de controlar la parada o activación del tiempo.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`Cronometro(WebsocketController websocket)` Constructor de la clase.

Method Detail

- **getSegundos**
  `public int getSegundos()`

- **setSegundos**
  `public void setSegundos(int segundos)`

- **getSesionProfesor**
  `public javax.websocket.Session getSesionProfesor()`

- **getParticipaciones**
  `public java.util.List<Participacion> getParticipaciones()`

- **setSesionProfesor**
  `public void setSesionProfesor(javax.websocket.Session sesionProfesor)`

- **setParticipaciones**
  `public void setParticipaciones(java.util.List<Participacion> participaciones)`

- **isParado**
  `public boolean isParado()`

- **setParado**
  `public void setParado(boolean parado)`

- **getPartida**
  `public int getPartida()`

- **setPartida**
  `public void setPartida(int partida)`

- **isPregunta**
  `public boolean isPregunta()`

- **setPregunta**
  `public void setPregunta(boolean pregunta)`

- **run**
  `public void run()`

  Metodo encargado de lanzar la ejecución del cronómetro y llevar el control.

  Specified by:
  `run` in interface `java.lang.Runnable`

  Overrides:
  `run` in class `java.lang.Thread`

org.servidor.vista

Class Summary

Class	Description
AgrupacionEquipos	Esta clase es utilizada para agrupar los jugadores por equipos, es necesaria para reordenar en la vista de registro los jugadores según la especificación.
VistaRegistroJugadores	Esta clase guarda toda la información que respalda la vista del registro de jugadores, seguimiento de preguntas/votaciones y estadisticas por pregunta/votacion.

Class VistaRegistroJugadores

- java.lang.Object
  - org.servidor.vista.VistaRegistroJugadores
- **All Implemented Interfaces:**
- java.io.Serializable

```
@Named
@SessionScoped
public class VistaRegistroJugadores
extends java.lang.Object
implements java.io.Serializable
```

Esta clase guarda toda la información que respalda la vista del registro de jugadores, seguimiento de preguntas/votaciones y estadisticas por pregunta/votacion.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
VistaRegistroJugadores()

Method Detail

- **init**
  ```
  @PostConstruct
  public void init()
  ```

- **getLista**
  ```
  public java.util.List<Participacion> getLista()
  ```

- **setLista**
  ```
  public void setLista(java.util.List<Participacion> lista)
  ```

- **getSeguimientoP**
  ```
  public java.util.List<SeguimientoRespuestasPreVot> getSeguimientoP()
  ```

- **getSeguimientoPSel**
  ```
  public SeguimientoRespuestasPreVot getSeguimientoPSel()
  ```

- **setSeguimientoP**
  ```
  public void setSeguimientoP(java.util.List<SeguimientoRespuestasPreVot> seguimient
  oP)
  ```

- **setSeguimientoPSel**
  ```
  public void setSeguimientoPSel(SeguimientoRespuestasPreVot seguimientoPSel)
  ```

- **getJugadoresConectados**
  ```
  public int getJugadoresConectados()
  ```

- **setJugadoresConectados**
  ```
  public void setJugadoresConectados(int jugadoresConectados)
  ```

- **getEstadisticaSel**
  ```
  public EstadisticaPorPregunta getEstadisticaSel()
  ```

- **setEstadisticaSel**
  ```
  public void setEstadisticaSel(EstadisticaPorPregunta estadisticaSel)
  ```

- **getEstadisticasCorrectas**
  ```
  public java.util.List<EstadisticaPorPregunta> getEstadisticasCorrectas()
  ```

- **getEstadisticasIncorrectas**
  ```
  public java.util.List<EstadisticaPorPregunta> getEstadisticasIncorrectas()
  ```

- **setEstadisticasCorrectas**
  ```
  public void setEstadisticasCorrectas(java.util.List<EstadisticaPorPregunta> estadi
  sticasCorrectas)
  ```

- **setEstadisticasIncorrectas**
  ```
  public void setEstadisticasIncorrectas(java.util.List<EstadisticaPorPregunta> esta
  disticasIncorrectas)
  ```

- **getPieModel1**
  ```
  public org.primefaces.model.chart.PieChartModel getPieModel1()
  ```

- **setPieModel1**
  ```
  public void setPieModel1(org.primefaces.model.chart.PieChartModel pieModel1)
  ```

- **getRespuestaCorrecta**
  ```
  public java.lang.String getRespuestaCorrecta()
  ```

- **getPorcentaje**
  ```
  public double getPorcentaje()
  ```

- **setRespuestaCorrecta**
  ```
  public void setRespuestaCorrecta(java.lang.String respuestaCorrecta)
  ```

- **setPorcentaje**
  ```
  public void setPorcentaje(double porcentaje)
  ```

- **getSeguimientoV**
  ```
  public java.util.List<SeguimientoRespuestasPreVot> getSeguimientoV()
  ```

- **getSeguimientoVSel**
  ```
  public SeguimientoRespuestasPreVot getSeguimientoVSel()
  ```

- **setSeguimientoV**
  ```
  public void setSeguimientoV(java.util.List<SeguimientoRespuestasPreVot> seguimient
  oV)
  ```

- **setSeguimientoVSel**
  ```
  public void setSeguimientoVSel(SeguimientoRespuestasPreVot seguimientoVSel)
  ```

- **getEstadisticasVotacion**
  ```
  public java.util.List<EstadisticasPorVotacion> getEstadisticasVotacion()
  ```

- **getEstadisticaVSel**
  ```
  public EstadisticasPorVotacion getEstadisticaVSel()
  ```

- **setEstadisticasVotacion**
  ```
  public void setEstadisticasVotacion(java.util.List<EstadisticasPorVotacion> estadi
  sticasVotacion)
  ```

- **setEstadisticaVSel**
  ```
  public void setEstadisticaVSel(EstadisticasPorVotacion estadisticaVSel)
  ```

- **getListaE**
  ```
  public java.util.List<AgrupacionEquipos> getListaE()
  ```

- **setListaE**
  ```
  public void setListaE(java.util.List<AgrupacionEquipos> listaE)
  ```

- **getSeleccion**
  ```
  public java.util.List<Participacion> getSeleccion()
  ```

- **setSeleccion**
  `public void setSeleccion(java.util.List<Participacion> seleccion)`

- **getNombres**
  `public java.util.List<java.lang.String> getNombres()`

- **setNombres**
  `public void setNombres(java.util.List<java.lang.String> nombres)`

- **getGrupoSeleccionado**
  `public java.lang.String getGrupoSeleccionado()`

- **setGrupoSeleccionado**
  `public void setGrupoSeleccionado(java.lang.String grupoSeleccionado)`

- **getEquipoSeleccionado**
  `public java.lang.String getEquipoSeleccionado()`

- **setEquipoSeleccionado**
  `public void setEquipoSeleccionado(java.lang.String equipoSeleccionado)`

- **getPartida**
  `public int getPartida()`

- **setPartida**
  `public void setPartida(int partida)`

- **getListadoEquipos**
  `public java.util.List<java.lang.String> getListadoEquipos()`

- **setListadoEquipos**
  `public void setListadoEquipos(java.util.List<java.lang.String> listadoEquipos)`

- **getListadoGrupos**
  `public java.util.List<java.lang.String> getListadoGrupos()`

- **setListadoGrupos**
  `public void setListadoGrupos(java.util.List<java.lang.String> listadoGrupos)`

- **isListadoCargado**
  `public boolean isListadoCargado()`

- **setListadoCargado**
  `public void setListadoCargado(boolean listadoCargado)`

- **getNumeroJugadores**
  `public java.lang.String getNumeroJugadores()`

  Devuelve el numero de jugadores conectados a la partida.

  Returns:
  Numero de jugadores conectados

- **eliminarTodosJugadores**
  `public void eliminarTodosJugadores() throws org.json.JSONException`

  Permite eliminar todos los jugadores registrados en la partida.

  Throws:
  `org.json.JSONException`

- **eliminarSeleccionados**
  `public void eliminarSeleccionados() throws org.json.JSONException`

  Permite eliminar todos los jugadores seleccionados en los checkboxes.

  Throws:
  `org.json.JSONException`

- **eliminarJugador**
  `public void eliminarJugador(Participacion jug) throws org.json.JSONException`

  Elimina un jugador en concreto.

  Parameters:
  `jug` - Jugador que queremos eliminar.

  Throws:
  `org.json.JSONException`

- **encuentraJugadorPorNick**
  `public Participacion encuentraJugadorPorNick(java.lang.String nick)`

Funcion que devuelve un jugador encontrado por su nick en la lista de jugadores registrados en la partida.

Parameters:

`nick` - Nombre del jugador que queremos localizar

Returns:

Jugador enncontrado

- **recargaPagina**

```
public void recargaPagina()
```

Función utilizada para recargar la vista desde el código del Bean.

- **nuevoJugadorRegistrado**

```
public void nuevoJugadorRegistrado()
```

Esta funcion obtendra los nuevos jugadores registrados y los cargara en la lista que se le muestra al profesor.

- **jugadorDesconectado**

```
public void jugadorDesconectado()
```

Funcion que se ejecuta cuando un jugador se desconecta y pierde la sesion, hay que sacarlo de la partida, solo si la partida está confirmada el jugador se mantendrá dentro.

- **reordenarAgrupacionesEquipos**

```
public void reordenarAgrupacionesEquipos()
```

Funcion encargada de realizar el paso de una lista desordenada de jugadores a un listado de agrupaciones de jugadores por equipos para ser mostrados de forma organizada en la vista.

- **confirmarPartida**

```
public void confirmarPartida() throws org.json.JSONException
```

Esta función confirma la partida, lo que significa que se dara comienzo al juego, para ello es necesario que se cargue el objeto de datos iniciales en el javascript que será el enviado al juego.

Throws:

`org.json.JSONException`

- **detenerTiempo**

```
public void detenerTiempo()
```

Funcion que detiene el cronometro de la pregunta/votacion para los jugadores dándoles el tiempo de respuesta considerado por el profesor.

- **activarTiempo**
  `public void activarTiempo()`

  Funcion que activa el cronometro de la pregunta/votacion para los jugadores dándoles el tiempo de respuesta considerado por el profesor.

- **getTotalRespuestasP**
  `public java.lang.String getTotalRespuestasP()`

  Devuelve el numero total de respuestas dadas por los jugadores a la pregunta, este método es usado en la página de seguimiento de respuestas.

  Returns:
  Número de respuestas

- **getTotalRespuestasV**
  `public java.lang.String getTotalRespuestasV()`

  Devuelve el numero total de respuestas dadas por los jugadores a la votacion, este método es usado en la página de seguimiento de respuestas.

  Returns:
  Número de respuestas

- **addMessage**
  `public void addMessage(java.lang.String texto)`

  Metodo que lanza un mensaje de tipo información en la pestaña del profesor.

  Parameters:
  `texto` - mensaje

- **addErrorMessage**
  `public void addErrorMessage(java.lang.String texto)`

  Metodo que lanza un mensaje de tipo error en la pestaña del profesor.

  Parameters:
  `texto` - mensaje

- **refrescarSeguimientoPregunta**
  `public void refrescarSeguimientoPregunta()`

  Funcion que obtiene del servidor los jugadores que estan respondiendo a la pregunta actual para refrescar su seguimiento.

- **refrescarSeguimientoVotacion**
  `public void refrescarSeguimientoVotacion()`

  Funcion que obtiene del servidor los jugadores que estan respondiendo a la votacion actual para refrescar su seguimiento.

- **refrescarEstadisticasPregunta**
  `public void refrescarEstadisticasPregunta()`

  Funcion que obtiene del servidor la estadistica por grupo de la pregunta que se está jugando en ese momento.

- **refrescarEstadisticasVotacion**
  `public void refrescarEstadisticasVotacion()`

  Funcion que obtiene del servidor la estadistica por grupo de la votacion que se esta jugando en ese momento.

- **findAgrupacionByEquipo**
  `public AgrupacionEquipos findAgrupacionByEquipo(java.lang.String equipo)`

  Funcion que se encarga de localizar en la lista de agrupaciones de la vista la agrupacion correspondiente a un determinado equipo.

  Parameters:
  `equipo` - nombre del equipo

  Returns:
  Agrupacion de dicho equipo

- **establecerSeleccion**
  `public void establecerSeleccion() throws org.json.JSONException`

  Establece el grupo y equipo de los jugadores seleccionados en funcion de la seleccion realizada en los desplegables.

  Throws:
  `org.json.JSONException`

- **edicionPorCelda**
  `public void edicionPorCelda(Participacion participacion) throws org.json.JSONException`

  Se encarga de modificar en el servidor el jugador que se ha modificado a partir de la edicion de celdas, en la vista lo hacen automaticamente las PrimeFaces.

  Parameters:
  `participacion` - jugador

Throws:
`org.json.JSONException`

- **numeroJugadoresPorGrupo**
  `public int numeroJugadoresPorGrupo(java.lang.String grupo)`

  Devuelve el numero de jugadores que contiene un grupo.
  Parameters:
  `grupo` - nombre del grupo

  Returns:
  Numero de jugadores

- **jugadoresConectados**
  `public java.lang.String jugadoresConectados()`

  Devuelve el numero de jugadores conectados a la partida.
  Returns:
  Jugadores en formato cadena

- **refrescar**
  `public void refrescar()`

  Funcion encargada de realizar un refresco de jugadores, primero limpia la lista de jugadores y despues hace un reordenamiento, necesario en caso de haber cambiado de equipo o grupos.

- **onEquipoChange**
  `public void onEquipoChange()`

  Detecta un cambio en la seleccion de equipos, al cambiar la seleccion de equipo debe mostrar en la seleccion de grupos los grupos pertenecientes a ese equipo para la comodidad del profesor.

- **getGruposParaEquipo**
  `public java.util.List<java.lang.String> getGruposParaEquipo(java.lang.String equipo)`

  Herramienta para tener los grupos pertenecientes a un equipo a la hora de editar desde celda.
  Parameters:
  `equipo` - nombre del equipo

  Returns:
  Lista de nombres de grupo

- **reset**
  `public void reset()`

  Si volvemos a crear una partida dentro de la misma sesion, es necesario un reset de los datos del bean, ya que es necesario que el bean sea de sesion pero tambien es necesario resetearlo para nuevas partidas. Es un caso especifico, no valdría usar otro Scope.

Class AgrupacionEquipos

- java.lang.Object
  - org.servidor.vista.AgrupacionEquipos

```
public class AgrupacionEquipos
extends java.lang.Object
```

Esta clase es utilizada para agrupar los jugadores por equipos, es necesaria para reordenar en la vista de registro los jugadores según la especificación. Este objeto contiene los datos mostrados en los fieldset de equipos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`AgrupacionEquipos()`

Method Detail

- **getEquipo**
  `public java.lang.String getEquipo()`

- **setEquipo**
  `public void setEquipo(java.lang.String equipo)`

- **getParticipaciones**
  `public java.util.List<Participacion> getParticipaciones()`

- **setParticipaciones**
  `public void setParticipaciones(java.util.List<Participacion> participaciones)`

- **añadirParticipacion**
  `public void añadirParticipacion(Participacion par)`

org.servidor.modelo

Class Summary

Class	Description
ConfiguracionPartida	Esta clase guarda datos de configuración generales para la partida.
DatosIniciales	Esta clase gaurda la información referente a la configuración que el profesor ha establecido para la partida.
EstadisticaPorPregunta	Esta clase guarda información referente a las respuestas que se han dado para una pregunta en concreto y poder mostrar una gráfica con dicha información.
EstadisticasPorVotacion	Esta clase guarda información referente a las respuestas que se han dado para una votación en concreto y poder mostrar una gráfica con dicha información.
Participacion	Esta clase guarda la información de una solicitud de registro por parte de un jugador, en ella tambien se almacenan las respuesta que ha dado durante la partida.
PartidaGlobal	Esta clase guarda toda la información referente a una partida: sesion del profesor, jugadores conectados, pregunta sy votaciones contenidas, cronometro, configuraciones e informacion sobre grupos y equipos.
Pregunta	Esta clase guarda la información sobre una pregunta.
ResultadoPartida	Esta clase alamcena los resultados de una partida.
ResultadoPartidaGrupo	Esta clase almacena los resultados de una partida por un grupo determinado.
SeguimientoRespuestasPreVot	Esta clase guarda la información de los jugadores que han respondido una votación o pregunta agrupados por grupo para llevar un seguimiento en tiempo real.
Votacion	Esta clase guarda la información de una votación registrada en una partida.

Class Participacion

- java.lang.Object
  - org.servidor.modelo.Participacion

```
public class Participacion
extends java.lang.Object
```

Esta clase guarda la información de una solicitud de registro por parte de un jugador, en ella tambien se almacenan las respuesta que ha dado durante la partida.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`Participacion()`

Method Detail

- **getNick**
  `public java.lang.String getNick()`
- **getIdPartida**
  `public int getIdPartida()`
- **getGrupo**
  `public java.lang.String getGrupo()`
- **getEquipo**
  `public java.lang.String getEquipo()`
- **getFechaRegistro**
  `public java.util.Date getFechaRegistro()`
- **setNick**
  `public void setNick(java.lang.String nick)`

- **setIdPartida**
  `public void setIdPartida(int idPartida)`

- **setGrupo**
  `public void setGrupo(java.lang.String grupo)`

- **setEquipo**
  `public void setEquipo(java.lang.String equipo)`

- **setFechaRegistro**
  `public void setFechaRegistro(java.util.Date fechaRegistro)`

- **getSesion**
  `public javax.websocket.Session getSesion()`

- **setSesion**
  `public void setSesion(javax.websocket.Session sesion)`

- **getRespuestas**
  `public java.util.List<RespuestaPregunta> getRespuestas()`

- **setRespuestas**
  `public void setRespuestas(java.util.List<RespuestaPregunta> respuestas)`

- **getVotaciones**
  `public java.util.List<RespuestaVotacion> getVotaciones()`

- **setVotaciones**
  `public void setVotaciones(java.util.List<RespuestaVotacion> votaciones)`

- **añadirRespuestaPregunta**
  `public void añadirRespuestaPregunta(RespuestaPregunta rp)`

- **añadirRespuestaVotacion**
  `public void añadirRespuestaVotacion(RespuestaVotacion rv)`

- **obtenerRPPorNumeroPregunta**
  `public RespuestaPregunta obtenerRPPorNumeroPregunta(int numeroPregunta)`

  Se encarga de obtener la respuesta a una pregunta en funcion del numero de la pregunta dentro de la partida.

  Parameters:
  `numeroPregunta` - numero de la pregunta

  Returns:
  respuesta dada por el jugador

- **toString**
  `public java.lang.String toString()`

  Overrides:
  `toString` in class `java.lang.Object`

Class PartidaGlobal

- java.lang.Object
  - org.servidor.modelo.PartidaGlobal

```
public class PartidaGlobal
extends java.lang.Object
```

Esta clase guarda toda la información referente a una partida: sesion del profesor, jugadores conectados, pregunta sy votaciones contenidas, cronometro, configuraciones e informacion sobre grupos y equipos.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`PartidaGlobal()`

Method Detail

- **getSesionProfesor**
  ```
  public javax.websocket.Session getSesionProfesor()
  ```

- **getPartida**
  ```
  public Partida getPartida()
  ```

- **getCronometro**
  ```
  public Cronometro getCronometro()
  ```

- **setCronometro**
  ```
  public void setCronometro(Cronometro cronometro)
  ```

- **getMaximo_jugadores_grupo**
  ```
  public int getMaximo_jugadores_grupo()
  ```

- **isReparto_aleatorio**
  ```
  public boolean isReparto_aleatorio()
  ```

- **setMaximo_jugadores_grupo**
  ```
  public void setMaximo_jugadores_grupo(int maximo_jugadores_grupo)
  ```

- **setReparto_aleatorio**
  ```
  public void setReparto_aleatorio(boolean reparto_aleatorio)
  ```

- **getConfiguracion**
  ```
  public ConfiguracionPartida getConfiguracion()
  ```

- **getJugadores**
  ```
  public java.util.List<Jugador> getJugadores()
  ```

- **getGrupos**
  ```
  public java.util.List<Grupo> getGrupos()
  ```

- **getEquipos**
  ```
  public java.util.List<Equipo> getEquipos()
  ```

- **setSesionProfesor**
  ```
  public void setSesionProfesor(javax.websocket.Session sesionProfesor)
  ```

- **setPartida**
  ```
  public void setPartida(Partida partida)
  ```

- **getIndice_de_reparto**
  ```
  public int getIndice_de_reparto()
  ```

- **setIndice_de_reparto**
  ```
  public void setIndice_de_reparto(int indice_de_reparto)
  ```

- **setConfiguracion**
  ```
  public void setConfiguracion(ConfiguracionPartida configuracion)
  ```

- **setJugadores**
  ```
  public void setJugadores(java.util.List<Jugador> jugadores)
  ```

- **setGrupos**
  ```
  public void setGrupos(java.util.List<Grupo> grupos)
  ```

- **setEquipos**
  ```
  public void setEquipos(java.util.List<Equipo> equipos)
  ```

- **getDatosiniciales**
  ```
  public org.json.JSONObject getDatosiniciales()
  ```

- **setDatosiniciales**
  ```
  public void setDatosiniciales(org.json.JSONObject datosiniciales)
  ```

- **getParticipaciones**
  ```
  public java.util.List<Participacion> getParticipaciones()
  ```

- **setParticipaciones**
  ```
  public void setParticipaciones(java.util.List<Participacion> participaciones)
  ```

- **getRelaciones_GE**
  ```
  public java.util.HashMap<java.lang.String,java.lang.String> getRelaciones_GE()
  ```

- **setRelaciones_GE**
  ```
  public void setRelaciones_GE(java.util.HashMap<java.lang.String,java.lang.String>
  relaciones_GE)
  ```

- **getVotaciones**
  ```
  public java.util.List<Votacion> getVotaciones()
  ```

- **setVotacion**
  ```
  public void setVotacion(java.util.List<Votacion> votaciones)
  ```

- **getPreguntas**
  ```
  public java.util.List<Pregunta> getPreguntas()
  ```

- **setPreguntas**
  ```
  public void setPreguntas(java.util.List<Pregunta> preguntas)
  ```

- **getMapaRelacionVariableNombreGrupo**
  ```
  public java.util.HashMap<java.lang.String,java.lang.String> getMapaRelacionVariabl
  eNombreGrupo()
  ```

- **setMapaRelacionVariableNombreGrupo**
  ```
  public void setMapaRelacionVariableNombreGrupo(java.util.HashMap<java.lang.String,
  java.lang.String> mapaRelacionVariableNombreGrupo)
  ```

- **getResultadoPartida**
  ```
  public ResultadoPartida getResultadoPartida()
  ```

- **setResultadoPartida**
  ```
  public void setResultadoPartida(ResultadoPartida resultadoPartida)
  ```

- **isConfirmada**
  ```
  public boolean isConfirmada()
  ```

- **setConfirmada**
  ```
  public void setConfirmada(boolean confirmada)
  ```

- **getConexionesEstadisticas**
  `public java.util.List<javax.websocket.Session> getConexionesEstadisticas()`

- **setConexionesEstadisticas**
  `public void setConexionesEstadisticas(java.util.List<javax.websocket.Session> cone xionesEstadisticas)`

- **añadirGrupo**
  `public void añadirGrupo(Grupo g)`

- **añadirEquipo**
  `public void añadirEquipo(Equipo e)`

- **añadirParticipacion**
  `public void añadirParticipacion(Participacion p)`

- **quitarJugador**
  `public void quitarJugador(Participacion par)`

- **añadirPregunta**
  `public void añadirPregunta(Pregunta p)`

- **añadirVotacion**
  `public void añadirVotacion(Votacion v)`

- **añadirConEstadistica**
  `public void añadirConEstadistica(javax.websocket.Session ses)`

- **buscarPreguntaPorNumeroPregunta**
  `public Pregunta buscarPreguntaPorNumeroPregunta(int numeroPregunta)`

  Localiza una pregunta dado su numero de pregunta.

  Parameters:

  `numeroPregunta` - numero de pregunta

  Returns:

  pregunta

- **obtenerSeguimientoRespuestas**
  `public java.util.List<SeguimientoRespuestasPreVot> obtenerSeguimientoRespuestas()`

  Necesario para el seguimiento de respuesta de jugadores en la vista del profesor. Este metodo devuelve todos los jugadores que han respondido a la pregunta actual para el grupo al que pertenecen.
  Returns:
  listado con el seguimiento de las repsuestas de una Pregunta

- **obtenerSeguimientoVotaciones**
  `public java.util.List<SeguimientoRespuestasPreVot> obtenerSeguimientoVotaciones()`

  Necesario para el seguimiento de respuesta de jugadores en la vista del profesor. Este metodo devuelve todos los jugadores que han respondido a la votacion actual para el grupo al que pertenecen.
  Returns:
  listado de seguimiento de las respuestas a una Votacion

- **obetenerEstadisticaPorPregunta**
  `public java.util.List<EstadisticaPorPregunta> obetenerEstadisticaPorPregunta()`

  Funcion encargada de calcular las estadisticas de la pregunta seleccionada que serán mostradas en la página del profesor al finalizar la pregunta.
  Returns:
  listado con las estadisticas por pregunta de cada grupo

- **obetenerEstadisticaPorVotacion**
  `public java.util.List<EstadisticasPorVotacion> obetenerEstadisticaPorVotacion()`

  Funcion encargada de calcular las estadisticas de la votacion seleccionada que serán mostradas en la página del profesor al finalizar la pregunta.
  Returns:
  listado con las estadisticas de la Votacion de cada grupo

Class Pregunta

- java.lang.Object
  - org.servidor.modelo.Pregunta

```
public class Pregunta
extends java.lang.Object
```

Esta clase guarda la información sobre una pregunta.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`Pregunta()`

Method Detail

- **getTipo**
  `public int getTipo()`

- **getNumeroPregunta**
  `public int getNumeroPregunta()`

- **getNumeroOpciones**
  `public int getNumeroOpciones()`

- **getRespuesta**
  `public java.lang.String getRespuesta()`

- **getRepeticion**
  `public int getRepeticion()`

- **getTiempo**
  `public int getTiempo()`

- **getGrupoVarRespuesta**
  `public java.util.HashMap<java.lang.String,java.lang.String> getGrupoVarRespuesta()`

- **setTipo**
  `public void setTipo(int tipo)`

- **setNumeroPregunta**
  `public void setNumeroPregunta(int numeroPregunta)`

- **setNumeroOpciones**
  ```
  public void setNumeroOpciones(int numeroOpciones)
  ```

- **setRespuesta**
  ```
  public void setRespuesta(java.lang.String respuesta)
  ```

- **setRepeticion**
  ```
  public void setRepeticion(int repeticion)
  ```

- **setTiempo**
  ```
  public void setTiempo(int tiempo)
  ```

- **setGrupoVarRespuesta**
  ```
  public void setGrupoVarRespuesta(java.util.HashMap<java.lang.String,java.lang.Stri
  ng> grupoVarRespuesta)
  ```

- **getContradorRespuesta**
  ```
  public int getContradorRespuesta()
  ```

- **setContradorRespuesta**
  ```
  public void setContradorRespuesta(int contradorRespuesta)
  ```

- **getMapaRelacionVariableRespuestaGrupo**
  ```
  public java.util.HashMap<java.lang.String,java.lang.String> getMapaRelacionVariabl
  eRespuestaGrupo()
  ```

- **setMapaRelacionVariableRespuestaGrupo**
  ```
  public void setMapaRelacionVariableRespuestaGrupo(java.util.HashMap<java.lang.Stri
  ng,java.lang.String> mapaRelacionVariableRespuestaGrupo)
  ```

- **getOpciones**
  ```
  public java.util.List<java.lang.String> getOpciones()
  ```

- **setOpciones**
  ```
  public void setOpciones(java.util.List<java.lang.String> opciones)
  ```

- **nuevaRespuesta**
  `public void nuevaRespuesta()`

- **construirArrayDeOpciones**
  `public void construirArrayDeOpciones()`

  Funcion herramienta para construir un Array de opciones de respuesta a la hora de mostrarlas al jugador.

Class ResultadoPartidaGrupo

- java.lang.Object
  - org.servidor.modelo.ResultadoPartidaGrupo

```
public class ResultadoPartidaGrupo
extends java.lang.Object
```

Esta clase almacena los resultados de una partida por un grupo determinado.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`ResultadoPartidaGrupo()`

Method Detail

- **getVariableIdentificacionGrupo**
  `public java.lang.String getVariableIdentificacionGrupo()`

- **getNumeroResuestasCorrectas**
  `public int getNumeroResuestasCorrectas()`

- **getNumeroRespuestasIncorrectas**
  `public int getNumeroRespuestasIncorrectas()`

- **getPuntosGrupo**
  `public int getPuntosGrupo()`

- **isGrupoGanador**
  `public boolean isGrupoGanador()`

- **isGrupoEliminado**
  `public boolean isGrupoEliminado()`

- **setVariableIdentificacionGrupo**
  `public void setVariableIdentificacionGrupo(java.lang.String variableIdentificacionGrupo)`

- **setNumeroResuestasCorrectas**
  `public void setNumeroResuestasCorrectas(int numeroResuestasCorrectas)`

- **setNumeroRespuestasIncorrectas**
  `public void setNumeroRespuestasIncorrectas(int numeroRespuestasIncorrectas)`

- **setPuntosGrupo**
  `public void setPuntosGrupo(int puntosGrupo)`

- **setGrupoGanador**
  `public void setGrupoGanador(boolean grupoGanador)`

- **setGrupoEliminado**
  `public void setGrupoEliminado(boolean grupoEliminado)`

Class ResultadoPartida

- java.lang.Object
  - org.servidor.modelo.ResultadoPartida

```
public class ResultadoPartida
extends java.lang.Object
```

Esta clase alamcena los resultados de una partida.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`ResultadoPartida()`

Method Detail

- **getIdPartida**
  `public int getIdPartida()`

- **getNumeroPreguntasFormuladas**
  `public int getNumeroPreguntasFormuladas()`

- **getMaxPuntosAlcanzables**
  `public int getMaxPuntosAlcanzables()`

- **getResultadosPorGrupos**
  `public java.util.List<ResultadoPartidaGrupo> getResultadosPorGrupos()`

- **setIdPartida**
  `public void setIdPartida(int idPartida)`

- **setNumeroPreguntasFormuladas**
  `public void setNumeroPreguntasFormuladas(int numeroPreguntasFormuladas)`

- **setMaxPuntosAlcanzables**
  `public void setMaxPuntosAlcanzables(int maxPuntosAlcanzables)`

- **setResultadosPorGrupos**
  `public void setResultadosPorGrupos(java.util.List<ResultadoPartidaGrupo> resultadosPorGrupos)`

- **añadirResultadoPartidaGrupo**
  `public void añadirResultadoPartidaGrupo(ResultadoPartidaGrupo rpg)`

Class SeguimientoRespuestasPreVot

- java.lang.Object
  - org.servidor.modelo.SeguimientoRespuestasPreVot

```
public class SeguimientoRespuestasPreVot
extends java.lang.Object
```

Esta clase guarda la información de los jugadores que han respondido una votación o pregunta agrupados por grupo para llevar un seguimiento en tiempo real.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`SeguimientoRespuestasPreVot()`

Method Summary

Modifier and Type	Method and Description
`void`	`añadirJugador(java.lang.String jug)`
`java.util.List<java.lang.String>`	`getJugadoresRespondido()`
`java.lang.String`	`getNombreGrupo()`
`void`	`setJugadoresRespondido(java.util.List<java.lang.String> jugadoresRespondido)`
`void`	`setNombreGrupo(java.lang.String nombreGrupo)`

Class Votacion

- java.lang.Object
  - org.servidor.modelo.Votacion

```
public class Votacion
extends java.lang.Object
```

Esta clase guarda la información de una votación registrada en una partida.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`Votacion()`

Method Detail

- **getTipo**
  `public int getTipo()`

- **getOpciones**
  `public java.util.List<java.lang.String> getOpciones()`

- **getGruposVarVotacion**
  `public java.util.HashMap<java.lang.String,java.lang.String> getGruposVarVotacion()`

- **getRepeticionVotacion**
  `public int getRepeticionVotacion()`

- **getTiempo**
  `public int getTiempo()`

- **setTipo**
  `public void setTipo(int tipo)`

- **setOpciones**
  ```
  public void setOpciones(java.util.List<java.lang.String> opciones)
  ```

- **setGruposVarVotacion**
  ```
  public void setGruposVarVotacion(java.util.HashMap<java.lang.String,java.lang.
  String> gruposVarVotacion)
  ```

- **setRepeticionVotacion**
  ```
  public void setRepeticionVotacion(int repeticionVotacion)
  ```

- **setTiempo**
  ```
  public void setTiempo(int tiempo)
  ```

- **getMapaRelacionVariableRespuestaGrupo**
  ```
  public java.util.HashMap<java.lang.String,java.lang.String> getMapaRelacionVar
  iableRespuestaGrupo()
  ```

- **setMapaRelacionVariableRespuestaGrupo**
  ```
  public void setMapaRelacionVariableRespuestaGrupo(java.util.HashMap<java.lang.
  String,java.lang.String> mapaRelacionVariableRespuestaGrupo)
  ```

- **getContradorRespuesta**
  ```
  public int getContradorRespuesta()
  ```

- **setContradorRespuesta**
  ```
  public void setContradorRespuesta(int contradorRespuesta)
  ```

- **nuevaRespuesta**
  ```
  public void nuevaRespuesta()
  ```

Class ConfiguracionPartida

- java.lang.Object
  - org.servidor.modelo.ConfiguracionPartida

```
public class ConfiguracionPartida
extends java.lang.Object
```

Esta clase guarda datos de configuración generales para la partida.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`ConfiguracionPartida()`

Method Summary

Modifier and Type	Method and Description
`int`	`getPersonasporgrupo()`
`double`	`getPorcencorreccion()`
`int`	`getTiempopregunta()`
`int`	`getTiempovotacion()`
`boolean`	`isRepartoaleatorio()`
`void`	`setPersonasporgrupo(int personasporgrupo)`
`void`	`setPorcencorreccion(double porcencorreccion)`
`void`	`setRepartoaleatorio(boolean repartoaleatorio)`
`void`	`setTiempopregunta(int tiempopregunta)`
`void`	`setTiempovotacion(int tiempovotacion)`

Class DatosIniciales

- java.lang.Object
  - org.servidor.modelo.DatosIniciales

```
public class DatosIniciales
extends java.lang.Object
```

Esta clase gaurda la información referente a la configuración que el profesor ha establecido para la partida.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
DatosIniciales()

Method Summary

Modifier and Type	Method and Description
Int	getAle()
Int	getBote()
java.lang.String	getContinuarpartida()
Int	getContipar()
Int	getMat()
int	getN()
java.lang.String	getNom()
int	getNp()
int	getTipocorreccion()
int	getTiporespuesta()
java.lang.String	getTotalenvi()
void	setAle(int ale)
void	setBote(int bote)
void	setContinuarpartida(java.lang.String continuarpartida)
void	setContipar(int contipar)
void	setMat(int mat)
void	setN(int n)
void	setNom(java.lang.String nom)
void	setNp(int np)
void	setTipocorreccion(int tipocorreccion)
void	setTiporespuesta(int tiporespuesta)
void	setTotalenvi(java.lang.String totalenvi)

Class EstadisticaPorPregunta

- java.lang.Object
  - org.servidor.modelo.EstadisticaPorPregunta

```
public class EstadisticaPorPregunta
extends java.lang.Object
```

Esta clase guarda información referente a las respuestas que se han dado para una pregunta en concreto y poder mostrar una gráfica con dicha información.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
EstadisticaPorPregunta()

Method Summary

Modifier and Type	Method and Description
org.primefaces.model.chart.PieChartModel	getGraficaPastel()
java.lang.String	getNombreGrupo()
double	getPorcentaje()
java.util.HashMap<java.lang.String,java.lang.Integer>	getRelacionOpcionesNumeroRespuestas()
java.lang.String	getRespuestaCorrecta()
boolean	isPreguntaSuperada()
void	setGraficaPastel(org.primefaces.model.chart.PieChartModel graficaPastel)
void	setNombreGrupo(java.lang.String nombreGrupo)
void	setPorcentaje(double porcentaje)
void	setPreguntaSuperada(boolean preguntaSuperada)
void	setRelacionOpcionesNumeroRespuestas(java.util.HashMap<java.lang.String,java.lang.Integer> relacionOpcionesNumeroRespuestas)
void	setRespuestaCorrecta(java.lang.String respuestaCorrecta)

Class EstadisticasPorVotacion

- java.lang.Object
  - org.servidor.modelo.EstadisticasPorVotacion

```
public class EstadisticasPorVotacion
extends java.lang.Object
```

Esta clase guarda información referente a las respuestas que se han dado para una votación en concreto y poder mostrar una gráfica con dicha información.

Version:
1.0

Author:
Alberto Jiménez Vázquez

Constructor Summary

Constructor and Description
`EstadisticasPorVotacion()`

Method Summary

Modifier and Type	Method and Description
`org.primefaces.model.chart.PieChartModel`	`getGraficaPastel()`
`java.lang.String`	`getNombreGrupo()`
`java.util.HashMap<java.lang.String,java.lang.Integer>`	`getRelacionOpcionesNumeroRespuestas()`
`void`	`setGraficaPastel(org.primefaces.model.chart.PieChartModel graficaPastel)`
`void`	`setNombreGrupo(java.lang.String nombreGrupo)`
`void`	`setRelacionOpcionesNumeroRespuestas(java.util.HashMap<java.lang.String,java.lang.Integer> relacionOpcionesNumeroRespuestas)`