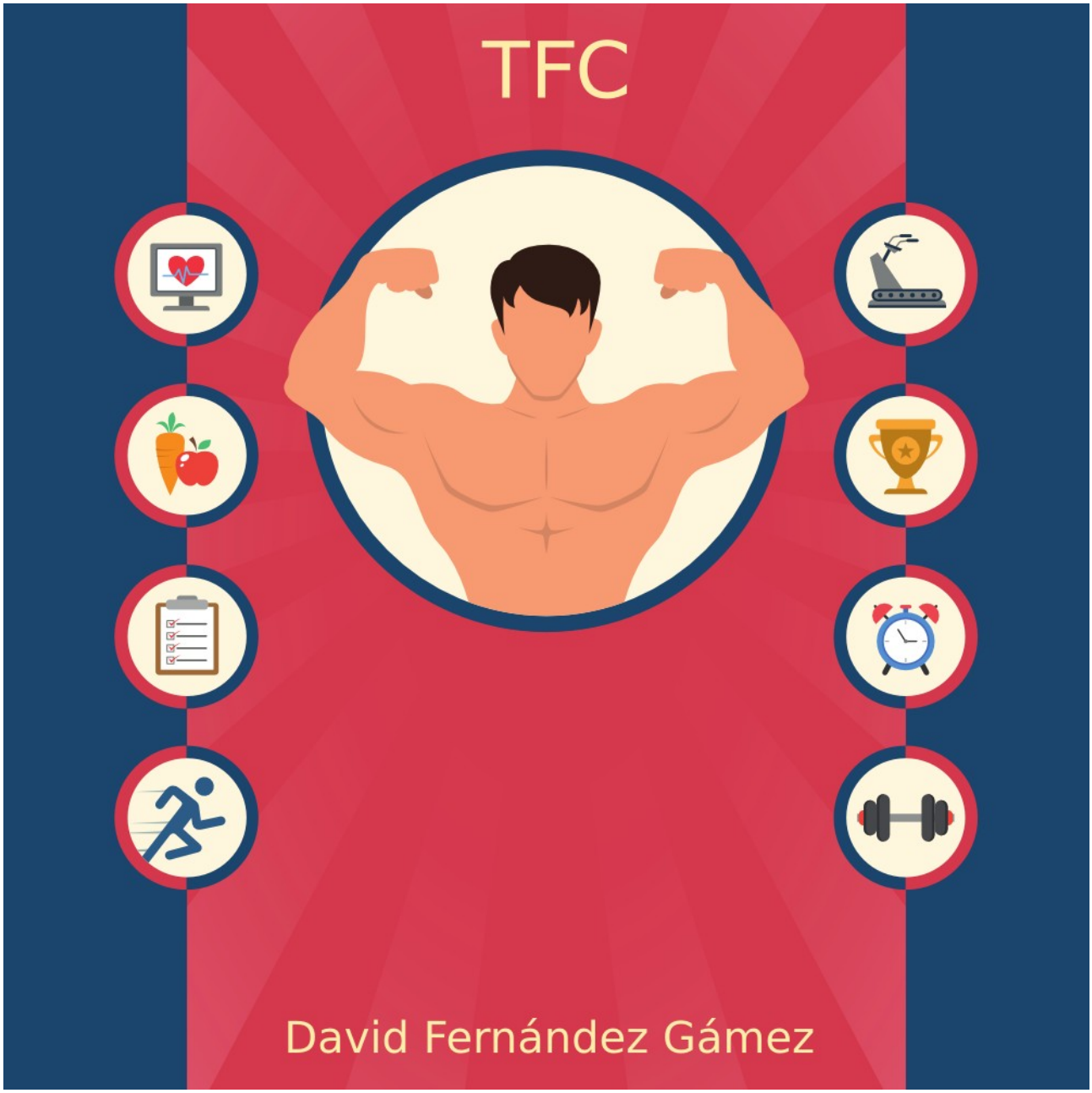

TFC
David Fernández Gámez

Índice de contenido

INTRODUCCIÓN

En la actualidad existen múltiples aplicaciones y herramientas, ya sea para PC o para los distintos dispositivos móviles, relacionados con el mundo del fitness o wellness. La mayoría de estas herramientas están enfocadas para su uso por atletas a modo de diario de entrenamiento o dieta. Lo interesante de este proyecto es que la herramienta no está pensada, al menos en un primer momento, para su uso directo por atletas si no que está enfocada a ayudar a los entrenadores personales a llevar un control sobre los atletas que entrenan tanto a nivel de dieta como a nivel de entrenamiento.

La idea del presente TFC es la realización de una aplicación para la gestión del entrenamiento físico y la supervisión nutricional de deportistas. El objetivo principal del proyecto es conseguir realizar una aplicación que ayude al seguimiento y progresión de los deportistas en términos de entrenamiento físico y supervisión nutricional. El punto de partida son varios excels que realizan funciones distintas (uno es para la generación de dietas y otro para la generación/seguimiento de rutinas).La idea es “compilar” la funcionalidad de ambos excels en una sola aplicación añadiéndole nuevas funcionalidades y mejorando las actuales.

DEFINICIÓN DEL PROYECTO

La aplicación para la gestión del entrenamiento físico y la supervisión nutricional de deportistas servirá para que lo entrenadores personales puedan hacer un seguimiento de sus atletas. Para ello la aplicación permitirá, tener un registro de las medidas corporales y biométricas de los deportistas, el progreso en la fuerza (pesos movidos en una sesión), dieta seguida, ejercicios realizados e incluso foto. Los entrenadores personales tendrán a su disposición una base de datos de alimentos y de ejercicios que les ayudará a buscar en ellos de manera ágil y en base a determinados parámetros.

La aplicación tendrá dos tipos de usuarios, por una parte los entrenadores personales, que a la postre serán los administradores de la misma y por otra los clientes de esos entrenadores personales.

Los entrenadores personales serán los encargados de dar de alta clientes, confeccionar plantes de entrenamiento y dietas y hacer el seguimiento de las

evoluciones de los clientes. Por su parte los clientes podrán acceder a sus fichas de cliente, crear entradas en sus bitácoras diarias (ya sea de entrenamiento o nutrición), subir fotos y comunicarse con sus entrenadores personales mediante un pequeño foro.

HISTORIAS DE USUARIO

Jordi es un preparador físico, su labor como preparador físico es el desarrollo, seguimiento y evolución tanto de rutinas de ejercicios musculares como de dietas personalizadas.

Fran es un chico que va asiduamente al gimnasio desde hace años y que ahora se ha planteado un objetivo claro: bajar su porcentaje graso al 10%.

Fran se pone en manos de Jordi y le cuenta su objetivo.

Jordi registra una "ficha de cliente" en el sistema con datos personales de Fran.

En una primera entrevista Jordi hace un pequeño cuestionario a Fran donde le pregunta sus hábitos alimenticios, la dieta que está llevando a cabo ahora, la rutina de ejercicios que sigue, le toma una serie de medidas corporales (altura, peso, cintura, brazo, pliegues corporales,...) y registra cuantas veces por semana entrenará Fran.

Jordi registra los datos obtenidos en la entrevista y los vincula a la ficha de cliente que se había creado previamente.

Jordi se pone a confeccionar una dieta y un entrenamiento según el objetivo que Fran le había marcado partiendo de los datos de la entrevista.

El sistema proveerá a Jordi de información detalla de alimentos y ejercicios para que a Jordi le sea fácil la confección tanto de la dieta como del programa de entrenamiento.

Una vez que Jordi tenga claro cuál es la distribución de alimentos (a nivel de macronutrientes) y el aporte calórico total diario que es necesario para la dieta de Fran, Jordi elegirá los alimentos.

El sistema proveerá, así mismo, de información detallada de ejercicios físicos, así Jordi podrá distribuir los ejercicios, el número de estos y la frecuencia semanal (de acuerdo a lo que le dijo Fran) en una programa de ejercicios.

Una vez que Jordi tenga finalizado tanto la dieta como el programa de ejercicios, Jordi enviará un email a Fran con un usuario/password de aplicación para que Fran pueda acceder tanto a su ficha como a su información.

Fran podrá insertar registros diarios de su evolución en los pesos de los ejercicios que le marcó Jordi.

Fran podrá subir fotos de su evolución (diario, semanal, etc...)Fran tendrá un espacio de discusión personal con Jordi donde poder intercambiar dudas (sobre los ejercicios, dieta, etc...). Sería una especie de foro privado.

Jordi y Fran acordarán una fecha de revisión, donde Jordi volverá a tomar las medidas oportunas y comentará con Fran posibles cambios.

Estas entrevistas se repetirán las veces que sea necesario hasta cumplir los objetivos de Fran.

PROPUESTA TECNOLÓGICA

El proyecto estará desarrollado íntegramente con tecnología Java Enterprise Edition, se hará uso de frameworks open source y estándares ya sean implícitos del lenguaje: como puede ser JSF para la capa de la vista, como estándares “de facto” como puede ser “hibernate” que si bien no es el estándar JPA sí es el más usado por la industria. Así pues la arquitectura tecnológica será la siguiente:

BASE DE DATOS

PostgreSQL. La decisión de decantarse por una base de datos relacional es sencilla: la aplicación necesita, entre otras cosas, de integridad referencial y estructuras de datos concretas. Sin lugar a dudas PostgreSQL es la base de datos relacional open source más potente que existe, si bien quizás no sea un proyecto en el que se explotará todas las bondades de dicha base de datos (como por ejemplo permitir cluster activo-activo), creo que es un buen punto de partida. Otra alternativa totalmente válida para el proyecto y que no implicaría ninguna diferencia sería usar una base de datos MySQL.

SERVIDOR DE APLICACIONES

Si bien Apache – Tomcat no es un servidor de aplicaciones JEE (sería su hermano mayor Apache TomEE), creo que es suficiente para desplegar la

aplicación que desarrollaremos ya que es compatible con varias especificaciones Java EE como Java Servlet, JavaServer Pages (JSP) y Java Expression Language que necesitaremos en nuestra aplicación.

CAPA DE ACCESO A DATOS – PERSISTENCIA

En este caso la elección es un framework ORM (Object-Relational mapping), concretamente se usará Hibernate iiien su última versión (5.0.1). Se hará uso de un ORM en vez del acceso JDBC tradicional porque:

- Así la aplicación es independiente de la base de datos, es el ORM quien se encargará de "traducir" las sentencias SQL a la base de datos que corresponda (en este caso postgreSQL).
- El código es mucho más escalable y mantenible.
- La gestión de la transaccionalidad la realizará el framework y no nosotros, siendo mucho más seguro y eficiente. Hibernate es una implementación del estándar JPA iv(Java Persistence API) y el framework ORM más ampliamente usado dentro del desarrollo JEE.

CAPA DE VISTA O FRONT-END

Siguiendo con el uso de frameworks open source ampliamente utilizados y basados en estándares EE, en este caso se usará JSF, concretamente la implementación Primefaces ven su versión más reciente (5.2). Primefaces es una librería de componentes JSF con un enorme conjunto de componentes enriquecidos que facilitan mucho el desarrollo ágil en la capa de vista.

OTROS FRAMEWORKS O TECNOLOGÍAS USADAS

Un problema que nos podríamos encontrar a la hora de desarrollar el proyecto (o a la hora de escalarlo o ampliar sus funcionalidades), es la de tener que trabajar con diferentes frameworks. Cada framework gestiona sus objetos y más importante aún, su ciclo de vida. Una manera que tenemos de enfrentarnos a esto es hacer uso de un framework que nos encapsule y aisle del problema. En este caso existe un framework open source llamado Spring vique nos ayudará en este sentido. Spring nos ayudará a solventar este problema cambiando las responsabilidades y en vez de nosotros los encargados de generar los objetos de cada uno de los frameworks será Spring (haciendo uso de su contenedor de inversión de control (IoC), que proporciona una forma consistente de configuración y administración de objetos Java usando la Reflexión).

PLAN DE PROYECTO

Una vez decidida la arquitectura tecnológica es el momento de realizar un plan de proyecto para ver las distintas fases en las que se dividirá el mismo, a continuación se muestra una estimación de trabajo donde se incluyen las diferentes fases divididas a su vez en tareas. Las tareas tienen una fecha estimada de inicio y una fecha estimada de finalización, siendo en todo momento orientativas.

ID	Tarea	Fecha inicio	Fecha Fin	Días
1	Plan de trabajo (PAC 1)	16/09/2015	30/09/2015	14
1.2	Decidir tema del TFC	16/09/2015	23/10/2015	37
1.3	Estudio de tecnologías existentes	23/09/2015	27/09/2015	4
1.4	Decidir arquitectura del proyecto	23/09/2015	27/09/2015	4
1.5	Preparación del plan de proyecto	27/09/2015	30/09/2015	3
1.6	Entrega del plan de trabajo	30/09/2015	30/09/2015	0
2	Fase de análisis y diseño (PAC 2)	01/10/2015	04/11/2015	33
2.1	Diseño arquitectura de aplicación	01/10/2015	07/10/2015	6
2.2	Análisis requerimientos funcionales	08/10/2015	18/10/2015	10
2.3	Análisis requerimientos no funcionales	19/10/2015	03/11/2015	14
2.4	Entrega documento de analisis y diseño	04/10/2015	04/11/2015	30
3	Fase de desarrollo (PAC 3)	05/11/2015	18/12/2015	43
3.1	Desarrollo modelo de datos	05/11/2015	07/11/2015	2
3.2	Desarrollo capa de persistencia	08/11/2015	10/11/2015	2
3.3	Desarrollo capa de negocio	11/11/2015	26/11/2015	15
3.4	Desarrollo capa de la vista	27/11/2015	30/11/2015	3
3.5	Desarrollo de test unitarios	01/12/2015	17/12/2015	16
3.6	Entrega documento de desarrollo	18/12/2015	18/12/2015	0
4	Memoria y presentación del proyecto	19/12/2015	11/01/2016	22
4.1	Desarrollo docuento de memoria de proyecto	19/12/2015	01/01/2016	12
4.2	Entrega proyecto	02/01/2016	10/01/2016	8
4.3	Defensa del proyecto	11/01/2016	11/01/2016	0

• ANÁLISIS DEL PROYECTO

El proyecto será una aplicación para la gestión del entrenamiento físico y la supervisión nutricional de deportistas. El objetivo principal del proyecto es conseguir realizar una aplicación que ayude al seguimiento y progresión de los deportistas en términos de entrenamiento físico y supervisión nutricional. El punto de partida son varios excels que realizan funciones distintas (uno es para la generación de dietas y otro para la generación/seguimiento de rutinas).La idea es “compilar” la funcionalidad de ambos excels en una sola aplicación añadiéndole nuevas funcionalidades y mejorando las actuales.

COMPONENTES

El proyecto se ha dividido en una serie de módulos funcionales que hará más fácil su mantenimiento y desarrollo posterior.

El módulo de gestión de clientes es el encargado de la administración de los deportistas que serán clientes de la aplicación. Mediante este módulo se podrá dar de alta, modificar, listar y dar de baja a los diferentes clientes.

El módulo de gestión de alimentos es el encargado de la administración de los alimentos que formarán parte del sistema. Estos alimentos servirán para confeccionar las dietas que llevarán los deportistas. Mediante este módulo se podrán dar de alta, modificar, listar y eliminar alimentos del sistema.

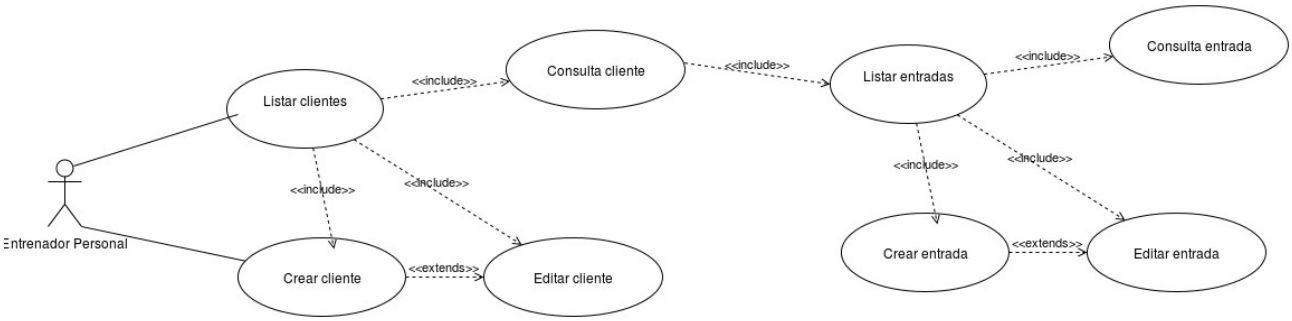
El módulo de gestión de ejercicios es el encargado de la administración de los distintos ejercicios físicos que formarán parte del sistema. Estos ejercicios servirán para confeccionar las rutinas que llevarán los deportistas. Mediante este módulo se podrán dar de alta, modificar, listar y eliminar ejercicios del sistema.
El módulo de gestión de seguimientos es el encargado de contener los seguimientos temporales que se irán haciendo a los deportistas. Mediante este módulo se podrá listar los seguimientos abiertos, añadir una entrada para un cliente, editar una entrada para un cliente o eliminar una entrada.

REQUERIMIENTOS FUNCIONALES

A continuación se enumeran los distintos módulos funcionales en los que se ha divido la aplicación. Para cada módulo funcional se ha especificado una serie de casos de uso, tanto a nivel textual como en formato UML.

MÓDULO DE GESTIÓN DE CLIENTES

Este módulo permite la gestión de clientes de la aplicación. Permite la gestión básica de los datos de un cliente,así como la gestión de las visitas que realice el cliente . A continuación se presenta el diagrama UML de los diferentes casos de uso.



A continuación una explicación textual de los casos de uso.

CASO DE USO: LISTAR CLIENTES	
Resumen de funcionalidad	Muestra un listado con los clientes actuales
Actores	Entrenador personal
Casos de uso relacionados	
Precondición	Ninguna al ser una consulta
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El Entrenador puede hacer click en el cliente que lo desee para ver su ficha completa(Caso de uso Editar cliente).
Flujo alternativo	4. El Entrenador puede hacer click en el botón de “Añadir cliente” (Caso de uso Alta de cliente)
Postcondición	Ninguna

CASO DE USO: ALTA DE CLIENTE	
Resumen de funcionalidad	Crea un nuevo cliente en el sistema
Actores	Entrenador personal

Casos de uso relacionados	Edición cliente
Precondición	El cliente no existe en el sistema
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El entrenador puede hacer click en el botón de “Añadir cliente” 4. El entrenador rellena los campos del nuevo cliente. 5. El sistema comprueba que todos los datos son correctos y que ningún cliente con el mismo dni existe. 6. El entrenador guarda los datos del cliente haciendo clic en el botón guardar y volvemos a la lista de clientes (caso de uso “Listar clientes”)
Flujo alternativo	5. El sistema detecta algún error y se muestra por pantalla al usuario 5 El entrenador cancela o cierra la ventana
Postcondición	Se crea un nuevo cliente en el sistema
CASO DE USO: EDICIÓN DE CLIENTE	
Resumen de funcionalidad	Se modifican los datos de un cliente
Actores	Entrenador personal
Casos de uso relacionados	Listar clientes; consulta de cliente
Precondición	Se está editando un cliente existente
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el cliente que lo desee para ver su ficha completa. 5. El entrenador modifica los datos que desee. 6. El sistema comprueba que todos los

	datos son correctos. 7. El entrenador guarda los datos del cliente haciendo clic en el botón guardar y volvemos a la lista de clientes
Flujo alternativo	5. El entrenador cancela la edición 6. El sistema detecta algún error y se lo muestra al usuario por pantalla.
Postcondición	El cliente es actualizado en el sistema

CASO DE USO: CONSULTA DE CLIENTE	
Resumen de funcionalidad	Se muestran los datos de un cliente
Actores	Entrenador personal
Casos de uso relacionados	Listar clientes
Precondición	Se está consultando un cliente existente
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el cliente que lo desee para ver su ficha completa. 5. El sistema devuelve los datos del cliente seleccionado en una nueva ventana 6. El entrenador cierra la ventana o pulsa el botón cancelar.
Flujo alternativo	
Postcondición	Ninguna

CASO DE USO: LISTAR ENTRADAS	
Resumen de funcionalidad	Muestra un listado con las entradas

	actuales del cliente seleccionado
Actores	Entrenador personal
Casos de uso relacionados	
Precondición	Ninguna al ser una consulta
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El Entrenador hace click en el cliente que lo desee para ver su ficha completa(Caso de uso Editar cliente). 5. El entrenador hace click en el apartado “Ver entradas”
Flujo alternativo	4. El Entrenador puede hacer click en el botón de “Añadir cliente” (Caso de uso Alta de cliente)
Postcondición	Ninguna

CASO DE USO: AÑADIR ENTRADA	
Resumen de funcionalidad	Añadir una nueva entrada a un cliente
Actores	Entrenador
Casos de uso relacionados	Listar entradas; editar cliente, consultar cliente
Precondición	El cliente existe previamente en el sistema
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el cliente que lo desee para ver su ficha completa.

	5. El entrenador puede hacer click en el botón añadir entrada. 6. El entrenador introduce los datos de la nueva entrada. 7. El sistema comprueba si todos los datos son correctos 8. La nueva entrada se añade al cliente
Flujo alternativo	7. Si el sistema detecta algún error lo muestra por pantalla al usuario
Postcondición	Una nueva entrada se ha creado en el sistema

CASO DE USO: CONSULTAR ENTRADA	
Resumen de funcionalidad	Poder consultar una entrada histórica de un cliente
Actores	Entrenador personal
Casos de uso relacionados	Listar entradas, editar cliente
Precondición	El cliente existente en el sistema
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el cliente que lo desee para ver su ficha completa. 5. El entrenador puede hacer click en la entrada que lo desee. 6. El sistema devuelve una pantalla con los datos de esa entrada
Flujo alternativo	5. El entrenador puede cancelar o cerrar la ventana del cliente
Postcondición	Ninguna

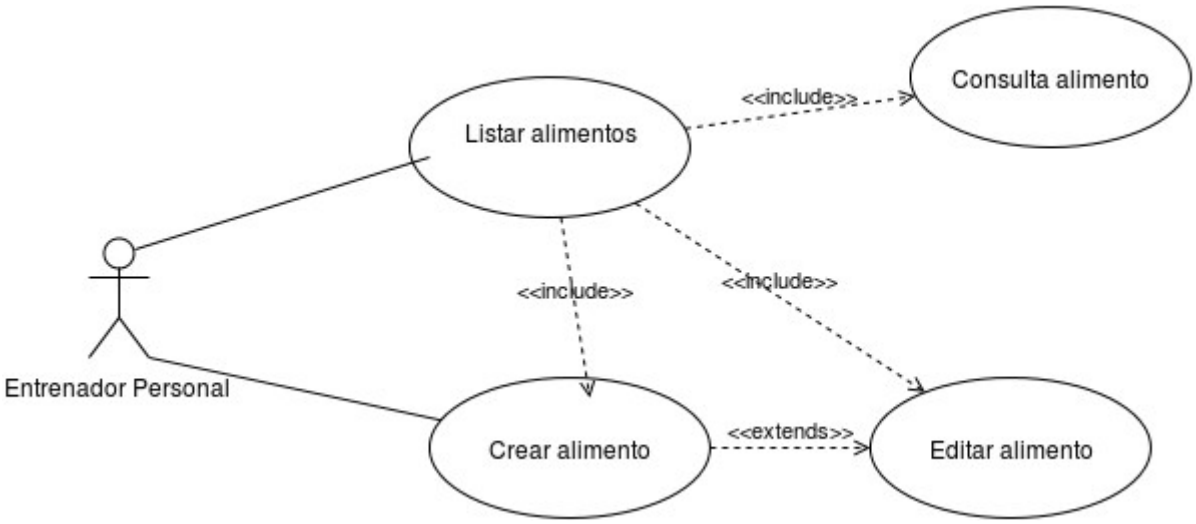
CASO DE USO: EDITAR ENTRADA	
Resumen de funcionalidad	Poder editar una entrada histórica de un cliente
Actores	Entrenador personal
Casos de uso relacionados	Listar entradas, consultar cliente
Precondición	El cliente existente en el sistema
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el cliente que lo desee para ver su ficha completa. 5. El entrenador puede hacer click en la entrada que lo desee. 6. El sistema devuelve una pantalla con los datos de esa entrada 7. El entrenador modifica los datos de esa entrada 8. El sistema comprueba que no hay errores 9. El entrenador pulsa guardar y los datos se guardan en el sistema
Flujo alternativo	5. El entrenador puede cancelar o cerrar la ventana del cliente 7. El entrenador cierra la ventana de la entrada
Postcondición	La entrada se actualiza correctamente

MÓDULO DE GESTIÓN DE ALIMENTOS

Este módulo permite la gestión de clientes de la aplicación. Permite la gestión

básica de los datos de un cliente . A continuación se presenta el diagrama UML de los diferentes casos de uso.

A



continuación una explicación textual de los casos de uso.

CASO DE USO: LISTAR ALIMENTOS	
Resumen de funcionalidad	Muestra un listado con los alimentos actuales
Actores	Entrenador personal
Casos de uso relacionados	
Precondición	Ninguna al ser una consulta
Flujo normal	1. El entrenador personal entra en el modulo alimentos. 2. Aparece una lista con todos los alimentos ordenada alfabéticamente 3. El listado permite filtrar los resultados por varios campos. 4. El Entrenador puede hacer click en el alimento que lo desee para ver su ficha completa(Caso de uso Editar alimento).
Flujo alternativo	4. El Entrenador puede hacer click en el botón de “Añadir alimento” (Caso de uso Nuevo Alimento)
Postcondición	Ninguna

CASO DE USO: NUEVO ALIMENTO	
Resumen de funcionalidad	Crea un nuevo alimento en el sistema
Actores	Entrenador personal
Casos de uso relacionados	Edición alimento
Precondición	El alimento no existe en el sistema
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los alimentos ordenados alfabéticamente 3. El entrenador puede hacer click en el botón de “Añadir alimento” 4. El entrenador rellena los campos del nuevo alimento. 5. El sistema comprueba que todos los datos son correctos. 6. El entrenador guarda los datos del alimento haciendo clic en el botón guardar y volvemos a la lista de alimentos (caso de uso “Listar alimentos”)
Flujo alternativo	5. El sistema detecta algún error y se muestra por pantalla al usuario 5 El entrenador cancela o cierra la ventana
Postcondición	Se crea un nuevo alimento en el sistema

CASO DE USO: EDITAR ALIMENTO	
Resumen de funcionalidad	Se modifican los datos de un alimento
Actores	Entrenador personal
Casos de uso relacionados	Listar alimentos; consultar alimento
Precondición	Se está editando un alimento existente
Flujo normal	1. El entrenador personal entra en el

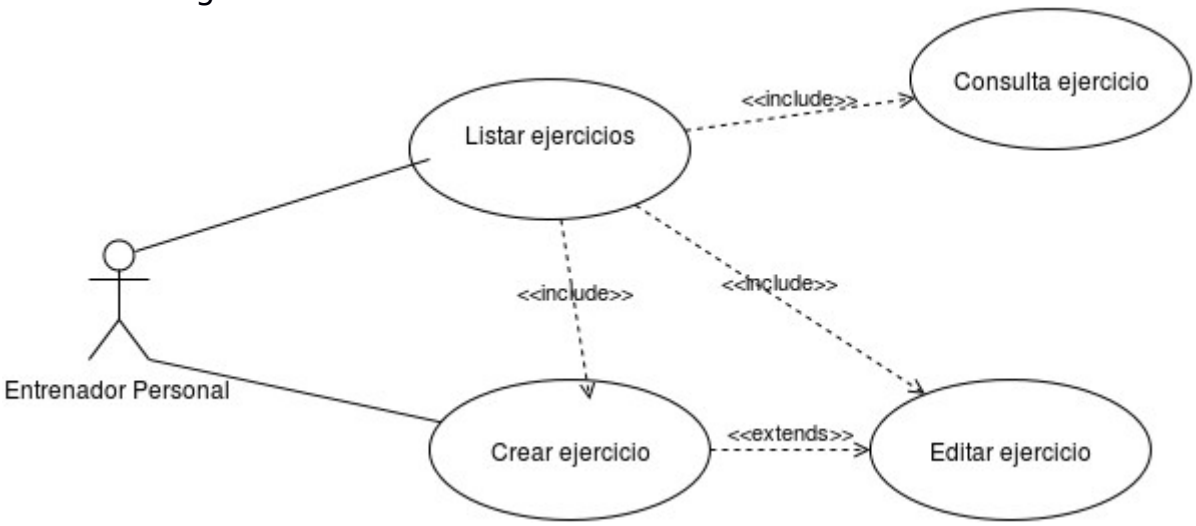
	modulo clientes. 2. Aparece una lista con todos los alimentos ordenados alfabéticamente 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el alimento que lo desee para ver su ficha completa. 5. El entrenador modifica los datos que desee. 6. El sistema comprueba que todos los datos son correctos. 7. El entrenador guarda los datos del alimento haciendo clic en el botón guardar y volvemos a la lista de alimentos
Flujo alternativo	5. El entrenador cancela la edición 6. El sistema detecta algún error y se lo muestra al usuario por pantalla.
Postcondición	El alimento es actualizado en el sistema
CASO DE USO: CONSULTAR ALIMENTO	
Resumen de funcionalidad	Se muestran las propiedades de un alimento
Actores	Entrenador personal
Casos de uso relacionados	Listar alimentos
Precondición	Se está consultando un alimento existente
Flujo normal	1. El entrenador personal entra en el modulo alimentos. 2. Aparece una lista con todos los alimentos ordenados alfabéticamente 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el alimento que lo desee para ver su ficha completa. 5. El sistema devuelve los datos del alimento seleccionado en una nueva ventana

	6. El entrenador cierra la ventana o pulsa el botón cancelar.
Flujo alternativo	
Postcondición	Ninguna

MÓDULO DE GESTIÓN DE EJERCICIOS

Este módulo permite la gestión de ejercicios físicos de la aplicación. Permite la gestión básica de los ejercicios de la aplicación que se usarán posteriormente para la confección de los programas de entrenamiento. A continuación se presenta el diagrama UML de los diferentes casos de uso.

A



continuación una explicación textual de los casos de uso.

CASO DE USO: LISTAR EJERCICIOS	
Resumen de funcionalidad	Muestra un listado con los ejercicios actuales
Actores	Entrenador personal
Casos de uso relacionados	
Precondición	Ninguna al ser una consulta
Flujo normal	1. El entrenador personal entra en el modulo de ejercicios. 2. Aparece una lista con todos los ejercicios ordenada alfabéticamente 3. El listado permite filtrar los resultados por varios campos.

	4. El Entrenador puede hacer click en el ejercicio que lo desee para ver su ficha completa(Caso de uso Editar ejercicio).
Flujo alternativo	4. El Entrenador puede hacer click en el botón de “Añadir ejercicio” (Caso de uso Nuevo ejercicio)
Postcondición	Ninguna

CASO DE USO: NUEVO EJERCICIO	
Resumen de funcionalidad	Crea un nuevo ejercicio en el sistema
Actores	Entrenador personal
Casos de uso relacionados	Edición ejercicio
Precondición	El ejercicio no existe en el sistema
Flujo normal	1. El entrenador personal entra en el modulo ejercicios. 2. Aparece una lista con todos los ejercicios ordenados alfabéticamente 3. El entrenador puede hacer click en el botón de “Añadir ejercicio” 4. El entrenador rellena los campos del nuevo ejercicio. 5. El sistema comprueba que todos los datos son correctos. 6. El entrenador guarda los datos del ejercicio haciendo clic en el botón guardar y volvemos a la lista de ejercicios (caso de uso “Listar ejercicios”)
Flujo alternativo	5. El sistema detecta algún error y se muestra por pantalla al usuario 5 El entrenador cancela o cierra la ventana
Postcondición	Se crea un nuevo ejercicio en el sistema

CASO DE USO: EDITAR EJECICIO	
Resumen de funcionalidad	Se modifican los datos de un ejercicio
Actores	Entrenador personal
Casos de uso relacionados	Listar ejercicios; consultar ejercicio
Precondición	Se está editando un ejercicio existente
Flujo normal	1. El entrenador personal entra en el modulo ejercicios. 2. Aparece una lista con todos los ejercicios ordenados alfabéticamente 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el ejercicio que lo desee para ver su ficha completa. 5. El entrenador modifica los datos que desee. 6. El sistema comprueba que todos los datos son correctos. 7. El entrenador guarda los datos del ejercicio haciendo clic en el botón guardar y volvemos a la lista de ejercicios
Flujo alternativo	5. El entrenador cancela la edición 6. El sistema detecta algún error y se lo muestra al usuario por pantalla.
Postcondición	El ejercicio es actualizado en el sistema

CASO DE USO: CONSULTAR EJECICIO	
Resumen de funcionalidad	Se muestran las propiedades de un ejercicio
Actores	Entrenador personal
Casos de uso relacionados	Listar ejercicios
Precondición	Se está consultando un ejercicio

	existente
Flujo normal	1. El entrenador personal entra en el modulo ejercicios. 2. Aparece una lista con todos los ejercicios ordenados alfabéticamente 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el ejercicio que lo desee para ver su ficha completa. 5. El sistema devuelve los datos del ejercicio seleccionado en una nueva ventana 6. El entrenador cierra la ventana o pulsa el botón cancelar.
Flujo alternativo	
Postcondición	Ninguna

ALCANCE DEL PROYECTO

Hay diversos casos de uso que debido al tiempo para el desarrollo de este TFC no se llevarán a cabo. Algunos de ellos son:

- Gestión de dietas
- Gestión de rutinas
- Búsqueda de alimentos similares para la confección de dietas
- Búsqueda de ejercicios similares para la confección de entrenamientos

OTROS REQUERIMIENTOS

En las diferentes reuniones con los usuarios finales se ha hecho bastante hincapié en el tema de la movilidad y la usabilidad de la aplicación. Es muy importante que la aplicación pueda verse en diferentes formatos y que todo se realice de manera muy fácil en intuitiva.

INTERFACES DE USUARIO

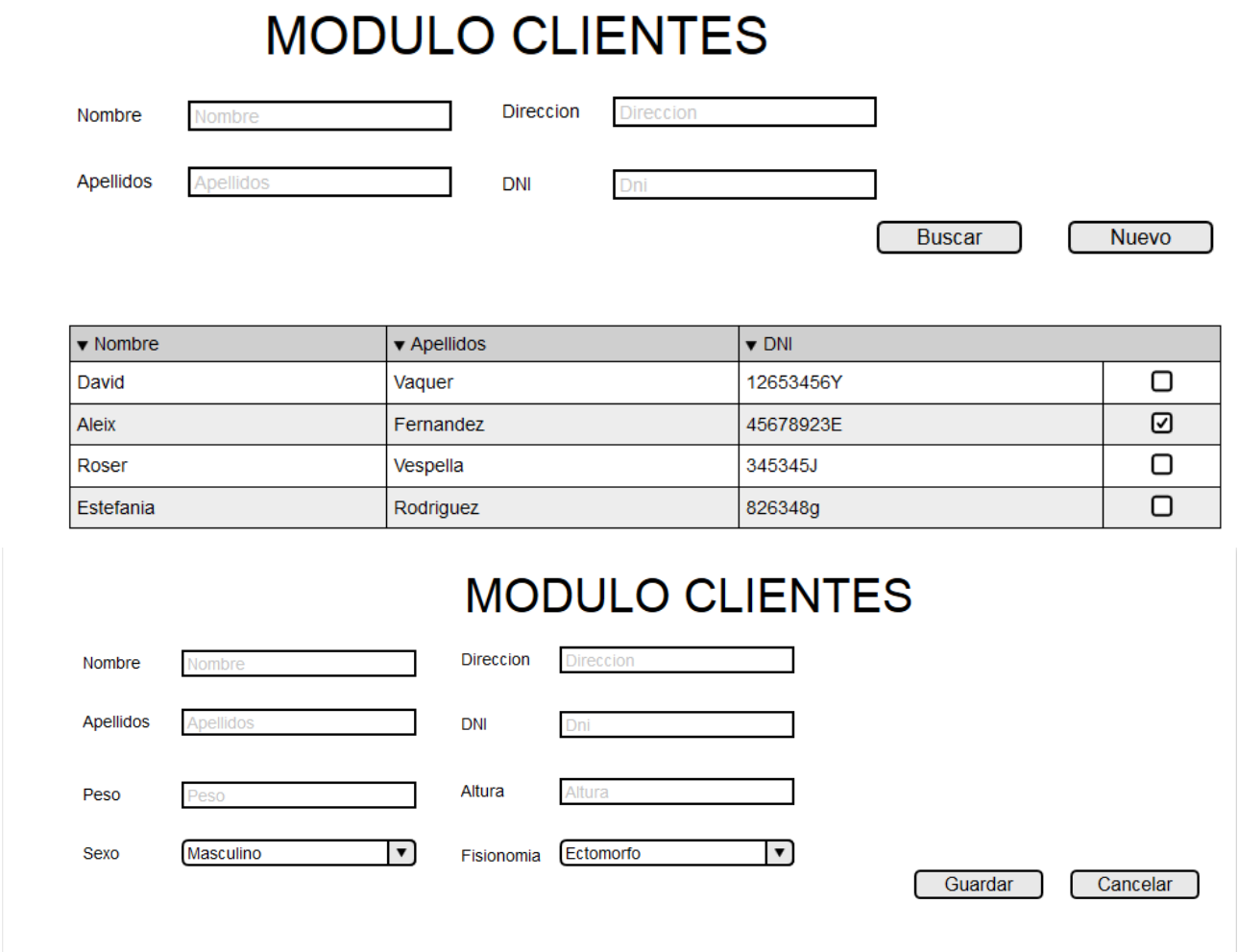
Toda la aplicación va a desarrollarse bajo una interfaz de usuario WEB, concretamente con el uso del framework "Primefaces". El usuario deberá hacer uso de su navegador para acceder a la aplicación. El uso del framework "Primefaces" da la posibilidad de tener la aplicación totalmente adaptada a los navegadores más usados (firefox, internet explorer y chrome) así como responsibilidad (posibilidad de adaptarse a cualquier dispositivo, ya sea tablet

o movil) casi automática. Gracias al uso intensivo de la tecnología ajax y a la gran variedad de componentes en esta librería, la experiencia del usuario se verá gratamente favorecida.

PROTOTIPO DE INTERFAZ GRÁFICA

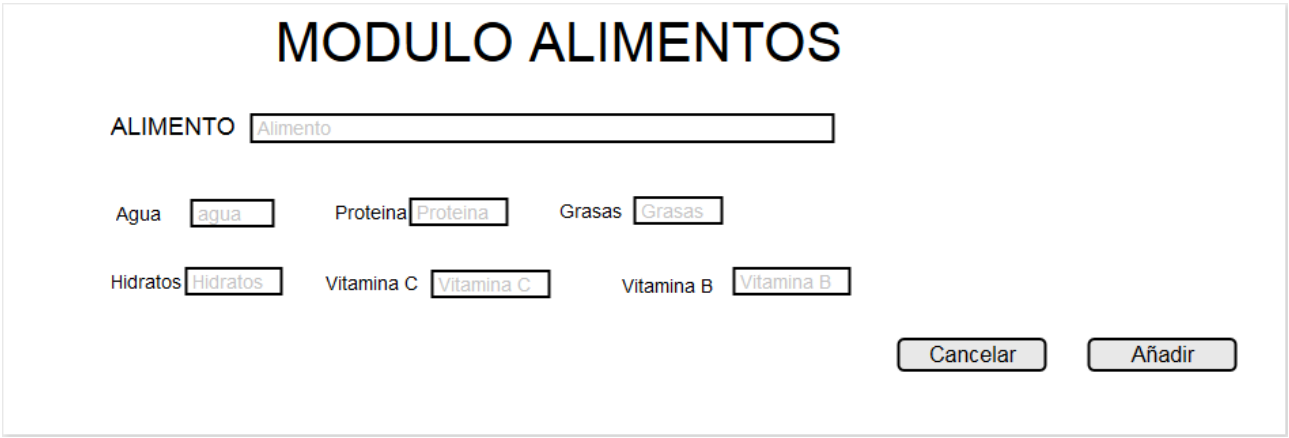
A continuación se muestran unos pequeños ejemplos de posibles pantallas que tendrían la aplicación. Hay que tener en cuenta que en la aplicación real, estos formularios contarán con hojas de estilo y diseños personalizados que harán la experiencia de usuario mucho más atractivo.
Hay que considerar también que el presente documento se entrega en una fase donde aún no se ha comenzado a implementar ningún tipo de desarrollo.

MÓDULO DE GESTIÓN DE CLIENTES



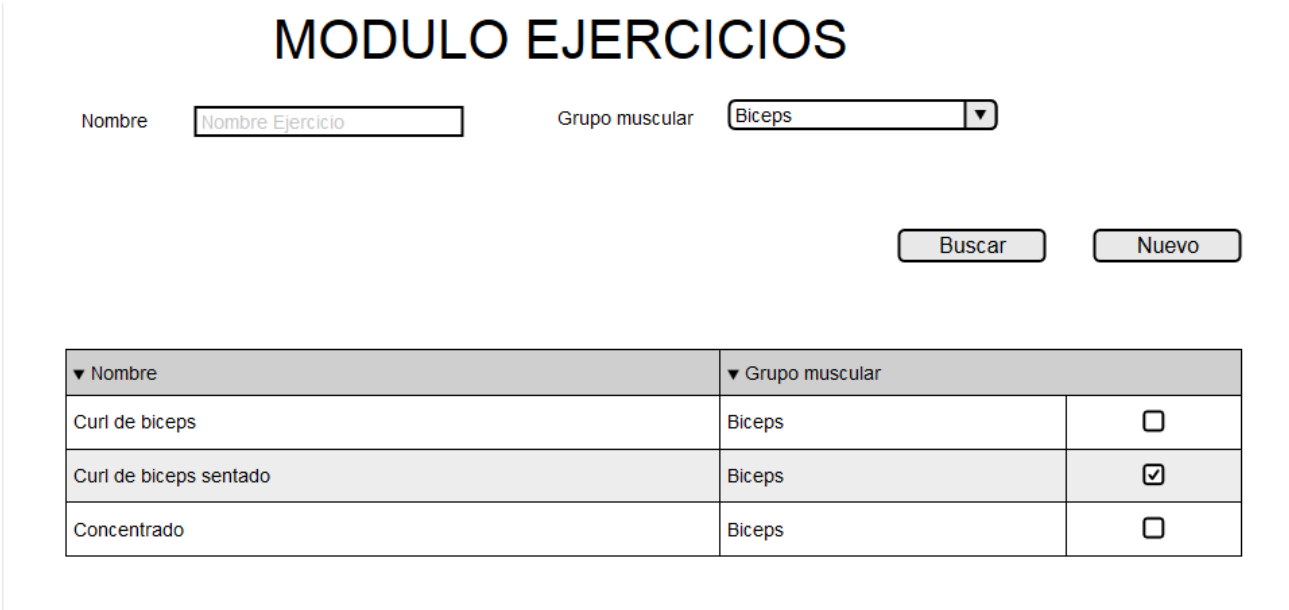
MÓDULO DE GESTTIÓN DE ALIMENTOS

A continuación se muestra un ejemplo de pantalla para añadir un elimento



MÓDULO DE GESTIÓN DE EJECICIOS

A continuación se muestra un ejemplo de pantalla para listar los ejercicios



• DISEÑO DE LA ARQUITECTURA

BUENAS PRÁCTICAS

Una de las consideraciones más importantes a nivel de diseño que creo que es importante remarcar, es la cuestión de las buenas prácticas a la hora de desarrollar el código fuente. Si lo que buscamos es que el código fuente de nuestro TFC (o de cualquier proyecto que desarrollemos en el futuro) cuente con un mínimo de calidad, es imprescindible que hagamos uso de las llamadas “buenas prácticas de la programación”. Uno de los pilares imprescindibles en estas buenas prácticas es el uso de patrones ¿qué es un patrón ? Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software. En otras palabras, lo que se busca con los patrones es “no reinventar la rueda”. A continuación describo algunos de los patrones que me gustaría implementar en el TFC.

PATRÓN	CAPA	DESCRIPCIÓN
Front Controller	Capa de presentación	Un objeto que acepta todos los requerimientos de un cliente y los direcciona a manejadores apropiados.
Composite view	Capa de presentación	Un objeto vista que está compuesto de otros objetos vista.
Intercepting filter	Capa de presentación	La capa de presentación recibe multitud de solicitudes que necesitan distintos tipos de procesamiento.
Business Delegate	Capa de negocio	Un objeto que reside en la capa de presentación y en beneficio de los otros componentes de la capa de presentación llama a métodos remotos en los objetos de la capa de negocios.
Transfer object	Transversal	Utilizar un objeto de transferencia para encapsular los datos de negocio. Una única llamada al método se utiliza para enviar y recuperar el objeto de transferencia.
Data Access object	Capa de persistencia	Utilizar un Data Access Object (DAO) para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtenerlos y almacenarlos.

CONVENCIÓN DE NOMBRES

Como toda aplicación “seria” que se precie, la nomenclatura del código fuente de la misma tiene que seguir unos estándares mínimos de calidad. En nuestro caso seguiremos las especificaciones que nos indica SUN: http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-

136057.html

A continuación extraemos un pequeño resumen de los puntos más importantes, aplicable a todos ellos es el tema de la claridad de los mismos donde se buscará la autodefinición siempre que se pueda:

PAQUETES
Siempre deberán ir en minúsculas y siempre empezarán por

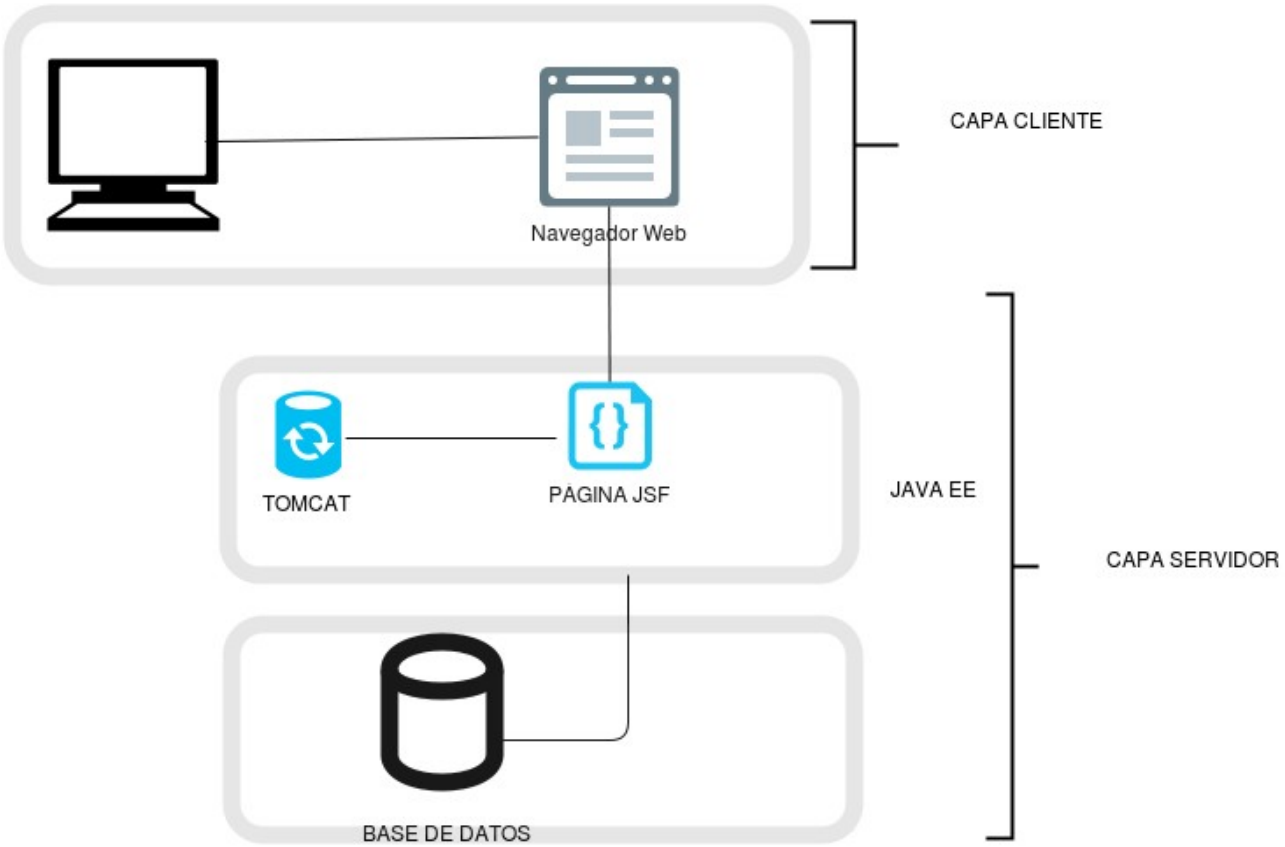
CLASES
El nombre de las clases tiene que empezar siempre por mayúscula y en modo “CamelCase” cuando proceda, a poder ser se usarán sustantivos.

MÉTODOS
Los métodos de las clases empiezan siempre en minúscula y siguen el patrón “CamelCase” cuando proceda. A poder ser se usarán verbos.

ATRIBUTOS
Los atributos de las clases van siempre en minúscula y siguen el patrón “CamelCase” cuando proceda.

DISEÑO Y PATRONES

Como se ha comentado a lo largo del presente documento, la aplicación para la gestión del entrenamiento físico y la supervisión nutricional de deportistas estará completamente desarrollada en un entorno JavaEE, es por ello que la solución aportada es una solución Web.
Al tratarse de una aplicación Web, necesitaremos de un servidor de aplicaciones, concretamente para este proyecto usaremos un servidor Tomcat en su última versión. A continuación se muestra un ejemplo de arquitectura física:

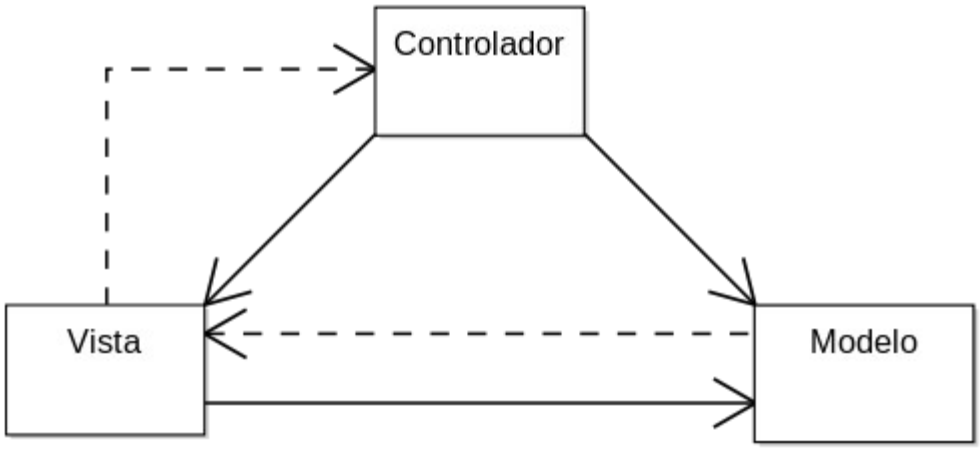


El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos.

Modelo: representación específica de la información con la cual el sistema opera. Ejemplo Client.java.

Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario. Ejemplo EditClient.xhtml

Controlador: Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista. Básicamente encapsula toda la lógica de la aplicación. Ejemplo EditClientBean.java.



CAPA DE PERSISTENCIA

Si la aplicación está diseñada con orientación a objetos, la persistencia se logra por serialización del objeto o almacenamiento en una base de datos. Las bases de datos más populares hoy en día son relacionales.

El modelo de objetos difiere en muchos aspectos del modelo relacional. La interfaz que une esos dos modelos se llama asociación objeto-relacional (ORM en inglés). Una capa de persistencia encapsula el comportamiento necesario para mantener los objetos. O sea: leer, escribir y borrar objetos en el almacenamiento persistente (base de datos).

La asociación objeto-relacional (más conocido por su nombre en inglés, Object-Relational Mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Hay paquetes comerciales y de uso libre disponibles que desarrollan la asociación relacional de objetos, aunque algunos programadores prefieren crear sus propias herramientas ORM.

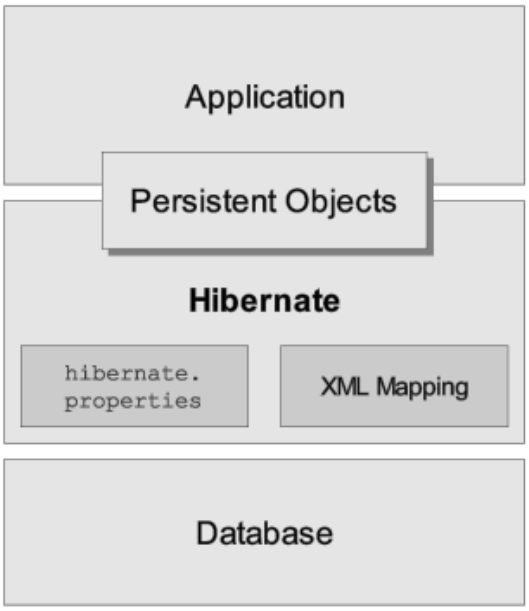
Un objeto está compuesto de propiedades y métodos. Como las propiedades representan a la parte estática de ese objeto, son las partes que se dotan de persistencia. Cada propiedad puede ser simple o compleja.Por simple, se entiende que tiene algún tipo de datos nativos como por ejemplo: entero, coma flotante o cadena de caracteres. Por complejo se entiende algún tipo definido por el usuario, ya sean objetos o estructuras. Por relación se entiende asociación, herencia o agregación. Para dotar de persistencia las relaciones, se usan transacciones, ya que los cambios pueden incluir varias tablas.

Para vincular las relaciones, se usan los identificadores de objetos (OID). Estos OID se agregan como una columna más en la tabla donde se quiere establecer la relación. Dicha columna es una clave foránea a la tabla con la que se está relacionada. Así, queda asignada la relación. Recordar que las relaciones en el modelo relacional son siempre bidireccionales.

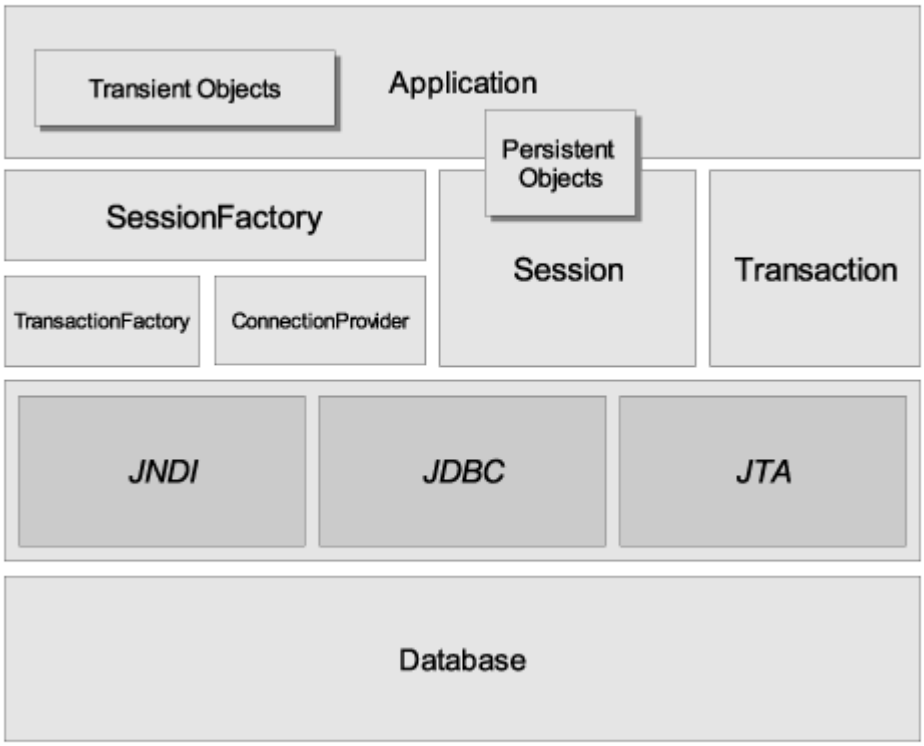
HIBERNATE

Hibernate es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

El diagrama a continuación brinda una perspectiva a alto nivel de la arquitectura de Hibernate:



La arquitectura "completa" abstrae la aplicación de las APIs de JDBC/JTA y permite que Hibernate se encargue de los detalles.



Hibernate necesita saber cómo cargar y almacenar objetos de la clase persistente. En este punto es donde entra en juego el archivo de mapeo de Hibernate. Este archivo le dice a Hibernate a que tabla tiene que acceder en la base de datos, y que columnas debe utilizar en esta tabla.

La estructura básica de un archivo de mapeo se ve así:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="uoc.example.domain">
   <class name=" uoc.example.domain " table="CLIENT" schema="dbo"
catalog="TFC">
            <id name="idClient" type="int">
                <column name="ID_CLIENT" />
            </id>
            <property name="nombreCliente" type="string">
```

```
                <column name="NOMBRE_CLIENTE" length="30" />
            </property>

    </class>
 [...]
</hibernate-mapping>
```

El DTD de Hibernate es sofisticado. Puede utilizarlo para autocompletar los elementos y atributos XML de mapeo en su editor o IDE. Abrir el archivo DTD en su editor de texto es la manera más fácil para obtener una sinopsis de todos los elementos y atributos y para ver los valores por defecto, así como algunos de los comentarios. Note que Hibernate no cargará el fichero DTD de la web, sino que primero lo buscará en la ruta de clase de la aplicación. El archivo DTD se encuentra incluido en hibernate-core.jar.

CAPA DE NEGOCIO

La capa de negocio consta de la lógica del sistema: reglas de negocio, workflows de negocio y operaciones no persistentes. Todas las operaciones persistentes son delegadas a la capa de acceso de datos. La capa de negocio debería ser vista como un conjunto de servicios expuestos a las capas de presentación de las aplicaciones. La idea es que todos los módulos que requieren una funcionalidad la encuentren en un solo lugar y con una sola versión, no hay réplica de funcionalidades en un formulario u otro lugar.

SPRING

Spring es un framework de aplicaciones Java/J2EE desarrollado usando licencia de OpenSource.Se basa en una configuración a base de javabeans bastante simple. Es potente en cuanto a la gestión del ciclo de vida de los componentes y fácilmente ampliable. Es interesante el uso de programación orientada a aspectos (IoC). Tiene plantillas que permiten un más fácil uso de Hibernate, iBatis, JDBC..., se integra "de fábrica" con Quartz, Velocity, Freemarker, Struts, Webwork2 y tienen un plugin para eclipse.

Ofrece un ligero contenedor de bean para los objetos de la capa de negocio, DAOs y repositorio de Datasources JDBC y sesiones Hibernate. Mediante un xml definimos el contexto de la aplicación siendo una potente herramienta para manejar objetos Singleton o “factorias” que necesitan su propia configuración.

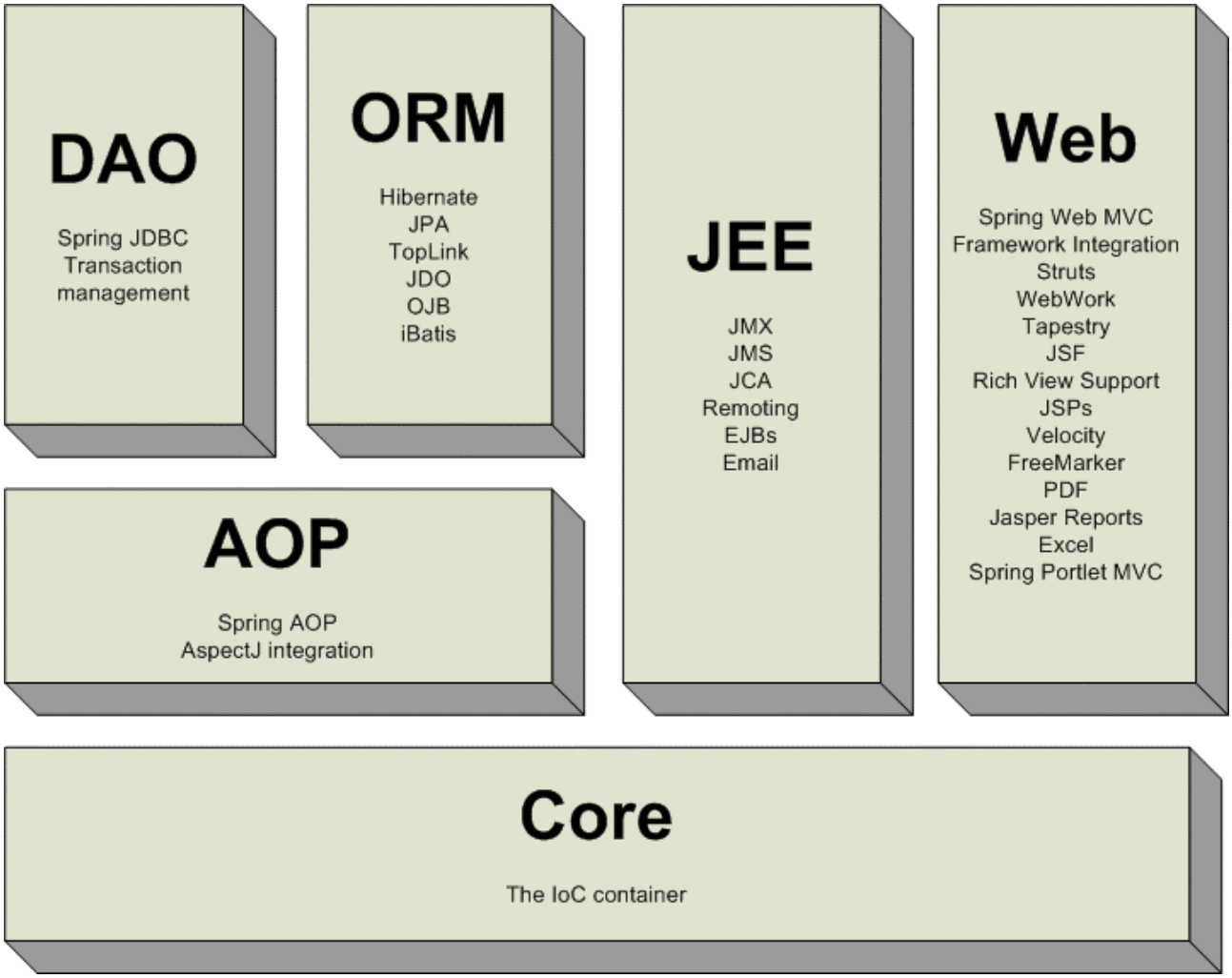
El objetivo de Spring es no ser intrusivo, aquellas aplicaciones configuradas para usar beans mediante Spring no necesitan depender de interfaces o clases

de Spring pero obtienen su configuración a través de las propiedades de sus beans.Este concepto puede ser aplicado a cualquier entorno, desde una aplicación J2EE a un applet

El paquete “Core” es la parte más fundamental del framework ya que provee a este, las características de Inversión de Control (IoC) y Inyección de dependencias (ID). El concepto básico dentro del “Core” es el “BeanFactory”, el cual provee una sofisticada implementación del “Patrón Factory”, que remueve la necesidad generalizada de “Singletons” y nos permite desacoplar la configuración y especificación de las dependencias de nuestra lógica de programación.

La inicial motivación era facilitar el desarrollo de aplicaciones J2EE, promoviendo buenas prácticas de diseño y programación. En concreto se trata de manejar patrones de diseño como Factory, Abstract Factory, Builder, Decorator, Service Locator, etc; que son ampliamente reconocidos dentro de la industria del desarrollo de software.

Dentro de una arquitectura en capas una ventaja de Spring es su modularidad, pudiendo usar algunos de los módulos sin comprometerse con el uso del resto:



Normalmente Spring es usado como “pegamento” o “cemento” entre los diferentes frameworks de una aplicación.

INYECCIÓN DE DEPENDENCIAS

Para Spring todo son “beans”, un bean, en el contexto de Spring, es un objeto que es creado y manejado por el contenedor Spring. Es importante destacar la diferencia con respecto al uso clásico de 'bean' en J2EE: en Spring el bean no es una clase que cumple una serie de normas o restricciones, sino que es un objeto.
Ejemplo:

```
<beans>

    <bean id="quest"
        class="HolyGrailQuest"/>          define el bean quest


    <bean id="knight"
        class="KnightOfTheRoundtable"/>   define el bean knight

        <contructor-arg>
            <value>Bedivere</value>
        </contructor-arg>
        <property name="quest">           set del bean quest al bean knight
            <ref bean="quest"/>           (inyección de dependencia)
        </property>

    </bean>
</beans>
```

AOP

La Programación Orientada a Aspectos (POA en español, AOP en inglés) es un concepto relativamente antiguo, y que ha tenido un éxito relativamente discreto.

Básicamente consiste en implementar por un lado un aspecto, que es un trozo de código común a ejecutar en varios métodos / clases (Ejemplo logger) y por otro lado configuraremos los puntos de cruce, que son los lugares en los que el aspecto cruza con el código, es decir, cuando serán ejecutados.

Para que funcione debemos interponer un Proxy entre la clase y el aspecto. De esta forma podemos acoplar / desacoplar aspectos comunes sin tocar código.

CAPA DE PRESENTACIÓN

La capa de presentación del TFC estará desarrollada con el framework JSF 2.0, concretamente con la implementación de Primefaces en su versión estable más reciente.

JSF

Es una tecnología para aplicaciones Java basadas en Web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

Las características principales de JSF son:

- Asocia a cada vista con un conjunto de objetos java manejados por el controlador (managed beans) que facilitan la recogida, manipulación y visualización de los valores mostrados en los diferentes elementos de los formularios.

- Introduce una serie de etapas en el procesamiento de la petición, como por ejemplo la de validación, reconstrucción de la vista, recuperación de los valores de los elementos, etc.

- Utiliza un sencillo fichero de configuración para el controlador en formato xml

- Es extensible, pudiendo crearse nuevos elementos de la interfaz o modificar los ya existentes.

- Y lo que es más importante: forma parte del estándar J2EE.

- JSF resuelve validaciones, conversiones, mensajes de error e internacionalización (i18n).

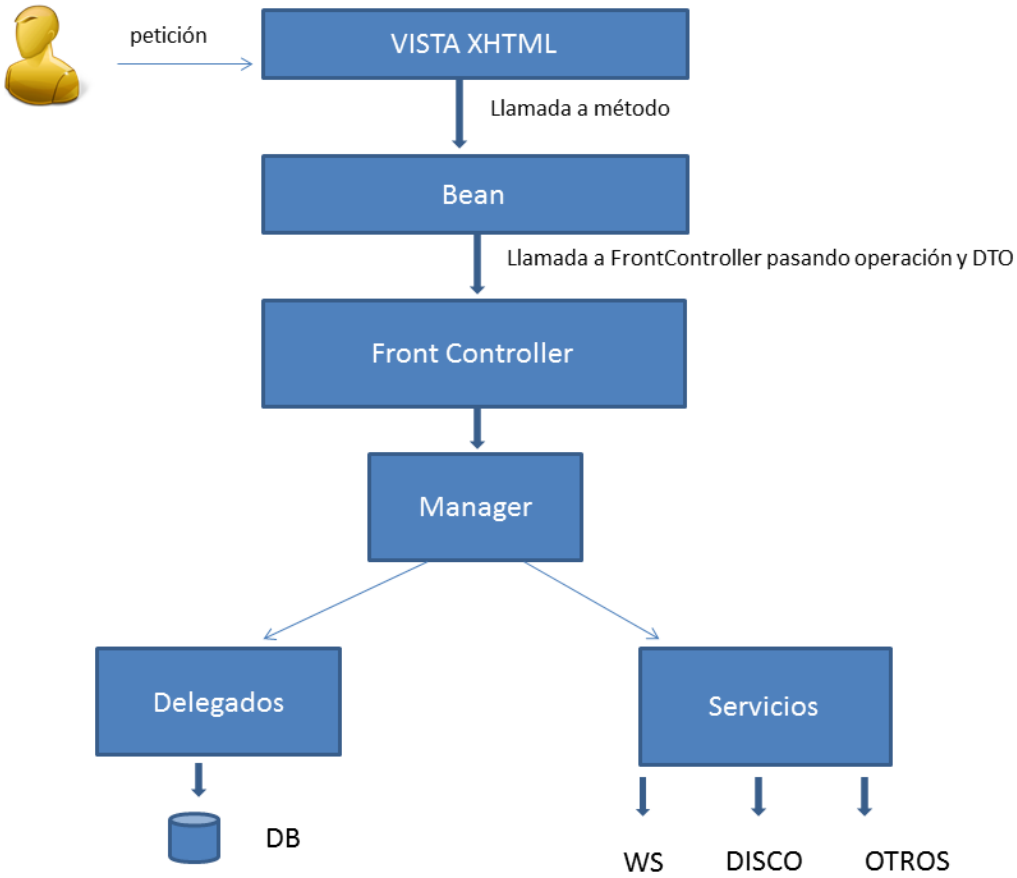
- JSF es extensible, por lo que se pueden desarrollar nuevos componentes a medida, También se puede modificar el comportamiento del framework mediante APIs que controlan su funcionamiento.

PRIMEFACES

Primefaces es un API con más de 90 componentes para JSF. Las principales características de Primefaces son:

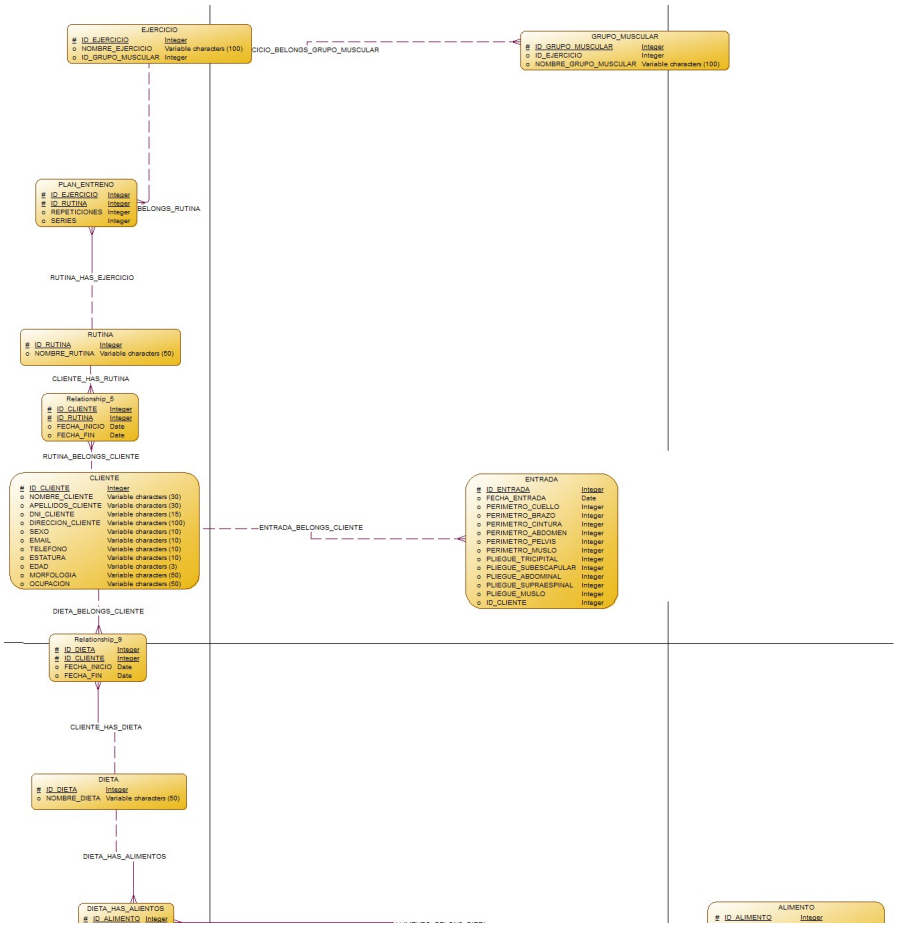
- Soporte nativo de Ajax, incluyendo Push/Comet.
- Kit para crear aplicaciones web para móviles.
- Es compatible con otras librerías de componentes, como JBoss RichFaces.
- Uso de javascript no intrusivo (no aparece en línea dentro de los elementos, sino dentro de un bloque <script>).
- Es un proyecto open source, activo y bastante estable entre versiones.

A continuación se muestra una imagen que intenta resumir nuestro diseño desde la petición hasta la consulta SQL.



DISEÑO DE LA BASE DE DATOS

Como se ha comentado en este y en el documento de Anális hay que tener en cuenta que este primer diseño de base de datos puede diferir en el diseño final de la aplicación. A medida que se empiece a crear el prototipo y los usuarios comiecen a usar la aplicación, este modelo sufrirá cambios.



TABLAS

De acuerdo a tener un documento más próximo a la realidad y ayudándome de un programa (power designer) he creado un modelo lógico basado en mi abstracción del modelo entidad relación, de eso modelo lógico (y de nuevo ayudándome del Power Designer) he creado un modelo físico adaptado a la base de datos con la que trabajaré a lo largo del proyecto (postgreSQL). De ese modelo físico he podido obtener un script de creación de base de datos donde se pueden apreciar tanto las tablas, como las relaciones entre tablas.

```
/*==============================================================*/
/* DBMS name:      PostgreSQL 8                                 */
/*
    */
/*==============================================================*/
```

```
drop index ALIMENTO_PK;

drop table ALIMENTO;

drop index CLIENTE_PK;

drop table CLIENTE;

drop index DIETA_PK;

drop table DIETA;

drop index ALIMENTO_BELONS_DIETA_FK;

drop index DIETA_HAS_ALIMENTOS_FK;

drop index DIETA_HAS_ALIENTOS_PK;

drop table DIETA_HAS_ALIENTOS;

drop index EJERCICIO_PK;

drop table EJERCICIO;

drop index ENTRADA_BELONGS_CLIENTE_FK;

drop index ENTRADA_PK;

drop table ENTRADA;

drop index EJERCICIO_BNGS_GRUPO_MUAR_FK;

drop index GRUPO_MUSCULAR_PK;

drop table GRUPO_MUSCULAR;

drop index EJERCICIO_BELONGS_RUTINA_FK;

drop index RUTINA_HAS_EJERCICIO_FK;

drop index PLAN_ENTRENO_PK;

drop table PLAN_ENTRENO;

drop index RUTINA_BELONGS_CLIENTE_FK;

drop index CLIENTE_HAS_RUTINA_FK;

drop index RELATIONSHIP_5_PK;

drop table RELATIONSHIP_5;

drop index CLIENTE_HAS_DIETA_FK;

drop index DIETA_BELONGS_CLIENTE_FK;

drop index RELATIONSHIP_9_PK;
```

```
drop table RELATIONSHIP_9;

drop index RUTINA_PK;

drop table RUTINA;

/*==============================================================*/
/* Table: ALIMENTO                                              */
/*==============================================================*/
create table ALIMENTO (
   ID_ALIMENTO          INT4                not null,
   NOMBRE_ALIMENTO      VARCHAR(50)         null,
   AGUA                 INT4                null,
   CALORIAS             INT4                null,
   PROTEINAS            INT4                null,
   GRASAS               INT4                null,
   GLUCIDOS             INT4                null,
   constraint PK_ALIMENTO primary key (ID_ALIMENTO)
);

/*==============================================================*/
/* Index: ALIMENTO_PK                                           */
/*==============================================================*/
create unique index ALIMENTO_PK on ALIMENTO (
ID_ALIMENTO
);

/*==============================================================*/
/* Table: CLIENTE                                               */
/*==============================================================*/
create table CLIENTE (
   ID_CLIENTE           INT4                not null,
   NOMBRE_CLIENTE       VARCHAR(30)         null,
   APELLIDOS_CLIENTE    VARCHAR(30)         null,
   DNI_CLIENTE          VARCHAR(15)         null,
   DIRECCION_CLIENTE    VARCHAR(100)        null,
   SEXO                 VARCHAR(10)         null,
   EMAIL                VARCHAR(10)         null,
   TELEFONO             VARCHAR(10)         null,
   ESTATURA             VARCHAR(10)         null,
   EDAD                 VARCHAR(3)          null,
   MORFOLOGIA           VARCHAR(50)         null,
   OCUPACION            VARCHAR(50)         null,
   constraint PK_CLIENTE primary key (ID_CLIENTE)
);

/*==============================================================*/
/* Index: CLIENTE_PK                                            */
/*==============================================================*/
create unique index CLIENTE_PK on CLIENTE (
ID_CLIENTE
);

/*==============================================================*/
/* Table: DIETA                                                 */
/*==============================================================*/
```

```
create table DIETA (
   ID_DIETA             INT4                 not null,
   NOMBRE_DIETA         VARCHAR(50)          null,
   constraint PK_DIETA primary key (ID_DIETA)
);

/*==============================================================*/
/* Index: DIETA_PK                                              */
/*==============================================================*/
create unique index DIETA_PK on DIETA (
ID_DIETA
);

/*==============================================================*/
/* Table: DIETA_HAS_ALIENTOS                                    */
/*==============================================================*/
create table DIETA_HAS_ALIENTOS (
   ID_ALIMENTO          INT4                 not null,
   ID_DIETA             INT4                 not null,
   CANTIDAD             INT4                 null,
   constraint PK_DIETA_HAS_ALIENTOS primary key (ID_ALIMENTO, ID_DIETA)
);

/*==============================================================*/
/* Index: DIETA_HAS_ALIENTOS_PK                                 */
/*==============================================================*/
create unique index DIETA_HAS_ALIENTOS_PK on DIETA_HAS_ALIENTOS (
ID_ALIMENTO,
ID_DIETA
);

/*==============================================================*/
/* Index: DIETA_HAS_ALIMENTOS_FK                                */
/*==============================================================*/
create  index DIETA_HAS_ALIMENTOS_FK on DIETA_HAS_ALIENTOS (
ID_DIETA
);

/*==============================================================*/
/* Index: ALIMENTO_BELONS_DIETA_FK                              */
/*==============================================================*/
create  index ALIMENTO_BELONS_DIETA_FK on DIETA_HAS_ALIENTOS (
ID_ALIMENTO
);

/*==============================================================*/
/* Table: EJERCICIO                                             */
/*==============================================================*/
create table EJERCICIO (
   ID_EJERCICIO         INT4                 not null,
   NOMBRE_EJERCICIO     VARCHAR(100)         null,
   ID_GRUPO_MUSCULAR    INT4                 null,
   constraint PK_EJERCICIO primary key (ID_EJERCICIO)
);

/*==============================================================*/
/* Index: EJERCICIO_PK                                          */
```

```
/*==============================================================*/
create unique index EJERCICIO_PK on EJERCICIO (
ID_EJERCICIO
);

/*==============================================================*/
/* Table: ENTRADA                                               */
/*==============================================================*/
create table ENTRADA (
   ID_ENTRADA           INT4                 not null,
   FECHA_ENTRADA        DATE                 null,
   PERIMETRO_CUELLO     INT4                 null,
   PERIMETRO_BRAZO      INT4                 null,
   PERIMETRO_CINTURA    INT4                 null,
   PERIMETRO_ABDOMEN    INT4                 null,
   PERIMETRO_PELVIS     INT4                 null,
   PERIMETRO_MUSLO      INT4                 null,
   PLIEGUE_TRICIPITAL   INT4                 null,
   PLIEGUE_SUBESCAPULAR INT4                 null,
   PLIEGUE_ABDOMINAL    INT4                 null,
   PLIEGUE_SUPRAESPINAL INT4                 null,
   PLIEGUE_MUSLO        INT4                 null,
   ID_CLIENTE           INT4                 null,
   constraint PK_ENTRADA primary key (ID_ENTRADA)
);

/*==============================================================*/
/* Index: ENTRADA_PK                                            */
/*==============================================================*/
create unique index ENTRADA_PK on ENTRADA (
ID_ENTRADA
);

/*==============================================================*/
/* Index: ENTRADA_BELONGS_CLIENTE_FK                            */
/*==============================================================*/
create  index ENTRADA_BELONGS_CLIENTE_FK on ENTRADA (
ID_CLIENTE
);

/*==============================================================*/
/* Table: GRUPO_MUSCULAR                                        */
/*==============================================================*/
create table GRUPO_MUSCULAR (
   ID_GRUPO_MUSCULAR    INT4                 not null,
   ID_EJERCICIO         INT4                 null,
   NOMBRE_GRUPO_MUSCULAR VARCHAR(100)         null,
   constraint PK_GRUPO_MUSCULAR primary key (ID_GRUPO_MUSCULAR)
);

/*==============================================================*/
/* Index: GRUPO_MUSCULAR_PK                                     */
/*==============================================================*/
create unique index GRUPO_MUSCULAR_PK on GRUPO_MUSCULAR (
ID_GRUPO_MUSCULAR
);
```

```
/*==============================================================*/
/* Index: EJERCICIO_BNGS_GRUPO_MUAR_FK                          */
/*==============================================================*/
create  index EJERCICIO_BNGS_GRUPO_MUAR_FK on GRUPO_MUSCULAR (
ID_EJERCICIO
);

/*==============================================================*/
/* Table: PLAN_ENTRENO                                          */
/*==============================================================*/
create table PLAN_ENTRENO (
   ID_EJERCICIO         INT4                 not null,
   ID_RUTINA            INT4                 not null,
   REPETICIONES         INT4                 null,
   SERIES               INT4                 null,
   constraint PK_PLAN_ENTRENO primary key (ID_EJERCICIO, ID_RUTINA)
);

/*==============================================================*/
/* Index: PLAN_ENTRENO_PK                                       */
/*==============================================================*/
create unique index PLAN_ENTRENO_PK on PLAN_ENTRENO (
ID_EJERCICIO,
ID_RUTINA
);

/*==============================================================*/
/* Index: RUTINA_HAS_EJERCICIO_FK                               */
/*==============================================================*/
create  index RUTINA_HAS_EJERCICIO_FK on PLAN_ENTRENO (
ID_RUTINA
);

/*==============================================================*/
/* Index: EJERCICIO_BELONGS_RUTINA_FK                           */
/*==============================================================*/
create  index EJERCICIO_BELONGS_RUTINA_FK on PLAN_ENTRENO (
ID_EJERCICIO
);

/*==============================================================*/
/* Table: RELATIONSHIP_5                                        */
/*==============================================================*/
create table RELATIONSHIP_5 (
   ID_CLIENTE           INT4                 not null,
   ID_RUTINA            INT4                 not null,
   FECHA_INICIO         DATE                 null,
   FECHA_FIN            DATE                 null,
   constraint PK_RELATIONSHIP_5 primary key (ID_CLIENTE, ID_RUTINA)
);

/*==============================================================*/
/* Index: RELATIONSHIP_5_PK                                     */
/*==============================================================*/
create unique index RELATIONSHIP_5_PK on RELATIONSHIP_5 (
ID_CLIENTE,
ID_RUTINA
```

```
);

/*==============================================================*/
/* Index: CLIENTE_HAS_RUTINA_FK                                 */
/*==============================================================*/
create  index CLIENTE_HAS_RUTINA_FK on RELATIONSHIP_5 (
ID_RUTINA
);

/*==============================================================*/
/* Index: RUTINA_BELONGS_CLIENTE_FK                             */
/*==============================================================*/
create  index RUTINA_BELONGS_CLIENTE_FK on RELATIONSHIP_5 (
ID_CLIENTE
);

/*==============================================================*/
/* Table: RELATIONSHIP_9                                        */
/*==============================================================*/
create table RELATIONSHIP_9 (
   ID_DIETA             INT4                 not null,
   ID_CLIENTE           INT4                 not null,
   FECHA_INICIO         DATE                 null,
   FECHA_FIN            DATE                 null,
   constraint PK_RELATIONSHIP_9 primary key (ID_DIETA, ID_CLIENTE)
);

/*==============================================================*/
/* Index: RELATIONSHIP_9_PK                                     */
/*==============================================================*/
create unique index RELATIONSHIP_9_PK on RELATIONSHIP_9 (
ID_DIETA,
ID_CLIENTE
);

/*==============================================================*/
/* Index: DIETA_BELONGS_CLIENTE_FK                              */
/*==============================================================*/
create  index DIETA_BELONGS_CLIENTE_FK on RELATIONSHIP_9 (
ID_CLIENTE
);

/*==============================================================*/
/* Index: CLIENTE_HAS_DIETA_FK                                  */
/*==============================================================*/
create  index CLIENTE_HAS_DIETA_FK on RELATIONSHIP_9 (
ID_DIETA
);

/*==============================================================*/
/* Table: RUTINA                                                */
/*==============================================================*/
create table RUTINA (
   ID_RUTINA            INT4                 not null,
   NOMBRE_RUTINA        VARCHAR(50)          null,
   constraint PK_RUTINA primary key (ID_RUTINA)
);
```

```
/*==============================================================*/
/* Index: RUTINA_PK                                             */
/*==============================================================*/
create unique index RUTINA_PK on RUTINA (
ID_RUTINA
);

alter table DIETA_HAS_ALIENTOS
   add constraint FK_DIETA_HA_ALIMENTO__ALIMENTO foreign key (ID_ALIMENTO)
      references ALIMENTO (ID_ALIMENTO)
      on delete restrict on update restrict;

alter table DIETA_HAS_ALIENTOS
   add constraint FK_DIETA_HA_DIETA_HAS_DIETA foreign key (ID_DIETA)
      references DIETA (ID_DIETA)
      on delete restrict on update restrict;

alter table ENTRADA
   add constraint FK_ENTRADA_ENTRADA_B_CLIENTE foreign key (ID_CLIENTE)
      references CLIENTE (ID_CLIENTE)
      on delete restrict on update restrict;

alter table GRUPO_MUSCULAR
   add constraint FK_GRUPO_MU_EJERCICIO_EJERCICI foreign key (ID_EJERCICIO)
      references EJERCICIO (ID_EJERCICIO)
      on delete restrict on update restrict;

alter table PLAN_ENTRENO
   add constraint FK_PLAN_ENT_EJERCICIO_EJERCICI foreign key (ID_EJERCICIO)
      references EJERCICIO (ID_EJERCICIO)
      on delete restrict on update restrict;

alter table PLAN_ENTRENO
   add constraint FK_PLAN_ENT_RUTINA_HA_RUTINA foreign key (ID_RUTINA)
      references RUTINA (ID_RUTINA)
      on delete restrict on update restrict;

alter table RELATIONSHIP_5
   add constraint FK_RELATION_CLIENTE_H_RUTINA foreign key (ID_RUTINA)
      references RUTINA (ID_RUTINA)
      on delete restrict on update restrict;

alter table RELATIONSHIP_5
   add constraint FK_RELATION_RUTINA_BE_CLIENTE foreign key (ID_CLIENTE)
      references CLIENTE (ID_CLIENTE)
      on delete restrict on update restrict;

alter table RELATIONSHIP_9
   add constraint FK_RELATION_CLIENTE_H_DIETA foreign key (ID_DIETA)
      references DIETA (ID_DIETA)
      on delete restrict on update restrict;

alter table RELATIONSHIP_9
   add constraint FK_RELATION_DIETA_BEL_CLIENTE foreign key (ID_CLIENTE)
      references CLIENTE (ID_CLIENTE)
      on delete restrict on update restrict;
```

```
    </class>
  [...]
</hibernate-mapping>
```

• DESVIACIONES DEL PROYECTO

INTRODUCCIÓN

Como todo proyecto de ingeniería del software existen una serie de desviaciones, ya sea de tiempo, alcance o coste que se producen al finalizar el proyecto y compararlo con el análisis inicial. En el caso que nos ocupa no tiene sentido hablar de desviaciones económicas porque no estamos hablando de un proyecto comercial, ni de desviaciones temporales ya que el tiempo estaba limitado a la duración del semestre. Lo que sí tiene mucho sentido hablar en este caso es de las desviaciones en el alcance del proyecto, puesto que al ser el tiempo limitado existía un riesgo aún mayor si cabe.

En esta última sección se detallarán las desviaciones aparecidas así como del origen, explicación y soluciones aplicadas a las mismas.

ALCANCE

La idea inicial del proyecto era tener dos tipos de usuarios, por una parte los entrenadores personales, que a la postre serían los administradores del sistema y por otra los clientes de esos entrenadores personales. Debido a razones de tiempo, el alcance del proyecto se ha limitado a centrarse en la parte que tiene que ver con los entrenadores personales, que a la postre es el tipo de aplicación que más se necesita (de hecho si se intenta buscar aplicaciones de este tipo se verá que no existen).

Los entrenadores personales serán los encargados de dar de alta atletas, confeccionar planes de entrenamiento y dietas y hacer el seguimiento de las evoluciones de éstos. Un arduo trabajo realizado previamente es la recopilación tanto de una base de datos de alimentos como el de una base de datos de ejercicios iniciales para que los entrenadores personales cuenten desde el primer instante con material para poder trabajar.

ARQUITECTURA

Dentro de los cambios realizados en el proyecto, con respecto a lo que se planteó en un principio, cabe destacar algunas pequeñas modificaciones en lo que a la arquitectura del proyecto se refiere.

Base de datos: debido a problemas con la configuración del SGBD PostgreSQL, que inicialmente iba a ser la opción elegida, se ha optado por usar su “hermana menor”, en este caso MySQL. La decisión de decantarse por una base de datos relacional es sencilla: la aplicación necesita, entre otras cosas, de integridad referencial y estructuras de datos concretas. MySQL aporta no solo robusted y fiabilidad, si no además una sencillísima configuración y administración tanto en los sistemas operativos Windows, como en los sistemas operativos Linux.

Capa de vista o front-end: la idea inicial para la capade la vista era delegar todo el peso en el framwrok Primefaces y su amplísima variedad de componentes. Junto con Primefaces se ha añadido una nueva librería que no estaba previsto en la versión inicial y es la de BootsFaces. Esta librería permitirá el desarrollo de interfaces gráficas de usuario mucho más amigable y, sobretodo, portable a todo tipo de dispositivos. He creído conveniente añadir esta librería porque en un futuro espero poder seguir ampliando las funcionalidades de la misma y sobretodo me interesa que sea portable a todo tipo de dispositivos. Bootsfaces aporta una amplia gama de componentes para JSF con el diseño “Bootstrap” tan famoso hoy en dia gracias a Twitter. Además de la amplia gama de componentes, también nos aporta la ser multidispositivo sin necesidad de programar las mismas interfaces gráficas para los distintos dispositivos.

Spring security: si bien es cierto que en la arquitectura inicial no estaba pensado añadir ninguna capa de seguridad, una vez que comencé el desarrollo de la aplicación y sobretodo pensando en su uso futuro, creí conveniente añadirle una sencilla capa inicial de seguridad. Esta capa está pensada por dos motivos, primero para dotar al proyecto de más robustez y riqueza y segundo porque pensando en una futura ampliación, tiene todo el sentido del mundo. Añadiendo esta capa de seguridad nos aseguramos de crear las bases para la autenticación/autorización de toda la aplicación. Siendo muy sencillo a partir de ahora añadir roles/usuarios con diferentes perfiles, siendo su escalabilidad muy sencilla.

Maven: una herramienta sencilla pero digna de mencionar que me ha ayudado muchísimo al desarrollo de la aplicación ha sido maven. Uno de los primero problemas que se encuentra un desarrollador cuando comienza un proyecto es la gestión de las librerías que va a usar. En muchos proyectos esa gestión se lleva de manera manual, incluyendo, elimiando, modificando versiones todo ello de manera manual, con los problemas que ello conlleva (falta de librerias, inconpatibilidad de versiones, etc...). Maven nos ayuda con esta gestión de una manera sencillísima, mediante un simple fichero de texto plano en formato XML

donde le iremos añadiendo las distintas librerías que conformarán nuestro proyecto. La manera de añadir librerías a nuestro proyecto es añadiendo “artefactos” a nuestro fichero pom.xml:

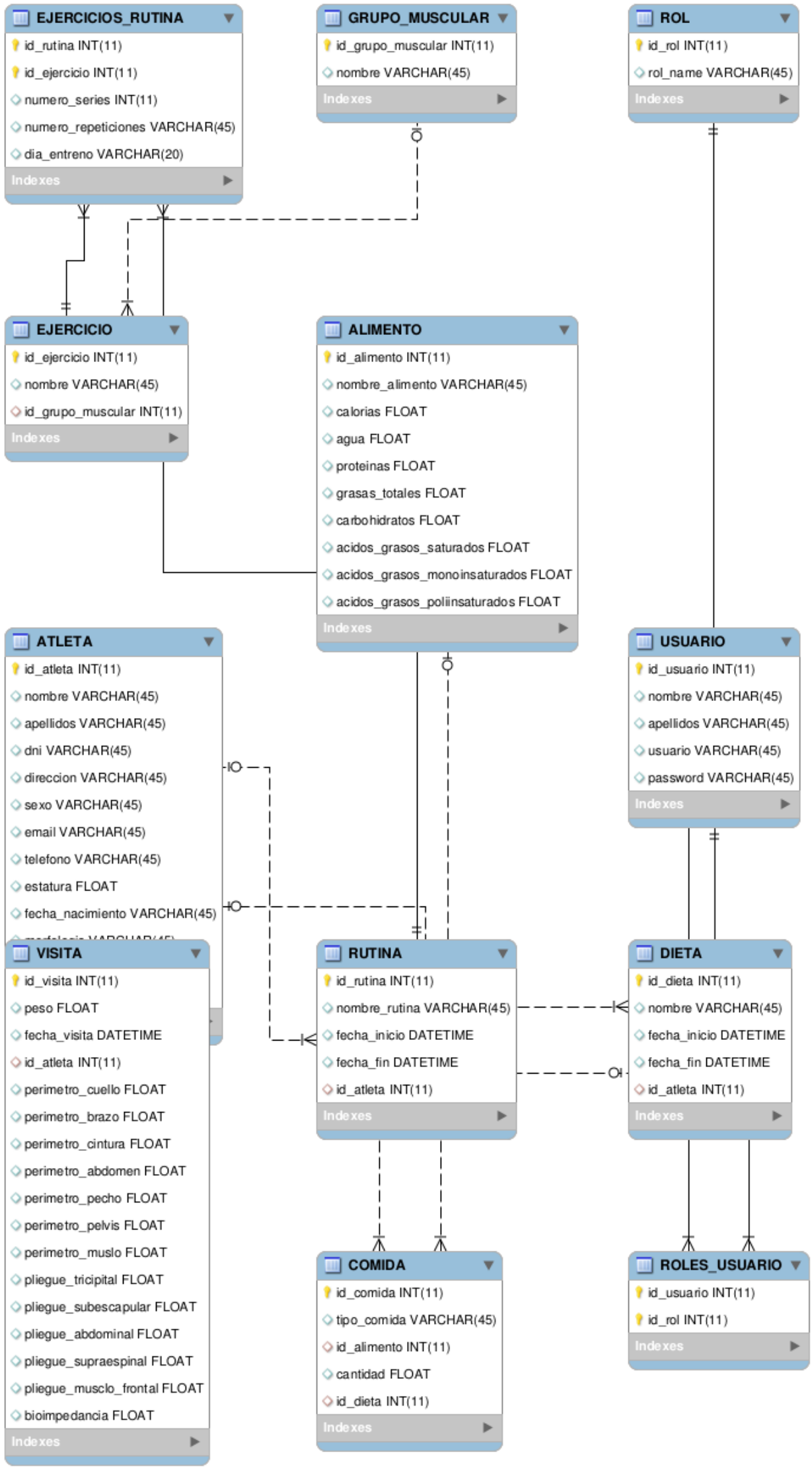
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

 <modelVersion>4.0.0</modelVersion>
 <groupId>com.genbetadev.proyecto1</groupId>
 <artifactId>proyecto1</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>
 <dependencies>

 <dependency>
 <groupId>log4j</groupId>
 <artifactId>log4j</artifactId>
 <version>1.2.17</version>
 </dependency>

</dependencies>
</project>
```

Evidentemente maven tiene un sin fin de funcionalidades más, pero no es alcance de este documento hablar de ellas.

i-18N: a pesar de no ser un requerimiento funcional necesario por ningún stakeholder, se ha decido implementar toda la aplicación en modo multi-idioma. En un principio la aplicación está traducida a dos idiomas distintos, español e inglés, pudiendo ampliarse en un futuro y de una manera muy sencilla a tantos idiomas como se necesiten. Creo que es una funcionalidad muy interesante y que puede ayudar al proyecto a diferenciarse del resto y así obtener una mejor nota.

DISEÑO

A nivel de diseño de aplicación, comentado en el apartado anterior las modificaciones pertinentes relacionadas con la arquitectura, sólo queda remarcar los cambios realizados a nivel de base de datos. En la versión final se han afinado muchos los detalles en este apartado, con nuevas tablas y relaciones, a continuación muestro el diagrama entidad-relación final donde se pueden apreciar los cambios con respecto a la versión inicial.


EJERCICIOS_RUTINA
id_rutina INT(11)
id_ejercicio INT(11)
numero_series INT(11)
numero_repeticiones VARCHAR(45)
dia_entreno VARCHAR(20)
Indexes
GRUPO_MUSCULAR
id_grupo_muscular INT(11)
nombre VARCHAR(45)
Indexes
ROL
id_rol INT(11)
rol_name VARCHAR(45)
Indexes
EJERCICIO
id_ejercicio INT(11)
nombre VARCHAR(45)
id_grupo_muscular INT(11)
Indexes
ALIMENTO
id_alimento INT(11)
nombre_alimento VARCHAR(45)
calorias FLOAT
agua FLOAT
proteinas FLOAT
grasas_totales FLOAT
carbohidratos FLOAT
acidos_grasos_saturados FLOAT
acidos_grasos_monoinsaturados FLOAT
acidos_grasos_poliinsaturados FLOAT
Indexes
ATLETA
id_atleta INT(11)
nombre VARCHAR(45)
apellidos VARCHAR(45)
dni VARCHAR(45)
direccion VARCHAR(45)
sexo VARCHAR(45)
email VARCHAR(45)
telefono VARCHAR(45)
estatura FLOAT
fecha_nacimiento VARCHAR(45)
USUARIO
id_usuario INT(11)
nombre VARCHAR(45)
apellidos VARCHAR(45)
usuario VARCHAR(45)
password VARCHAR(45)
Indexes
VISITA
id_visita INT(11)
peso FLOAT
fecha_visita DATETIME
id_atleta INT(11)
perimetro_cuello FLOAT
perimetro_brazo FLOAT
perimetro_cintura FLOAT
perimetro_abdomen FLOAT
perimetro_pecho FLOAT
perimetro_pelvis FLOAT
perimetro_muslo FLOAT
pliegue_tricipital FLOAT
pliegue_subescapular FLOAT
pliegue_abdominal FLOAT
pliegue_supraespinal FLOAT
pliegue_musclo_frontal FLOAT
bioimpedancia FLOAT
Indexes
RUTINA
id_rutina INT(11)
nombre_rutina VARCHAR(45)
fecha_inicio DATETIME
fecha_fin DATETIME
id_atleta INT(11)
Indexes
DIETA
id_dieta INT(11)
nombre VARCHAR(45)
fecha_inicio DATETIME
fecha_fin DATETIME
id_atleta INT(11)
Indexes
COMIDA
id_comida INT(11)
tipo_comida VARCHAR(45)
id_alimento INT(11)
cantidad FLOAT
id_dieta INT(11)
Indexes
ROLES_USUARIO
id_usuario INT(11)
id_rol INT(11)
Indexes

ANALISIS

A nivel de análisis de la aplicación, en esta versión final se han añadido algunas funcionalidades nuevas relacionadas con la gestión de los atletas. En el apartado de “ficha de atleta”,además de la funcionalidad comentada de “añadir entrada” para introducir las medidas corporales de los atletas en cada visita, se han añadido varias funcionalidades nuevas que comento a continuación:

CASO DE USO: LISTAR DIETAS	
Resumen de funcionalidad	Muestra un listado con las dietas históricas del cliente seleccionado
Actores	Entrenador personal
Casos de uso relacionados	
Precondición	Ninguna al ser una consulta
Flujo normal	1. El entrenador personal entra en el modulo atletas. 2. Aparece una lista con todos los atletas ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El Entrenador hace click en el cliente que lo desee para ver su ficha completa. 5. El entrenador hace click en el apartado “Dietas”
Flujo alternativo	4. El Entrenador puede hacer click en el botón de “Añadir cliente” (Caso de uso Alta de cliente)
Postcondición	Ninguna

CASO DE USO: AÑADIR DIETA	
Resumen de funcionalidad	Añadir una nueva dieta a un cliente
Actores	Entrenador

Casos de uso relacionados	Listar entradas; editar cliente, consultar cliente
Precondición	El cliente existe previamente en el sistema
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el cliente que lo desee para ver su ficha completa. 5. El entrenador hace click en la pestaña dietas 6. El entrenador puede hacer click en el botón añadir dieta. 7. El entrenador introduce los datos de la nueva dieta. 8. El sistema comprueba si todos los datos son correctos 9. La nueva entrada se añade al cliente
Flujo alternativo	7. Si el sistema detecta algún error lo muestra por pantalla al usuario
Postcondición	Una nueva dieta se ha creado en el sistema

CASO DE USO: EDITAR DIETA	
Resumen de funcionalidad	Poder editar una dieta histórica de un cliente
Actores	Entrenador personal
Casos de uso relacionados	Listar entradas, consultar cliente
Precondición	El cliente existe en el sistema
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los

	clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el cliente que lo desee para ver su ficha completa. 5. El entrenador hace click en la pestaña dietas 6. El entrenador puede hacer click en la entrada que lo desee. 7. El sistema devuelve una pantalla con los datos de esa dieta 8. El entrenador modifica los datos de esa dieta 9. El sistema comprueba que no hay errores 10. El entrenador pulsa guardar y los datos se guardan en el sistema
Flujo alternativo	5. El entrenador puede cancelar o cerrar la ventana del cliente 7. El entrenador cierra la ventana de la entrada
Postcondición	La dieta se actualiza correctamente

CASO DE USO: LISTAR ENTRENAMIENTOS	
Resumen de funcionalidad	Muestra un listado con los entrenamientos históricos del cliente seleccionado
Actores	Entrenador personal
Casos de uso relacionados	
Precondición	Ninguna al ser una consulta
Flujo normal	1. El entrenador personal entra en el modulo atletas. 2. Aparece una lista con todos los atletas ordenada por apellidos

	3. El listado permite filtrar los resultados por varios campos. 4. El Entrenador hace click en el cliente que lo desee para ver su ficha completa. 5. El entrenador hace click en el apartado “Entrenamientos”
Flujo alternativo	4. El Entrenador puede hacer click en el botón de “Añadir cliente” (Caso de uso Alta de cliente)
Postcondición	Ninguna

CASO DE USO: AÑADIR ENTRENAMIENTO	
Resumen de funcionalidad	Añadir un nuevo entrenamiento a un cliente
Actores	Entrenador
Casos de uso relacionados	Listar entrenamientos; editar cliente, consultar cliente
Precondición	El cliente existe previamente en el sistema
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el cliente que lo desee para ver su ficha completa. 5. El entrenador hace click en la pestaña entrenamientos 6. El entrenador puede hacer click en el botón añadir entrenamiento. 7. El entrenador introduce los datos del nuevo entrenamiento. 8. El sistema comprueba si todos los datos son correctos 9. El nuevo entrenamiento se añade al cliente

Flujo alternativo	7. Si el sistema detecta algún error lo muestra por pantalla al usuario
Postcondición	Un nuevo entrenamiento se ha creado en el sistema

CASO DE USO: EDITAR ENTRENAMIENTO	
Resumen de funcionalidad	Poder editar un entrenamiento histórico de un cliente
Actores	Entrenador personal
Casos de uso relacionados	Listar entrenamiento, consultar cliente
Precondición	El cliente existe en el sistema
Flujo normal	1. El entrenador personal entra en el modulo clientes. 2. Aparece una lista con todos los clientes ordenada por apellidos 3. El listado permite filtrar los resultados por varios campos. 4. El entrenador puede hacer click en el cliente que lo desee para ver su ficha completa. 5. El entrenador hace click en la pestaña entRenamientos 6. El entrenador puede hacer click en la entrada que lo desee. 7. El sistema devuelve una pantalla con los datos de ese entrenamiento 8. El entrenador modifica los datos de ese entrenamiento 9. El sistema comprueba que no hay errores 10. El entrenador pulsa guardar y los datos se guardan en el sistema
Flujo alternativo	5. El entrenador puede cancelar o cerrar la ventana del cliente 7. El entrenador cierra la ventana de la

	entrada
Postcondición	El entrenamiento se actualiza correctamente

CONCLUSIONES

Con el presente trabajo fin de carrera he podido cumplir uno de mis sueños y es llevar a cabo mi primero gran aplicación. Sobretodo me ha gusto poder aplicar mis conocimiento a algo no de ámbito académico y de carácter personal. He podido disfrutar mucho mientras daba vueltas a las cosas de cómo se haría esta o aquella funcionalidad, pero la realidad es que finalmente lo he conseguido. Sin pretender quedar de pedante, estoy muy contento del trabajo realizado, creo que esta es la demostración a mi mismo de que cualquier cosa que imagine podría llegar a plasmarla en una aplicación.