UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

# *Aplicación para la gestión de Organización Diagnóstica de Atención Temprana (ODAT)*

---

**MEMORIA PRESENTADA POR:**

***Lucía Tortosa Alcaraz***

GRADO DE *Ingeniería Informática*

**Tutor:** Manuel Llorca Alcón

**Convocatoria de defensa**: *09/2017*

# Contenido

## Índice de figuras



## Índice de tablas en Anexo I: Análisis de la base de datos



## Índice de tablas en Anexo II: Diseño gráfico de la Interfaz

# 1. Introducción

Con la informatización de sistemas actual, los programas de gestión se han vuelto fundamentales para estructurar y manejar datos eficientemente. Una aplicación para la gestión es un programa informático que, funcionando como una base de datos interactiva, almacena y muestra los datos de forma concreta. Se tratan de programas con múltiples ventajas como mejorar la gestión del tiempo, tener toda la información correctamente clasificada, facilidad a la hora de manipular los datos y se tratan de proyectos personalizados para cada caso concreto.

Por desgracia, hoy en día a pesar de las ventajas de los programas de gestión, no se tienen los suficientes recursos para crear este tipo de proyectos a gran escala, sobre todo en países poco desarrollados, empresas con recursos limitados o entidades públicas.

En la presente memoria se desarrolla un programa de gestión para Psicología y Medicina Infantil, usando la metodología de la Organización Diagnóstica de Atención Temprana (ODAT). Dicha metodología se basa en analizar el entorno de infantes de 0 a 6 años en los siguientes ámbitos: social, familiar, biológico y entorno.

El objetivo final será la obtención de una herramienta que permita clasificar factores de riesgo y trastornos en el desarrollo. Al mismo tiempo tiene la posibilidad de obtener estadísticas para el estudio científico en el área de Psicología Infantil y qué mejoras se pueden utilizar en el ámbito social para cada caso particular.

La gestión en este campo es muy importante, ya que permite diagnosticar problemas anticipadamente permitiendo intervenir usando recursos y apoyos disponibles, evitando que la situación empeore.

## 1.1. Antecedentes

En 1995 se creó la Federación Estatal de Asociaciones de Profesionales de la Atención Temprana (GAT), un grupo formado con diversos profesionales especializados con el objetivo común de elaborar el Libro Blanco de la Atención Temprana que fue publicado en el año 2000.

Esta idea resurge ante la necesidad de crear una clasificación común diagnóstica para la intervención temprana, por lo que surge la comisión encargada para este fin: la Organización Diagnóstica para la Atención Temprana (ODAT). En el periodo de 2003 y 2004, se redacta la segunda versión de ODAT.

A partir de 2004, se han estado creando métodos para la informatización de la ODAT, en colaboración con la Universidad Politécnica de Valencia se crea una herramienta de gestión especializada que permita su uso en servicios de maternidad, Centros de Desarrollo Infantil y Atención Temprana, servicios sociales y servicios educativos.

Sobre 2012, se creó la segunda versión definitiva de una plataforma online para la gestión diagnóstica temprana también realizado por alumnos de la misma Universidad. Se trata de un programa muy completo pero presentaba algunos problemas como la necesidad de conexión a Internet o la dificultad para realizar su correcto mantenimiento debido a la tecnología usada.

Se intentó solucionar algunos de estos problemas distribuyendo el programa en formato CD para que llegase al mayor número de usuarios posible.

A partir de 2016 en la Comunidad Autónoma de Andalucía, se empezó a distribuir otro programa de gestión con los mismos fines, que introduce gráficos con comparativas, entre otros.

## 1.2. Objetivos

El objetivo principal del presente proyecto, es realizar una nueva versión del programa anterior que cumpla los siguientes requisitos:

- Que sea principalmente un programa de escritorio que se pueda utilizar en múltiples plataformas.
- El uso de un lenguaje de programación actual, ampliamente usado y que facilite la creación de versiones posteriores.
- Gestionar información sin necesidad de acceso a Internet.
- Crear expedientes médicos, informes y diagnósticos.
- Que permita a los usuarios administradores modificar todas las variables de información de forma dinámica.
- Una solución de seguridad apta para datos catalogados como nivel alto por la LOPD.

Por lo que la primera tarea a realizan va a ser analizar el código del programa anterior, escrito con RubyOnRails[1], para traducirlo en la medida de lo posible al nuevo entorno. Algunos de los datos se pueden reutilizar, como gran parte de la estructura de la base de datos.

De ningún modo se busca copiar el funcionamiento lógico. Sólo se va a analizar el funcionamiento actual para obtener conocimientos suficientes, con el objetivo de realizar una nueva versión que cumpla con los objetivos iniciales y cualquier otra incidencia o necesidad que se encuentre durante el proceso.

Tras un análisis inicial de la interfaz, se añade el objetivo de diseñar de nuevo las interfaces para que estás sean visualmente más atractivas, con una estructura sencilla e interactiva, aprovechando las amplias posibilidades de los programas de escritorio.

Se han añadido otros objetivos en la fase de desarrollo, algunos recientemente añadidos son:

[1] (RubyOnRails)

- Posibilidad de tener un servidor centralizado para compartir datos entre los clientes de diferentes centros y que sirva como copia de seguridad.
- Creación de una versión móvil, especialmente diseñado para Tablet.
- Posibilidad de guardar en el servidor archivos y poder anexarlos en formato PDF.

## 1.3. Motivación

La principal motivación para realizar dicho proyecto, se basa en brindar ayuda social a un sector tan vulnerable como son los niños en situaciones de riesgo y familias con problemas sociales.

Una buena actuación en situaciones de riesgo es fundamental para la mejora en el desarrollo del menor, ya que o supera el problema o se palia notablemente. Por lo que este programa permite ordenar los datos de forma efectiva facilitando el trabajo de los profesionales y evitando errores de gestión.

A continuación se enumeran las razones adicionales:

- Ilusión personal de crear un proyecto de software libre y gratuito. Creando una comunidad accesible para cualquiera que quiera ayudar en la mejora, traducción o distribución del programa.
- Otorgar a entidades sanitarias y sociales una herramienta que les ayude a gestionar su trabajo en este campo concreto.
- Distribución libre de una colaboración de profesionales de distintos sectores.

No se va a permitir el uso comercial de esta obra o derivados, ya que se perdería una de sus principales motivaciones: el tener una herramienta potente accesible sin condiciones.

# 2. Organización Diagnóstica de Atención Temprana

## 2.1. ¿Qué es?

La Organización Diagnóstica de Atención Temprana[2] es una comisión encargada de ordenar los trastornos en el desarrollo y situaciones de riesgo en niños de 0 a 6 años. Parte de distintos ejes, para analizar el entorno de los menores y sus familias, tanto para el diagnóstico como para la intervención y asignación de recursos para la mejora la situación y entorno del menor.

[2] (Federación Estatal de Asociaciones de Profesionales de Atención Temprana, 2005)

Permite por tanto, facilitar la toma de medidas preventivas, contrastar formas de actuación, diseñar investigaciones, realizar estudios epidemiológicos y establecer un lenguaje común entre todos los profesionales que intervienen en Atención Temprana.

## 2.2. ¿Cómo funciona?

Los diversos ejes que intervienen en esta metodología se encuentran clasificados en tres niveles. El nivel 1 y 2 sirven para clasificar los factores de riesgo y los trastornos en el desarrollo, permite diseñar investigaciones, organizar las observaciones clínicas y facilitar la toma de medidas preventivas. El nivel 3, no contiene diagnósticos y se realiza en una fase posterior, y sirve para analizar los recursos, necesidades y apoyos de los infantes y su familia para poder realizar una intervención en busca de mejorar la situación actual.

Por lo que en primer lugar, se realiza un informe médico tomando todos los datos del infante, su familia y entorno. En segundo lugar se crea uno o varios diagnósticos analizando los motivos de la consulta, causas de origen, recursos del centro y un diagnóstico usando los elementos de los niveles 1 y 2. El último paso en el diagnóstico, es seleccionar los recursos del nivel 3.

Por último, se permite exportar los datos analíticos del centro, para poder realizar posteriores investigaciones y realizar estudios. Esto es de suma importancia ya que la Organización Diagnóstica de Atención Temprana es susceptible a modificaciones, y necesita estar constantemente actualizada.

## 2.3. Justificación

La ODAT tiene como objetivo crear una codificación diagnóstica especializada en Atención Temprana. Normalmente se solía recurrir a codificaciones existentes, creadas y diseñadas para adultos y muchas veces con distintos sistemas de clasificación. Por lo que surgía la necesidad de crear un sistema especializado en los rangos de edad en los que trabaja la Atención Temprana.

La población susceptible a AT es diversa y heterogénea, pues en ella se sitúan niños con deficiencias bien definidas y conocidas, niños con trastornos emocionales o de conducta, y también, niños que por sus trastornos o dificultades precisan de una clasificación que contemple las manifestaciones propias de los primeros meses y años de vida.

## 2.4. Objetivos

Se ha puesto en manifiesto la necesidad de disponer de una organización diagnóstica con criterios unificados. Los objetivos de la ODAT son los siguientes:

- Elaborar un instrumento útil que sirva para clasificar los factores de riesgo y los trastornos del desarrollo.
- Establecer un lenguaje común entre los distintos profesionales que intervienen en la AT.
- Aglutinar en la misma clasificación todos los aspectos que intervienen en AT (biológico, psicológico y social).
- Desarrollar estudios epidemiológicos, estableciendo la prevalencia de los distintos trastornos del desarrollo y situaciones de riesgo, a nivel estatal y de las diversas comunidades autónomas.
- Diseñar investigaciones.
- Organizar las observaciones clínicas.
- Facilitar la toma de medidas preventivas.

Para la gestión de recursos, encontrados en el nivel III de ODAT, se establecen estos objetivos:

- Dar una visión global de la situación concreta y de las intervenciones programadas.
- Diseñar la planificación de recursos para el niño y su familia desde la interdisciplinariedad.
- Recoger las necesidades en cuanto a infraestructuras de servicios de AT, a nivel sectorial, autonómico y estatal, que posibiliten una intervención de calidad.
- Contrastar formas de actuación en los distintos trastornos del desarrollo y establecer las más efectivas.

## 2.5. Finalidad

La finalidad principal del presente programa informático es llevarlo a países de Latinoamérica, lugares donde se hace imprescindible el uso de programas de estas características. Si bien la situación no es óptima en muchos países, en Latinoamérica se encuentran muchos factores de riesgo al tener un entorno con pocos recursos, mucha pobreza, falta de medios para servicios sociales, educativos y sanitarios.

Se estima que un 28% de la población en estos países, se encuentra en la pobreza, lo que asciende a un total de 167 millones de personas en esta situación. De estos, se estima que aproximadamente 3,2 millones de niños menores de 5 años, no han sido siquiera registrados al nacer[3], por lo que no tienen ni siquiera acceso a los servicios de educación y sanidad.

Aun así, hay una gran disposición por parte de la comunidad médica de mejorar el sistema sanitario actual, es decir, hay voluntad pero no los medios. También hay que

[3] (Agudo, 2017)

tener en cuenta que la situación está mejorando, por lo que sería de gran ayuda tener a su disposición un programa de gestión especializado en Atención Temprana.

Dicho programa, también ayudaría a crear estadísticas sobre la situación particular de cada zona de un país, permitiendo mejorar la asignación de recursos en distintas comunidades. Se va a realizar una distribución completamente gratuita del programa, por lo que cualquier interesado en el sector, va a tener acceso a su descarga.

En caso de tener una buena acogida, también se distribuirá en España. El problema se encuentra en que la legislación actual no ayuda en la distribución de un programa que almacene datos de nivel alto de seguridad y la poca asignación de recursos en ámbito sanitario y social.

En principio, se busca tener cuanta más distribución mejor, se va a ofrecer como un software libre en el cual cualquier persona va a tener acceso, permitiendo modificaciones, adaptaciones y traducciones para cada necesidad concreta sin poner en peligro la integridad de los datos de centros ajenos. El objetivo principal es mejorar una necesidad social común.

# 3. Análisis del problema

## 3.1. Análisis de requisitos

Se va a utilizar un lenguaje de programación ampliamente utilizado, que permita su uso en múltiples plataformas, es de suma importancia que sea un lenguaje actual.

En el aspecto de diseño de interfaz gráfica, se deben de tener en cuenta los siguientes puntos:

- Menú principal simple y completo, de forma que facilite el acceso a todos los puntos del programa.
- El uso de tablas con prácticamente el mismo diseño y estructura para no confundir al usuario que muestre los datos principales y permita su visualización, modificación y eliminación.
- Una mejora sobre las tablas es añadir subtablas que permitan acceder a más información, siendo una implementación difícil pero viable.
- Siempre que sea posible, visualizar la descripción de objetos que puedan llevar a confusión.
- Los formularios para crear informes y documentos van a tener los datos correctamente ordenados y etiquetados para facilitar la introducción de información.
- Mejorar la visibilidad utilizando un árbol para mostrar los elementos de diagnóstico, ya que se tratan de datos jerárquicos.

El aspecto lógico del programa tiene que seguir los siguientes puntos:

- Arquitectura cliente servidor, en el cual los clientes se puedan conectar a un servidor para sincronizar y compartir datos dentro del mismo centro.
- Parte cliente:
    - Contendrá una base de datos ligera, para tener independencia y poder acceder a la información de modo local, sin requerir acceso a Internet para comunicarse con el servidor.
    - Cada vez que se conecte con el servidor se sincronizarán los datos que no se hayan podido actualizar.
    - La base de datos contendrá la información del mismo centro.
    - La información guardada de forma local y no sincronizada, debe estar claramente diferenciada para permitir la actualización o inserción en el servidor.
- Parte del servidor:
    - Gestionará todos los clientes.
    - Debe contener una base de datos completa y eficiente, que contendrá la información de todos los clientes.
    - Implementará seguridad poniendo un límite de accesos al servidor, de forma que evite ataques informáticos al servidor.

Hay que hacer énfasis en la seguridad para cumplir la ley de protección de datos, ya que se trata de información confidencial con un nivel de seguridad alto:

- Las bases de datos, tienen que tener un acceso autorizado.
- Se tiene que evitar en la medida de lo posible, enviar datos a través de Internet.
- En caso de enviar datos a través de Internet, estos deben de ser ilegibles por terceros que puedan interceptar el mensaje.

## 3.2. Análisis de las soluciones

### 3.2.1. Lenguaje *de programación*

El lenguaje de programación tiene que ser ampliamente usado y actualmente en uso, esto es así para evitar que el programa quede obsoleto. Por el funcionamiento del programa, que permite acceso independiente del servidor, se rechazan aquellas soluciones que funcionen sobre plataforma web.

Entre los lenguajes más usados para programas de escritorio, se encuentra C con todas sus variantes y Java. Tienen bastantes puntos en común y es relativamente sencillo traducir de un lenguaje a otro teniendo conocimientos ya que pertenecen al mismo tipo de lenguaje de programación basado en objetos. Ambos lenguajes permiten realizar interfaces para móvil, por lo que otros lenguajes ampliamente usados para programas de escritorio como Python quedan descartados.

Sobre el lenguaje de C y todas sus variantes, la más actual es C# y permite el uso móvil con Xamarin para exportar a varias plataformas móviles. Mientras que Java es el lenguaje nativo para la plataforma móvil más utilizada con diferencia: Android. Ambas son buenas opciones, pero aunque C# es más potente que Java, y permite exportar a múltiples plataformas aunque puede dar problemas en sistemas Unix. Mientras que Java cuenta con un enorme abanico de librerías libres, solo requiere la máquina virtual para funcionar sobre cualquier dispositivo.

### 3.2.2. Diseño de la interfaz

El menú principal va a contener todas las categorías del programa mediante botones, con iconos representativos para mejorar el diseño del programa y los datos importantes del proyecto como el icono, el nombre del programa y el icono del ministerio de sanidad. De esta forma la información está clasificada de forma clara e intuitiva.

Las tablas con información van a contener botones, para que permitan la visualización, modificación y eliminación de los distintos elementos. También van a tener el mismo diseño, por lo que se ha realizado una clase aparte, que controle el diseño de la interfaz de la tabla. Ésta se divide en un renderizado de cabecera para controlar el color de fondo y de texto, renderizado de las celdas para la introducción de botones y una tabla auxiliar que controlará todo el proceso.

Para la creación de subtablas dentro de una tabla, existen varias soluciones complejas, ya que las tablas no pueden por si mismas unir celdas sin modificar todo el código de la interfaz de tablas o la introducción de múltiples elementos[4]. Por lo que la solución adoptada pasa por los siguientes puntos:

- Añadir un botón que permita desplegar la subtabla, o esconderla.
- Crear filas vacías debajo de la fila desplegada, con el tamaño de la subtabla.
- Añadir una tabla en el layout, calculando dinámicamente la posición teniendo en cuenta que la altura de las celdas siempre son las mismas.
- Introducir la subtabla en el layout anteriormente creado.
- Para esconder la subtabla, en caso de desplegar otra fila o pulsar el botón de ocultar subtabla, va a eliminar el layout con su contenido y eliminar las filas vacías.

A esta conclusión se ha llegado después de múltiples pruebas, al comprobar que se permite la introducción de elementos dentro de cualquier elemento gráfico, en cualquier posición dando un buen resultado final.

[4] (Java2s - Fuente original obsoleta)

Para que se muestre constantemente la visualización de descripciones, se han usado ToolTipText, el cual consiste en mostrar un pequeño texto cuando se pone el ratón sobre un elemento concreto. Para ello se ha creado una renderización para obtener la descripción del elemento, y para que lo muestre. Esto se utiliza para las cajas desplegables de información en la cual se ha creado un render adicional para que muestre esta información.

Para los formularios, se utilizan bordes con título para agrupar los distintos campos de forma ordenada, permitiendo introducir la información de forma estructurada.

Para la visualización de datos jerárquicos, utilizando un árbol, se ha usado un código sencillo en el cual introduce checkbox para permitir múltiples selecciones[5], el cual se ha modificado para que se ajuste a nuestras necesidades. Se ha modificado para permitir varios niveles y que se introduzca la información ordenadamente tal como se obtiene de la base de datos.

Se va incluir el "Anexo II: Diseño de interfaz gráfica", para poder ver algunas de estas implementaciones.

### 3.2.3. Lógica del programa

El programa se puede estructurar en distintas dimensiones:

- Hay un servidor, el encargado de almacenar los datos creando una copia de seguridad, controlando los accesos a una base de datos centralizada. Este servidor puede ser totalmente centralizado ofreciendo servicio a todos los clientes de distintos centros que deseen conectarse. Otra opción sería un servidor descentralizado, uno por centro dando servicios solo a sus clientes, por autonomía ofreciendo servicio a los clientes de centros pertenecientes a la comunidad.
- Un cliente, en el cual existen tres tipos:
    - Usuario: requiere para el primer inicio de sesión conectarse con el servidor para crearlo. Puede ver los datos generales del servidor pero no modificarlos. Solo permite crear informes y diagnósticos.
    - Usuario administrador: Puede modificar los datos precargados de la base de datos del servidor, y por tanto modificar al resto de usuarios al sincronizar los datos de todos los usuarios. Normalmente se requiere un usuario administrador por cada centro.
    - Usuario de uso local: Un nodo independiente, no se puede conectar a ningún servidor pero tampoco lo requiere para funcionar ya que accede únicamente a la base de datos local del programa cliente teniendo libertad para su modificación.

[5] (Zukowski)

### 3.2.4. Bases de datos

Respecto a la lógica del programa para el cliente, se dispone de un amplio abanico de sistemas gestores de bases de datos que se pueden utilizar como MySQL, Access, PostgreSQL o SQLite. Las más ligeras y sencillas de utilizar son Access y SQLite. Access permite un cifrado fácil y seguro pero solo se usa en Windows y no permite su uso en otras plataformas mientras que SQLite es la base de datos utilizada en plataformas móviles y aunque su encriptación es más complicada.

Para el servidor, se va a utilizar una base de datos más potente apta para servidor como Microsoft SQL Server o SQL Server. Como se ha comentado anteriormente, el uso de múltiples plataformas, permite acceder a un público más amplio, por lo que se va a utilizar SQL Server.

### 3.2.5. Comunicación cliente-servidor

La comunicación entre el cliente y el servidor, se va a realizar mediante Sockets, esta tecnología permite las comunicaciones mediante mensajes, controlar los accesos, gestionar los mensajes y se puede utilizar sobre múltiples lenguajes de programación.

Otra opción que ha tenido que ser descartada es el uso de SQL una base de datos servidora, con múltiples conexiones para que los clientes puedan actualizar sus datos cuando tengan acceso. Sin embargo, esta solución aunque es más sencilla, no es posible, ya que esta tecnología ha tenido anteriormente problemas de seguridad.

### 3.2.6. Sincronización

Los datos en el servidor, en caso de no estar conectado a Internet, se van a guardar utilizando identificadores con números negativos, y si se encuentra en el servidor, se va a utilizar números positivos. Esto se realiza así para poder diferenciar claramente los datos sincronizados y los que no. Para realizar esto, se va a introducir una tabla adicional en la base de datos del cliente, para almacenar el identificador del dato no sincronizado, la tabla a la que pertenece, y si es una operación de inserción o modificación.

En el mismo momento en el que se conecte al servidor, mediante usuario y contraseña, el cliente se sincronizará con el servidor utilizando la tabla auxiliar, enviando los datos a actualizar con el servidor, y éste responderá con el nuevo identificador si es una inserción o con un mensaje de confirmación si es una modificación.

## 3.3. Solución propuesta

### 3.3.1. Java

El lenguaje utilizado finalmente va a ser Java. La razón principal por la que finalmente se va a utilizar, es por la cantidad de librerías existentes que facilitan enormemente el trabajo y la gran versatilidad de sus componentes gráficos, a diferencia de otros lenguajes.

Esto es de gran importancia, porque sólo para crear una librería, como por ejemplo un componente gráfico para introducir fechas de forma cómoda, versátil y con un diseño atractivo, se necesitarían múltiples líneas de código y clases especializadas.

Además cumple con el propósito de ser multiplataforma, estar actualmente en uso e incluso cada vez es más usado, y la posibilidad de portarlo a móvil de forma nativa aunque sea sólo con Android. El uso de aplicaciones móviles nativas, es mucho más potente, cosa muy importante si se desea encriptar una base de datos móvil ya que usa muchos recursos.

### 3.3.2. SQLite

Para aplicaciones cliente, se va a utilizar SQLite, por ser ligero y se puede integrar sin necesidad de instalar ningún programa y por ser la base de datos principal en aplicaciones móviles. Por lo que se trata de una solución multiplataforma.

Existen todas las herramientas necesarias para trabajar con esta base de datos: el conector para Java y un listado de programas para encriptar la base de datos.

### 3.3.3. MySQL

Para aplicaciones del servidor, finalmente se ha elegido MySQL[6], principalmente porque es la primera referencia en cuanto a bases de datos. Es lo suficientemente potente para funcionar sobre un servidor con múltiples peticiones.

Se trata de una base de datos ampliamente usada, con una enorme documentación al respecto razón por la que se ha descartado PostgreeSQL. También es multiplataforma, así que no importa sobre qué Sistema Operativo se ejecute el programa servidor eliminando la opción de Windows SQL Server.

[6] (MySQL)

### 3.3.4. Sockets

Se van a utilizar sockets para la comunicación cliente-servidor. Se trata de una tecnología muy compleja de utilizar, pero es perfectamente compatible con los requisitos especificados por la LOPD para datos de nivel alto de seguridad, entre los que se encuentran los datos clínicos.

Los sockets utilizados son de tipo TCP, que aunque no son tan sencillos de usar como los UDP (también llamados datagramas), ofrecen más seguridad y menos pérdida de paquetes a través de la red. Abriendo un puerto en el router donde se conecte el servidor, va a permitir a los clientes acceder mediante Internet.

Permite enviar secuencias de datos bastante largas para la comunicación de hasta 64KB[7]. Obviamente no se van a utilizar cadenas tan largas, pero es interesante tenerlo en cuenta para encriptar datos personales, ya que van a ocupar mucho más espacio.

La ayuda para crear los sockets, se ha sacado de un antiguo programa personal realizado en 2013[8]. Se trata de una aplicación cliente-servidor para uso como chat, permite también pasar archivos mediante pequeños paquetes. El código se encuentra disponible en un repositorio creado con el fin de tener los programas personales creados accesibles.

## 3.4. Análisis de seguridad

Uno de los problemas más importantes a solucionar, es la seguridad y el cumplimiento de la ley de protección de datos[9]. Una de las primeras soluciones propuestas es que los clientes realicen consultas directamente con la base de datos del servidor, pero ha sido descartada. La Agencia de Protección de Datos, no permite el uso de tecnologías que hayan tenido graves fallos de seguridad con anterioridad[10].

Posteriormente, se iba a utilizar Sockets cifrados para la comunicación, llamados Sockets SSL, pero esto ha sido descartado al añadir una gran complejidad al programa para obtener certificados de parte del cliente. Por esta razón, se ha decantado por Sockets.

El uso de sockets en la comunicación, tiene el problema de envío de mensajes con texto plano, por lo que la solución aportada es cifrar los datos con datos sensibles en el cliente, para ser descifrados por el servidor y viceversa. Para esto se va a utilizar un método de cifrado punto a punto, usando AES cifrado por bloques de 128 bits[11], tratándose de un cifrado muy seguro y que se requiere mucho tiempo para descifrarlo. Con esto se cumpliría con las obligaciones de la LOPD ya que cualquiera que intercepte el mensaje, no va a poder leerlo.

[7] (Soós, 2009)
[8] (Tortosa Alcaraz, 2017)
[9] (Gonzalez, 2016)
[10] (Crespo, 2016)
[11] (Chinchilla, 2014)

Para conectarse con el servidor, se va a almacenar una contraseña con el método SHA2, concretamente SHA 256 bits[12]. SHA2, aunque tiene su antigüedad, continúa siendo bastante seguro, al igual que se va a bloquear al usuario con 3 intentos, esto añadirá más seguridad a la cuenta. Otro mecanismo para evitar accesos indeseables, es realizar una pequeña pausa, cada vez que se falle en el intento de sesión.

Con este funcionamiento, el único con acceso a la base de datos centralizada, es el propio programa servidor, de forma que trabaja en local evitando poner en peligro la integridad de los datos de todos los clientes.

Para evitar problemas, los mensajes también tendrán un patrón de funcionamiento, almacenando todos los accesos y sincronizaciones de los clientes, para mayor seguridad. En caso de un mensaje con una estructura desconocida, se cortará la comunicación inmediatamente con el cliente denegando el acceso al servidor, tomándose esto como una incidencia de seguridad.

Otro mecanismo de seguridad, será que el servidor abrirá tantas conexiones como usuarios haya en el sistema, pudiéndose modificar en caso de necesidad, para que no se puedan conectar más usuarios de los que existan.

Se va a tener un control de todos los accesos del cliente al servidor, y las modificaciones en la base de datos centralizada. De forma que se tenga constancia de quien, cuando y qué ha modificado.

## 3.5. Análisis de protección de datos

Las soluciones adoptadas, cumplen con los requisitos que especifica la ley de protección de datos para datos de nivel alto, ya que los datos clínicos son considerados sensibles.

Antes de distribuir el programa, vamos a tener que cifrar las bases de datos del cliente, para que nadie pueda acceder a la base de datos integrada.

Gracias a la base de datos en local, no se necesitan consultas repetitivas de datos, ya que solo va a enviar la información necesaria encriptada a través de Internet para la sincronización.

Sí que sería conveniente, generar el fichero para la LOPD, así como un manual de uso para todos los clientes en España tanto para servidores como clientes.

[12] (baeldung)

## 3.6. Análisis de internacionalización

Probablemente, en versiones posteriores se tenga que implementar la internacionalización, ya que el objetivo del trabajo final es que se distribuya en cuantos más sitios mejor. Por ahora no se aplica ya que los clientes van a ser de habla hispana.

En caso de requerirlo, sólo se va a necesitar modificar ligeramente en el código las cadenas literales y la base de datos para añadir múltiples idiomas y regiones. El lenguaje de programación utilizado permite realizar esta tarea fácilmente tanto para versión de escritorio como móvil.

## 3.7. Planificación

El proyecto se ha desglosado en varias etapas aproximadas, con un total de 11 semanas de trabajo, y aproximadamente 6 horas de lunes a viernes es un total de 330 horas, por el momento.

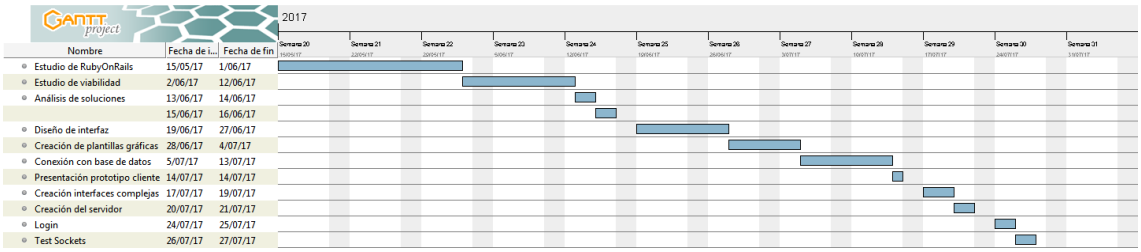


Figura 1: Diagrama de Gantt hasta la actualidad.

A continuación se describen las tareas realizadas:

- Estudio de RubyOnRais: Se trata del framework utilizado en la anterior versión del programa, funciona con el lenguaje de programación Ruby. Se ha requerido el estudio de su funcionamiento ya que no tenía conocimientos previos y necesitaba conocer los profesos y funciones del programa. Se ha empezado con esta etapa semanas más pronto, pero no se ha tenido en cuenta, ya que sólo se ha podido sacar información útil y clara más o menos a mediados de Mayo. En esta etapa se ha realizado el análisis de entidades de la base de datos y el diseño de interfaz gráfica anterior.
- Estudio de viabilidad: Antes de continuar, había que tener como mínimo el lenguaje de programación, tecnologías que valide la LOPD, y qué herramientas podían ser útiles para el proyecto.
- Análisis de soluciones: Comprobar que las soluciones tomadas anteriormente son válidas para los requisitos de la aplicación. En esta etapa se ha cambiado de opinión en la seguridad relacionada con la encriptación.

- Creación de la base de datos: Se ha creado la base de datos en SQLite a partir del archivo de RubyOnRails llamado schema.rb y se ha conectado con la aplicación.
- Diseño de interfaz: Aunque tenía una idea más o menos concisa de cómo iba a ser gráficamente el proyecto, se ha tardado aproximadamente una semana en crear la interfaz que cumpla con las necesidades y sea nítida. En esta fase se han realizado las conexiones con la base de datos.
- Creación de plantillas gráficas: Se ha creado la plantilla de las tablas y subtablas para poder ser utilizadas en la mayoría de frames que listen la información. El código de las listas se ha cambiado para crear una plantilla para que muestren la descripción de cada elemento al poner el ratón encima.
- Conexión con la base de datos: En esta etapa, se han realizado las consultas más complicadas de la aplicación, consultas relacionales con las tablas creadas de las relaciones muchos a muchos.
- Presentación prototipo cliente: Se ha enseñado la aplicación al cliente, le parece que tiene un diseño adecuado, reúne los requisitos principales, y en general está muy satisfecho con el trabajo.
- Creación de interfaces complejas: Usando las consultas relacionales, se ha creado el árbol gráfico con checkbox para tener
- Creación del servidor: Al tener conocimientos y un programa que sirve de ejemplo, la creación del servidor ha sido muy sencilla y rápida. Para las pruebas, se ha usado la misma base de datos en SQLite.
- Login: Creación de usuarios y autenticación con la base de datos local (requisito de la LOPD que la aplicación tenga control de acceso incluso en el mismo equipo) y a la base de datos del servidor.
- Test Sockets: Comprobar que es viable la comunicación con sockets. Se ha realizado una sincronización sencilla.

Confirmada la viabilidad del proyecto y las soluciones escogidas, se va a proceder a realizar un análisis de lo que resta de proyecto:

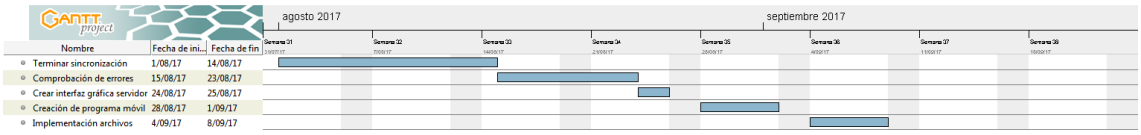


**Figura 2: Diagrama de Gantt de las tareas futuras.**

El resto de las tareas, se ha programado realizarlo en 6 semanas, con un total de 180 horas extras. La definición de las siguientes tareas es la siguiente:

- Terminar sincronización: Con un total de 60 horas para poder sincronizar el resto de las tablas del cliente con la base de datos del servidor. Si se realiza bien esta tarea, la siguiente fase de comprobación de errores será más corta o directamente se podrá suprimir.

- Comprobación de errores: Opcional, se ha asignado un total de 42 horas a la comprobación de errores en la sincronización.
- Creación de la interfaz gráfica del servidor: Sólo 12 horas, ya que hay un prototipo creado. Aquí sería interesante cambiar la base de datos a MySQL.
- Creación del programa móvil: Se han asignado 30 horas para crear el programa para Tablet. No va a ser un cliente muy completo, ya que su función va a ser poder realizar consultas a domicilio y va a reutilizar código lógico del programa de escritorio.
- Implementación de archivos: Al igual que se pueden crear diagnósticos e informes, se va a poder adjuntar archivos. Se han asignado 30 horas, ya que hay que poder mandar estos archivos al servidor mediante sockets, cosa que dificulta la tarea.

Aún faltaría realizar múltiples tareas de comprobación de errores con múltiples usuarios.

# 4. Diseño de la solución

## 4.1. Análisis de las herramientas

Como se ha descrito anteriormente en el Análisis de Soluciones, se va a utilizar Java[13] como lenguaje operativo por la gran cantidad de librerías de software libre disponibles, facilitando el desarrollo y dando más posibilidades de diseño.

Se va a necesitar instalar el Java Developer Kit (JDK) que contiene todas las herramientas para empezar a programar con este lenguaje de programación.

En este apartado, también se van a analizar el resto de herramientas que han hecho posible la creación de este proyecto.

### 4.1.1. Entorno de desarrollo integrado

Para poder programar sobre el lenguaje de programación escogido, vamos a necesitar un entorno que sea amigable y facilite el desarrollo del proyecto.

Principalmente existen dos entornos principales para Java que son Eclipse y Netbeans. Realmente ambos son muy parecidos en cuando prestaciones, pero se ha escogido finalmente Netbeans[14] por tener un abanico más extenso de widgets.

[13] (Java)
[14] (Netbeans)

### 4.1.2. Soporte de Base de datos

Para la base de datos en SQLite se va a utilizar SQLiteBrowser[15], un software gratuito, intuitivo y muy ligero que va a facilitar la tarea de crear la base de datos. Las bases de datos creadas van poder ser en múltiples plataformas y es más sencillo de usar que la interfaz por comandos oficial.

Para la comunicación del programa de escritorio, la única opción disponible es el conector JDBC-SQLite[16]. Funciona de forma similar al conector oficial de MySQL, es sencillo e intuitivo de usar, aparte permite devolver el identificador de una consulta insertada con identificador autoincremental.

### 4.1.3. JCalendar

Una de las mayores dificultades al trabajar con bases de datos, es conseguir la misma codificación para introducir fechas. Un cambio en el formato, puede provocar fácilmente un fallo del sistema y gestionar los posibles errores se convierte en una ardua tarea.

Por esta razón, se ha decidido introducir una librería fiable y segura, que permita la introducción de la fecha en un formato unificado mediante una pequeña interfaz. Se llama JCalendar[17], es un software libre y es ampliamente usado, es considerada una referencia a la hora de introducir widgets de fechas en Java.

### 4.1.4. iTEXT

El programa requiere la exportación de los informes a PDF, la librería más actual y novedosa para este fin es iTEXT[18]. Se trata de una librería de la cual se puede obtener mucha información para la creación de texto a PDF desde el lenguaje de programación Java. Tiene el inconveniente, de tratarse de un programa con licencia AGPL, pero no afecta en este caso particular ya que el presente proyecto se va a distribuir como software libre, estando exento de pagar ninguna licencia.

### 4.1.5. JFreeChart

Una de las mejoras de este programa, va a ser la introducción de gráficos para comparar cómodamente los datos. JFreeChart[19] es una librería de código abierto especializada en la creación de múltiples tipos de gráficos, actualmente no hay ninguna otra librería especializada para este fin. Las funciones que van a interesar en el proyecto son los gráficos circulares y de barras.

[15] (SQLiteBrowser)
[16] (L.Saito, 2017)
[17] (Toedter)
[18] (iText)
[19] (JFreeChart, 2014)

### 4.1.6. Apache Commons

Una parte fundamental del proyecto, es que se requiere trabajar con datos muy confidenciales, por lo que hay que tener en cuenta la seguridad. Aunque la creación de funciones hash o encriptación y desencriptación de datos se puede realizar de forma manual, es un proceso laborioso.

Apache Commons[20] permite realizar todas las codificaciones de forma sencilla e intuitiva. El componente utilizado en el proyecto es Codec, y permite un amplio abanico de posibilidades respecto a la encriptación y desencriptación, así como la creación de funciones de tipo SHA2 utilizando sólo una línea de código.

## 4.2. Arquitectura del software

### 4.2.1. Estructura del proyecto

Con un total por el momento de 100 clases en el proyecto servidor, se hace imprescindible estructurar el proyecto, aunque a pesar de estar estructurado, modificar un proyecto masivo es bastante complicado.

Por lo que se va a utilizar un patrón de arquitectura de software para almacenar toda la información del proyecto. Como se trata de un programa de gran envergadura, es fundamental que los datos se encuentren correctamente ordenados dentro del proyecto, para facilitar modificaciones o ampliaciones en el código.

El patrón usado para realizar el proyecto es Modelo-Vista-Controlador (MVC), ya que es una de las formas más sencillas de estructurar un programa de grandes dimensiones con acceso a bases de datos. Se estructura de la siguiente forma:

- Modelo: Representación lógica de cada elemento de la base de datos.
- Vista: Representación gráfica de las interfaces. Realiza acciones llamando al controlador.
- Controlador: Va a ser el encargado de acceder a la base de datos y realizar todas las consultas y modificaciones. Para guardar la información va a usar el modelo.

[20] (Apache Commons, 2014)

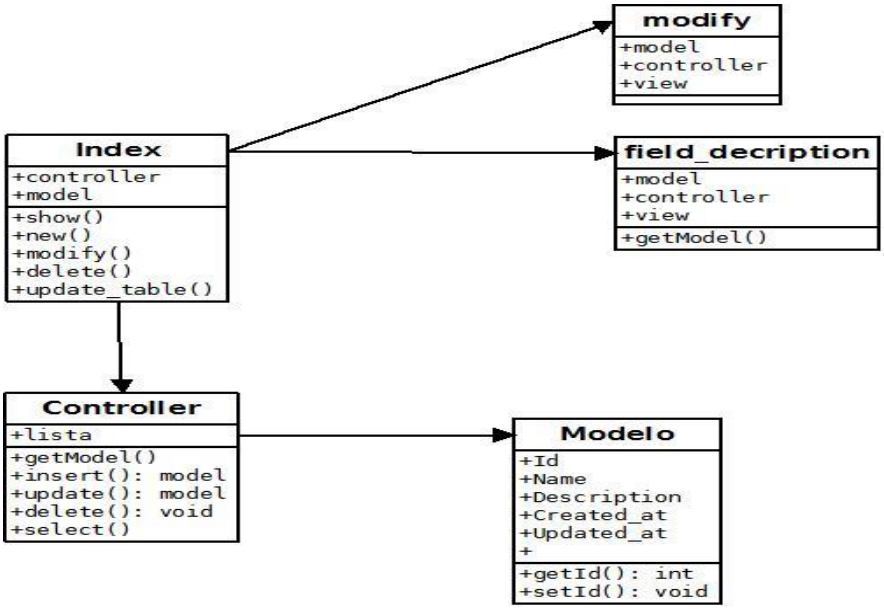


Figura 3: Diagrama sencillo de clases con patrón MVC

Contiene algunas modificaciones respecto al patrón original, como por ejemplo el control de interfaces lo va a realizar únicamente la vista, al igual que la gestión de acciones gráficas como apretar botones, llamando para ello al método correspondiente del controlador. El modelo no gestiona, solo almacena la información de forma estructurada. Por último el controlador básicamente lo que hace es realizar la comunicación con la base de datos.

Aparte, se van a almacenar los distintos elementos gráficos como las imágenes y a generar plantillas. Las plantillas van a guardar la lógica del diseño, para mantener la armonía del diseño, esto quiere decir que se va a utilizar el mismo diseño para las listas y tablas para no causar confusión en el usuario.

También se van a utilizar dos paquetes distintos, uno para la sincronización y otro para abrir o cerrar la base de datos.

### 4.2.2. Base de datos

La base de datos, va a ser muy parecida a la del programa anterior, a continuación de muestra las principales relaciones entre las tablas:

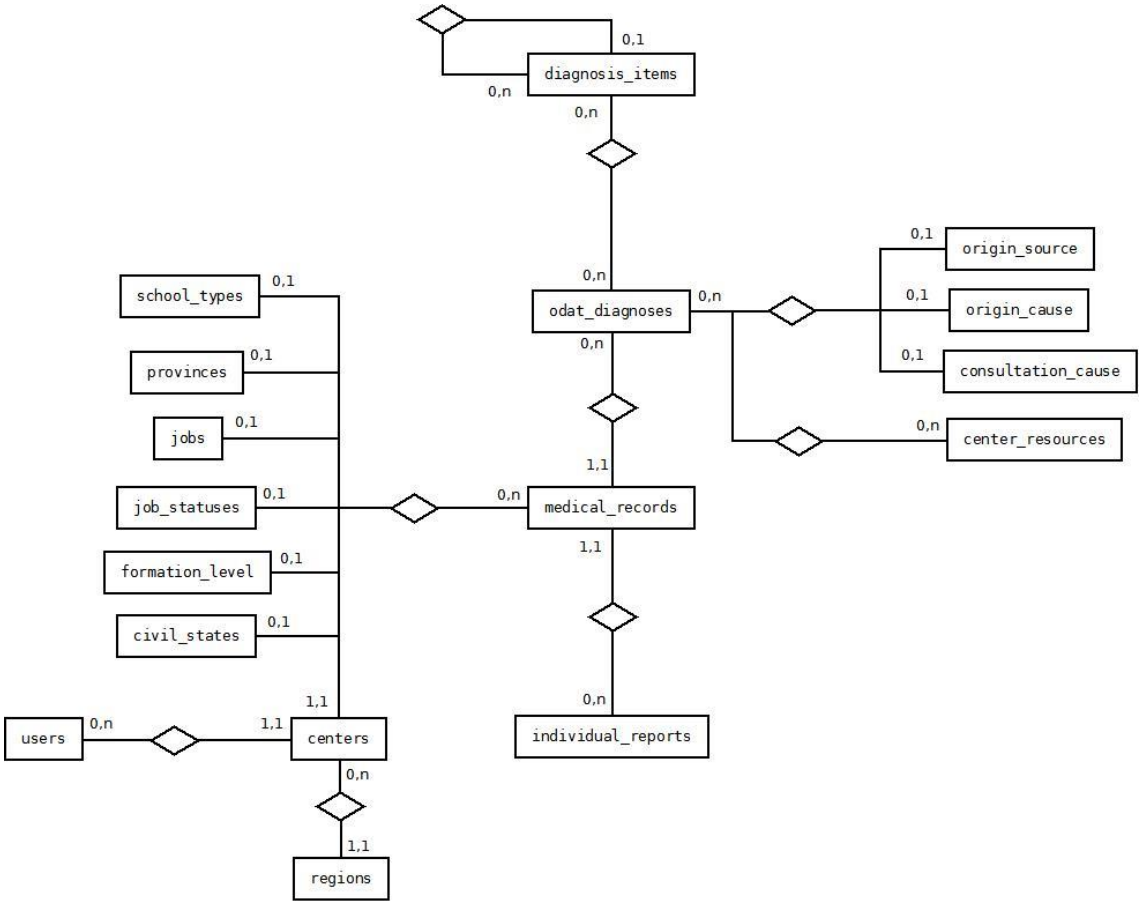


Figura 4: Entidad-Relación de la base de datos

Se van a añadir tablas adicionales para las relaciones muchos a muchos.

Se ha reorganizado la relación jerárquica, de forma que en vez de funcionar como un árbol que aunque es muy eficiente para bases de datos, con el funcionamiento actual del programa se convierte en un método complejo. Se ha construido una relación padre-hijo, contabilizando los hijos del padre. Este método aunque no es el más eficiente, como esta parte del programa va a tener pocas o nulas modificaciones, apenas va a influir en el resultado final.

Esta parte coincide tanto en el servidor como en el cliente. A continuación se describe resumidamente las funciones de cada tabla:

- Medical_records: Informe médico del paciente, con información personal y familiar.
- Individual_reports: Informe personalizado del paciente, donde se almacena qué datos del informe médico mostrar para su impresión o exportación a PDF.
- Odat_diagnoses: Almacena los datos de la consulta del paciente y diagnósticos.
- Origin_source: Almacena que persona ha derivado al menor a Atención Temprana.
- Origin_cause: Almacena los motivos médicos de la derivación

- Consultation_cause: También llamado causas de consulta, o la razón por la que han derivado al menor a Atención Temprana.
- Center_resources: También llamado recursos de centro, almacena todos los recursos disponibles para el tratamiento para el menor.
- School_types: Almacena los tipos de colegio, para poder indicar en qué tipo de escuela se encuentra el menor.
- Provinces: Guarda los nombres de provincias
- Jobs: Almacena los distintos trabajos que puedan tener los padres.
- Jobs_statuses: Almacena las distintas situaciones laborables que puedan tener los padres.
- Formation_level: Almacena los distintos niveles de formación que puedan tener los padres.
- Civil_states: Almacena los distintos estados civiles que puedan tener los padres.
- Regions: Almacena el nombre de las distintas autonomías.
- Centers: Almacena la información de un centro y de qué autonomía es.
- Usuarios: Almacena los distintos usuarios del sistema en el servidor, y los usuarios que se hayan conectado anteriormente en el cliente para uso sin conexión.

En las tablas muchos a muchos se han generado las tablas de:

- Center_resources_odat_diagnoses: Almacena todos los recursos disponibles en una consulta diagnóstica.
- Diagnosis_items_odat_diagnoses: Almacena todos los diagnósticos y elementos de nivel III para una consulta diagnóstica ODAT.

Adicionalmente el cliente va a tener dos tablas más para su lógica de sincronización:

- Sync: Almacena los datos pendientes de sincronización, para que cuando se conecte en el servidor, si hay datos que no se han podido actualizar, se consulta en esta tabla para localizarlos fácilmente. Contiene el identificador y la tabla a la que pertenece.
- Tables: Contiene los nombres de las tablas en identificadores, para quitar peso a la tabla Sync.

Debido a la gran cantidad de campos que contienen las tablas, se puede encontrar un análisis detallado en el “Anexo I: Análisis de la base de datos” que tiene en cuenta los diferentes tipos de datos entre MySQL y SQLite[21].

[21] (SQLite)

# 5. Resultados

Las pruebas del funcionamiento del programa, por la parte cliente, se han ejecutado en dos equipos distintos:

- En un equipo de sobremesa con Windows7 con 4GB de RAM DDR3, procesador Intel Quad y una placa compatible con DDR.
- En un portátil con Debian con 512 MB de RAM DDR2, procesador Celeron.

En ambos casos, los resultados han sido satisfactorios, aunque iba un poco más lento con el portátil al tener muy pocos recursos.

Se ha entregado el prototipo del proyecto con resultados positivos. Por lo que se puede decir que se ha llegado a los primeros objetivos marcados anteriormente.

Aún tiene mucho desarrollo por delante, es por eso que se va a subir el presente proyecto a un repositorio público, para que todo el mundo pueda ver las futuras implementaciones.

Para realizar el test con el servidor, se ha usado la base de datos SQLite al ser muy ligera y rápida. Por ahora debido a los posibles que problemas que pueda generar, se ha desactivado la opción del servidor. Obviamente cuando se termine la segunda parte del proyecto, hay que pasar la base de datos a MySQL, y llamar al inicio de sesión para que pueda conectarse con el servidor.

## 5.1. Discusión

¿Se han obtenido los resultados esperados? Sí, aunque pueden ser mejores. Respecto al diseño de interfaz, es una mejora increíble. Se ha conseguido traducir correctamente dos versiones del programa en plataforma web y un lenguaje desconocido a una plataforma de escritorio multiplataforma. Por último, tiene lo que solicitaba el cliente al principio: uso del programa sin necesidad de Internet.

También realizando pruebas de sincronización aunque limitadas a un par de tablas: login, usuarios y recursos de centro, demuestra que funciona la solución presentada. Ahora hay que realizar la segunda parte de la implementación con las mejoras disponibles y su posterior corrección de errores.

Una futura mejora, sería en el rendimiento para que el cliente sea aún más ligero. Esto se optimizando el código. Otra mejora al respecto, aunque no influye demasiado, es que algunas tablas tienen hasta 34 columnas, esto es un problema bastante grave, ya que el sistema gestor de bases de datos pierde mucho tiempo para gestionar un volumen de datos tan grande.

## 5.2. Conclusiones

Se trata de un programa de gestión para escritorio, que permite su ejecución independientemente del servidor (aunque es recomendable para tener copias de seguridad, que se tenga acceso al servidor). Tiene un diseño claro y sencillo, con formularios nítidos, que facilitan la introducción de los datos. Es relativamente sencillo realizar modificaciones sobre el código.

Pero aun así hay muchas mejoras que va a necesitar, sobretodo facilitar la internacionalización, para permitir traducciones posteriores y que tenga más cobertura a través de diferentes países.

Los principales errores se han dado en la fase inicial del proyecto, analizando el código anterior. RubyOnRails es una tecnología bastante diferente de lo visto hasta ahora, y se ha perdido mucho tiempo intentando ejecutar el código y entenderlo. Cuando en realidad, el análisis es bastante más sencillo si se tiene iniciativa y se empieza a trabajar sobre lo conocido e ir analizando poco a poco las partes siguientes a implementar, ya que hay bastante documentación al respecto.

Esta dificultad se ha encontrado porque el lenguaje de Ruby contiene múltiples versiones, al igual que sus herramientas y el entorno de desarrollo. Por lo que no ha sido posible ejecutar un código con las versiones estipuladas en el proyecto al no encontrar las versiones correspondientes de estas utilidades, llamadas gems. Además la versión del framework está obsoleta por lo que si no se tienen las librerías compatibles, hay que pasar todo el código a una versión del framework más nueva con todo lo que ello conlleva: cambio en la estructura del proyecto, nuevas librerías y posibles incompatibilidades.

Otro problema encontrado, ha sido la seguridad. Ya que es bastante difícil encontrar la información sobre los requisitos que impone la Agencia de Protección de Datos a nivel informático. Por esta razón se ha perdido bastante tiempo eligiendo la mejor forma de sincronizar el servidor con sus respectivos clientes.

Por último, se trata de un proyecto masivo, y el hecho de usar tecnologías complejas para cumplir con la LOPD ralentiza mucho el trabajo de implementación.

## 5.3. Trabajos futuros

Ahora queda por delante desarrollar el resto del proyecto. Utilizando las mejoras adicionales expresadas en los objetivos y las soluciones presentadas. Por falta de tiempo, el tamaño del proyecto y tecnologías desconocidas, se han quedado algunos puntos por resolver.

Por ahora, las futuras implementaciones estarán disponibles en el repositorio de Github mencionado en la bibliografía.

# Bibliografía

Agudo, A. (14 de 03 de 2017). *El Pais*. Obtenido de Los seis retos de los niños en Latinoamérica: https://elpais.com/elpais/2017/03/13/planeta_futuro/1489386449_778558.html

Apache Commons. (09 de 11 de 2014). *commons*. Obtenido de apache: https://commons.apache.org/

baeldung. (s.f.). *baeldung*. Obtenido de Hashing SHA256 : http://www.baeldung.com/sha-256-hashing-java

Chinchilla, J. (27 de 07 de 2014). *Wordpress*. Obtenido de Encriptar y Desencriptar AES CBC 128 bits: https://bit502.wordpress.com/2014/06/27/codigo-java-encriptar-y-desencriptar-texto-usando-el-algoritmo-aes-con-cifrado-por-bloques-cbc-de-128-bits/

Crespo, A. (05 de 11 de 2016). *RedesZone*. Recuperado el 12 de 05 de 2017, de graves fallos seguridad mysql permiten acceder al servidor permisos root: https://www.redeszone.net/2016/11/05/graves-fallos-seguridad-mysql-permiten-acceder-al-servidor-permisos-root/

Federación Estatal de Asociaciones de Profesionales de Atención Temprana. (2005). *Riberdis*. Obtenido de Organización Diagnóstica para la Atención Temprana (ODAT): http://riberdis.cedd.net/handle/11181/2978

Gonzalez, A. (12 de 07 de 2016). *Ayuda Ley de Protección de Datos*. Obtenido de Nivel alto: https://ayudaleyprotecciondatos.es/2016/07/12/medidas-seguridad-nivel-alto/

iText. (s.f.). Obtenido de http://developers.itextpdf.com/

Java. (s.f.). *Java*. Obtenido de https://www.java.com/es/

Java2s - Fuente original obsoleta. (s.f.). *Java2s*. Obtenido de Multi Span Cell Table Example: http://www.java2s.com/Code/Java/Swing-Components/MultiSpanCellTableExample.htm

JFreeChart. (31 de 07 de 2014). *jfree*. Obtenido de jfreechart: http://www.jfree.org/jfreechart/

L.Saito, T. (23 de 06 de 2017). *bitbucker*. Obtenido de https://bitbucket.org/xerial/sqlite-jdbc/downloads/

MySQL. (s.f.). *MySQL*. Obtenido de Development: https://dev.mysql.com/downloads/

Netbeans. (s.f.). *Netbeans*. Obtenido de https://netbeans.org/

RubyOnRails. (s.f.). *RubyOnRails*. Obtenido de http://rubyonrails.org/

Soós, I. (26 de 09 de 2009). *Drillio*. Obtenido de String limit 64KB: https://www.drillio.com/en/2009/java-encoded-string-too-long-64kb-limit/

SQLite. (s.f.). *SQLite*. Obtenido de Datatypes: https://sqlite.org/datatype3.html

SQLiteBrowser. (s.f.). *SQLiteBrowser*. Obtenido de http://sqlitebrowser.org/

Toedter. (s.f.). *Toedter*. Obtenido de https://toedter.com/jcalendar/

Tortosa Alcaraz, L. (28 de 07 de 2017). *Github*. Obtenido de Kitsune_yuu: https://github.com/Kitsuneyuu

Zukowski, J. (s.f.). *java2s.com*. Obtenido de CheckboxTree: http://www.java2s.com/Code/Java/Swing-JFC/CheckBoxNodeTreeSample.htm

# Anexo I: Análisis base de datos

Debido a la gran cantidad de datos que se encuentran en esta aplicación, se añade esta guía donde explica todas las tablas con sus respectivos campos y tipos de campo para SQLite y MySQL:

**Center_resources**

Lista todos los posibles recursos que existen en un centro para asignarlos a los diagnósticos, con la finalidad de poder usar esos recursos para la mejora del paciente.

| Campo | Descripción | SQLite | MySQL |
|---|---|---|---|
| Id* | Identificador del recurso | INTEGER | INTEGER |
| Name | Nombre del recurso | TEXT | VARCHAR(50) |
| Description | Qué significa el recurso | TEXT | VARCHAR(250) |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 1: Análisis de la tabla center_resources

**Center_resources_odat_diagnoses**

Tabla auxiliar creada a partir de una relación de muchos a muchos. Un diagnóstico puede requerir muchos recursos de centro, y al mismo tiempo un recurso se puede encontrar en múltiples diagnósticos.

| Campo | Descripción | SQLite | MySQL |
|---|---|---|---|
| Odat_diagnosis_id* | Identificador del diagnóstico | INTEGER | INTEGER |
| Center_resources_id* | Identificador del recurso de centro | INTEGER | INTEGER |

Tabla 2: Análisis de la tabla center_resources_odat_diagnoses

**Centers**

Lista todos los centros disponibles, que se hayan dado de alta en un servidor.

| Campo | Descripción | SQLite | MySQL |
|---|---|---|---|
| Id* | Identificador del recurso | INTEGER | Int |
| Name | Nombre del recurso | TEXT | VARCHAR(50) |
| Region_id | En qué autonomía se encuentra | INTEGER | Int |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 3: Análisis de la tabla centers

**Civil_states**

Lista los posibles estados civiles que pueden tener los padres del paciente.

| Campo | Descripción | SQLite | MySQL |
|---|---|---|---|
| Id* | Identificador del estado | INTEGER | INTEGER |
| Name | Nombre del estado | TEXT | VARCHAR(50) |
| Description | Qué significa el estado | TEXT | VARCHAR(250) |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 4: Análisis de la tabla civil_states

| **Consultation_causes**<br>Lista todas las posibles causas de consulta que hayan expresado los padres para introducir en el diagnóstico. | | | |
|---|---|---|---|
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador de la causa | INTEGER | INTEGER |
| Name | Nombre de la causa | TEXT | VARCHAR(50) |
| Description | Qué significa la causa | TEXT | VARCHAR(250) |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 5: Análisis de la tabla consultation_causes

| **Formation_levels**<br>Lista los posibles y máximos niveles formativos que pueden tener los padres del paciente | | | |
|---|---|---|---|
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador del nivel de formación | INTEGER | INTEGER |
| Name | Nombre del nivel de formación | TEXT | VARCHAR(50) |
| Description | Qué significa el nivel de formación | TEXT | VARCHAR(250) |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 6: Análisis de la tabla formation_levels

| **Job_status**<br>Lista las posibles situaciones laborables que pueden tener los padres del paciente. | | | |
|---|---|---|---|
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador de la situación | INTEGER | INTEGER |
| Name | Nombre de la situación | TEXT | VARCHAR(50) |
| Description | Qué significa de la situación | TEXT | VARCHAR(250) |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 7: Análisis de la tabla job_status

| **Jobs**<br>Lista los posibles trabajos que pueden tener los padres del paciente. | | | |
|---|---|---|---|
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador del trabajo | INTEGER | INTEGER |
| Name | Nombre del trabajo | TEXT | VARCHAR(50) |
| Description | Qué significa el trabajo | TEXT | VARCHAR(250) |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 8: Análisis de la tabla jobs

| **Origin_causes**<br>Lista los motivos médicos de la derivación, es decir el por qué se ha derivado al menor a Atención temprana. Requisito del diagnóstico. | | | |
|---|---|---|---|
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador del motivo | INTEGER | INTEGER |
| Name | Nombre del motivo | TEXT | VARCHAR(50) |
| Description | Qué significa el motivo | TEXT | VARCHAR(250) |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 9: Análisis de la tabla origin_causes

| **Origin_sources** | | | |
|---|---|---|---|
| Lista las posibles fuentes de derivación, es decir, quien ha derivado al menor a Atención Temprana. Requisito del diagnóstico. | | | |
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador de la fuente | INTEGER | INTEGER |
| Name | Nombre de la fuente | TEXT | VARCHAR(50) |
| Description | Qué significa la fuente | TEXT | VARCHAR(250) |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 10: Análisis de la tabla origin_sources

| **School_types** | | | |
|---|---|---|---|
| Lista los posibles tipos de colegio del menor, como dato adicional para el diagnóstico. | | | |
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador del colegio | INTEGER | INTEGER |
| Name | Nombre del colegio | TEXT | VARCHAR(50) |
| Description | Qué significa el colegio | TEXT | VARCHAR(250) |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 11: Análisis de la tabla school_types

| **Users** | | | |
|---|---|---|---|
| Lista todos los usuarios que hay actualmente en el sistema, tanto conectados a un servidor como en local | | | |
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador del usuario | INTEGER | INTEGER |
| login | Nombre de usuario | TEXT | VARCHAR(50) |
| Crypted_passw | Contraseña encriptada con SHA256 | TEXT | VARCHAR(250) |
| Rol | Qué tipo de usuario es | TEXT | VARCHAR(20) |
| nombre | Nombre completo del usuario | TEXT | VARCHAR(50) |
| Center_id | A qué centro pertenece el usuario | INTEGER | INTEGER |
| Last_login | Última vez que inició sesión el usuario | NUMERIC | TIMESTAMP |
| active | Si el usuario puede conectarse o no | NUMERIC | TIMESTAMP |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 12: Análisis de la tabla users

| **Odat_diagnoses** | | | |
|---|---|---|---|
| Lista todos los diagnósticos de un paciente. | | | |
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador del estado | INTEGER | INTEGER |
| Medical_record | Nombre del estado | TEXT | VARCHAR(50) |
| Origin_source | Identifica la persona que ha derivado | TEXT | VARCHAR(250) |
| Origin_couse | Identifica la causa médica de origen | INTEGER | INTEGER |
| Consultation_c | Identifica causa de consulta | INTEGER | INTEGER |
| Main_diagnosis | Identifica diagnóstico principal | INTEGER | INTEGER |

| | | | |
|---|---|---|---|
| description | Datos adicionales que se requieran | TEXT | VARCHAR(250) |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 13 Análisis de la tabla odat_diagnoses

| **Medical record**<br>Lista todos los expedientes médicos de los pacientes, con los datos personales y de su entorno | | | |
|---|---|---|---|
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador del expediente | INTEGER | INTEGER |
| Center_id | Identificador del centro | INTEGER | INTEGER |
| Name | (Encriptado) Nombre del paciente | TEXT | VARCHAR(25) |
| Surname | (Encriptado) Apellidos del paciente | TEXT | VARCHAR(50) |
| Birth_date | Fecha de nacimiento del paciente | NUMERIC | DATE |
| Archive_date | Fecha de creación del expediente | NUMERIC | DATE |
| City | Ciudad donde es residente el paciente | TEXT | VARCHAR(40) |
| Address | Dirección donde vive el paciente | TEXT | VARCHAR(60) |
| Birth_Position | Posición durante el parto si fue multiple | INTEGER | INTEGER(2) |
| gender | Sexo del paciente | TEXT | VARCHAR(9) |
| Father_name | (Encriptado) Nombre del padre | TEXT | VARCHAR(25) |
| Father_surname | (Encriptado) Apellido del padre | TEXT | VARCHAR(50) |
| F_birth_date | Fecha nacimiento del padre | NUMERIC | DATE |
| Father_job_id | Identificador del trabajo del padre | INTEGER | INTEGER |
| F_civil_state | Identificador del estado civil del padre | INTEGER | INTEGER |
| F_job_status | Identificador situación laboral del padre | INTEGER | INTEGER |
| F_formation_lvl | Nivel máximo de formación del padre | INTEGER | INTEGER |
| F_email | (Encriptado) Email del padre | TEXT | VARCHAR(30) |
| F_extra_info | Información extra del padre | TEXT | VARCHAR(250) |
| Mother_name | (Encriptado) Nombre de la madre | TEXT | VARCHAR(25) |
| Mother_surname | (Encriptado) Apellido de la madre | TEXT | VARCHAR(50) |
| M_birth_date | Fecha nacimiento de la madre | NUMERIC | DATE |
| Mother_job_id | Identificador del trabajo de la madre | INTEGER | INTEGER |
| M_civil_state | Identificador del estado civil de la madre | INTEGER | INTEGER |
| M_job_status | Identificador situación laboral de madre | INTEGER | INTEGER |

| | | | |
|---|---|---|---|
| M_formation_lvl | Nivel máximo de formación de la madre | INTEGER | INTEGER |
| M_email | (Encriptado) Email de la madre | TEXT | VARCHAR(30) |
| M_extra_info | Información extra de la madre | TEXT | VARCHAR(250) |
| Home_phone | (Encriptado) Teléfono fijo | VARCHAR(9) | VARCHAR(9) |
| Portable_phone | (Encriptado) Teléfono móvil | VARCHAR(9) | VARCHAR(9) |
| Work_phone | (Encriptado) Teléfono de trabajo | VARCHAR(9) | VARCHAR(9) |
| Total_siblings | Número total de hermanos | INTEGER | INTEGER(8) |
| Postal | Código postal | INTEGER | INTEGER(5) |
| Sanitary_services | Servicios sanitarios | NUMERIC | BOOLEAN |
| Social_services | Servicios sociales | NUMERIC | BOOLEAN |
| Educative_service | Servicios educativos | NUMERIC | BOOLEAN |
| Created_at | Fecha de creación | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de última modificación | NUMERIC | TIMESTAMP |
| Multiple_birth | Especifica si es parto múltiple | NUMERIC | BOOLEAN |
| Province_id | Identificador de provincia | INTEGER | INTEGER |
| Position_siblings | Posicion entre hermanos | INTEGER | INTEGER(8) |
| Hándicap | Si el paciente tiene dependencia | NUMERIC | BOOLEAN |
| Dependency_deg | Grado dependencia mayor o igual a 33 | INTEGER | INTEGER() |
| School_type | Identificador de tipo de colegio | INTEGER | INTEGER |

Tabla 14 Análisis de la tabla medical_records

| **Diagnosis_items_odat_diagnoses** | | | |
|---|---|---|---|
| Lista todos los elementos diagnósticos de un diagnóstico, se usa para nivel III y diagnóstico detallado. Creado por tabla muchos a muchos. | | | |
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Diagnosis_item | Identificador de elementos diagnósticos | INTEGER | INTEGER |
| Odat_diagnose | Identificador del diagnóstico del paciente | TEXT | VARCHAR(50) |

Tabla 15 Análisis de la tabla diagnosis_items_odat_diagnoses

| **Diagnosis_items** | | | |
|---|---|---|---|
| Lista todos los posibles elementos de diagnóstico y nivel III. | | | |
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador del elemento | INTEGER | INTEGER |
| Name | Nombre del elemento | TEXT | VARCHAR(50) |
| Description | Descripción del elemento | TEXT | VARCHAR(250) |
| Parent_id | Nodo padre del elemento | INTEGER | INTEGER |
| Childs | Número de hijos del elemento | INTEGER | INTEGER |
| Position | Número de posición en lista ordenada | INTEGER | INTEGER |
| lvl | Número de profundidad | INTEGER | INTEGER |

| | | | |
|---|---|---|---|
| Classification_lvl | Nivel del diagnóstico o nivel III | INTEGER | INTEGER |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |

Tabla 16 Análisis de la tabla diagnosis_items

| **Individual_reports**<br>Lista todos los informes individuales creados para un paciente. Sirve para exportar a PDF. | | | |
|---|---|---|---|
| **Campo** | **Descripción** | **SQLite** | **MySQL** |
| Id* | Identificador del elemento | INTEGER | INTEGER |
| Medical_record | Nombre del elemento | TEXT | VARCHAR(50) |
| Odat_diagnosis | Descripción del elemento | TEXT | VARCHAR(250) |
| Topic | Nodo padre del elemento | INTEGER | INTEGER |
| Tests | Número de hijos del elemento | INTEGER | INTEGER |
| Results | Número de posición en lista ordenada | INTEGER | INTEGER |
| Created_at | Cuando se ha creado | NUMERIC | TIMESTAMP |
| Updated_at | Fecha de la última actualización | NUMERIC | TIMESTAMP |
| Signature | Escribe la firma del profesional | TEXT | VARCHAR(20) |
| Signed_on | Guarda la fecha de creación | NUMERIC | BOOLEAN |
| Show_signature | Muestra la firma | NUMERIC | BOOLEAN |
| Show_signed_on | Muestra la fecha de creación diagnóstico | NUMERIC | BOOLEAN |
| Sh_archive_data | Muestra la fecha del expediente médico | NUMERIC | BOOLEAN |
| Sh_birth_date | Muestra la fecha de nacimiento | NUMERIC | BOOLEAN |
| Sh_age | Muestra la edad | NUMERIC | BOOLEAN |
| Sh_full_name | Muestra el nombre completo | NUMERIC | BOOLEAN |
| Sh_gender | Muestra el sexo del paciente | NUMERIC | BOOLEAN |
| Sh_birth_pos | Muestra la posición de nacimiento | NUMERIC | BOOLEAN |
| Sh_address | Muestra la direción | NUMERIC | BOOLEAN |
| Sh_siblings | Muestra los datos de los hermanos | NUMERIC | BOOLEAN |
| Sh_father | Muestra los datos del padre | NUMERIC | BOOLEAN |
| Sh_mother | Muestra los datos de la madre | NUMERIC | BOOLEAN |
| Sh_phone_num | Muestra los teléfonos | NUMERIC | BOOLEAN |
| Sh_diagnosis_da | Muestra | NUMERIC | BOOLEAN |
| Sh_consultation | Muestra los datos de consulta | NUMERIC | BOOLEAN |
| Sh_center_res | Recursos seleccionados del centro | NUMERIC | BOOLEAN |
| Sh_detailed_diag | Los datos de diagnóstico detallado | NUMERIC | BOOLEAN |
| Sh_coordination | Los datos de coordinación | NUMERIC | BOOLEAN |
| Sh_main_diagno | Muestra el diagnostico principal | NUMERIC | BOOLEAN |
| Extra_info | Información adicional | TEXT | VARCHAR(250) |
| Sh_handicap | Muestra datos de la dependencia | NUMERIC | BOOLEAN |
| Sh_level3 | Muestra datos del nivel 3 | NUMERIC | BOOLEAN |

Tabla 17 Análisis de la tabla individual_reports

# Anexo II: Diseño de interfaz gráfico

En este anexo se va a mostrar un ejemplo del diseño de interfaz gráfico. En primer lugar se muestra la página principal como un menú con todas las opciones de la interfaz:


Ilustración 1: Diseño gráfico del menú

A continuación un ejemplo de cómo funcionan las plantillas de tablas y subtablas, con sus botones correspondientes para mostrar, modificar y eliminar campos:

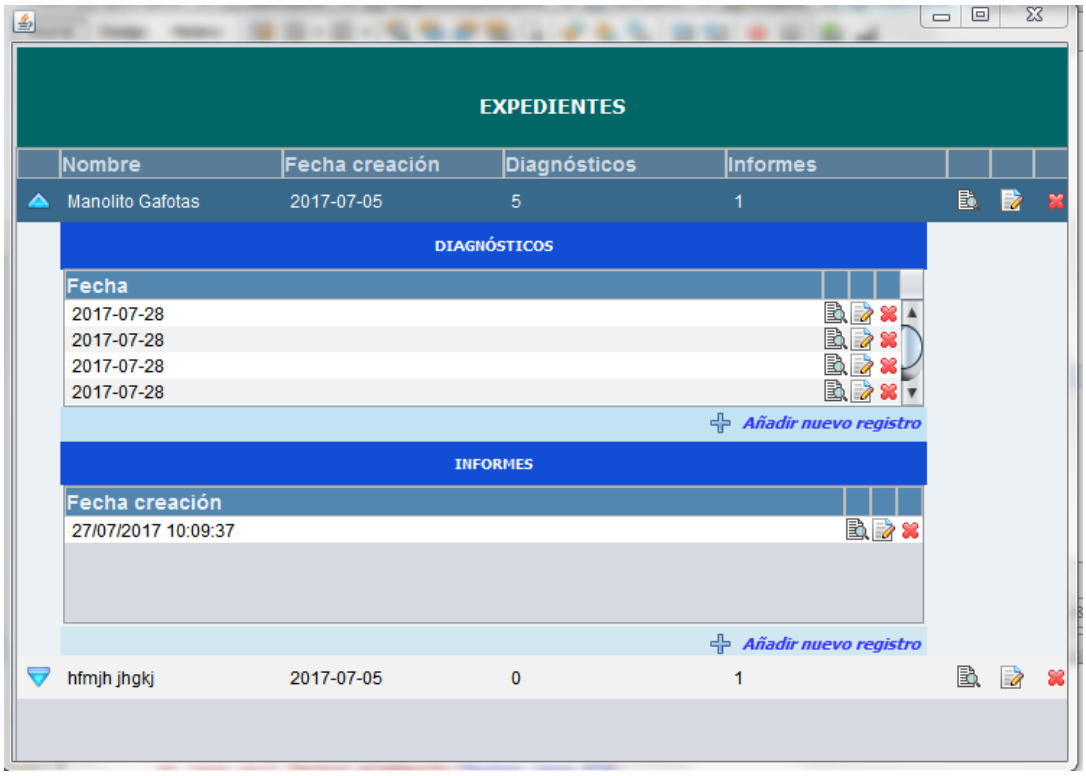


Ilustración 2: Diseño gráfico tablas y subtablas

Ahora un formulario para crear los expedientes médicos, donde se puede ver el uso de la librería JCalendar:

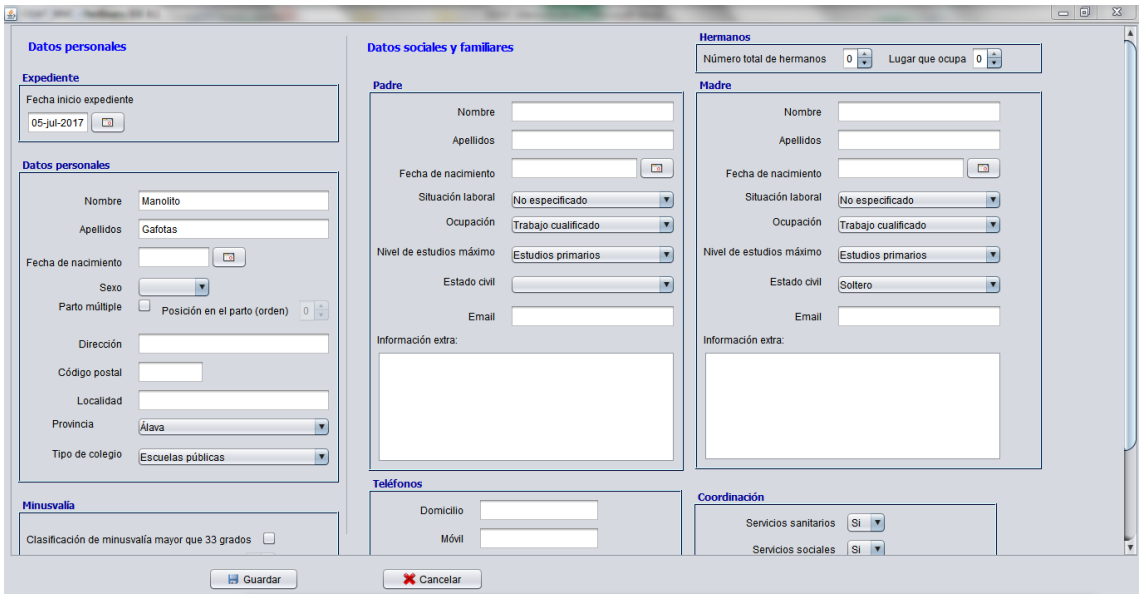


Ilustración 3: Diseño gráfico de formulario con JCalendar

Por último, una muestra de cómo se muestran los diagnósticos, con el árbol gráfico capaz de seleccionar múltiples elementos usando para ello checkbox. En el caso del diagnóstico principal, solo va  a permitir seleccionar uno de ellos:

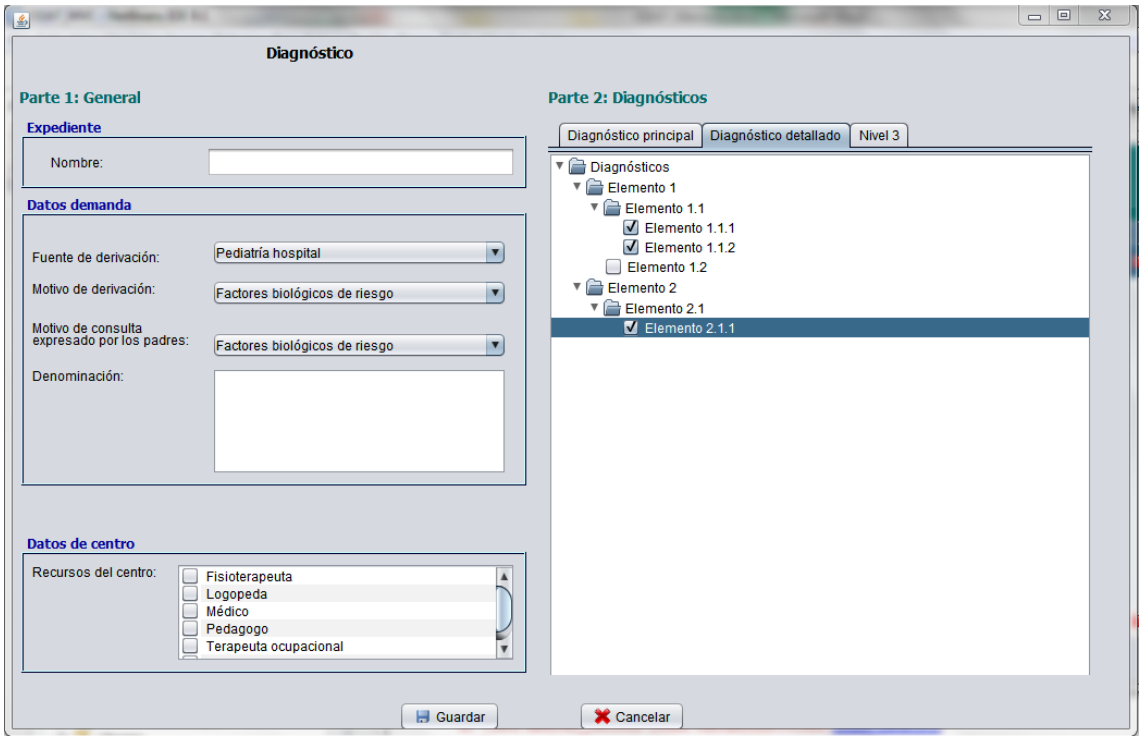


Ilustración 4: Diseño gráfico JTree con checkbox