RÍO NEGRO
UNIVERSIDAD NACIONAL

Licenciatura en Sistemas

IMPLEMENTACIÓN DE SOLUCIÓN MÓVIL PARA LA MONITORIZACIÓN DE ATLETAS DE ALTO RENDIMIENTO

Alumno: Muñoz Abbate Horacio Andres

Directora: Dra. Britos Paola

Co-director: Mg. Vivas Luis

{pbritos, lvivas}@unrn.edu.ar

Año:2017

Agradecimientos

Me gustaría que estas líneas sirvieran para expresar mi más profundo y sincero agradecimiento a todas aquellas personas que con su ayuda han colaborado en la realización del presente trabajo, en especial a la Dra. Paola Britos, directora de este trabajo, por la orientación, el seguimiento y la supervisión continúa de la misma, pero sobre todo por la motivación y el apoyo recibido a lo largo de estos años. Especial reconocimiento merece el interés mostrado por mi trabajo y las sugerencias recibidas por el Mg. Luis Vivas Director de la carrera de Sistemas y del Laboratorio de Informática Aplicada (LIA), con la que me encuentro en deuda por el ánimo infundido y la confianza en mí depositada. Quisiera hacer extensiva mi gratitud a mis compañeros del LIA y todos los profesores que acompañaron este proceso de aprendizaje. También quiero dar las gracias a Lic. Matías Scavo por su colaboración en el suministro de información y métodos de entrenamientos necesarios para la realización del trabajo y la correlación empírica de esta investigación. Un agradecimiento a mi familia y otro muy especial merece la comprensión, paciencia y el ánimo recibidos por mi compañera de vida Lic. Maria Laura Sosa que además me asistió en la corrección y redacción de la tesina. A todos ellos, muchas gracias.

Índice

I. Introducción

Resumen

Este trabajo presenta una solución para monitorizar atletas en tiempo real. También, se presentan los estándares para el desarrollo de la mismo y tecnologías necesarias para llevar a cabo dicha implementación. La creciente demanda de dispositivos móviles y del sector para generar aplicaciones son los factores que motivan la investigación sobre plataformas móviles que hace posible realizar este producto. La implementación de websocket hace esta comunicación en tiempo real a través de dispositivos móviles con los servicios web y la visualización la información para el entrenador en tiempo real.

Palabras Claves

Teléfonos Inteligentes, Mhealth, Android, WebSocket, JavaEE, Wearable.

El concepto de mhealth, que viene del inglés salud y móvil, se puede definir como el cuidado de la salud mediante dispositivos móviles. En la actualidad existe un amplio mercado de aplicaciones móviles que ayudan a las personas a mejorar su calidad de vida. Por ejemplo, tanto en la tienda de Google, como en la de Apple, se presenta una categoría llamada “salud y bienestar” donde se pueden comprar o adquirir de forma gratuita aplicaciones para el monitoreo y seguimiento de actividad física, control de peso, alimentación saludable entre otros.

Las personas que realizan actividad física, pueden utilizar dispositivos móviles, como teléfonos inteligentes y/o wearables[1], para seguir sus

[1] Wearable, palabra que proviene del inglés wear que significa vestible.

entrenamientos a través de dichas aplicaciones. Algunos valores importantes a tener en cuenta, en este tipo de actividad son: la frecuencia cardiaca (FC), pasos, peso, temperatura (corporal y de ambiente), posición (latitud, longitud y altura) y velocidad. Estos parámetros sirven para establecer métricas de progreso y planificación de futuros entrenamientos.

Los dispositivos móviles cuentan generalmente con sensores embebidos que permiten obtener algunas de estas variables, como por ejemplo el GPS[2], que nos permite obtener la velocidad de desplazamiento y posición geográfica (latitud, longitud y altura). Pero con esto no alcanza para poder cumplir con los requerimientos del entrenador, es por ello que para extender funcionalidades se pueden utilizar sensores externos, como balanzas inteligentes, que nos permiten obtener nuestro peso, proporciones de grasa y masa ósea. Otro sensor muy útil es el pulsómetro, que permiten obtener la FC de forma inalámbrica (Polar y Garmin son marcas líderes en este tipo de sensores). Cabe aclarar que los dispositivos wearables, como pulseras y relojes inteligentes, suelen incorporar pulsómetros a través de una luz led; este mecanismo de medición es de menor precisión que un pulsómetro que funciona midiendo la diferencia de potencial que genera el corazón.

En cuanto a la conexión inalámbrica necesaria para conectar sensores externos y dispositivos móviles se puede utilizar bluetooth, que permite generar una conexión de datos segura entre el teléfono y el sensor a conectar. Actualmente, la versión 4.0 permite un ahorro de energía mayor que sus predecesores y mejoras en la seguridad. Otro tipo de conexión, es ANT+, tipo de conexión de código abierto utilizada en la actualidad en pocos los modelos de teléfono; ya que son los modelos conocidos como “de gama alta” como pueden ser Samsung Galaxy S6 y S7, LG G3-G4 o Sony Z1.

[2] GPS siglas en inglés que significan Sistema de Posicionamiento Global.

II. Estado de la Cuestión

II.1. Entrenamiento y Atletas de Alto Rendimiento

El rendimiento del atleta es el resultado de una combinación de diversos factores, probablemente el más importante a la hora de determinar el potencial sea la dotación genética. Otros factores que tiene un profundo efecto sobre el rendimiento son, la cantidad y calidad del entrenamiento previo a las competiciones y también el rendimiento conseguido por un deportista en un momento dado puede estar condicionado por su estado nutricional y de salud.

Desde principios del siglo XX médicos, fisiólogos del ejercicio y entrenadores se vienen preocupando para desarrollar indicadores fisiológicos que permitan valorar las capacidades y características del atleta que realiza esfuerzos físicos y prever su futuro rendimiento en competición.

El control fisiológico[3] del deportista es el resultado de la evolución de la ciencia del ejercicio y la metodología del entrenamiento producto del avance tecnológico.

La Fisiología del Ejercicio es un campo de la Ciencia que estudia las respuestas del organismo humano a los estímulos del ejercicio físico. Comprender los mecanismos y procesos fisiológicos, ayuda a perfeccionar las adaptaciones, prevenir sobre-entrenamiento lesiones y aumentar el rendimiento.

Las respuestas al ejercicio, son influenciadas por la denominada “carga externa”, la cual contempla un volumen, duración, intensidad, frecuencia y densidad de la carga.

[3] La fisiología del ejercicio debe considerarse como una ayuda para el entrenamiento el uso de indicadores fisiológicos básicos como: cinética del lactato, consumo de oxígeno, frecuencia cardiaca, umbrales de entrenamiento entre otros. Los cuales deberían utilizar los entrenadores y preparadores físicos.

La intensidad, siendo uno de los cinco componentes de la carga externa, es de trascendencia para determinar áreas de entrenamiento y direccionar las adaptaciones fisiológicas. Por ej. en el ciclismo a la intensidad del ejercicio la podemos expresar en términos de porcentaje de la frecuencia cardiaca máxima, consumo de oxígeno, concentraciones de lactato sanguíneo, producción de potencia o percepción del esfuerzo (Jeukendrup y Van Diemen, 1998).

II.1.1. Frecuencia Cardiaca (FC)

Se define la FC como el número de contracciones ventriculares efectuadas por el corazón en un minuto, medidas en latidos o pulsaciones por minuto. Es uno de los parámetros cardiovasculares más sencillos de medir y más utilizados por los deportistas como control de la intensidad, a causa de su relación casi directa con el gasto cardiaco.

II.1.2. Registro de Frecuencia Cardiaca (Fc)

El registro FC en latidos por minuto (Lpm) mediante el uso de pulsómetros ya es habitual en diferentes contextos de actividad física. Este registro de la respuesta de la FC al ejercicio permite a los entrenadores y preparadores físicos el diseño de tareas acorde con la planificación y objetivos de entrenamiento, teniendo datos fiables a tiempo real sobre la intensidad del ejercicio. Mucho de estos instrumentos de medición de la FC, son capaces de grabar todas las posibles variaciones de la FC del deportista durante la sesión de entrenamiento y posteriormente descargar estos datos para su posible análisis e interpretación.

Otras manifestaciones de FC como la frecuencia cardiaca en reposo (FCR) y el comportamiento de la FC post ejercicio también son de gran

utilidad, ya que permiten establecer criterios de valoración de la condición física del deportista.

La FCR es la que poseemos en el momento de menos actividad física, es decir en reposo. Para determinar la FCR se registra el pulso al despertar por la mañana cada día durante una semana, posteriormente se realiza una media semanal.

Existen unas tablas que clasifican la condición física del deportista en función de su FCR. A continuación se puede ver la tabla completa II.7.

	Edad					
Clasificación	18-25	26-35	36-45	46-55	56-65	Sobre 65
Mujeres						
Excelente	54-60	54-59	54-59	54-60	54-59	54-59
Bueno	61-65	60-64	62-64	61-65	61-64	60-64
Sobre el Promedio	66-69	66-68	66-69	66-69	67-69	66-68
Promedio	70-73	69-71	70-72	70-73	71-73	70-72
Bajo el Promedio	74-78	72-76	74-78	74-77	75-77	73-76
Pobre	80-84	78-82	79-82	78-84	79-81	79-84
Muy Pobre	86-100	84-94	84-92	85-96	85-96	88-96
Varones						
Excelente	49-55	49-54	50-56	50-57	51-56	50-55
Bueno	57-61	57-61	60-62	59-63	59-61	58-61
Sobre el Promedio	63-65	62-65	64-66	64-67	64-67	62-65
Promedio	67-69	66-70	68-70	68-71	68-71	66-69
Bajo el Promedio	71-73	72-74	73-76	73-76	72-75	70-73
Pobre	76-81	77-81	77-82	79-83	76-81	75-79
Muy Pobre	84-95	84-94	86-96	85-97	84-94	83-98

Tabla II.7. Tabla de clasificación de atletas por edad y sexo según su FCR.

Existen pruebas de valoración de la condición física como el test de Lían que establecen un nivel de condición física basándose en el principio fisiológico que establece que “la recuperación de la frecuencia cardiaca después de un esfuerzo protocolizado es más rápida cuanto mayor sea la aptitud y preparación física del deportista.” (Chicharro 2001)

II.1.3. Frecuencia Cardiaca Máxima (FCmax)

Relacionamos la FCMax con el mayor valor de la FC que se alcanza en un esfuerzo de máxima intensidad, con una elevada participación muscular y hasta el agotamiento (Wilmore et al, 2007). La

FCMax es un valor individual, determinado genéticamente y relativamente constante en el tiempo, que desciende ligeramente con la edad.

Para determinar la FCMax existen dos formas, la estimación indirecta o teórica, basadas en cálculos estadísticos según distintas fórmulas que se pueden ver a continuación (ver tabla II.1):

Fórmula	FCMax
Robson (1938)	212- (0,775 x edad)
Karvonen y Cols (1957)	220 – edad
Whaley y Cols (1992) Código Fumador - CF (1 fumador, 0 no fumador) Peso Corporal - PC.	hombres= 203,9 - 0,812 x edad + 0,276 x FCR - 0,084 x PC - 4,5 x CF mujeres =204,8 - 0,718 x edad + 0,162 x FCR - 0,105 x PC - 6,2 x CF.
Tanaka y Cols (2001)	hombre= [208,7 - (0,73 * edad en años)] mujer=[208,1 - (0,77 * edad en años)]
Gellish (2007)	207- (0,7 * edad)

Tabla II.1 Fórmulas cálculo FCR.

II.2 Dispositivos Móviles

Los teléfonos inteligentes, también llamados smartphone, se han convertido en dispositivos imprescindibles para sociedad actual, gracias a la evolución de tres factores, hardware, software e Internet.

Cabe destacar que esta necesidad se vincula con los servicios que Internet ofrece como, acceso a noticias, entretenimiento, videojuegos y redes sociales. Internet y los teléfonos Inteligentes logran una sinergia que dia a dia amplía la frontera de lo posible.

Las smartphone más destacados en la actualidad son iPhone, Android y Windows Phone. Dichas plataformas hacen uso eficientemente de todos los componente de hardware que actualmente brindan los smartphone como: pantallas táctiles (paneles LCD, IPS o AMOLED), cámara de foto y video, sensores embebidos (acelerómetros, giroscopio, luz), conectividad de corto alcance (Bluetooth y NFC) y largo alcance (WIFI, 3G y 4G) entre los componentes más destacados.

Cada plataforma tiene su propio Sistema Operativo (SO). En caso de iPhone implementa iOS, un SO basado en OSx desarrollado por Apple. En las primeras versiones el código de desarrollo era en Object-C, en la actualidad se puede utilizar Swift 3.0, un lenguaje de código abierto.

Android comprado por Google en el 2001 y desde entonces desarrollado por dicha empresa. Es un SO basado en Linux y de código abierto. Se desarrolla en Java, aunque también se puede trabajar en C++, y en la actualidad los ultimos modelo de Samsung y Motorola ofrecen la versión 6.0, código de nombre Marshmallow (traducción al español de Malvavisco).

Por último Windows Phone, con el sistema operativo basado en Windows es el primero que intenta unificar este SO para todas las plataformas en la versión Windows 10. Desarrollado por Microsoft han logrado implementar un mismo sistema para todas sus plataformas como: Smartphone, tablets, notebook y PC. Windows es la plataforma de escritorio más popular en Argentina sin lugar a duda, pero los dispositivos móviles de la empresa no logra ser significativa respecto a Android y iPhone.

En la figura II.1 se puede ver la distribución de plataformas por SO. Android tiene una amplia ventaja con el 79% de la cuota del mercado.

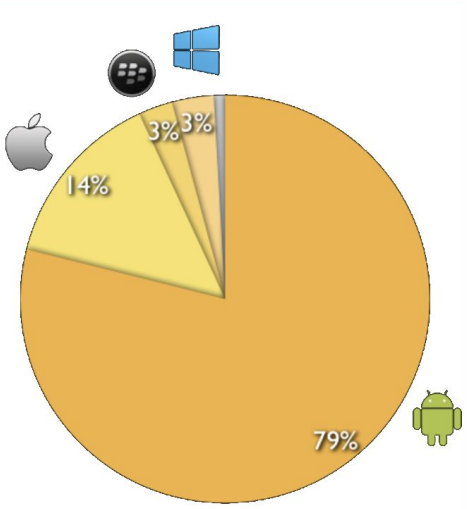


Fig.II.1 - Cuota de Mercado Mundial de Smartphones (Gartner).

Para implementar una aplicación en dichas plataformas, se debe desarrollar una aplicación por cada SO antes mencionado, esto se conoce como aplicaciones Nativas. Estas se desarrollan con un IDE[4] y SDK[5] propia de la plataforma en la cual se va a desarrollar.

Para iPhone, necesitamos el entorno Xcode sobre plataforma xOS. Para Windows phone se utiliza Visual Studio en Plataforma Windows y para Android se pueden utilizar todos los SO antes mencionados y también cualquier distribución de linux con el IDE Android Studio[6].

En este punto Android tiene una ventaja que no restringe el SO del desarrollador y permite utilizar código abierto.

Existe la posibilidad de implementar una única solución que se pueda acceder en todos los SO móviles antes mencionados. Esta característica es posible mediante tecnología web en lo que se conoce como aplicaciones multiplataformas. En este sentido HTML5 aporta muchos componentes para el desarrollo de aplicaciones móviles y un creciente número de frameworks que permiten desarrollar aplicaciones móviles web

[4] IDE acrónimo en inglés Integrated development environment.

[5] SDK acrónimo en inglés Software Development Kit.

[6] Google recomienda utilizar Android Studio como IDE.

cada vez más parecidas a las Nativas, no solo en lo estético sino también en rendimiento.

La decisión de una implementación Web o Nativa depende de los requisitos de la aplicación que debamos implementar. El acceso a funcionalidades avanzadas de hardware como Bluetooth en el desarrollo Web tiene limitaciones. Dependen en gran medida del explorador que la ejecute como puede ser Chrome, Safari, Firefox o Egde. En este sentido, actualmente solo Android con Chrome permite hacer uso de Bluetooth mediante una aplicación web.

II.3. Metodología de Desarrollo

Realizar un desarrollo de software con éxito y de calidad depende de varios factores como, por ejemplo, las personas seleccionadas, las herramientas y tecnologías a utilizar, la arquitectura de software y la metodología que guiará el proceso. Su correcta elección es un factor éxito.

Por lo tanto, las metodologías ágiles permiten generar productos de calidad, basándose en la adaptabilidad del proceso de desarrollo de software para aumentar sus posibilidades de éxito por la flexibilidad y eficacia sobre las metodologías tradicionales.

La metodología Scrum se encuentra clasificada como una metodología ágil la cual puede ser aplicada a cualquier proyecto, sin embargo, la esta metodología es más comúnmente utilizado en el desarrollo de software (Henrik Kniberg, 2007).

El proceso Scrum es adecuado para los proyectos que cambian rápidamente o surgen requisitos de forma continua. El desarrollo de software con Scrum progresa a través de una serie de iteraciones llamados sprints, que duran de una a cuatro semanas. El modelo Scrum sugiere cada sprint comienza con una breve reunión de planificación y concluye con una revisión.

En la figura II.2 se puede ver un esquema del proceso completo de la metodología.

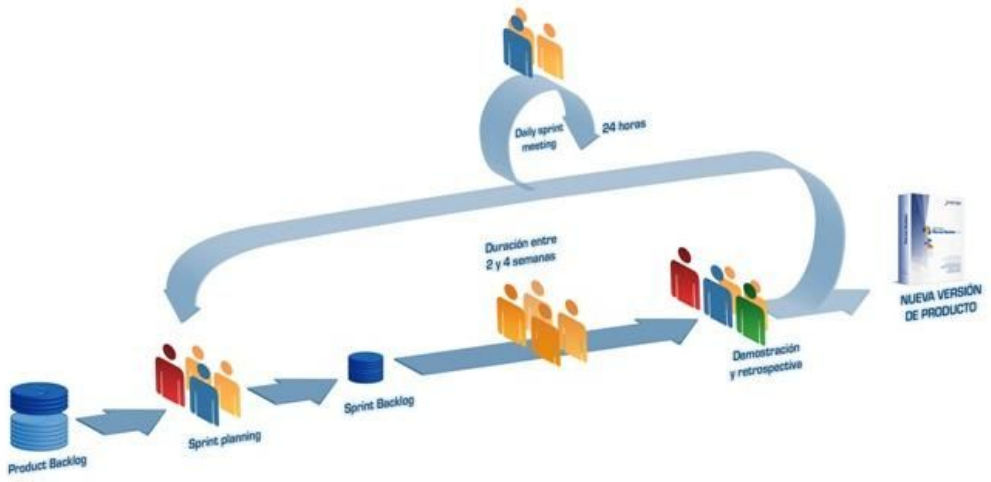


Fig.II.2. Proceso Scrum.

El desarrollo se realiza de forma iterativa e incremental. Cada iteración, denominada Sprint, tiene una duración preestablecida de entre 2 y 4 semanas, obteniendo como resultado una versión del software con nuevas prestaciones listas para ser usadas. En cada nuevo Sprint, se va ajustando la funcionalidad ya construida y se añaden nuevas prestaciones priorizándolos siempre aquellas que aporten mayor valor de negocio.

El proceso se divide en las siguientes etapas:

- Product Backlog: Conjunto de requisitos denominados historias, escritas en un lenguaje no técnico y priorizados por valor de negocio. Los requisitos y prioridades se revisan y ajustan durante el curso del proyecto a intervalos regulares.
- Sprint Planning: Reunión durante la cual el Product Owner presenta las historias del backlog por orden de prioridad. El equipo determina la cantidad de historias que puede comprometerse a completar en ese sprint, para en una segunda parte de la reunión, decidir y organizar cómo lo va a conseguir.

- Sprint: Iteración de duración prefijada durante la cual el equipo trabaja para convertir las historias del Product Backlog a las que se ha comprometido, en una nueva versión del software totalmente operativo.
- Sprint Backlog: Lista de las tareas necesarias para llevar a cabo las historias del sprint.
- Daily sprint meeting: Reunión diaria de cómo máximo 15 min. en la que el equipo se sincroniza para trabajar de forma coordinada. Cada miembro comenta que hizo el día anterior, que hará hoy y si hay impedimentos.
- Demo y retrospectiva: Reunión que se celebra al final del sprint y en la que el equipo presenta las historias conseguidas mediante una demostración del producto. Posteriormente, en la retrospectiva, el equipo analiza qué se hizo bien, qué procesos serían mejorables y discute acerca de cómo perfeccionarlos.

La gestión de un proyecto Scrum se centra en definir cuáles son las características que debe tener el producto a construir (qué construir, qué no y en qué orden) y en vencer cualquier obstáculo que pudiera entorpecer la tarea del equipo de desarrollo.

El equipo Scrum está formado por los siguientes roles:

Scrum master: Persona que lidera al equipo y facilita que se cumpla las reglas y procesos de la metodología. Gestiona la reducción de impedimentos del proyecto.

Product owner: Es el cliente que usará el software. Se focaliza en la parte de negocio del proyecto. Traslada la visión del proyecto al equipo, formaliza las prestaciones en historias a incorporar en el Product Backlog y las prioriza de forma regular.

Team: Grupo de profesionales con los conocimientos técnicos necesarios y que desarrollan el proyecto de manera conjunta llevando a cabo las historias a las que se comprometen al inicio de cada sprint.

II.4. Arquitectura de Software

La arquitectura de software conforma la columna vertebral de cualquier sistema y constituye uno de sus principales atributos de calidad.

Generalmente las arquitecturas de aplicaciones empresariales se componente de tres capas bien definidas: presentación, modelo de negocios y persistencia. Esta división permite que cada componente tenga su responsabilidad y esconda su lógica a las demás, permitiendo escalar de forma más sencilla y hace de un producto más mantenible, otro factor de calidad.

En la capa de presentación los objetos trabajan directamente con las interfaces de de negocios, implementando el patrón Model View Controller. En este, el modelo (Model) es modificable por las funciones de negocio, siendo éstas solicitadas por el usuario, mediante el uso de un conjunto de vistas (View) que solicitan dichas funciones de negocio a través de un controlador (Controller), que es quien recibe las peticiones de las vistas y las procesa.

La capa de negocio está formada por servicios implementados por objetos de negocio. Estos delegan gran parte de su lógica en los modelos del dominio que se intercambian entre todas las capas.

Finalmente, la capa de persistencia facilita el acceso a los datos y su almacenamiento en una base de datos.

La necesidad de tener una comunicación en tiempo real entre servidor y clientes cada día cobra mayor importancia. Puedo citar ejemplo como juegos multijugador online, donde cada acción de un jugador tiene que ser replicada a todos los participantes del juegos.

II.4.1. Comunicación en Tiempo Real

WebSocket es un protocolo de comunicación Web que permite la comunicación de dos vías entre clientes y servidor. Consiste en lo que se define como un apretón de manos (en inglés handshake) donde se genera un enlace, seguido de envío de mensajes a través de la capa TCP.

Para entender cómo Websocket beneficia nuestra aplicación debemos analizar los paradigmas previos a este protocolo.

El paradigma web funciona básicamente con petición/respuesta, en inglés request/response (w3, 2016). Un cliente envía una petición al servidor y este le responde con el contenido. En la figura II.3 se puede ver un esquema donde un cliente realiza la petición de una página y el servidor retorna una respuesta retornando un archivo .html. Con este cliente pueda ver el contenido.

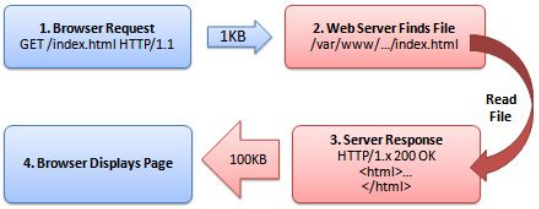


Fig.II.3 Solicitud de un cliente de una página HTML.

Si necesitamos que el servidor envía datos al cliente constantemente, como por ejemplo, obteniendo información de un sensor; existen una técnica que permiten resolver este problema llamada Polling.

Polling resuelve el problema generando peticiones constantemente, dependiendo de la frecuencia que sea necesaria consultar el estado del sensor, ejemplo una vez por segundo.

En 2011, la IETF (Grupo de trabajo en ingeniería de internet - The Internet Engineering Task Force) estandarizó el protocolo RFC2-6455 (IETF, 2016) referido al protocolo websocket. Desde entonces, la mayoría de los navegadores web implementaron clientes que soportan este protocolo.

También se desarrollaron librerías java para implementar websocket desde otros clientes como por ejemplo aplicaciones Android o JavaFX.

Una solución al paradigma antes mencionado es utilizar una única conexión TCP para el tráfico bidireccional, esto es el protocolo websocket.

Websocket define un interfaz de programación de aplicaciones (Application Programming Interface - API) que establece conexiones socket entre un servidor y navegador web. Permite una conexión persistente entre el cliente y el servidor, y ambas partes pueden empezar a enviar mensajes en cualquier momento. En la figura II.3 se puede ver una comparativa con Polling que se explicó anteriormente (websocket, 2015).

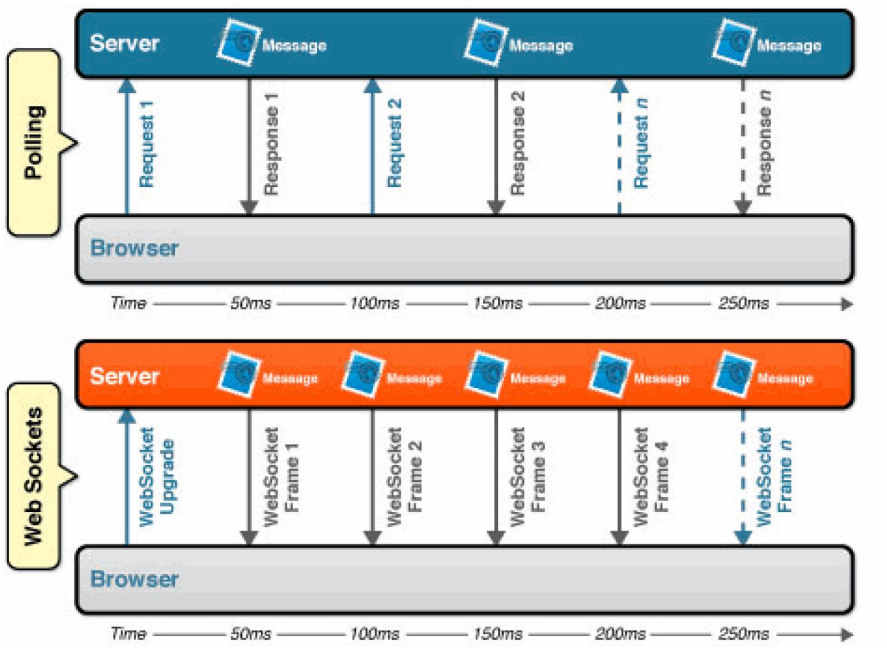


Fig. II.3. Comparativa entre websocket y Polling.

Otro aspecto importante de websocket es la optimización del uso de red respecto otras tecnologías en la figura II.4 se puede ver una comparativa. Para cuantificar la mejora vamos a retomar el paradigma request/response y estudiar cuánta información se transmite por cada mensaje.

Entre el encabezado de la petición y el encabezado de la respuesta tenemos un total de 871 bytes como promedio, teniendo en cuenta un mensaje de menos de 20 caracteres. Supongamos tres casos distintos:

- Caso A, con 1.000 cliente: 871 x 1,000 = 871.000 bytes = 6.6 Mbps
- Caso B, con 10.000 cliente: 871 x 10.000 = 8.710.000 bytes = 66 Mbps Caso C, con 100.000 cliente: 871 x 100.000 = 87.100.000 bytes =665 Mbps

Es una cantidad de información innecesaria que viaja por la red. Si pudiéramos transmitir sólo los datos ahorraríamos gran parte del tráfico. Así es como funciona websocket, solo viajan los mensajes una vez establecida la conexión. Cada mensaje nos consume 2 bytes, entonces para los casos anteriores utilizando websocket quedaria asi:

- Caso A, con 1.000 cliente: 2 x 1,000 = 2.000 bytes =0,015 Mbps
- Caso B, con 10.000 cliente: 2 x 10.000 = 20.000 bytes =0.153 Mbps
- Caso C, con 100.000 cliente: 2 x 100.000 = 200.000 bytes = 1.526 Mbps

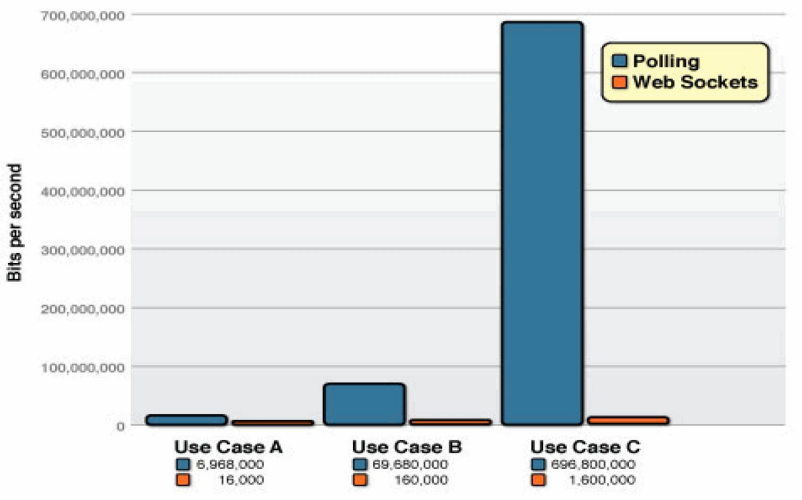


Fig. II.4. Tráfico de red con distintas cantidades de usuarios.

Cada claro de Polling genera muchas peticiones HTTP, por lo cual cuando aumenta el número de clientes aumenta rápidamente el tráfico de red. Además de tener una alta latencia entre que se genera el nuevo

evento y se notifica al cliente. Para una aplicación que requiera notificaciones en tiempo real estas soluciones son inviables.

Para implementar Websocket, JSR-356 (Jcp.org, 2016) es la especificación que debemos seguir. Esta es la base para incluir websocket dentro de la arquitectura Java Empresarial. En la figura II.6 se presenta un esquema de relación de la especificación entre clientes y servidor.

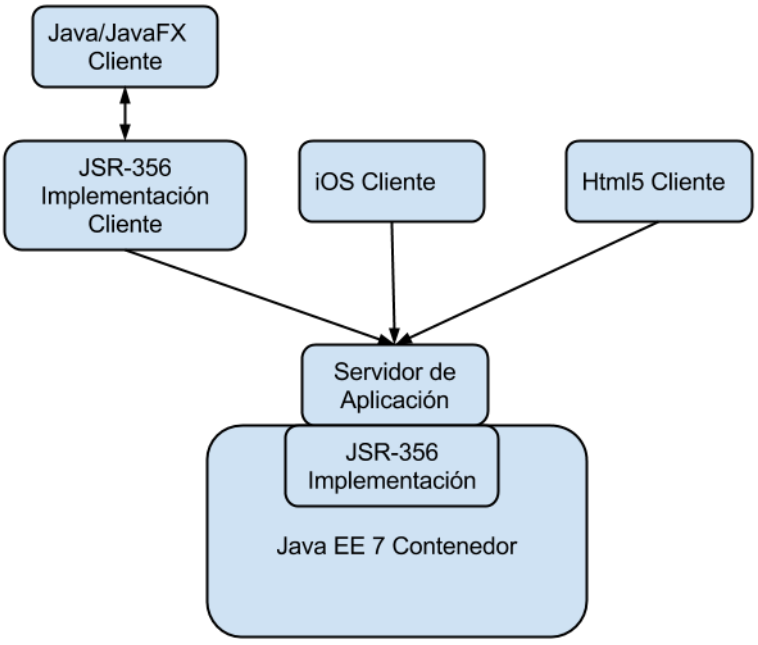


Fig. II.6. Implementación JSR-356 sobre un contenedor Java EE (Oracle.com, 2016).

Cada implementación del protocolo websocket que pretende ser compatible con JSR-356 debe implementar esta API. Como consecuencia de ello, los desarrolladores pueden escribir sus aplicaciones basadas en websocket independientemente de la aplicaciones subyacentes.

Dicha especificación es parte del estándar Java EE 7 (Oracle.com, 2016); por lo tanto, todos los servidores de aplicaciones compatibles Java EE 7 tendrán una implementación del protocolo websocket que se adhiere al estándar JSR-356. También define una API de cliente de Java, que es un subconjunto de la API completa requerida en Java EE 7.

III. Problema a resolver e importancia de resolverlo

Según lo mencionado en el capítulo anterior, puedo establecer las siguientes premisas con respecto al entrenamiento de atletas y tecnologías para desarrollar una solución que le permita a un entrenador mejorar y optimizar sus entrenamientos y permitir un seguimiento en tiempo real.

III.1. FC, Android y Websocket como solución

III.1.1. Frecuencia Cardiaca FC

Queda claro la importancia de la FC como parámetro de control para el entrenamiento y la evaluación del rendimiento del atleta.

También, conceptos como FCR y FCmax, ayudan a establecer límites y guías de progreso en el entrenamiento diario, permitiendo generar comparativas de rendimiento, no solo para para un atleta individual sino también entre atletas.

Otro recurso importante es la recuperación de la FC, que permite diagnosticar la salud de un atleta sometido a los esfuerzos de entrenamiento máximos y también como evoluciona según las sesiones de entrenamiento, una herramienta muy útil para un entrenador.

III.1.2. Plataforma Móvil

Entre las plataforma móviles descritas en el apartado anterior, Android posee varias ventajas respecto a Iphone y Windows Phone.

En primer lugar, Android es la plataforma más elegida por los consumidores, mediante esta plataforma podemos llegar a más usuario y por ende la aplicación puede ser utilizada por más gente.

Otro factor importante para la elección de Android, se debe a que el IDE y la SDK son multiplataforma y de código abierto. Este me permitió desarrollar el proyecto sobre Linux, particularmente Ubuntu. En cambio

para iPhone y Windows Phone se requiere OSX y Windows respectivamente.

El lenguaje de desarrollo también formó parte de la elección de Android, no solo por mi experiencia previa con Java, sino que también el desarrollo de los servicios Web y plataforma Web serían en este mismo Lenguaje.

III.1.3. Arquitectura de la Aplicación y Servicios Web

En cuanto al desarrollo de la aplicación web que provee los servicios necesaria para implementar la solución, Java EE y el protocolo Websocket cubren de forma completa los alcances de la solución a implementar.

Java EE posee una robustez ampliamente reconocida por toda la comunidad. Existen gran cantidad de Frameworks de código abierto que facilitan el desarrollo como Spring, Hibernate y JSF.

Websocket es un protocolo innovador que permite lograr el objetivo de monitorizar en tiempo real, pero también permite optimizar el uso del ancho de banda gracias a la utilización del canal para el envío de mensajes exclusivo, sin la necesidad de enviar todo el encabezado del mensaje como en protocolos anteriores.

III.2. Importancia de la Aplicación

Con el desarrollo de esta aplicación se pretende hacer una contribución al campo del entrenamiento y la salud. Este tipo de software no existe en la región de Río Negro, lugar que se caracteriza por la variedad de deportes posibles de realizar por su geografía, que dispone de río, nieve y mar. Además las creciente demanda de aplicaciones móviles y particularmente en el sector de salud y bienestar genera un mercado que da la posibilidad concreta de implementar este tipo de aplicación con éxito.

Este trabajo lo que busca aportar un análisis de la arquitectura necesaria para generar una herramienta de monitoreo en tiempo real, además estudiar los estándares vigentes a la fecha para la implementación.

III.3. Objetivos de la Aplicación

El objetivo general de este proyecto es desarrollar un sistema fácil de usar que se encargue del seguimiento en tiempo real de atletas de alto rendimiento.

Otros objetivos también importantes, relativos al sistema global, son:

- Comunicación bidireccional entre las dos plataformas (Web y Android) a través de websocket.
- Comunicación entre dispositivos móviles y sensores externos a través de Bluetooth (M2M[7]).
- Demostración del funcionamiento del sistema completo de la arquitectura.

Por otro lado, los objetivos específicos de la plataforma Android son:

- Las aplicaciones deben tener una interfaz de usuario intuitiva y fácil de usar.
- Las aplicaciones deben estar realizadas para dispositivos que tengan al menos la versión 4.3.
- Las aplicaciones se deben ajustar a pantallas de entre 3,2 y 5 pulgadas smartphones.

[7] M2M del Inglés Machine to Machine que Significa Maquina a Maquina.

IV. Solución

En este capítulo se explica cómo se desarrolló la implementación que forma parte de la solución del problema planteado en el capítulo anterior.primera parte se describe la planificación de proyecto en general y como se realizó el seguimiento del mismo.

También se detalla las herramientas de trabajo que se utilizaron en el desarrollo y las tecnologías relevantes como Java EE, Android y Websocket.

Para explicar el desarrollo completo de la solución, se presentan las HU que formaron parte del proyecto divididas en tres Sprint.

IV.1. Planificación

En la siguiente lista se muestra la descomposición de tareas del proyecto:

1. Gestión de Proyecto (32 horas):
    a. Realizar reuniones con el entrenador (22 horas). Todos los lunes de cada mes, se llevó a cabo una reunión en el gimnasio del Lic. Matías Scavo quien es el dueño del producto, con el fin de verificar y validar los avances del proyecto y avanzar con la toma de requisitos.
    b. Documentar objetivos y redacción de la introducción (10 horas). Durante las dos primeras semanas se confeccionaron los objetivos que deberá cumplir la el desarrollo para satisfacer las necesidades del cliente.

2. Aplicación Web para los Entrenadores (110 Horas):
    a. Análisis, en esta se etapa se realizó las siguientes actividades:
        i. Identificación los participantes del sistema,
        ii. Establecimiento el catálogo de requisitos, y
        iii. Creación los diagramas de casos de uso.
    b. Diseño, en esta se etapa se realizó las siguientes actividades:
        i. Elaboración los prototipos de la nueva aplicación,
        ii. Establecimiento de cómo deben almacenarse los datos,
        iii. Creación el diagrama de clases y se diseñará el plan de pruebas.
    c. Implementación, en esta se etapa se realizó las siguientes actividades:
        i. Se crearon las interfaces de usuario,
        ii. Se construyó las capas según el modelo M.V.C.[8], y
        iii. Se desarrolló los servicios de comunicación con la aplicación móvil.
    d. Prueba: Se realizaron pruebas unitarias de los módulos construidos y pruebas de integración.
3. Aplicación Móvil para los Atletas (90 Horas):
    a. Análisis: Se analizará la situación de la aplicación y se establecerán una serie de directrices para el desarrollo.
    b. Diseño: Se establecerán criterios para realizar las interfaces de la aplicación. Se implementara la comunicación del Móvil con los sensores.

[8] M.V.C. Patrón de Diseño Model View Controller.

c. Implementación: Se desarrollarán los servicios y las actividades. También se crearán las interfaces de configuración.
d. Prueba: Se realizarán pruebas unitarias de los módulos construidos y pruebas de integración.

Duración Total estimada: 232 Horas.

IV.2. Seguimiento del Proyecto

El seguimiento del proyecto se realizó a través de las herramientas de Google Drive y Trello. La fundamentación de la elección está basada en que cumplen con los requisitos básico de la metodología Scrum. Trello permite realizar los tableros propios de la metodología (Henrik Kniberg, 2007). Las mismas son aplicaciones basadas en web para que permitan el seguimiento y documentación del proyecto.

Se destaca que al utilizar scrum en este proyecto se realizó una adaptación del mismo de acuerdo al estilo del proyecto y el equipo del trabajo.

Los roles para este proyecto se dividieron de la siguiente manera:

- Scrum Master y Team (director del proyecto): Muñoz Abbate, Horacio (desarrollador Java EE y Android).
- Product Owner (dueño del producto): Lic. Scavo Matias (Conocedor del dominio).

Durante los dos primeros días se estableció la pila del producto y las características de los sprints. Se decidió que estos tuviesen una duración de 2 semanas. A continuación se muestra una breve descripción del desarrollo del proyecto por cada sprint:

- Sprint 0
    - Definición de Arquitectura

- Sprint 1
    - HU.1-1 alta de usuarios
    - HU.1-2 modificación de usuarios
    - HU.1-3 eliminar y baja de usuarios
    - HU.1-4 listar usuarios filtros y paginado
    - HU.5 crud de provincias
    - HU.6 crud de ciudad
    - HU.3 crud de atleta
    - HU.4 crud de actividad y lap
- Sprint 2
    - HU.12 login web
    - HU.11 cambio de clave
    - HU.8-1 websocket actividad y atleta
    - HU.8-2 manager de actividades
    - HU.8-3 desarrollo de entrenamiento
    - HU.9-1 conexión ble pulsometro
    - HU.9-2 servicio cliente websocket
    - HU.9-3 interface de monitoreo de la actividad
    - HU.8-4 timer y tareas asíncronas
    - HU.10 configuración de la aplicación móvil
- Sprint 3
    - HU.13 gráfico de resultado del entrenamiento
    - HU.14 gráfico del mapa del recorrido del entrenamiento
    - HU.15 análisis de recuperación

IV.3. Herramientas de trabajo y Tecnologías

IV.3.1. Sistema de control de versionamiento de Código

Para versionar el código se decidió por Subversion[9]. La decisión está basada fundamentalmente porque cumple con los requisitos de tener un control y resguardo del código, pero también por la experiencia previa del tesista en esta herramienta.

IV.3.2. Entorno de desarrollo

Para desarrollar aplicaciones móvil, se utilizó el entorno que Google propone, Android Studio, basado en Intellij Idea (Google, 2015). Desde el 2015 tiene su primera versión estable 1.0 y actualmente con una versión beta 2.2 que funciona correctamente para los fines del proyecto.

Google además del IDE, ofrece una SDK[10] que permite desarrollar cualquier tipo de aplicación para Android. Para la generación del proyecto y manejo de dependencias elegí Gradle.

Para el desarrollo de la aplicación Java EE[11], utilice el IDE Intellij Idea 2016, actualmente una versión de prueba y para la gestión del proyecto Gradle. Utilizar el mismo entorno de desarrollo para ambas aplicaciones facilitó el desarrollo.

IV.3.3 Tecnologías y Arquitectura del Sistema

Dentro de la metodología Scrum, se puede definir un Sprint 0, en el cual se determinan cuestiones de tecnologías y arquitectura de software que el sistema podría requerir para la implementación (Anurag Prakash, 2013).

---

[9] Subversion, también llamado SVN.

[10] SDK Significa Kit de Software de Desarrollo.

[11] Java EE Edicion Empresarial.

Existen requerimientos no funcionales que pueden convertirse en críticos, y afectar el resultado del sistema, como puede ser la comunicación de los atletas y el entrenador, que necesita una implementación que garantice un ancho de banda mínimo y tiempos de respuesta bajos. Este requisito, establece un punto crítico en el desarrollo e implementación que debe ser evaluado y testeado en el principio del proyecto.

Dentro de los estándares y especificaciones que utilice para el desarrollo de la tesina que facilitan la implementación de WebSocket en el contexto de una arquitectura de JavaEE es la JSR[12]-356. Esta especificación detalla cómo implementar WebSocket en el contexto antes mencionado.

En el sitio web para desarrolladores de Android, se presenta toda la documentación para desarrollar sobre la interfaz Bluetooth (Google, 2015).

Sprint 0

Dentro de este Sprint, se analizó la comunicación del pulsómetro con el celular. Para ello se utilizó la interfaz Bluetooth en su versión 4.0 LE.

En la figura IV.1. se puede ver un esquema de arquitectura de los elementos principales en la comunicación del sistema.

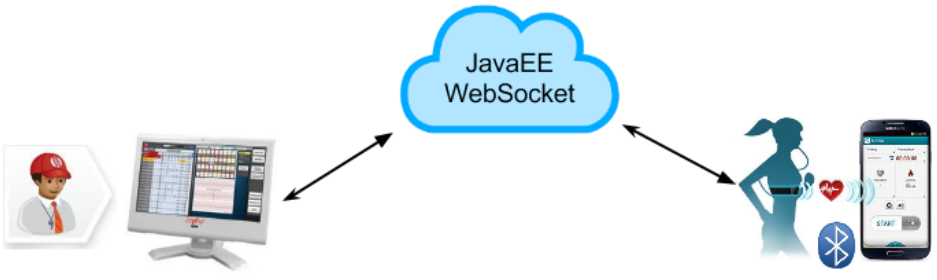


Fig.IV.1. Esquema de conexión producto del Sprint 0.

[12] JSR significa solicitud de especificación java del inglés Java Specification Request

IV.4. Desarrollo de la solución

Para el desarrollo de la solución se utilizaron los siguientes artefactos que provee la metodología ágil Scrum y UML:

- Diagrama de Casos de Uso.
- Diagramas de Clase.
- Historias de Usuario.
- Tablero de Tareas.

IV.4.1. Diagrama de caso de usos

Para un entendimientos de los componentes globales y los actores del sistema, se elaboró el siguiente diagrama de caso de uso del nivel 0, también llamado diagrama de contexto como se ve en la Fig. IV.2.

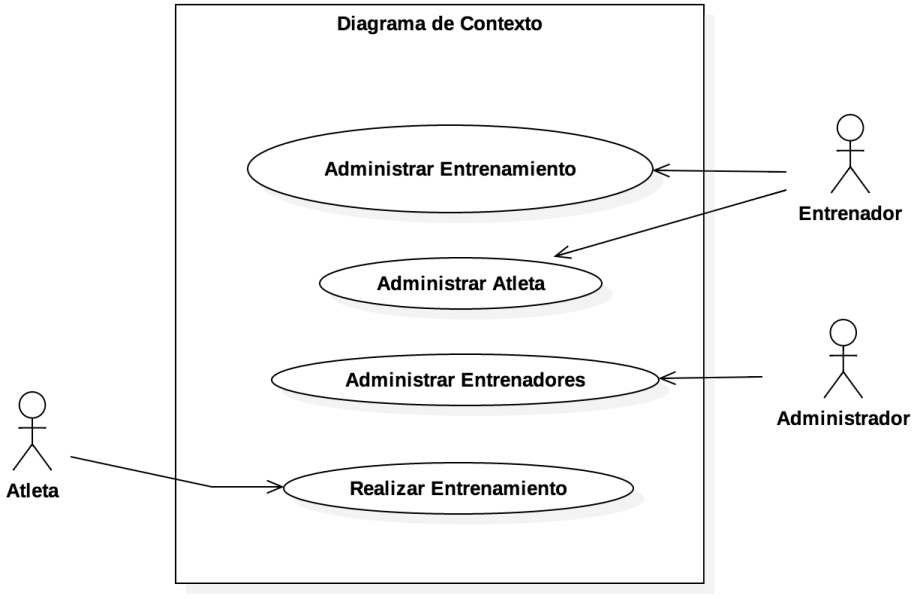


Fig.IV.2. Diagrama de Contexto

IV.4.2. Historias de Usuario

Las historias de usuario (HU) va a tener el formato que sugiere la bibliografía consultada que se tomó para la documentación del desarrollo. En la figura IV.3 se puede ver un ejemplo donde define las siguientes columnas:

- ID: Identificador.
- Nombre: Un nombre descriptivo.
- Imp: Importancia, valor numérico de 0 a 100 que establece la importancia del negocio.
- Est: Estimación, un valor numérico entero, que representa el número de días que llevaría desarrollar la HU (un dia esta compuesto de 8 horas de un recurso).
- Cómo probarlo: Una descripción de cómo probar el sistema para comprobar que hace lo que el dueño del producto desea.
- Notas: Cualquier observación o comentario que puede ayudar al desarrollo de la tarea.

Pila de Producto (ejemplo)

ID	Nombre	Imp.	Est.	Como probarlo	Notas
1	Depósito	30	5	Entrar, abrir página de depósito, depositar 10€, ir a página de balance y comprobar que se ha incrementado en 10€	Necesita un diagrama UML. No preocuparse por encriptación aun
2	Ver tu historial de transacciones	10	8	Entrar, ver transacciones. Realizar un depósito de 10€. Ir a transacciones y comprobar que se ha actualizado con el nuevo depósito	Utilizar paginación para no hacer consultas muy grandes a la BB.DD. Diseño similar a la página de usuario.

Fig.IV.3. Ejemplo de formato de las Historias de usuario.

Cada HU puede descomponerse en una o varias tareas para implementar la funcionalidad descrita.

Para comenzar con este procedimiento, inicié una HU tipo CRUD, acrónimo de Create, Read, Update y Delete (en castellano crear, leer, modificar y eliminar), la cual va a tener mucha funcionalidad similar a otras HU del mismo tipo.

En esta primera instancia, generalice lo máximo posible, lo cual me permitió realizar todos los CRUD de forma mucho más ágil. Para el desarrollo de esta tesina se toma como criterio no generar tareas

estrictamente de tecnología, excepto cuando agreguen valor al negocio. Estas actividades, como pueden ser la implementación del patrón DAO[13] o estilos para la vista web, van a estar incluidos dentro de las tareas de desarrollo como podría ser: realizar el listado de los usuarios de sistema.

Sprint 1

Para el seguimiento del Sprint, se implementó el siguiente tablero de Trello https://trello.com/b/OSVACRU2/spring-1, en el cual se pueden ver en detalle todas las tareas de las HU de la iteración. En este Sprint se desarrollaron la mayoría de los CRUD del sistema.

HU. 1 - CRUD de Usuario

Detalle de la H.U.:

- **ID**: 1
- **Nombre**: CRUD de Usuario.
- **Importancia**: 50
- **Estimación**: 2
- **Como probarlo**: Crear, ver, modificar y eliminar Usuarios.
- **Nota**: Usuario del sistema, tendrán username y clave para autenticarse en el sistema.

Esta HU es un CRUD con un esfuerzo de 2 días de trabajo. En el desarrollo de esta actividad se realizó una estructura genérica para todos los futuros CRUD del sistema. En la implementación de la HU 1 surgieron tareas que fueron creadas en el tablero Scrum para el Spring 1 como se puede ver en la figura IV.4.

[13] DAO patrón de diseño, acrónimo de Data Access Object.

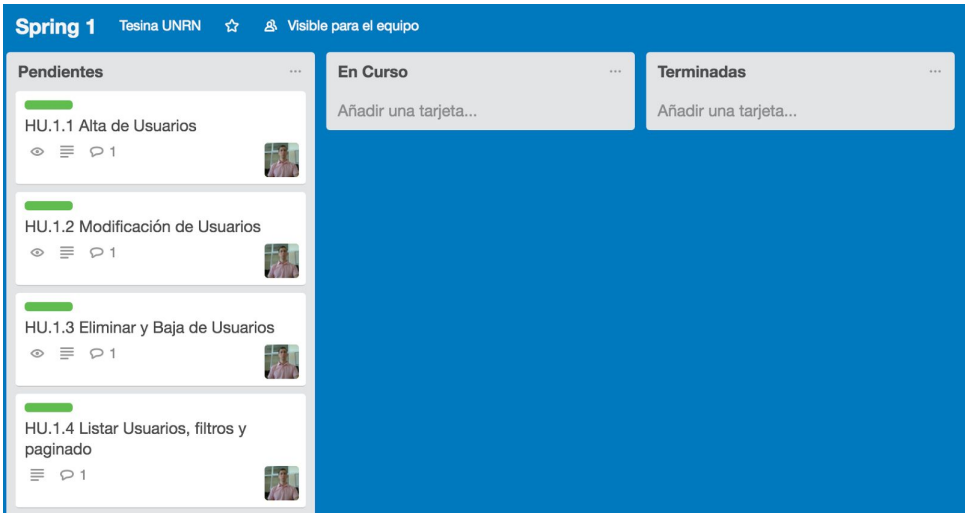


Fig.IV.4. Tareas de la HU 1 del Tablero Scrum.

HU.1.1 Alta de Usuario

El detalle de la tarea HU.1.1 se puede ver en la figura IV.5 y acceder mediante la URL: https://trello.com/c/MzCxISuo/1-hu-1-1-alta-de-usuarios. En la misma se detalla con más precisión los requisitos de la tarea y mediante progresa el desarrollo de la tarea, se incorpora más información, comentarios y documentación.

Fig.IV.5. Detalle de la Tarea Alta Usuario de la HU 1.1

Para la implementación de esta HU y todas todas las demás, utilice el patrón de diseño MVC[14]. Esta arquitectura nos permite separar los datos y

[14] MVC Model View Controller patrón de Diseño.

la lógica de negocio de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones, también llamado Controlador. En función de las características de este patrón comenzaremos a construir el modelo y la lógica de negocio en primer lugar.

Modelo y lógica de negocio

El primer paso para realizar esta tarea fue crear el modelo que representa un usuario mediante una conjunto de Clases y la API[15] de persistencia de Java conocida como JPA[16]. En la figura IV.6 se puede ver un diagrama de clase donde se representan todos los componentes necesarios para la implementación. Se describen las anotaciones utilizadas como `@Entity` y `@Table`, que se utilizan para definir que la clase será un Objeto persistente y que tabla de la base de datos se asociará. También `@XmlRootElement` que permite transformar una instancias de la clase al formato Json[17].

La clase usuario hereda de `BaseEntity`, esto me permitió generalizar las implementaciones de las demás entidades, dado que todas las entidades van a tener una clave `ID` de tipo `Long`, con sus `getter y setter`, además de la implementación de `Iqual y hashcode`, métodos que nos permiten comparar instancias de objetos por el atributo `ID`.

Se creó la clase `Role` que es un enumerado, la cual cuenta con los roles de administrador, atleta y entrenador.

[15] API acrónimo en Inglés de Application Programming Interface

[16] JPA significa API de persistencia Java.

[17] Json acrónimo en Inglés de JavaScript Object Notation.

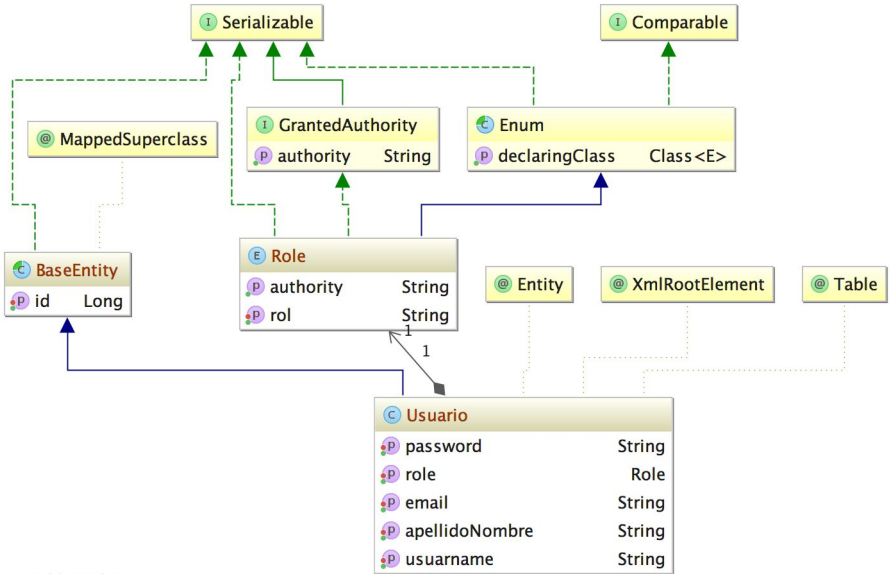


Fig.IV.6. Implementación de la Clase Usuario y Rol con JPA.

Utilizando JPA, no necesite generar script para las tablas y relacione de base de datos. En la misma configuración, se declara que JPA cree las tablas y las modificaciones que se puedan hacer de forma automática.

El próximo paso fue generar el DAO y servicio para persistir la clase usuario. En el mismo se construyó una generalización que nos permite reutilizar código y avanzar más rápido en la implementación de las demás clases.

En la figura IV.7 se presenta un diagrama de clase con los elementos que conforman la implementación genérica de DAO.

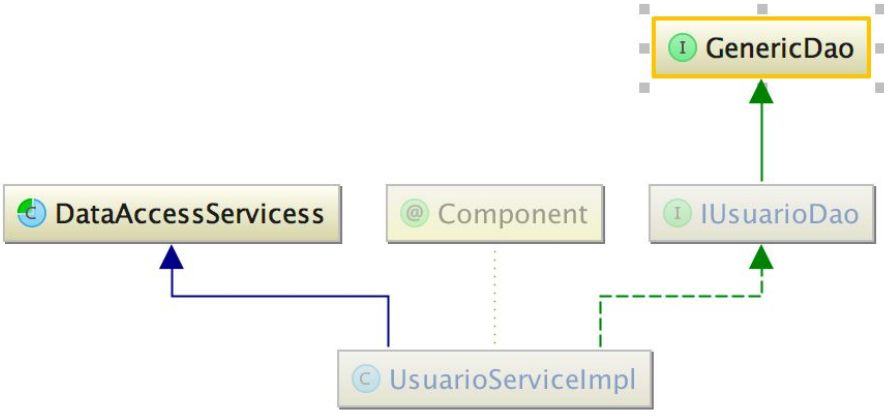


Fig.IV.7. Implementación de DAOs y Servicios Generalizados.

Básicamente se implementa un interfaz genérica, con los métodos de un CRUD. A continuación se puede ver el código fuente de la implementación descrita anteriormente:

```
public interface GenericDao<T>
  public T create(T t)
  public T find(BaseEntity id)
  public void delete(BaseEntity id)
  public T update(T item)
  public List<T> findAll()
```

La implementación de los métodos está en la clase `DataAccesServices`, en la cual mediante el framework Spring[18] se inyecta el `EntityManager`[19], el cual nos permite manejar todas las entidades que estén dentro del contexto de la aplicación (Todas las clases que estén dentro del paquete `unrn.isiii.model` con la anotación `@Entity`)

```
public abstract class DataAccesServices<T>
@PersistenceContext
private EntityManager em;

  @Transactional
  public T create(T t) {
     this.em.persist(t);
     return t; }
  @Transactional(readOnly  = true)
  public T find(Object id) {
     return this.em.find(this.type, id);}
  @Transactional
  public void delete(Object id) {
     Object ref = this.em.getReference(this.type, id);
     this.em.remove(ref); }
  @Transactional
  public T update(T item) {
     return (T) this.em.merge(item); }
  @Transactional(readOnly = true)
```

[18] Spring Framework que permite implementar los patrones de IoC y DI en Java EE

[19] EntityManager es una implementación de Java que permite el manejo de Entidades.

```
  public List<T> findAll() {
     return this.em.createQuery("select o from " +
this.type.getName() + " o").getResultList();
```

El último elemento es el servicio, el cual hereda toda la implementación de un CRUD y además posee métodos propios que se van creando mediante se avanza el desarrollo. En el caso de `UsuarioServiceImpl`, además de los métodos heredados, también se implementó el método `findByUserName`, que permite recuperar un usuario mediante el nombre de usuario.

```
@Component
public class UsuarioServiceImpl
extends DataAccesServices<Usuario> implement  IUsuarioDao {
  public Usuario findByUserName(String username) {
     return (Usuario) getEm()
           .createQuery("select o from Usuario o "
                 + "where o.usuarname = :username")
           .setParameter("username", username).getSingleResult();
```

En la clase se puede ver la anotación `@Component`, esta permite agregar la clase dentro del contexto de Spring para ser inyectado después en otros componentes.

Una vez finalizado el modelo y el servicio, podemos implementar el controlador. La nomenclatura utilizada para los controladores fue `<NombreEntidad>Bean`[20], para el caso de usuarios definimos la clase `UsuarioBean`. En la figura IV.8 se presenta un diagrama de clase en el cual se puede ver la relación del controlador con el servicio o modelo. Esta relación se establece mediante componentes de Sprint.

[20] JavaBean o Bean, es una clase Java que cumple con ciertas normas con los nombres de sus propiedades y métodos.

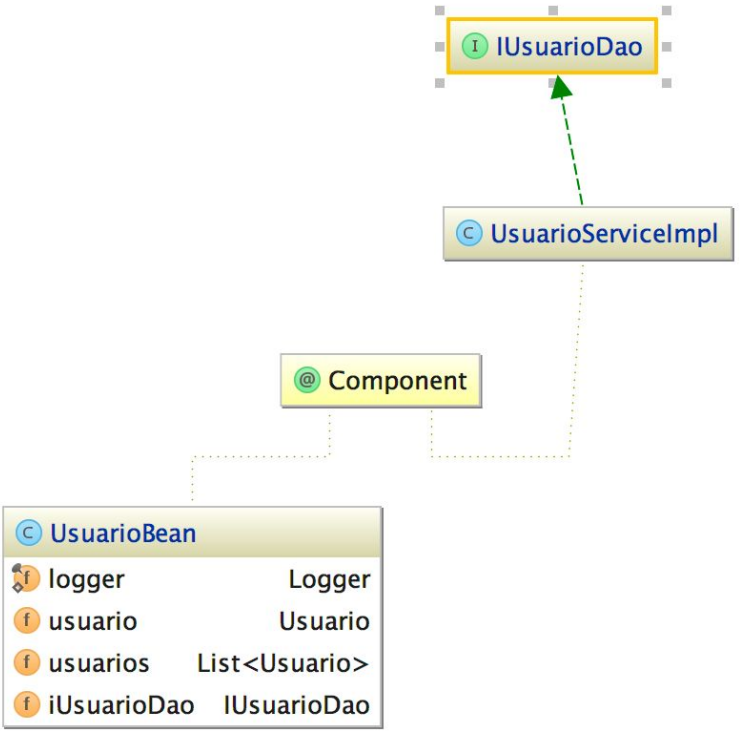


Fig.IV.8. Diagrama de Clase del controlador y la relación con el servicio

Para el caso de la HU.1.1 Alta de Usuario, se implementó el método save que permite persistir una instancia del objeto usuario, a continuación se puede ver el código que implementa esta funcionalidad:

```
@Component
public class UsuarioBean {
private Usuario usuario = new Usuario();
@Autowired
private IUsuarioDao iUsuarioDao;

public void save() {
  if (usuario.getId() == null)
     getiUsuarioDao().create(usuario);
  else
     getiUsuarioDao().update(usuario);
      FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_INFO,
"Alta/Modificación", "Operación realizada con éxito.");
```

El método `save`, también nos sirve para la modificación, siempre que el `ID` de usuario sea distinto de `Null`. Cuando la operación es exitosa, envía un mensaje a la vista para que informe al usuario el resultado de la operación.

Para la vista utilicé JSF, un framework para aplicaciones Java web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE, por lo cual las paginas seran archivos JSP, es una tecnología que ayuda a los desarrolladores de software a crear páginas web dinámicas. Como componentes visuales seleccione Primefaces con el estilo Modena, que ofrece una amplia gama de elementos visuales como listas paginadas, filtros, listas desplegables, mensajes de información. Además implementa el diseño completo de la aplicación como, menús desplegables, diseño adaptativo, estilos de fuentes, páginas de login y errores comunes cómo 404.

En la figura IV.9 se puede ver el formulario de Alta de Usuario y como se puede adaptar a diferentes pantallas.

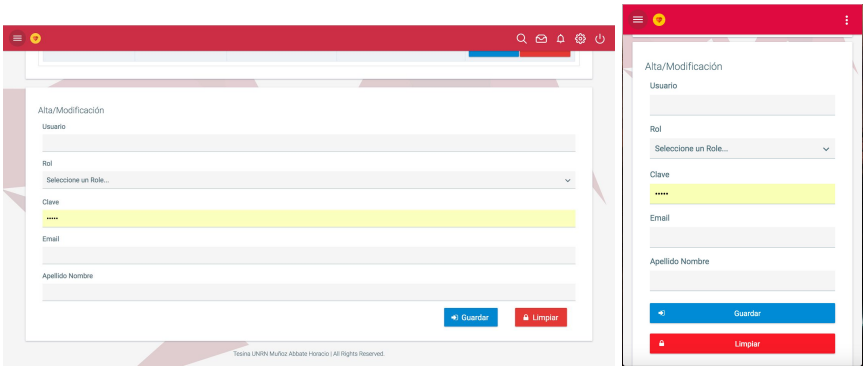


Fig.IV.9. Formulario de Alta de Usuario, adaptable para móviles

La integración de la vista y el controlador se expresa de la siguiente forma dentro de los archivos de vista usuario.xhtml:

```
<p:outputLabel value="Usuario" for="provinciaDesc"/>
<p:inputText value="#{usuarioBean.usuario.username}" id="provinciaDesc"/>
<p:message for="provinciaDesc"/>

<p:commandButton action="#{usuarioBean.save}" value="Guardar"
update=":formLista:lista :alta" icon="fa fa-sign-in"/>
```

El componente `p:inputText` se relacione con el controlador mediante la propiedad `value = #{usuarioBean.usuario.username}`. El valor ingresado en esta entrada de texto será guardada en una instancia del objeto usuario en su propiedad `username`, que está en del controlador. Una vez que se completa el formulario, se guarda la instancia de usuario a través de la `action guardar` mediante el componente `p:commandButton` y el atributo `action` que establece la relación con el método `save` del controlador.

En la figura IV.10 se puede observar las validaciones de los atributos de la clase usuario cuando se quiere guardar sin ingresar valores.

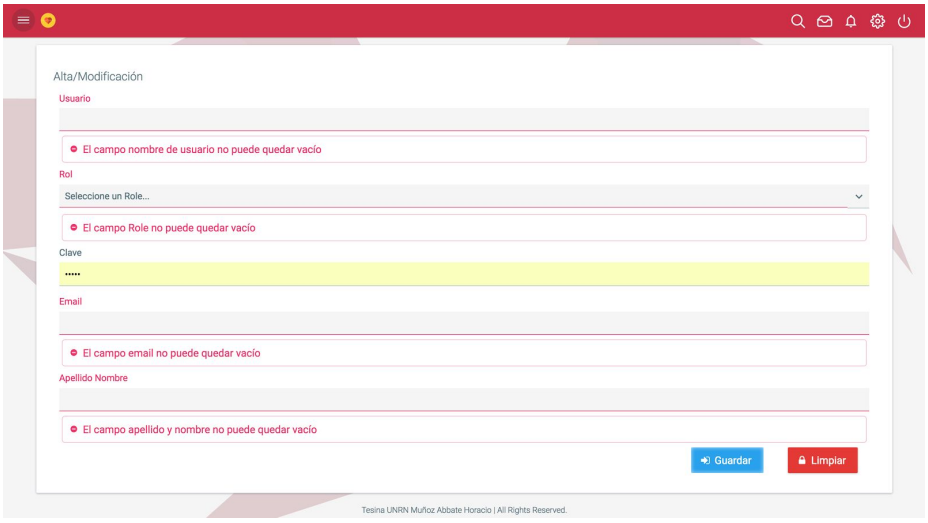


Fig.IV.10 Validaciones extraídas del Modelo.

La validación de los datos está presente en todo formulario, desde la capa de presentación hasta la capa de persistencia. A menudo la misma lógica de validación se implementa en todas las capas, lo que nos lleva mucho tiempo y aumenta la posibilidad de errores. Para evitar la duplicación de estas validaciones en cada capa surge la especificación JSR-303 (Emmanuel Bernard, 2013) que define un modelo de metadatos y una API para la validación de JavaBean. Esta metadata se configura sobre el

modelo mediante anotaciones, en el siguiente fragmento de código se puede ver dos tipo de validaciones implementadas mediante dicha especificación.

```
public class Usuario extends BaseEntity {
   @NotEmpty(message = "El campo nombre de usuario no puede quedar vacío")
  private String username;

  @NotEmpty(message = "El campo email no puede quedar vacío")
  @Email(message = "El formato del email es inválido")
   private String email;
```

Los atributos `username` y `email` poseen la anotación `@NotEmpty` con un mensaje definido por el desarrollar cuando el valor es vacío, y para el atributo `email` tambien tiene la validación de formato válido con la anotación `@Email`.

Test de Unidad e Integración de Alta de Usuario

Para el el caso de la HU.1.1 no se realizó ningún test de unidad, dado que no había ninguna funcionalidad unitaria a evaluar, pero si se genero un test de integración. Este test tiene la posibilidad de probar varios componentes del sistema, para este caso se probó: Base de Datos, DAO y servicio. En la figura IV.10 se puede ver el test `UsuarioTest.class`, donde tiene dos pruebas evaluadas de forma exitosa: buscar usuario por nombre de usuario y alta de usuario.

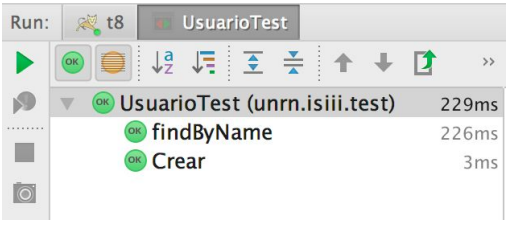


Fig.IV.10. Resultado del test de unidad de la HU.1.1

Cabe mencionar que Spring permite crear este tipo de test, en los cuales podemos inyectar cualquier componente de igual manera que en la aplicación, con la posibilidad de configurar distintas conexiones de base de datos; Toda la configuracion esta dentro del archivo `test-context.xml`.

En el fragmento de código siguiente se puede ver como se inyecta la interfaz del servicio de Usuario.

```
@ContextConfiguration("file:src/test/resources/test-context.xml")

@RunWith(SpringJUnit4ClassRunner.class)
@Transactional
public class UsuarioTest {
@Autowired
private IUsuarioDao iUsuarioDao;
 private Usuario usuario;
@Before
public void init()
  {
setUsuario(new Usuario("admin","123123","email@com.ar","Horario",
Role.ROLE_ADMIN));
  }

  @Test
  public void Crear() {
     //Alta de Usario
     getiUsuarioDao().create(getUsuario());
     //Commit a la Base
     getiUsuarioDao().utils(getUsuario());
Assert.assertNotNull(getiUsuarioDao().find(getUsuario().getId()));
  }

  @Test
  public void findByName() {
     getiUsuarioDao().create(getUsuario());
     getiUsuarioDao().utils(getUsuario());

Assert.assertNotNull(getiUsuarioDao().findByUserName("admin"));
```

Con la anotación `@Before` preparamos el test antes de ser ejecutado, en mi caso creo una instancia de usuario de forma correcta. Se ejecuta el test, ya sea el de `crear` y el de `findByName` y por último se comprueba el resultado del test.

HU.1.3 Lista de Usuario

El detalle completo de la tarea se puede acceder en el siguiente recurso https://trello.com/c/x8givLEu/4-hu-1-4-listar-usuarios-filtros-y-paginado, en la cual especifica la implementacion de un listado de usuarios con paginado y filtros segun atributos caracteristicos de la entidad como nombre de usuario, apellido, nombre y rol.

En la historia anterior se implementó gran parte todos los componentes de software y estructura que me permitieron crear esta tarea con poco esfuerzo. En el momento de comenzar con esta tarea ya tengo el modelo, capa de servicios, controlador y parte de la vista.

Solo nos queda agregar en el controlador un atributo de tipo lista de usuarios y un método para cargar la lista de usuario llamando al servicio, en el siguiente fragmento de código se puede observar los componentes mencionados.

```
private List<Usuario> usuarios;
@PostConstruct
public void init(){
  this.usuarios = getiUsuarioDao().findAll();
```

La annotation `@PostConstruct` permite invocar el método `init()` al crearse el componente UsuarioBean, en esta etapa de desarrollo le adicione la anotación `@scope("view")`, lo que me permite que el controlador tenga alcance de vista, todas las instancias de los objetos van a permanecer en el servidor siempre y cuando no se cambie de página.

Para terminar esta tarea solo nos queda implementar la vista, para ello utilice un componente visual de Primefaces de lista, que me facilita el paginado y los filtros por atributo, en el siguiente código se puede ver la implementación de la misma:

```
<p:dataTable id="lista" value="#{usuarioBean.usuarios}"
var="v_user" paginator="true" rows="10">
   <p:column headerText="Usuario" sortBy="#{v_user.username}"
             filterBy="#{v_user.username}">
       #{v_user.username}
   </p:column>
```

`p:dataTable` es el componente que permite renderizar una lista, en el atributo `value` que vincula con la lista del controlador de usuario. También se puede ver una columna, con el titulo Usuario la cual se va a ordenar y filtrar mediante el atributo `username`.

HU.1.2 Modificación de Usuario

Con las HU.1.1 y HU.1.4 tengo gran parte del desarrollo desarrollado. En la siguiente URL se puede se puede ver el detalle de la tarea https://trello.com/c/3s3iTOrR/2-hu-1-2-modificacion-de-usuarios.

Lo que falta para terminar esta tarea es agregar un botón en la lista de la tarea HU.1.4, para modificar una instancia de Usuario en la vista realizada en la tarea HU.1.1.

En la figura IV.11 se puede ver la lista con los botones de editar y eliminar, para esta tarea sólo se implementó la funcionalidad de editar.

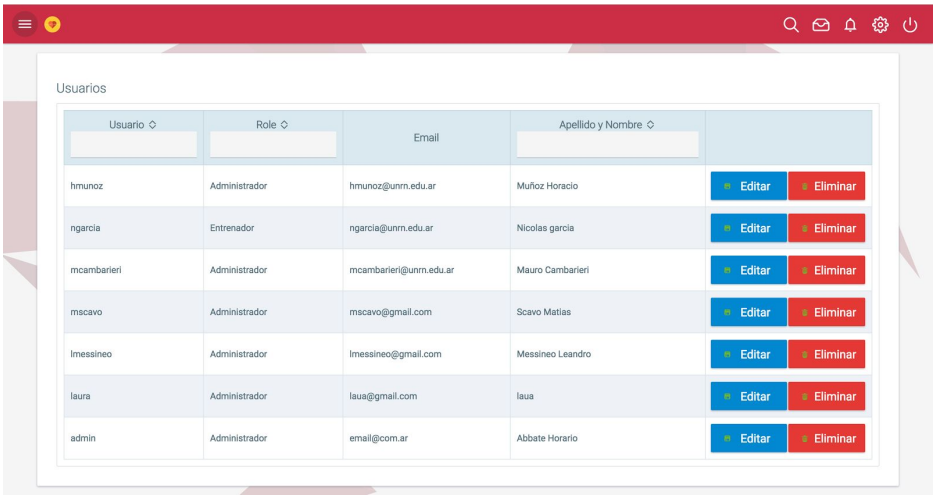


Fig.IV.11. Lista de usuario con los botones de Editar y Eliminar

La acción de editar envía una instancia de Usuario al controlador mediante el siguiente botón:

```
<p:commandButton
actionListener="#{usuarioBean.seleccionar(v_user)}"
value="Editar" />
```

Con la instancia en el Controlador y la vista que se creó en la HU.1.1 ya muestra los valores para editar. Al tener ID asignado, cuando se graba la modificación utiliza el método de modificación del servicio.

En la figura IV.12 se puede ver la ejecución del test de Modificar un Usuario. En el test de persistir un nuevo usuario, después se lo recupera de la base se realiza una modificación, se guardan los datos a la base. Para hacer la verificación se comprueba que recuperando esa instancia de usuario de la base tenga la modificación. En el siguiente fragmento de código se puede ver la comparación final del test.

```
Usuario usuario = getiUsuarioDao().find(getUsuario().getId());

Assert.assertEquals("Texto Modificado", usuario.getApellidoNombre());
```

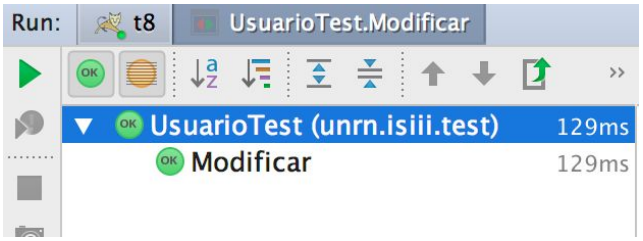


Fig.IV.12. Test de Ingreacion de Modificar un Usuario

HU.1.3 Eliminar y baja de usuario

En el siguiente enlace se puede ver el detalle completo de la tarea https://trello.com/c/Z70ku0nd/3-hu-1-3-eliminar-y-baja-de-usuarios. En la misma hace diferencia de la acción de eliminar y dar de baja un usuario.

La acción de eliminar el usuario, borra el usuario de la base siempre que no tenga relaciones con otras entidades. La baja, hace una modificación de una instancia de usuario que demuestre que un usuario está deshabilitado para operar el sistema, sin borrar ningun datos.

En la capa de negocio el método de eliminar y modificar ya está implementado. Para terminar esta tarea falta los métodos de eliminar y baja en el controlador, los botones en la vista y los test de integración y unidad de las operaciones propias de la tarea.

El siguiente método, `eliminar`, pertenece al UsuarioBean, al cual se le pasa como parámetro el `ID` del usuario a eliminar. Si al eliminar el usuario tiene relaciones con otras entidades, la excepción del motor de persistencia Hibernate está programada para capturarla e informar al usuario.

```
public void eliminar(Long id) {
try {
   getiUsuarioDao().delete(id);
   FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_INFO,
           "Borrar", "Operación realizada con éxito.");
     FacesContext.getCurrentInstance().addMessage(null, msg);
```

```
        } catch (HibernateJdbcException e) {
        FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_ERROR,
        "Borrar",
"El usuario tiente relaciones activas. Elimine primero los sitios.");
         FacesContext.getCurrentInstance().addMessage(null, msg);
       }
```

Para dar de baja agregue un atributo booleano de nombre activo dentro de Usuario, este atributo se puede modificar como cualquier otro campo en la HU.1.2.

El test para evaluar la tarea se representa en el siguiente código:

```
@Test
public void eliminar() {
  //Alta de Usario y Commit
  getiUsuarioDao().create(getUsuario());
 //Elimino usuario por ID
  getiUsuarioDao().delete(getUsuario().getId());
  //Pruebo que el usuario no exista
  Assert.assertNull(getiUsuarioDao().find(getUsuario().getId()));
```

Esta prueba consiste en dar de alta un usuario, después eliminarlo y por último hacer la comprobación de que no esté en la base.

HU Tipo CRUD

Para las demás HU tipo CRUD, no voy a desarrollar con el detalle de la HU.1, dado que la implementación es muy similar y no va a aportar ningún valor a la tesina.

En el tablero scrum, genere una tareas por cada CRUD sin dividirlas en tareas más pequeñas como en el caso de la HU.1, debido a que gran parte de la dificultad de implementar un CRUD se resolvió en esa HU.

En la figura IV.13 se puede observar el tablero y todas la tareas terminadas.

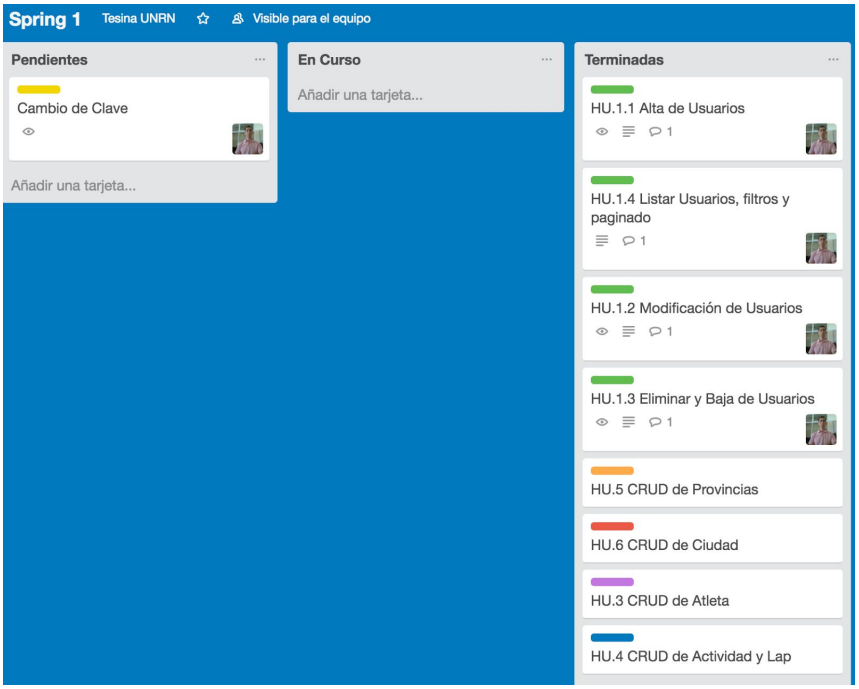


Fig.IV.12. Tablero del Sprint 1 con los CRUD terminados

El diagrama de clase en este primer Sprint se puede ver en la figura IV.14, en el cual están todas las entidades que logre implementar.

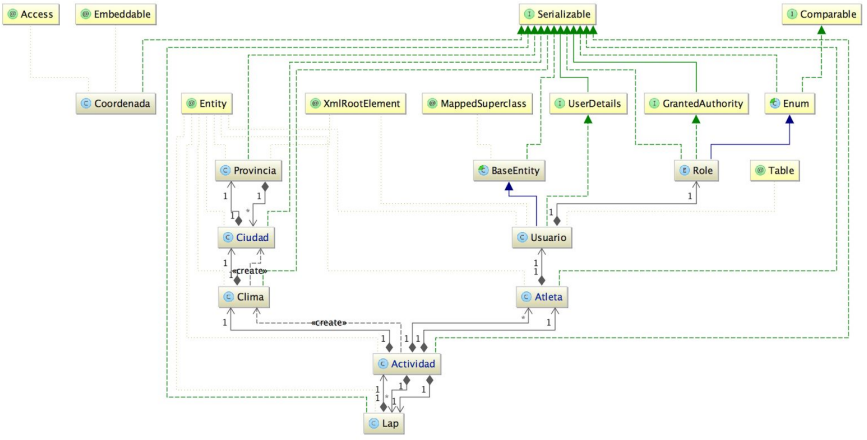


Fig.IV.14 Diagrama de Clase al finalizar el Sprint 1.

Las entidades de Provincia y Ciudad, son necesarias para tener una referencia geográfica del lugar del entrenamiento que se va a realizar. La entidad Clima nos permite en función de la ubicación seleccionada para el

entrenamiento obtener y almacenar los datos del clima como: temperatura, humedad, presión atmosférica y viento. Estos valores influyen en el desarrollo de los entrenamiento y fueron sugeridos por el dueño del producto.

La entidad Atleta nos permite gestionar a los deportistas registrados en el sistema y permite ingresar los datos de las pulsaciones por minuto (PPM) mínima y máxima, que se utilizan para calcular el porcentaje de esfuerzo del atleta durante el entrenamiento.

Para la gestión de los Actividades o entrenamientos se modelo con la entidad Actividad. Un entrenador mediante este módulo puede gestionar las actividades que planea realizar seleccionando los atletas con los que desea realizar el entrenamiento. En esta primera fase, el entrenador puede seleccionar cualquier atleta registrado en el sistema, en futuras iteraciones un entrenador solo podrá trabajar con sus atletas. También le permite seleccionar tipo de entrenamiento, el tiempo de la actividad dividida por Intervalos llamados Laps, en los cuales se le puede asignar además del tiempo las PPM mínimas y máximas, que se utilizan para informar al atleta si está por encima o debajo de lo programado por el entrenador.

Sprint 2

El siguiente Sprint se puede ver en detalle en el siguiente tablero Scrum https://trello.com/b/hELfX5Ap/sprint-2, en cual se utilizó para el seguimiento.

En este Sprint se desarrollo temas de seguridad como autenticación y también la parte fundamental de la arquitectura del sistema sistema, desarrollar el módulo que permita gestionar una actividad en tiempo real con los atletas conectados desde sus teléfono inteligentes.

HU.12 Autenticación y Login al sistema

El detalle de la HU se puede ver en la siguiente URL https://trello.com/c/f7bRJnVY/2-hu-12-autenticacion-y-login-al-sistema.

Para desarrollar esta HU se utilizó como framework Spring Security, el cual permite gestionar y configurar la seguridad de cada recurso del sistema.

En el primer Sprint implementamos las entidades de Usuario y Rol que son requeridas por este framework.

Como se explicó en anteriores HU para implementar esta HU se requiere de servicios para validar el usuario y clave, un controlador llamado LoginBean y una página para ingresar los campos antes mencionados. En la figura IV.15 se puede ver la página de login del sistema.

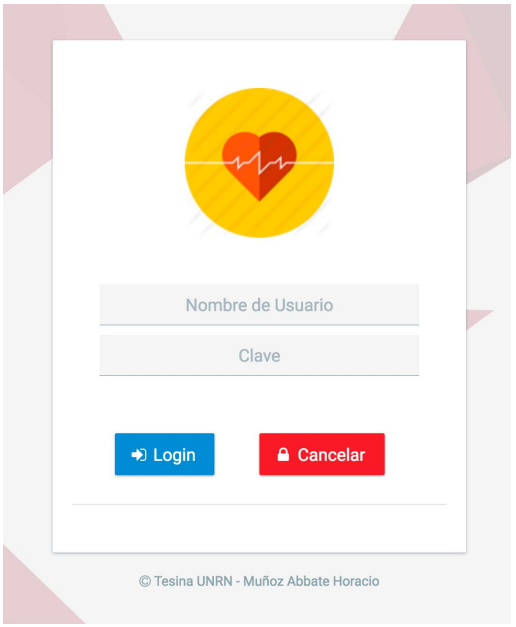


Fig.IV.15. Página de Login del Sistema.

El controlador `LoginBean` tiene la particularidad de tener alcance de sesión, esta propiedad sirve para guardar los valores durante la sesión del usuario esté activa. En el siguiente fragmento de código se puede ver la anotación correspondiente para establecer qué componente tenga alcance de sesión.

```
@Scope("session")
public class LoginBean
```

Para configurar la seguridad dentro del contexto de la aplicación, se agrego el siguiente archivo de configuración con el nombre de `security-context.xml`:

```
//Recursos que no necesitan seguridad
<security:http pattern="resources/**" security="none" />
<security:http pattern="WEB-INF/templates/**" security="none" />

<security:http>
<security:intercept-url pattern="/home.xhtml"
access="hasAnyRole('ROLE_ADMIN','ROLE_ENTRENADOR')"/>
<security:intercept-url pattern="/usuario.xhtml"
                    access="hasRole('ROLE_ADMIN')" />
<security:form-login login-page="/login.jsf" />
</security:http>
```

En este archivo se configura cuál es la página de login y los interceptores para establecer qué tipo de rol se necesita para acceder a una página o recurso. Como ejemplo, la página de inicio, home.xhtml, pueden acceder cualquier usuario que tenga el rol de administrador y entrenador, en cambio para la página de administración de usuarios solo puede acceder un usuario con rol administrador. Esta configuración queda de forma estática configurada en el archivo anteriormente descrito.

HU.8 Gestión de Entrenamiento Web

Esta HU junto a la HU.9, son las más complejas de desarrollar, dado que en ellas se implementa la parte de arquitectura con más componentes y tipo de conexiones. Por ello las dividí en varias tareas más pequeñas.

HU.8.1 Websocket Actividad y Atleta

Websocket es la tecnología que utilice para la comunicación en tiempo real. La cual permite enviar datos mediante el protocolo Http de forma bidireccional, lo que permitió conectar el atleta con el entrenador, dispositivo móvil con browser.

Para implementar el servicio de Websocket seguí la especificación JSR-356, que define el estándar para aplicaciones Java empresariales.

En la figura IV.16, se puede ver un esquema de los servicio que fueron necesarios implementar para el sistema.

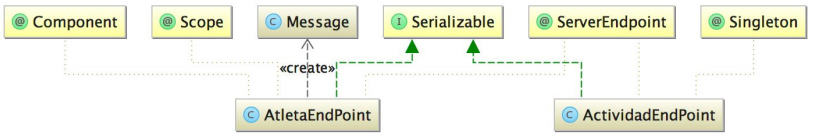


Fig.IV.16. Diagrama de Clase de la implementación de Websocket

La anotación `@ServerEndpoint` es la que define una clase como websocket. Para el sistema se implementaron dos websocket `AtlateEndPoint` y `ActividadEndPoint`. Los atletas con los entrenadores no se pueden comunicar directamente, el servidor coordina en flujo de los mensajes. La clase `Message` tiene la estructura de todos los datos que se envían por los canales de Websocket.

En el siguiente fragmento de código se implementa el Websocket para los atletas, en el cual define un parámetro usuario que nos permite identificar a un atleta en particular así poder enviarle mensajes particularmente.

La clase `MessageEncoder` nos permite codificar y decodificar el mensaje que viaja por el canal, cabe mencionar que se utilizó el formato Json. La clase Message tiene todos los atributos necesarios para seguir el entrenamiento, como ppm, posición lat y lng, velocidad y un mensaje que permitió establecer un protocolo de comunicación para los eventos que se van sucediendo.

```
@ServerEndpoint(value = "/atleta/{usuario}",
encoders = { MessageEncoder.class }, decoders = { MessageDecoder.class })
public class AtletaEndPoint implements Serializable
static Set<Session> sessions = Collections.synchronizedSet();
```

Dentro de la implementación se define una colección que contendrá todas las conexiones o sesiones activas, particularmente para este caso serán todos los atletas, conectados al servidor mediante cualquier cliente websocket, ya sea mediante una aplicación android como un browser.

Toda implementación de Websocket debe tener los siguientes métodos:

```
@OnOpen
public synchronized void open(Session session,@PathParam("usuario")
String usuario) {
        session.getUserProperties().put(usuario, true);
        sessions.add(session);
```

OnOpen, se invoca cuando un cliente inicia una conexión, para el caso de un atleta le pasa como parámetro el nombre de usuario, el cual se guarda en el objeto session y se agrega a la colección de sesiones activas. Una vez que se ejecutó este método queda establecida la conexión del cliente con el servidor, este proceso también se conoce como handshake que significa apretón de manos.

```
@OnClose
public void close(Session session,
@PathParam("username") String username){
sessions.remove(session);

@OnError
public void error(Session session, Throwable t) {
  sessions.remove(session);
```

Los métodos `OnClose` y `OnError` se utilizan para eliminar una conexión de la colección, con la diferencia que OnClose se invoca cuando se cierra una conexión de forma controlada o explícita, en cambio `OnError` se invoca cuando existe algún problema de comunicación.

```
@OnMessage
public void message(final Session session, Message msg,
@PathParam("usuario") String usuario) {
    if (msg.getMessage().equals("ppm")){
       ActividadManager().saveDato(msg);
    }
   ActividadEndPoint.send(msg);
```

`OnMessage` se invoca cuando un cliente previamente conectado, envía un mensaje. En este caso, cuando un atleta envía un mensaje de tipo ppm, pulsaciones por minuto, se guarda en la base de datos siempre que exista una actividad activa. Después se envía un mensaje a la actividad que se utiliza para ver los datos del atleta en una interfaz de seguimiento.

```
public static synchronized void send(Message msg, String usuario)
{
for (Session session : sessions) {
if (session.isOpen()&& session.getUserProperties().get(usuario)) {
            session.getAsyncRemote().sendObject(msg)
}

public static synchronized void sendAll(Message msg) {
     for (Session session : sessions) {
        if (session.isOpen()) {
           session.getAsyncRemote().sendObject(msg);
        }
     }
```

Por último, se definen dos métodos que nos permiten enviar mensajes para uno o todos los atletas que estén registrados. La propiedad synchronized permite acceder al recurso de manera exclusiva mediante un mecanismo de bloqueo.

La implementación de Websocket para las actividades, el medio de comunicación de los atletas con el entrenador se va a establecer por una actividad en particular, en el siguiente fragmento de código se puede ver la implementación.

```
@ServerEndpoint(value = "/actividad/{actividad}",
encoders = { MessageEncoder.class }, decoders = { MessageDecoder.class })
public class ActividadEndPoint implements Serializable {
```

En este caso el parámetro que se debe enviar para conectarse es el código de actividad, después la implementación tiene los mismo métodos que el websocket de atleta.

Una vez implementados los servicios de websocket, podemos conectarnos con los clientes, ya sean mediante un browser como por una aplicación Android. En las próximas tareas detallare la cómo se implementan e interactúan.

HU.8.2 Manager de Actividades

Además del bean de Actividad que forma parte del patrón MVC, para implementar esta HU vamos a necesitar un componentes que nos permita manejar todas las actividades que se creen en el sistema. Este manejador lo configure con un alcance de aplicación, esto significa que los valores que se guarden dentro no se van a perder mientras el servidor esté funcionando. Como se explicó en secciones anteriores los bean tiene alcance de vista, una vez que se cambia de página, todos los valores dentro del bean se pierden.

En el siguiente código se puede ver la definición del manejador de actividades, siendo `@Scope("singleton")` la anotación que permite tener una única instancia del componente para toda la aplicación. También se define un `Map` con las actividades que están en curso.

```
@Component
@Scope("singleton")
public class ActividadManager {
private Map<Long, Actividad> actividades;
```

HU.8.3 Temporizador y Tareas asíncronas

En esta HU se desarrollaron los mecanismos que se necesitan para controlar el transcurso de tiempo de las actividades. Cada actividad que se ejecuta consta de uno o más intervalos de tiempo llamado Lap. En cada uno se configura el tiempo de duración, en minutos, con una descripción del trabajo que el entrenador quiere realizar y las PPM máximas y mínimas.

En la figura IV.17 se puede ver cómo se configuran dos Lap con diferentes valores y el total de minutos de entrenamiento.

Fig.IV.17 Configuración de Laps dentro del CRUD de Actividades.

Una vez que se inicia una sesión de entrenamiento, el componente `ActividadManager` se encarga de administrar y controlar el tiempo por cada actividad y Lap, a través de un servicio que que se implementó con el nombre de `AsynTaskService`. Este servicio tiene una clase `ActividadTimerTask` que hereda de `TimerTask`, de Java Utils que permite implementar un temporizador.

En el siguiente fragmento de código se representa la implementación que permite crear un temporizador para una actividad. Los parámetros del constructor son el tiempo que dura la actividad, que se calcula con la suma de los Laps previamente definidos, y el identificador de la actividad.

```
public class ActividadTimerTask extends TimerTask {
  private int cuentaAtras;
  private Long actividadId;

public ActividadTimerTask(int cuentaAtras, Long actividadId) {
     super();
     this.cuentaAtras = cuentaAtras*60;
     this.actividadId = actividadId;
  }

  @Override
  public void run() {
     completeTask();
     if (cuentaAtras == 0 ){
        actividadManager.finLap(actividadId);
        this.cancel();
     }
  }

  private void completeTask() {
        if (cuentaAtras != 0){
           cuentaAtras--;
getActividades().get(actividadId).lapEnCurso().
setCuentaAtras(cuentaAtras);
ActividadEndPoint.send(newMessage("timer",cuentaAtras));
}
```

El método `run` controla que la cuenta regresiva sea igual a cero para notificar mediante un mensaje que la actividad finalizó. El método `completeTask` realiza la resta del contador y notifica el avance del contador para que el entrenador tenga la información de la cuenta regresiva en pantalla.

En la figura IV.18 se ve la pantalla que tiene el entrenador para ver el progreso de la actividad, y como avanza el tiempo por cada Lap que previamente se configuró.

Fig.IV.18. Progreso de Tiempo Vista entrenador.

HU.8.4 Desarrollo del Entrenamiento

La figura IV.19 muestra la interfaz del seguimiento de la actividad, esta pantalla la utiliza el entrenador para el control del entrenamiento.

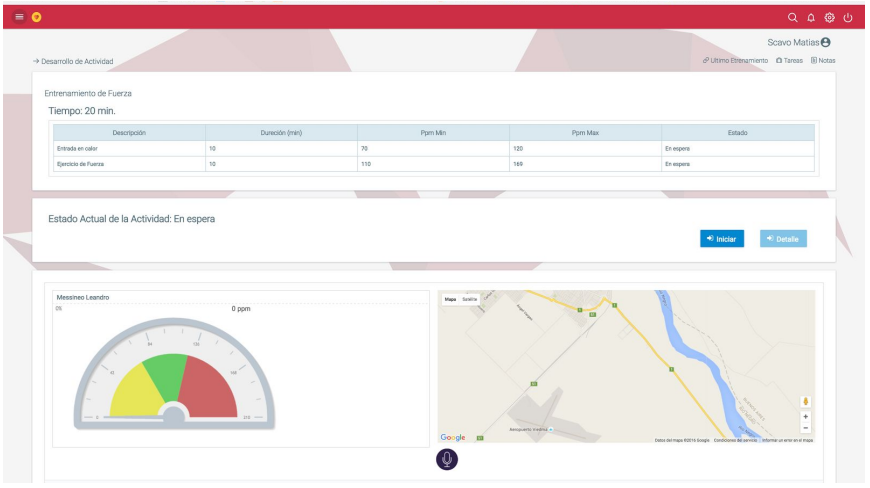


Fig.IV.19 Seguimiento de Entrenamiento

Aquí también se puede ver el detalle de la actividad y tiempo total de entrenamiento, además de todos los atletas que previamente se configuraron para la actividad.

Por cada atleta, la pantalla ofrece información en dos paneles:

- Datos de las pulsaciones y las zonas de entrenamiento configuradas para el Lap de entrenamiento actual, y
- Posición geográfica del atleta y recorrido que realiza durante el entrenamiento.

La actividad tiene implementado un atributo de tipo enumerado que permite modelar los estados posibles de una actividad. En el siguiente fragmento de código se puede ver la implementación del enumerado `Estado`:

```
public enum Estado {
 EN_ESPERA("En espera"),
INICIADO("Iniciado"),
FINALIZADO("Finalizado");
  private String estado;
  private Estado(String estado) {
     this.estado = estado;
  }
```

Los estados en espera, iniciado y finalizado, permitió modelar el ciclo de cada actividad. Cuando una actividad se crea, inicia con el estado `EN_ESPERA`. En este estado la única opción posible es iniciar la actividad, es el momento previo al comienzo de la misma. Cuando el entrenador elige una actividad para trabajar, sólo puede seleccionar aquellas que estén en este estado. Una vez que realizado esto, notifica a todos los atletas que se encuentren conectados que la actividad está por comenzar, la forma de notificación ha sido puesta en funcionamiento con WebSocket. En el siguiente fragmento de código se muestra el envío de mensaje a todos los atletas de una actividad la cual se seleccionó para empezar a trabajar.

```
for (Atleta atleta : actividad.getAtletas()) {
AtletaEndPoint.send(new Message(
     Actividad.Estado.EN_ESPERA,
     artividadId(),
     atleta.getUsuario()));
}
```

Una vez que el entrenador puede ver a todos los atletas conectados en la pantalla previamente descrita, puede iniciar la actividad. Al comenzar la actividad se cambia al estado de `INICIADO`. Para informar a los atletas se envía un mensaje a todos ellos como indicando el cambio de estado que se pasa como parámetro. La actividad va a permanecer en este estado hasta que transcurra el tiempo que tiene configurada la misma de forma automática. Una vez concluida la actividad cambia al estado `FINALIZADO`, este evento es informado de forma similar a que los estado anteriores.

En el transcurso de la actividad se guardan todos los datos de los atletas que provienen desde la aplicación en Android, por medio del canal Websocket del Atleta. Los atletas conectados envían un mensaje con los datos de PPM, velocidad y posición (latitud y longitud). En el siguiente fragmento de código se puede ver el mensaje que llega desde el cliente con la palabra clave “ppm”, que identifica un mensaje desde el atleta para informar los datos antes mencionados.

```
if (msg.getMessage().equals("PPM")){
  getiActividadManager().saveDato(msg);
}
```

La clase `Message` es un POJO (Plain Old Java Object)[21] tiene todas las propiedades que se necesitan para guardar los datos en tiempo real del atleta.

[21] POJO es una Clase simple que no tiene ningún comportamiento.

Para almacenar los datos implementa la clase `ActividadDato` que se puede ver en la figura IV.20. Esta entidad es una clase persistente con la implementación de su DAO y servicio para ser persistida.

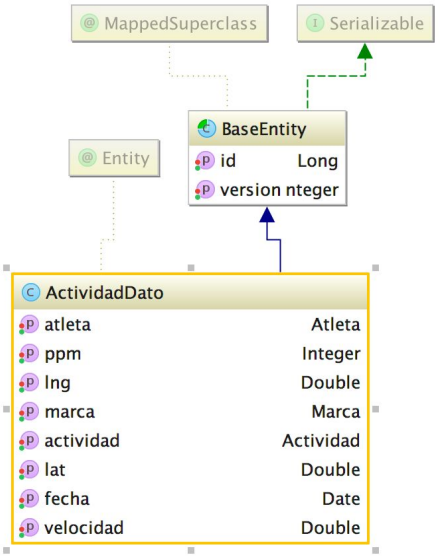


Fig.IV.20 Diagrama de Clase de ActividadDato.

Mediante los datos almacenados posteriormente se podrán utilizar para ver todo el entrenamiento y hacer cálculos sobre el entrenamiento. Estos requerimientos forman parte del último Sprint.

HU.9 Gestion de Entrenamiento Mobile

La siguiente HU se descompone en varias tres tareas, todas referente al desarrollo de la aplicación móvil en Android.

HU.9.1 Conexión BLE con Pulsómetro

Para recibir las pulsaciones del pulsómetro, es necesario utilizar el protocolo de conexión inalámbrica de bluetooth. En este caso utilizamos un Polar H7 que soporta la especificación de BLE (Bluetooth Low Energy) 4.0.

Para la implementación en Android existe documentación y un proyecto de ejemplo disponible por parte de la página oficial de Google para los desarrolladores (Google, 2015), que explica con detalle todo lo necesaria

para trabajar con este protocolo de comunicación. Básicamente esta implementación tiene tres componentes principales:

- Configuración.
- Búsqueda de dispositivos bluetooth.
- Obtención de datos.

Las aplicaciones en Android poseen un archivo principal de configuraciones, este archivo se llama manifest.xml. Como su nombre lo indica, es un manifiesto de lo que la aplicación puede hacer con el sistema operativo. En este manifiesto se definen todos los componentes que la aplicación tendrá, además de los permisos sobre elementos de hardware y de software.

Para este caso, la aplicación tiene que tener permisos para acceder al GPS, Bluetooth e Internet (Wifi o Red de datos móviles). A continuación se puede observar parte del archivo manifest.

```
<uses-feature android:name="android.hardware.bluetooth_le"/>

<uses-permission android:name="android.permission.BLUETOOTH" />

<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="providers.gsf.permission.READ_GSERVICES"

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
```

Con la expresión `uses-feature`, especificamos que vamos a requerir el hardware Bluetooth. Con la expresión `uses-permission`, le doy permisos a la aplicación sobre los elementos antes descripción. Todos estos permisos son informados al usuario cuando instala la aplicación, para lo cual acepta o no la instalación.

El siguiente paso es agregar filtros que permitan capturar eventos relacionados con el dispositivo Bluetooth. A continuación se ve parte del archivo manifest.xml donde se implementan los filtros dentro un componente `receiver` (una subclase BroadcastReceiver). Estos

permiten que la aplicación pueda recibir las intenciones que se transmiten por el sistema, incluso cuando la aplicación no se está ejecutando.

```
<receiver  android:name=".BtReceiver" >
<intent-filter>
<action android:name="lia.bluetooth.le.ACTION_GATT_CONNECTED" />
<action android:name="lia.bluetooth.le.ACTION_GATT_DISCONNECTED" />
<action   android:name="lia.bluetooth.le.ACTION_GATT_SERVICES_DISCOVERED"
<action android:name="lia.bluetooth.le.ACTION_DATA_AVAILABLE" />
</intent-filter>
</receiver>
```

Cada filtro tiene una action con el acrónimo GATT (Generic Attribute Profile), es una especificación para el envío y recepción de piezas cortas de datos conocidos como atributos.

Para poder interceptar estos filtros creamos una clase del tipo `BroadcastReceiver` llamada `BtReceiver`. Esta clase sobre escribe un metodo llamado onReceive, en el siguiente código detalla:

```
        @Override
        public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        if (BtleService.ACTION_GATT_CONNECTED.equals(action)) {  }
        else if (BtleService.ACTION_GATT_DISCONNECTED.equals(action)) { }
        else                                                    if
(BtleService.ACTION_GATT_SERVICES_DISCOVERED.equals(action)){

displayGattServices(BtleService.getSupportedGattServices());
        } else if (BtleService.ACTION_DATA_AVAILABLE.equals(action)) {
```

El método de `displayGattServices` nos permite obtener las propiedades que nos interesan para un pulsómetro dentro de la especificación GATT. La siguientes variables estáticas permiten obtener las características del pulsómetro dentro de la implementación del método antes mencionado.

```
        static String HEART_RATE_MEASUREMENT
="00002a37-0000-1000-8000-00805f9b34fb";

        static String CLIENT_CHARACTERISTIC_CONFIG
="00002902-0000-1000-8000-00805f9b34fb"
```

Para poder agregar un pulsómetro a la aplicación, se desarrolló un módulo dentro de la interfaz de configuración, que nos permite buscar un pulsómetro cercano. En la figura IV.21, se puede ver la pantalla de búsqueda de pulsómetros. Esta pantalla lista los dispositivos cercanos con las características antes mencionadas.

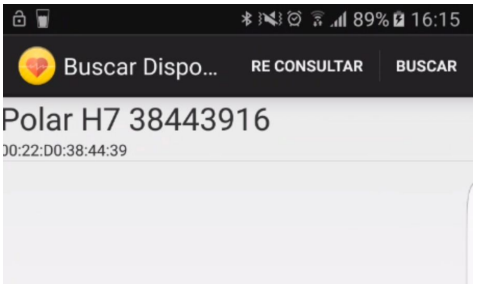


Fig.IV.21 Búsqueda de pulsómetros.

Una vez encontrado se selecciona y se agrega a la configuración de nuestra aplicación guardando las variables de nombre y dirección del dispositivo. En la figura IV.22 se puede ver los datos guardados de un pulsómetro ya configurado.

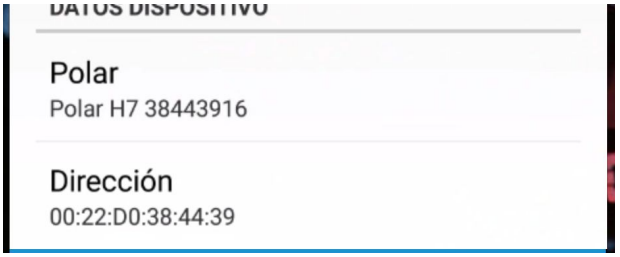


Fig.IV.22 Pulsometro Polar configurado.

En el siguiente fragmento de código se implementan las variables donde estos valores de persistirán.

```
        <PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android">

        <Preference  android:key="DEVICE_NAME" />

        <Preference android:key="DEVICE_ADDRESS" />
```

En este archivo de preferencias xml, nos permite guardar la configuración de la aplicación de forma persistente. Los valores se almacenar y recuperar como en una estructura de Mapa, por medio de una clave nos retorna una valor. Para el caso del pulsómetro utilizamos las claves de `key="DEVICE_NAME"` y `key="DEVICE_ADDRESS"`.

Para poder recuperar las PPM del pulsómetro, previamente configurado, cree un componente de tipo Servicio, que tiene la particularidad de operar en background sin la necesidad que exista interfaz gráfica.

Desde el servicio se envía el valor de la PPM a una interfaz gráfica que le permite al atleta ver el valor actual, está HU es la 9.3, la cual se desarrolló más adelante.

HU.9.2 Servicio Cliente Websocket

Para implementar el cliente de websocket sobre Android, utilice una librería que está disponible en los repositorios centrales de maven agregando la siguiente línea a las dependencias de gradle.

```
compile group: 'org.java-websocket', name: 'Java-WebSocket', version: '1.3.0'
```

Desarrolle una clase llamada `WsCliente`, que me permite encapsular la implementación websocket. Esta clase extiende de `WebSocketCliente` que está provista por la librería previamente importada. A continuación se puede ver parte de la clase.

```
public class WsClient extends WebSocketClient {

   private WSClientCallback mCallback;

    public WsClient(URI serverUri, Draft draft) {

       super(serverUri, draft);

   }

   @Override

   public void onOpen(ServerHandshake serverHandshake) {

       mCallback.onWSConnected();

   }

   @Override

   public void onMessage(String s) {

       mCallback.onWSMessageReceived(s);

   }

   @Override

   public void onClose(int i, String s, boolean b) {

       mCallback.onWSClose(i, s);

   }

   @Override

   public void onError(Exception e) {

       mCallback.onWSError(e);
```

```
    }
    public void setWSClientCallback(WSClientCallback clbck) {
       mCallback = clbck;
    }
```

El comportamiento del `callback` se implementa con la siguiente interfaz de java.

```
interface WSClientCallback {
  void onWSMessageReceived(String msg);
  void onWSConnected();
  void onWSClose(int code, String reason);
  void onWSError(Exception e);
```

Con el cliente websocket implementado podemos agregar este comportamiento al servicio que obtiene las pulsaciones y la posición del atleta, información que será transmitida por el canal establecido por websocket.

Una vez que el servicio se inicia, se crea una instancia del cliente de websocket con una referencia de la instancia del servicio, esto se puede hacer porque el servicio implementa la interfaz de `WSClientCallback`.

En el siguiente fragmento de código se puede ver parte del servicio que será el encargado de la comunicación con el pulsómetro, la conexión websocket y obtener los cambios de la posición.

```
public class BtleService extends Service implements WSClientCallback {
@Override
public void onCreate() {
locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
       locationListener = new LocationListener() {
            public void onLocationChanged(Location location) {
```

```
            posicionActual = location;
        }};}

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
      connect(app.getAdressPulsometro());
      return START_STICKY;
    }

    public boolean connect(String address) {
    BluetoothDevice device =
mBluetoothAdapter.getRemoteDevice(address);
mBluetoothGatt = device.connectGatt(context, false, mGattCallback);
     wscliente = new WsClient(new URI(app.getUrl());
     wscliente.setWSClientCallback(this);
     wscliente.connect();
     return true;
    }
```

Cabe destacar que la implementación del servicio anterior sólo representa las partes más relevante del código. También destacó que en próximas iteraciones se pueden descomponer en servicios más pequeños para facilitar y mejorar esta implementación. Actualmente este servicio envía un mensaje por el canal websocket cada vez que llega un nuevo dato de pulsación (1 o 2 veces por segundo). En este mensaje envía la PPM y también la latitud, longitud y velocidad del atleta.

Para crear un servicio en Android la clase debe extender de Service, que es propia de la SKD de Android. Además implementa la interfaz `WSClientCallback` que permite agregar el comportamiento del cliente de websocket y también implementar los métodos de la interfaz. Los métodos de conectar, desconectar y error se utilizan para informar al usuario si existe algún problema con la conexión. El método más

relevante es `onWSMessageReceived`, que permite manejar la recepción de los mensajes desde el servidor (Entrenador).

```
public void onWSMessageReceived(String msg) {
       JSONObject o = new JSONObject(msg);
         if (o.getString("message").equals("INICIADO")) {

app.savetrainingPpm(o.getString("ppmMax"),o.getString("ppmMin"));
           sendBroadcast(new Intent("INICIADO"));
        } else if (o.getString("message").equals("EN_ESPERA")) {
            app.setActividad(o.getString("actividad"));
            } else if (o.getString("message").equals("LAP_INICIO")) {
               // Inicio de Lap, setear Tiempo y ppmMin y ppmMax
            } else if (o.getString("message").equals("LAP_FIN")) {
               // Inicio de Lap, setear Tiempo y ppmMin y ppmMax
            } else if (o.getString("message").equals("FINALIZADO")) {
               // Inicio de Lap, setear Tiempo y ppmMin y ppmMax
             }
```

El parámetro que recibe este método es un String, en el cual el servidor le envía un POJO serializado en JSON. La clase JSONObject me permite pasar este JSON a un objeto, con el fin de extraer fácilmente los valores que viajan por el canal. El atributo `message` es el que indica que tipo de acción debe seguir el atleta, como el inicio de una actividad o establecer las PPM máximas y mínimas para un entrenamiento.

Mediante el método `sendHR`, propio del cliente de `websokcet`, permite enviar un mensaje al servidor por el canal previamente conectado.

```
       private void sendHR(Integer ppm) {
       Pojo p = new Pojo(ppm,
       app.getUsername(),
       posicionActual,
```

```
app.getActividad());
wsCliente.send(p.toJSON());
```

El objeto `Pojo`, tiene todos los atributos necesarios para que el entrenador pueda seguir la actividad del atleta en tiempo real.

Cuando el servicio se crear, se invoca el método `onCreate`, que inicializa los parámetros del servicio. El fragmento que utilice para explicar este método, inicia el servicio de geolocalización, además de implementar un `Listener` para obtener los cambios de la posición y velocidad.

El método `onStartCommand` se ejecuta cuando se inicia explícitamente el servicio desde la aplicación Android. Al iniciar el servicio se conecta con el pulsómetro previamente configurado, mediante su dirección física, y se conecta al canal websocket atleta con el parámetro el nombre de usuario. Ejemplo para el atleta `hmunoz` sería: `ws://server/atleta/hmunoz`. Una vez conectados al canal de websocket podemos enviar y recibir mensajes de forma bidireccional con el servidor.

HU.9.3 Interfaz de Monitoreo de la Actividad

Para que el atleta pueda interactuar con la aplicación y el entrenador, desarrolle una interfaz de Monitoreo de la actividad. Esta pantalla tiene un menú que permite iniciar y detener el servicio previamente explicado, cerrar la aplicación y acceder a la configuración. Este menú se puede ver en la figura IV.23.

Fig.IV.23 Menú de la aplicación Android.

Esta interfaz también posee dos relojes de medicion analogico-digital que informan el valor de la PPM y el porcentaje de esfuerzo. Para calcular este porcentaje se aplica una fórmula (ver ecuación IV.1.) basada en la

PPM actual y las pulsaciones máxima y mínima del Atleta, que fueron cargadas previamente al dar de alta en atleta en el sistema.

```
% Esfuerzo = (ppm - ppmMin) * 100 / (ppmMax - ppmMin)
```

Ecuación.IV.1 Ecuación de Esfuerzo.

Para obtener la PPM maxima y minima, el entrenador debe realizar un estudio específico para cada caso.

En la figura IV.24 se puede ver la interfaz, con los datos del monitoreo y progreso de la actividad.

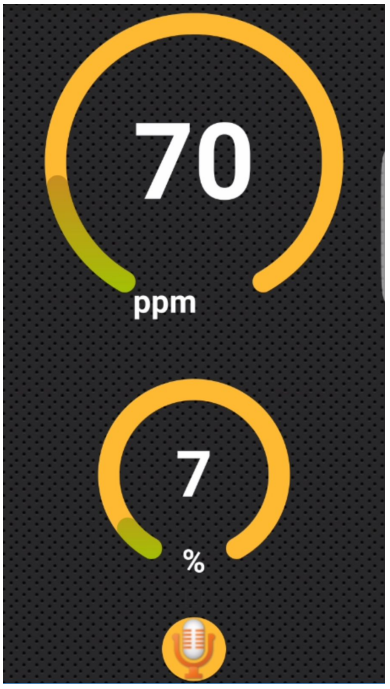


Fig.IV.24 Actividad de Monitoreo funcionando con un Pulsómetro.

Para implementar esta interfaz cree un componente de tipo `Activity`, propio de la SDK de Android. Los componentes visuales tipo reloj, también llamados Gauges, son componentes externos que se importan como librería mediante la siguiente línea de código.

```
compile 'pl.pawelkleczkowski.customgauge:CustomGauge:1.0.1'
```

Para implementar interfaces visuales se utilizan archivos xml que forman parte del layout (diseño de interfaz gráfica) de la aplicación que forman parte de los recursos de la aplicación en la carpeta `res/layout`.

A continuación, se puede ver parte de la implementación del diseño gráfico de la interfaz de monitoreo.

```
<RelativeLayout
android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@drawable/background"
        android:gravity="center"
        android:orientation="vertical"
        android:keepScreenOn="true">

   <pl.pawelkleczkowski.customgauge.CustomGauge
        android:id="@+id/gaugePpm"
        android:layout_width="300dp"
        android:layout_height="300dp"
        android:layout_centerHorizontal="true"
        android:paddingBottom="20dp"
        android:paddingLeft="20dp"
        android:paddingRight="20dp"
        android:paddingTop="20dp"
        app:pointStartColor="@color/green"
        app:pointEndColor="@color/red"
        app:startAngel="120"
        app:strokeCap="ROUND"
        app:strokeColor="@color/orange"
        app:strokeWidth="20dp"
        app:startValue="40"
        app:endValue="220"
        app:sweepAngel="300" />

   <TextView
```

```
android:id="@+id/txtPpm"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:textSize="100dp"

android:textStyle="bold"

android:layout_marginTop="79dp"

android:layout_alignParentTop="true"

android:layout_centerHorizontal="true" />
```

El primer tag del xml es una RelativeLayout, este componente permite contener otros componentes dentro de un marco, con la particularidad ubicar los componentes en su interior de forma relativa entre sí. Con este tipo de layout facilita la ubicación de los componentes mediante el mouse haciendo arrastrar y pegar.

El componente de `CustomGauge` puse un grupo de atributos, que nos permiten configurarlo a nuestra necesidad. En este caso, definimos un identificador `gaugeppm`, que nos sirve para poder manejar este objeto en el controlador. También posee configuración de colores y tamaño para que faciliten la lectura del atleta.

También se implementa un `TextView`, que nos permite informar las PPM de forma digital, se puede acceder mediante el identificador `txtppm`.

HU.10 Configuración de la Aplicación Móvil

La aplicación del atleta debe tener configurado ciertos parámetros para poder operar, como por ejemplo, IP del servidor, nombre de usuario y la dirección física del pulsómetro.

Estos valores tiene que estar guardados de forma persistente en el teléfono y poder recuperarse de forma fácil para poder realizar la actividad.

Para resolver este problema de guardar datos de forma persistentes la SDK de Android provee varias alternativas, como podría ser utilizar la base de datos SQlite, una base de datos relacional con ciertas limitaciones, o trabajar con archivos físicos. Pero Android provee una herramienta particular para resolver este problema que nos facilita la implementación desde la persistencia, la obtención de los datos y la interfaz de usuario.

Esta herramienta es parte de la SDK y se conoce con el nombre de `Preference`, que proviene del inglés preferencias.

Para definir la variable que voy a utilizar para la aplicación, existe un XML donde voy a definir todas las propiedades del sistema que el usuario de la aplicación puede modificar. En el siguiente fragmento de código se puede ver parte de la implementación de la configuración de la aplicación. El paquete `android.support.v7.preference` hace referencia a la librería previamente agregada al `buil.gradle` que permite utilizar esta clase. Esta versión tiene la particularidad de adaptarse a cualquier pantalla y versión de Android con mayor facilidad.

El Tag `<PreferenceScreen/>` define todos los elementos que van a formar parte de la interface. Este TAG permite generar bloques de elementos agrupados por un título. En el caso de esta aplicaciones Datos de Usuario agrupa varios elementos de tipo `<EditTextPreference/>`, un componente para ingresar un valor.

```
<android.support.v7.preference.PreferenceScreen>
   <android.support.v7.preference.PreferenceCategory
       android:summary="@string/pref_summary_category_user"
       android:title="@string/pref_title_category_user" >
       <android.support.v7.preference.EditTextPreference
           android:defaultValue="@string/pref_default_username"
           android:key="username"
```

```
                    android:title="@string/pref_title_username" />
                <android.support.v7.preference.EditTextPreference
                    android:defaultValue="@string/pref_default_edad"
                    android:key="edad"
                    android:title="@string/pref_title_edad" />
                <android.support.v7.preference.EditTextPreference
                    android:key="minppm"
                    android:title="@string/pref_title_ppm_min" />
                <android.support.v7.preference.EditTextPreference
                    android:key="maxppm"
                    android:title="@string/pref_title_ppm_max" />
        <android.support.v7.preference.Preference
android:defaultValue="Polar H7" android:key="DEVICE_NAME"
                android:title="@string/pref_title_device" />
          </android.support.v7.preference.PreferenceCategory>
       <android.support.v7.preference.PreferenceCategory
android:title="Configuracion Avanzada">
<android.support.v7.preference.EditTextPreference

android:defaultValue="@string/pref_default_url_ws_atleta"
                    android:key="URL_WS_ATLETA"
                    android:title="@string/pref_title_url_ws_atleta" />
                <android.support.v7.preference.EditTextPreference

android:defaultValue="@string/pref_default_port_ws_atleta"
                    android:key="PORT_WS_ATLETA"
                    android:title="@string/pref_title_port_ws_atleta" />
            </android.support.v7.preference.PreferenceCategory>
```

El elementos `EditTextPreference` tiene varias propiedades como el título, el valor por defecto si todavía no se le asignó un valor, pero la propiedad más relevante es la `Key`, que debe ser única y mediante este

valor va a permitir guardar y recuperar la propiedad. En el caso del nombre de usuario utilice el valor de `android:key="username"`.

En la figura IV.25 se puede ver como editar la propiedad `username` con solo hacer click sobre el elemento `EditTextPreference`.

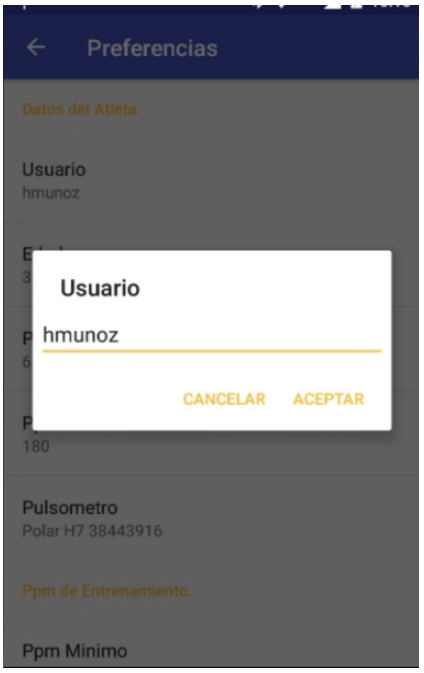


Fig.IV.25. Editar la propiedad username.

Estos valores se puede recuperar mediante la clase `SharedPreferences`, que se puede inicializar en cualquier Actividad o en el contexto de aplicación.

El primer paso es inicializar la clase cuando inicia la aplicación, como se puede ver en el siguiente código.

```
private SharedPreferences preference;

public void onCreate() {

super.onCreate();

 preference = PreferenceManager.getDefaultSharedPreferences(this);

}
```

Una vez que inicializado, ya se puede obtener cualquier valor que previamente definimos en el XML descrito anteriormente.

Para obtener el nombre de usuario cree el siguiente método utilizando como parámetro de entrada la Key que cree anteriormente:

```
public String getUserName() {
  return preference.getString("username");
}
```

Para obtener el canal de WebSocket en la aplicación podemos configurar la IP del server y el puerto. El valor del canal es una constante que se define en la aplicación. El método para recuperar la conexión al WebSocket se define de la siguiente manera.

```
static final String CANAL_ATLETA = "/tesis/atleta/";
static final String WS = "ws://";
public String getUrlWebSocket() {
return WS + preference.getString("IP_SERVER") + ":" +
preference.getString("PORT_SERVER") + CANAL_ATLETA + getUsername();
}
```

Sprint 3

El siguiente Sprint se puede ver en detalle en el siguiente tablero https://trello.com/b/vygAdMwG/sprint-3, en cual se utilizó para el seguimiento.

En este Sprint se desarrollo la visualización y análisis del entrenamiento finalizado. Una vez que la actividad termina, pasa a estado finalizado, luego de haber transcurrido el total del tiempo de todos los laps. El entrenador puede ver el detalle del entrenamientos mediante gráficos y mapa para facilitar el acceso a la información resultante del entrenamiento. En la figura IV.26 se puede ver el Diagrama de estado de la Actividad, donde el último estado posible es el finalizada.

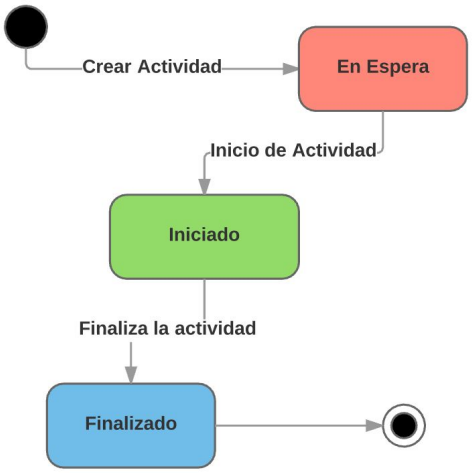


Fig.IV.26. Diagrama de Estado de una Actividad

HU.13 Visualización de Resultado del entrenamiento

El primer gráfico que el entrenador puede ver es la evolución de las PPM a lo largo del tiempo total de la actividad, que es la suma de todos los Laps.

Para generar este gráfico, el primer paso es obtener los datos que fueron grabados durante la actividad. Para ello utilizamos la clase `ActividadDato` que tiene toda la información necesaria para construir el gráfico. Cabe recordar de una `Actividad` tiene una lista de `ActividadDato`, por ende, esta nueva interfaz llamada `actividadDetalle.xhtml`, va a recibir como parámetro el `id` de la actividad. En el siguiente código se puede ver en qué lugar va a guardar el parámetro en el controlador que llegó por parámetro y un método que se va a ejecutar por única vez para inicializar los valores del gráfico.

```
<f:metadata>
  <f:event type="preRenderView"
    listener="#{actividadDetalleBean.initDetalle()}" />
  <f:viewParam name="id" value="#{actividadDetalleBean.id}" />
</f:metadata>
```

El controlador llamado `actividadDetalleBean` posee todo la implementación para obtener todos los gráficos de la actividad. En el siguiente código se carga la actividad a partir del `id` con el servicio de actividad que ya había implementado para el CRUD. Dentro de la instancia de actividad están todos los datos necesarios para construir el gráfico. En el código siguiente se puede ver parte del controlador y el método que inicializa los datos.

```
private Actividad actividad;

private Long id;

public void initDetalle() {

     this.actividad = getiActividadDao().find(this.id);

     crearGraficoLinea(this.actividad);
```

El Framework de vista que seleccione para el desarrollo, PrimeFaces, tiene complementos para implementar gráficos de línea, torta, barra entre otros. Para el caso particular de esta HU, utilice gráficos de líneas con el siguiente tag.

```
<p:chart type="line"

model="#{actividadDetalleBean.modelActividad.get(v_username)}"/>
```

La variable `v_username` es el parámetro para obtener el gráfico de un atleta en particular. La variable `modelActividad` es una `Map<String, LineChartModel>` donde se cargaron todos los graficos de linea, uno por cada atleta y se obtienen mediante el nombre de usuario. La clase `LineChartModel`, propia del Primefaces, tiene la estructura y métodos para guardar un gráfico de líneas. En el siguiente código se carga el Map `modelActividad` con los gráficos de linea de todos los atletas que participaron de la actividad.

```
private void crearGraficoLinea(Actividad actividad) {

  for (Atleta a : actividad.getAtletas()) {

     LineChartModel lineModel = initLinearModel(a);

     lineModel.setTitle(a.getUsuario().getApellidoNombre());
```

```
 lineModel.getAxes().put(AxisType.X, new CategoryAxis("Min."));
 Axis yAxis = lineModel.getAxis(AxisType.Y);
 yAxis.setMin(40);
 yAxis.setMax(220);
 this.modelActividad.put(a.getUsuario().getUsername(), lineModel);
  }
}
private LineChartModel initLinearModel(Atleta a) {
  LineChartModel model = new LineChartModel();
  LineChartSeries series = new LineChartSeries();
  series.setLabel(a.getUsuario().getApellidoNombre());

     for (ActividadDato dato : getActividad().getDatos()) {
        if (dato.getAtleta().equals(a)) {
              series.set(dato.getFecha, dato.getPpm());
        }
     }
  model.addSeries(series);
  return model;
```

El método `initLinearModel` es el que carga los valores de la serie con los datos de la `actividadDato` filtrando por cada usuario. En figura IV.27 se puede ver el resultado del gráfico.

Fig.IV.27 Grafico de Lineas de PPM y Tiempo

HU.14 Gráfico del mapa del recorrido del entrenamiento

El gráfico del recorrido resulta de gran ayuda al entrenador para constatar el ejercicio del atleta. En esta primera etapa del proyecto el entrenador solo puede visualizar en recorrido realizado, en próximas iteraciones el entrenador podría recomendar un recorrido para el entrenamiento.

Como en el gráfico de la HU.13, comparten el controlador y la misma vista. Todos los graficos estan dentro de un contenedor tabView. En el siguiente código se puede ver la estructura del contenedor para cada gráfico. El atributo `dynamic="true"` permite que solo se renderice el gráfico que se seleccione.

```
<p:tabView dynamic="true">
<p:tab title="Actividad">
........
</p:tab>
<p:tab title="Recuperación">
.........
</p:tab>
<p:tab title="Recorrido">
.........
</p:tab>
</p:tabView>
```

Para realizar este gráfico utilice el componente de gmap de primefaces, que hace uso internamiento de la implementación de Google Maps V3. Por ello requiere importar una librería de JavaScript en la vista de la siguiente forma.

```
<script
type="text/javascript"src="https://maps.googleapis.com/maps/api/js?v=3">
```

Para implementar el componente de gmap hace falta un modelo que se relaciona con el controlador. De la misma forma que la HU anterior, cuando se inicia el controlador inicializamos los valores del mapa para cada unos de los atletas en variable `Map<String, MapModel> recorrido`. Esta estructura nos permite recuperar el Modelo en función del atleta. En el siguiente código permite dibujar en gráfico en la vista.

```
<p:gmap  zoom="12" type="MAP"
model="#{actividadDetalleBean.recorrido.get(v_atleta.username)}" >
```

El siguiente método carga en la variable `recorridos` los mapas para cada atleta.

```
private void createRecorridoModels() {
  for (Atleta a : getActividad().getAtletas()) {
     MapModel map = new DefaultMapModel();
     Polyline polyline = new Polyline();
List<ActividadDato> datos =
getiActividadDatoDao().datosByAtletaActividad(a, getActividad());
        for (ActividadDato actividadDato : datos) {
polyline.getPaths().add(new LatLng(actividadDato.getLat(),
actividadDato.getLng()));
        }
 map.addOverlay(polyline);
getRecorrido().put(a.getUsuario().getUsername(), map);
}
```

Para dibujar el recorrido se utiliza una clase Polyline, propia de la API de Primefaces, la cual tiene la estructura para agregar los puntos por donde el atleta recorrió. Para obtener todos los datos del atleta filtrados por atleta y la actividad seleccionada, implemente el siguiente método

```
public List<ActividadDato> datosByAtletaActividad(Atleta atleta,
Actividad actividad ) {
```

```
  return getEm()
  .createQuery(
  "select o from ActividadDato o where o.atleta = :atleta
and o.actividad = :actividad
order by o.fecha")
  .setParameter("atleta", atleta)
  .setParameter("actividad", actividad).getResultList();
```

En la figura IV.28 se puede ver el resultado de la implementación descrita anteriormente, se trata del recorrido de prueba de un atleta.

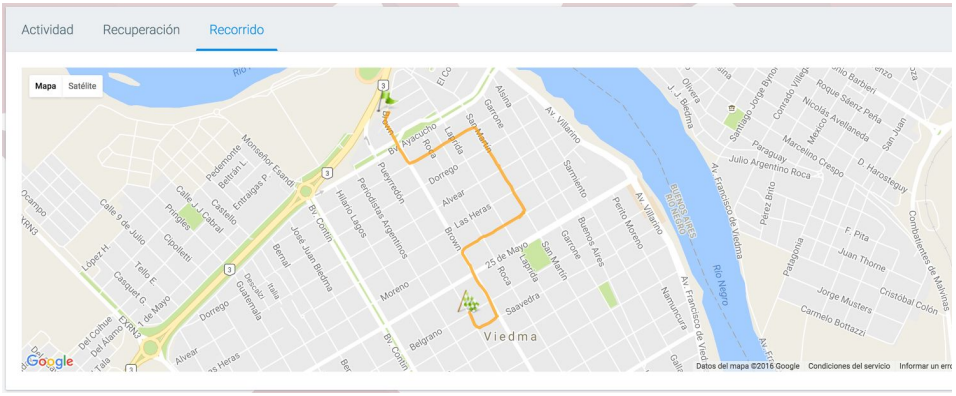


Fig.IV.28. Mapa con el recorrido de un atleta

HU.15 Análisis de Recuperación

El análisis de recuperación parte de un índice de recuperación de la frecuencia cardiaca (FC). Este índice define la recuperación al pasar un minuto y cuatro minutos después de realizar un esfuerzo máximo. En la figura IV.29 se puede ver la tabla con dos columnas.

	% de descenso de la FC postesfuerzo	
	Al 1'	A los 4'
EXCELENTE	>20%	> 40%
BUENO	15-19%	35-39%
ACEPTABLE	10-14%	30-34%
INSUFICIENTE	6-9 %	25-29%
POBRE	< 6%	<25%

Fig.IV.29. Tabla de Índice de recuperación de la FC.

Mediante este índice podemos saber la capacidad que tiene el atleta de recuperarse después de un esfuerzo. Una baja capacidad de recuperación podría determinar alguna patología cardiaca. Esta información ayuda al entrenador a establecer planes de mejora de la recuperación como también para anticipar algún problema de salud del atleta.

Para implementar este gráfico, utilice el mismo controlador y viste que los anteriores, también está ubicado en el mismo Tab.

La principal diferencia con el gráfico de la HU.13, radica en obtener solo cuatro minutos del entrenamiento, partiendo del pico más alto en la FC. En la figura IV.30 se puede ver el gráfico con los resultados obtenidos para uno y cuatro minutos.

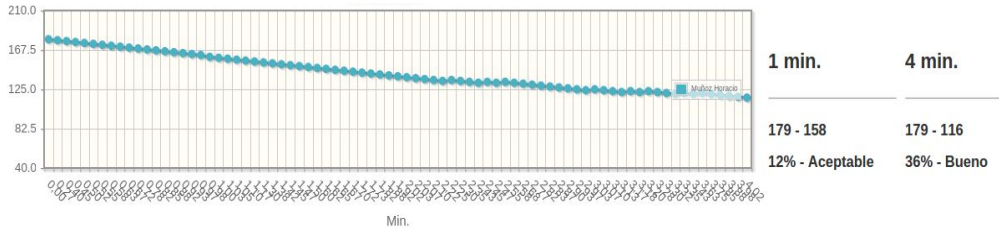


Fig.IV.30. Gráfico de Recuperación para 1 y 4 minutos.

En la primera fase de desarrollo puse realizar una prueba del sistema en el Congreso de Educación Física, organizado por la Universidad Nacional de Río Negro en Agosto del 2014. El atleta era un ciclista de élite Leandro

Messione. Mediante una bicicleta fija que permite agregar carga, Leandro alcanzó las 191 PPM y logró una recuperación para uno y cuatro minutos excelente según el índice anteriormente mencionado. En la figura IV.31 se puede ver el resultado de dicha recuperación.

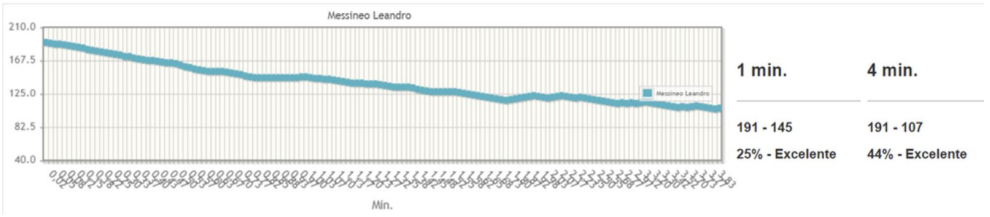


Fig.IV.31. Recuperación Atleta de de Élite.

V. Verificación de Resultado

En el capítulo anterior se describió cómo implementar pruebas unitarias y de integración, lo que permitió construir codigo de calidad y asegurar el correcto funcionamiento descrito en las historias de usuario.

En este capítulo se desarrolló un conjunto de pruebas integrales que permitieron determinar el funcionamiento completo de la aplicación, el cual permite verificar la totalidad de los componentes del sistema.

En esta primera versión, el sistema permite realizar un monitoreo en tiempo real de los atletas y también genera un diagnóstico de la recuperación después un esfuerzo máximo. Para el Lic. Matías Scavo, era relevante realizar las pruebas con dos atletas conocidos en lo referente a su rendimiento físico, para poder constatar los resultados de la aplicación.

V.1 Casos de Prueba

Se convocaron dos ciclistas de alto rendimiento con diferentes niveles de preparación. Leandro Messineo[22] ciclista reconocido de Élite y el Mauro Tardon un triatlonista reconocido en la región patagónica.

V.1.1 Condiciones generales de la prueba

La aplicación Web está alojada en un servidor CRX con procesador Intel Xeon y 16 Gram. Este servidor es equipamiento de Laboratorio de Informática Aplicada (LIA) de la Universidad Nacional de Río Negro (UNRN).

Se utilizó un celular Moto G1 con Android 5.1, el cual se conectó al servidor mediante servicio de 3G de la empresa Claro. En la prueba se utilizó un pulsómetro Polar H7 mediante conexión Bluetooth.

[22] Formó parte de la Selección Nacional Argentina y también parte del equipo de San Luis.

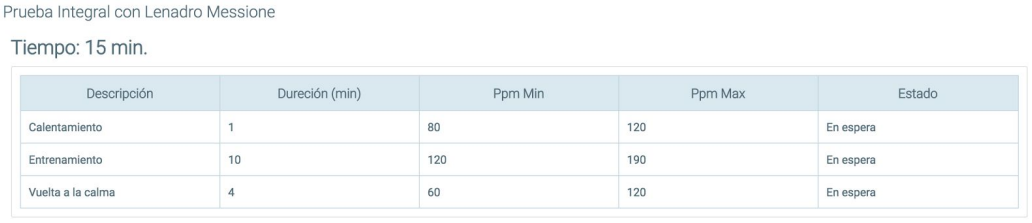
Las pruebas se realizaron en similares condiciones en lo que se refiere a ubicación y condiciones climáticas, dado que se decidió desarrollar en espacio cerrado. También se utilizó la misma bicicleta fija, la cual permite agregar carga mediante pesas.

V.1.2 Prueba Leandro Messineo

La evaluación que se realizó con Leandro Messione fue en el marco del congreso provincial de educación física organizado por la UNRN.

V.1.2.1 Configuración del Entrenamiento

Se creó una nueva actividad la cual fue asignada a Messineo. Dicha actividad constó de tres intervalos o laps los cuales suman 15 minutos en total. El primer lap es de calentamiento con un minuto de duración y PPM mínima de 80 y 120 de máxima. El segundo lap es donde se va a desarrollar todo el trabajo de fuerza, por esta razón la PPM máxima configura en el máximo del atleta con una duración de 10 minutos. Finalmente un lap de vuelta a la calma o recuperación de cuatro minutos, con este tiempo nos permite realizar el diagnóstico de recuperación. En la figura V.1. se puede ver la pantalla del entrenador una vez configurada la actividad.


Prueba Integral con Lenadro Messione

Tiempo: 15 min.

Descripción	Dureción (min)	Ppm Min	Ppm Max	Estado
Calentamiento	1	80	120	En espera
Entrenamiento	10	120	190	En espera
Vuelta a la calma	4	60	120	En espera

Fig. V.1. Configuración de la prueba con tres Laps y un total de 15 minutos de duración.

V.1.2.2 Resultados obtenidos

Como el experto esperaba, los resultados del ciclista de Élite fueron correctos según el conocimiento empírico previo. Durante el desarrollo de la prueba se pudo ver en tiempo real los valores de las PPM y el porcentaje de esfuerzo. En la figura V.2 se puede ver el gráfico de recuperación al minuto y a los cuatro minutos, tiempo que corresponde al lap tres de vuelta a la calma, en ambos momentos el resultado obtenido por el sistema determina la excelente recuperación y preparación física del atleta.

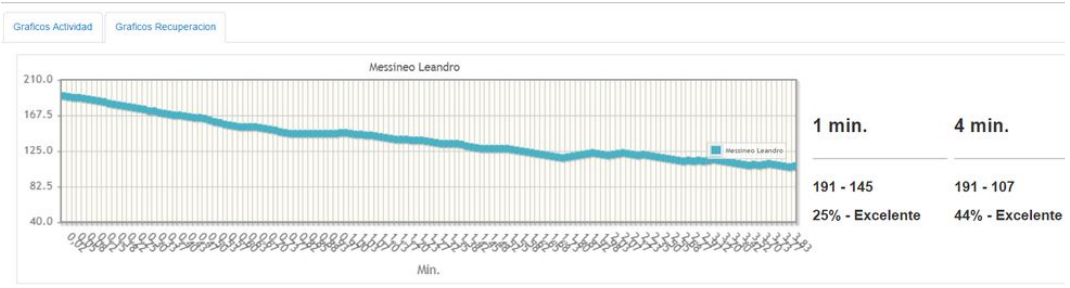


Fig.V.2. Gráfico de recuperación de Messineo, el cual destaca la preparación física del atleta.

V.1.3 Prueba Mauro Tardon

La evaluación de Mauro Tardon se realizó en el Instituto de Formación Docente en Educación Física en el contexto la de materia Fisiología que dicta el Lic. antes mencionado.

V.1.3.1 Configuración del Entrenamiento

Se creó una nueva actividad la cual fue asignada a Tardon. Al igual que la prueba con Messineo constó de tres intervalos, con la diferencia que el total en minutos fueron de 25 minutos. El primer lap es de calentamiento con tres minuto de duración y PPM mínima de 80 y 120 de máxima. El segundo lap es donde se va a desarrollar todo el trabajo de fuerza, por esta razón la PPM máxima configura en el máximo del atleta

con una duración de 18 minutos. Finalmente un lap de vuelta a la calma o recuperación de cuatro minutos, con este tiempo nos permite realizar el diagnóstico de recuperación. En la figura V.3. se puede ver la pantalla del entrenador una vez configurada la actividad.

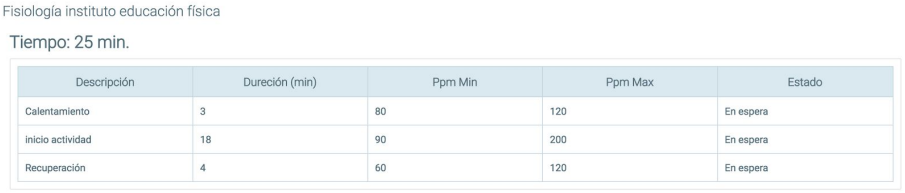


Fig. V.3. Configuración de la prueba con tres Laps, un total de 15 min.

V.1.3.2 Resultados obtenidos

Al igual que en el caso de Messineo, los resultados obtenidos fueron los esperados por el experto. En todo momento el entrenador pudo seguir la prueba y ver los valores de las PPM en tiempo real desde su notebook. En la figura V.4 se puede ver el gráfico del total de la actividad en un eje de tiempo y PPM.

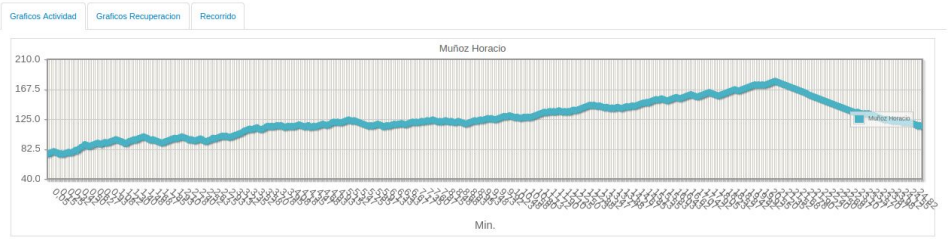


Fig.V.4. Gráfico de PPM y tiempo de toda la actividad de Tardon.

En la figura V.5 sigue un gráfico de recuperación al minuto y a los cuatro minutos, tiempo que corresponde al lap tres de recuperación. En ambos momentos el resultado obtenido por el sistema determinó de forma correcta según el entrenador la aptitud física del Atleta.

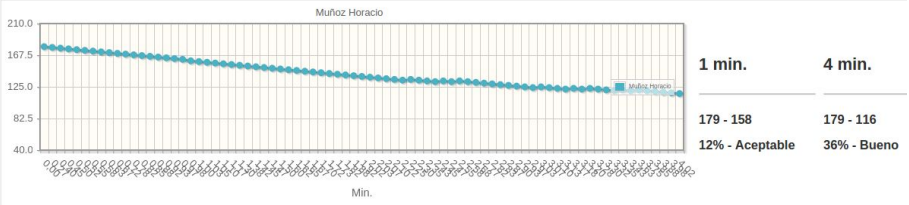


Fig.V.5. Gráfico de recuperación de Tardon, el cual demuestra la buena recuperación del atleta.

V.2 Conclusiones

El desarrollo de las dos pruebas permitió determinar el correcto funcionamiento de la arquitectura de la solución propuesta. Cada componente del sistema (pulsometro, celular y servidor) se comunican uno con otro de forma constante y estable, logrando obtener información fiable en tiempo real para el entrenador y los atletas.

Para el entrenador fue fácil configurar los entrenamientos desde la aplicación web tarea que se realizó con asistencia técnica del tesista por ser los primeros usos de la aplicación.

La configuración de la aplicación móvil en la actualidad requiere ingresar los datos del atleta como nombre de usuario, el cual es único para toda la aplicación. Esta tarea también la realizó el tesista.

El resultado de la prueba se compone de dos partes bien definidas, los resultados en tiempo real sobre los datos de atleta y los resultados obtenidos después de finalizar el entrenamientos.

En el primer caso, el entrenador observa los valores de PPM y porcentaje de esfuerzo del atleta en todo momento sin pérdida de conexión. También utilizó el envío de mensajes los cuales llegaron todos al instante.

Una vez finalizado el entrenamiento obtuvimos dos gráficos importes para el entrenador. El primero una línea de tiempo combinado con las PPM a lo largo del total del entrenamiento y el segundo un gráfico de recuperación

que se explicó anteriormente. Ambos gráficos analizados en conjunto con el especialista, fueron generados de forma correcta.

V.2.1 Conclusiones del entrenador

El Lic. Matias Scavo arriba a la siguiente conclusión:

La aplicación resultó ser apropiada para el control de carga durante el entrenamiento, por lo que facilita la elección de la carga antes de aplicar un estímulo en el desarrollo de la actividad. Permite planificar estímulos en función de los tiempos de trabajo y pausa establecido en la gestión del entrenamiento.

La característica principal de la aplicación es el monitoreo en tiempo real, lo que permite adaptar las cargas en función de la retroalimentación de la respuesta fisiológica del deportista.

Otra característica importante es la valoración post esfuerzo. La aplicación determina la valoración de la recuperación de PPM con valores de referenciales que son los óptimos para uno y cuatro minutos, generando el cálculo de este índice de forma automática. Dicho valor de recuperación de las PPM establece el nivel de entrenabilidad y de salud cardiovascular, parámetros esenciales para la planificación de próximos entrenamientos y seguimientos del progreso del atleta.

VI. Conclusiones y Líneas Futuras

En este apartado se resumen las conclusiones más importantes de todo lo trabajado. Se ha evaluado el grado de cumplimiento de los objetivos iniciales y se exponen los problemas planteados y el método de resolución, los conocimientos adquiridos frutos del desarrollo del mismo y las líneas futuras o mejoras de la solución que no se han podido implementar por falta de tiempo.

VI.1 Conclusiones

Como se mencionó anteriormente y en particular en el capítulo III, podemos desarrollar una conclusión en función de los pilares de la investigación a partir de los siguientes conceptos:

La importancia de la FC como valor para la toma de decisiones en el desarrollo del atleta, Android como plataforma de desarrollo para teléfonos inteligentes y la arquitectura de la aplicacion Web Java EE con Websocket. A continuación se desarrollan estos tres conceptos.

VI.1.1 FC

Para el desarrollo de la aplicación, la FC fue sumamente importante. No solo por los fundamentos del parámetro durante el transcurso del entrenamiento sino también cómo influyó en la arquitectura de la solución. Desde investigar qué pulsómetro se ajustaba mejor a la solución, con las dificultades de conseguir en el país este tipo de sensores, hasta el análisis e implementación de los protocolos de comunicación entre los componentes del sistema.

Con un solo sensor externo permitió desarrollar una aplicación de mucha utilidad para atletas y entrenadores permitiéndoles un verdadero seguimiento en línea.

VI.1.2 Android

En el capítulo IV se explicó como la plataforma Android sufrió varios cambios importantes durante el proceso de desarrollo, los cuales fueron incorporados al proyecto. Desde el cambio de entorno de trabajo de Eclipse por Android Studio, como la incorporación de Gradle para la gestión de la aplicación y dependencias. También se incorporaron algunos conceptos de Material Design provistas en las nuevas librerías de la SDK de la plataforma. Todas estas herramientas que provee e incorpora Google para el desarrollo en Android facilitaron la creación de la aplicación móvil.

La elección de la plataforma Android para desarrollar esta aplicación fue correcta en función de los resultados obtenidos mencionados en el capítulo V y la facilidad para crear cualquier tipo de aplicación.

Componentes como los Servicios, librerías externas de Clientes de Websocket y elementos de UI permitieron cumplir con los objetivos planteados en las bases de este proyecto.

Uno de los puntos más importantes fue la conectividad del pulsómetro mediante Bluetooth y la conexión Websocket como servicio en segundo plano. Dichos requerimientos fueron implementados sin problemas, en gran parte por la documentación y ejemplos provistos en la página para desarrolladores Android de Google y las características antes mencionadas.

VI.1.3 Arquitectura Web

La implementación de esta solución requería servicios Web para lograr los objetivos planteados. Esta arquitectura permitió que un entrenador pueda ver en tiempo real los datos de sus atletas con solo acceder a un explorador.

La elección de Java EE logró una aplicación sólida y estable respetando los estándares como se explicó en el capítulo IV donde se detalla para paso del desarrollo.

Con Frameworks como Primefaces se crearon interfaces de usuario intuitivas y con poco esfuerzo gracias a la variedad de componentes que el mismo provee.

Otro componente relevante en la arquitectura es Websocket, que se incorporó a la arquitectura siguiendo las especificaciones mencionadas en el capítulo IV. Con este protocolo se logró establecer una comunicación bidireccional con poca latencia y tráfico de información. En este sentido y basado en los resultados obtenidos dicha implementación puedo asegurar el cumplimiento de todos objetivo planteado.

VI.2 Líneas Futuras

El desarrollo de este trabajo se centró en el entrenamiento y el deporte de alto rendimientos. En próximos proyectos, sería interesante utilizar la arquitectura propuesta, para aplicaciones orientadas a la medicina o mHealth, ya que actualmente existen en el mercado todo tipo de sensores que permitirían sumar variables que ayudarían al monitoreo en tiempo real del paciente, pero también al seguimiento y progreso de distintas afecciones.

Podemos mencionar algunos de estos sensores: Balanzas Inteligentes, Presión arterial, Glucosa para Diabéticos, Temperatura corporal, Electrocardiograma y Pulsómetro.

Con estos sensores y la arquitectura propuesta, se podría desarrollar una solución que permitiría realizar seguimientos a los pacientes de forma ambulatoria y generar alertas en función de distintos parámetros como niveles de Glucosa, temperatura corporal, presión arterial entre otros.

VI.2.1 ANT+, una alternativa a bluetooth

Si bien el protocolo Bluetooth resultó exitoso y permitió conectar de forma fácil y estable el pulsometro con el celular. Considero que el protocolo ANT+ permitirá realizar el mismo trabajo y con la ventaja de estar desarrollado bajo código abierto.

Empresas como Garmin y Samsung ofrecen en la actualidad un gran número de sensores que pueden incorporarse mediante este sistema.

VI.2.2 Arquitectura de la Solución

En el transcurso del desarrollo de la tesina y particularmente en la evaluación de la Arquitectura de la aplicación, fueron surgiendo modificaciones producto del avance de las tecnologías, como en el caso de la plataforma Android, servicios web, nuevos framework, patrones de diseño y herramientas que podrían mejorar el producto de esta trabajo.

VI.2.2.1 Patrón MVP para desarrollo Android

El patrón Model View Presenter (MVP) ayudaría a generar una aplicación Android más fácil de mantener y escalable. Ya que este es un patrón arquitectónico que sirve para modelar la capa de presentación con la que interactúa el usuario. Lo que hace este patrón es delegar toda la lógica a una entidad llamada Presenter, que será la que se encargue de decidir qué se dibuja en la vista y actuar ante los eventos que el usuario solicite.

La vista no será más que el punto de interacción con el usuario y en Android estará representado por una Activity.

El modelo es toda la lógica de negocio, persistencia de datos, y conexión con APIS como servicios Rest.

Una característica importante, y especialmente útil en un entorno tan invasivo como Android es la independencia del framework, gracias a los Presentadores, se consigue que a la SDK de Android no se utilice por ninguna parte. En este sentido podemos mencionar las ventajas de MVP:

- Escribir un código más limpio y semánticamente más correcto.
- Realizar tests unitarios más sencillos evitando gran parte de tests de instrumentación. Al hacer las vistas tan simples, no necesitamos probar prácticamente nada de lógica en ellas.

VI.2.2.2 Servicio de Mensajería de Google FCM

En la implementación de la tesina, la comunicación entre celular y servicio web se efectuó mediante Websocket y servicios REST. En próximas investigaciones la utilización de Firebase Cloud Message (FCM) o servicio en la nube de mensajería podría mejorar la comunicación y recepción de mensajes y notificaciones.

En el año 2016 Google adquiere FireBase y lo incorpora en el Play Service (Capa de servicios de Google para Android) mediante el cual proporciona mensajería de forma nativa entre otros servicios. Éste servicio tiene la ventaja de garantizar la recepción del mensaje, inclusive en los casos de estar sin conexión. Android verifica si existen mensajes pendientes y sincroniza todos los mensajes que no se hayan recibido al recuperar la conectividad.

Anteriormente este servicio se llamaba Google Cloud Mensaje (GCM) y la utilización del servicio era con cargo por cantidad de tráfico. FCM a diferencia de GCM no tiene ningún costo.

VI.2.2.3 MQTT y HTTP 2.0, alternativas a Websocket

Existen otros protocolos que podrías reemplazar la implementación realizada con Websocket para el envío de mensajes como se planteó en este trabajo. En próximas investigaciones se podría analizar ventajas con protocolos como MQTT[23] o la nueva implementación del protocolo HTTP 2.0 el cual provee mecanismos de envío de mensajes de forma nativa bajo el nombre de Server Push.

MQTT es un protocolo ideado por IBM y liberado para que cualquiera podamos usarlo enfocado a la conectividad Machine-to-Machine (M2M). Está enfocado al envío de datos en aplicaciones donde se requiere muy poco ancho de banda. Además, sus características le permiten tener un consumo realmente bajo así como precisar de muy pocos recursos para su funcionamiento, ideal para hardware como Arduino o Raspberry Pi entre otros.

Estas características han hecho que rápidamente se convierta en un protocolo muy empleado en la comunicación de sensores y, consecuentemente, dentro del Internet de las Cosas.

VI.2.2.4 Angular, una alternativa a JSF para la UI

La administración y gestión de los entrenamientos en la aplicación se opera mediante servicios web explicados en detalle en el capítulo IV. En el mismo se explicaron varios frameworks que se utilizaron y estándares como JSF 2.0 con las especificaciones de Java que implementan la interfaz de usuario en el contexto de una aplicación Java EE.

Sin embargo, existen otros frameworks que permiten lograr iguales resultados para el usuario final, pero con una mejor escalabilidad de la aplicación optimizando la carga de renderizado orientada hacia el cliente.

[23] MQTT son las siglas de Message Queue Telemetry Transport o Transporte de telemetría de cola de mensajes.

Angular, antiguamente llamado AngularJs, está desarrollado en JavaScripts y se ejecuta del lado del Cliente mediante un explorador como Chrome o Firefox, a diferencia de JSF que el procesamiento se realiza en el servidor.

Esta herramienta de trabajo tiende a crecer fuertemente en la comunidad de desarrolladores de todo el mundo. En el gráfico VI.1 se presenta una comparación de tendencias entre JSF y Angular, esto describe incremento de interés de Angular y la disminución gradual de JSF.

Fig.VI.1. Comparación entre JSF y Angular en el servicio de Google Trends.

Para implementar Angular dentro de la arquitectura Java EE y respetar los estándares y especificaciones debemos seguir el documento JSR 371 con el nombre MVC version 1.0. Esta especificación, la cual tiene poco tiempo de finalizada (30/01/2017). El objetivo final de esta especificación es es definir un framework MVC completamente independiente de JSF.

Tanto Angular como las nuevas especificaciones de Java EE deben ser analizadas e investigadas en próximos trabajos para determinar si existe realmente una mejora en el rendimiento y calidad del software.

Bibliografía

- Anurag, Prakash (2013), https://www.scrumalliance.org/community/articles/2013/september/what-is-sprint-zero [Accedido Marzo. 2013].
- Apache, Websocket Tomcat (2013), https://tomcat.apache.org/tomcat-8.0-doc/web-socket-howto.html. [Accedido Marzo. 2013].
- Caules, C. L. (2012). Arquitectura java sólida.
- Emmanuel Bernard, JRS-303 Bean Validation, http://beanvalidation.org/1.1/spec/. [Accedido Marzo. 2013].
- Gargenta, M. (2011). Learning Android. Sebastopol, CA: O'Reilly.
- Google, Android Studio, https://developer.android.com/studio/index.html. [Accedido Abril. 2014].
- Google, Bluetooth LE, (2013), https://developer.android.com/guide/topics/connectivity/bluetooth-le.html?hl=es [Accedido Marzo. 2013].
- Grupo Sobre Entrenamiento (G-SE). (2017). Frecuencia Cardiaca Máxima - Entrenamiento de la Resistencia. http://g-se.com/es/entrenamiento-de-la-resistencia/blog/frecuencia-cardiaca-maxima. [Accedido 2 Feb. 2017].
- Grupo Sobre Entrenamiento (G-SE). (2017). Control fisiológico del Deportista. Registro de Frecuencia cardiaca - Fisiología del Ejercicio. http://g-se.com/es/fisiologia-del-ejercicio/blog/control-fisiologico-del-deportista-registro-de-frecuencia-cardiaca. [Accedido 2 Feb. 2017].
- Internet Engineering Task Force (2016). RFC 6455 - The WebSocket Protocol. https://tools.ietf.org/html/rfc6455. [Accedido 23 Oct. 2016].

- Jcp (2016). The Java Community Process(SM) Program - JSRs: Java Specification Requests - JSR 356, https://jcp.org/en/jsr/detail?id=356. [Accedido 23 Oct. 2016].
- Jeukendrup, A., A. Van Diemen (1998). Heart rate monitoring during training and competition in cyclists. J.Sports Sci.16:S91-S99.
- Kniberg, H. (2007). Scrum and xp from the trenches: How we do scrum.
- López Chicharro, J. and Fernández Vaquero, A. (2006). Fisiología del ejercicio. Madrid: Médica Panamericana.
- Oracle (2016). JSR 356, Java API for WebSocket. http://www.oracle.com/technetwork/articles/java/jsr356-1937161.html [Accessed 23 Oct. 2016].
- Oracle (2016). Java Platform, Enterprise Edition (Java EE) | Oracle Technology Network | Oracle. http://www.oracle.com/technetwork/java/javaee/overview/index.html [Accedido 23 Oct. 2016].
- Pescatello, L. S. (2014). ACSM's guidelines for exercise testing and prescription. Philadelphia: Wolters Kluwer/Lippincott Williams & Wilkins Health.
- Peter Lubbers & Frank Greco, (2016). http://www.websocket.org/quantum.html [Accedido 23 Oct. 2016].
- W3 (2016). Request / Response. https://www.w3.org/Library/User/Architecture/Request.html. [Accedido 23 Oct. 2016].
- Websocket (2016). HTML5 WebSocket - A Quantum Leap in Scalability for the Web. http://www.websocket.org/quantum.html [Accedido 19 Dec. 2016].
- Wilmore, J. and Costill, D. (2004). Fisiología del esfuerzo y del deporte. 1st ed. Barcelona: Editorial Paidotribo.