Universidad de Valladolid

E.U. DE INFORMÁTICA (SEGOVIA)

Ingeniería Técnica en Informática de Gestión

Gestión Integral de Clubes Deportivos.

Alumno: Diego Cebrián Martínez

Tutor: Juan Antonio Velasco Mate

Índice de contenido

1. Introducción

Descripción de las diferentes ideas que dan lugar a la realización de este proyecto, ideas iniciales, evoluciones y posibilidades de la aplicación.

1.1 Idea inicial

La idea inicial de este proyecto nace de la necesidad detectada por el alumno en su entrenamiento diario del deporte del atletismo. Esta necesidad viene de la carencia de herramientas de comunicación y gestión inteligente de la información recabada por el entrenador y ofrecida por el deportista. También existen dificultades que existen cuando se trata de gestionar económicamente un club deportivo con un programa de gestión comercial generalista. El programa debería tener la capacidad de controlar y gestionar todos los entrenamientos de los deportistas usando para ello la metodología de cada entrenador. También debería ser capaz de gestionar las cuentas propiamente dichas, el activo de socios, deportistas, instalaciones, material deportivo, seguros médicos, organización de competiciones propias, desplazamientos a competiciones, publicación de resultados en la web y prensa, y otros detalles que seguro que se escapan. Además debería proporcionar métodos para poder incluir módulos que amplíen las funciones del programa de una forma relativamente fácil. Las necesidades iniciales se atribuyen a un club de atletismo pero este programa debería ser perfectamente válido para la gestión de cualquier club deportivo, ya sea este de fútbol o de tenis de mesa, haciendo el uso correcto de las extensiones o del API.

1.2 Estructura de los clubes

En este programa nos vamos a ocupar meramente de la gestión deportiva y de los flujos de trabajo que relacionan a los sujetos entrenados con sus entrenadores. Si bien se hace esta pequeña introducción a la estructura interna de los clubes teniendo en cuenta la posibilidad de extensión del programa. La relación entre entrenador y entrenado es una de las bases del organigrama deportivo y se podría construir una aplicación más completa partiendo de esta base.

Como hemos podido comprobar en el punto anterior las áreas de gestión de un club son muy amplias. Probablemente a los deportistas no les importen muchos aspectos económicos, y a los entrenadores no les interese conocer los entresijos de la organización de una prueba. Así pues, cada persona que forme parte del club deberá tener acceso a unos datos diferentes. Más adelante estudiaremos la manera de conseguirlo, pero de lo que si que podemos hablar es de una diferenciación entre diferentes estamentos de la organización, con sus permisos de visualización, edición, borrado y adición de datos a la aplicación.

En la estructura de todo club deportivo hay muchas relaciones internas, flujos de trabajo y económicos. Estos procesos se han de representar correctamente. Cada persona asociada a un proceso tendrá que recibir toda la información necesaria para llevar a cabo su trabajo, así como permitir introducir los datos que genere ese trabajo. La información que reciba deberá ser debidamente filtrada para que el exceso de información no dificulte las tareas y se completen lo más pronto posible.

Muchas de las figuras que trabajan en club deportivo tienen acceso a datos confidenciales de otras. Los entrenadores a datos médicos de los deportistas, el tesorero a datos bancarios, el club puede contar con médicos propios o fisioterapeutas. El programa debe cuidar también de conservar esta confidencialidad, e impedir que puedan producirse filtraciones. Se estudiará la normativa española de obligado cumplimiento en protección de datos, así como los diferentes medios digitales para salvaguardar dicha ley en el funcionamiento del software desarrollado.

1.3 Ampliación de funciones

La gestión de un club deportivo se puede extrapolar a otras organizaciones que tienen que

ver con el deporte sin ser necesariamente clubes. Estos otros posibles destinatarios objetivos de la aplicación son los gimnasios, empresas organizadoras de eventos deportivos, ayuntamientos, empresas de gestión de recintos deportivos. Al tener la idea de ser un programa completamente modular, cualquier otra entidad que pueda necesitar de un software especializado podría beneficiarse de las características.

1.4 Funcionamiento de la aplicación

Para este proyecto se ha pensado en la relación entre el entrenador y el deportista a título individual, en el método de entrenamiento planificado y en la posibilidad de aprovechamiento de esta plataforma para dar soporte a varios entrenadores sin que la información almacenada esté disponible para los demás puesto que, salvo que el entrenador quiera publicar libremente sus entrenamientos y resultados de sus atletas, el resultado de este trabajo es secreto. De esta manera en un mismo club los diferentes entrenadores de distintos equipos podrían tener una plataforma de trabajo común.

Cuando un deportista en un deporte individual o colectivo termina el entrenamiento lo normal es que regrese a su casa y ya allí, tranquilamente, haga el análisis de su trabajo. Además el entrenador en el campo de trabajo suele tener los medios ni las capacidades para retener toda la información que le ofrecen sus pupilos. Esta condición hace que la aplicación deba poder ser ejecutada desde muchos lugares diferentes, como las diferentes casas de cada deportista, a pesar de tener los datos centralizados en un solo lugar. Tampoco conocemos los sistemas operativos utilizados por los diferentes posibles clientes en sus instalaciones. Es por eso que el programa deberá funcionar, al menos en cliente, en casi cualquier sistema operativo, así que deberá ser un sistema multiplataforma, ya sea a través de web con navegador, o utilizando tecnologías multiplataforma propiamente dichas, como entornos de desarrollo Qt, programación en .NET (con soporte para plataformas no windows con Mono), o la opción más conocida del lenguaje Java, que funciona en casi todo tipo de dispositivos, desde ordenadores con diferentes sistemas operativos, Linux, Solaris, Windows, MacOS y dispositivos móviles.

2. Análisis del sistema

Estudio de las necesidades del proyecto.

2.1 Ámbito de la aplicación

En nuestra sociedad española los valores han cambiado mucho en los últimos 30 años. Hoy en día el deporte es visto como algo accesible a todos los públicos. La proliferación de instalaciones deportivas en todas las ciudades y en los más pequeños municipios indican que la práctica del deporte ha pasado a formar parte del común de la sociedad. Debido a esto los gestores de estas instalaciones han tenido que buscar los recursos adecuados para gestionar el buen uso y la organización de las mismas.

Asociado a este gran crecimiento de la práctica del deporte también se ha generado la necesidad de agruparse, además de en el deporte de equipo donde esto es obvio, para obtener ventajas o compartir costes en la instalaciones. Así se han generado federaciones, clubes y asociaciones relacionadas con las prácticas deportivas.

También ha crecido en un gran porcentaje el tiempo dedicado al deporte en los medios de comunicación, de manera que la gente se siente más cercana al deporte. Muchas personas toman como pasatiempo la práctica de diferentes modalidades deportivas esporádicas debido a esta influencia de los medios. En torno a esta nueva necesidad han surgido empresas que se dedican a gestionar actividades deportivas, organizarlas, o que han creado instalaciones vacacionales basadas en el ejercicio físico.

Las empresas, como organizaciones dedicadas a hacer dinero a cambio de servicios, han realizado diferentes sistemas para el control de muchos de estos aspectos, centrándose sobre todo en la gestión de instalaciones, generación de carnés de socios y almacenamiento de unos pocos datos como control de accesos y uso de máquinas. Estos usos de las aplicaciones solo tienen una vertiente económica y no entran a valorar más allá de si las personas tienen derecho a usa la máquina una vez más o si han venido 30 veces al mes a la instalación. Ignoran por completo otros aspectos como los que se refieren a la gestión de los entrenamientos o las comunicaciones entre el club y sus deportistas.

Gesport intentará cubrir ese hueco dejado en los ERP, enterprise resource planning o sistemas de planificación de recursos empresariales, o crear un nuevo concepto de programa: sistema de planificación de recursos deportivos, siempre partiendo de la base de la gestión de entrenamientos.

2.1.1 Gestión de asociaciones y clubes deportivos

Las asociaciones y clubes deportivos son entidades sin ánimo de lucro que tienen entre sus objetivos el facilitar la práctica deportiva a sus socios o afiliados. Son el objetivo principal del mercado del proyecto. Las necesidades de estas organizaciones pueden ser muy diferentes dependiendo de su tamaño y masa social. En los últimos años la industria del ocio ha crecido mucho. Cada vez son más los clubes de golf que inundan nuestros campos y en ellos no solo se realiza la práctica de ese deporte, sino que incluyen servicios de gimnasio, tenis, entrenamientos personales. La calidad de vida viene ligada al ejercicio deportivo regular y en todos los ámbitos sociales se practica en la medida de lo posible.

Los clubes deportivos pueden dedicarse tanto a una labor de socialización del deporte, como pueden ser los relativos a golf o tenis, a la competición, o a la formación de nuevos deportistas de categorías inferiores. Además pueden cumplir varios de estos objetivos en un mismo club. Aunque los a los clubes deportivos se les suponga entidades sin ánimo de lucro son numerosos los casos en los que la rentabilidad es importante.

Cuentan con una junta directiva que toma decisiones en los aspectos económicos, deportivos generales, organizativos y de estructura de club. La información que llega a esas juntas y la que sale de ella debe ser correspondientemente filtrada. Esta información y la toma de decisiones se debe poder aplicar y consultar en cualquier momento.

En los clubes existen equipos de profesionales, desde entrenadores, hasta médicos, fisioterapeutas, psicólogos y los propios deportistas en el caso de que compita en círculos de élite. Cada uno de estos profesionales tiene que tener acceso a todos los datos necesarios para realizar su trabajo, así como la posibilidad de comunicarse entre ellos, de manera que los masajistas o fisioterapeutas tengan claro cuando viene mejor un masaje de descarga o relajación, o ejercicios de fortalecimiento, o cuando los médicos tienen que realizar mayor aporte de vitaminas y estimulantes del sistema inmunológico, dependiendo de las cargas de entrenamiento, descansos y competiciones.

En el aspecto logístico los problemas que puede solucionar la aplicación pueden venir dados por la organización de material de entrenamiento y competición, transportes internos y externos, y localización y repartición de los almacenes para ese control.

2.1.2 Gestión de organizadores de eventos deportivos

En la práctica del deporte existen eventos, ya sean competiciones o exhibiciones, donde se muestra a un público como se realiza la actividad. Si bien los clubes desarrollan esta actividad por si mismos, también es cierto que existen empresas, federaciones y asociaciones que organizan este tipo de acontecimientos.

Gesport deberá cumplir con esta necesidad y facilitar las gestiones, organizaciones de personal, arbitraje, seguridad, material necesario, premios, publicidad de patrocinadores de la prueba y de la propia prueba.

2.1.3 Gestión de empresas de que ocio y deporte

Las empresas que se dedican al mundo del deporte y ocio (gimnasios, piscinas, escuelas de danza, etc...) tienen que tener la misma capacidad de organización de un club deportivo con el añadido de la posibilidad de facturación, control de tiempos de accesos a instalaciones, pagos de trabajadores o gestión de otros productos que puedan tener en tienda, como complementos alimenticios o máquinas para el trabajo personal en casa.

2.2 Objetivos del proyecto

El objetivo de este proyecto es crear un aplicación modular y extensible que posibilite la gestión de los aspectos deportivos y que permita añadir posteriormente los logísticos, económicos, sociales, legales y organizativos de cualquier organización, ya sea empresa, asociación o club, que tenga que ver con el deporte.

Se trata también de que la gestión de estos recursos sea eficiente y provea las herramientas necesarias de conocimiento de todas las áreas del club a los diferentes usuarios, cada uno en su medida, de manera que tengan un acceso fácil y sencillo a cualquier información de su incumbencia.

Deberá ser de fácil manejo, pues al usuario final no se le presuponen capacidades del manejo de la informática más allá de una alfabetización básica.

Tendrá que ser multiplataforma, pues no se puede asegurar que todos los componentes de una organización tendrán el mismo tipo de ordenador, y menos cuando se piensa en esta aplicación

para múltiples organizaciones.

Estará obligado por ley a guardar la privacidad de los datos de los usuarios en los niveles que así lo requiera, tendrá que cumplir con la Ley orgánica de Protección de datos y los usuarios podrán borrar o rectificar los datos personales contenidos en los ficheros del programa. Deberá permitir la importación o exportación de firmas digitales. La codificación de datos para permitir la lectura de los mismos por las personas adecuadas será asimétrica.

Deberá presentar la información justa para la toma de decisiones en cada momento. Los datos deberán poder simplificarse en resúmenes, gráficos o esquemas.

2.3 El lenguaje de modelado unificado (UML)

El lenguaje de modelado unificado es un lenguaje gráfico que se usa principalmente para visualizar, especificar, construir y documentar componentes de software y sistemas de software. El UML no es de uso exclusivo del software ya que con el mismo lenguaje se pueden documentar gráficamente todo tipo de sistemas, desde empresariales hasta de cualquier ingeniería. El UML permite el modelado del sistema completo como modelo en diferentes diagramas, que pueden ser de diferentes tipos. Existen otras técnicas de ingeniería del software, como el método Jackson basado en las estructura de los datos, pero se ha elegido esta por la simplicidad que tiene el representar a los diferentes elementos que componen la organización real con objetos de software de parecida estructura interna.

El UML no interviene solo en la fase de diseño de software, a la hora de establecer la clases, objetos, relaciones y contenidos, sino que a través de los diferentes tipos de diagramas también se usa en el análisis, con los diagramas de casos de uso, y en la implementación, si el software usado es el apropiado permite la generación de las clases para facilitar la tarea del programados. Aunque este último caso no sea propiamente UML hoy en día casi todas las herramientas de ingeniera de software propietarias o libres proveen esta característica por defecto.

- Los tipos de diagramas que componen el modelo software son los siguientes:

- Diagramas de Casos de Uso, donde se describen a los diferentes actores y casos de uso, funcionalidades muy concretas del sistema que pueden formar parte de un proceso más largo, que entran en contacto con el sistema modelado y que mensajes se transmiten entre ellos.

- Diagramas de colaboración y de secuencia, que definen las relaciones entre los distintos elementos del programa y los actores. Con esto quiero decir que establecen de manera formal a que llamadas de los usuarios van a responder y como los diferentes módulos del programa. En este caso interviene el factor temporal y cada caso de uso debería tener una secuencia para su realización. Hasta este punto estamos trabajando en el ámbito del análisis, a partir de aquí se trata el diseño del modelo.

- Diagramas de clases, que consisten en la representación de los elementos de la organización como clases, conjuntos de estructuras de datos con instrucciones para su manejo interno.

Se trabaja con diferentes diagramas según sean necesarios. Puede que en un modelo no haya diagrama de casos de usos (algunas referencias bibliográficas desaconsejan su uso para el modelado de software orientado a objetos), y puede que tenga múltiples diagramas de un mismo tipo. Para que el modelo sea correcto todos los diagramas de un mismo tipo podrían transformarse en uno solo que sería el diagrama del modelo. En este diagrama no habría funciones de clases ni clases duplicadas.

Con un buen modelado en todos los aspectos tendremos una gran base para construir un

programa sólido y coherente tanto con las necesidades de los usuarios, como con una estructura interna fácil de mantener y ampliar.

2.4 Metodología de análisis

Para la realización del análisis de las posibles necesidades organizativas, técnicas y económicas se han realizado una serie con Pedro Lorenzo. Actualmente es el Director Técnico del Club Deportivo Atlético Joaquín Blume - Caja-Segovia. Es Licenciado en Ciencias de la Actividad Física y del Deporte por la Universidad Europea de Madrid con la especialización en Atletismo y Golf, Entrenador Nacional de Atletismo por la Real Federación Española de Atletismo y monitor de diferentes disciplinas de ocio y deportivas como Natación, Tiempo Libre, Piragüismo y Polideportivo. Además ha realizado cursos de especialización en Dietética y Nutrición, Gestión de Instalaciones Deportivas, Esquí, Fútbol, Fútbol Sala, Aquatic Fitness y Fisiología del Ejercicio y ha publicado artículos en diferentes revistas especializadas. Por todo ello, y por su dilatada experiencia laboral en diferentes clubes deportivos como el Real Club de Golf Puerta de Hierro, creo que es una persona apropiada y que conoce suficientemente los entresijos de este dominio que vamos a abarcar en el proyecto.

Las entrevistas se realizaron en tres fases diferentes.

En la primera se definieron las diferentes necesidades del personal técnico y deportistas a nivel deportivo. En esta fase se definieron los diferentes documentos que intervienen en la planificación deportiva, sus fases y periodos, los datos necesarios en cada periodización y la información que el sujeto debe transmitir al entrenador en forma de realimentación para que este pueda adaptar los futuros entrenamientos. También se estableció el glosario de términos deportivos para aclarar las definiciones que se iban a tratar durante el desarrollo de la aplicación y se tradujo al inglés para hacer el código fuente de la aplicación accesible y extensible a cualquier programador del mundo.

Una vez terminada esta fase se comenzó a hacer el análisis mediante diagramas de casos de uso, y, en colaboración con el "cliente" (Pedro Lorenzo), se encontraron los errores de concepto mediante la técnica de tarjetas CRC y la corrección de los primeros diagramas de clases básicos realizados.

Tras estas entrevistas se seguirá una metodología de análisis orientada a objetos. La primera parte consistirá en una especificación de requisitos, donde se van a recoger todos los requisitos funcionales y no funcionales que va tener la aplicación. Estos requisitos se reflejarán en diagramas de casos de uso. De esta manera al finalizar el proyecto podremos comprobar que todas las necesidades marcadas en un principio se están cumpliendo.

Sobre estos requisitos se estudiarán las clases y objetos que van a intervenir en ella. A este proceso se le llama identificación de las clases del dominio. Para este proceso se utilizará la técnica de la identificación de sustantivos. Consiste en extraer los nombres y grupos nominales de la descripción del problema y las entrevistas con los usuarios, expertos en el dominio y clientes. Sobre estos nombres obtenidos se realizará una criba donde se eliminaran los que estén duplicados en significado o que representen lo mismo, los que no intervengan realmente como elementos del problema a resolver, los elementos propios del sistema software que vaya a tratar el problema, pantalla, teclado, etcétera, y los que indiquen frecuencia temporal. Además de estos que se eliminan otros tendrán que transformarse pues pueden ser atributos de una clase, en vez de una clase propiamente dicha, representar una acción, que en el sistema se convertirán en métodos, o un estado, que será un valor de un atributo.

2.5 Glosario de términos deportivos

Durante el transcurso de las entrevistas surgieron términos técnicos del área del deporte y de la salud que vamos a definir en este punto para que el resto del texto de análisis y posterior diseño y desarrollo se entiendan de la mejor manera posible. También se han traducido estos conceptos al inglés para su posterior inclusión en los diagramas de clases.

Temporada (Season): Periodo determinado de tiempo, generalmente de 9 meses a 1 año, sin necesidad de que sea año natural. En dicho periodo se centra la actividad principal de entrenamientos y competiciones de cada deporte.
Sesión (Session): La sesión es la estructura elemental de todo proceso de entrenamiento. La planificación anual o plurianual de un deportista se basa en la combinación adecuada de las diferentes sesiones.
Microciclo (Microcycle): Es un conjunto de sesiones planificadas en un corto periodo de tiempo.
Mesociclo (Mesocycle): Es un conjunto de Microciclos.
Macrociclo (Macrocycle): Es un conjunto de Mesociclos. Cada uno de estos ciclos temporales anteriores tiene su razón de ser en el desarrollo de cualidades concretas, en la consecución de objetivos concretos y en las fases competitivas que hay a lo largo de la temporada.
Objetivos (Objective): Son las metas marcadas para conseguir en un determinado tiempo, que puede ser, un microciclo, macrociclo, temporada o periodo plurianual. Pueden ser objetivos principales, secundarios, operativos en función de la meta que busquemos. Siempre tienen que seguir una progresión lógica y asequible.
Carga : La carga constituye la categoría central del entrenamiento, se realiza a través de ejercicios físicos, con una determinada finalidad. El entrenamiento deportivo se basa en la aplicación de cargas de trabajo en las sesiones de entrenamiento y competiciones. Se puede entender como medida fisiológica del organismo, provocada por un esfuerzo físico, con una cierta duración y profundidad (volumen / intensidad).
Periodización: La periodización consiste en distribuir las cargas de forma lógica y progresiva, a lo largo de una temporada, o un periodo determinado de tiempo, en función de los objetivos marcados y propuestos para cada deportista, para conseguir una mayor adaptación y progresión deportiva.
Ejercicio:
Intensidad (Intensity): Es el porcentaje de trabajo que se aplica durante un esfuerzo* (fza maxima, VO2 max, Velocidad max)
Test: Es una prueba o valoración de cualquiera de las capacidades físicas básicas de un deportista.
Tipos de trabajo (fuerza resistencia, anaeróbico, aeróbico)
Consumo máximo de oxigeno (VO2max): También llamado VO2 max., es la cantidad de oxigeno consumida por el cuerpo en condiciones de máximo esfuerzo. Es uno de los mejores (por no decir el mejor) indicadores de la competencia física.
Marca : Periódico deportivo de mayor tirada en el territorio nacional, y que se caracteriza por su falta de criterio en lo que a temas deportivos se refiere, además de llenar sus

paginas con falacias sin ninguna fuente verídica y con fundamento. Resultado obtenido por un deportista en cualquier prueba, test, competición, etcétera. A lo largo de este documento usaremos sin excepción la segunda acepción.
Espirometría: Es un método de diagnostico que estudia la función pulmonar en relación a su capacidad funcional y de reserva (cuánto aire se puede introducir en los pulmones y la rapidez con la que se puede expulsar). En la espirometría se evalúa el coeficiente VEF/CV (Volumen espiratorio forzado en el 1º segundo/ capacidad vital).
Test de Esfuerzo: También llamado Prueba de esfuerzo, ECG de ejercicio, etc. Es una herramienta de examen general para evaluar los efectos del ejercicio en el corazón. Durante la prueba se registra la actividad eléctrica del corazón, mientras la persona camina (también se puede hacer con carrera si hablamos de sujetos entrenados) en una cinta sin fin, o pedalea en una bicicleta estática. Se mide la reacción del corazón a una mayor demanda de oxigeno por parte del organismo, provocado por el progresivo incremento de la intensidad del ejercicio.
Monitor de pulso cardiaco o monitor (HRM): 1. Aparato que mide y almacena los datos de la frecuencia cardiaca del deportista que lo porta durante el entrenamiento. En la actualidad aparte de la frecuencia cardiaca media y máxima hay modelos capaces de almacenar el pulso en periodos muy cortos de tiempo, permitiendo la visualización de curvas y gráficas con la evolución del pulso. También existen los que miden la distancia recorrida, la diferencia de altura sobre el nivel del mar e incluso trazan las rutas entrenadas por GPS. 2. Persona encargada de la iniciación deportiva en cualquier categoría. En la iniciación se trata de desarrollar las características físicas básicas sin entrar en la planificación del deportista para la competición. Durante toda esta memoria se tratará de expresar siempre el monitor de visualización de un ordenador como "pantalla" para no dar lugar a equívocos con los anteriormente definidos.

Tabla 1: Glosario de términos deportivos

2.6 Análisis de Requisitos

2.6.1 Requisitos Funcionales

En esta sección vamos a describir las necesidades que tiene el cliente. El primer detalle a destacar es que será un software que podrán usar todos los miembros de la organización. Cada uno tendrá acceso a realizar diferentes tareas de cada área según sea su interés real en las mismas. Los diferentes funcionamientos de cada área vienen dados por las entrevistas realizadas al cliente.

2.6.1.1 Gestión de entrenamientos

El programa debe soportar diferentes tipos de personas encargadas de la gestión deportiva. Para ello tendremos que tener una base, los datos personales. Cada una de estas personas tendrá acceso a ver, añadir, modificar y borrar ciertos datos de las otras personas que intervienen en la gestión. En la gestión de entrenamientos intervienen unos siguientes documentos, que pertenecen a las diferentes personas y que solo ellas podrán consultar. Estos documentos son la esencia del entrenamiento deportivo.

Documentos necesarios para la correcta gestión de entrenamientos:

1. Documento de datos personales. Donde se definen los datos personales: nombre, apellidos, teléfono, direcciones, fecha de nacimiento, correo electrónico.

2. Documento de datos del deportista. Donde se recoge el deporte que practica, el grupo con el que entrena, el equipo donde milita (para casos en los que haya diferentes equipos dentro de un mismo club: canteras, categorías inferiores, diferentes deportes) y el entrenador.

3. Documento de objetivos a conseguir durante la temporada presente.

4. Documento de periodización de la temporada. Donde se describen por semanas los diferentes aspectos a trabajar durante la misma, así como el número de kilómetros, kilos o repeticiones trabajados en cada semana. También está reflejado en este documento la progresión que tiene el deportista a la hora de realizar los diferentes ejercicios y un .

5. Documento de progresión. Es el documento que describe los ejercicios que hay que hacer dependiendo del aspecto a trabajar y la intensidad deseada para el mismo.

6. Documento de programación. Es el que nos describe los ejercicios a realizar en cada día de la semana. A través de este documento se produce la realimentación hacia el entrenador, pues tiene campos abiertos para que el sujeto entrenado describa la realización del ejercicio. En este documento también aparecen los gráficos de las zonas de trabajo físico realizadas: carga (como volumen e intensidad), indicaciones de zonas cardiacas de trabajo y otras estadísticas de utilidad para el profesional del deporte.

7. Documento de testings. Es un documento donde se acumulan los diferentes test físicos, como realizarlos y las condiciones necesarias.

8. Documento de marcas. Contienen todas las marcas y resultados de la vida deportiva del deportista entrenado, así como los resultados de los test.

9. Documento de datos médicos. Contiene informes médicos de los diferentes sujetos entrenados. Los informes vendrán dados por la necesidad del entrenador de cierto conocimiento a la hora de planificar la temporada. Así se utilizarán plantillas de espirometría, analítica completa, test de esfuerzo, consumo máximo de oxigeno. Al ser estos datos médicos y de carácter personal será el atleta quien decida para quien estarán disponibles y el entrenador no tiene porque no verlos si así lo desea el deportista. Además de los documentos citados podrían añadirse otros documentos médicos.

10. Documento de tablas de ejercicios. Tiene la información de diferentes ejercicios que van a realizarse conjuntamente en una sesión para realizar un trabajo de un tipo concreto.

En la organización, o club habrá varios entrenadores que entrenarán a cada uno de sus deportistas. Existen documentos propios por entrenador que son iguales independientemente del atleta entrenado: 5. Documento de progresión , 7. Documento de Testings y 10. Documento de tablas de ejercicios. Existen documentos que tiene cada atleta personalizado: 1. Documento de datos personales, 2. Documento de datos del deportista, 3. Documento de objetivos, 4. Documento de periodización, 6. Documento de Programación , 8. Documento de marcas y 9. Documento de datos médicos.

Estos últimos datos, al ser de carácter personal médico del deportista, tendrían una visibilidad exclusiva para el propietario y necesitarían de un permiso expreso establecido para que pudieran ser vistos por otras personas, como por ejemplo el entrenador.

Vamos a proceder a describir las informaciones completas que contienen estos documentos:

2.6.1.1.1 Documento de datos personales:

Al crear un nuevo deportista, socio, o trabajador se generará este documento que contiene todos los datos como NIF, nombre, apellidos, dirección, teléfono, número de cuenta bancaria y el tipo o tipos de papeles que representa en el club (socio, entrenador, deportista, trabajador, director técnico, presidente, secretario, vicepresidente, tesorero, vocal).

Los entrenadores podrán añadir o editar personas de tipo deportista, el director técnico a entrenadores y trabajadores del club, aparte de deportistas, el secretario a los socios y cambiar el estatus de cada uno de ellos según las actas de la junta directiva, el tesorero o los organizadores de pruebas a los proveedores.

2.6.1.1.2 Documento de datos del deportista:

En este documento se describe el grupo al que pertenece, el deportista, su entrenador. El propio deportista podrá cambiar partes de su documento.

2.6.1.1.3 Documentos de objetivos:

Un documento de objetivos por temporada. Contiene un listado de objetivos principales, uno de objetivos operativos y uno de objetivos secundarios. El entrenador define el periodo que dura la temporada. Puede añadir, editar y borrar el documento de objetivos que tiene cada atleta para esa temporada.

2.6.1.1.4 Documento de periodización:

Consiste en una planificación total de la temporada, luego existe uno diferente por cada deportista y temporada. Contiene:

- Nombre de la temporada

- Tipo de progresión

- Listado de Macrociclos, Periodos, Mesociclos y Microciclos.

- Volumen por microciclo.

- Fechas correspondientes a cada microciclo.

- Listado de ejercicios diarios del documento de progresión, con el nivel correspondiente del trabajo en cada microciclo.

- Listado de tests (del documento de tests) que se realizarán coincidiendo en los microciclos.

El entrenador del atleta es el que edita este documento. No está visible para el atleta normalmente, si bien el entrenador puede hacerlo visible si lo desea.

2.6.1.1.5 Documento de progresión:

Listado de tipos de entrenamientos con diferentes niveles de intensidad. En cada nivel de cada tipo de ejercicio se presentará uno o varios ejercicios diario a desarrollar. En el caso de que sean varios será para comodidad del entrenador a la hora de elegir uno de ellos, no se realizarán todos en la misma sesión o microciclo. En este documento se define la intensidad del trabajo realizado en la sesión en que se realiza el ejercicio. Este ejercicio diario se mostrará en el documento de planificación en la lista de ejercicios a desarrollar y puede ser modificado por el

entrenador en ese documento. Este documento en teoría varía muy poco, pero el sistema, al añadir una nueva sesión, comprobará que el ejercicio existe y si no añadirá el ejercicio de esa sesión a la lista. En el sistema cada entrenador puede definir el número de documentos de progresión de especialidad de cada deporte, en función de los niveles de los deportistas (en deporte de equipo se define al equipo entero en un mismo nivel, sin distinción de jugadores). El entrenador puede añadir, editar y borrar tipos de entrenamientos, deportes y especialidades, y añadir editar y borrar ejercicio diario para cada nivel de cada tipo de entrenamiento. Cada entrenamiento diario podrá tener varios tipos según lo que tenga que realimentar el deportista. Series

2.6.1.1.6 Documento de programación:

Es el documento donde se tiene el entrenamiento que tiene que hacer cada día el deportista. Está estructurado en microciclos, y en cada día que dure el microciclo se pueden poner hasta 3 sesiones de entrenamiento, que si bien no es lo usual, si que pueden llegar a alcanzarse en deportistas de muy alto nivel.

Cada microciclo tiene:

- La carga de trabajo establecida en la periodización

- La carga efectiva realizada según los datos de realimentación del sujeto entrenado.

- Las horas de entrenamiento.

- El ejercicio diario que tiene que realizarse según las periodización

- El ejercicio que al buen criterio del entrenador deba realizarse en vez de el propuesto por la tabla de progresión.

- Datos que el atleta ofrece como resultados de su entrenamiento para que el técnico pueda comprobar su estado y evolución. Consiste en que el deportista introduzca en una tabla el tiempo dedicado a cada tipo de trabajo de cada sesión. Esta tarea se tratará de automatizar en lo posible mediante la captura de los datos del monitor de pulso cardiaco que el atleta acostumbra a portar durante el entrenamiento.

Según lo indicado anteriormente es el entrenador el que introduce los datos posteriormente a que el programa haya presentado unos ejercicios teóricos según la periodización.

2.6.1.1.7 Documento de testing:

Contiene la información de los diferentes test que realiza cada entrenador Para cada test se guarda: el código del test, la capacidad que mide, el nombre del test, la medida en que se realiza, porque se realiza, material necesario, otros. Existe uno por entrenador. El entrenador puede añadir, editar y borrar test de su documento únicamente.

2.6.1.1.8 Documento de estadísticas en competición:

Hay un documento de estadísticas del deportistas por temporada. Se recogen la fecha, la prueba, y el resultado objetivo de su realización. En el resultado objetivo pueden ser tanto los goles marcados, como los pases bien o mal dados en los deportes de balón, o las marcas de una prueba así como los diferentes pasos por las distancias o el puesto. También se pueden incluir los test realizados con sus resultados.

2.6.1.1.9 Documento de datos médicos:

El entrenador podrá solicitar a sus deportistas los datos médicos que considere necesarios. Estos datos estarán en un fichero encriptado del que solo se podrán extraer con una clave privada de la persona que tenga permiso para verlos, y que el deportista haya concedido el permiso. Este documento consistirá es el conjunto de informes médicos de un deportista del club. Cada tipo de informe tiene diferentes campos de datos médicos. Un deportista puede crear,editar y borrar sus informes médicos. Los documentos médicos serán estándares para toda la organización y los 4 básicos que vendrán configurados de serie son: Espirometría, Test de Esfuerzo, Consumo máximo de oxigeno y analítica sanguínea y de orina completa.

El sujeto analizado es el propietario del documento.

Puede permitir y denegar acceso al documento, al informe o a alguno de sus datos concretos a cualquiera del personal técnico del club.

2.6.1.1.10 Documento de tablas de ejercicios:

Contiene una lista de conjuntos de ejercicios. Se llama tabla de ejercicios al conjunto de ejercicios que se realiza como ejercicio diario de un entrenamiento. Cada tabla de ejercicios podrá tener características de realización como tiempos de ejecución, repeticiones, tiempos de recuperación, ejercicios de recuperación entre ejercicios, ejercicios de recuperación entre tablas, o la importancia o no del orden en la ejecución consecutiva de los ejercicios. Cada ejercicio tendrá un nombre, un músculo que trabajar, una descripción, un dibujo explicativo y una zona de trabajo.

Tiene la posibilidad de añadir, borrar y editar los ejercicios diferentes para las tablas.

Se puede generar una tabla de ejercicios idéntica a una ya existente con nombre diferente.

Se pueden añadir y quitar ejercicios de una tabla individualmente o por músculo o zona de trabajo.

2.6.1.2 Gestión de inscripciones en competiciones

Este módulo es un añadido posterior que demuestra la facilidad de encontrar maneras para incluir nuevos módulos en Gesport. Es un programa que recoge inscripciones públicas en carreras a través de internet.

Las necesidades que cubre este programa son la inscripción de uno o varios atletas en una carrera determinada. Al finalizar la inscripción el atleta o persona que ha inscrito debe tener una hoja que resuma con un listado la personas inscritas durante el proceso, el número de inscripción, para poder identificarle posteriormente, el nombre de la carrera para la que se ha realizado la inscripción y la cantidad a ingresar en el caso de que haya que pagar esa inscripción.

Las lineas con cada uno de los atletas inscritos deben especificar el Nombre, Apellidos, Fecha de Nacimiento, Sexo, Talla de camiseta y Dorsal.

Como tiene que valer para múltiples carreras, el programa tiene que tener una parte de administrador donde se puedan definir las carreras, las fechas en las que están disponibles la inscripción para ellas, los años que comprenden cada categoría, los diferentes circuitos que pueda tener, y que categoría corre cada circuito. En el apartado de categoría se tendrán los precios que cueste la inscripción. Tendrá en cuenta que haya inscripciones para marchas no competitivas, como en el caso de las carreras para menores donde muchos padres suelen acompañar a sus hijos.

El documento de todos los inscritos por internet debe poder ser exportable en diferentes

formatos. Esta última parte del programa no es necesariamente válida, ya que, usando herramientas de software libre sobre la base de datos (phpMyAdmin) podemos hacer filtros muy precisos para el momento de la exportación para el sistema de resultados. Esta labor sería del administrador del sistema y siempre se podrían desarrollar sistemas de informes que devolvieran esta información a administradores de la aplicación.

Otros objetivos no contemplados serían la inscripción de equipos, la reserva de dorsales según puestos en años pasados o cualquier otra cosa que se nos pudiera ocurrir. No se ha realizado por ser un ejemplo del funcionamiento de las extensiones.

2.6.2 Requisitos no funcionales

Para el análisis se han determinado las siguientes condiciones en las que se va a mover la aplicación:

- La aplicación resultante deberá ser accesible a todos los usuarios a través de Internet. Esta disponibilidad podrá ser va web o por descarga directa de un programa que permita el acceso a los datos pertinentes del servidor central.

- Funcionará desde algún terminal móvil, teléfono o agenda electrónica, para la toma de tiempos, resultados y variación de entrenamientos durante el trabajo de campo para los entrenadores y técnicos. Las funcionalidades de este módulo serán las mínimas posibles para permitir un trabajo con varios deportistas o grupos.

- Tendrá un entorno gráfico de usuario de fácil manejo y preparado para una adaptación a tecnologías de accesibilidad avanzada (personas con problemas de vista, oído o movilidad en sus extremidades).

- Soportará la internacionalización. En la finalización del proyecto funcionará en 3 idiomas, Español, Inglés y Francés, y tendrá la posibilidad de añadir idiomas tan solo con añadir los ficheros de traducción adecuados. El idioma por defecto de funcionamiento será el Inglés, pero el software detectará del sistema operativo o del navegador el idioma de la persona que está trabajando y en el arranque lo adaptará. Permitirá la elección manual de idioma.

2.7 Identificación de las clases de análisis

Con los datos que tenemos hasta ahora vamos a identificar las clases del programa. Primero se muestra un listado de candidatas que luego se filtrará. Estas clases que identificamos ahora son las que componen el dominio del problema a resolver, que no tienen porque ser clases efectivas en el diseño y programación de software. Se corresponden a las clases de objetos que intervienen en todo el ámbito del problema en el mundo real. Así mismo en la fase de diseño se crearán nuevas clases para el intercambio de mensajes, las relaciones, los listados y el trabajo interno de software. Estas últimas clases de objetos se diseñan pensando en la solución software y en esta fase del proyecto estamos analizando el problema.

2.7.1 Clases candidatas

2.7.1.1 Del análisis de la gestión de entrenamientos

Documento de datos personales, deportista, socio, trabajador, NIF, nombre apellidos dirección, teléfono, número de cuenta bancaria, entrenador, presidente, vicepresidente, secretario,

vocal, tesorero, director técnico, personal técnico, director de área, coordinador, secretario técnico, documento de datos del deportista, grupo de entrenamiento, categoría, equipo, documento de objetivos, objetivo principal, objetivo secundario, objetivo, temporada, documento de periodización, tipo de progresión, macrociclo, periodo, mesociclo, microciclo, sesión, ejercicio diario, documento de progresión, tipo de entrenamiento, nivel de trabajo, test, documento de programación, carga, carga de la periodización, carga realizada, horas de entrenamiento, deporte, especialidad, entrada de realimentación, tipo de carga, documento de testing, código de test, capacidad física básica, medidas de test, documento de estadísticas en competiciones, fecha, prueba, resultado, documento de datos médicos, tipo de informe, informe médico, dato médico, documento de tablas de ejercicios, tabla de ejercicios, músculo, dibujo explicativo,zona de trabajo, ejercicio.

2.7.2.2 Del análisis de la gestión de inscripciones en carreras

Inscripcion, evento, circuito, categoría, sexo, precio, dorsal, deportista, nombre, apellido, nif, email, dirección, fecha de nacimiento y talla.

2.7.2 Clases del dominio del problema

2.7.2.1 Del análisis de la gestión de entrenamientos

Del listado de clases candidatas hay algunas que por la propia definición del problema no plantean ninguna duda de son clases efectivas en la solución del problema. Las que me han planteado la duda son:

- NIF: es un tipo de documento personal que está formado por una serie de números y una letra. Si bien cabría contemplar otros tipos de identificadores, pues es común entre los clubes contar con deportistas extranjeros. El hecho de si es clase o no viene definido por la capacidad para realizar operaciones. Si en la resolución del problema se va a necesitar comprobar la veracidad y la corrección del número introducido y esa comprobación la va a hacer una operación propia del documento entonces el Documento identificativo tendría el estatus de clase. Digo Documento identificativo, que ya no es NIF, por la variedad de documentos que pueden admitirse por la situación de los documentos extranjeros (desde NIE hasta pasaportes o documentos de otros paises) y, aprovechando la coyuntura de la comprobación, se podría utilizar para hacer lo mismo con los datos empresariales de terceros.

- Nombre, apellidos: en este caso los consideraré atributos de persona, como clase base para todos los tipos de persona que habrá después en el club.

- Dirección: Aquí viene la duda de si se va a trabajar la dirección como una serie de caracteres o va contener más campos como código postal (comprobable al igual que NIF para hallar la provincia correcta), de si cada persona va a poder tener más de una dirección y si esta misma dirección va a servir para otras entidades como empresas. Como estas características se dan (las empresas tienen una o varias direcciones y va a reutilizarse), entonces consideramos a dirección como clase, y añadimos lista de direcciones también, pues a cada persona o tercero le corresponderá una lista de direcciones.

-Teléfono, numero de cuenta bancaria: al igual que dirección, por poder verificar la correcta disposición de dígitos de control, prefijos o poder enlazar directamente con programas de llamada (asterisk, Skype, Ekiga con protocolo SIP, etcétera) consideramos ambos datos clases, puesto que pueden tener operaciones internas.

- Deportista, entrenador, administrado y el resto de diferentes puestos en la infraestructura del club. Se podría pensar que son objetos de la clase persona con diferentes características. En esta fase de análisis son clases de personas que intervienen en la organización y en la fase de diseño se decidirá la implementación apropiada para ellos estudiando cada caso particular de individualidad en instanciación, o por áreas.

- Objetivo puede ser perfectamente una cadena de texto que pertenezca a uno de los objetivos principales o secundarios, la problemática es que a lo largo de la vida deportiva de los deportistas los objetivos se repiten, o se comparten por varios de ellos (y más aún en los deportes de equipo). Visto de esta manera los objetivos deberían poderse almacenar y guardar de una temporada para otra y así constituir una clase efectiva en la resolución del problema. Además propongo que sea posible listarles a la hora de seleccionar, pues es probable que un objetivo haya sido propuesto antes una vez se haya manejado el programa durante un tiempo. Así se añadiría la clase lista de objetivos.

- Ejercicio diario se refiere al ejercicio realizado en una sesión por lo que estaría mejor denominado "ejercicio de la sesión". Constituiría una clase en la que se podrían incorporar ejercicios predeterminados en el documento de progresión y otros que el entrenador del deportista considere oportunos.

- Tipo de entrenamiento y nivel de trabajo son dos conceptos cruzados. Para un tipo de trabajo aeróbico de nivel 1 se corresponde una carrera continua de 15 minutos con una recuperación andando a la mitad del tiempo de 5 minutos, y para ese mismo tipo de trabajo en un nivel superior puede hacer una hora de carrera continua lenta a un ritmo determinado que una persona sin preparación física no consideraría, ni por asomo, lento. Con esto quiero decir que la clase en si vendría representada por cada relación tipo de entrenamiento y nivel, o bien cada tipo de entrenamiento tendría el carácter de clase con una serie de "ejercicio de la sesión" asociados a cada nivel. Teniendo en cuenta que hay que definir en algún momento las características del tipo de entrenamiento opto por declarar clase al tipo de entrenamiento, con los diferentes niveles asociados como atributos de esta y que corresponden a ejercicio de la sesión.

- Fecha. No aparece en los datos personales pero debería aparecer con la fecha de nacimiento. En la fase de análisis es un atributo de la clase persona (fecha de nacimiento) y de la clase prueba o test (fecha de la prueba, competición o test). En la fase de diseño probablemente se convierta en una clase, aunque venga dado por el lenguaje de programación en su especificación, porque necesitará de métodos y condiciones para que la fecha sea válida.

2.7.2.2 Del análisis de las inscripciones en carreras

Son todas clases efectivas salvo aquellas que se refieren a datos concretos personales, luego quedan: evento, categoría, circuito, deportista, inscripción, precio y talla.

3. Planificación

Detalles organizativos a la hora de resolver el problema.

3.1 Desarrollo temporal

El tiempo estimado de desarrollo de esta aplicación son 9 meses realizada por una sola persona con una dedicación de 7 horas al día.

La primera parte de este tiempo (1 meses) corresponde a la realización de estudios previos de mercado y de funcionamiento de las organizaciones para las que está destinada Gesport. Durante este periodo se han entrevistado a directivos, entrenadores, deportistas y trabajadores de clubes de atletismo. Se han estudiado las necesidades anteriormente detalladas.

La segunda fase (3 meses) ha consistido en plasmar todo esa información obtenida en la redacción del análisis, en la creación de los diferentes gráficos y esquemas que provee el Lenguaje de Modelado Unificado para facilitar una cómoda codificación de la aplicación. Se han seguido manteniendo las entrevistas para corregir los posibles errores en los conceptos iniciales. En esta parte se han aportado al proyecto todos los diagramas de Clases y Actividades. Aparte de los gráficos propiamente dichos se han desarrollado las explicaciones de los términos más complejos de temas deportivos y de las relaciones en los diagramas que pudieran no llegarse a entender. Se ha hecho el estudio de las soluciones posibles para los problemas planteados y se han tomado las decisiones relativas a metodología y tecnología de clientes y servidores a usar. En esta etapa también se han diseñado las pruebas a las que se habría de someter cada parte de la aplicación y la aplicación en general.

La tercera fase (4 meses) ha consistido en la implementación y escritura de todos los códigos fuentes necesarios, búsqueda de bibliotecas que ya proveyeran de funciones necesarias (evitar inventar la rueda), diseño de los interfaces gráficos de usuario y pruebas de conexiones entre bases de datos, servidor de aplicaciones e interfaces de usuarios. En esta parte se ha ido avanzando en ciclos por objetivos.

El último mes se ha dedicado a completar la presentación de diapositivas, los manuales de usuario y administrador, terminar detalles y flecos sueltos, y probar más combinaciones posibles que puedan generar error para evitar efectos indeseados en la presentación ante el jurado - cliente.

Durante todo este periodo de tiempo el alumno ha dedicado una media de 6 horas al día, todos los días de la semana y los fines de semana y festivos 4 o 5 horas.

3.2 Ciclo de vida y metodología

Se ha utilizado un desarrollo en espiral, porque, aunque se estudió la posibilidad de realizarlo en cascada por que se tenía conocimiento de todas las áreas del proyecto desde el estado inicial, he creído más apropiado hacer un ciclo de la espiral con cada funcionalidad añadida al programa. Así de esta manera tendremos versiones funcionales de partes del programa en cada ciclo que podrán ser utilizadas por los miembros del club mientras se sigue el desarrollo de otras áreas.

El orden a seguir en esta metodología ha sido el siguiente:

- Generación de estructura de datos maestros: unidades y factores de conversión.

- Sistema de acceso de usuarios.

- Sistema gestor de sesiones de entrenamiento para entrenador.

- Aportes del sujeto entrenado.

- Construcción de las estructuras de que engloban a las sesiones (macro, meso y microciclos).

- Cálculos de acumulación de carga en las anteriores estructuras.
- Añadido de soporte para diferentes progresiones.
- Añadido de soporte para tablas de ejercicios.
- Añadido de sistema de subida de ficheros (dibujos para explicación de ejercicios)
- Añadido de informe médico.
- Añadido sistema de inscripciones.
- Gestión de datos médicos con datos cifrados.

3.3 El diseño del interfaz de usuario

Este programa tiene como objetivo el facilitar la gestión de los aspectos deportivos, organizativos y económicos a un club deportivo. Así pues el manejo del mismo tiene que ser lo más sencillo e intuitivo posible. Para que resulte sencillo se utilizará un sistema de ventanas gráfico como puede ser el de Windows, Qt, Gtk, alguno de los múltiples posibles para Java (AWT, Swing con integración en escritorio) o el interface web del navegador. La elección del sistema de ventanas vendrá determinada por sistema base que soportará la aplicación.

Para que sea intuitivo se estudiarán las operaciones de trabajo en el campo de los profesionales de forma que la manera de introducir y obtener los datos sea parecida a la que se realiza en vivo y en directo en la instalación deportiva.

Además se tendrán en cuenta las recomendaciones de usabilidad recogidas en las diferentes organizaciones que tratan sobre el tema, como en las normas de GUI de Gnome Fundation, las recomendaciones para el diseño de aplicaciones Java, o las especificaciones WAI de la W3C de accesibilidad web para personas con discapacidad. El seguimiento de unas recomendaciones u otras dependerá del interfaz finalmente utilizado.

3.4 Presupuesto y financiación

Para el desarrollo de esta aplicación se ha estimado un coste por hora de 45€ en los trabajos de ingeniería (análisis y diseño) y de 30€ en los trabajos técnicos (programación y pruebas). Se excluye de este presupuesto la instalación y mantenimiento. Aún siendo el mantenimiento la fase de producción de software más longeva y las puestas en marcha las fases más críticas y propensas a fallos, serían las fuentes de financiación directas para costear el desarrollo del programa y se estudiaría cada cliente en particular. En las diferentes fases de la creación del programa se ha contado con el siguiente presupuesto.

Concepto	Cantidad	Coste detalle	Coste Total
- Horas de análisis:	50h		2.250€
*Entrevistas con cliente:	20h	900€	
*Análisis de los requisitos:	30h	1.350€	
- Horas de diseño del software:	40h	1.800€	1.800€
- Horas de diseño de pruebas:	16h	720€	720€
- Horas de diseño de la interfaz:	32h	1.440€	1.440€

- Horas de programación:	60h	1.800€	1.800€
- Horas de pruebas:	20h	600€	600€
- Hardware necesario para pruebas:			2.050€
* PDA	1 unidad	250€	
* Ordenador personal para el desarrollo de la aplicación + garantía ampliada de 3 años	1 unidad	1.800€	
* Servidor para pruebas[1]		0€	
* Servidor de producción para versiones finales en funcionamiento[2]		0€	
- Servidores de control de versiones, copia de seguridad y distribución[3]		0€	0€
Total			**10.660€**

Tabla 2: Presupuesto global de desarrollo

El proyecto, según la normativa de la Universidad de Valladolid, será licenciado como software libre al amparo de la GPL v.3. El tiempo de desarrollo del proyecto se considera una inversión para dar un posterior mantenimiento y servicio personalizado a clubes, gimnasios, entrenadores personales. De esta manera la financiación se obtendrá de la repercusión del coste del mantenimiento e instalaciones del software en los potenciales clientes que puedan hacer uso de el, así como de las ventas e instalaciones de hardware en los locales de los mismos. Se prevé una amortización total del coste de la inversión en 1 año y 6 meses, con una cartera inicial de 2 clientes (con precio especial por haberse propuesto para la realización de las pruebas) y una cartera final del primer año de 7. Se estudiará, según el entorno económico, si es conveniente afrontarlo en tan solo un ejercicio.

1 Se utilizó el PC de escritorio con el sistema libre de virtualización Qemu y un Sistema operativos de servidor Ubuntu Server para realizar las pruebas.
2 No se ha realizado ninguna instalación de aplicación en entorno real.
3 Alojado en la forja de software de RedIris. Servidor para desarrolladores de software libre y comunidad de los mismo para el intercambio de ideas y colaboración entre proyectos.

4. Diseño de la solución final

Elección de la plataforma de servidor, lenguaje de programación, diagramas de clases, de secuencia y colaboración, algoritmos de cálculos de índices de trabajo físico y relaciones entre las diferentes áreas del programa.

4.1 Elección de la plataforma

4.1.1 Soluciones multiplataforma

Uno de los requisitos no funcionales de la solución final es que el programa del cliente funcionase en varias plataformas. El motivo de esto era que en una organización concreta suele usarse la misma plataforma en todos los ordenadores, pero la aplicación debe ser exportable a futuros posibles clientes. Si bien la mayoría de los usuarios del posible mercado en el que nos encontramos usa sistemas operativos Microsoft Windows ®, en el caso concreto planteado del Club de Atletismo Joaquín Blume existen al menos dos puestos en los que el sistema operativo es Linux o MacOS. Ante este problema se han barajado las siguientes soluciones:

4.1.1.1 Lenguajes mono-plataforma con interfaz de usuario común

Consiste en crear la aplicación con un interfaz de usuario basado en librerías libres utilizables en cualquiera de estos sistemas operativos de escritorio. El lenguaje de base es mono-plataforma, pero las librerías que usa son comunes a varios sistemas operativos, de manera que el mismo código fuente con diferente compilador (para cada sistema operativo) darían lugar al mismo programa.

El que sean libres viene dado por tres razones principalmente:

- El coste del uso de librerías propietarias.

- El soporte por una comunidad de usuarios y desarrolladores de estas librerías muy involucrados por su voluntariedad.

- Por último, y no menos importante, por el aporte de componentes al movimiento del software libre que puede generar el proyecto.

Esta posibles librerías de interfaz de usuario son GTK (Gimp Tool Kit) o Qt.

- GTK es una implementación de un sistema de ventanas en las que se basa el entorno de ventanas para sistemas UNIX Gnome. Comenzó su desarrollo por los creadores de Gimp, un software de edición gráfica de mapa de bits para sistemas UNIX. Hoy en día tiene capacidades parecidas a Adobe Photoshop ® gracias a las extensiones programables en diferentes lenguajes, aunque con una velocidad de ejecución más lenta debido a que estas extensiones se hacen con lenguajes interpretados. La base de la programación para GTK es C, aunque tiene extensiones para C++ y últimamente ha salido, a través del proyecto Mono, para C#. Esta última manera de programas se considerará en las plataformas interpretadas, puesto que, al igual que con Java, una misma compilación sirve para los tres sistemas operativos.

- Qt es una implementación de una empresa privada llamada Trolltech. En sus inicios tenía licencia privativa, pero su uso para el desarrollo del entorno de ventanas KDE, también para sistemas UNIX y actualmente funcionando en Windows y en Mac sustituyendo a los sistemas por defecto de estos sistemas operativos, hizo que al final Trolltech liberara el código con un doble licenciamiento: si la aplicación que usa Qt es libre el licenciamiento es libre, si la aplicación tiene una licencia no libre entonces hay que pagar para usar las librerías. El lenguaje que se usa con Qt es C++, que permite orientación a objetos y supone una ventaja en el desarrollo sobre GTK.

Es posible que existan otras alternativas, pero estas son las más conocidas, más usadas y con más soporte de la comunidad. Existe la posibilidad de usar GTK+ que es GTK orientado a objetos. El uso de estas tecnologías tiene ciertas ventajas e inconvenientes.

- Ventajas: Se basan en lenguajes compilados, lo que implica una mayor velocidad de ejecución, tienen un interfaz común en todos los sistemas operativos, lo que simplifica, tanto los manuales de usuario, como el aprendizaje de los usuarios,

- Desventajas: Al ser lenguaje compilado necesita una compilación por cada sistema operativo destino. GTK en MacOS solo funciona con el servidor X de UNIX instalado. Al tener la base de datos en un lugar diferente al programa, que se conectaría remotamente, el trabajo sobre ella será lento y en las transferencias con bloqueo se puede bloquear la aplicación si la conexión falla o es lenta. Las diferentes especificaciones de los lenguajes de programación para los diferentes compiladores, no es lo mismo el C de Visual Studio que el de GCC, harían necesarias modificaciones en la escritura del programa para transformarlo de una plataforma a otra. Si bien estas diferencias serían mínimas, harían necesaria una revisión casi completa del código escrito.

4.1.1.2 Interfaz web a través de navegador

Esta solución consiste en introducir el software en un servidor web al que tengan acceso los usuarios. Este servidor web ejecutaría los programas en algún lenguaje y ofrecería las respuestas al que consulta o trabaja. Los lenguajes que se soportarían en web son múltiples, ya sean compilados a través de CGI, PHP, Ruby on Rails, JSP o ASP. La base de datos que almacenaría los datos del programa se ejecutaría en el mismo servidor o en un servidor dedicado en la misma red, de manera que las consultas a la base de datos serían bastante rápidas.

Las opciones que se han contemplado para el desarrollo sobre servidor web son:

- PHP: lenguaje orientado a objetos de tipado débil. Se puede ejecutar sobre servidores Windows, Linux, MacOS, Solaris, y casi todos los posibles. Se necesita un servidor web sobre el que se ejecute el interprete PHP. Este puede ser Apache, Roxen, Microsoft Internet Information Server y muchos otros, entre ellos todos los que soporten CGI, pues se puede configurar como programa que funciona sobre CGI. Es un lenguaje de scripting que ha añadido el soporte de objetos desde la versión 4 y que desde la 5 el soporte es casi total. Con determinados frameworks la potencia de PHP en programación puede asimilarse a Java o .NET, con la ventaja o desventaja del tipado débil. Digo desventaja porque se pueden producir asignaciones en la programación de objetos que no corresponden con los que son, pero ventaja, porque el propio motor de interpretación del lenguaje, al menos en sus últimas versiones, llama a los métodos que transforma un objeto en otro si la operación lo requiere y con total transparencia.

- ASP: Actualmente el desarrollo de ASP se realiza en .NET en su mayoría. Los desarrollos en ASP basados en Visual Basic compilado y puesto en servidor ya no se usan mucho por la mayor facilidad de trabajo en .NET. El servidor debe ser necesariamente Windows salvo que se trabaje con el proyecto Mono, en cuyo caso se perdería la posibilidad de trabajar con Visual Studio, que sería una de las mayores ventajas que tendría, ya que tiene asistente para diseño de formularios web muy avanzado. La ventaja sobre PHP es que el código es compilado en vez de interpretado y durante el la ejecución funciona más deprisa.

- JSP: Se trabaja prácticamente igual que en ASP. El único cambio es que ya no depende tanto del sistema operativo ya que la base del software es Java. El programa se ejecuta en un servidor de aplicaciones web sobre Java. Existen múltiples alternativas en el mercado, tanto libres como propietarias, lo que nos da una capacidad de adaptación mucho mayor que ASP. Existen múltiples posibilidades de diseñar el interfaz como una aplicación de gestión al uso, siempre y cuando nuestro entorno de desarrollo las soporte:

- Struts: Plataforma de desarrollo para Java. Consiste en una serie de componentes preparados para su integración en una página web con un sistema de datos y clases detrás. Existe una implementación de Struts.

- JSF: Java Server Faces es una definición de Java para desarrollar una plataforma de desarrollo. La gran diferencia con Struts consiste en la expansibilidad de sus objetos, que permite crear componentes basados en los genéricos. La otra gran diferencia viene dada porque al ser una definición existen diferentes implementaciones del estándar, con lo cual se puede elegir la que mejor nos convenga o mejores extensiones tiene aplicables para nuestra aplicación. El problema viene por el manejo de estos objetos en el entorno de desarrollo, que no se maneja igual que en un interfaz cliente, donde el nuevo objeto creado puede incrustarse con facilidad en las barras de herramientas para su inclusión en el programa (al menos con Netbeans y Eclipse, que son los dos IDEs que se han planteado en el caso de que se realice con esta tecnología).

Al igual que las tecnologías de lenguajes mono-plataforma tienen sus ventajas e inconvenientes:

- Ventajas: Interfaz para el usuario reconocida de antemano, porque casi todos los usuarios tienen la experiencia de navegar por la web anteriormente. Mayor velocidad de conexión a las bases de datos desde el programa pues se encuentran localizadas en la misma red o servidor (dependiendo del caso del cliente). Experiencia del programador en este ámbito de varios años.

- Desventajas: Necesidad de conexión a la red en todo momento durante el manejo del software.

4.1.1.3 Tecnologías multiplataforma

Las tecnologías multiplataforma son sistemas que se componen de un compilador del lenguaje y un interprete para cada plataforma destino. Están realizados de manera que una misma compilación sirva para todos los sistemas operativos y arquitecturas de procesadores, pues los mensajes con el sistema operativo los realiza el interprete. Así si en un programa usamos un la posibilidad de varios hilos de ejecución y el procesador y el sistema operativo lo soportan será la maquina virtual que interpreta el programa compilado la que cree las nuevas instancias de ejecución y no el programa por si mismo. De esta manera y con una fuente de datos común se puede trabajar en diferentes ordenadores, con diferentes sistemas operativos y procesadores con un mismo programa.

Las tecnologías multiplataforma disponibles actualmente en el mercado y que cuentan con una documentación abundante son Java y el proyecto Mono, una especificación libre del estándar ECMA para programación, equivalente a la tecnología .NET.

- Ventajas: Un mismo código fuente sirve para todos los sistemas operativos. Hay Plataformas de desarrollo lo suficientemente avanzadas como para integrar la programación de servidor, con el diseño de interfaces de usuario y ayuda en la búsqueda de funciones de librerías avanzadas. Java permite el almacenamiento de certificados de seguridad en sus almacenes, esto es una ventaja respecto a la tecnología de navegador, donde el almacén es compartido por varios usuarios si no se usa correctamente, y respecto a lenguajes compilados, donde el almacenamiento es externo o hay que implementarlo a mano.

- Inconvenientes: Es ligeramente más lento en ejecución puesto que el programa funciona sobre una máquina virtual, otro programa, si bien con Java este inconveniente se está solucionando desde la apertura de su código al empezar a incluirlo en los núcleos de algunos sistemas operativos. La conexión de la base de datos tiene el mismo problema que los lenguajes mono-plataforma, con la desventaja añadida de que el fallo puede resultar en el sistema operativo, en el programa, o en la máquina virtual que interpreta el código. Esta última posibilidad de fallo solo se da en lenguajes interpretados como es el caso de java. El aspecto gráfico es diferente entre los diferentes sistemas operativos si se usa la integración con los mismo, o el aspecto es diferente al del resto del sistema

operativo. El primer inconveniente dificulta el proceso de aprendizaje, pues existen sistemas operativos que cambian de sitios los menús y los integran a su manera. MacOS X es el caso más llamativo

4.1.2 La gestión de los datos

Esta aplicación va a contener una gran cantidad de datos de diferentes deportistas. El diseñador de la aplicación no concibe el producto sin la potencia de un Sistema Gestor de Bases de Datos (SGBD). Para la elección del sistema gestor de bases de datos se han tenido en cuenta diferentes opciones, desde bases de datos orientadas a objetos hasta las soluciones comerciales más conocidas. A continuación vamos a pasar a describir las diferentes posibilidades:

4.1.2.1 Gestores de Bases de Datos Orientados a Objetos

Estos sistemas de bases de datos son una relativa novedad. A pesar de que las metodologías orientados a objetos tienen ya cierta solera en el ámbito de la informática, el almacenamiento de la información como objetos no se ha desarrollado mucho. Hay que tener en cuenta que un objeto no es un conjunto de datos, sino los datos de ese conjunto, las operaciones que se pueden realizar con ellos, los mensajes que puede recibir para devolver información y los mensajes que puede recibir para alterar su estado. Con todos estos detalles se nos hace muy difícil la representación de un objeto en una base de datos.

Aún así el mundo de la informática ha evolucionado lo suficiente como para que exista este tipo de software en el mercado. Así tenemos la casi recién nacida JavaDB, DB4o (DataBase for Objects, del 2004), o con mucha más experiencia en el mercado, pero con carácter comercial Cache. Puesto que es un sistema aún bastante nuevo y, a pesar del estándar declarado OQL (Object Query Language), aún las diferentes opciones son demasiado diferentes como para pensar en la posibilidad de una migración de una a otra y demasiado jóvenes como para pensar que en un futuro van a persistir. Es por esta razón, por algunos motivos tecnológicos (se necesitan ejecutar como una clase dentro del sistema, salvo Cache,con lo que tendríamos que programar un servidor de manera imprescindible, y no permiten conexión remota si no es programada en la aplicación a través de un socket) y algunos otros de carácter teórico (el lenguaje de consulta no está basado en una matemática formal como en los sistemas de bases de datos relacionales y relacional – objeto) que se ha descartado este tipo de base de datos en la realización del proyecto.

4.1.2.2 Gestores de Bases de Datos Relacionales

Las bases de datos más comunes en el mercado son las bases de datos relacionales. Son sistemas gestores de bases de datos basados en el modelo entidad-relación. Su principal característica es que los lenguajes de consulta, edición y administración se basan en casi todos los casos en el estándar SQL (Structured Query Language). Esto significa que en cualquier base de datos relacional se puede realizar una consulta estándar de búsqueda de datos, de creación o edición de datos, o de administración de permisos de usuarios de una manera única para todas ellas. Este estándar no se cumple al completo en todas las bases de datos, aunque es cierto que los cambios son debido a las mejoras y nuevas aplicaciones de las sistemas gestores que lo modifican.

En este grupo de bases de datos se encuentran MySQL, SQL Server, SQLite o sybase. Soportan mejor el trabajo en red porque tienen integrado un servidor que atiende a las consultas a través de un socket abierto en el puerto en el programa cliente (en el caso anterior había que hacer un programa en servidor que era el que recogía las consultas y las pasaba a la base de datos). El problema es que no se pueden representar objetos y las diferentes herencias se tienen que construir

en tablas relacionadas con condicionantes de claves foráneas. Esto dificulta mucho el desarrollo del diseño de la base datos en relación con el diseño orientado a objetos de la funcionalidad de la aplicación.

4.1.2.4 Gestores de Bases de Datos Relacional-Objeto

Una solución intermedia a estas dos es aplicar las extensiones al SQL que tienen ciertas bases de datos relacionales (Oracle, PostgreSQL) para trabajar con relaciones entre clases. El SQL de estas bases de datos se sale del estándar (lo que no significa mucho problema, pues entre las que son solo relacionales ya hemos comentado que también existen diferencias) y aporta nuevas palabras clave para trabajas con herencia, creación de nuevas implementaciones de tipos de datos y, aún sin estar relacionado con las tablas donde se almacenan estos objetos, permiten extensiones de funcionalidad con lenguajes de programación integrados en la misma base datos (Extensiones PL o Procedural Language). MySQL 5 ya contempla alguna de las funciones de las bases de datos Relacional-Objeto.

El licenciamiento de una y otra base de datos es una de las grades diferencias. Oracle, a pesar de tener una licencia de uso gratuito, no es libre. En la licencia de uso gratuito ofrecen soporte para un solo procesador y funcionalidad limitada. La variedad de pago es una licencia de soporte anual del producto sin derecho a actualizaciones en el caso de no renovación. PostgreSQL por el contrario es libre. Si bien no tiene el respaldo de una comunidad de usuarios como pueda ser el conjunto de programadores web que tiene MySQL, en su desarrollo participan multinacionales como Fujitsu o Sun Microsystems, y almacena los datos del sistema de telefonía IP Skype.

Este último tipo de base de datos sería el más apropiado para esta aplicación, ya que, en sistemas de producción, estimo que es más rentable la estabilidad de un producto estandarizado con posibles extensiones tecnológicas, que la extrema innovación de las bases de datos orientadas a objetos y que el trabajoso proceso de transformar un diseño orientado a objetos en un diagrama entidad-relación. Pero aún queda otra posibilidad que con la base de datos relacional de toda la vida puede ser bastante útil.

4.1.2.3 Sistemas intermediarios entre Base de Datos Relacional o Relacional Objeto y Programación Orientada a Objetos

Además de las diferentes posibilidades de sistemas gestores de bases de datos, existen en el mercado diferentes interfaces que hacen posible la persistencia de los objetos de una manera sencilla al programador. Con esto quiero decir que se trabaja con el objeto directamente sin tener que hacer declaraciones SQL para obtenerlo, guardarlo o modificarlo. Existen varios programas que realizan esta función de interface. Me he fijado en aquellos que por ser software libre podrían encajar con la licencia del proyecto: Hibernate, de RedHat, e iBatis, de Apache para Java y Propel y Doctrine de Sensiolabs para PHP. Aparte está el sistema de persistencia de objetos de Java.

La ventaja de estos programas consiste en que la gestión de la base de datos no es necesaria desde la lógica del programa. Partimos de los objetos y son los propios objetos los que tienen la persistencia adquirida como propiedad, ya sea porque implementa un interface que nos la define, o por que herede de alguna clase que ya la tenga implementada. Con una definición de modelo de datos del objeto (explicar los campos que tiene el objeto en un fichero que entienda el sistema, como un XML) una clase puede generar desde la estructura de la tabla donde se van a almacenar los objetos hasta realizar las típicas operaciones de recuperar, guardar y editar el contenido del objeto sin que el programador escriba ni una sola sentencia SQL. Así conseguimos un programa totalmente orientado a objetos sin constructores ni métodos extraños con lenguajes externos al de programación (SQL, OQL).

Cada uno de estos sistemas utiliza un método para garantizar una persistencia correcta. Hibernate forma parte del proyecto de JBoss de RedHat. Aparte del sistema de persistencia en si mismo tiene añadidos que permiten la indexación de campos para facilitar la búsqueda en diferentes campos de la tabla. Además, a partir de la definición de las clases de datos, el fichero de configuración de conexión, y la equivalencia de las clases con la base de datos ,te crea en el sistema gestor de base de datos la estructura de las tablas que se van a utilizar

iBatis tiene la ventaja de que es más simple de manejar, pues no tiene tantos añadidos. A pesar de ello, la documentación es mucho mayor, así como la comunidad de usuarios, en Hibernate.

Doctrine sería el equivalente a Hibernate en PHP.

Symfony es un entorno de trabajo que abarca varios aspectos. No solo integra la persistencia de la que hablábamos antes mediante Propel, sino que facilita la generación de código y está preparado para la una integración total de páginas con AJAX. Tiene integrado el módulo de internacionalización en el propio framework, lo que representa una gran ventaja respecto a los otros que lo basan en la internacionalización del lenguaje base. En este caso realiza la gestión de la relación entre objeto y dato a través de Propel (otro proyecto de software libre) pero se puede integrar directamente con Doctrine, alcanzando un mejor control de las sentencias. Symfony es un entorno de trabajo que permite abarcar todos los aspectos de una aplicación web con una gran sencillez utilizando el patrón Modelo Vista Controlador.

La decisión de usar un sistema intermediario va a permitir escalabilidad en la aplicación. No va a depender de un sistema gestor de base de datos almacenado en un servidor. Cambiando el fichero de configuración el mismo esquema valdría para una base de datos pequeña almacenada en el propio disco de la aplicación (por ejemplo SQLite) como para una base de datos Oracle distribuida en varios servidores. Esto permite además al desarrollador que, una vez que tiene definidas las clases y los esquemas, se dedique solo a programar en el lenguaje de programación elegido, sin tener que preocuparse de la corrección de las sentencias de consulta a la base de datos. Además este sistema tiene la ventaja de que en búsquedas complejas y muy detalladas (en el caso de que no se use el servidor de indexación de Hibernate, que las realizaría con métodos orientados a objetos) se pueden pasar partes de la sentencia del lenguaje de consulta a la base de datos para concretar los resultados.

4.1.3 El patrón Modelo Vista Controlador (MVC)

Como hemos visto en el anterior punto hay varios sistemas que permiten una aproximación al patrón de desarrollo de webs MVC. En este patrón se estructura la programación en varias partes, cada una dedicada a una tarea específica. A saber:

- Modelo: Es la representación de la información con la que trabaja la aplicación. Esto incluye también el tratamiento y transformación de esta información. Nos referimos tanto a la estructura como al contenido.

- Vista: Es la lógica que transforma el modelo en una página web visible para que el usuario pueda interactuar.

- Controlador: Es el encargado de obtener los datos del modelo, pasárselos a la vista, procesar las peticiones de los usuarios o realizar tareas de autenticación. Realmente es el que relaciona las otras partes entre sí y con las respuestas del usuario en la vista.

Además de estas capas hay algunas más que se suelen utilizar más en el desarrollo de la aplicación. Estas se ven en el siguiente diagrama:

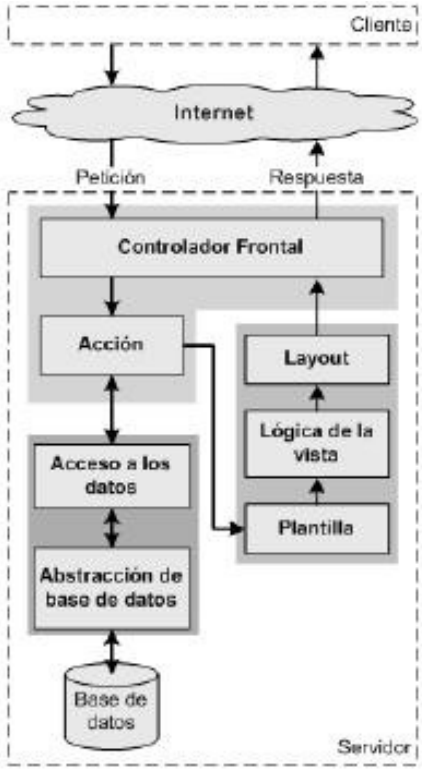

Ilustración 1: Esquema ampliado MVC

4.1.4 El servidor

Los datos actualizados que maneja el programa estarán almacenados en el servidor de base de datos. Además del propio servidor de los datos es muy posible existan procesos automáticos de cálculos, o incluso la posibilidad de que se consulte, a través de web ciertos datos públicos contenidos en las bases de datos. Dada esta situación la elección de un servidor apropiado es un punto crítico en el posterior funcionamiento correcto del sistema. En esta elección entran en juego el tipo de procesador, el procesador o procesadores, la memoria, la capacidad de almacenamiento y el sistema operativo.

La necesidad de capacidad de cálculo, almacenamiento en memoria y conexiones múltiples dependerá en todo momento del club deportivo y su magnitud. Puede que en ciertos clubes multidisciplinares se necesite un sistema de dos servidores, uno de almacenamiento de base datos y otro de procesamiento y ejecución del programa, o una granja de varios servidores con las tareas compartidas, de manera que si falla uno el sistema completo siga funcionando, y se pueda añadir capacidad de proceso y almacenamiento según la necesidad.

Uno de los puntos que puede ser clave en el funcionamiento de la aplicación, sea cual sea la magnitud de la organización, es la posibilidad de la virtualizacioń. La capacidad de ejecutar el servidor de aplicaciones y la base de datos en un mismo equipo, pero con sistemas operativos virtualizados da la posibilidad de cambiar en cualquier momento la estructura del sistema con impacto mínimo en el tiempo de migración. Además la seguridad que representaría el poder volver

a levantar cualquiera de los servidores (el de la aplicación o el de los datos) en cualquier ordenador que tengamos a mano con la máquina virtual instalada en caso de errores nos da la confianza de tener un tiempo de respuesta a fallos muy rápido.

El caso que nos ocupa es el de un club pequeño, que puede tener 10 o 20 conexiones simultaneas a lo sumo, unos 150 los socios, deportistas, directivos y trabajadores, y un uso del sistema informático por parte de 2 o 3 de los técnicos. Luego las necesidades son muy pequeñas, pero escalables. También puede ser muy interesante la capacidad de crecimiento que aporta al club la aplicación.

La tarjeta de red no será necesariamente más que la estándar de 100Mb/s pues el cuello de botella de la transferencia de datos se encuentra en la linea de conexión a internet de los usuarios con ADSL en el mejor de los casos. En el caso de que la infraestructura sea virtualizada se tratará de dar un interfaz fisico de red a cada maquina virtual.

En la elección del sistema operativo tenemos que tener en cuenta, además de que soporte virtualización, que sea totalmente compatible con la base de datos elegida (Oracle o PostgreSQL) y con el servidor de aplicaciones. Si bien el sistema operativo del servidor de aplicaciones podría ser diferente al de almacenamiento de datos, en futuro la formación del mantenedor de sistemas se podría encarecer, luego la elección sería la misma para ambos sistemas. Aquí tenemos las opciones:

4.1.3.1 Windows

Sistema operativo de Microsoft ® que en sus diferentes versiones de servidor ha evolucionado positivamente. Windows Server 2008 representa grandes avances de seguridad para un servidor que por la naturaleza cerrada de su código siempre ha sido objeto de ataques. La experiencia del diseñador con Windows Server (2003 en este caso) es positiva en medios de producción con software de gestión de terceros. Sus mejores características son la posibilidad de tener asistentes de todo tipo para configuración de software y hardware y la facilidad de manejo por su gran parecido al los Windows Home y Pro. Su gran desventaja es el incremento del consumo que representa cada incremento de versión, que hace que de una versión de a otra de Windows el hardware que la soporta quede obsoleto. La gran ventaja vendría si la elección del lenguaje de programación es .NET.

4.1.3.2 *NIX

Los sistemas operativos de la familia *NIX (UNIX, Linux, Solaris, *BSD, MacOS) tienen un origen común en los laboratorios de AT&T y se han ramificado dando lugar a versiones tanto propietarias, como libres. Tienen la ventaja de que entre la gran gama de núcleos de sistema operativo deferentes todos tienen las mismas herramientas de aplicación en la superficie, consola bash, servidor de ventanas X11, aplicaciones basadas en GTK y Qt y comandos de administración de sistema parecidos. El diseñador de la aplicación tiene larga experiencia en la administración de sistemas Linux de producción y MacOS de escritorio. La experiencia con estos sistemas operativos es muy buena en el ámbito de la estabilidad. Además, la reciente liberación de Java y las incorporaciones al núcleo del sistema operativo Solaris y Linux hacen que si el lenguaje es finalmente Java, se considere esta opción como la más aceptable. Además PostgreSQL nació para sistemas *NIX y, aunque la última versión ya lo soporte, en Windows está aún lejos de tener la misma estabilidad.

4.1.5 La plataforma elegida

La primera cuestión que se va dilucidar en este punto es el interface que van a usar los

usuarios para acceder a la aplicación. Hemos puesto en algunos de los requisitos que el usuario podría acceder desde diferentes lugares, sistemas operativos e incluso otros dispositivos, como móviles o PDAs. Para que esto sea posible el entorno de trabajo del usuario final tiene que ser web.

En la elección del lenguaje de programación hemos tenido en cuenta la capacidad de soporte externo, tanto de la comunidad como de una empresa, la facilidad de diseño con IDE o plataforma completa de desarrollo (que abarque desde el diseño de las clases, hasta de la interfaz gráfica), la posibilidad de reutilización del código, y la mayor abstracción y parecido de las capacidades del lenguaje con la metodología orientada a objetos. Descartando los lenguajes mono-plataforma por que no tienen el interface web, tan solo nos quedan .NET, Java y PHP (de los que han sido tenidos en cuenta), pero como hemos descartado previamente .NET por ser tecnología de Microsoft y tener un coste total de adquisición de la propiedad muy alto, tan solo nos quedarían Java, y PHP. El servidor Java, una vez cargado, es de una ejecución más rápida, pero en cambio consume muchísimos más recursos de memoria que un servidor Apache con PHP5 corriendo sobre él. Tras una intensa sesión de trabajo con PHP5 sin haber llamado en ningún momento al servidor de aplicaciones Tomcat (servidor de Java) estos son los números:

```
USER       %CPU %MEM    VSZ   RSS TTY      STAT COMMAND
tomcat55   0.1  6.1 297396 77608 ?        Sl   /usr/bin/jsvc -user
tomcat55
www-data   0.0  0.9  37420 11808 ?        S    /usr/sbin/apache2 -k
start
```

Como puede comprobarse la diferencia de uso de la memoria es grandísimo, y más si se tiene en cuenta que tomcat es un servidor de aplicaciones java bastante ligero. En una comparativa parecida Glassfish llegaba al 10% de la memoria total del sistema sin haber hecho ninguna operación con él.

Después de haber considerado la opción de una aplicación web tenemos que pensar en el sistema operativo que nos servirá de base a todas estas opciones lo primero que se descartó fue cualquier plataforma basada en Windows tanto por el alto coste total de la implantación y mantenimiento para el cliente, como por las pocas ventajas económicas que aportaría para hipotética empresa que explotara la aplicación. El coste de la estación de trabajo donde se programaría la aplicación incrementaría su coste en 122,85€ solo por el licenciamiento de Windows Vista Bussines si lo adquirimos al comprar el ordenador nuevo o de 479,01€ si ya tenemos el ordenador con un sistema operativo diferente y queremos instalar el nuevo (como es el caso). Para aprovechar las ventajas del Windows utilizaríamos Visual Studio y programaríamos en .NET con lo que habría que adquirir la licencia del IDE de Microsoft. Esto supondría 914,73€ de licencia. En este precio no se ha incluido ningún antivirus, mantenimiento ni seguro por parte de otras empresas que incrementarían notablemente el precio. El coste del licenciamiento de la estación de trabajo es necesario, pues no se puede programar para un servidor Windows sin un cliente Windows. El coste del sistema operativo de servidor mínimo son 218,18€. Además de todo ello para hacer funcionar esta serie de características es necesario un hardware muy potente, pues, aún en el caso del sistema operativo de servidor Windows Server carga interface gráfica para facilitar la administración del mismo.

A la hora de elegir el servidor se plantea la duda entre los sistemas operativos *NIX Open Solaris o Linux. Mac se ha descartado por la obligatoriedad de hardware de Apple ® para hacerlo funcionar. Tanto Open Solaris como algunas distribuciones de Linux tienen programas de certificación que nos permitirían elegir entre profesionales a la hora de mantener los sistemas en funcionamiento. Los sistemas BSD se encuentran en el camino de lograr una certificación para administradores como producto. A fecha de hoy y a pesar de no dudar de la capacidad de los

profesionales dedicados al trabajo con este sistema operativo tenemos que descartarle por la ausencia de una formación especifica en el mercado para su administración.

De las distribuciones que tienen certificación se ha elegido finalmente Ubuntu Server 8.04 LTS. Ubuntu es la comunidad de usuarios que más ha crecido en el mundo de Linux en los últimos años tanto a nivel de usuario como de servidor. Red Hat y sus derivados (CentOS) también se tuvieron en cuenta, pero los precios de las licencias de Red Hat y el retardo en las actualizaciones críticas de seguridad de CentOS hicieron que Ubuntu fuese la elegida. Open Solaris tiene una comunidad de usuarios bastante pequeña, presenta incompatibilidades con las distribuciones de Linux, pues el núcleo del sistema y los aplicativos de control son diferentes y presenta un sistema de gestión de actualizaciones bastante lento y desorganizado en su descarga.

Puesto que ya no nos ata la tecnología propietaria de Windows, podemos elegir libremente y en igualdad de condiciones entre Oracle, PostgreSQL o MySQL. Aunque las características de Oracle y PostgreSQL son muy parecidas, tanto en potencia como en modularidad, capacidad de escalar con clusters o granjas de ordenadores, y lenguajes procedurales, y en ambas mayores que MySQL optamos por MySQL que tiene la propiedad de ser libre, y en entornos web bastante más rápida y con menos consumo de recursos que las otras dos. Además al estar bajo una capa de abstracción, si deseamos hacer la aplicación compatible con las otras dos bases de datos no tenemos más que cambiar los ficheros de configuración a la nueva base de datos elegida y generar los contenidos.

El lenguaje de programación elegido es PHP debido a la larga experiencia con el mismo del programador, así como por el amplio soporte de la comunidad, extensa y actualizada documentación y viveza en actualizaciones de seguridad. Pero sobre todo lo elegimos porque nos va a permitir trabajar rápido los prototipos para seguir el modelo de desarrollo en espiral a través del entorno de trabajo Symfony. Además este entorno de trabajo nos va a gestionar la conexión a la base de datos, generación de muchos formularios, separación de Modelo, Vista y Controlador (MVC).

4.1.6 Diseño de la infraestructura

A continuación se presentan los diferentes diagramas de la infraestructura sin virtualización (Ilustración 2) y virtualizada (Ilustración 3).

Se ha valorado la inclusión de diferentes sistemas de réplica de la base de datos. En el caso de que alguna de ellas fallara las demás responderían. Aún así se ha desestimado, pues esta opción requiere una configuración, seguimiento y mantenimiento mayores y preferimos delegar la seguridad de los datos en discos con sistemas de ficheros RAID invisibles tanto al SGBD como al sistema operativo. En el caso de un crecimiento exponencial de la cantidad de datos acumulados se optaría por una granja de servidores de gestión de base de datos con un sistema de balanceo de carga.

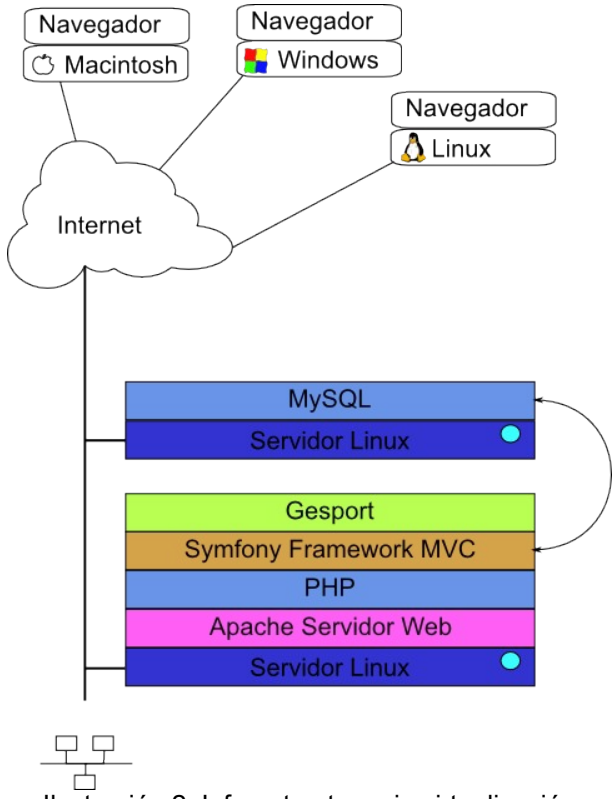

Ilustración 2: Infraestructura sin virtualización

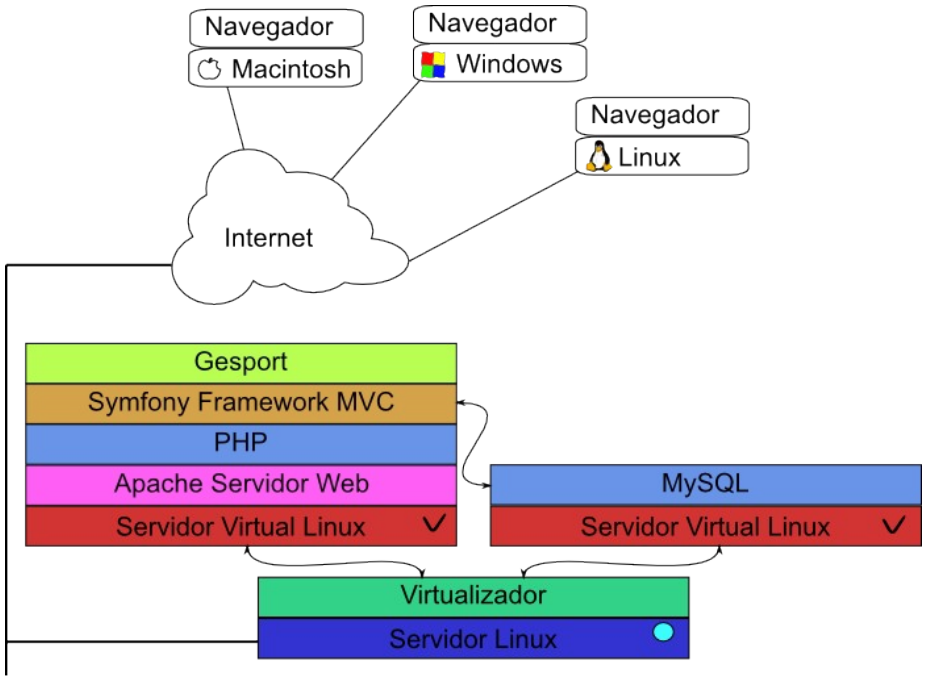

Ilustración 3: Infraestructura virtualizada

A pesar de lo representado en estos diagramas, que es el ideal de separación de datos y máquinas para ofrecer un servicio de calidad existen otras opciones que no dan mal resultado y que se basan en un solo servidor.

Estas soluciones se pueden conseguir con un solo servidor Linux con RAID, para duplicidad de datos, con las particiones realizadas de una manera que permitan la extracción de los datos importantes sin afectar al funcionamiento. Esto significaría tener una partición principal del sistema / con las aplicaciones de sistema, una partición de sistema /var, con los datos variables, entre los que incluimos los de la base de datos y una última partición /opt, donde instalaríamos la aplicación.

4.2 Diseño de las clases de objetos

Tomando como base las clases de análisis se realiza un proceso de diseño de las aplicación basada en clases de objetos efectivas en el programa. En este diseño ya se tienen en cuenta las propiedades del lenguaje de programación elegido. Además de las clases que intervienen en el dominio del problema habría que definir las clases auxiliares de almacenamiento de datos, de ventanas y paneles del interface gráfico de usuarios o de intercambio de mensajes complejos entre clases del dominio, pero al utilizar el patrón MVC estas clases nos van a venir dadas por la generación automática de interfaces, modelos de datos, o internacionalización. En las siguientes secciones se describen por tanto las clases del dominio del problema. Las clases de utilidad, interfaz gráfico o acumulación de objetos del mismo tipo (Listas, pilas, colas, etcétera), las proporciona symfony en base a las clases originales. En estos diagramas de clases se han omitido los "getters" y "setters", y otros accesos predeterminados a objetos contenidos para centrarnos tan solo en las operaciones que afectan al funcionamiento real de los elementos que componen el

sistema.

4.2.1 Diccionario de clases

4.2.2 Diagramas de clases

4.3 Diseño de la base de datos

El diseño de la base de datos está íntimamente ligado a los objetos. Se definen los objetos y la base de datos en el mismo momento de la fase de desarrollo. Genéricamente, cada clase de objeto se corresponde con una tabla de la base de datos. Esta definición se puede cambiar en los ficheros de configuración y utilizar una tabla para definir múltiples objetos o un objeto estar definido en múltiples tablas. En un principio no se hará uso de estas propiedades y, para simplificar, se asignará cada objeto a una tabla y en aquellos con referencias múltiples simultaneas (relaciones n:n) una tabla intermedia que será representada en el diagrama de clases por una clase funcional.

4.4 Diseño del interfaz gráfico de usuario

El interfaz gráfico lo define la capa de vista. Esta capa nos muestra los elementos visuales, pero para darle un aspecto más agradable y conveniente trabajaremos con una hoja de estilos que definirá colores, posiciones, tamaños y tipografías. Gesport tendrá un aspecto de programa de gestión estándar con un menú superior, un panel principal con los elementos que pertenezcan al trabajo seleccionado en el menú superior, y una barra de estado donde se indicarán la corrección de la conexión al servidor, el nombre del usuario conectado, el estado de seguridad en la transmisión de los datos y algún otro dato que se considere importante, dependiendo del rol de usuario.

4.5 Diseño de las pruebas

Las pruebas que se van a realizar para comprobar el correcto funcionamiento del programa son las siguientes.

- Dada la especificación de las clases se van a probar cada uno de los métodos de clase con valores al azar, con valores fuera del rango admitidos de los elementos, valores extremos, y valores de otros tipos. Como las clases llevan la persistencia por defecto no se va comprobar la validez de los valores en la base de datos, sino a comprobar la correcta generación de excepciones.

- Se realizarán pruebas simultaneas de varias ejecuciones de clases con persistencia para comprobar la integridad de los datos en tiempo de ejecución, bloqueos de acceso y modificación de informaciones, y concurrencia de acceso a datos.

- Se realizarán pruebas del funcionamiento correcto del programa. Para ello se crearán datos de ejemplo desde cero en una base de datos con los datos justos para funcionar sin problemas de integridad referencial. Con esta prueba también se comprobará la seguridad de la instalación, pues el programa en su primera ejecución creará un usuario administrador con una contraseña predeterminada.

- Se realizará otra prueba de concurrencia de acceso a datos con el programa completo para comprobar que el aumento de funcionalidades no afecta al rendimiento de las clases.

- Se realizarán pruebas de funcionamiento del programa con monitores de consumos de procesador, memoria, referencias circulares, consumo de ancho de banda y otras herramientas de optimización de rendimiento de la aplicación. Para ellos se utilizarán las herramientas top, apachetop, y la lectura de los logs de Symfony.

5. Manuales de usuario y administrador

5.1 Manual de instalación y administración del administrador

Advertencia: para el seguimiento de ciertos puntos de este manual es necesario conocer los conceptos básicos de administración de sistemas Linux, operaciones con bases de datos, configuración básica de servidores y conexión remota de ordenadores. Basándonos en esos principio describiremos la instalación completa en un sistema determinado con unos comandos concretos. Este manual describe la instalación de la aplicación de servidor Gesport en una máquina con arquitectura x86 o x64, utilizando la distribución Ubuntu Server LTS 8.04.1 del sistema operativo GNU/Linux, con MySQL en el mismo servidor sin virtualizar, apache 2 como servidor web en la carpeta /opt, recomendada por Debian para software opcional que no provenga de paquetes. Se puede adaptar a otras distribuciones de Linux e incluso a Windows con IIS.

5.1.1 Instalación del sistema operativo de servidor y los componentes necesarios para el funcionamiento de Gesport.

El siguiente manual se refiere a una máquina Intel. Vamos a instalar un sistema operativo Ubuntu Server con todos los programas necesarios para hacer funcionar Gesport: Servidor web apache con módulo de PHP 5 para hacer funcionar Symfony y módulos Include y Rewrite para ocultar al navegador los ficheros donde está el programa, servidor de bases de datos Mysql y por último el programa Gesport.

5.1.1.1 Instalación del sistema operativo

Para instalar Ubuntu Server LTS 8.04.1 descargamos la imagen del CD de instalación de la siguiente dirección http://ftp.sh.cvut.cz/MIRRORS/ubuntu-releases/hardy/ubuntu-8.04.1-server-i386.iso. Esta dirección corresponde a un servidor "mirror" (espejo) . En la web de Ubuntu http://www.ubuntu.com/getubuntu/download tenemos acceso a diferentes formas de descarga de este mismo sistema operativo.

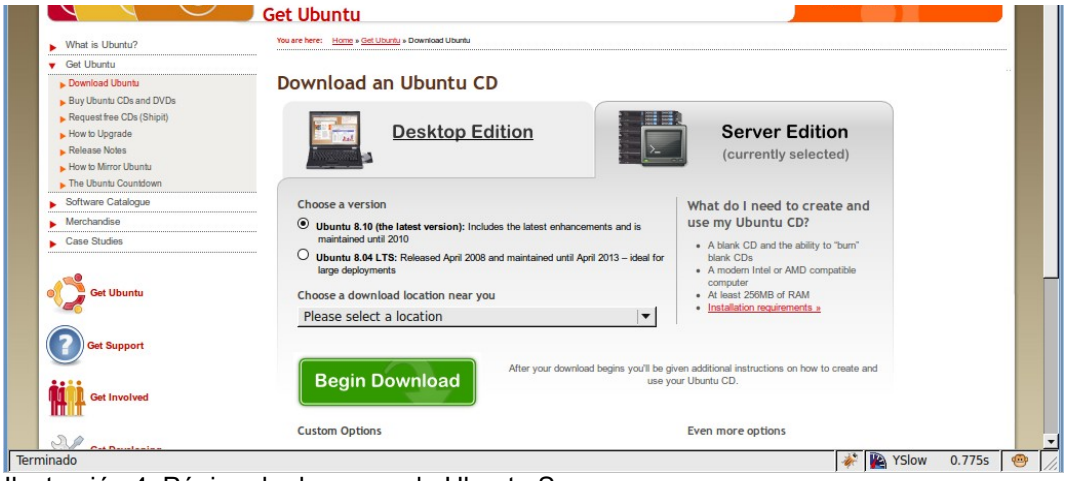

Ilustración 4: Página de descarga de Ubuntu Server

Una vez descargada la imagen la grabamos en un CD con un software de grabación de CDs disponible para nuestro sistema operativo. En el caso de que dispongamos de alguna herramienta de verificación de resumen de la imagen descargada la podemos utilizar para comprobar que el resumen de nuestro CD y el de la web de Ubuntu coinciden antes de comenzar el proceso de grabación. Esto nos asegura que el CD es el mismo que la empresa Canonical ha creado y que no tiene ningún problema de datos perdidos o vulnerabilidades provocadas. Los resúmenes están disponible en https://help.ubuntu.com/community/UbuntuHashes en el caso que nos ocupa es 7232c6004ba438890cd09aded162dc8e. Como vemos en la imagen del software de grabación de CDs K3B coincide:

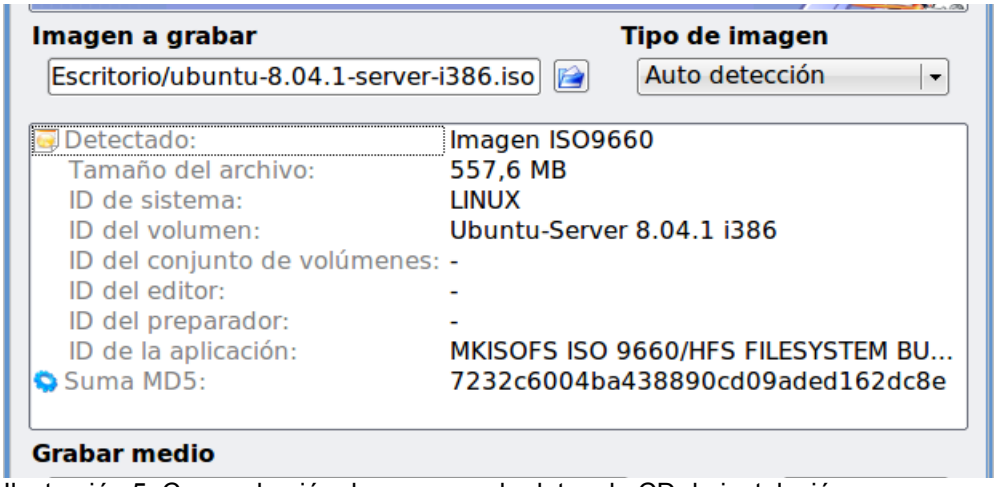

Ilustración 5: Comprobación de resumen de datos de CD de instalación

Cuando se ha grabado y etiquetado, para evitar confusiones, pasamos a instalarlo en la máquina que hará las funciones de servidor. Para ello pulsamos el botón de encendido y, siguiendo las recomendaciones del fabricante de la placa base, accedemos a la configuración de la BIOS, en el caso de que sea un sistema con BIOS, para configurar el orden de arranque y establecer el CD-ROM como dispositivo de primera lectura. Una vez configurado colocamos el CD de instalación de Ubuntu, guardamos y reiniciamos.

En el caso de que este ordenador tenga un sistema EFI, es posible que la configuración para acceder a arrancar desde el CD-ROM se consiga pulsando la tecla C en el momento previo a la carga del sistema operativo con el CD ya dentro.

Sea cual sea el caso una vez arrancado el sistema operativo nos encontramos con unos menús que nos permiten elegir las opciones de la primera ejecución previa a la instalación. Valga como ejemplo el primero, en el que se nos pregunta por el idioma (Ilustración 6). Una vez seleccionado el idioma con el teclado podemos seleccionar la acción a realizar, que en este caso será "Install Ubuntu Server". A partir de aquí comienzan varios procesos como la copia de los ficheros del CD de instalación al disco duro de nuestro servidor o la detección de los dispositivos que tenemos instalados.

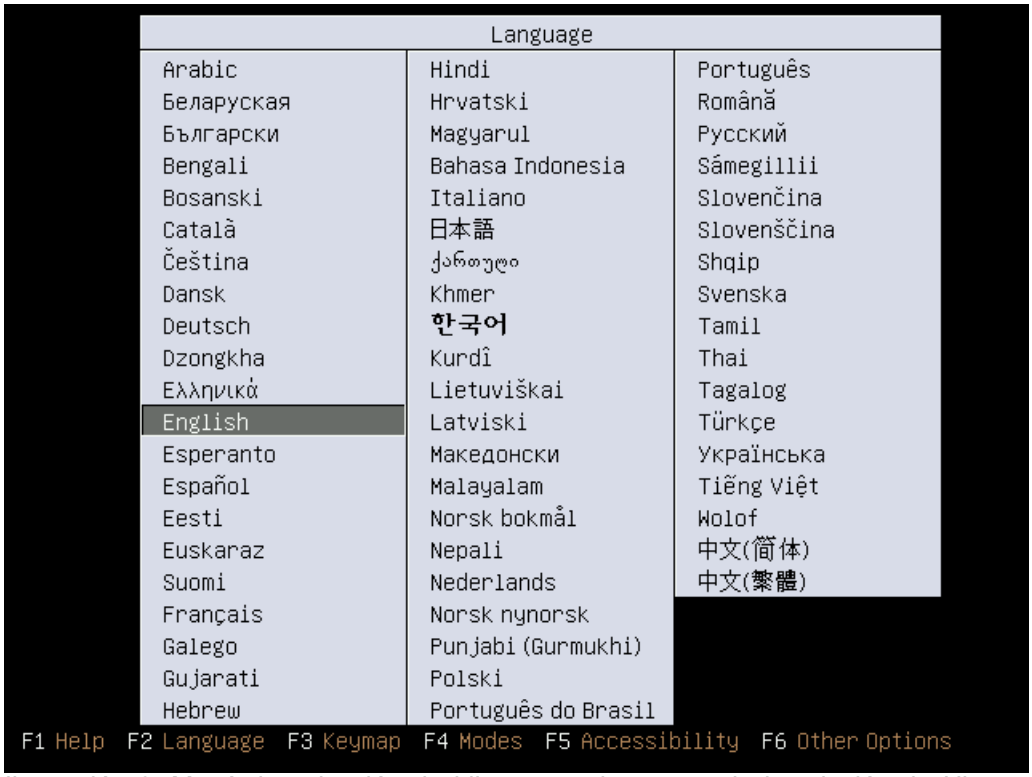

Ilustración 6: Menú de selección de idioma en el proceso de instalación de Ubuntu Server

Una vez terminado esto configuramos el nombre de la máquina, "gesport" en este caso, la zona horaria, y como vamos a distribuir el espacio de nuestro disco. Para esta distribución de espacio el instalador nos da varias opciones. El "particionado guiado" o manual. Con el guiado tenemos la opción de usar LVM, que permite una redistribución posterior de nuestro espacio. Si el servidor va a estar dedicado por completo a la aplicación la recomendación es usar "particionado manual". El "particionado guiado" normal que usa todo el disco no tiene en cuenta posibles recuperaciones de sistema con particiones diferentes de datos y ejecutables. El particionado LVM nos obliga a usar un sistema de ficheros determinado, pues al ser su principal característica la movilidad de volúmenes, el sistema de ficheros tiene que tener la característica de poder crecer por delante y por detrás físicamente en el disco duro.

Los requerimientos mínimos para un funcionamiento de Gesport en cuanto a distribución de disco son los siguientes:

- Para los ficheros de arranque, sistema operativo, y ejecutables necesitamos una partición de 1Gb, Primaria, al principio del disco (excepto EFI, que se explicará a continuación), con el punto de montaje "/" y la marca de arranque activada.
- Para los ficheros de datos de las aplicaciones vamos a crear otra partición . Esta partición depende en tamaño de la capacidad de generar datos del club deportivo o empresa para el que la realizamos. El sistema Gesport introduce todas las imágenes y diagramas de ejercicios en la base de datos con la clase Picture, luego el espacio real necesario para el

funcionamiento correcto de la aplicación depende únicamente de este directorio. Esta partición se monta en "/var" y ya es indiferente que sea primaria o lógica. Las tablas de particiones de UNIX soportan hasta cuatro particiones primarias, por lo que no se dan los problemas de Windows con el arranque.

- Para que la instalación de la aplicación sea independiente de la instalación del sistema en "/" se pueden poner particiones diferentes para "/etc" donde se almacenan los ficheros de configuración y "/opt" donde debemos instalar Gesport siguiendo las directrices de Debian y Ubuntu para instalación de software ajeno a la distribución. "/etc" es un directorio donde solo hay ficheros de texto y ocupa bastante poco. En ocasiones las soluciones a problemas de seguridad vienen con pequeñas modificaciones de los ficheros de configuración contenidos en este directorio omitiendo servicios o partes de la configuración vulnerables. Es por esta razón que tendremos nuestro ficheros de configuración de Apache, PHP y MySQL guardados en una copia de seguridad para restaurar mediante comparación de diferencias en los originales de las actualizaciones. En el caso de que el fichero de configuración pueda nutrirse de ficheros externos deberemos configurar los servicios de nuestra aplicación en estos ficheros y no en los principales.
- Para una mejorar sustancial del rendimiento es también recomendable dejar parte del espacio como memoria de paginación en disco. En el caso de Linux hay que destinar una partición de tamaño determinado en tiempo de instalación. Esta partición no tiene punto de montaje y se denomina Swap o área de intercambio. Esta partición debería tener una capacidad entre el tamaño de memoria RAM instalado y el triple de este tamaño.

La marca de arranque activa viene marcada en la primera partición por la letra B (del inglés Boot) detrás de la capacidad de la misma.

Al seleccionar "Finalizar el particionado y escribir los cambios en el disco" comienza el proceso de instalación.

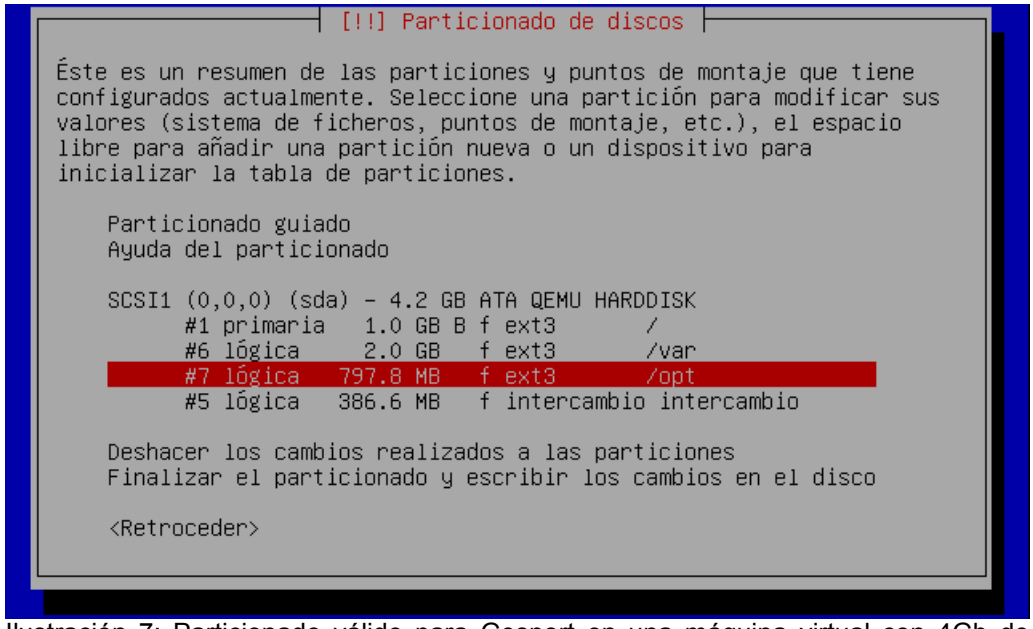

Ilustración 7: Particionado válido para Gesport en una máquina virtual con 4Gb de espacio y 128 Mb de RAM

Tras el proceso de instalación de todos los ficheros de la instalación base comienza el proceso de configuración, donde crearemos un usuario con su contraseña, que en el caso de Ubuntu es también el administrador del sistema. Con el usuario instalado el programa de instalación configura el proxy, si es que usamos uno, y luego nos da a elegir los componentes del sistema que vamos a instalar. Seleccionamos los siguientes:

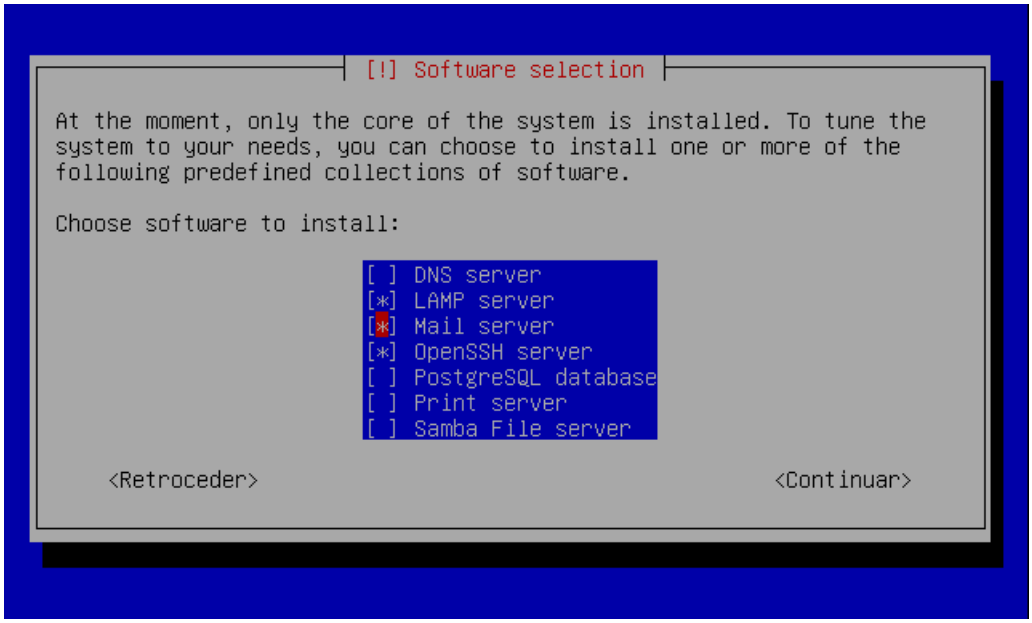

Ilustración 8: Selección de programas a instalar

Los paquetes de software seleccionados son:

- LAMP server: Servidor compuesto por Linux, Apache, Mysql y PHP o Perl.
- Mail server: para notificaciones automáticas del programa.
- OpenSSH server: para poder realizar una administración remota del servidor.

En el proceso de instalación tendremos que poner clave al usuario administrador, a la base de datos MySQL, el nombre de dominio adecuado al servidor, el nombre del usuario administrador y la configuración correcta para conectarse a las diferentes redes del entorno del servidor.

Es posible que, para que el funcionamiento sea totalmente correcto, tengamos que comunicar al administrador de la red donde estamos el nombre de usuario y dirección IP para que configure los DNS con el nombre de dominio adecuado.

5.1.1.2 Instalación de componentes

El comando principal de instalación de componentes en Ubuntu es apt o aptitude. Ambos utilizan las dependencias contenidas en los paquetes .deb, para asegurar el correcto funcionamiento del paquete instalado, y el sistema de repositorios, que permite seleccionar las fuentes de donde obtenemos el software. La diferencia es que aptitude es más completo, pues al instalar una dependencia de algún programa que queremos recuerda, al intentar desinstalar, las dependencias que instaló, y las elimina si no son necesarias. Por esta razón utilizaremos aptitude.

Los programas necesarios no instalados se consiguen así:

```
sudo aptitude install php5-cli
```

Aunque en un principio todos los elementos necesarios para el funcionamiento de gesport podrían parecer ya instalados con esta configuración vamos a utilizar algunas herramientas que no están disponibles por defecto, y que sin ser necesarias podemos usar para facilitar nuestro trabajo:

```
sudo aptitude install phpmyadmin
sudo aptitude install rsync
```

5.1.1.3 Configuración de Apache

Hay que activar los módulos Include y Rewrite de Apache para un correcto funcionamiento de Symfony y, por lo tanto, de Gesport. Para realizar esta tarea tan solo hay que crear los enlaces simbólicos de la carpeta modules-available a modules-enabled:

```
cd /etc/apache2/mods-enabled

sudo ln -s ../mods-available/include.load include.load
sudo ln -s ../mods-available/rewrite.load rewrite.load
```

El módulo Include nos permite incluir ficheros de fuera del directorio principal, lo que hará que Gesport pueda incluir los ficheros necesarios para su funcionamiento desde fuera del directorio web. De esta manera cualquier fallo en el servidor que devuelva el código hará inaccesibles los ficheros que se incluyan como referencia en el fichero principal, salvaguardando, al menos las contraseñas de acceso a nuestra base datos guardada en algún fichero PHP.

El módulo Rewrite es el artífice de las direcciones agradables que nos mostrará el navegador en vez de las secuencias con interrogaciones y nombres y valores de datos.

5.1.2 Instalación de Gesport

5.1.2.1 Instalación de ficheros de Gesport y puesta en marcha de web

En la siguiente dirección http://forja.rediris.es/frs/?group_id=431 están los ficheros descargables. Existen dos tipos, "complete" y "for PEAR". La descripción de la instalación se refiere a la instalación "complete" ya que no se recomienda usar el PEAR de Symfony en entornos de producción.

Así pues, accedemos al servidor, ya sea de manera presencial o a través de ssh de manera remota. Descargamos, instalamos y configuramos la última versión de Gesport-complete con los siguientes comandos, como administradores de sistema:

```
#cd /opt

#wget http://forja.rediris.es/frs/download.php/1069/gesport-
complete-0.0.9.tar.bz2

#tar xvjf gesport-complete-version.tar.bz2
```

Una vez realizado esto tenemos que indicar al servidor apache donde está la carpeta que

tiene que ejecutar y que nombre de dominio vamos a tener. Para ello creamos con nuestro editor favorito un nuevo fichero con un nombre indicativo del sitio.

```
#nano /etc/apache2/sites-available/gesport
```

Lo editamos de la siguiente manera, guardamos y salimos del editor:

```
NameVirtualHost ip_del_servidor:80

<VirtualHost ip_del_servidor:80>
  ServerName gesport
  DocumentRoot "/opt/gesport/web"
  DirectoryIndex index.php


  <Directory "/opt/gesport/web">
   AllowOverride All
   Allow from All
  </Directory>
</VirtualHost>
```

Una vez que tenemos el fichero en la carpeta de sitios disponibles creamos un enlace simbólico hacia este fichero desde la carpeta de sitios activos. El número indica el orden de carga, 000 es para default que carga los servidores básicos, así que por lo menos tendremos que ponerle 001. Si el servidor aloja más aplicaciones podremos elegir el orden de carga de cada una en el inicio de apache. Posteriormente reiniciamos el servidor web:

```
#ln -s /etc/apache2/sites-available/gesport /etc/apache2/sites-
enabled/001-gesport

#/etc/init.d/apache2 restart
```

Para comprobar el funcionamiento correcto de esta parte de la configuración de Gesport tendremos que configurar un ordenador cliente con la dirección IP de la máquina asociada al nombre de servidor y escribir este nombre en el navegador. Como resultado nos debe dar un error, puesto que aún no hay base de datos ni datos dentro.

5.1.2.2 Configuración de la base de datos y creación de datos base

Una vez funcionando el directorio de la aplicación de una manera correcta procederemos a crear un usuario en la base de datos para la ejecución del programa, una base de datos e introduciremos los datos básicos para empezar a funcionar con la aplicación.

Para crear ese usuario podemos hacerlo directamente en la linea de comando o con el programa phpyadmin si lo instalamos anteriormente. En esta documentación se describen los comandos para bash, ya que el programa lo facilita en sobremanera.

Los siguientes comandos parten del punto anterior y describen lo siguiente:

1. Salir de modo superusuario

```
#exit
```

2. Entrar en la consola de mysql como superusuario, puesto que vamos a dar permisos:

```
$mysql –u root –p
```

3. Creación del usuario y permisos básicos

```
mysql>CREATE USER 'gesport-app'@ 'localhost' IDENTIFIED BY
'**********';

mysql>GRANT USAGE ON * . * TO 'gesport-app'@'localhost' WITH
MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0
MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0 ;
```

4. Creación de la base de datos

```
mysql>CREATE DATABASE `gesport-app` ;
```

5. Otorgar permisos al usuario

```
mysql>GRANT ALL PRIVILEGES ON `gesport-app` . * TO 'gesport-
app'@'localhost' WITH GRANT OPTION ;
```

6. Tras identificarnos de nuevo como superusuario, modificar el ficheros de Gesport que guardan la información de la clase Propel de acceso a la base de datos

```
$sudo su

password: ******

#nano /opt/gesport/config/databases.yml
```

En el que tenemos que configurar los parámetros database, username y password respetando los espacios a la izquierda para representar la jerarquía del fichero.

7. Y el fichero que guarda los datos de conexión con la base datos

```
#nano /opt/gesport/config/propel.ini
```

Donde tendremos que configurarlas siguientes lineas

```
propel.database.createUrl  = mysql://gesport-app:******@localhost/

propel.database.url        =
mysql://gesport:******@localhost/gesport-app

propel.output.dir              = /opt/gesport
```

8. A partir de aquí solo hay que introducir los datos y limpiar la caché, por si hemos hecho pruebas y hemos metido la pata en la configuración, y tenemos el sistema funcionando

```
#php /opt/gesport/symfony propel-insert-sql

#php /opt/gesport/batch/load_data.php
```

```
#php /opt/gesport/symfony clear-cache
```

Los datos para entrar como administrador son:

Nombre de usuario: admin

Clave: gesport

Se pueden cambiar dentro del programa, pero eso viene reflejado en el manual de usuario.

5.1.2.3 Configuración de DNS y acceso a la página

Con todos los pasos anteriormente realizados tenemos la aplicación funcionando. Ahora quedaría configurar el servidor de nombres de dominio (DNS) para que los el acceso a Gesport suceda de la manera adecuada. Esto significa que al poner el nombre de servidor configurado en el paso de puesta en marcha de la web debería aparecer. Para ello hay varias alternativas:

- Si la ejecución es local (entorno de pruebas) tendremos que dar un alias a nuestra propia dirección modificando el fichero /etc/hosts.
- Si va a funcionar en una red local, tendremos que añadir el registro con el nombre asignado en el fichero de configuración de apache, al servidor DNS de la red interna apuntando a la IP del equipo que ejecuta Gesport.
- Si va a funcionar en un servidor abierto a internet tendremos que registrar el dominio correspondiente en un registrador oficial de dominios. Una vez registrado tendremos que hacer que apunte a la/las IPs de nuestro servidor. Dependiendo del tipo de enlace registrado en el registrador de dominios puede ser necesario un sistema DNS interno del servidor para resolver este dominio.

5.1.2.4 Personalizar Gesport

Si con los pasos anteriores ya tenemos funcionando Gesport perfectamente hay que decir que tenemos que configurar algunos otros parámetros para, por ejemplo, que los correos electrónicos lleguen a la dirección de nuestro administrador, en vez de a la por defecto, o para cambiar la barra de título, aspecto de la página, o, en el caso de que estemos actualizando entre versiones, parar la aplicación para evitar daños en los datos.

Los ficheros principales para hacer estas tareas son, desde el directorio donde está instalado Gesport:

- apps/frontend/config/app.yml, donde vienen configuradas ciertas variables como los correos electrónicos de los administradores, webmaster y contacto comercial.
- apps/frontend/config/settings.yml. Este fichero es el corazón de la aplicación. En el campo available de settings podemos escribir off para dejar el sitio temporalmente apagado y cuando volvamos a estar preparados para ponerlo en funcionamiento ponerlo a on.
- apps/fronten/config/view.yml es el fichero que describe la salida por pantalla de nuestra aplicación. Podemos ver el título de la ventana de navegador las hojas de estilos que utiliza y las palabras clave y descripción de la cabecera de la página, entre otras.
- web/css/ es la carpeta donde se sitúan los ficheros css que hemos definido en view.
- web/images es la carpeta donde están la imágenes de Gesport.

5.2 Manual de usuario

5.2.1 Solicitud de acceso al programa

5.2.2 Configuración de un usuario

5.2.3 Usuario entrenador, director técnico y profesional técnico del club

5.2.4 Usuario deportista

5.2.5 Usuario junta directiva

5.2.6 Usuario tesorería

5.3 Manual de desarrollador

5.3.1 Normativa de colaboraciones en Gesport

5.3.2 Control de versiones

5.3.3 Estructura del árbol del ficheros y documentación

5.3.4 Autoría del software y licencia

6. Índices de tablas, dibujos e ilustraciones del proyecto

Relación de ilustraciones y tablas del proyecto

Índice de Ilustraciones

Índice de tablas

7. Bibliografía

7.1 Sobre la teoría del entrenamiento deportivo

7.2 Sobre el diseño de software

Requirements Engineering [Ian Sommerville & Pete Sawyer, 1997].

7.3 Sobre el lenguaje de programación PHP

7.4 Sobre la persistencia de los datos en el software

7.5 Sobre el diseño de interfaces gráficas web

7.6 Sobre los sistemas operativos y su manejo como administrador

8. Índice de anejos

Relación de material adicional incluido en el soporte óptico

Se ha evitado el uso de caracteres extraños en todo momento para garantizar la compatibilidad del CD con todos los sistemas operativos y todas las configuraciones locales de caracteres.

8.1 Modelo de la aplicación en UML para NetBeans

El modelo de la aplicación está contenido en la carpeta Diseno/Diseno. La carpeta es un proyecto UML válido para NetBeans IDE 6.*. Debería funcionar con cualquier versión del programa a partir de la 6.0. Detallo que se ha realizado con la 6.1 por si hubiera algún problema de compatibilidad.

La carpeta Diseno/PruebasDiseno contiene las clases generadas con el generador de Java de NetBeans para su editor de diagramas de UML.

8.2 Código fuente de la aplicación

8.3 Documentación consultada para la realización del programa

En la carpeta "Documentacion" se han reunido todos los documentos electrónicos consultados durante la realización del programa. Aparecen todos en la bibliografía, pues aunque son fuentes documentales digitales, páginas web de comunidades de usuarios de programadores y artículos escritos sin formato de papel para diferentes sitios de internet, los he considerado fuente fundamental de información a la hora de realizar este proyecto dada la poca experiencia, casi ninguna fuera del aula de prácticas, que el alumno tenía en el trabajo con el lenguaje de programación Java. A su vez he comprobado que las licencias de la documentación distribuida permitan este medio para su difusión.

8.4 Programas usados para la realización del programa

8.5 Logotipos del programa en formato vectorial y mapa de bits

8.6 Pruebas de las tecnologías utilizadas (muestreo de imágenes e Hibernate)

8.7 Manual de usuario y de administrador.

Se han realizado impresiones externas de los manuales de la aplicación para su publicación en la web del proyecto. Se incluyen como documentos anejos por el cambio en la paginación y portadas, ya con la imagen corporativa asociada al software.