# Documento de Arquitectura de Software

## Histórico de versiones

*Esta sección contiene la información de los cambios que se realizaron en este documento.*

| Versión | Fecha | Resumen de los cambios producidos |
|---|---|---|
| 1.0 | 31/05/2012 | Versión Inicial del documento |
| 1.1 | 02/06/2012 | Se agrego aspectos relevantes para la arquitectura. |
| 1.2 | 08/06/2012 | Cambio radical en los procesos y diagrama de dominio. |
| 1.3 | 12/06/2012 | Se modifico el nombre de la entidad “Activity” en el diagrama de dominio. |

## Control de la Preparación, Revisión y Aprobación

*Esta sección contiene la información de los responsables de los cambios que se realizaron en este documento.*

| Responsabilidad | Nombre del Responsable | Cargo / Rol | Fecha |
|---|---|---|---|
| Revisión | Esteban Arlando | Analista | 08/06/2012 |
| Preparación | Damián Lezcano | Analista | 31/05/2012 |
| Supervisión | Ezequiel Fernández | Analista | 08/06/2012 |
| Aprobación | Arratta Sebastián | Analista | 08/06/2012 |

# Table de Contenidos

# Introducción

## Objetivo

El documento tiene como objetivo definir la arquitectura de la aplicación @BuddyTraining organizada en vistas, a través de las cuales podrán especificarse los distintos aspectos técnicos y funcionales del sistema, así como también las decisiones involucradas en cada incumbencia.

## Alcance

Se considera al Documento de Arquitectura de Software como un documento "vivo", que sufrirá modificaciones a lo largo de la construcción del sistema. Se espera que provea información complementaría al código fuente.

A alto nivel, el documento abarca:

- La descripción detallada de la arquitectura de software, incluyendo los elementos o componentes principales del sistema y sus interacciones.
- Un entendimiento común de los aspectos relevantes para la arquitectura, incluyendo atributos de calidad, restricciones que afectan a las decisiones tomadas, principios adoptados y una lista de aspectos técnicos y funcionales que quedan fuera de alcance.
- Una descripción de los artefactos de software generados por el proyecto y las operaciones para realizar el despliegue de los mismos sobre la infraestructura subyacente.
- Justificaciones explícitas de las tecnologías y decisiones adoptadas para satisfacer los requerimientos funcionales, no funcionales y adaptarse a las restricciones del proyecto.

## Definiciones, Acrónimos y Abreviaturas

| Término | Descripción Breve | Descripción Larga |
|---|---|---|
| JavaEE 5 | Java Enterprise Edition 5 | Conjunto de especificaciones y plataforma de desarrollo montada sobre el lenguaje Java estándar, implementada por los servidores de aplicaciones. |
| JPA | Java Persistence API | Especificación estándar de un ORM para JavaEE. En el mercado hay muchas implementaciones, entre las más conocidas se encuentran: Hibernate, EclipseLink, OpenJPA. |
| ORM | Object-Relational Mapper | Framework de Persistencia basado en un patrón de Martin Fowler conocido como |

| | | |
|---|---|---|
| | | DataMapper. Un ORM sirve para mapear un modelo de objetos del mundo de la programación orientada a objetos a un modelo de tablas del mundo de la lógica relacional. |
| EJB | Enterprise JavaBeans | Especificación estándar de JavaEE. Los EJB son objetos Java preparados para guardar lógica de negocio, abstrayendo al desarrollador de aspectos no funcionales como manejo de transacciones, seguridad, concurrencia, etc. |
| JSF | JavaServer Faces | Especificación estándar de un framework MVC para desarrollo de aplicaciones web con la plataforma JavaEE. |
| MVC | Model-View-Controller | Patrón de diseño para capa de presentación, que sirve para separar la lógica de los objetos que se muestran en pantalla, de la lógica para mostrar esas pantallas, de la lógica de los objetos en sí. |
| ADF | Application Development Framework | Framework de Oracle basado en JSF, que extiende la funcionalidad básica de la especificación, ofreciendo nuevos componentes con funcionalidades más completas. |
| Servidor de Aplicaciones | Un servidor de aplicaciones sirve como servidor de aplicaciones web y de lógica de negocio. | Un Servidor de Aplicaciones en JavaEE es quien provee la implementación de las especificaciones más importantes de JavaEE, entre ellas: JPA, EJB y JSF. Algunos ejemplos de los servidores de aplicaciones más conocidos en el mercado son Oracle Weblogic, JBoss AS, IBM Websphere, Glassfish. |
| POJO | Plain Old Java Object | Simplemente objetos Java tradicionales, que no implementan ni extienden ninguna interfaz o clase propia de algún framework o tecnología. |
| SOLID | Single responsability, Open-closed, Liskov substitution, Interface segregation and Dependency inversion. | Acrónimo inventado por Robert C. Martin para establecer los cinco principios básicos de la programación orientada a objetos y patrones de diseño. |

# Referencias

| # | Documento | Descripción | Autor |
|---|---|---|---|
| 1 | Casos de Uso | Diagramas y especificación de los casos de uso del sistema | Esteban Arlando |
| 2 | Guía de Programación | Lineamientos para la programación en lenguaje Java, estándares y buenas prácticas | Ezequiel Fernandez |
| 3 | Composite Pattern: http://en.wikipedia.org/wiki/Composite_pattern | Breve descripción del patrón Composite de GoF | Wikipedia |
| 4 | DDD: Layered Architecture: http://domaindrivendesign.org/node/118 | Breve descripción del patrón Layered Architecture, del libro Domain-Driven Design | Eric Evans |
| 5 | DataMapper: http://martinfowler.com/eaaCatalog/dataMapper.html | Breve descripción del patrón enterprise DataMapper | Martin Fowler |
| 6 | SOLID: http://en.wikipedia.org/wiki/SOLID_%28object-oriented_design%29 | Breve descripción de los cinco principios básicos de la orientación a objetos. | Wikipedia |
| 7 | Diagrama de Contexto | Muestra como se relaciona el sistema con el entorno. | Esteban Arlando |
| 8 | Plan de Negocio | Descripcion general de la compania, mercado y plan de negocio. | Sebastian Arrata |
| 9 | Diagrama de Actividades mas Relevante | Incluye diagramas de activadad del usaurio Standard y Professional. | Esteban Arlando |

# Organización del Documento

## Introducción

En esta sección se provee un rápido anticipo de lo que contiene el documento. Al principio se define el objetivo y el alcance. Se sigue con una tabla de definiciones, acrónimos y abreviaturas que se usarán en todas las secciones. Luego las referencias a otros documentos, internos del proyecto, aportados por el grupo de trabajo de @Solution: artículos de referencia en Internet, bibliografía, etc. Por último se incluye esta sección que describe la organización del documento y detalla el objetivo y el alcance de cada punto.

## Vista Funcional

La Vista Funcional comienza con una introducción al contexto del sistema. Un breve resumen de lo que el sistema debe hacer. Luego, sigue extendiéndose en este análisis funcional, mostrando un diagrama con los casos de uso más relevantes, los que definen al sistema, y una rápida explicación del proceso de negocio involucrado, a alto nivel.

No se espera que el análisis funcional sea completo, ya que para ello están los documentos funcionales y de casos de uso a los que se hace referencia.

## Aspectos Relevantes Para la Arquitectura

Esta sección comienza exponiendo los atributos de calidad que afectan a la arquitectura del sistema; en cada uno, se explicará la forma en la que se espera lograr cumplir con ellos y las decisiones involucradas. Luego sigue una lista de restricciones internas y externas relevadas del contexto en el que se desarrolla el proyecto, restricciones que afectan a la arquitectura. Siguen principios adoptados para la arquitectura, el diseño y el desarrollo del software, principios que se cumplirán a lo largo de toda la construcción del sistema.

## Vista Lógica

La Vista Lógica es el “panorama general” de la aplicación. Permite presentar la estructura del sistema a través de los componentes y sus interacciones. Incluye la especificación completa del modelo de dominio, diagramas de estados de los elementos fundamentales del modelo y otros diagramas que puedan ser útiles a la hora de comunicar la arquitectura del sistema.

## Vista de Interfaces

Incluye un diagrama de contexto en el que se muestran las interacciones del sistema con sistemas externos y un detalle de cada una de las interacciones con el o los modos propuestos de integración. Es muy importante exponer de forma temprana las interfaces externas, ya que representan alto riesgo para el proyecto.

## Vista de Desarrollo

En la vista de desarrollo se incluye información relevante para la construcción de la aplicación, la programación del código fuente. Comienza con una breve descripción de una arquitectura de capas JavaEE 5 estándar y genérica, para luego explicar cómo se puede implementar con productos de Oracle. Luego, sigue información más específica del proyecto como la estructura de proyecto, organización del código, nomenclatura de paquetes y directivas a seguir para lograr una efectiva integración continua.

## Vista de Despliegue

En la Vista de Despliegue se detallan los artefactos generados por el desarrollo y que implementan el sistema, con una explicación de dónde deben ir desplegados esos artefactos, servidores y clientes físicos, y cómo deben ser configurados.

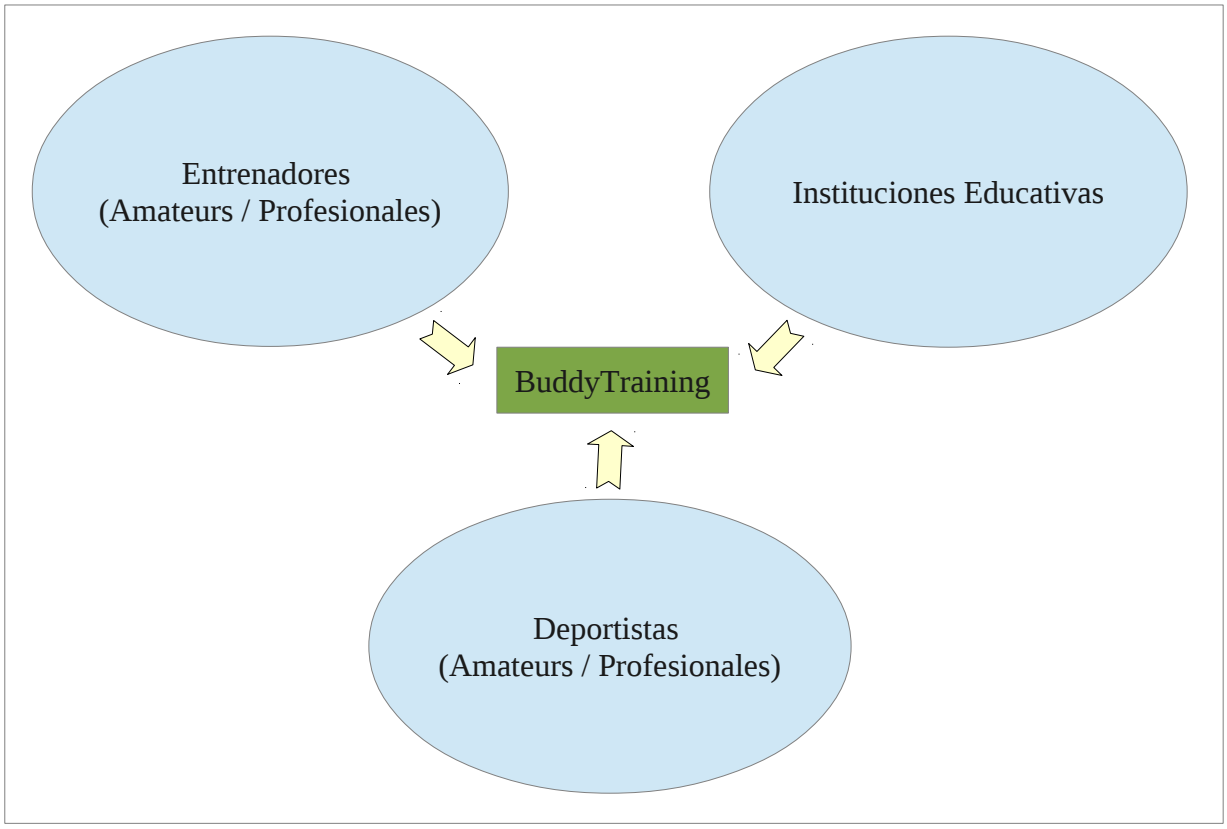
## Selección de la Tecnología

El documento concluye con una sección en la que se fundamentan las decisiones tomadas a la hora de seleccionar o descartar las diferentes tecnologías que fueron evaluadas para el proyecto. En el caso de las tecnologías seleccionadas, se explica por qué fueron incluidas. En el caso de las descartadas, se explica por qué no van a ser utilizadas.

# Vista Funcional

## Contexto

El software @Buddy Training permite controlar la evolucion fisica de los profesionales de la educación deportiva, entrenadores amateurs y a usuarios en general. Para esto contiene un conjunto de planes de entrenamiento precargados para el usuario standar y la posibilidad de ajustar a las necesidades para el usuario professional. Cualquier insititucion / persona sera capaz de medir su evolucion fisica.




Para una vision mas completa de la vista funcional, consultar el documento de Plan de Negocio (ver #8 de la sección de Referencias).

# Casos de Uso

A continuación se muestra un diagrama con los casos de uso que representan la funcionalidad principal del sistema.

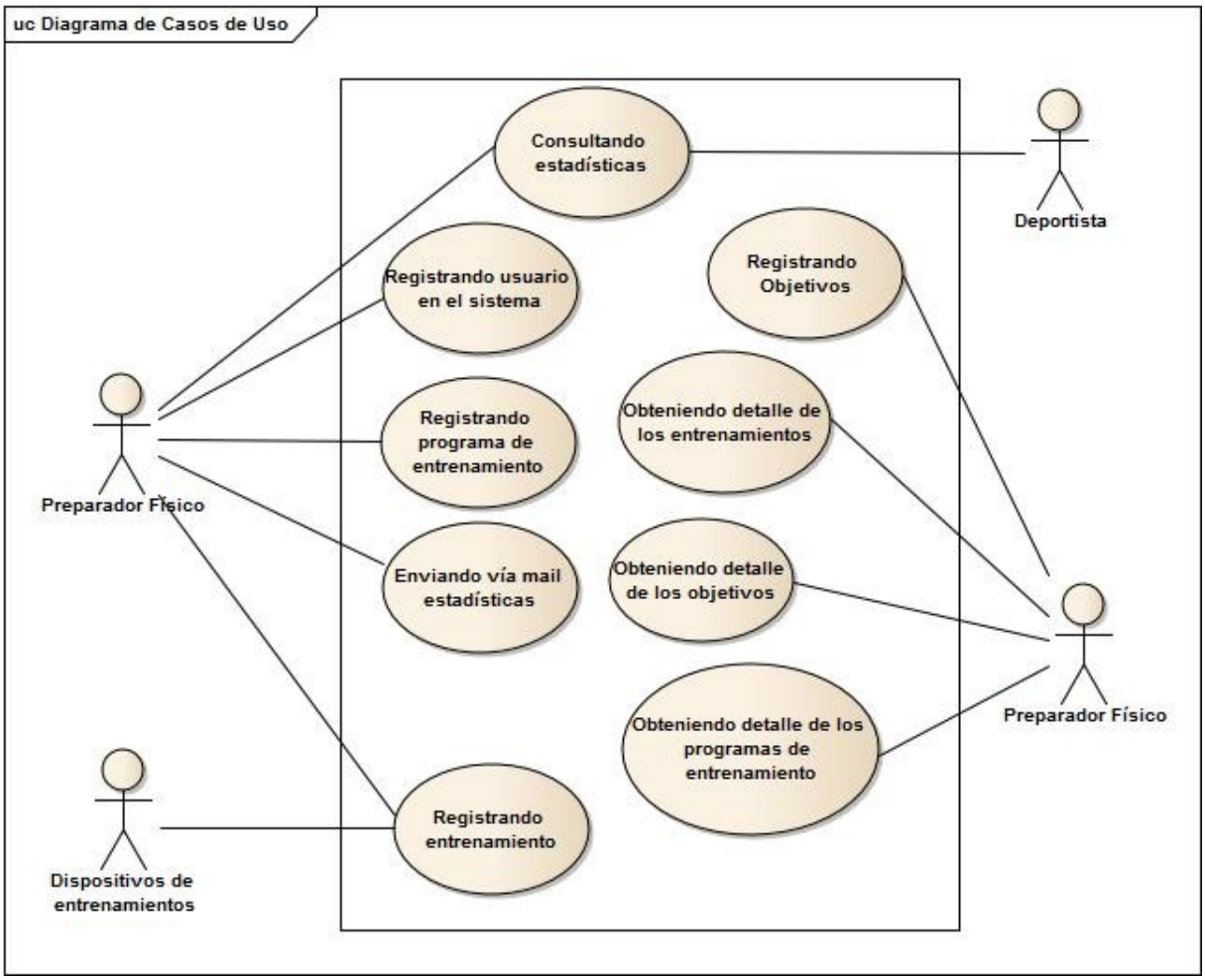



Diagrama de Casos de Uso de alto nivel. Los casos de uso aquí dibujados representan la funcionalidad principal del sistema.

Para ver los diagramas completos de casos de uso y leer la especificación de los mismos, consultar los documentos de Especificación de Casos de Uso (ver #1 de la sección de Referencias).

# Diagrama de Actividades

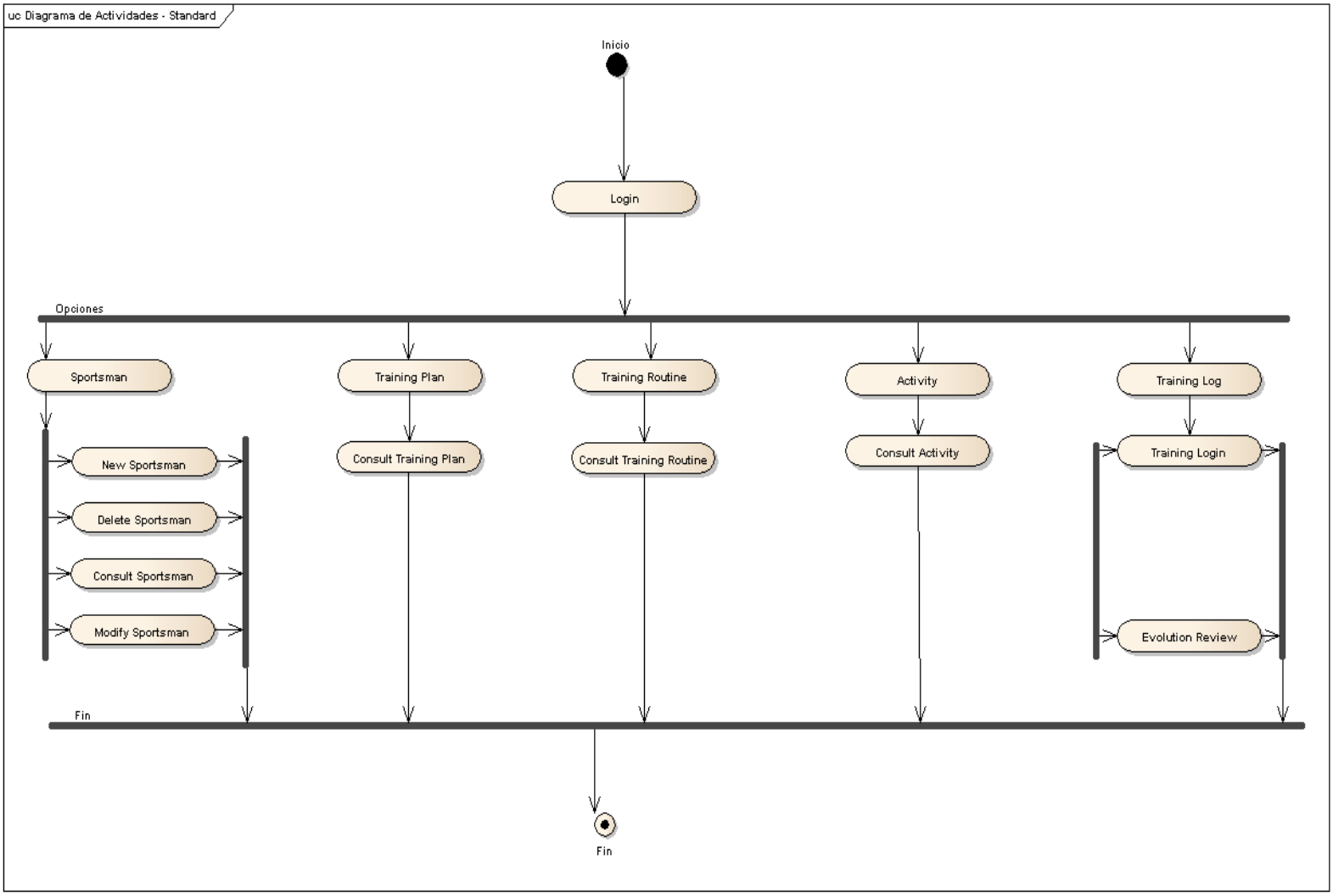
El proceso de BuddyTrainnig esta sujeto al tipo de licencia contratada, actualmente contamos con dos:

- Version Standar: Permite la administracion individual del deportista.
- Version Professional: Permite administrar de forma grupal a deportistas.

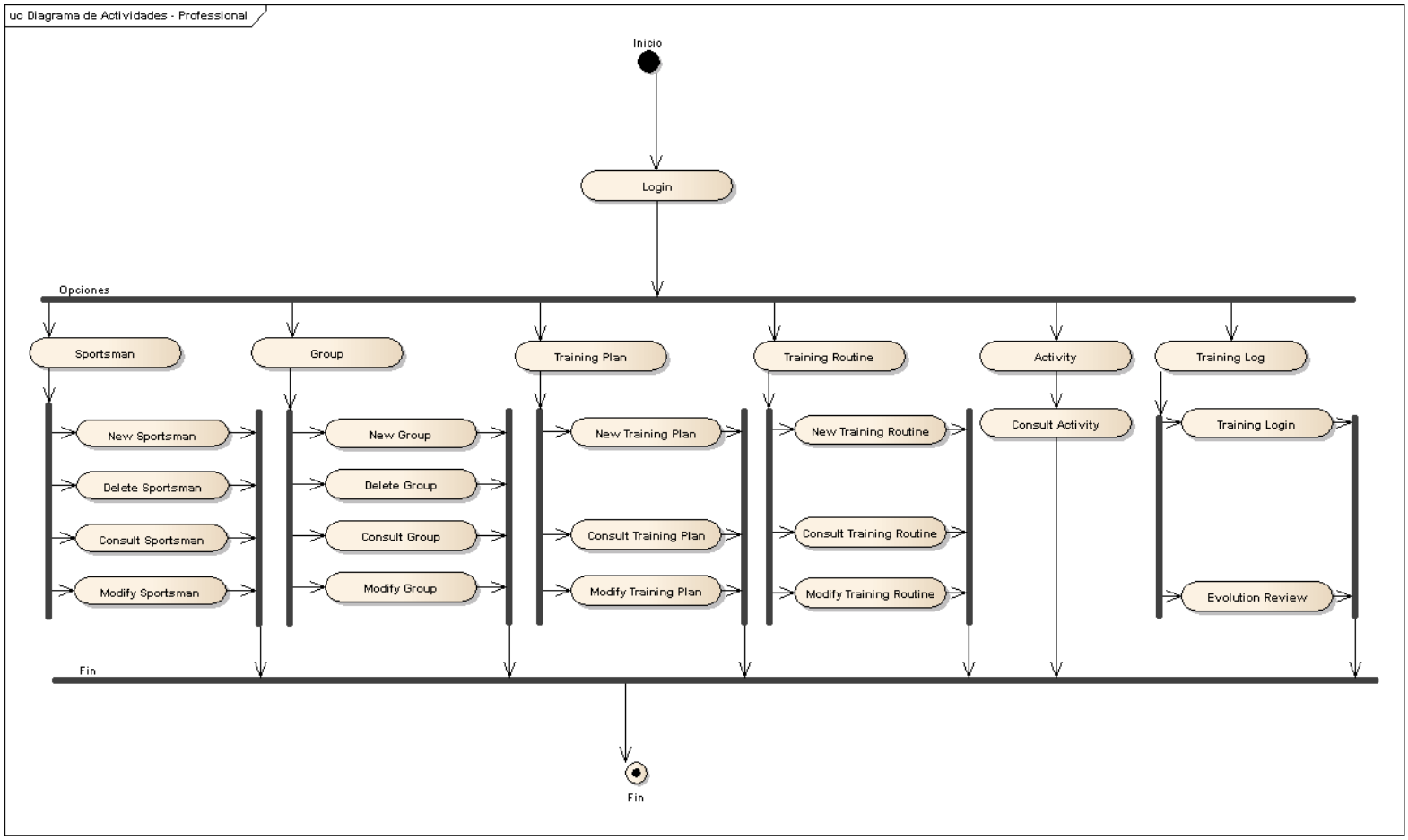
Por lo que el circuito de trabajo en su ciclo de visa son altamente dinamicos y deben ser configurables.

En las siguientes figuras se muestra en forma aislada el diagrama de activdiades de cada uno de ellos.

## Usuario Standard



## Usuario Professional




Para mas informacion consultar los documentos de Diagrama de Actividades mas relevantes (ver #9 de la sección de Referencias).

# Aspectos Relevantes para la Arquitectura

## Atributos de Calidad

En la sección anterior se hizo foco en los requerimientos funcionales del sistema, en qué debe hacer y cómo debe funcionar, desde el punto de vista de los usuarios. La arquitectura de la aplicación se diseñó con el objetivo de satisfacer estos requerimientos, pero no hay que perder foco en los requerimientos no funcionales, que son los que aportan valor a la calidad del producto entregado.

Estos atributos de calidad muchas veces impactan directamente en el diseño de la arquitectura. A continuación se detallan los más relevantes.

### Usabilidad

Este atributo está relacionado con la facilidad con la que un usuario puede cumplir una tarea o utilizar un servicio ofrecido por el sistema. Dado que es importante que el sistema sea cómodo, fácil de usar y su interfaz gráfica sea amigable y agradable a la vista.

Con esto se busca que el usuario pueda:

- Aprender la funcionalidad del sistema fácilmente
- Usar el sistema eficientemente
- Minimizar el impacto de los errores

Para lograr este objeto haremos uso de los componentes que provee la biblioteca de componentes JSF de Oracle ADF. Muchos de estos componentes son complejos pero proveen alto grado de usabilidad.

También será importante, desde el punto de vista de la interfaz gráfica, que prioricemos la estandarización. Por ejemplo: mantener los botones de acciones siempre en los mismos lugares, si hay que realizar cargas de listados que la forma de hacerlo no varíe demasiado en una carga o en otra, y si tenemos que mostrar el mismo listado en distintas pantallas hacerlo siempre de la misma manera.

### Flexibilidad

Nos interesa el atributo de flexibilidad desde el punto de vista del usuario. La flexibilidad relativa a la variedad de posibilidades con las que el usuario y el sistema pueden intercambiar información.

Para lograr esto, proveeremos interfaces de usuario en las que la misma tarea se pueda realizar de distintas maneras, pantallas donde los datos se puedan localizar por diferentes vías, diferentes filtros de búsqueda, los listados se puedan personalizar, organizar de acuerdo a la preferencia del usuario.

## Facilidad de Testeo

Atributo que mide la facilidad que representa un sistema para que se ejecuten sobre él actividades de testing. Para lograr este objetivo, haremos uso de la librería JUnit de Java para realizar test unitarios. Cada capa de la arquitectura de la aplicación podrá ser probada de forma aislada. Cada servicio, cada funcionalidad, tendrá su prueba unitaria asociada. Una funcionalidad que no se pueda probar, no es una funcionalidad que sirva.

Además de JUnit, usaremos Apache Maven como herramienta de building de código. Apache Maven permitirá ejecutar los unit tests de forma masiva y ejecutar pruebas de integración cuando lo amerite, levantando los servidores que las pruebas requieran.

Acompañando a estas herramientas, incluiremos dos más, el controlador de versiones de código, el SVN, provisto por @Solution, y el servidor de integración continua, Hudson. Con Hudson las pruebas unitarias y de integración podrán ejecutarse de forma masiva y periódica y generar alertas cuando las mismas fallen.

Para más información sobre la arquitectura de capas del sistema y cómo se aplicará la integración continua, consultar la sección de Vista de Desarrollo más adelante en este mismo documento.

Además, el área de QA de @Solucion, elaborará casos de prueba para las pantallas del sistema, pudiendo así testear la interfaz de usuario.

## Extensibilidad

Hablamos de extensibilidad desde el punto de vista del código fuente. La extensibilidad es la capacidad de agregar o modificar el comportamiento de un software. La programación orientada a objetos provee una variedad de mecanismos para que el código fuente del sistema sea extensible, como la herencia, el polimorfismo o la encapsulación.

La orientación a objetos permite la aplicación de técnicas de diseño avanzadas mediante sus cinco principios básicos, SOLID (ver #6 de la sección de Referencias), que facilitan la alta cohesión y el bajo acoplamiento.

Por esto es muy importante contar con un Modelo de Dominio orientado a objetos. La tecnología de persistencia elegida, JPA, fomenta la utilización de estos principios. En la sección de Vista Lógica se explica en detalle el modelo de dominio de la aplicación. Se verá que en ocasiones, el modelo se deja preparado para que en futuras versiones pueda extenderse con nueva funcionalidad.

## Reusabilidad

La reusabilidad es la posibilidad de que un segmento de código fuente pueda volver a ser utilizado para agregar nuevas funcionalidades, sin modificación o con modificaciones muy pequeñas. Módulos y clases reutilizables reducen los tiempos de implementación, aumentan la facilidad de testeo, reduce la cantidad de bugs y por supuesto facilita la extensibilidad y la mantenibilidad.

La programación orientada a objetos nos ayudará a conseguir este objetivo. Un modelo de dominio bien estructurado y depurado contribuye a la reusabilidad. La especificación de EJBs también hará lo suyo, facilitando la exposición del dominio en forma de servicios que pueden consumirse de

forma local o remota. Estos servicios fácilmente podrían exponerse como servicios web, si fuera necesario consumirlos desde otra aplicación que no estuviera desarrollada en Java.

## Mantenibilidad

La mantenibilidad hace referencia a la facilidad con la que el producto puede ser mantenido en el futuro, a la hora de encontrar y corregir defectos, aislarlos, implementar nuevos requerimientos. La adhesión a estándares facilitará enormemente la mantenibilidad del código. También la organización del modelo y la arquitectura de capas. Usaremos convenciones y principios que se respetarán en todos los módulos de la aplicación.

Las pruebas unitarias aportarán un valor agregado, ya que muchas veces esto servirá para que quede documentada la forma de utilizar un servicio o alguna funcionalidad específica.

## Portabilidad

La portabilidad es la facilidad de un sistema para poder ser operado en distintas plataformas. Cuando un sistema está desarrollado en Java, obtenemos automáticamente el beneficio de que Java corre sobre una máquina virtual, que es multi-plataforma. Por ende, podemos decir que el sistema es portable entre distintos sistemas operativos.

También podremos obtener alto grado de portabilidad trabajando contra especificaciones siempre que sea posible y no contra las implementaciones de algún proveedor. Actualmente se encuentra especificado cuáles son las funcionalidades estándares que un servidor de aplicaciones JavaEE debe poseer. Cuando hablamos de una implementación, hablamos de Oracle Weblogic, JBoss, Glassfish, por ejemplo.

Siempre que sea posible trabajaremos dentro del estándar (JPA, EJB, JSF) logrando así la portabilidad a nivel servidor de aplicaciones, evitando así un posible proprietary vendor lock-in con Oracle o cualquier otro proveedor.

## Integridad Conceptual

Este atributo hace referencia a la visión subyacente que unifica el diseño del sistema en todos sus niveles. La integridad conceptual es la consideración más importante en el diseño de software. Es mejor que un sistema omita ciertas funcionalidades y mejoras anómalas, pero que refleje un conjunto de ideas de diseño, a tener uno que contiene muchas ideas buenas pero no coordinadas.

Es muy importante para nosotros mantener una integridad conceptual a lo largo de todo el sistema. Esto deberá lograrse a través del modelo de dominio y del lenguaje ubicuo que atraviesa todas las capas de la arquitectura, todas las vistas del sistema y las funcionalidades del negocio.

# Restricciones

A continuación se enumeran algunas de las restricciones y decisiones externas que afectan al diseño de la arquitectura.

## Puesta en Producción

No esta considerada la construccion del producto.

## Servidor de Aplicaciones

La aplicación de BuddyTraining deberá poder ser desplegada en un servidor de aplicaciones Weblogic 10.3.4 con las librerías ADF instaladas versión: 11.1.1.4.

## Oracle Application Development Framework (ADF)

Se deberá utilizar ADF para la aplicación web, la versión 11.1.1.4.

## Base de Datos

El sistema BuddyTraining deberá persistir su modelo de objetos en una base de datos PostgreSQL 9.1.4

## Idioma

Todos los textos que aparezcan en las pantallas de la aplicación web deberán estar en lenguaje inglés. También el código fuente deberá ser escrito en inglés, a excepción de los comentarios que podrán escribirse en español. La documentación será presentada en español.

## Navegadores Web

La aplicación web debe soportar los siguientes navegadores:

- Internet Explorer 7 o superior
- Google Chrome
- Mozilla Firefox
- Safari

# Principios

A continuación se enumeran los principios y/o reglas fundamentales de la arquitectura de la aplicación.

## Modelo de Dominio

- El Modelo de Dominio será implementado en la capa de modelo.
- No se harán accesos directos por JDBC a la base sino a través de JPA. Si se necesita invocar un STORED PROCEDURE se usarán Native Queries.

- Los mapeos JPA se escribirán con anotaciones en las mismas entidades JPA. A menos que la decisión esté muy bien justificada, ninguna entidad tendrá su mapeo en un descriptor XML.
- Los nombres de los atributos de las entidades deberán seguir la convención de Java y ser descriptivos.
- Siempre que se pueda se usarán IDs independientes del negocio que servirán como claves auto-generadas con secuenciales en la base. Se preferirán siempre IDs autogenerados antes que claves compuestas.
- Siempre que se declaren Named Queries para las entidades se proporcionará una constante en la misma entidad con el nombre de la Named Query para que pueda ser referenciada desde la capa de servicio a través de ella.
- Nunca se ejecutarán queries en el modelo.
- Nunca se ejecutará lógica de negocio en el modelo. Sólo se podrá incluir lógica simple de acceso a los atributos como métodos que faciliten la carga de una lista o la manipulación de algún atributo.
- El modelo no debe tener dependencia con ningún otro proyecto de BuddyTraining.
- Todos los objetos Mocks generados para los Unit Test se construirán en objetos MockBuilders ubicados en un paquete dedicado.
- Cada cambio en el modelo deberá estar debidamente justificado.

## Lógica de Negocio

La lógica de negocio será implementada en la capa de servicios y siempre dentro de un EJB.

Siempre se declarará una interfaz Local para ser accedida desde la aplicación web y una interfaz Remota para ser accedida desde los Unit Test.

El nombre de la interfaz Local de un EJB siempre terminará en Service.

Por ejemplo:

```
SportsmanService
```

El nombre de la interfaz Remota de un EJB siempre terminará en ServiceRemote y, a menos que no sea conveniente por alguna razón particular, siempre extenderá la interfaz Local.

Por ejemplo:

```
SportsmanServiceRemote extends SportsmanService
```

El nombre de la implementación de un EJB siempre terminará en ServiceBean.

Por ejemplo:

```
SportsmanServiceBean
```

A menos que se declaren métodos específicos a una interfaz remota, la implementación de un EJB siempre implementará por lenguaje la interfaz Local, y hará referencia por anotaciones a la interfaz Local y a la Remota.

Por ejemplo:

```
@Stateless(name = "SportsmanService", mappedName = "javaee-poc/SportsmanService")
@Local(value = SportsmanService.class)
@Remote(value = SportsmanServiceRemote.class)
public class SportsmanServiceBean implements SportsmanService { ... }
```

Las entidades siempre se sincronizarán con la unidad de persistencia a través del Entity Mana JPA. El Entity Manager siempre se inyectará en los EJBs a través de la anotación @PersistenceContext, a menos que el acceso de otra forma esté debidamente justificado.

Siempre que se quiera invocar alguna query se hará a través de Native Queries, a menos que el uso de Dynamic Queries o Native Queries esté debidamente justificado.

Siempre que un EJB requiera el uso de otro EJB se inyectará con el uso de la anotación @EJB. Nunca se localizarán con JNDI.

## Aplicación Web

El nombre de los Managed Beans de ADF y de JSF deberán terminar con la palabra Controller.

Por ejemplo:

```
SportsmanController
```

Las extensiones de las vistas serán jsf y los nombres se escribirán en minúscula y con guión del medio para separar las palabras.

Por ejemplo:

```
create-sportsman.jsf
```

No se accederá al Entity Manager de JPA de ninguna manera desde la aplicación web. Tampoco se

armará ninguna query. Todo lo que implique un cambio en la unidad de persistencia se hará a través de servicios de EJBs.

No se escribirá lógica de negocio en la aplicación web. Sólo lógica que esté asociada a los input y los output del usuario. Cada vez que se tenga que escribir lógica de negocio, se creará un EJB y se hará la invocación del mismo desde un Controller.

Si se usan Managed Beans de JSF y es posible la inyección de EJBs a través de @EJB, se preferirá antes del uso del Service Locator.

Si se usa Managed Beans de ADF, las instancias de los EJBs se recuperarán a través del Service Locator.

Siempre se preferirá el uso directo de atributos de los Controllers desde la Vista antes que el binding de los componentes de ADF para que sean accedidos a través de sus correspondientes clases Java.

## Guía de Programación

Para la codificación en el lenguaje Java se seguirá la Guía de Programación referenciada (ver #2 de la sección de Referencias).

## Logging

No se imprimirá nunca en la consola directamente.

Se prohíbe el uso de:

```
System.out.println()
```

Para la aplicación de BuddyTraining utilizaremos la librería de Apache Log4j y siempre imprimiremos mensajes en nivel DEBUG o, si lo amerita, en nivel INFO.

## Manejo de Excepciones

Se usarán sólo excepciones no chequeadas, a menos que se justifique muy bien el uso de una excepción chequeada.

Siempre que algún método del mismo JDK o de una librería de terceros requiera atrapar una excepción chequeada, se atrapará y se tratará allí mismo si es posible o en su defecto se volverá a lanzar hacia arriba encapsulándola en una RuntimeException con un mensaje de error personalizado y la misma excepción que fue atrapada.

Por ejemplo:

```
catch (Exception e) {
        throw new RuntimeException(e, “Algún mensaje de error bien descriptivo”);
}
```

No se usarán excepciones tipadas, a menos que se justifique por alguna razón. Por ejemplo: se quiera transportar algún código de error o algún otro dato y se necesiten agregar atributos. En ese caso siempre se extenderá de RuntimeException.

Siempre que encapsulamos una excepción chequeada en una excepción no chequeada para volverla a lanzar debemos incluir un mensaje descriptivo. Si se le puede agregar información de los parámetros o las variables que causaron el error mejor. Recordar que cuanto más descriptivos sean los mensajes, más fácil será la detección de errores en el futuro.

### Uso del Modelo en las Demás Capas

Las entidades de JPA podrán ser accedidas desde todas las capas. Esto se explica con más detalle en la Vista de Desarrollo, en la sección llamada **“Sin Objetos DTO”**.

Cuando la lógica de la aplicación web sea muy complicada, se podrán construir objetos View que contengan la información necesaria que se debe mostrar, para no tener que hacer modificaciones innecesarias en el modelo.

*Los objetos View sólo deben declararse si son necesarios.*

Cada objeto View deberá poseer un método de creación llamado create() que reciba el o los objetos de modelo que están wrappeando y devuelva el objeto view en cuestión. También estos métodos create() podrán devolver listas si se necesitan.

## Fuera de Alcance

A continuación se incluye una lista de las funcionalidades que quedarán fuera de alcance en esta versión del sistema:

### Soporte para Multi-Lenguaje

La interfaz gráfica de la aplicación BuddyTraining no tendrá soporte para distintos lenguajes. Todos los textos aparecerán en las pantallas en idioma inglés.

# Vista Logica

## Modelo de Dominio

### General

A continuación se muestran los diagramas de modelo de dominio de alto nivel de la aplicacion BuddyTraining.

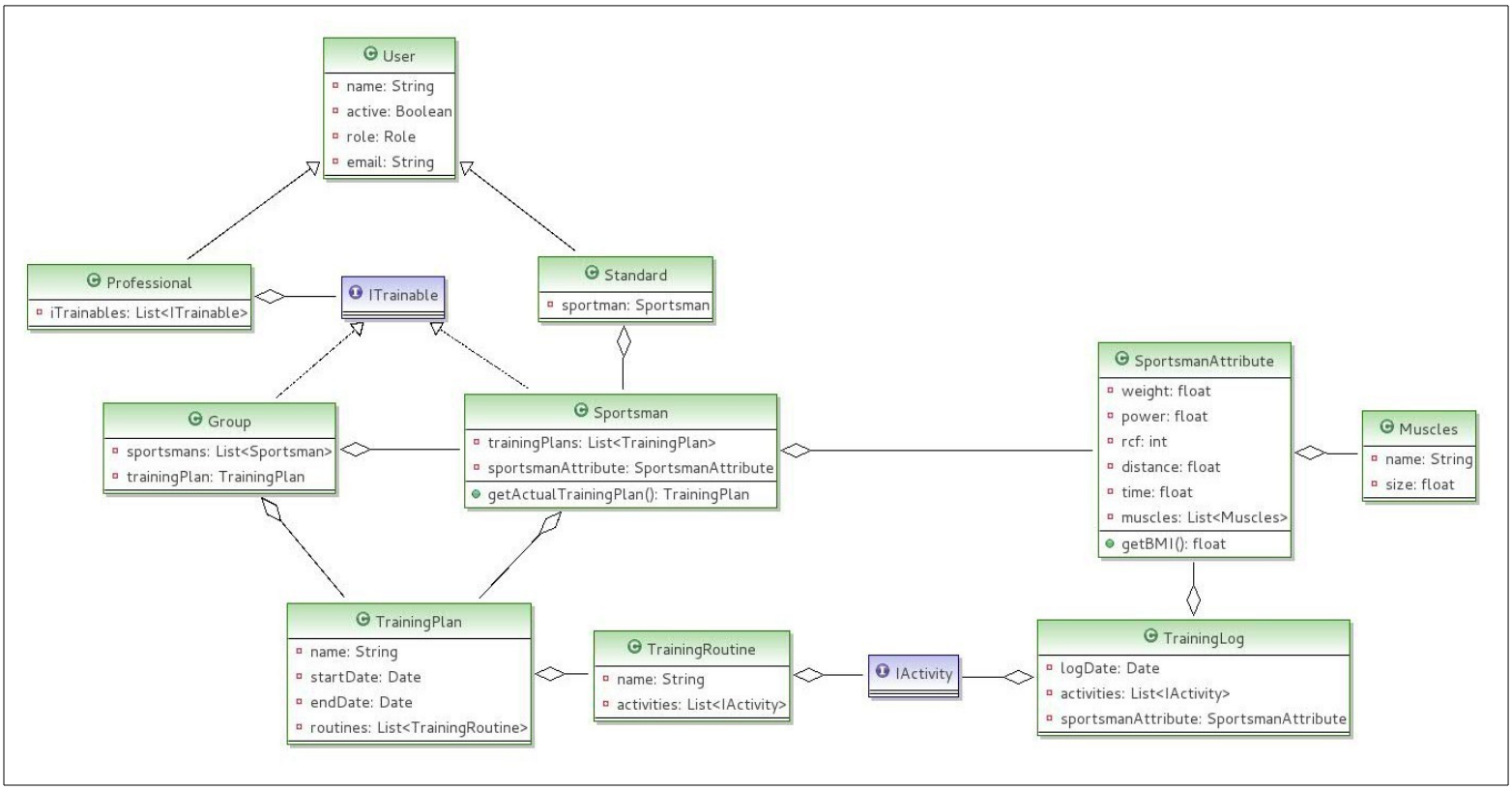



Figura que representa los dos tipos de usuario / licencia que va a manejar la aplicacion, para el caso del Professional se estima que maneje grupos o deportista de forma individual, aplicando sea el caso el plan de entrenamiento a cada uno de sus entrenados. Para el caso de Standard se aplica de la misma forma que la Professional pero sin manejo de grupos.

#### *User*

Representa los usuarios que utilizan la aplicacion.

#### *Professional*

Representa a los usuarios que entrenan otra deportistas, son los entrenadores que pueden

adminsitrar grupos o deportistas de forma individual.

### *Standard*

Representa a los usuarios que entrenan de forma autonoma y que utilizan el sistema como administrador de su entrenamiento.

### *Group*

Representa un conjunto de entrenados propios de los usuarios Professional, es decir los Sportsman.

### *Sportsman*

Representan a los deportista que van a ser entrenados y auditados por la aplicacion.

### *Training Plan*

Representan donde se agrupan las rutinnas de entrenamiento, son los objetivos del entrenamiento. Por ejemplo: Bajar de Peso, Aumentar la resistencia, etc.

### *Training Routine*

Representan las rutinas que debe realizar cada ves que se entrena, son equivalente a los formullarios que acompañan a los sportsman con las actividades a realizar.

### *Activity*

Representan las actividades concretas, estas pueden ser por ejemplo: correr, abdominales, flexiones de brazos, etc.

### *Sportsman Atributte*

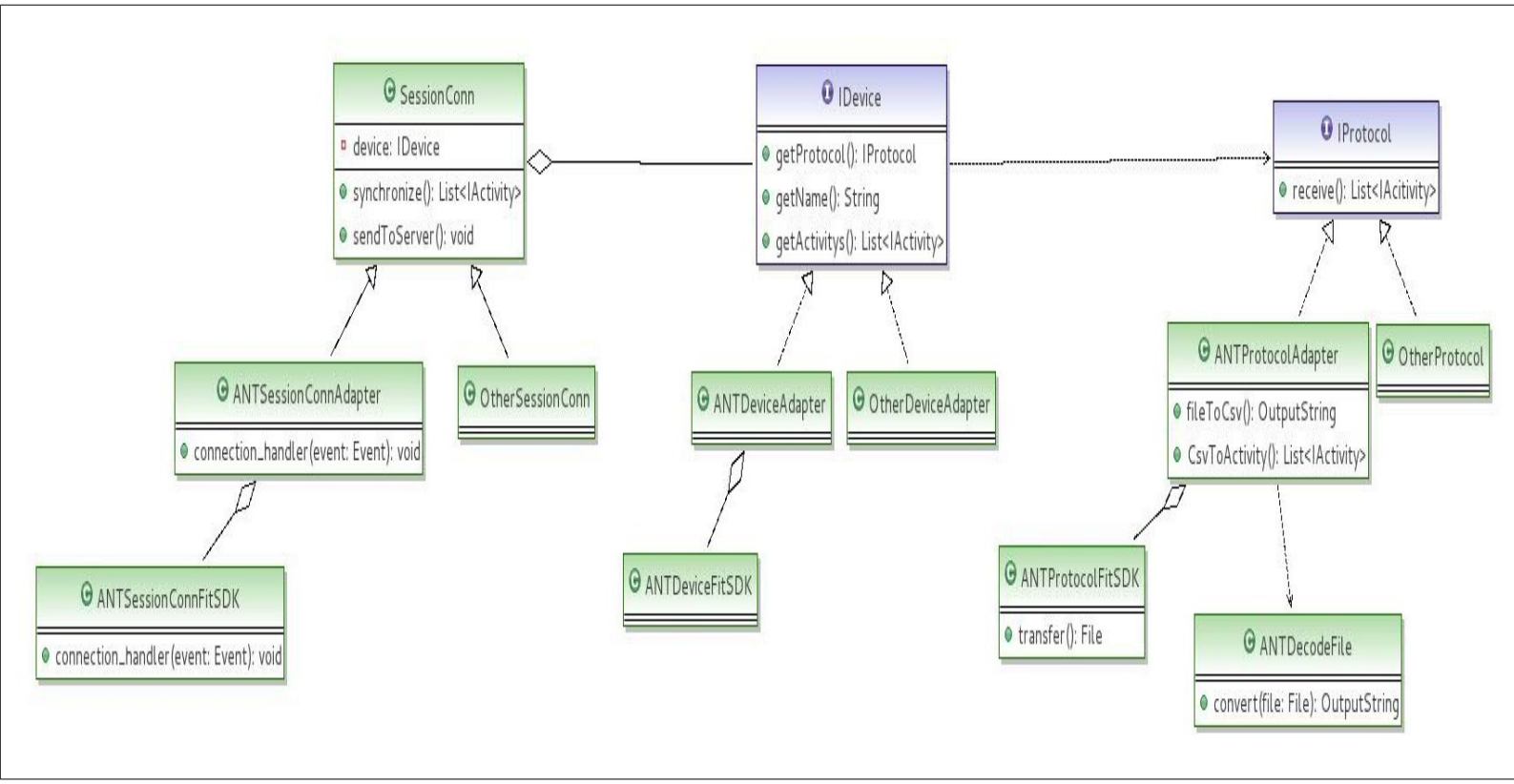
Representan los indicadores necesarios para marcar la evolucion del entrenado, por ejemplo: peso, IMC, ritmo cardiaco. Estos indicadores deben ser tomados cada ves que se realiza una rutina para poder representar un evolucion o no.

### *Training Log*

Representa el registro de entrenamiento del deportista, asi se puede relacionar las actividades realizadas, la fecha.

## Integracion con Protocolos

A continuación se muestran los diagramas de modelo de dominio de alto nivel de la aplicacion respectode la integracion con los distintos dispositivos y protocolos.




### *SessionConn*

Permite adiminstrar el estado de los dispositovos, como asi tambien avisar cuando se establecer una comunicacion.

### *ANTSessionConnFitSDK*

Libreria / Clase que viene incluida en el pack de desarrollo de ANT+. Son sessiones propias del framework.

### *ANTSessionConnAdapter*

Adapter de ANTSessionConnFitSDK que permite ademas recibir la notificacion cuando un dispositivo es conectado.

### *IDevice*

Interfaz necesaria para homogeneizar los dispositivos dependiendo de la tecnologia a utilizar.

### *ANTDeviceFitSDK*

Libreria / Clase que viene incluida en el pack de desarrollo de ANT+. Son los dispositivos propias del framework, maneja mucha informacion que no es necesaria para nuestra aplicacion.

### *ANTDeviceAdapter*

Adapter de ANTDeviceFitSDK.

### *IProtocol*

Interfaz necesaria para homogeneizar el manejo de los protocolos.

### *ANTProtocolFitSDK*

Libreria / Clase que viene incluida en el pack de desarrollo de ANT+. Es la encargada de admnistrar de forma transparente la transferencia de informacion a traves del protocolo ANT.

### *ANTProtocolAdapter*

Adapter de ANTProtocolFitSDK

### *ANTDecodeFile*

Libreria / Clase que viene incluida en el pack de desarrollo de ANT+. Es la encargada de la conversion del formato fit a cvs.

### *Framework ANT+*

ANT+ FitSDK provee mecanismo de administracion de dispositivos, esto quiere decir que se encarga de controlar si un dispositivo esta disponible o no. Esto es totalmente transparente para nuestra aplicacion, por lo que el proceso de sincronizacion se haria de la siguiente forma:

1) Cuando un dispositivo solicita sincronizar los datos, FitSDK dispara el metodo "connection_handler" el cual hay que caputurar dado que contiene la informacion del dispositivo que solicito la sincronizacion. Esto se hace subcribiendose a la accion de la siguente manera:

```
public class ANTSessionConnAdapter extends SessionConn {

        private ANTSessionConnFitSDK sessionConn;

        public ANTSessionConnAdapter(){
                sessionConn.addConnectionActionListener(new ActionListener() {
```

```
                        public void actionPerformed(ActionEvent e) {
                                //metodo a invocar cuando se conecte un dispositivo
                                connection_handler(event);
                        }
                }
        }

        public void connection_handler(event){
                //Nos guardamos el dispositivo para establecer la comunicacion para la
                // trasferencia. Tambien podemos disparar un evento a la pantalla avisando
                //que se conecto un dispositovo
                super.device = (DeviceFitSDK) event.getDeviceSource();
                System.out.println("El dispositivo conectado es: " + device.toString();
        }

}
```

2) Una ves establecida la comunicacion, el usuario debe ejecutar de forma manual el proceso de sincronizacion, esto es:

- Transferencia de los *.fit del dispositivo a la aplicacion
- Convertir esos *.fit en *.csv
- Procesar esos *.csv y generar las actividades resultantes.

# Vista de Interfaz

## Diagrama de Contexto

Figura: Aquí se pueden ver los sistemas externos y las interacciones con los dispositivos externos. Muestra la interaccion de los diferentes actores con el sistema. Por un lado tenemos al preparador fisico (licencia professional) quien puede administrar grupos o deportista individuales. Por otro lado el deportista (licencia stadard) el cual es asistido por la aplicacion para la administracion de su plan de entrenamiento. Por ultimo, los dispositivos de entrenamientos que permiten la carga automatica de informacion para medir el desempeño / evolucion de su estado fisico.

Para ver una descripcion completa del diagrama, consultar los documentos de Diagrama de Contexto (ver #7 de la sección de Referencias).

Entre los sistemas externos que identificamos, son:

- Dispositivos de entrenamientos con sus propios software de administracion.
- Protocolo de comunicacion ANT+ y de otros vendors.
- Sistemas externos que utilizen nuestros servicios expuestos (API)

# Vista de Desarrollo

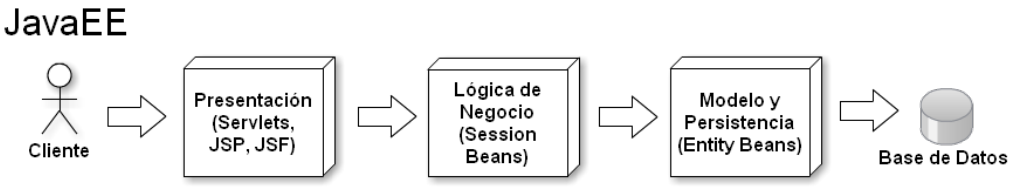
## Introducción

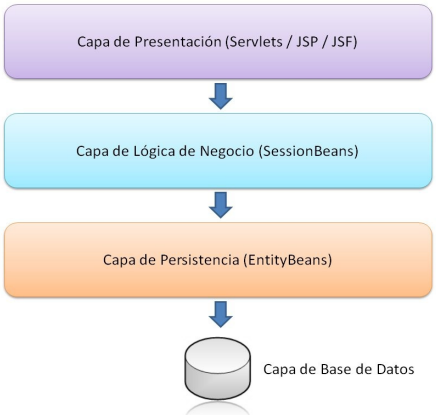
Para comprender la arquitectura propuesta para la aplicación, se va a empezar la investigación explicando conceptos básicos de arquitectura java EE 5 por los cuales esta basado nuestra aplicación. Luego se abordara en forma detallada las distintas capas que conforman la arquitectura propuesta, finalizando con la descripción de despliegue.

## Arquitectura JavaEE 5

La plataforma JavaEE son los cimientos por los que se construirá la aplicación propuesta, dado que ofrece una arquitectura de capas determinada para las aplicaciones empresariales. El objetivo principal es que cada capa tenga una responsabilidad bien definida y así lograr una clara separación de conceptos. No es bueno trabajar en un código fuente que mezcla lógica de negocio con lógica de aplicación. Cuando pensamos en cómo persistir objetos de negocio en un almacenamiento físico o cómo mostrarlos estamos pensando en ambos casos en lógica de aplicación, en tecnología; la persistencia por el lado de la infraestructura, el mostrar los objetos por el lado de la interfaz de usuario. En cambio, cuando pensamos en cómo calcular el descuento para un cliente que compra un producto o cómo calcular el porcentaje de avance de un deportista, estamos pensando en lógica de negocio, que es independiente de la tecnología.

Por esto es necesaria la separación en capas:




En la figura se muestra dos distintas representaciones de la arquitectura JavaEE 5 típica. Los clientes acceden a través de la capa de presentación, donde viven los Servlets, páginas JPS y páginas JSF. La capa de presentación invoca métodos o servicios de la capa de Lógica de Negocio, donde viven los EJB SessionBeans. Los EJBs son los que ejecutan la lógica de negocio y manipulan las entidades JPA, los EntityBeans, que se sincronizan con la base de datos.

## Base de Datos

En el modelo de arquitectura JavaEE el almacenamiento físico generalmente es una base de datos relacional.

## Modelo y Persistencia

En esta capa, muchas veces llamada simplemente modelo, es en donde viven los objetos de negocio, las entidades y los objetos de valor. A partir de JavaEE 5, la especificación estándar de esta capa es la Java Persistence API (JPA).

JPA es la especificación de un Object-Relational Mapper. Los ORM están basados en el patrón DataMapper.

En esta capa conviven los objetos de negocio, objetos Java comunes, con sus respectivos mapeos objeto-relacionales, configuración embebida en el código en forma de anotaciones o separada en archivos XML que le indican a JPA cómo debe persistirse y recuperarse los objetos en la base de datos.

## Sin DER (Diagrama Entidad Relación)

La arquitectura propuesta por JEE, sugiere la utilización de un ORM para la comunicación y administración de la base de datos, basándose en el modelo de objetos de la aplicación. Esto nos permite una mayor abstracción en nuestra aplicación, dado que antes de la aparición de ORM, las consultas se tenían que realizar manualmente dentro de los propias aplicaciones, con lo cual la ventaja de los lenguajes orientados a objetos se perdía, ya que había que generar una petición a la base de datos de manera manual (y especifica para cada sistema, ya que no todos los gestores de base de datos tienen la misma implementación de SQL). La ventaja principal de estos sistemas es que reduce la cantidad de código necesario para lograr lo que se conoce como una persistencia de objetos. Esto permite además lograr una mejor integración con otros patrones como el MVC donde el modelo puede ser este objeto.

## Lógica de Negocio

En esta capa, muchas veces llamada capa de servicio, es en donde viven los Enterprise JavaBeans (EJB). Los EJB son objetos Java que encapsulan de forma transparente gran cantidad de lógica de aplicación, requerimientos no funcionales como: concurrencia, control de hilos de ejecución, acceso remoto, transacciones de negocio, seguridad, timers, interceptores, etc.

En los EJB se debe ubicar la lógica de negocio que manipulará las entidades de negocio de la capa de modelo.

## Presentación

En esta capa, muchas veces llamada capa de aplicación, se ubica la lógica relacionada con la interfaz de usuario. En general las presentaciones de las aplicaciones JavaEE suelen ser aplicaciones

web, pero no siempre. La interfaz de usuario podría ser una aplicación desktop o una aplicación para dispositivos móviles. Incluso podría no haber interfaz de usuario y esta capa de aplicación podría ser simplemente un cliente liviano que consuma los servicios que expone la lógica de negocio para entregar los resultados a otra aplicación.

Pero en JavaEE, en la mayoría de los casos, cuando se habla de capa de presentación, se habla de aplicación web, y una de las especificaciones más importantes en esta incumbencia es la de JavaServer Faces (JSF).

JSF especifica un framework RAD (Rapid Application Development) orientado a componentes para aplicaciones web desarrolladas en Java, que se monta sobre la tecnología de JavaServer Pages (JSP). Se basa en el patrón de arquitectura Model-View-Controller (MVC).

En JSF la Vista se escribe en archivos XHTML (las extensiones pueden variar; algunas de las más usadas son xhtml, jsf, jsp) con taglibs especiales que representan componentes visuales o widget, como botones, etiquetas, inputTexts, etc.

Los Controladores son los que implementan la lógica del lado del servidor, la lógica Java asociada a una Vista.

El Modelo son los objetos de negocio, las entidades de la capa de modelo, y la lógica de negocio asociada (si hablamos de EJBs, serían los servicios que exponen los EJBs).

## Cliente

El Cliente de una aplicación web puede ser cualquier cliente HTTP, comúnmente un navegador web.

## Sin Objetos DTO

Una de las ventajas que propone el modelo de desarrollo de JavaEE 5 es la posibilidad de utilizar las entidades JPA en todas las capas.

En una arquitectura tradicional de capas, una determinada capa X sólo puede acceder a las funcionalidades de una capa Y que sea inmediatamente inferior. No puede acceder a funcionalidades de capas más inferiores y mucho menos de una capa superior.

Esta arquitectura forzaba a los desarrolladores a la construcción de objetos View u objetos Data Transfer Object (DTO) que eran una copia de los objetos del modelo pero que podían compartirse entre todas las capas. Más concretamente: había que encontrar una forma de que la información de los objetos de negocio viajaran por todas las capas en un sentido y en el otro.

JPA propone usar POJOs para la representación de las entidades, objetos Java comunes, que a la vez puedan usarse para transferir información entre las capas.

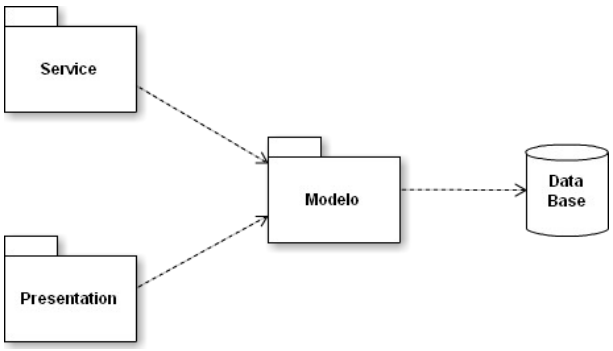
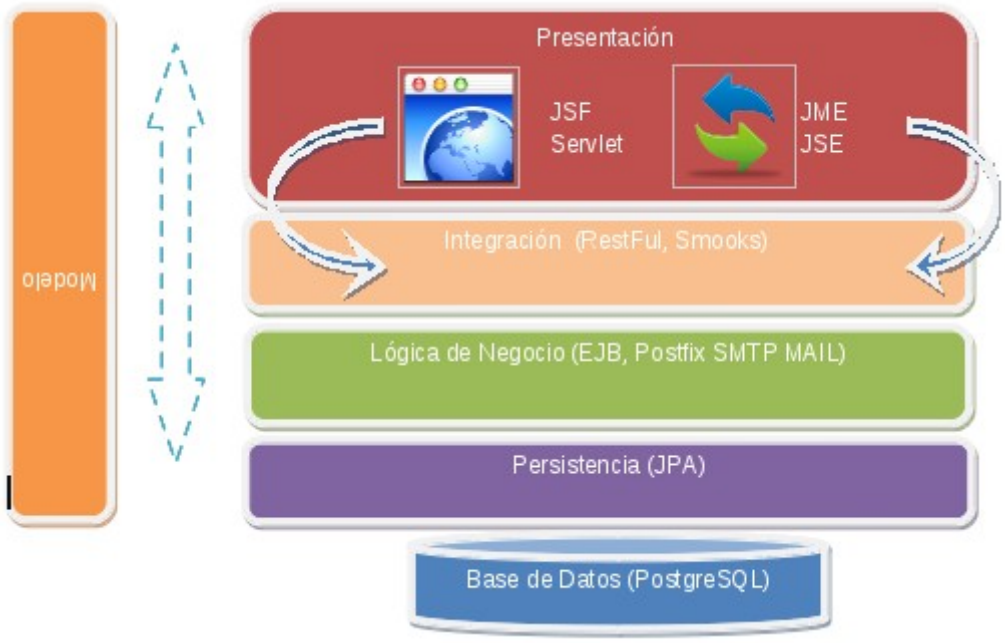



Diagrama de paquetes de las capas de una arquitectura JavaEE 5 típica. Las flechas muestran dependencias entre capas. En el diagrama se ve cómo tanto la capa de Servicio como la de Presentación pueden manipular los objetos de negocio del Modelo.

El contexto de persistencia vive en la capa de lógica de negocio. La sesión de negocio es abierta y cerrada de forma transparente por los EJBs. Esto quiere decir que cuando la capa de presentación manipula objetos de negocio lo que en realidad usa son entidades JPA no sincronizadas con el motor de persistencia, lo que en un ORM se conoce como entidades detached.

# Arquitectura Propuesta

Para el proyecte @Buddy se propone una arquitectura orientada a Servicio (SOA) estructurada en 4 capas, presentación, integración con los servicios, lógica de negocio y repositorio de datos. A continuación se explicara cada una de las capas representadas:




## Presentación

Esta capa representa los clientes que se conectan a la aplicación, es decir aquellos que interactúan con las capas inferiores (Integración (WebService) y Lógica de Negocio (EJB)), estos clientes puede ser:

### *Cliente Web Browsers*

Este punto de entrada, desde el punto de vista de la carga solo permite hacerla de forma manual, dado que nos conectamos con un navegador que interpreta código html. No obstante es ideal para realizar las consultas de rendimiento y demás.

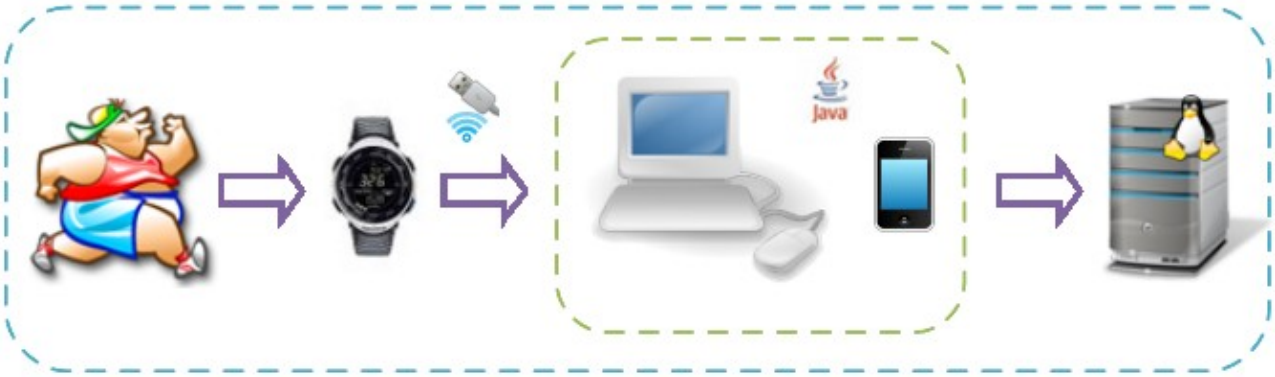
Navegadores Soportados:

- Internet Explorer 7 o superior
- Google Chrome
- Mozilla Firefox
- Safari

### *Cliente Aplicación Java para la Sincronización de datos al Servidor*

El motivo de esta aplicación es la de recibir los datos suministrados por los distintos

dispositivos de entrenamientos para luego enviarlos al servidor, entre los cuales se proponen, la creación de 2 aplicaciones, la primera desarrollada sobre Java SE para PC y la segunda en Java ME para mobile (en esta categoría entran los celulares clasificados como Smartphone y Tablets).




**Java SE** permite la creación de aplicaciones multiplataforma orientadas para el escritorio, esto nos permite que nuestra aplicación sea extensible a cualquier sistema operativo moderno, entre los cuales se destacan:

- Windows Vista / 7
- Mac OS
- Linux

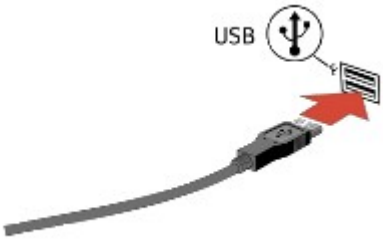
Jave ME al igual que Java SE permite la creación de aplicaciones multiplataforma orientadas a distintos dispositivos mobile optimizadas para este fin, esto nos permite que nuestra aplicación sea extensible a cualquier sistema operativo moderno para esos dispositivos, entre los cuales se destacan:

- Android
- IPhon
- Blackberry

### *Sincronización Asincrónica de Datos al Servidor*

Unas ves generadas la rutina de entrenamiento con el dispositivo correspondiente (Reloj, pulsera, etc), se procede a la descarga total de la información generada para luego transmitirla al servidor.

1. Como primera medida, es necesario establecer la comunicación entre el dispositivo receptor (PC o Mobile) y el dispositivo de entrenamiento emisor a través del USB, Bluetooth o Wifi, para el caso de comunicaciones inalámbricas se debe respetar la banda ISM que establece un rango de frecuencia permitido para operar. (Para más información ver Documento de Investigación).




2. Una vez establecida el canal de comunicación, el dispositivo de entrenamiento envía los datos de la actividad realizada al receptor, esto es mediante el protocolo ANT++, permitiendo en la transferencia seguridad e integridad de datos (Para más información ver Documento de Investigación).

3. El dispositivo receptor transfiere los datos a la aplicación para luego enviarla al servidor a través del web service expuesto (capa de integración).


### *Sincronización Sincrónica de Datos al Servidor*

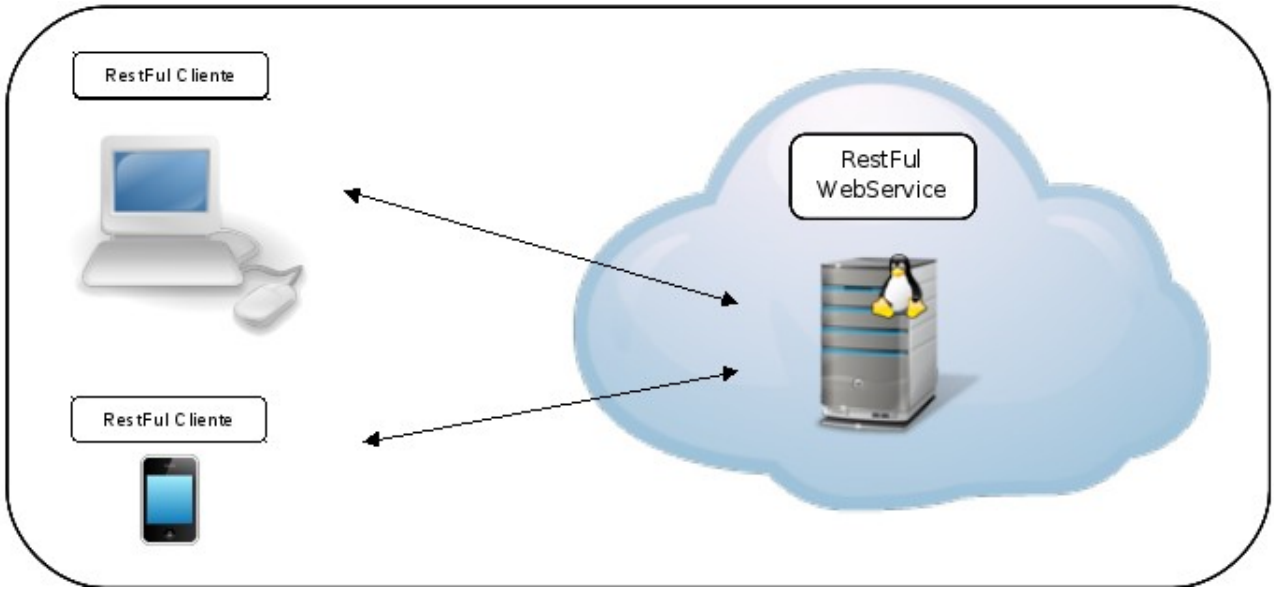
La particularidad con lo visto anteriormente, es que los datos capturados en los dispositivos de entrenamientos (el emisor), son sincronizados al servidor en forma inmediata o en un lapso no mayor a 1 minuto, permitiendo a los entrenadores llevar un control en tiempo real del rendimiento de los deportistas.

## Integración

Esta capa es el punto de interacción para aquellas aplicaciones que quieran consumir los servicios que la aplicación ofrece, en nuestro caso particular son las aplicaciones de sincronización de datos. Esta capa como las que vamos a ir viendo, se ejecutan dentro del servidor de aplicaciones. Esto nos permite hacer mucho mas extensible nuestra aplicación dado que la interfaz de comunicación es atreves de XML, todo esto es posible gracias a RESTful como WebService y Smooks para la trasformación de los XML a Objetos Java.

### *RESTful (WebService)*

REST define un set de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. REST emergió en los últimos años como el modelo predominante para el diseño de servicios. De hecho, REST logró un impacto tan grande en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de usar.




### Los 4 principios de REST

Una implementación concreta de un servicio web REST sigue cuatro principios de diseño fundamentales:

- utiliza los métodos HTTP de manera explícita
- no mantiene estado
- expone URIs con forma de directorios

- transfiere XML, JavaScript Object Notation (JSON), o ambos

**URI: Uniform Resource Identifier**

Un Uniform Resource Identifier o URI (en español «identificador uniforme de recurso») es una cadena de caracteres corta que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.).

Aunque se acostumbra llamar URL a todas las direcciones web, URI es un identificador más completo y por eso es recomendado su uso en lugar de la expresión URL.

Un URI se diferencia de un URL en que permite incluir en la dirección una subdirección, determinada por el “fragmento”.

A continuación vamos a ver en detalle estos cuatro principios, y explicaremos porqué son importantes a la hora de diseñar un servicio web REST.

### REST utiliza los métodos HTTP de manera explícita

Una de las caraterísticas claves de los servicios web REST es el uso explícito de los métodos HTTP, siguiendo el protocolo definido por RFC 2616. Por ejemplo, HTTP GET se define como un método productor de datos, cuyo uso está pensado para que las aplicaciones cliente obtengan recursos, busquen datos de un servidor web, o ejecuten una consulta esperando que el servidor web la realice y devuelva un conjunto de recursos.

REST hace que los desarrolladores usen los métodos HTTP explícitamente de manera que resulte consistente con la definición del protocolo. Este principio de diseño básico establece una asociación uno-a-uno entre las operaciones de crear, leer, actualizar y borrar y los métodos HTTP. De acuerdo a esta asociación:

- se usa POST para crear un recurso en el servidor
- se usa GET para obtener un recurso
- se usa PUT para cambiar el estado de un recurso o actualizarlo
- se usa DELETE para eleminar un recurso

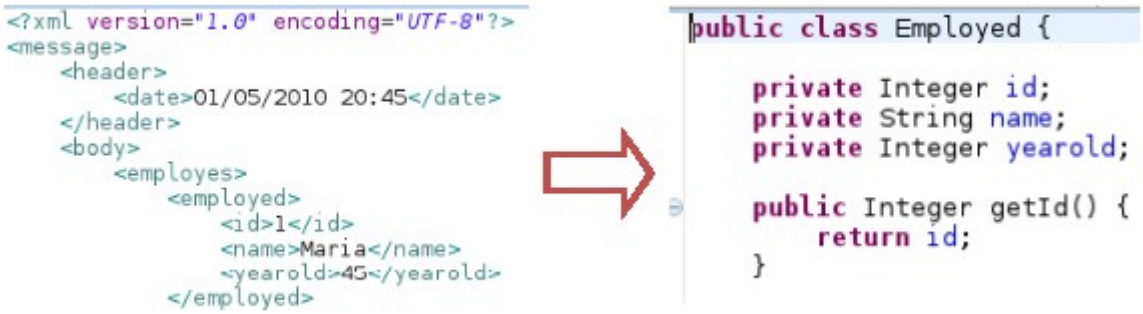
Cabe destacar que los servicios de las redes sociales como Flickr, Twitter, Facebook, etc son basados en RESTful. Por lo que abordar en esta tecnología nos adelanta hacia futuras integraciones con estos servicios.

### *Smooks (Motor de Transformaciones)*

Smooks es un motor de transformaciones que nos permite de forma sencilla convertir XML (y otra diversidad de opciones que no son necesarias para el proyecto) a Objetos y viceversa.


De esta forma cuando las aplicaciones clientes requieren enviar información al servidor, esta se recibe en formato XML y luego es convertida a Objetos Java para ser procesado por el negocio. Sucede de la misma forma cuando lo que se genera es una consulta al servidor por el pedido de información.

### Ejemplo de transformación de XML a Java



```xml
<?xml version="1.0" encoding="UTF-8"?>
<message>
    <header>
        <date>01/05/2010 20:45</date>
    </header>
    <body>
        <employes>
            <employed>
                <id>1</id>
                <name>Maria</name>
                <yearold>45</yearold>
            </employed>
```

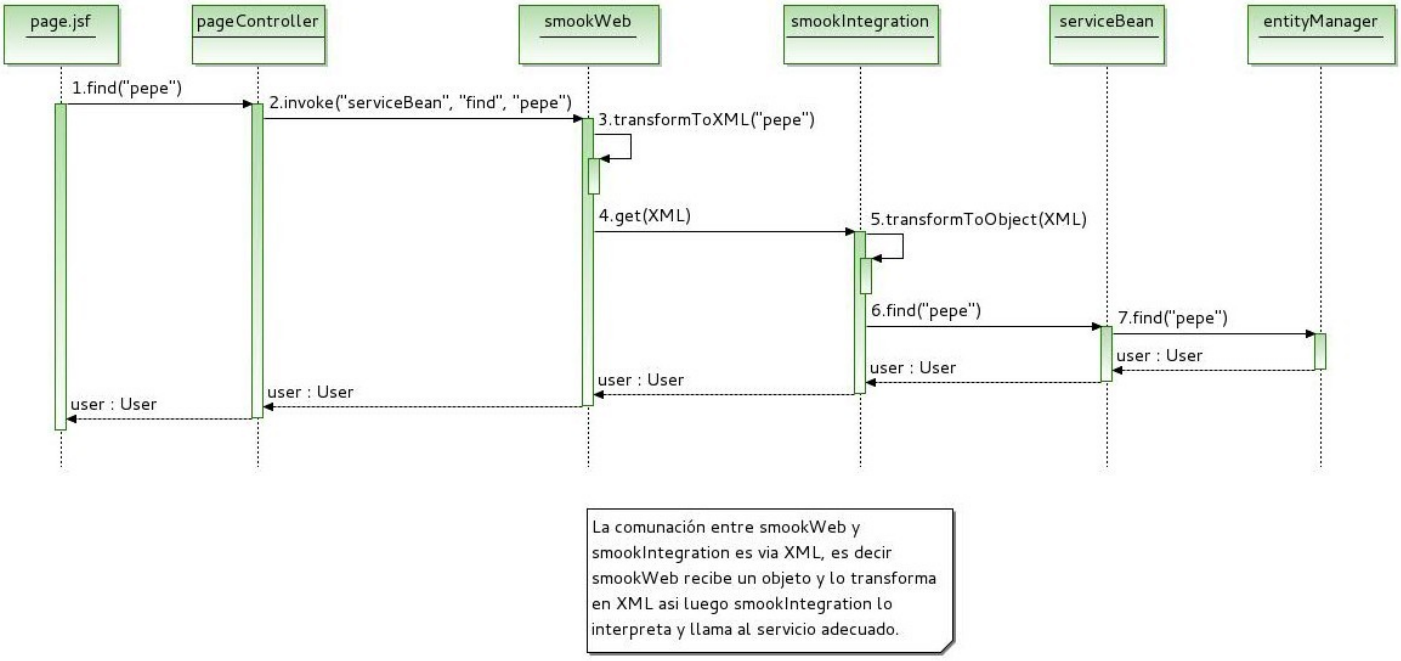
```java
public class Employed {

    private Integer id;
    private String name;
    private Integer yearold;

    public Integer getId() {
        return id;
    }
```

### *Ejemplo de invocacion de un servicio*

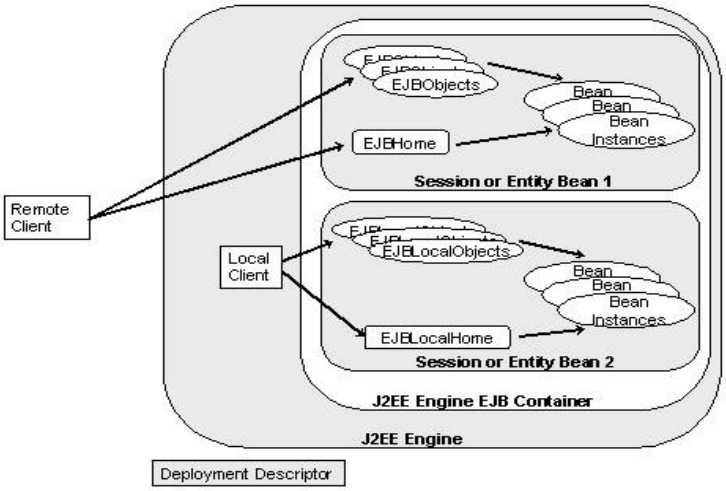
En este ejemplo se explicara la forma en la que un cliente consume los servicios expuestos por la aplicacion.




1. **Page.jsf:** Es la pagina de donde se genera la consulta de busqueda.
2. **PageController:** recibe esta peticion de la pagina y llama al smookWeb quien contiene la logica de conversion a XML de esa consulta.
3. **SmookWeb:** una ves convertido la consulta a XML se encarga de hacer la llamada al webservice publicado en una URL determinada.
4. **SmookIntegration:** Es el encargado de recibir los XML y convertirlos a objetos java para luego llamar al servicio explicito en la solicitud.
5. **ServiceBean:** Es el servicio ejb (logica de negocio)
6. **EntityManager:** es el responsable de controlar los accesos a la base de datos.

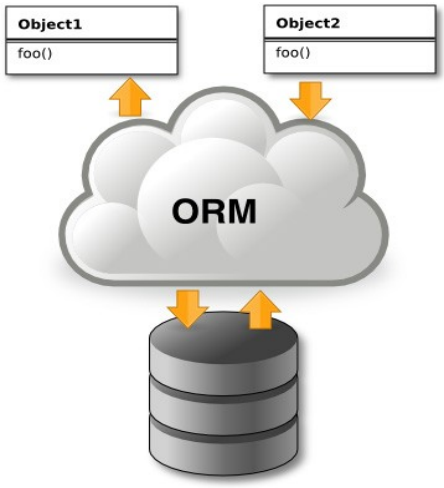
## Lógica de Negocio

Esta capa es la responsable de la lógica de nuestro negocio utilizando las entidades de la capa de modelo para esto. Es decir muchas veces va a funcionar de "pasa manos" de información, pero su objetivo principal es la de procesar los datos provenientes de las capas adyacentes para su correcta distribución, no se permite procesar estos datos ni en la base de datos a través de un procedimiento o en la capa de presentación en un controlador de pagina por ejemplo. Para ello también vamos a utilizar Enterprise Java Beans (EJB), que como se describió anteriormente, dado que encapsulan de forma transparente gran cantidad de lógica de aplicación.




## Persistencia

Las bases de datos relacionales almacenan la información mediante tablas, filas, y columnas, de manera que para almacenar un objeto hay que realizar una correlación entre el sistema orientado a objetos de Java y el sistema relacional de nuestra base de datos. JPA (Java Persistence API - API de Persistencia en Java) es una abstracción sobre JDBC que nos permite realizar dicha correlación de forma sencilla, realizando por nosotros toda la conversión entre nuestros objetos y las tablas de una base de datos. Esta conversión se llama ORM (Object Relational Mapping - Mapeo Relacional de Objetos), y puede configurarse a través de metadatos (mediante xml o anotaciones). Por supuesto, JPA también nos permite seguir el sentido inverso, creando objetos a partir de las tablas de una base de datos, y también de forma transparente.



# Servidor de Mail

Se asume que ya existe un dominio registrado en nic.ar (buddytraining.com.ar) para la configuración de la salida de correo electrónico utilizado para las notificaciones a los usuarios, los datos suministrados por el organismo de control son los siguientes:

**Servidores DNS:**
**DNS Primario:** *Nombre:*ns1.cdmon.net
*Dirección IP:*
**DNS Secundario:** *Nombre:*ns2.cdmon.net
*Dirección IP:*
**Datos opcionales:**
**Servidor alternativo:** *Nombre:*ns3.cdmon.net
*Dirección IP:*
**Servidor alternativo:**
**Servidor alternativo:**

Por otra parte se asume que se cuenta con un servidor para hospedar el servicio de correo electrónico con las siguientes características:

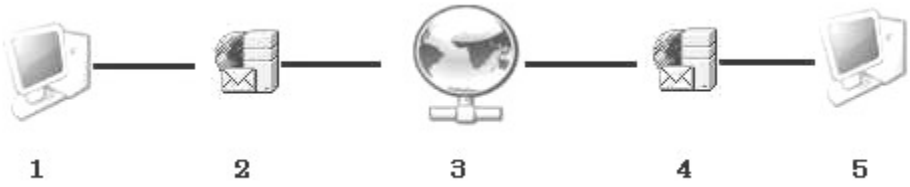
PRIMERGY MX130 S2

| | |
|---|---|
| **Chipset** | AMD 880G |
| **Mainboard type** | D3090 |
| **Product Type** | Mono Socket Micro Server |
| **Processor** | AMD Phenom™ II |
| **Memory** | 2 GB - 16 GB, DIMM (DDR3) ECC |
| **Memory protection** | ECC |
| **Storage drives** | HDD SATA, 3 Gb/s, 1000 GB, 7200 rpm |
| **I/O controller onboard** | SATA II |
| **OS** | CentOS 6.0 |

## Introduccion

Cuando enviamos un correo, el mensaje se enruta de servidor a servidor hasta llegar al MTA (Mail Transport Agent) del destinatario. La tarea de los Servers MTA es justamente comunicarse entre sí y llevar el “paquete” de un servidor a otro, ésta comunicación la realizan utilizando servidores SMTP (Simple Mail Transfer Protocolo). Luego el MTA del destinatario entrega el correo electrónico al servidor del correo entrante, llamado MDA (Mail Delivery Agent), el cual retiene el correo electrónico mientras espera que el usuario lo acepte. Existen dos protocolos principales que se utilizan para la recuperación de un correo electrónica en el MDA:

- POP3 (Post Office Protocol) es el más antiguo de los dos y se utiliza para recuperar el correo electrónico y, en algunos casos, dejar una copia en el servidor.
- IMAP (Internet Message Access Protocol Se usa para coordinar el estado de los correos electrónicos (leído, eliminado, movido) a través de múltiples clientes de correo electrónico. Con IMAP, se guarda una copia de cada mensaje en el servidor, de manera que esta tarea de sincronización se pueda completar.




1) Usuario de “Gmail” escribe un correo para usuario de “Yahoo”
2) El correo llega al MTA de Gmail, éste lo hace circular por internet.
3) Internet
4) Llega el correo hasta el MTA de Yahoo!, éste lo almacena en el MDA de Yahoo! hasta que el usuario se digne a abrir el correo, borrarlo o lo que fuere.
5) El usuario de Yahoo! lee su correo desde el MDA de Yahoo!

Yendo a una analogía, los MTA serían los “carteros” dentro del correo, los MDA vendrían a ser los bolsos del cartero donde están “las cartas”,y el MUA somos nosotros obteniendo esa cada carta de ese MDA que viene a nuestro nombre.

## Postfix

Es un Agente de Transporte de Correo (MTA) de software libre / código abierto, un programa informático para el enrutamiento y envío de correo electrónico, creado con la intención de que sea una alternativa más rápida, fácil de administrar y segura.

# Estructura del Proyecto

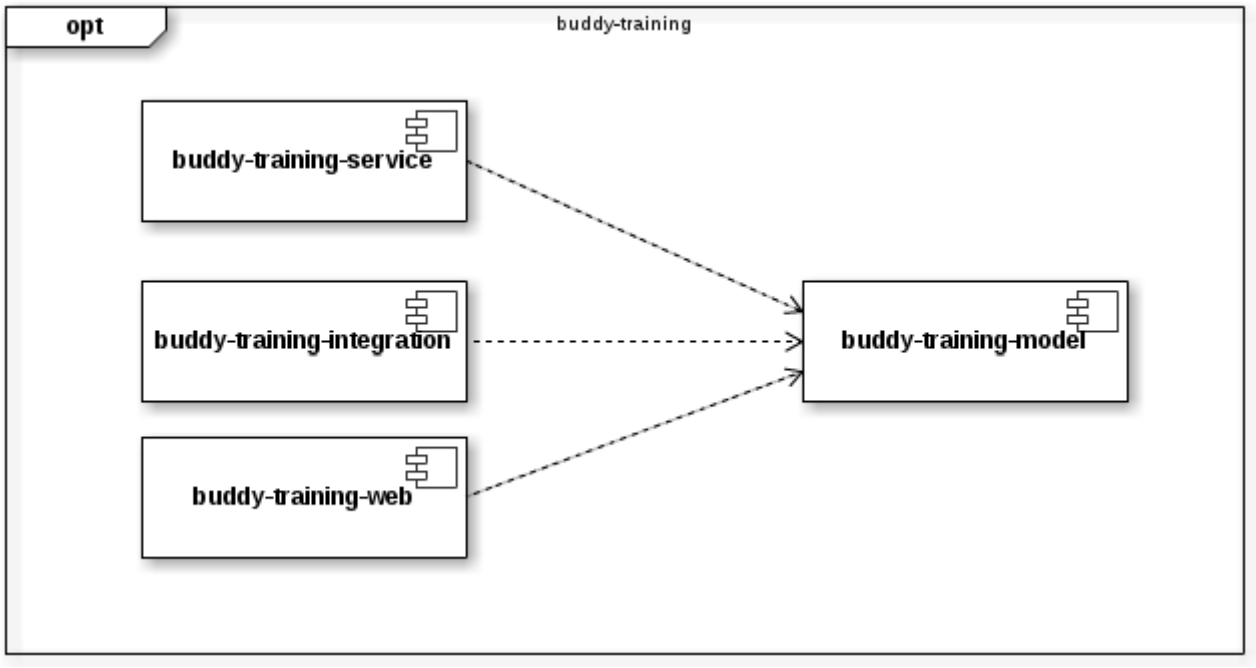



Figura: Diagrama de Componentes que muestra cada uno de los artefactos/proyectos que componen la aplicación de BuddyTraining. El nombre de la aplicación es “buddy-training”. A su vez, la aplicación cuenta con cuatro proyectos, uno por cada capa: “buddy-training-model”, “buddy-training-service”, “buddy-training-integration” y “buddy-training-web”.

La aplicación de BuddyTraining está construida con Apache Maven 2. Se trata de un multi-proyecto Maven con cuatro módulos, donde cada módulo representa cada una de las capas de la arquitectura JavaEE 5 explicada en la sección anterior.

## buddy-training-model

Es el modelo de dominio con las entidades JPA.

## buddy-training-service

Son los EJB 3.0 que implementan la lógica de negocio.

## buddy-training-integration

Encagado de recibir las peticiones de los clientes y transformar dichas solicitudes en XML / Objetos java.

## buddy-training-web

Se trata de la aplicación web construida con ADF Faces, ADF TaskFlows y JSF 2.0.

## La Ventaja de Utilizar Maven

Maven es una herramienta open source de Apache que sirve para administrar proyectos de desarrollo Java. Es una herramienta que está basada en plugins, lo que implica que sus funcionales siempre pueden ser extendidas por la comunidad.

Con Maven, los desarrolladores de BuddyTraining van a poder:

- Resolver de forma automática las dependencias de librerías (propias y de terceros).
- Compilar el código fuente.
- Ejecutar los Unit Test de forma masiva.
- Generar los artefactos y empaquetarlos.
- Desplegar los artefactos en el servidor de aplicaciones.

Maven ofrece muchas otras funcionalidades, gracias a su arquitectura orientada a plugins, pero éstas son las que los desarrolladores de BuddyTraining más van a utilizar.

Las ventajas más importantes de utilizar Maven como administrador del código fuente son:

- **Independencia de la plataforma:** el proyecto se puede compilar y empaquetar con los mismos comandos en cualquier sistema operativo.
- **Independencia del IDE**: Maven ofrece fácil integración con todos los IDE de desarrollo en Java; si bien utilizaremos Eclipse como IDE principal, fácilmente podríamos pasar a Jdeveloper, NetBeans o cualquier otro.
- **Repositorio de dependencias:** Manejar las dependencias de un proyecto JavaEE a veces puede tornarse muy complicado, por la cantidad y por lo delicado que puede resultar juntar algunas librerías con otras. En lugar de usar el SVN para subir todas las dependencias del proyecto, Maven utiliza repositorios públicos y repositorios locales para conseguir las librerías en el momento en que lo necesite, resolviéndonos además el problema de las dependencias transitivas (las dependencias que trae una dependencia de terceros que estamos usando).
- **Declaración de las dependencias:** cada proyecto cuenta con un descriptor XML llamado POM (Project Object Model) donde se listan cada una de las dependencias del proyecto; de esta forma, las librerías que los desarrolladores utilizan quedan documentadas de una forma estándar, incluso con sus respectivas versiones.
- **Buena integración con JUnit:** la opción de poder ejecutar todos los Unit Test con Maven, de forma masiva, es también otra de las grandes ventajas.

# Nomenclatura de Paquetes

- Todos los paquetes de la aplicación comenzarán con la ruta:
  **com.solution.buddytraining**
- Todos los paquetes del proyecto buddy-training-model comenzarán con la ruta:
  **com.solution.buddytrainin.model**
- Todos los paquetes del proyecto buddy-training-service comenzarán con la ruta:
  **com.solution.buddytraining.service**
- Todos los paquetes del proyecto buddy-training-web comenzarán con la ruta:
  **com.solution.buddytraining.web**
- Todos los Controller del proyecto buddy-training-web comenzarán con la ruta:
  **com.solution.buddytraining.web.controller**

El proyecto buddy-training-model usará objetos MockBuilder para fabricar objetos de modelo dummy para ser usado en pantallas mock y en los UnitTest de todas las capas. Si existe la entidad Sportsman, por ejemplo, habrá un SportsmanMockBuilder que sirva para devolver listados de Sportsman dummies y distintos Sportsman generadas en memoria.

- Todos los MockBuilders del proyecto buddy-training-model comenzarán con la ruta:
  **com.solution.buddytraining.model.mockbuilder**

# Integración Continua

La integración continua es una metodología propuesta inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes.

En este contexto, se entiende por integración a la compilación y ejecución de tests de todo un proyecto.

Se espera que el proyecto BuddyTraining nos sirva para utilizar esta metodología. Para esto nos apoyaremos en cuatro herramientas fundamentales:

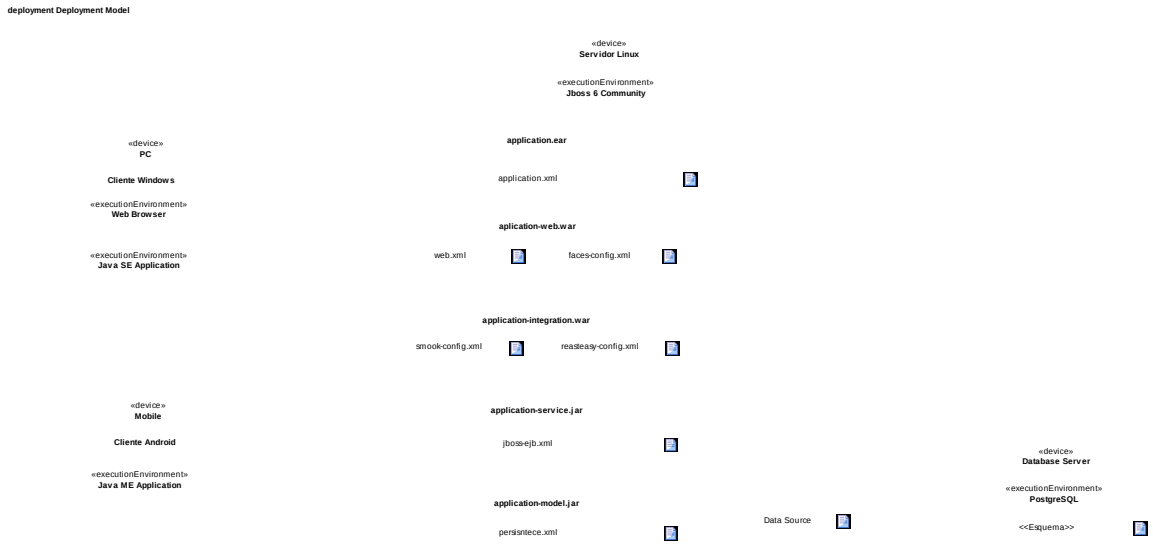
- **JUnit**: para realizar pruebas unitarias; se realizarán test unitarios para cada funcionalidad en cada capa, intentando cubrir la mayor cantidad de código funcional posible.
- **Maven**: como herramienta de build, para compilar el proyecto y ejecutar los tests.
- **SVN**: para versionar el código fuente en un único repositorio centralizado.
- **Hudson**: como servidor de integración continua para que utilice todas estas herramientas y

automatice el proceso.

Todos los días Hudson ejecutará estos pasos:

1. Actualizará el código fuente de la aplicación descargándolo del SVN.
2. Compilará los proyectos utilizando Maven.
3. Ejecutará todos los Unit Test escritos con JUnit a través de Maven.
4. Notificará mediante mails al equipo si hubo un error en alguno de los pasos anteriores.

# Vista de Despliegue




El diagrama representa tres clientes, 1 con su respectivo Web Browser y los otros 2 con las aplicaciones java de sincronización de datos.

En el primer caso el Web Browsers se comunican por HTTP con la aplicación web dentro del EAR desplegado en el servidor de aplicaciones WebLogic. El jar de modelo y persistencia es quien se comunica con los orígenes de datos configurados en el servidor que hacen referencia a los esquemas de la base de datos PostgreSQL. Dentro de los componentes de la aplicación se pueden ver algunos de los archivos de configuración más importantes, como el persistence.xml o el application.xml.

En el segundo caso, las aplicaciones java para la sincronización de datos con el servidor, utilizando los servicios expuestos (WebService) para el envió y recepción de datos, es decir mensajes compuestos por datos XML entre un consumidor (la aplicación que usa los datos) y un proveedor (el servidor que contiene los datos).

El artefacto que contiene toda la aplicación es un EAR (buddy-training.ear) que se despliega en el servidor de aplicaciones WebLogic como Aplicación Enterprise.

Dentro, el EAR contiene 3 módulos principales:

- buddy-training-service.jar: módulo EJB donde se encuentra la lógica de negocio.
- buddy-training-integration.war: webservice restfull.
- buddy-training-web.war: aplicación web donde se encuentra la aplicación JSF, las vistas

(páginas web) y sus controladores asociados (clases java Controller).

Como librería, el buddy-training.ear contiene el artefacto de modelo (buddy-training-model.jar) con las entidades JPA y sus mapeos objeto-relacional asociados.

La configuración de la unidad de persistencia se encuentra en el archivo de configuración XML llamado **persistence.xml**, dentro de buddy-training-model.jar. La unidad de persistencia hace referencia a los Data Source para cada esquema de la base de datos PostgreSQL.

Los descriptores o archivos de configuración estándares de JavaEE 5 son:

- **persistence.xml**: recién mencionado.
- **web.xml**: descriptor de la aplicación web, ubicado dentro de buddy-training-web.war; contiene las configuraciones básicas de una aplicación web Java como declaración de Servlets, Listeners, parámetros para los Servlets, referencias a EJBs (opcional), etc.
- **faces-config.xml**: descriptor de JSF, ubicado dentro de buddy-training-web.war, donde se declaran los Managed Beans (Controllers) de JSF estándares y la navegación básica provista por JSF; si se quiere usar un componente ADF se debe declarar en el descriptor no estándar adfc-config.xml.
- **application.xml**: descriptor del EAR, ubicado dentro de buddy-training.ear; aquí se declaran cada uno de los módulos de la aplicación empresarial; BuddyTraining tiene tres módulos: el módulo de EJBs: buddy-trainings-service.jar, el mòdulo de Integracion: buddy-training-integration.war y el módulo web: buddy-training-web.war.

Los descriptores o archivos de configuración propietarios de Oracle WebLogic son:

- **weblogic-ejb-jar.xml:** descriptor del módulo de EJBs, ubicado dentro de buddy-training-service.jar; contiene configuración específica de Weblogic para los EJB como por ejemplo optimizaciones en la lectura de las referencias.
- **weblogic.xml:** descriptor del módulo web, ubicado dentro de buddy-training-web.war; contiene configuración específica de Weblogic para la aplicación web.
- **adfc-config.xml:** descriptor de ADF, ubicado dentro de buddy-training-web.war; aquí se declaran los Managed Beans de ADF (Controllers) y la navegación de ADF Task Flows.
- **trinidad-config.xml:** descriptor de Trinidad, parte del framework de componentes JSF de Oracle ADF.
- **weblogic-application.xml:** descriptor específico de Weblogic para el EAR con configuración de bibliotecas compartidas y optimizaciones para el servidor de aplicaciones.

Como cualquier aplicación web puede ser accedida a través de un navegador web desde cualquier estación de trabajo de cliente que tenga acceso al servidor de aplicaciones. La comunicación, por supuesto, es a través del uso del protocolo HTTP.

Por el lado de la base de datos se usarán 1 solo esquemas que se creará para las tablas específicas de

BuddyTraining llamado BUDDYTRAINING.

# Selecciòn de Tecnología

## Tecnologìas Seleccionadas

| Tecnología | Tipo | ¿Por qué fue elegida? |
|---|---|---|
| JPA | Biblioteca de Desarrollo | Por ser estándar y porque permite implementar un modelo de dominio orientado a objetos. |
| EJB 3.0 | Biblioteca de Desarrollo | Por ser estándar y porque permite utilizar las implementaciones de requerimientos no funcionales del servidor de aplicaciones, como por ejemplo un manejo declarativo y transparente de transacciones de negocio. Además permite exponer la lógica de negocio como servicios. |
| JSF | Biblioteca de Desarrollo | Por ser estándar y porque provee un framework web MVC orientado a componentes y de rápido desarrollo. |
| ADF Faces | Biblioteca de Desarrollo | Porque provee componentes JSF con mayor funcionalidad y mejor aspecto visual. |
| ADF Task Flows | Biblioteca de Desarrollo | Porque provee un sistema declarativo de navegación entre páginas más poderoso que el estándar de JSF. |
| JUnit 4 | Biblioteca de Desarrollo | Porque permite realizar pruebas unitarias de forma rápida y sencilla. |
| Maven 2 | Herramienta | Porque permite compilar, ejecutar pruebas unitarias de forma masiva (con buena integración con JUnit), empaquetar, desplegar, ejecutar pruebas de integración, administrar las bibliotecas de desarrollo y mucho más. Además se eligió porque es independiente de la plataforma y del IDE de desarrollo. |
| Hudson | Servidor | Porque provee un servidor de integración continua sencillo de usar y fácil de configurar. |
| Eclipse | Entorno de Desarrollo | Porque provee buena integración con los productos de Oracle que más utilizamos: ADF Faces, ADF Task Flows y Weblogic. |

# Tecnologías Descartadas

| Tecnología | Tipo | ¿Por qué no fue elegida? |
|---|---|---|
| Oracle ADF Business Components | Biblioteca de Desarrollo | Por ser un framework propietario de Oracle, orientado a tablas y no a objetos y por ser muy compleja la configuración de sus descriptores XML. |