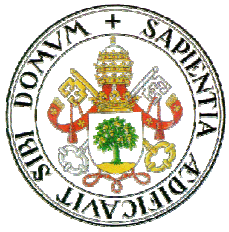

Universidad de Valladolid

E. T. S. DE INGENIERÍA INFORMÁTICA

Ingeniería Técnica en Informática de Gestión

Aplicación Web para la monitorización remota de pacientes

Alumno: Mario Santos Serrano

Tutor: Miguel Ángel Laguna Serrano

Para ti Rafa, por todo lo que me enseñaste.
Mario Santos Serrano

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

Capítulo 1

INTRODUCCIÓN

1.1. Introducción al proyecto

El proyecto “Aplicación Web para la monitorización remota de pacientes” se incluye dentro del trabajo que el Grupo de Investigación en Reutilización y Orientación a Objeto (GIRO) de la Universidad de Valladolid realiza sobre líneas de productos software (LPS). En el caso concreto de la LPS de Monitorización Remota, la intención era proporcionar un sistema de teleasistencia a los pacientes de una residencia de ancianos por diversos motivos: desorientación en paseos externos a la residencia, caídas… Cada posibilidad representa una característica opcional de la LPS que se incluirá o no en cada aplicación final.

La idea original consistía en asignar dos dispositivos a cada anciano, un teléfono móvil, el mismo que usa para hablar con su familia, y el otro es un pequeño dispositivo que se fija a la muñeca como un reloj.

El teléfono móvil tendría varias utilidades. Por un lado serviría como dispositivo de localización. Por otra parte, mediante el envío de SMS, serviría como medio de comunicación con la residencia y por último se le podría dar el uso que habitualmente tiene un teléfono móvil, que es el de realizar llamadas, solo que éstas se realizarían de forma automática en caso de emergencia.

La finalidad del dispositivo colocado en la muñeca, sería controlar parámetros fisiológicos, detectar posibles caídas, generar alarmas y dar la posibilidad de confirmarlas o cancelarlas.

Las múltiples dificultades técnicas y económicas encontradas han modificado la idea inicial para acomodarla a nuestras posibilidades (utilización de una PDA en lugar de un teléfono móvil cualquiera, utilización del mando de la Wii como detector de caídas…).

Finalmente, para que todas estas situaciones puedan ser controladas por el personal de la residencia, se pensó en desarrollar un sistema complementario de explotación que mostrara en tiempo real todos los datos recogidos y que permitiera interaccionar con ellos. Es aquí donde encaja mi proyecto.

En este primer capítulo introductorio, daré una descripción general del proyecto, expondré los objetivos del mismo y explicaré como se organizan los contenidos de la documentación.

1.2. Descripción del proyecto

El proyecto se basa en el desarrollo de una aplicación Web, que sirva como sistema de explotación para una serie de proyectos de desarrollo paralelo, enfocados a la monitorización remota de distintas características de los pacientes de la residencia sobre la que vamos a trabajar formando una Línea de Productos Software.

Uno de los pilares del grupo GIRO es la reutilización de software y una de las ideas iniciales era aplicar dicha técnica a este proyecto mediante una línea de producto. Por diversos motivos, que han ido retrasando el proyecto y debido a la complejidad del mismo esto no ha sido posible, pero se deja abierta la posibilidad de adaptarlo en un futuro.

La idea básica de lo que es una Línea de Producto Software está inspirada en los procesos de producción industrializada de los productos físicos, tales como la producción de vehículos o hardware. Consiste en el ensamblaje de partes de software previamente elaboradas.

"Una Línea de Producto Software es un conjunto de sistemas software que comparten un conjunto común y gestionado de aspectos que satisfacen las necesidades específicas de un segmento de mercado o misión y que son desarrollados a partir de un conjunto común de activos fundamentales de software de una manera preescrita" (Clements y Northrop; Software Products Lines: Practices an Patterns, 2001)

El beneficio principal de utilizar la tecnología de Líneas de Producto Software es que la entrega de productos se hace de una manera más rápida y económica porque se reducen los costes de ingeniería y con una calidad mucho mayor ya que se reducen las tasas de errores.

El desarrollo de las Líneas de Producto Software introduce un cambio en la forma de crear software, ya que este trabajo se convierte en un proceso cada vez más industrializado. Lo que se pretende es reutilizar las partes que tienen en común todos los productos y desarrollar de forma eficiente y sistemática nuevos miembros de la familia simplemente con añadirle nuevas funcionalidades.

En el diseño se muestra la variabilidad utilizando el concepto de combinación de paquetes ("package merge"), presente en el metamodelo de infraestructura de UML 2 y utilizado de forma exhaustiva en la definición misma de UML 2. El mecanismo de "package merge" consiste fundamentalmente en añadir detalles de forma incremental. Se define como una relación entre dos paquetes que indica que los contenidos de ambos se combinan. Es similar a la generalización y se utiliza cuando elementos en distintos paquetes tienen el mismo nombre y representan el mismo concepto.

Dicho concepto se extiende incrementalmente en cada paquete añadido. Seleccionando los paquetes deseados es posible obtener una definición a la medida de entre todas las posibles.

Evidentemente las reglas que establece la especificación UML2 son muy estrictas para evitar inconsistencias. Por ejemplo, no puede haber ciclos, las multiplicidades resultantes son las menos restrictivas de entre las posibles, o las operaciones deben conformar en número, orden y tipo de parámetros.

Para llevar esta idea a la práctica nos servimos de las clases parciales de C#. Si el Framework que implementa la arquitectura de la línea de productos está organizado en paquetes de clases parciales (un paquete base y tantos paquetes auxiliares como variaciones existen), para derivar una aplicación concreta basta con importar o referenciar los paquetes que se correspondan directamente con la configuración elegida en el modelo de características.

Eso sí, a la hora de programar se deberá tener un especial cuidado para impedir que se produzcan errores en tiempo de ejecución que podrían generarse al existir llamadas en un paquete hacia otro paquete que no se encuentra compilado.

Para mi aplicación podría generarse un paquete para la búsqueda de pacientes y las correspondientes consultas a sus datos personales, médicos e historial y por otra parte un paquete para la localización. De esta forma podríamos generar aplicaciones a gusto del consumidor dependiendo de sus necesidades, pudiendo extrapolar el proyecto a otros campos, como podría ser un hospital de personas discapacitadas donde solo necesitan conocer la localización de los mismos en todo momento, una guardería o incluso a una prisión.

Por lo tanto, mi aplicación Web junto con el resto de proyectos, incluirán varios tipos de monitorización que pueden ser personalizados para cada paciente dando lugar a un gran número de variantes a partir del mismo producto inicial gracias a la Línea de Productos Software.

La aplicación Web se encontrará alojado en un servidor, al igual que la base de datos y éste será el medio de comunicación con el resto de proyectos.

En el proceso de desarrollo de la aplicación Web me serviré de la tecnología de programación ASP.NET y su entorno de desarrollo integrado Visual Studio 2008. Para el desarrollo y administración de la base de datos utilizaré SQL Server y por último, introduciré diversas tecnologías que aportarán la funcionalidad que necesitamos para la aplicación.

A continuación se muestra un diagrama de despliegue para dar una idea de la que puede ser la estructura del proyecto:

- El usuario interactuará con un navegador Web alojado en un equipo de trabajo;
- La estación de trabajo conectará con el servidor, mediante un protocolo http, donde se encuentra hospedada la aplicación Web y el sistema gestor de base de datos. La estación de trabajo y el servidor Web pueden encontrarse instalados en la misma máquina;
- Por último, existirá una conexión entre el servidor Web y un servidor Google que nos proporcionará los servicios necesarios para explotar la API de Google Maps;

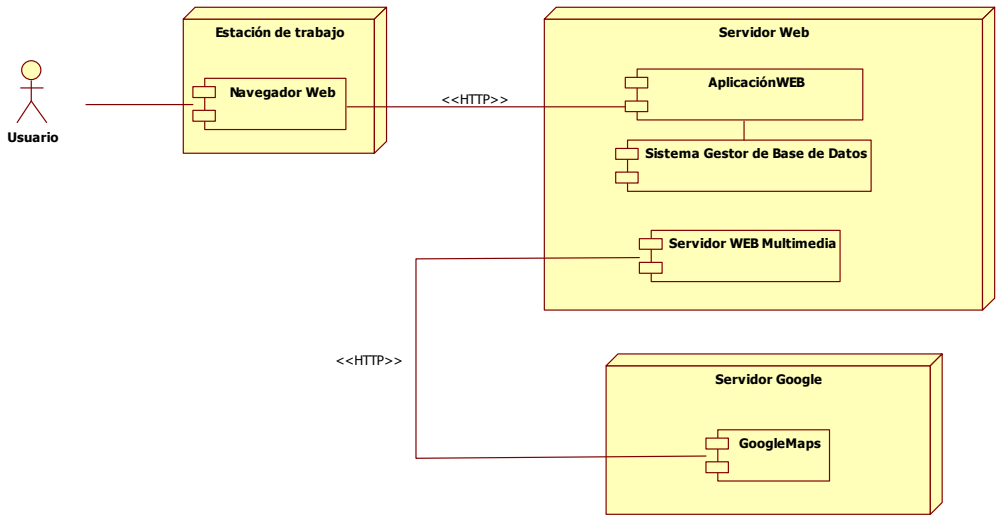

Figura 1.1.

1.3. Objetivos del proyecto

A continuación se exponen los objetivos que se intentan cubrir con este proyecto:

Realización de una aplicación Web capaz de mostrar los datos capturados por una serie de dispositivos externos pertenecientes a otros proyectos de una forma vistosa pero a la par lo más funcional y profesional posible.

Esta aplicación será desarrollada en ASP.NET, así, la interfaz tendrá el aspecto de una aplicación Web tradicional con su cabecera, menú lateral, menú de navegación… de tal forma que sea amigable e intuitiva para el usuario final.

Al estar desarrollada en ASP.NET se necesitará de un servidor con las características necesarias para que aloje este tipo de aplicación. Dicho servidor también deberá contener la base de datos con la que interactúa la aplicación.

Aprendizaje de las tecnologías empleadas. Familiarizarse con ASP.NET, adquirir experiencia en el entorno de desarrollo integrado Visual Studio, manejo de bases de datos y utilización de diversas APIS que aporten funcionalidad extra a la aplicación como son: Google Maps (para la localización de los pacientes) o Microsoft Chart Control (para la edición de gráficas).

Simular un proceso de trabajo profesional siguiendo todos los pasos de análisis, diseño e implementación, tratando de cumplir con todos los requisitos establecidos en un período de tiempo establecido.

Cumplir con las peticiones y sugerencias del personal médico de la residencia ya que serán ellos los usuarios finales de la aplicación Web.

1.4. Contenido de la memoria

La memoria se organiza en capítulos y estos a su vez en diferentes puntos, incorporando al final de la misma una serie de apéndices para la compresión de ciertos aspectos importantes del proyecto:

• Capítulo 1. Introducción: Sección en la que nos encontramos actualmente. Se trata de una breve presentación del tema y objetivos que se abordarán en el proyecto. Por último aparece el contenido tanto de la memoria como del CD.

• Capítulo 2. Desarrollo del proyecto: Este capítulo contiene toda la documentación necesaria para hacer un seguimiento del desarrollo del proyecto. Aparecen los modelos de Análisis (incluye el SRS y los casos de uso), de Diseño (diferenciando las capas Vista, Lógica y Persistencia) e Implementación (herramientas utilizadas para desarrollar el proyecto)

• Capítulo 3. Pruebas: Sección donde se expondrá un resumen general de las pruebas realizadas sobre la aplicación para encontrar fallos y mejorar la misma.

• Capítulo 4. Manuales de Usuario: Sección dirigida a los usuarios finales, en la que se da una explicación detallada de la instalación y manejo del sistema.

• Capítulo 5. Conclusiones: Esta sección pretende mostrar futuras líneas de trabajo a la vez que se hace una reflexión sobre los conocimientos adquiridos.

• Bibliografía: Recopilación de las fuentes bibliográficas, así como referencias Web, consultadas durante la elaboración del proyecto.

• Apéndice A. Tecnologías utilizadas: Introducción y explicación sobre las características de la plataforma .NET y la tecnología AJAX utilizadas en el desarrollo del proyecto.

• Apéndices B. APIS utilizadas: Sección en la que explico las APIS utilizadas en el proyecto (GoogleMaps.Subgurim.NET y ASP.NET Chart Control) para conseguir los objetivos y sugerencias propuestas.

1.5. Contenidos del CD-ROM

El contenido del CD-ROM adjunto al proyecto es el siguiente:

- **Memoria:** La versión en PDF del documento impreso completo, en un archivo único con el nombre memoria.pdf.
- **Software:** Carpeta que contiene todo el software desarrollado en versión fuente.
- **Instalación:** Carpeta con los ejecutables para la inicialización de la aplicación.
- **Manuales:** La versión digital de los manuales de uso e instalación.
- **Programas y ejemplos:** Carpeta con programas de libre difusión utilizados para el desarrollo de la documentación del proyecto, ejemplos para las APIS utilizadas y sus correspondientes archivos DLL.
- **Desplegables:** Carpeta con los desplegables que se incluyeron en la memoria impresa.

Capítulo 2

DESARROLLO DEL PROYECTO

ANÁLISIS

2.1. Especificación de Requisitos Software (SRS)

El propósito de este apartado, es desarrollar la especificación del comportamiento de un sistema de monitorización remota de pacientes. El análisis presentado en este documento pretende cubrir las necesidades descritas por el cliente.

Contiene los requisitos funcionales, de información y no funcionales. Dentro de los requisitos funcionales aparece la definición de actores y el modelo de casos de uso (diagrama y explicación de los casos de uso).

2.1.1. Requisitos funcionales

En los siguientes subapartados detallo los requisitos funcionales encontrados para el sistema. En los cuales trato de expresar con exactitud los servicios que el sistema debe proporcionar y cómo se debe comportar ante situaciones particulares.

2.1.1.1. Requisitos de usuario

FRQ-0001	**Ayuda**
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *proveer al usuario de una ayuda para cada parte de la aplicación.*

Figura 2.1.

FRQ-0002	**Mostrar estado**
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá, *en la zona de localización, identificar los posibles estados de un paciente con un color diferente (rojo o verde).*

Figura 2.2.

FRQ-0003	**Localizar paciente**
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *situar a cada paciente en las coordenadas (latitud, longitud y altura) en las que se encuentre, independientemente de si está en el interior o en el exterior de la residencia.*

Figura 2.3.

FRQ-0004	**Planos edificio**
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *mostrar el plano de cada una de las plantas del edificio de la residencia.*

Figura 2.4.

FRQ-0005	Tratar alertas
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *permitir al usuario marcar como 'tratada' una alerta, de forma que esta se elimine de la lista de alertas pendientes por tratar.*

Figura 2.5.

FRQ-0006	Acceso alertas
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *permitir acceder a las alertas desde cualquier punto de la aplicación.*

Figura 2.6.

FRQ-0007	Alertas
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *proporcionar al usuario una zona en la que pueda ver los pacientes pendientes de tratar.*

Figura 2.7.

FRQ-0008	Identificador único de paciente
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *identificar de forma única a cada paciente de la residencia.*

Figura 2.8.

FRQ-0009	Datos personales
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *mostrar los siguientes datos personales del paciente cuando se consulte tanto mediante la búsqueda como mediante la localización: nombre, apellidos, edad, sexo, habitación, localización actual, estado paciente, foto.*

Figura 2.9.

FRQ-0010	Datos médicos
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *mostrar los siguientes datos médicos en tiempo real del paciente cuando se consulte tanto mediante la búsqueda como mediante la localización: grafica pulso cardiaco, gráfica saturación oxigeno.*

Figura 2.10.

FRQ-0011	**Restringir campo a campo**
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *restringir la entrada de datos campo a campo del formulario de búsqueda para facilitar el uso al usuario.*

Figura 2.11.

FRQ-0012	**Búsqueda total**
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *mostrar todos los pacientes existentes si no se rellena ningún campo de búsqueda.*

Figura 2.12.

FRQ-0013	**Búsqueda de pacientes**
Versión	1.0 (12/06/2009)
Descripción	El sistema deberá *permitir la búsqueda de un paciente por los siguientes campos: nombre, primer apellido, segundo apellido, edad, sexo, idpaciente, habitación, estado paciente.*

Figura 2.13.

2.1.1.2. Definición de actores

En este subapartado se enumeran los actores del sistema y se da una breve explicación de los mismos.

- **Personal Médico [ACT-0001]:** Usuario principal de la aplicación que consulta los datos que la aplicación Web provee.
- **Tiempo [ACT-0002]:** Actor asociado a todos los casos de uso que capturan funcionalidades automáticas.

2.1.1.3. Modelo de casos de uso

El objetivo de este subapartado es detallar el Modelo de casos de uso de la aplicación Web para la monitorización remota de pacientes. Contiene el diagrama de casos de uso, donde se representan los casos de uso existentes, las relaciones entre los mismos y con los actores del sistema y una descripción de cada uno de los casos de uso.

<<Página incluida en la carpeta *Desplegables* del CD-ROM>>

2.1.1.3.2. Detalle de los casos de uso

Casos de uso de Personal Médico:

- **Consultar_ayuda [UC-0001]:** El personal médico quiere consultar la ayuda disponible para interactuar con la aplicación.

- **Reiniciar_busqueda [UC-0002]:** El personal médico puede restaurar los campos de búsqueda tras realizar una búsqueda satisfactoria o para borrarlos en caso de error.

- **Actualizar_alertas_manual [UC-0003]:** El personal médico pide la actualización de la lista de alertas pendientes.

- **Consultar_alerta [UC-0004]:** El personal médico desea consultar el estado del paciente que ha sufrido la alerta.

- **Tratar_alerta [UC-0005]:** El personal médico decide dar por tratada la alerta, de forma que ésta se elimine de la lista de alertas pendientes.

- **Centrar_paciente_automatico [UC-0006]:** El personal médico desea situar en el mapa, ya sea dentro de la residencia o en el exterior, al paciente del cual están consultado sus datos.

- **Buscar_datos_paciente [UC-0007]:** El personal médico realiza la búsqueda de uno o varios pacientes a partir de diferentes campos.

- **Consultar_datos_paciente [UC-0008]:** El personal médico desea consultar los datos personales, médicos o el historial de un paciente determinado.

- **Localizar_pacientes_manual [UC-0009]:** El personal médico controla la situación de todos aquellos pacientes que se encuentran localizados en las diferentes plantas de la residencia o en el exterior.

Casos de uso de Tiempo:

- **Actualizar_graficas [UC-0010]:** El actor tiempo actualiza las gráficas correspondientes a los datos médicos con los nuevos datos capturados.

- **Actualizar_alertas [UC-0011]:** El actor tiempo actualiza la lista de alertas para informar al personal médico en tiempo real de posibles complicaciones.

2.1.1.3.3. Descripción de los casos de uso

UC-0001	**Consultar_ayuda**	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el personal médico desee obtener ayuda acerca de las posibles tareas a realizar*	
Secuencia normal	**Paso**	**Acción**
	1	El actor Personal Médico (ACT-0001) *solicita consultar ayuda*
	2	El sistema *muestra la ayuda relacionada con el formulario en el que se encuentra*
Poscondición	Se muestra la ayuda	

Figura 2.15.

UC-0002	**Reiniciar_busqueda**	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando o *el personal médico desee eliminar las criterios de búsqueda utilizados con anterioridad*	
Precondición	Existen criterios de búsqueda o una tabla de resultados	
Secuencia normal	**Paso**	**Acción**
	1	El actor Personal Médico (ACT-0001) *solicita reestablecer la búsqueda*
	2	El sistema *elimina el contenido de los campos de búsqueda y la tabla de resultados*
Poscondición	La aplicación está preparada para una nueva búsqueda	

Figura 2.16.

UC-0003	**Actualizar_alertas_manual**	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando o *el personal médico desee actualizar la lista de alertas*	
Precondición	El sistema se deberá encontrar en el formulario de Localización	
Secuencia normal	**Paso**	**Acción**
	1	El actor Personal Médico (ACT-0001) *solicita actualizar la lista de alertas*
	2	El sistema *muestra una lista actualizada con las alertas pendientes de tratar*
Poscondición	La lista de alertas se actualiza con las alertas existentes en ese momento	

Figura 2.17.

UC-0004	**Consultar_alerta**	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando o *el personal médico desee saber el motivo de la alarma* o durante la realización de los siguientes casos de uso: [UC-0007] Tratar_alerta	
Precondición	Debe existir una alerta	
Secuencia normal	**Paso**	**Acción**
	1	El actor Personal Médico (ACT-0001) *solicita consultar una alerta existente*
	2	El sistema *obtiene los datos correspondientes al paciente responsable de la alerta y muestra la información correspondiente a la alerta consultada*
Poscondición	El sistema muestra los datos del paciente	

Figura 2.18.

UC-0005	**Tratar_alerta**	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el personal médico desee establecer como tratada una alerta*	
Precondición	El paciente en cuestión posee una alerta y el sistema se encuentra en el formulario Datos Personales	
Secuencia normal	**Paso**	**Acción**
	1	Se realiza el caso de uso Consultar_alerta (UC-0004) o Consultar_datos_paciente (UC-0008)
	2	El actor Personal Médico (ACT-0001) *solicita tratar la alerta consultada*
	3	El sistema *actualiza los datos dando la alerta por tratada*
Poscondición	La alerta se eliminará de la lista de alertas	

Figura 2.19.

UC-0006	Centrar_paciente_automatico	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el personal médico desee localizar a un paciente del cual está viendo los datos personales*	
Precondición	El paciente en cuestión está localizado y la aplicación debe encontrarse en el formulario de Datos Personales	
Secuencia normal	**Paso**	**Acción**
	1	El actor Personal Médico (ACT-0001) *solicita localizar al paciente*
	2	El sistema *muestra un mapa en cuyo centro se encuentra el paciente a localizar*
Poscondición	El sistema muestra un mapa, centrado en las coordenadas del paciente	
Excepciones	**Paso**	**Acción**
	1	Si *el paciente no está localizado*, el sistema *no mostrará el enlace que posibilita localizar al paciente automáticamente*, a continuación este caso de uso

Figura 2.20.

UC-0007	Buscar_datos_paciente	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el personal médico desee buscar a un paciente para consultar sus datos* o durante la realización de los siguientes casos de uso: [UC0008]Consultar_datos_paciente	
Secuencia normal	**Paso**	**Acción**
	1	El actor Personal Médico (ACT-0001) *rellena uno o varios campos de texto según los criterios de búsqueda deseados*
	2	El sistema *muestra una tabla de resultados con los pacientes concordantes a los criterios de búsqueda*
Poscondición	El sistema muestra una lista con los pacientes coincidentes con los criterios de búsqueda introducidos	
Excepciones	**Paso**	**Acción**
	1	Si *no introduce correctamente los caracteres permitidos en cada campo de búsqueda*, el sistema *no le permitirá la escritura en los mismos*, a continuación este caso de uso *continúa*
	2	Si *no existe ningún paciente que coincida con los criterios de búsqueda*, el sistema *no mostrará ningún resultado*, a continuación este caso de uso queda sin efecto

Figura 2.21.

UC-0008	**Consultar_datos_paciente**	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el personal médico desee consultar los datos personales, médicos o el historial de un paciente*	
Precondición	El personal médico ha escogido un paciente para ver sus datos	
Secuencia normal	**Paso**	**Acción**
	1	Se realiza el caso de uso Buscar_datos_paciente (UC-0006) o Localizar_pacientes_manual (UC-0009)
	2	El actor Personal Médico (ACT-0001) *selecciona el paciente del cual desee consultar los datos*
	3	El sistema *obtendrá todos los datos necesarios del paciente*
	4	El actor Personal Médico (ACT-0001) *solicita consultar los datos que desee del paciente seleccionado*
	5	El sistema *muestra los datos escogidos*
Poscondición	El sistema muestra un conjunto de pestañas entre las que se encuentran los datos personales, los datos médicos y el historial	
Excepciones	**Paso**	**Acción**
	5	Si *el paciente no se encuentra monitorizado para alguna de las opciones médicas*, el sistema *informará de ello mediante un mensaje*, a continuación este caso de uso *continúa*

Figura 2.22.

UC-0009	Localizar_pacientes_manual	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el personal médico desee conocer la localización general de los pacientes ya sea dentro o fuera de la residencia*	
Precondición	Existen pacientes localizados	
Secuencia normal	**Paso**	**Acción**
	1	El actor Personal Médico (ACT-0001) *solicita localizar un paciente*
	2	El sistema *muestra un formulario con tantas pestañas como plantas tenga el edificio*
	3	El actor Personal Médico (ACT-0001) *selecciona la planta en la que desea buscar*
	4	El sistema *muestra los pacientes existentes en dicha planta*
	5	El actor Personal Médico (ACT-0001) *selecciona el paciente que desea*
Poscondición	El sistema muestra un mapa, centrado en las coordenadas del paciente	
Excepciones	**Paso**	**Acción**
	4	Si *el paciente no está localizado*, el sistema *no mostrará al paciente en el mapa*, a continuación este caso de uso *continúa*
	4	Si *los pacientes se encuentran fuera del foco del mapa*, el actor Personal Médico (ACT-0001) *puede desplazarse libremente por el mapa hasta encontrarlo o pulsar sobre las flechas existentes que le llevarán al paciente automáticamente*, a continuación este caso de uso *continúa*
	5	Si *el zoom es inadecuado*, el actor Personal Médico (ACT-0001) *puede ampliar o disminuir el zoom a su gusto para facilitar la búsqueda*, a continuación este caso de uso *continúa*<

Figura 2.23

UC-0010	Actualizar_graficas	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *sea necesario actualizar los datos que componen las gráficas.*	
Precondición	Ha discurrido el tiempo establecido para llevarse a cabo la actualización	
Secuencia normal	**Paso**	**Acción**
	1	El actor Tiempo (ACT-0002) *solicita actualizar los datos de las gráficas*
	2	El sistema *muestra las gráficas con los nuevos datos*
Poscondición	El sistema muestra la gráfica con los nuevos datos	

Figura 2.24.

UC-0011	**Actualizar_alertas**	
Versión	1.0 (20/06/2009)	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *sea necesario actualizar las alertas*	
Precondición	Ha discurrido el tiempo establecido para llevarse a cabo la actualización	
Secuencia normal	**Paso**	**Acción**
	1	El actor Tiempo (ACT-0002) *solicita actualizar la lista de alertas*
	2	El sistema *muestra una lista actualizada con las alertas pendientes de tratar*
Poscondición	Se muestra una lista actualizada de las alertas existentes	

Figura 2.25.

2.1.2. Requisitos de información

IRQ-0001	**Datos Paciente**
Descripción	El sistema deberá almacenar la información correspondiente a *un paciente*. En concreto:
Datos específicos	idPaciente: cadena de caracteres que identificará de forma única al paciente nombre: se corresponde al nombre del paciente apellidos: los apellidos del paciente sexo: indicará si un paciente es hombre o mujer edad: la edad correspondiente al paciente habitación: dormitorio del paciente estado: situación en la que se encuentra (rojo=alerta; verde=normal) foto: fotografía del paciente

Figura 2.26.

IRQ-0002	Datos centro
Descripción	El sistema deberá almacenar la información correspondiente a *la residencia sobre la que trabajamos.* En concreto:
Datos específicos	idCentro: identificador del centro dirección: localización del centro población código postal país responsable teléfono servicio Web

Figura 2.27.

2.1.3. Requisitos no funcionales

NFR-0001	Formato de coordenadas
Versión	1.0 (12/06/2009)
Dependencias	Ninguno
Descripción	El sistema deberá *utilizar un formato de coordenadas estándar además de tratar de forma homogénea las coordenadas tomadas por diferentes dispositivos.*
Importancia	Vital

Figura 2.28.

NFR-0002	Abstracción de sensores
Versión	Ninguno
Descripción	El sistema deberá *ser capaz de almacenar y evaluar los datos de cualquier sensor.*
Importancia	Vital

Figura 2.29.

NFR-0003	**Facilidad de uso**
Versión	1.0 (12/06/2009)
Dependencias	Ninguno
Descripción	El sistema deberá *facilitar la usabilidad de la aplicación, con el objeto que este pueda ser utilizado por personal con baja cualificación informática.*
Importancia	Vital

Figura 2.30.

NFR-0004	**Lenguajes de programación**
Versión	1.0 (12/06/2009)
Dependencias	Ninguno
Descripción	El sistema deberá *será programado bajo la arquitectura .NET de Microsoft y el lenguaje C#.*
Importancia	Vital

Figura 2.31.

NFR-0005	**Estándar navegadores**
Versión	Ninguno
Descripción	El sistema deberá *ser accesible a través de los principales navegadores Web: Internet Explorer, Fire Fox.*
Importancia	Vital

Figura 2.32.

2.2. Modelo de Dominio

Este apartado ofrece una breve descripción de las clases involucradas en el modelo de dominio obtenido de la Aplicación Web para la monitorización remota de pacientes.

El modelo de dominio recoge las clases que componen la aplicación halladas durante el análisis del sistema, las relaciones entre ellas y los atributos que contienen, con el fin de satisfacer todos y cada uno de los requisitos software relativos al sistema que estamos tratando.

En primer lugar se muestra el diagrama de clases inicial obtenido y a continuación se encuentra una descripción de las clases participantes.

2.2.1. Diagrama de dominio

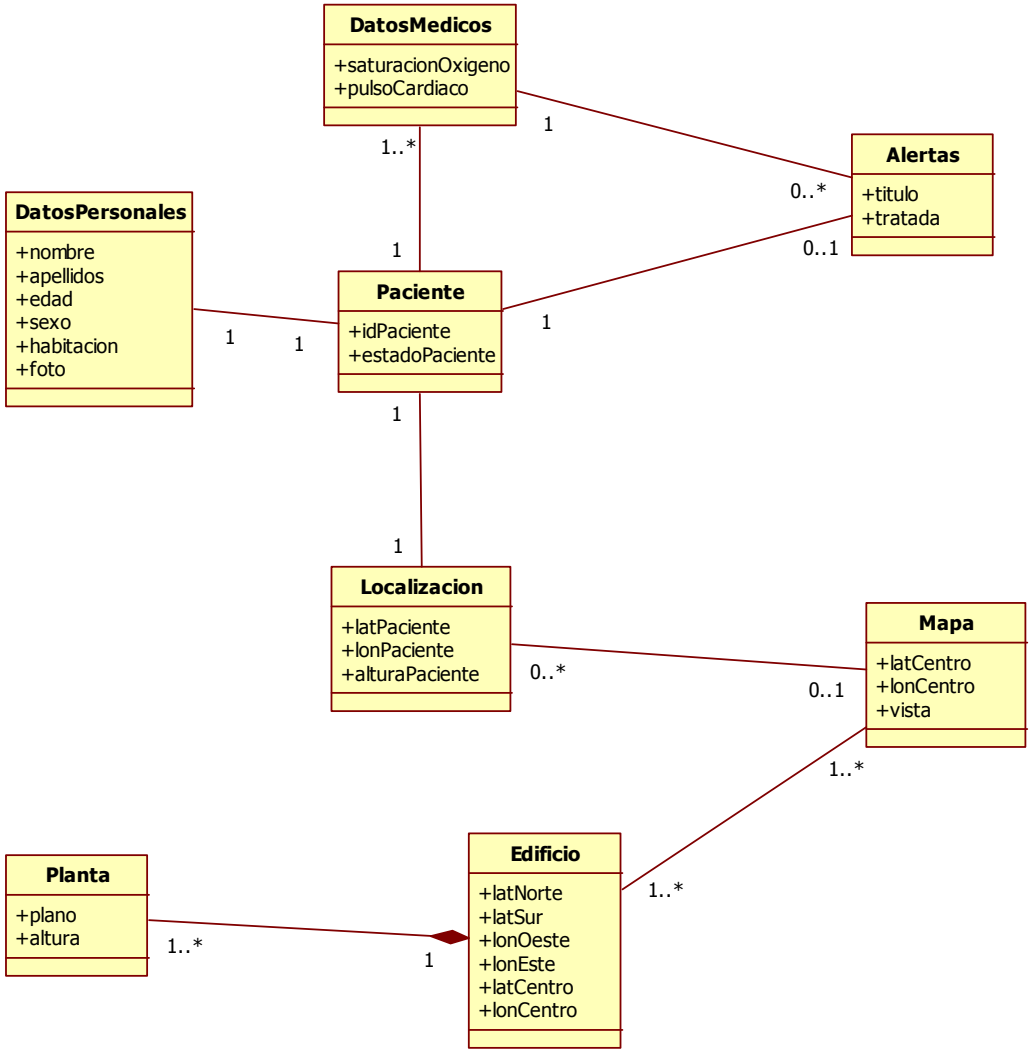

Figura 2.33.

2.2.2. Explicación clases diagrama de dominio

- **Planta**
  Esta clase representa a una planta cualquiera de un edificio. Cada planta tiene un plano de la misma asociado y se encuentra a una altura, que se utilizará para colocar al paciente en la planta correcta.

- **Edificio**
  Esta clase representa al edificio de la residencia a monitorizar. Tiene unas coordenadas para situarlo y está formado por una o más plantas.

- **Mapa**
  Esta clase representa el mapa sobre el cual visualizaremos la residencia. Posee unas coordenadas para centrar el mapa sobre la residencia y una vista para determinar el tipo de mapa a representar.

- **Localizacion**
  Esta clase se refiere a la localización que tendrá un paciente en un determinado mapa, representada mediante los atributos latPaciente, lonPaciente y un tercer atributo, alturaPaciente, que indicará la planta en la que se encuentra el paciente. Esta localización es única para cada paciente.

- **Paciente**
  Mediante esta clase se representa a un paciente cualquiera de la residencia. Cada paciente cuenta con un identificador que le distingue del resto de pacientes.

- **DatosPersonales**
  Esta clase contiene una serie de datos personales de cada paciente como: el nombre, apellidos, sexo, edad… Su existencia independiente de la clase paciente se debe a un posible interés por mantener los datos aún cuando el paciente en cuestión no se encuentre en la residencia.

- **DatosMedicos**
  Esta clase contiene los datos médicos disponibles para cada paciente. Se encuentran relacionados con las alertas ya que éstos serán los causantes de la activación de las alertas. Se encuentra separada de la clase paciente para facilitar su modificación.

- **Alertas**
  Esta clase representa las alertas que han sido captadas por diversos dispositivos y que serán mostradas en la aplicación. Cada alerta es individual para cada paciente.

DISEÑO

2.3. Modelo estructural (Estructura de tres capas)

En este apartado se encuentra el Modelo de Diseño del sistema. Este modelo servirá de guía en las siguientes etapas del desarrollo del sistema.

El Modelo de Diseño profundiza en las estructuras y comportamiento descritos en el Modelo de Análisis para acercarlos a la implementación real del sistema, por tanto, el diagrama de dominio sufrirá una evolución a clases software, con la posible modificación, creación o fusión de las clases existentes.

Optamos por la aplicación del patrón arquitectónico de tres capas para establecer el modelo de diseño del sistema.

La programación por capas es un estilo de programación en la que el objetivo primordial es la separación de la lógica de negocio de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de presentación al usuario.

La ventaja principal de este estilo, es que el desarrollo se puede llevar a cabo en varios niveles y en caso de algún cambio sólo se ataca al nivel requerido sin tener que revisar entre código mezclado.

En dichas arquitecturas a cada capa se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

La arquitectura propuesta se basa en tres capas bien diferenciadas. Una primera capa que englobará todo lo referente al interfaz de usuario, una capa intermedia que se encargará de llevar la lógica de negocio y una capa inferior responsable de gestionar la persistencia del sistema.

1.- **Capa de presentación**: es la que ve el usuario (Interfaz). Esta capa se comunica únicamente con la capa de negocio.

2.- **Capa de negocio**: es donde residen los programas que se ejecutan, recibiendo las peticiones del usuario y enviando las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.

3.- **Capa de persistencia**: es la encargada del acceso a datos. Conecta directamente con uno o más gestor de bases de datos que realiza todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

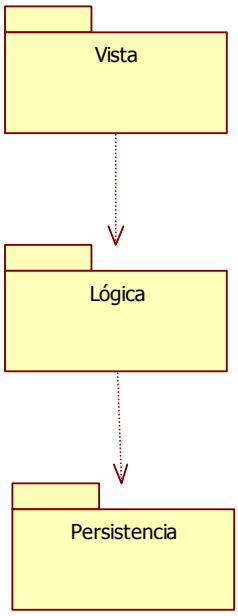

Figura 2.34.

2.3.1. Capa de vista o interfaz

Se entiende por interfaz como el artefacto que se usa para acceder e interactuar con un sistema. Se trata a la vez un límite y un espacio común entre sistema y usuario. Este artefacto nos informa qué acciones son posibles, el estado actual del sistema y los cambios producidos, y nos permite actuar con o sobre él.

Las interfaces Web tienen ciertas limitaciones en las funcionalidades que se ofrecen al usuario. Hay funcionalidades comunes en las aplicaciones de escritorio como dibujar en la pantalla o arrastrar-y-soltar que no están soportadas por las tecnologías Web estándar. Los desarrolladores Web generalmente utilizan lenguajes interpretados o de script en el lado del cliente para añadir más funcionalidades, especialmente para ofrecer una experiencia interactiva (páginas dinámicas).

Recientemente se han desarrollado tecnologías que mejoran enormemente la experiencia del usuario con la página. Es el caso de Ajax, tecnología utilizada en este proyecto, con la que se pueden construir aplicaciones que se salen del modelo tradicional Web de envíos sucesivos, proporcionando una interfaz de usuario mejorada gracias a la comunicación asíncrona entre la interfaz de usuario y el servidor Web.

Una interfaz mal diseñada se dice que genera problemas de usabilidad (*Def.-Aplicada a un sistema o herramienta, medida de su utilidad, facilidad de uso, facilidad de aprendizaje y apreciación para una tarea, un usuario y un contexto dado*). El mal diseño de las interfaces puede generar aumento de costos, algunos de ellos son medibles y otros no. Actualmente, hasta el 45% del código de una aplicación está dedicado a la interfaz, persiguiendo en todo momento la facilidad de uso. Sin embargo, se dedica algo menos del 10% del presupuesto global de un proyecto al desarrollo de la interfaz, lo que indica que aumentar los recursos destinados al desarrollo de la interfaz es una excelente inversión, teniendo en cuenta la relación costo/beneficio medible y segura, aún sin tener en cuenta los beneficios no medibles en dinero como el aumento de la satisfacción.

Dado que el diseño de interfaces no es mi objetivo, sino un medio de llegar al sistema, me serví de una plantilla de dominio público ya desarrollada (existe una pequeña reseña al diseñador Web como forma de agradecimiento en el margen derecho de la aplicación) sobre la cual realicé los cambios que consideré oportunos.

Considerando lo explicado anteriormente, opté por una interfaz muy sencilla, siguiendo el estilo estándar actual de Internet:

- Cabecera de la aplicación con un título identificativos;
- Menú de navegación superior;
- Una zona a la derecha con funciones de acceso rápido como son la ayuda y la lista de alertas;
- Zona central de gran tamaño donde poder visualizar los contenidos;

En la siguiente página podemos ver como es el diseño de la interfaz de la aplicación:

<<Página incluida en la carpeta *Desplegables* del CD-ROM>>

Formularios:

Tras tomar las decisiones de diseño de la interfaz y siguiendo el modelo de capas anteriormente planteado, a continuación se muestran los formularios correspondiente a la capa interfaz, seguido de una breve explicación de cada uno de los formularios. Cada uno de ellos hereda de la clase *System.Web.UI.Page* que nos permite tener acceso a una serie de propiedades como son: *Request, Response, Session, Trace,…*

- **Site1.Master**
  Se trata de la página maestra de la aplicación. Define una interfaz común para toda la aplicación que el resto de formularios utilizarán.

- **Default.aspx**
  Se trata de uno de los dos posibles formularios de inicio de la aplicación. Posee la interfaz definida en el archivo *Site1.Master* y una serie de cuadros de texto para realizar búsquedas de pacientes.

- **Datos.aspx**
  Este formulario posee la interfaz definida en el archivo *Site1.Master* y contiene tres pestañas: Datos Personales; Datos Médicos e Historial de incidencias.

- **Localizacion.aspx**
  Es el otro formulario que puede servir de inicio de la aplicación. Posee la interfaz definida en *Site1.Master* y contiene 3 pestañas, tantas como plantas la residencia. Utiliza la API de Google Maps para situar a los pacientes en el mapa.

- **IframeGraficas.aspx**
  Se trata de un formulario incluido en el formulario *Datos.aspx* en forma de iframe, que contiene las gráficas correspondientes a los datos personales del paciente.

2.3.2. Capa lógica o de negocio

Esta capa se encuentra situada entre las otras dos capas. Se comunica con la capa de presentación, de la cual recibe los eventos generados por el usuario y presenta en la misa los resultados generados. A su vez se comunica con la capa de datos, para solicitar al gestor de base de datos el almacenamiento o la recuperación de datos.

Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. En la siguiente página podemos ver el modelo de diseño aplicado al proyecto.

<<Página incluida en la carpeta *Desplegables* del CD-ROM>>

2.3.2.2. Explicación de las clases de diseño

- **Edificio**

  Esta clase representa a la residencia sobre la que trabajamos. Al no existir la necesidad de implementar una gestión de los edificios, de forma que el usuario pudiera editar los edificios (modificar el número de plantas, los planos para cada planta…), he optado por fusionar las clases *Planta* y *Edificio* del modelo de dominio en una única clase que genere de forma automática el edificio de la residencia. Un edificio podrá estar en uno o en más mapas, ya que en distintos mapas pueden aparecer plantas de un mismo edificio.

- **Mapa**

  Esta clase representa el mapa sobre el cual visualizaremos la residencia y sus alrededores. Respeta la clase *Mapa* del modelo de domino y se le añaden una serie de métodos para editar al propio mapa y las figuras que representarán a los pacientes localizados. En un mapa puede haber más de un edificio y también varios pacientes.

- **Paciente**

  Esta clase contiene todos los atributos necesarios para trabajar con un paciente de la residencia, además de una serie de métodos para rescatar dichos datos de la base de datos. Al no existir una gestión de pacientes y al ser utilizada únicamente como clase de consulta de datos, fusionamos las clases *Localizacion*, *DatosPersonales*, *DatosMedicos* y Paciente en esta clase. Cada paciente se encontrará en uno y sólo un mapa ya que no puede estar en dos sitios a la vez, puede tener alguna gráfica asociada y tener una alerta.

- **Graficas**

  Esta clase se genera por la necesidad de manejar las gráficas que aparecen en la aplicación. En un primer momento no las tuvimos en cuenta y fueron añadidas por petición del personal médico de la residencia para un mejor aprovechamiento de la aplicación. Cada gráfica pertenece a un único paciente.

- **Alertas**

  Esta clase se encarga de administrar las alertas generadas por dispositivos externos para poder representarlas en la aplicación. Cada alerta pertenece a un único paciente, pudiendo poseer varias alertas el mismo paciente.

2.3.2.3. Diagramas de secuencia

Un diagrama de secuencia muestra las interacciones entre objetos ordenadas en secuencia temporal. Muestra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita por el escenario.

- A continuación se muestran los diagramas de secuencia que he considerado más importantes:

Tratar_alerta [UC-0005]

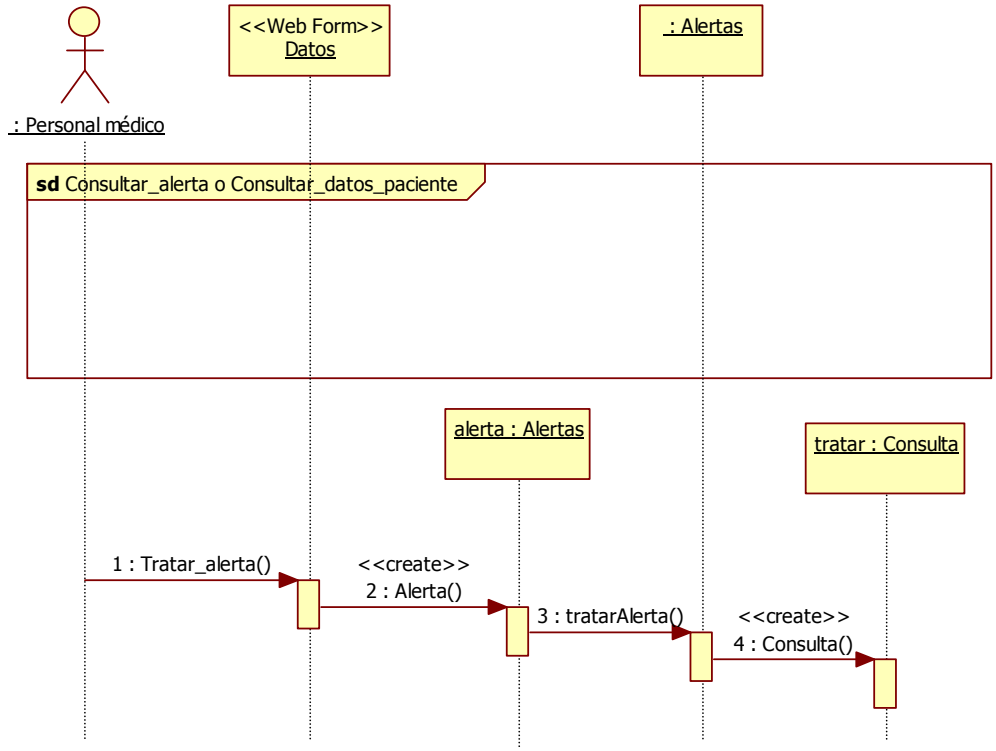

Figura 3.37.

- **Buscar_datos_paciente [UC-0007]**

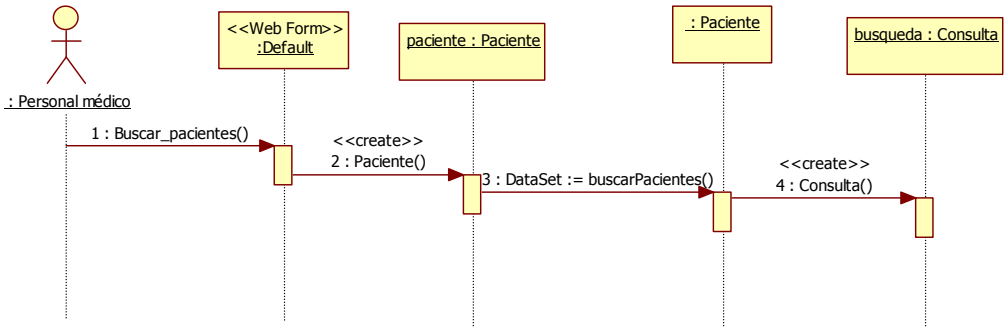

Figura 2.38.

<<Página incluida en la carpeta *Desplegables* del CD-ROM>>

<<Página incluida en la carpeta *Desplegables* del CD-ROM>>

Actualizar_graficas [UC-0010]

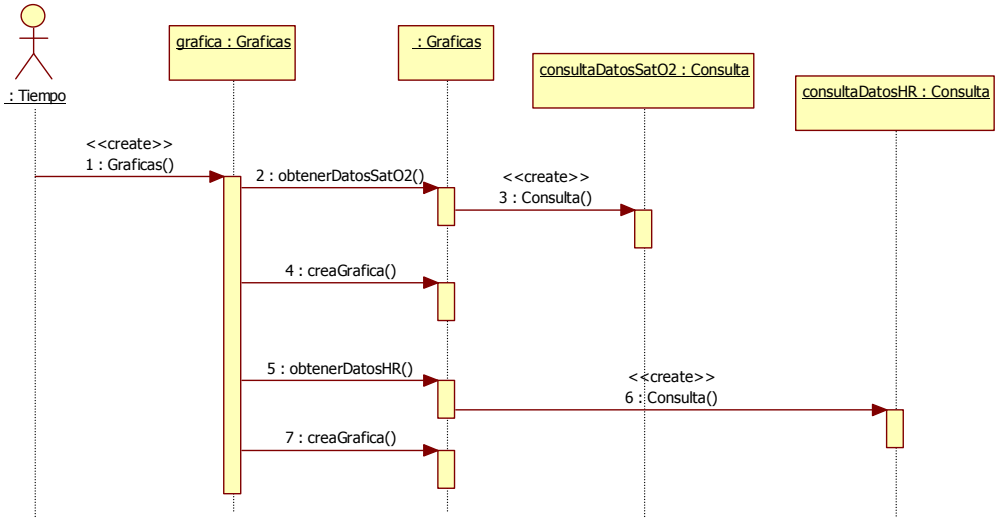

Figura 2.41.

2.3.3. Capa de persistencia

Las Bases de Datos a lo largo de los años han ido cambiando su forma de operar. Hay tres tipos de Bases de Datos:

• **Estructurales**: sólo se le pueden hacer consultas de bajo nivel desde un lenguaje de programación propio de la Base de Datos, están obsoletas.

• **Relacionales**: las consultas se realizan a través de un lenguaje de consultas estándar llamado SQL (Standard Query Language) y los datos se almacenan en tablas. Son las más utilizadas actualmente.

• **Orientadas a Objetos**: en un futuro lejano se prevé que los datos se almacenarán en Bases de Datos Orientadas a Objetos, almacenándose los datos en Objetos.

Con el fin de conseguir una independencia total sobre la base de datos empleada, decidí diseñar una capa de persistencia que aislara a la lógica de la aplicación de los accesos a base de datos, para posibles cambios futuros en el cambio de base de datos. La capa de lógica se encuentra comunicada con la capa de persistencia mediante una clase llamada *Consulta.cs.*

La persistencia se resuelve mediante el sistema gestor de base de datos Microsoft SQL Server, ya que proporciona mayor facilidad, comodidad y seguridad que un sistema tradicional de ficheros. Debido a que existe un apartado específico donde describo este sistema gestor de base de datos, no profundizaré más en su explicación.

El responsable de diseñar el esquema de la base de datos que utilizo fue Jose Antonio González Vesga, un compañero encargado de diseñar el servidor donde se alojará mi proyecto, la base de datos y otras diversas funcionalidades.

Durante el diseño aporté algunas ideas que creía necesarias para el mejor aprovechamiento de la aplicación y facilitar ciertas consultas. El resumen de las tablas que utilizo se muestra en el siguiente subapartado.

2.3.3.1. Descripción de las tablas utilizadas

• **tblLectura**: datos que se deben guardar de toda lectura hecha por un dispositivo.

tblLectura		Tamaño	Valor	Restricciones
idLectura	Bigint	-	not null	primary key
fecha	Datetime	-	not null	-
idpaciente	Bigint	-	not null	foreign key
idsensor	Bigint	-	not null	-
cadenaXml	Varchar	2000	not null	-

Figura 2.42.

• **tblLecturaDesglosado**: cada tabla tblLectura posee una de estas tablas asociadas donde se encuentra el xml desglosado.

tblLecturaDesglosado		Tamaño	Valor	Restricciones
idLectura	Bigint	-	not null	primary key
idDato	Bigint	-	not null	primary key
nombre	Nchar	30	not null	-
tipo	Nchar	10	not null	-
valor	Nchar	10	not null	-

Figura 2.43.

• **tblNotificacion**: tabla necesaria para almacenar las notificaciones que se produzcan y si están tratadas o no.

tblNotificacion		Tamaño	Valor	Restricciones
idNotificacion	Bigint	-	not null	Primary key
fecha	Datetime	-	not null	-
idpaciente	Bigint	-	not null	Foreign key
estadoPaciente	Nchar	10	not null	-
idDestinatario	Nchar	30	not null	-
textoMensaje	Varchar	50	not null	-
cadenaEstado	Xml	-	not null	-
tratada	Int	-	not null	

Figura 2.44.

• **tblPaciente**: datos de cada uno de los pacientes de la residencia.

tblPaciente		Tamaño	Valor	Restricciones
idPaciente	Bigint	-	not null	primary key
Nombre	Varchar	20	not null	-
Apellidos	Varchar	50	not null	-
foto	Varchar	20	-	-
Habitación	Nchar	10	not null	-
EstadoPaciente	Varchar	50	not null	(‘rojo’, ‘verde’)
Sexo	Nchar	10	not null	(‘hombre’, ‘mujer’)
Edad	Int	-	not null	-

Figura 2.45.

IMPLEMENTACIÓN

2.4. Entorno de programación: Visual Studio 2008

En este apartado comento el entorno de programación utilizado para el desarrollo del proyecto: Microsoft Visual Studio 2008.

Microsoft Visual Studio 2008 cumple con la visión de Microsoft sobre aplicaciones inteligentes, al permitir que los desarrolladores creen rápidamente aplicaciones conectadas con la más alta calidad y con atractivas experiencias de usuario.

Visual Studio 2008 ofrece avances para desarrolladores en función de los siguientes tres pilares:

- Desarrollo rápido de aplicaciones
- Colaboración eficiente entre equipos
- Innovación en experiencias de usuario

Este entorno de programación de Microsoft ofrece herramientas de desarrollo avanzadas, funciones de debugging, funciones para bases de datos, funciones innovadoras que permiten crear rápidamente aplicaciones futuras para distintas plataformas… Incluye mejoras como diseñadores visuales para un desarrollo más rápido con .NET Framework 3.5, mejoras sustanciales en las herramientas de desarrollo Web y mejoras de programación que aceleran el desarrollo a partir de todo tipo de datos.

Visual Studio 2008 ofrece todo el soporte requerido para marcos y herramientas, necesario para crear aplicaciones AJAX para Web, completas y expresivas. Los desarrolladores pueden aprovechar estos marcos con una faceta para clientes y otra faceta para servidores, marcos que permiten construir fácilmente aplicaciones Web concentradas en los clientes, que se integran con cualquier proveedor de datos de back-end, que se ejecutan dentro de cualquier explorador moderno, y que tienen acceso total a los servicios de aplicaciones ASP.NET y de la plataforma Microsoft.

Una de las características que ofrece este entorno de programación es Microsoft IntelliSense. Dicha característica es una potente herramienta para el programador, que al saber manejar, agiliza enormemente la escritura de código. Proporciona la capacidad de completar el símbolo de los nombres que el programador está escribiendo, sirve como documentación y desambiguación de los nombres de variables, funciones y métodos… A continuación enumero sus principales funciones:

- **Información de parámetros:** abre una lista de parámetros que proporciona información sobre el número, los nombres y los tipos de parámetros necesarios para una función o un procedimiento almacenado. El parámetro en negrita indica el siguiente parámetro que se necesita al escribir una función o un procedimiento almacenado.

  La lista de parámetros también aparece para las funciones anidadas. Si se escribe una función como parámetro de otra función, la lista de parámetros muestra los parámetros de la función interna. Así, cuando la lista de parámetros de la función interna está completa, pasa a mostrar los parámetros de la función externa.

- **Información rápida:** muestra la declaración completa de cualquier identificador del código. Al mover el puntero del ratón sobre un identificador, su declaración se muestra en una ventana emergente de color amarillo.

- **Palabra completa:** escribe el resto de una variable, un comando o un nombre de función una vez que se han especificado los suficientes caracteres para identificar de forma exclusiva esos elementos.

Por todas estas características, la potencia que ofrece y debido a que integra el lenguaje de programación orientado a objetos C#.NET, el acceso a bases de datos gracias a ADO.NET y herramientas para desarrollo en Internet como son ASP.NET, todas ellas tecnologías utilizadas en la realización de mi aplicación Web, es por ello que Visual Studio 2008 ha sido el entorno de programación escogido para desarrollar mi proyecto.

2.5. Interfaz

ASP.NET trata de diferenciar en todo momento el código destinado para la lógica de la aplicación del código empleado para definir la interfaz. Para ésta última me he servido principalmente de código HTML, acompañado de dos tecnologías como son las CSS y una nueva característica de ASP.NET como son las Master Page, que a continuación explico.

2.5.1. CSS

Una 'hoja de estilo' es una plantilla, guía o conjunto de instrucciones, que dicta al navegador como debe mostrar el contenido de una o varias páginas Web. Cualquier cambio en la plantilla de estilo se refleja en un cambio inmediato en la apariencia de las páginas relacionadas, sin necesidad de modificar físicamente éstas.

El creador de la página define estilos (tamaño de letra, color, fuente, márgenes…) que se visualizarán con preferencia a los definidos por defecto en el propio navegador. Si después desea cambiar la apariencia bastará que cambie la definición de estilo.

Las CSS (hojas de estilo en cascada) mejoran las posibilidades de diseño y presentación de documentos en la red, facilitando además su mantenimiento, ya se trate de un único archivo HTML, o de grandes sitios, con multitud de páginas. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

En el desarrollo de mi proyecto he utilizado una plantilla de dominio público ya desarrollada, sobre la cual realicé los cambios que consideré oportunos para adaptarlos a mis necesidades.

En .NET, el archivo CSS debe encontrarse ubicado en el directorio App_Themes del proyecto para poder utilizar las hojas de estilo. En el caso de que se quieran utilizar dos o más estilos se deben crear subdirectorios. En mi caso se utiliza tan solo un estilo llamado Style.css.

2.5.2. Master Page

Una Master Page (página maestra) es una página que contiene marcas y controles que deben ser compartidas a través de múltiples páginas de nuestra aplicación Web ASP.NET. La característica de las Master Pages nos proporciona la habilidad de definir una estructura y unos elementos de interfaz comunes para nuestro sitio, tales como la cabecera de página o la barra de navegación, en una ubicación común, para ser compartidos por varias páginas del sitio. Esto evita la duplicación innecesaria de código para estructuras o comportamientos del sitio que son compartidos.

La principal ventaja de la Master Page es que tan sólo tenemos que crear una o varias plantillas para todo nuestro sistema. Este hecho facilitará enormemente el mantenimiento, ya que solamente con modificar la Master Page, los cambios se verán reflejados en todas las páginas que la utilicen. Otra ventaja muy importante es que tendremos soporte en diseño WYSIWYG (lo que ves, es lo que obtienes), en las páginas aspx, y podremos ver en todo momento cómo será la presentación final.

La definición de una Master Page es como la de cualquier página. Las Master Pages pueden contener marcar, controles, código o cualquier combinación de estos elementos. Sin embargo, una Master Page puede contener un tipo especial de control llamado **ContentPlaceHolder**. Un ContentPlaceHolder define una región de la representación de la master page que puede substituirse por el contenido de una página asociada a la maestra. Un ContentPaceHolder también puede contener contenido por defecto, por si la página derivada no necesita sobrescribir este contenido. Una página que se asocia a una Master Page se llama Content Page (Página de Contenido).

Para diferenciar una Master Page de una página normal, la Master Page se guarda con una extensión .master. Una página puede derivar de una Master Page simplemente con definir un atributo MasterPageFile en su directiva Page:

<%@ Page MasterPageFile=”Site.master” %>

Las Content Pages también pueden ser Master Pages. Esto quiere decir que es posible derivar una Master Page a partir de otra Master Page. Por ejemplo, podríamos tener una Master Page de primer nivel que represente la cabecera/pie de página y la navegación global del sitio, y después Master Pages separadas que deriven de esta Master para definir los aspectos de las diferentes sub-secciones del sitio. Las Content Pages derivarán de la página maestra correspondiente a la sección a la que pertenece la Content Page.

2.6. Negocio: C#

C# (leído en inglés "C Sharp" y en español "C Almohadilla") es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el **lenguaje nativo de .NET**

La sintaxis y estructuración de C# es muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic.

Un lenguaje que hubiese sido ideal utilizar para estos menesteres es Java, pero debido a problemas con la empresa creadora del mismo -Sun-, Microsoft ha tenido que desarrollar un nuevo lenguaje que añadiese a las ya probadas virtudes de Java las modificaciones que Microsoft tenía pensado añadirle para mejorarlo aún más y hacerlo un lenguaje orientado al desarrollo de componentes.

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. El hecho de ser relativamente reciente no implica que sea inmaduro, pues Microsoft ha escrito la mayor parte de la BCL usándolo, por lo que su compilador es el más depurado y optimizado de los incluidos en el *.NET Framework SDK*

2.7. Datos: SQL Server

Microsoft SQL Server es un sistema de gestión de base de datos relacionales (SGBD) basado en el lenguaje SQL (**S**tructured **Q**uery **L**anguage, Lenguaje Esctructurado de Consultas) capaz de poner a disposición de muchos usuarios grandes cantidades de datos de forma simultanea.

Microsoft SQL Server constituye la alternativa de Microsoft a otros potentes sistemas gestores de bases de datos que se encuentran en el mercado como son Oracle, Sybase ASE o MySQL.

Para la realización del presente proyecto he utilizado SQL Server 2005 debido a las anteriores características y a su facilidad para ligarse a Visual Studio.

2.8. Problemas técnicos

En este subapartado voy a citar algunas de las dificultades técnicas a las que me he tenido que enfrentar durante la realización de la implementación y la solución que encontré para cada una de ellas:

- **Posicionamiento de elementos en aplicaciones Web:** En una aplicación de escritorio es muy sencillo asignar la posición que ocupará un determinado elemento. En el modo de diseño del entorno de programación que estemos utilizando, simplemente tenemos que arrastrarlo hasta la posición que deseemos. En una aplicación Web los elementos se irán colocando uno tras otro, en fila, siguiendo una distribución conocida como *flowlayout*, por lo que deberemos buscar alternativas.

  En el diseño de la interfaz es de gran ayuda los archivos de estilo .css, pero para situar elementos como tablas, botones, imágenes resulta tedioso. Personalmente solucioné el problema utilizando tablas como rejillas donde colocar los elementos y si esto no era suficiente, cambiaba la configuración de la interfaz de Visual Studio 2008 a *gridlayout*, que permite colocar libremente los elementos como si se tratara de una aplicación de escritorio. Se debe tener cuidado con esta solución si no queremos llevarnos desagradables sorpresas cuando cambiemos de navegador.

  Para cambiar a *gridlayout*: Herramientas; Opciones; Diseñador HTML; Aplicación de estilos CSS y finalmente marcamos el último cuadrito de la zona de la derecha que dice: "Cambiar el posicionamiento a absoluto para controles…".

- **Asignación de eventos a controles generados dinámicamente:** Visual Studio facilita enormemente la asignación de eventos a cualquier control que tengamos en la interfaz. Con el simple hecho de hacer doble click sobre el que deseemos, se generará un método al cual estará asociado dicho control y que será ejecutado cuando éste tenga que entrar en acción. Esto no es tan sencillo cuando se generan elementos dinámicamente. Para este tipo de controles, Visual Studio dispone de los delegados y eventos. De esta forma puedes añadir funcionalidad a un control en el momento de su creación. A continuación muestro un pequeño ejemplo para aclarar la explicación donde se crea un botón en tiempo de ejecución, a dicho botón se le asigna un método, en el cual se identifica el control que lo ha llamado y se le da la funcionalidad deseada :

```
Button Btratar = new Button();
Btratar.Click += new EventHandler(Btratar_Click);
…
…
 void Btratar_Click(object sender, EventArgs e)
{
     Button boton = (Button)sender;
      boton.Visible = false;
}
```

Figura 2.46.

- **Paso de parámetros entre formularios:** Para pasar datos entre formularios existen diferentes posibilidades como son la utilización de variables de sesión, por POST, por GET… cada una de ellas tiene sus limitaciones, ventajas e inconvenientes. Finalmente me decante por el método GET. La desventaja de esta opción es que podemos observar en la URL los datos que se pasan y si estos son contraseñas o datos privados no es muy buena opción, además de que hacemos vulnerable la aplicación.

  Los usuarios finales de esta aplicación no van a realizar ningún uso malintencionado, pero aún así, decidí utilizar un método de encriptación que presenta Microsoft para la URL. Consiste simplemente en llamar a una función que codificará el texto que le pasemos. A continuación muestro un pequeño ejemplo:

```
…
string id = Server.UrlEncode(""+idpaciente+"");
consultar.NavigateUrl = "Datos.aspx?id="+id+"";
…
```

Figura 2.47.

- **Utilización de UpdatePanel:** se trata de un control de servidor disponible en la versión *ASP.NET AJAX* que permiten actualizar las partes seleccionadas de una página en lugar de actualizar toda la página con una devolución de datos. Esto se conoce como *actualización parcial de la página*. Una página Web ASP.NET que contiene un control ScriptManager y uno o varios controles UpdatePanel puede participar automáticamente en actualizaciones parciales de la página, sin un script de cliente personalizado.

  Debido a la posibilidad de situar varios UpdatePanel en el mismo formulario Web, al actualizarse uno de ellos, actualizaba a todos los demás. Para evitar esto, se debe establecer la propiedad *UpdateMode* como *Conditional* además de habilitar un control *ScriptManager*, indispensable en un formulario Web que contenga controles Ajax, deberemos establecer la propiedad *EnablePartialRendering* del propio *ScriptManager* como *true*, la cual habilitará las actualizaciones parciales de página.

```
…
<asp:ScriptManager ID="ScriptManager" EnablePartialRendering=”true”
runat="server"/>
<asp:UpdatePanel ID="UpdatePanel1" UpdateMode="Conditional"
runat="server">
    <ContentTemplate>
        …
    </ContentTemplate>
</asp:UpdatePanel>
…
```

Figura 2.48.

- **Localización de pacientes:** Para cumplir con la necesidad de representar la localización de los pacientes estudié varias posibilidades que muestro a continuación:

    - Los famosos *applets* de Java, estándar para la mayoría de los navegadores, con gran funcionalidad pero un tanto desfasados. Fueron mi primera opción, pero los descarté debido a las limitaciones que presentan a la hora de integrarlos en una aplicación .NET.

    - Los controles activeX de Microsoft. Estos quedaron rápidamente descartados debido a que únicamente se pueden ejecutar en Internet Explorer y uno de los requisitos era que la aplicación fuera estándar para los principales navegadores existentes.

    - API de Google Maps. Esta opción era la mejor de todas. Utilizaría el motor de Google, me aseguraba un aspecto cuidado, la reutilización de código… pero el problema era su mala integración con la tecnología .NET al estar desarrollado en JavaScript. Debido a que era la mejor opción decidí seguir investigando, hasta q finalmente, encontré una biblioteca que adaptaba el API de Google Maps a la tecnología ASP.NET. Por ello, esta fue la opción escogida.

- **Generación de gráficas:** La realización de las gráficas fue todo un desafío. Su implementación no formaba parte de los requisitos iniciales, pero fue el propio personal médico el que nos pidió su inclusión debido a la mayor información que aportan y a su facilidad de uso.

  Para implementar las gráficas probé con varias bibliotecas desarrolladas especialmente para ASP.NET hasta que encontré la que cumplía mis necesidades. Por mencionar algunas utilicé: *WebChart*, *ZedGraph* y finalmente me decanté por *ASP.NET Chart Control* ya que a parte de ser gratuita, tenía un aspecto muy cuidado, estaba desarrollada por Microsoft y permitía funcionalidad mucho mayor que cualquier otra.

  Para todas ellas tuve que aprender a manejarlas, ver las posibilidades que me ofrecían y ponerlas a prueba de cara al cumplimiento de mis objetivos.

- **Actualización de datos en gráficas:** para llevar a cabo esta funcionalidad ligué ASP.NET Chart Control con AJAX. Con esto conseguí que cada cierto tiempo, de forma automática la aplicación consulta los diez últimos datos obtenidos y los representa en la gráfica. La idea es sencilla pero su implementación me produjo varios quebraderos de cabeza.

  En primer lugar tuve que formatear las fechas para que se visualizasen de forma correcta. Para ello me serví de un objeto de tipo *TimeSpan* y a continuación anteponer ceros a los minutos y segundos si su valor únicamente tenía un digito (las cinco, seis minutos y 2 segundos lo representaba como: “17:5:2” y debía representarse como: “17:05:02”).

  En segundo lugar, tuve que realizar una consulta que obtuviese los diez último datos. Para ello ordené los resultados de forma decreciente y almacené los 10 primeros en un vector en orden inverso, es decir, el más reciente en la última posición del vector y el más

antiguo en la primera. Almacené los datos de esta forma para que a la hora de representar, empezase por el más antiguo y fuera añadiendo a su derecha puntos más recientes.

- **Superposición de imágenes en Google Maps:** Una vez escogida la forma en que iba a representar la localización de los pacientes surgió otra duda: Google Maps no dispone del interior de edificios. Esto limitaba enormemente la funcionalidad llegando a pensar en descartar Google Maps como solución para la localización.

  Buscando información encontré la posibilidad de superponer imágenes mediante archivos KML. Esta fue la gran solución que encontré al problema, porque este tipo de archivos son un estándar para Google Maps de forma que me aseguro una solución profesional y eficaz. No detallo más la utilización de estos archivos porque hablaré de ellos más adelante, en el apéndice B, donde comento las APIS utilizadas.

Capítulo 3

PRUEBAS

3.1. Introducción

Uno de los últimos pasos realizados tras terminar la implementación del código fuente, fue la exhaustiva realización de pruebas para asegurarme del correcto funcionamiento de la aplicación. El objetivo de esta fase no es convencerse de que el programa funciona bien, sino ejercitarlo con la peor intención a fin de encontrarle fallos. Para ello he llevado a cabo una serie de casos de prueba diseñados para determinar si los requisitos establecidos durante el diseño del proyecto se han cumplido total o parcialmente.

El desarrollo de las pruebas se ha llevado a cabo desde dos perspectivas diferentes:

- Las **pruebas de caja blanca** son aquellas que se realizan sobre funciones internas de un módulo. Se comprueba así la lógica interna de un programa.

- En las **pruebas de caja negra** no conocemos la implementación del código, sólo la interfaz. Se centran en lo que se espera de un módulo, por ello se denominan pruebas funcionales, y el probador se limita a suministrar datos de entrada y a estudiar la salida.

3.2. Pruebas de caja blanca

Estas pruebas también llamadas estructurales o de caja trasparente, se han ido realizando a medida que desarrollaba el código, de forma que cada método, clase y la colaboración entre todas ellas han sido probadas.

A medida que probaba el código he tratado de recorrer todos los posibles caminos de ejecución; he testado las operaciones lógico-aritméticas; he revisado la definición y uso de las variables y finalmente he comprobado los bucles. Para muchas de estas tareas he contado con la ayuda de Visual Studio, que ofrece numerosas herramientas para detención de errores como: alertar de numerosos fallos en tiempo de diseño, sin necesidad de ejecutar el código; proporciona la creación de puntos de interrupción para ejecuciones controladas…

3.3. Pruebas de caja negra

Este tipo de pruebas está enfocado a comprobar que los requisitos funcionales se han cumplido. Es decir, la prueba de caja negra permite obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. No son una alternativa a las pruebas de caja blanca sino que se han de realizar las dos por separado y de forma complementaria, con el fin de detectar diferentes tipos de errores.

Para organizar los casos de prueba hemos realizado su organización en la siguiente tabla:

Prueba	Resultado obtenido	V
Descripción de la prueba: interacciones, caracterización de los datos de entrada, observaciones a hacer sobre la interfaz de usuario, etc.…	Resultado observado al ejecutar la prueba.	Valoración del resultado de la prueba: - Correcto - Incorrecto

Figura 3.1.

En primer lugar se comprobó uno a uno si todos los requisitos funcionales habían sido cubiertos y en segundo lugar han sido probados cada caso de uso:

Buscar paciente

Prueba	Resultado	V
Buscar un paciente existente.	El sistema muestra una tabla con una fila para el paciente.	Correcto
Buscar un paciente inexistente.	El sistema avisa de que no existe un paciente coincidente con los datos de búsqueda introducidos.	Correcto
Buscar varios pacientes.	El sistema muestra una tabla con tantas filas como pacientes coincidentes existan.	Correcto
Pulsar el botón buscar sin rellenar ningún campo de búsqueda.	El sistema muestra una tabla con todos los pacientes almacenados en la base de datos.	Correcto
Pulsar botón reestablecer.	El sistema prepara el formulario para una nueva búsqueda.	Correcto
Introducir números en el cuadro de búsqueda del nombre o apellidos.	El sistema impide la escritura.	Correcto
Introducir letras en el cuadro de búsqueda de la edad.	El sistema impide la escritura.	Correcto

Figura 3.2.

Datos paciente

Prueba	Resultado	V
Consultar los datos de un paciente resultado de una búsqueda.	El sistema abre un nuevo formulario con los datos de dicho paciente.	Correcto
Consultar datos médicos paciente monitorizado para algún dato.	El sistema muestra las gráficas para los datos del paciente.	Correcto
Consultar datos médicos paciente no monitorizado.	El sistema informa sobre que datos no está monitorizado el paciente.	Correcto
Consultar historial paciente con incidencias.	El sistema muestra una tabla con las incidencias del paciente.	Correcto
Consultar historial paciente sin incidencias.	El historial se encuentra vacío.	Correcto

Figura 3.3.

Alertas

Prueba	Resultado	V
Consultar alerta.	El sistema muestra los datos del paciente.	Correcto
Tratar alerta.	El sistema elimina la alerta de la lista.	Correcto
Actualizar alertas automático	El sistema actualiza la lista de alertas de forma automática.	Correcto
Actualizar alertas manual.	El sistema actualiza la lista de alertas.	Correcto

Figura 3.4.

Localización

Prueba	Resultado	V
Situar en mapa paciente localizado.	El sistema abre un nuevo formulario con el mapa centrado en la posición del paciente.	Correcto
Situar en mapa paciente no localizado.	El sistema no muestra dicha opción.	Correcto
Consultar datos paciente localizado.	El sistema abre un bocadillo con los datos personales del paciente.	Correcto

Ampliar datos paciente localizados.	El sistema amplia el anterior bocadillo con las gráficas del paciente.	Correcto
Localizar paciente manual.	El sistema permite navegar por las plantas y el mapa libremente.	Correcto
Actualizar localización manual	El sistema actualiza la localización y el estado de los pacientes	Correcto

Figura 3.5.

Acceso ayuda

Prueba	Resultado	V
Indicar al sistema que queremos consultar la ayuda.	El sistema despliega una lista con la ayuda.	Correcto

Figura 3.6.

Las pruebas anteriores han sido realizadas sobre la aplicación final, es por ello que el resultado de todas ellas sea el esperado. Sin embargo esto no fue así durante el desarrollo del proyecto. Los resultados incorrectos fueron numerosos, causados principalmente a errores de programación, de diseño, o de planteamiento. En el capítulo 5, subapartado 5.1.1 describo una serie de problemas reseñables que aparecieron y la solución que encontré a los mismos.

3.4. Pruebas propias de una aplicación Web

Una aplicación Web tiene aspectos comunes con cualquier otro tipo de aplicación, como podría ser una aplicación de escritorio, pero también tiene peculiaridades propias. Es por ello que se le debe dedicar un apartado específico de pruebas.

El contenido de una aplicación Web puede albergar errores, desde simples fallos ortográficos, hasta la violación de las leyes de propiedad intelectual. Es por ello que se debe tener cuidado en la presentación y contenido de la aplicación, además de prestar especial atención a la información procedente de la base de datos.

La facilidad de uso es la capacidad de que cualquier usuario pueda realizar las funciones que desarrolla la aplicación sin ningún tipo de dificultad. Es decir, que la navegabilidad a través de nuestras páginas sea sencilla y que el usuario pueda acceder a la selección que desee en cualquier momento.

En mi caso, para llevar a cabo esta prueba, conté con la ayuda de compañeros. Se trata de una aplicación sencilla de usar, pero para realizar una prueba exhaustiva, proporcioné el manual de usuario y a continuación les pedí que realizasen varias pruebas. En la gran mayoría de los casos el resultado fue satisfactorio.

Capítulo 4

MANUALES

4.1. Manual de instalación

En este apartado se describe la forma de realizar la instalación de la Aplicación Web que forma el proyecto, la cual, junto con la base de datos, necesitarán de una instalación en la máquina de servidor que se desee utilizar para tal fin. Una vez instalados los recursos que se explican a continuación, el asistente de instalación le ayudará a crear un directoria virtual donde almacenar la aplicación Web.

4.1.1. Instalación de Internet Information Server (IIS)

Para poder llevar a cabo una correcta instalación y ejecución de la Aplicación Web será obligatorio disponer en la máquina reservada para ese fin Internet Information Server (IIS) para la publicación de la aplicación.

Internet Information Server (IIS) es un servidor de páginas Web de la plataforma de Microsoft, por ello es el componente ideal para mi aplicación, ya que está desarrollada en ASP.NET.

Para poder realizar la instalación se necesita el CD de instalación del sistema operativo, en mi caso, Windows XP. Para poder instalarlo debe seleccionar la opción “Instalar componentes opcionales de Windows” en la ventana que se abre al introducir el CD que se muestra en la imagen *Figura 4.1*.

Figura 4.1.

Si no se produce el autoarranque podemos ir a *Panel de control>Agregar o quitar programas>Agregar o quitar componentes de Windows.* Una vez aquí se elige la opción *Servicios de Internet Information Server (IIS)* como vemos en la imagen *Figura 4.2.*

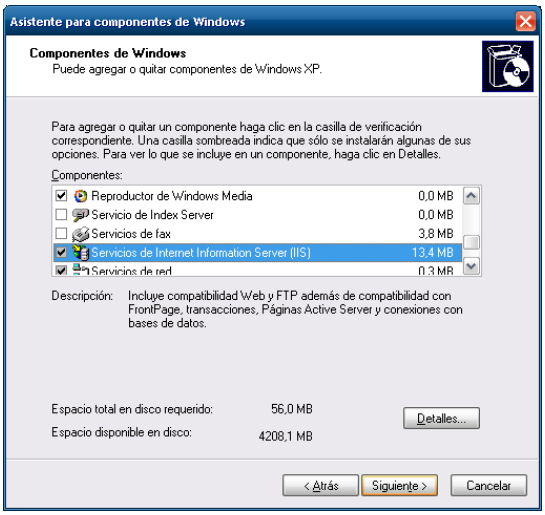

Figura 4.2.

4.1.2. Instalación Framework ASP.NET

Necesitaremos tener instalado ASP.NET versión 2.0.50727 para la correcta interpretación del lenguaje en que se encuentra desarrollada la aplicación. Podremos encontrar este archivo fácilmente en Internet, en páginas oficiales de Microsoft.

Una vez descargado e instalado, tendremos que dar de alta el Framework en nuestro Directorio virtual. Para ello pulsamos con el botón secundario sobre nuestro directorio virtual, seleccionamos *Propiedades*, seleccionamos la pestaña *ASP.NET* y en la lista desplegable *Versión de ASP.NET*, escogemos dicha versión como vemos en la imagen *Figura 4.3.*

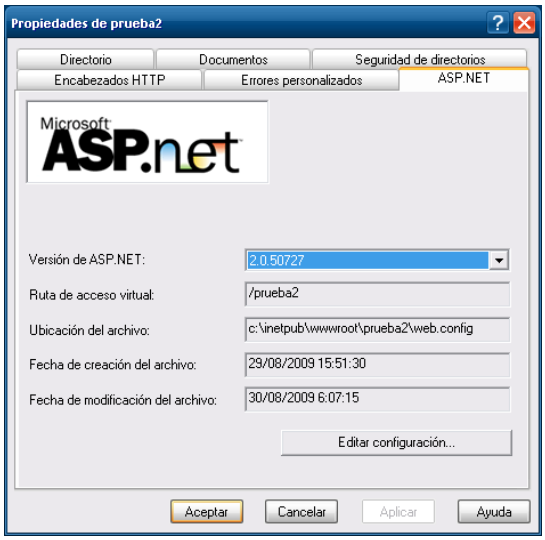

Figura 4.3.

4.1.2. Instalación de SQL Server

Para el funcionamiento correcto de la base de datos necesitaremos SQL Server 2005 Management Studio para la ejecución de consultas, bien sea en la misma máquina donde se instalará la aplicación Web o en otra habilitada para dicho uso, ya que no tiene por qué ser en la misma.

Debe existir un usuario que disponga de permisos en la base de datos a la que queremos conectarnos, más concretamente la propiedad *connect*, para que pueda realizar la conexión a dicha base de datos. Resaltar que no valdrá solo con la copia de los archivos al servidor, sino que habrá que realizar el enlazado a SQL Server 2005.

En la configuración de SQL Server 2005 deberá tener activado las conexiones remota, de no ser así podría producirse algún error. Para ello abrimos la consola *Configuración de superficie de SQL Server 2005*. Una vez abierta escogemos la opción *Configuración de superficie para servicios y conexiones*, como vemos en la imagen *Figura 4.4.*

Figura 4.4.

Dentro del motor de la base de datos, pulsamos la opción *Conexiones remotas* y activamos las *Conexiones locales y remotas.* Este cambio no se aplicará hasta que se reinicie el servicio, por ello nos vamos a la opción *Servicio*, lo detenemos y a continuación lo iniciamos de nuevo para que se apliquen los cambios realizados.

4.2. Manual de usuario

4.2.1. Inicio de la aplicación

Al tratarse de una aplicación Web, su inicio es común al de todas las aplicaciones de este tipo. No hay más que abrir una ventana de un navegador Web que utilice el protocolo http, típicamente Explorer o Mozilla y escribir la dirección correspondiente al servidor donde se encuentre instalado. Un ejemplo sería:

http://nombredeservidor/nombrecarpeta/default.aspx

Donde el nombre del servidor, podrá venir dado bien por una IP o por un nombre DNS y el nombre de la carpeta será aquel con el que se haya publicado en el servidor.

Finalmente, la aplicación muestra la pantalla de inicio, y otorga el control al usuario.

4.2.2. Usabilidad

Todas las pantallas de la aplicación mantienen múltiples características comunes:

- Cabecera con el título de la aplicación.
- Menú de navegación en la parte superior.
- Zona central donde se mostrarán los distintos contenidos y formularios.
- Zona derecha con ayuda y lista de alertas.

Toda la aplicación está desarrollada para un único tipo de usuario. Es decir, cualquier usuario tendrá accesibilidad a toda la funcionalidad de la aplicación, por tanto no haré distinciones durante la explicación.

4.2.2.1. Búsqueda

Se trata del inicio de la aplicación. Está formado por varios campos de texto y dos listas desplegables, que nos facilitarán la búsqueda de uno o varios pacientes. Simplemente deberemos rellenar aquellos controles que deseemos y pulsar sobre el botón *Buscar*, como indica la imagen *Figura 4.5*.

Figura 4.5.

Tras pulsar sobre el botón *Buscar*, se mostrará una lista de los pacientes, en la parte superior, que concuerden con la búsqueda realizada, desplazando los controles de búsqueda hacia abajo, para facilitar la vista de los resultados. Si se pulsa el botón *Buscar* sin haber rellenado ningún control de búsqueda, se mostrarán todos los pacientes existentes en la residencia.

Si la búsqueda ha sido satisfactoria y deseamos ampliar la información de un paciente, simplemente deberemos pulsar sobre *Consultar*, remarcado en la imagen *Figura 4.6*. De esta forma se abrirá un nuevo formulario formado por tres pestañas donde podremos visualizar toda la información disponible del paciente.

Figura 4.6.

Si deseamos reiniciar la búsqueda, simplemente deberemos pulsar sobre el botón *Reestablecer*, para eliminar los resultados obtenidos y reiniciar los controles de búsqueda.

4.2.2.2. Datos

A este formulario se ha podido acceder desde diferentes caminos, pero su funcionalidad siempre es la misma: muestra los datos del paciente que estamos consultando. Para ello dispone de tres pestañas (*Datos Personales, Datos Médicos, Historial Incidencias*) con diferentes contenidos.

- **Datos Personales:** en esta primera pestaña, activada de forma predeterminada, encontramos todos los datos personales del paciente consultado. Además, nos muestra el estado del paciente mediante un representativo icono y la posibilidad de tratar al paciente en caso de encontrarse en *Alerta*. Finalmente, si el paciente se encuentra localizado, nos muestra un enlace que nos abrirá un mapa centrado en la posición donde se encuentre el paciente.

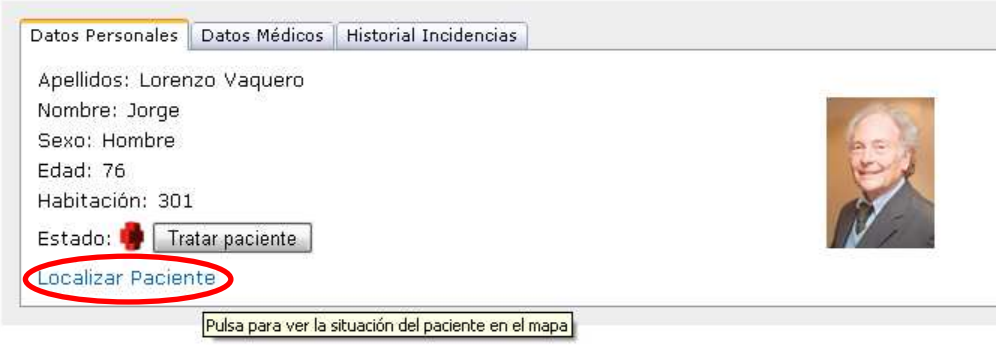

Figura 4.7.

- **Datos Médicos:** en esta segunda pestaña encontramos las gráficas para las cuales está monitorizado el paciente. Como no todos los pacientes están monitorizados para todas las medidas, en lugar de aparecer una gráfica en blanco, se muestra un mensaje avisando de dicha situación como podemos ver en la imagen.

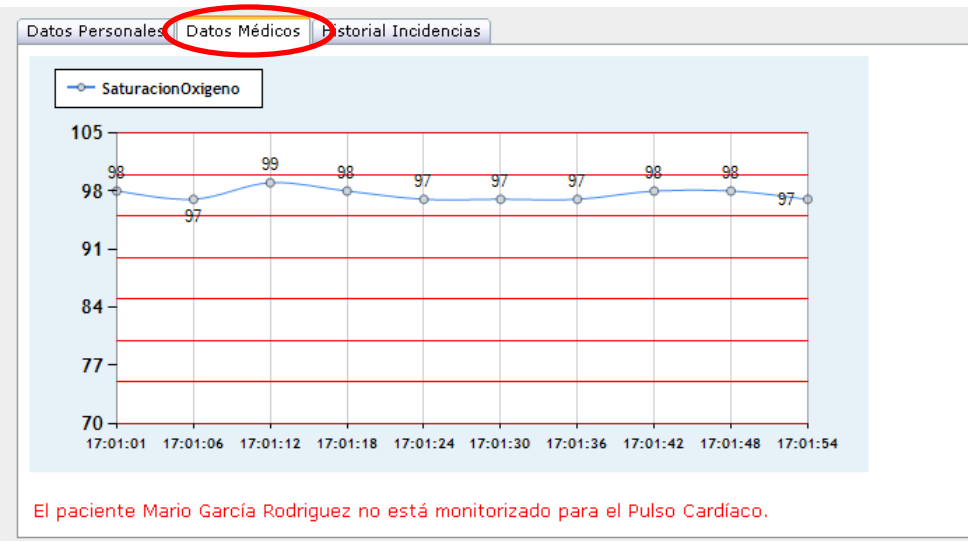

Figura 4.8.

- **Historial Incidencias:** en el caso de que hayan existido alertas sobre el paciente en cuestión, se mostrará una tabla descriptiva

Figura 4.9.

4.2.2.3. Localización

En el siguiente formulario podremos consultar la posición de todos los pacientes que se encuentren localizados. Está formado por tres pestañas, cada una correspondiente a una de las plantas de la residencia, de forma que podremos controlar a los pacientes que se encuentren en el exterior (situándonos en la planta primera) y a los que se encuentren en el interior de la residencia (situándonos en la planta que deseemos).

Los pacientes se encuentran representados por pequeños globos con colores representativos según su estado. Situando el cursor sobre uno de estos globos, se abrirá una pequeña ventana con la información personal del paciente. Si se desean consultar los datos médicos del paciente, simplemente bastará con pulsar sobre el icono que remarcamos en la imagen. Esto ampliará la ventana anterior y mostrará las gráficas del paciente.

Figura 4.10.

También existe la posibilidad de interactuar con el mapa. Podremos desplazarnos y modificar el zoom del mismo. Para ello podemos utilizar los controles existentes en la parte superior izquierda del mapa, para un control más sencillo, o hacer uso del cursor del ratón. Si optamos por esta segunda opción simplemente deberemos mantener pulsado sobre el mapa y mover el cursor para desplazarnos por él. Para aumentar el zoom simplemente deberemos hacer doble click sobre el mapa con el cursor derecho y doble click con el cursor izquierdo para disminuir el zoom del mapa.

Para actualizar las posiciones de los pacientes, simplemente deberemos actualizar la página de nuestro navegador.

4.2.2.4. Ayuda

En todos los formularios, en la parte derecha de cada uno de ellos, encontramos una lista despegable que corresponde con la ayuda de dicho formulario.

Para visualizar la ayuda no tendremos más que pinchar sobre el recuadro que marcamos en la imagen y se desplegará una breve explicación acerca de la funcionalidad del formulario en el que nos encontramos. Si volvemos a pinchar, la lista recuperará su posición inicial.

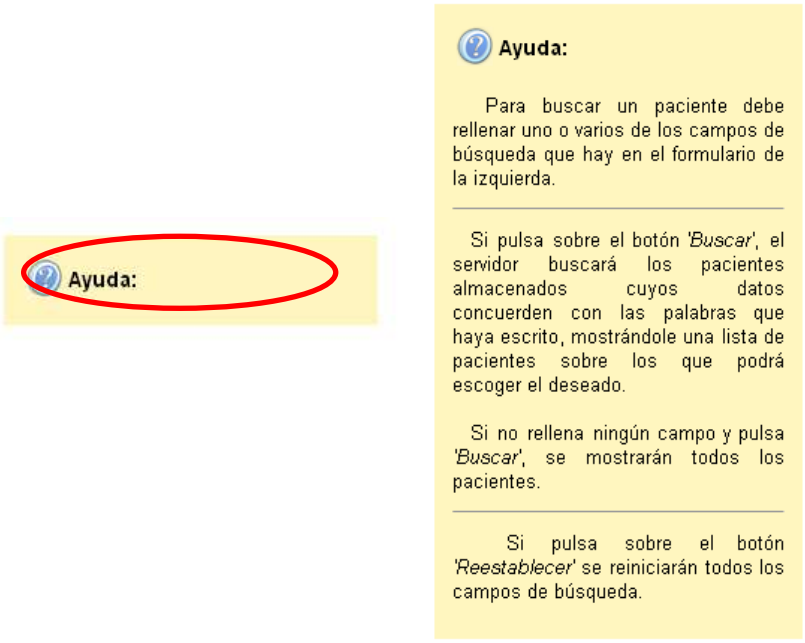

Figura 4.11.

4.2.2.5. Alertas

Al igual que la ayuda, existe un recuadro en la parte derecha de la página en cada uno de los formularios reservado para albergar la lista de alertas existentes en dicho momento. Se trata de una lista desplegable que podremos minimizar o extender a nuestro gusto.

La lista está formada por una serie de breves indicaciones representativas de la alerta a la que se refieren. Para consultar una alerta simplemente deberemos pulsar sobre el nombre que la representa, como se puede ver en la imagen, y nos dirigirá al formulario que contiene tres pestañas, anteriormente explicado, con los datos personales del paciente, los datos médicos y el historial de incidencias.

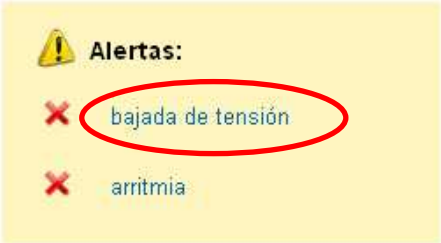

Figura 4.12.

Dicha lista se actualiza automáticamente, excepto si nos encontramos en el formulario de *Localización*, donde las alertas se actualizarán mediante la actualización de nuestro navegador, pulsando la tecla F5, por ejemplo.

Capítulo 5

CONCLUSIONES

5.1. Conocimientos adquiridos

En este quinto y último apartado me gustaría comentar los conocimientos que me ha aportado la realización de este proyecto. Existe un segundo subapartado donde comentaré algunos de los problemas técnicos que me surgieron durante el desarrollo y la solución que encontré.

En primer lugar este proyecto me ha enseñado a enfrentarme a una aplicación de cierta envergadura. He tenido que realizar las tareas de análisis y desarrollo, así como la implementación y las pruebas, dando como resultado el proyecto que aquí se describe.

En segundo lugar la investigación necesaria para llevar a cabo cualquier tarea me ha aportado una serie de conocimientos teóricos de los que carecía. He adquirido una numerosa terminología, filosofías de trabajo, nuevas tecnologías, futuras líneas de desarrollo… indispensable de cara a mi incorporación al mercado laboral.

Y en tercer y último lugar, la realización del proyecto ha supuesto la adquisición de una gran cantidad de conocimientos prácticos. Desde el aprendizaje de la tecnología .NET hasta la utilización del API de Google Maps, pasando por el manejo del entorno de programación Visual Studio 2008, la utilización de AJAX... Para todo ello he necesitado conocer su filosofía, los conceptos que engloban, ver ejemplos de utilización, interactuar con la herramienta y finalmente ponerla en práctica.

5.2. Futuras mejoras

Todo proyecto es mejorable, por ello he pensado en una serie de ampliaciones que podrían hacerse sobre esta aplicación Web. La interfaz podría mejorarse con un diseño mucho más profesional, el código podría sufrir modificaciones buscando una mayor modularidad o la inclusión de diversos patrones, pero las mejoras que me dispongo a exponer están dirigidas a aumentar la funcionalidad de la misma.

5.2.1. Administración de pacientes

Otra de estas mejoras podría estar encaminada en incluir un apartado para la gestión de los pacientes de la residencia. Dar de alta nuevos pacientes, modificar los ya existentes y borrar aquellos que dejen de sernos útiles o que han sido introducidos por error.

Se incluiría una pestaña más a las ya existentes, siendo de libre acceso para los usuarios de la aplicación, ya que deberían poder acceder a ella de una forma fácil y rápida.

5.2.2. Administración de edificios

Con esta mejora se conseguiría una especie de asistente para añadir y configurar nuevos edificios sobre los que localizar a pacientes sin tener que generar nuevas líneas de código.

Se podría aplicar a diferentes situaciones tales como: el traslado temporal de los pacientes a otro edificio, su fácil utilización al no ser necesario escribir código, la posibilidad de controlar otros edificios si se amplía la residencia…

Esta zona podría ser de acceso restringido, pudiendo acceder a ella el desarrollador de la aplicación o algún responsable de la propia residencia.

En la zona de la derecha se añadiría una zona de acceso que tras una correcta identificación permitiera el acceso a un formulario donde se editaría el nuevo edificio: número de plantas, plano de cada planta y sus correspondientes coordenadas para superponerlo a la imagen, coordenadas para centrar el mapa…

5.2.3. Ampliar consultas de datos médicos

Finalmente esta es una mejora que aportaría una increíble funcionalidad a la aplicación, pero debido a su enorme amplitud podría ser objetivo de un futuro proyecto. Las posibilidades son muy numerosas, desde incluir simples datos numéricos como las medias del día, de la semana, de intervalos de tiempo que puedan editarse… hasta permitir la posibilidad de generar gráficas a petición del usuario, pasando por la posibilidad de exportar a un formato de impresión completos historiales de cada paciente.

5.2.4. Control de localización por áreas

Se trata de una idea de gran potencial cuya finalidad sería la de alertar a los responsables de la residencia de la posible localización indebida de ciertos pacientes.

Se definirían distintas áreas concéntricas, tantas como se desease: interior de la residencia, jardín, alrededores cercanos, alrededores alejados… Mediante esta nueva funcionalidad, se crearía una alerta en el momento que un determinado paciente sobrepasase los límites que tiene permitidos. Podría restringirse a todos los pacientes a una misma área o personalizar esta característica para cada paciente según su situación personal.

5.2.5. Actualización automática de la localización

Esta mejora buscaría actualizar la posición de los pacientes de forma automática sin la necesidad de la interacción del usuario. Los pacientes se desplazarían de una forma más o menos fluida (dependiendo del tiempo de actualización), de forma que veríamos como se mueven por el mapa.

Para esta mejora se necesitaría utilizar novedosas tecnologías como AJAX, que liberarían al servidor de recargar todos aquellos elementos que no fueran necesarios.

BIBLIOGRAFÍA

1. Fuentes bibliográficas:

- CEBALLOS, Francisco. Javier. "Microsoft C#. Lenguaje y aplicaciones". 2ª Ed. Madrid: Ra-Ma, 2007.

- COX, Ken. "ASP.NET 3.5 FOR DUMMIES". Indianápolis: Wiley Publishing, 2008.

- DAWSON, Christian W.; MARTÍN, Gregorio. "El proyecto fin de carrera en ingeniería informática: una guía para el estudiante". Madrid: Pearson Educación, 2002.

- EVJEN, Bill; HANSELMAN, Scott y RADER, Devin. "Professional ASP.NET 3.5 In C# and VB". Indianápolis: John Wiley & Sons, 2008.

- LARMAN, Craig. "UML Y PATRONES. Una introducción al análisis y diseño orientado a objetos y al proceso unificado". 2ª Ed. Madrid: Pearson Educación, 2003.

- MAYO, Joseph. "C# Al descubierto". Madrid: Pearson Educación, 2002.

- McCLURE, Wallace B.; CATE, Scott; GLAVICH, Paul y SHOEMAKER, Craig. "Ajax con ASP.NET". Anaya Multimedia, 2007.

2. Referencias Web:

- http://blog.evonet.com.au/post/Integrating-Google-Maps-into-an-ASPNET-page.aspx
  Blog en el que podemos encontrar un pequeño ejemplo de cómo integrar Google Maps en una página ASP.NET. (Enero-2009)

- http://googlemaps.subgurim.net/
  Completo manual con multitud de ejemplos reales para aprender a utilizar el API GoogleMpas.Subgurim.NET. (Mayo-2009)

- http://msmvps.com/blogs/cwalzer/archive/2007/10/30/antipracticasnetlecturaadonet2.aspx
  Comparativa en la recuperación de datos de una base de datos con ADO.NET mediante tres estrategias distintas: DataReader cargado en una lista genérica; DataSet y DataTable. (Abril-2009)

- http://support.microsoft.com/?scid=kb;es;305140
  Artículo que proporciona una guía básica al aprendizaje y dominio de ASP.NET con vínculos a información de utilidad, incluyendo documentación en línea, artículos de Microsoft Knowledge y notas del producto de Microsoft. (Diciembre-2009)

- http://www.asp.net/AJAX/AjaxControlToolkit/Samples/
  Página oficial de AJAX con múltiples ejemplos para aplicar la nueva librería AjaxControlToolkit. (Febrero-2009)

- http:// www.clikear.com/adonet_clase_conexion_17414.aspx
  Muestra un ejemplo para implementar una clase de conexión genérica para cualquier motor de bases de datos, utilizando .Net Providers. (Marzo-2009)

- http://www.desarrolloweb.com/articulos/superponer-imagen-mapa-google.html
  Manual para superponer imágenes en Google Maps mediante ASP.NET. (Enero-2009)

- http://www.elguille.info/NET/ADONET/novedadesVB9_DataSet_tipado.aspx
  Añadir un DataSet tipado a un proyecto de Visual Studio 2008. (Marzo-2009)

- http://www.es-asp.net/articulos-asp-net/
  Buscador de artículos muy útiles para principiantes y expertos en la programación de ASP.NET. (Mayo-2009)

- http://www.es-asp.net/tutoriales-asp-net/tutorial-0-61/tutorial-de-asp-net.aspx
  Tutorial para la creación de una aplicación Web en ASP.NET. (Diciembre-2009)

- http://www.es-asp.net/tutoriales-asp-net/tutorial-5312-5318/accordion.aspx
  Manual en español de ASP.NET Ajax Control Toolkit, nueva biblioteca de Ajax para Visual Studio 2008. (Febrero-2009)

- http://www.subgurim.net/Articulos/ajax-y-javascript/64/atlas-ajax-para-asp-net-el-timercontrol.aspx
  Tutorial que explica como utilizar uno de los recursos más útiles de ASP.NET Ajax, el Timer Control. (Julio-2009)

- http://www.subgurim.net/Articulos/csharp/50/listas-genericas-system-collections-generic-list.aspx
  Tutorial acerca de la utilización de las listas genéricas. (Marzo-2009)

Apéndice A

TECNOLOGÍAS UTILIZADAS

A.1. ASP.NET

Una vez nos hemos decidido por una aplicación Web, queda decidir qué tecnología vamos a utilizar de todas las existentes en el mercado. Opciones hay muchas y muy buenas siendo PHP, JSP, ASP y ASP.NET las más comunes. No se pretende enumerar todas y cada una de las características de todos estos lenguajes, simplemente voy a exponer una breve descripción de cada una de ellas y posteriormente trataré de justificar la elección escogida.

A.1.1. Barrida de alternativas

- **PHP:** es un lenguaje script, embebido en dentro de una página HTML, procesado en el lado del servidor. Se trata de la tecnología Web más extendida actualmente. Nació para trabajar en Linux con servidor Apache, pero hoy en día puede alojarse en casi cualquier tipo de servidor. El código fuente está abierto por lo que los errores están muy controlados y son inmediatamente solucionados, y además tiene una gran cantidad de módulos prefabricados que ya vienen instalados en los servidores, y que no hay más que aprender a utilizar.

  Su sintaxis es muy similar a C, quizás algo más simple, y destaca que fue creada para programar páginas Web, aunque también se puede programar en local. Se comunica con bases de datos sin necesidad de usar ODBC. Las últimas versiones ya trabajan con orientación a objetivos.

  Las principales características de PHP son su rapidez, su facilidad de aprendizaje, su soporte multiplataforma tanto de diversos Sistemas Operativos, como servidores http y de bases de datos y el hecho de que se distribuye de forma gratuita bajo una licencia abierta.

- **JSP:** se trata de una tecnología Java que permite generar contenido dinámico para Web. Las Java Server Pages son páginas HTML corrientes con porciones de código Java incrustadas junto con una serie de etiquetas. De esta forma, cuando una página es solicitada por un usuario y procesada por un servidor HTTP, el código HTML pasará directamente al usuario, mientras que las porciones de código Java será ejecutadas en el servidor cuando la solicitud haya sido recibida, para genera el contenido dinámico de la página. Cuando el usuario acceda al código de la página que le llega sólo verá HTML, sin poder acceder al código JSP subyacente.

  Al estar basado en Java, presenta las ventajas que este lenguaje ofrece con respecto a la portabilidad entre plataformas y las derivadas de la orientación a objetos de este lenguaje.

  Las peticiones de páginas JSP son normalmente implementadas mediante *servlets*, de forma que el contendor *servlet*, maneja múltiples solicitudes a la vez, requiriendo menos recursos. Esto hace que JSP sea mucho más eficiente que otros modelos como los programas CGI.

  Para muy grandes aplicaciones suele elegirse JSP en lugar de PHP, dado que PHP es un lenguaje de más bajo nivel que JSP y dificulta la modularización y organización por capas de la aplicación.

- **ASP:** las Actives Server Pages son una tecnología del lado de servidor para páginas generadas dinámicamente desarrollada por Microsoft y optimizada para su ejecución en servidores Windows con tecnología NT bajo IIS, aunque también hay opciones para Windows 98 con el Personal Web Server y para otras plataformas gracias a tecnologías como InstantASP y ChiliASP. Se trata de un lenguaje orientado a intercalarlo entre código HTML, muy al estilo de PHP.

  Al ser una tecnología propietaria, no tiene la enorme cantidad de módulos extra que sí tiene PHP, aunque abriendo objetos COM trabaja fácilmente con archivos DLL.

  El hecho de que habitualmente los servidores de hospedaje en Internet para las Active Server Pages sean más caros, han hecho que ASP deje de ser la tecnología para páginas Web dinámicas orientadas a servidor más utilizada, tal y como lo llegó a ser en sus inicios. Esto es debido a que la tecnología ASP está estrechamente relacionada con el modelo tecnológico de su fabricante ya que una página ASP es procesada por un servidor Microsoft Internet Information Server (del lado del servidor) y luego el resultado es mostrado al usuario en su navegador Web (del lado del cliente).

  La mayoría de las páginas ASP son escritas en Visual Basic Script, lo que para los desarrolladores Visual Basic era una ventaja pues no tenían que aprender otro lenguaje, para otros es una desventaja, pues consideran que no es más que un parche Web de un lenguaje ya existente, y que no fue creado expresamente para los servidores Web, como sí pasa con PHP. También soporta el lenguaje JScript (Javascript de Microsoft). Se comunica con las bases de datos vía ODBC, y la comunicación con SQL Server es óptima.

- **ASP.NET:** rompe totalmente con el pensamiento de script que se tenía hasta el momento. El cambio en la arquitectura es radical. De hecho, lo único que mantiene de ASP es el nombre, el propietario y la evolución de Visual Basic a Visual Basic .NET (VB.NET)… el resto es todo nuevo.

  Debido a que se trata de la tecnología escogida para la implementación de la aplicación Web, dedico el siguiente subapartado a su descripción.

A.1.2. Opción escogida: ASP.NET

ASP.NET es un Framework para aplicaciones Web desarrollado y comercializado por Microsoft. Es usado por programadores para construir sitios Web dinámicos, aplicaciones Web y servicios Web XML. Apareció en enero de 2002 con la versión 1.0 del .NET Framework y es la tecnología sucesora de las Active Server Pages (ASP).

ASP.NET intenta que la metodología del desarrollo Web sea similar a la del desarrollo mediante interfaz gráfica de usuario (GUI). Un control de servidor en ASP.NET funciona de un modo similar a los controles GUI en otros entornos. Los botones, cuadros de texto, etiquetas y rejillas de datos tienen propiedades que pueden ser modificadas y ofrecen eventos que pueden ser procesados. Los controles de servidor de ASP.NET saben como mostrar su contenido en una página HTML, de igual modo que los controles de usuario basados en GUI saben cómo deben ser mostrados en su entorno GUI. Una ventaja añadida de ASP.NET es que

las propiedades y los métodos de los controles de servidor Web son similares, y en algunos casos idénticos, a los controles que son comparables en entorno Windows GUI/Winforms.

A continuación expondré las principales características de esta nueva y revolucionaria tecnología para el desarrollo Web y que han hecho decantarme por su elección.

Las páginas ASP.NET son ficheros de texto con la extensión .aspx. Consisten en código y marcas y son compiladas y ejecutadas dinámicamente en el servidor para producir una traducción para el navegador (o dispositivo) cliente. Se pueden desplegar a través de un árbol de directorios raíz de IIS. Cuando un navegador hace una petición de un recurso .aspx, la rutina ASP.NET analiza y compila el fichero a una clase del Framework .NET. El fichero .aspx se compila únicamente la primera vez que es accedido; la instancia compilada se reutiliza en las sucesivas peticiones.

Además de código y marcas, las páginas ASP.NET pueden contener **controles de servidor**, que son objetos programables del lado del servidor que típicamente representan un elemento UI en la página, como un *textbox* o una imagen. Los controles de servidor participan en la ejecución de la página y producen sus propias etiquetas para el navegador cliente. La principal ventaja de los controles de servidor es que permiten a los desarrolladores obtener un comportamiento y un renderizado complejo a partir de componentes sencillos, reduciendo así dramáticamente la cantidad de código necesario para crear una página Web dinámica. Otra ventaja de los controles de servidor es su facilidad para personalizar su comportamiento o renderizado. Los controles de servidor muestran propiedades que pueden ajustarse bien de forma declarativa (en la etiqueta) o bien de forma programada (con código). Los controles de servidor (y la página en si) también tienen eventos que los desarrolladores pueden controlar para realizar acciones específicas durante la ejecución de la página, o en respuesta a una acción del lado del cliente que envíe la página al servidor. Los controles de servidor también simplifican el problema de mantener el estado durante diferentes idas y venidas del servidor, manteniendo sus valores de forma automática en sucesivos *postbacks*.

Los controles de servidor se declaran en un fichero .aspx mediante etiquetas propias o etiquetas HTML intrínsecas que contienen el atributo ***runat="server"***. Las etiquetas HTML intrínsecas son manejadas con uno de los controles del *namespace System.Web.UI.HtmlControls.* A cualquier etiqueta que no corresponda a uno de los controles se le asigna el tipo *System.Web.UI.HtmlControls.HtmlGenericControl***.**

ASP.NET introduce el concepto del *code-behind*, por el que una misma página se compone de dos ficheros: el de la interfaz de usuario y el de código. La página del código hace referencia al fichero de *code-behind* en el atributo *CodeFile* de la directiva ***<%@ Page %>***, especificando en nombre de la clase en el atributo ***Inherits***. Con ello se facilita la programación de aplicaciones en múltiples capas, lo que en definitiva se traduce en la total separación entre lo que el usuario ve y lo que la base de datos tiene almacenado. Por tanto, cualquier cambio drástico de especificaciones minimiza los cambios en la aplicación y maximiza la facilidad de mantenimiento.

Asimismo, ASP.NET nos sirve tanto para Webs sencillas como para grandes aplicaciones. No debemos olvidar que la orientación a objetos y la naturaleza compilada permiten que hagamos uso de herramientas de creación de Webs, las más importantes de la familia del Visual Studio, que nos facilitan mucho la tarea de programación. Estas herramientas permiten hacer Webs sencillas y de bajas prestaciones en un tiempo record, así como llevar el mantenimiento de grandes aplicaciones de forma más sencilla.

Se puede almacenar en la caché del servidor tanto páginas enteras, como controles personalizados o simples variables. En páginas críticas con mucha carga de base de datos nos es muy útil almacenar datos de la base de datos en la caché, reduciendo enormemente el consumo de recursos.

Mediante esta tecnología disponemos de carpetas especializadas, como por ejemplo *App_Code* que compila automáticamente las clases que se alojan en él, o la carpeta A*pp_Theme* que alojan ficheros que marcan los temas de estilos de la Web. Por defecto, el directorio *App_Code* sólo puede contener ficheros del mismo lenguaje. Sin embargo, podemos particionar dicho directorio en subdirectorios (cada uno conteniendo ficheros de un lenguaje) para contener múltiples lenguajes (*C#, VB.NET, J#,…*) bajo el directorio *App_Code*.

El directorio Bin es como el directorio *Code*, a excepción que puede contener ficheros precompilados (archivos del tipo: *.DLL). Esto es útil cuando necesitamos utilizar código que posiblemente haya sido escrito por otra persona, y no tenemos acceso al código fuente (un fichero VB o C#) pero si que tenemos una DLL compilada. Simplemente podemos colocar dicho fichero en el directorio Bin para que sea accesible desde nuestro site. Por defecto, todo lo que hay en el directorio Bin se carga automáticamente en la aplicación y se hace accesible a las páginas.

Los archivos de configuración Web.config y Machine.config permiten realizar operación de configuración en ficheros que hasta ahora había que realizar en el servidor.

Por todas estas ventajas y adelantos, por ser una tecnología en auge y como reto personal, ASP.NET ha sido la tecnología empleada para el desarrollo de este proyecto.

A.2. AJAX

Ajax es un conjunto de tecnologías de cliente que sirve para la comunicación asíncrona entre la interfaz de usuario y el servidor Web. Ajax son las siglas de *Asynchronous JavaScript And XML* (JavaScript asíncrono y XML).

A.2.1. Historia de Ajax

A pesar de que el término Ajax fuese creado en 2005, la historia de las tecnologías que permiten Ajax se remonta a una década antes con la iniciativa de Microsoft en el desarrollo de Scripting Remoto. Sin embargo, las técnicas para la carga asíncrona de contenidos en una página existente sin requerir recarga completa remontan al tiempo del elemento *iframe* (introducido en Internet Explorer 3 en 1996) y el tipo de elemento *layer* (introducido en Netscape 4 en 1997, abandonado durante las primeras etapas de desarrollo de Mozilla). Ambos tipos de elemento tenían el atributo *src* que podía tomar cualquier dirección URL externa, y cargando una página que contenga JavaScript que manipule la página paterna, pueden lograrse efectos parecidos al AJAX.

El Microsoft's Remote Scripting (o MSRS, introducido en 1998) resultó un sustituto más elegante para estas técnicas, con envío de datos a través de un applet Java el cual se puede comunicar con el cliente usando JavaScript.

Desde que XMLHttpRequest está implementado en la mayoría de los navegadores, raramente se usan técnicas alternativas. Sin embargo, todavía se utilizan donde se requiere una mayor compatibilidad, una reducida implementación, o acceso cruzado entre sitios Web.

Varias versiones de Ajax fueron viendo la luz: *Comfort ASP.NET*, *MagicAjax.NET*, *Anthem.NET*, *Atlas…* La versión utilizada para este proyecto corresponde a *ASP.NET Ajax*, incluido con la versión 3.5 del .NET Framework. Realmente se trata de una evolución de *Atlas*, que desencadenó en tres productos: *Microsfot Ajax Library*, que contiene las librerías de JavaScript; *ASP.NET Ajax Extensions*, que contiene el código .NET del lado del servidor *Ajax Control Toolkit*, que incluye controles de código compartido que pueden ser utilizados con *ASP.NET Ajax*.

A.2.2. Tecnologías que forman Ajax

A continuación enumero los elementos específicos que forman Ajax:

- **XmlHttpRequest:** permite al navegador comunicarse con un servidor remoto. Este objeto permite al navegador hablar con el servidor sin la necesidad de hacer envíos sucesivos de la página Web completa. En Internet Explorer, esta funcionalidad es posible gracias al componente *ActiveX MSXML*. En Mozilla Firefox y otros navegadores, esta funcionalidad la proporciona un objeto denominado literalmente *XmlHttppRequest*. El objeto *XmlHttppRequest* fue diseñado después del componente *MSXML*. Las bibliotecas JavaScript del lado del cliente ocultan las diferencias entre los distintos entornos de los navegadores. A veces estas comunicaciones se llevan a cabo mediante un marco o un marco incorporado (*iframe*) oculto.

- **JavaScript:** proporciona la capacidad de comunicarse con un servidor remoto. La versión de JavaScript debe ser 1.5 o posterior. Aunque específicamente JavaScript no es obligatorio, es necesario desde el punto de vista de que JavaScript es el único entorno de programación del lado del cliente admitido por la mayoría de los navegadores Web modernos. Existen otros lenguajes de programación del lado cliente, pero sin embargo, no son admitidos por todos los navegadores.

- **Soporte DHTML/DOM:** el navegador debe poseer la capacidad de actualizar dinámicamente los elementos de formulario, y la capacidad de hacerlo de manera estandarizada viene dad debido a que trabaja con DOM (*Document Object Model*, modelo de objetos de documento). Al trabajar con DOM, a los desarrolladores les resulta más fácil escribir un único código que sea admitido por los distintos navegadores.

- **Transporte de datos con XML o JSON:** el uso de XML tiene en cuenta la capacidad de comunicarse con el servidor Web mediante un mecanismo estándar. Existen situaciones en las que JSON (*JavaScript Object Notation*, notación de objetos de JavaScript) se utiliza como notación para la comunicación en lugar del XML normal.

A.2.3. Las ventajas de Ajax

Las ventajas de Ajax sobre las aplicaciones Web clásicas son las siguientes:

- **Asíncrono:** Ajax tiene en cuenta la capacidad de realizar llamadas asíncronas al servidor Web. Esto permite al navegador evitar tener que esperar a recibir todos los datos antes de permitir al usuario una nueva acción.

- **Mínima transferencia de datos:** al no realizar envíos sucesivos completos y enviar todos los datos de formulario al servidor, se minimiza el uso de la red y las operaciones tienen lugar más rápidamente. En sitios y servidores con ancho de banda restringido para la transferencia de datos, esto puede mejorar sustancialmente el rendimiento de la red.

- **Procesamiento limitado en el servidor:** en base al hecho de que sólo se envían al servidor los datos necesarios, el servidor no necesita procesar todos los elementos de formulario. Al enviar sólo los datos necesarios, se limita el procesamiento en el servidor. No hay necesidad de procesar todos los elementos de formulario, devolver imágenes al cliente, ni se necesita devolver una página completa al cliente.

- **Reacción:** como las aplicaciones Ajax son asíncronas en el cliente, se deduce que deben ofrecer muy buena respuesta.

- **Contexto:** con los envíos sucesivos completos, los usuarios pueden perder el contexto en el que se encuentran. El usuario puede estar al final de la página, hacer clic sobre el botón “Enviar”, y encontrarse de repente en la parte superior de la página. Con Ajax no hay envíos sucesivos. El hacer clic cobre el botón “Enviar” en una aplicación que utiliza Ajax, el usuario mantiene su ubicación. Se mantiene el estado del usuario, y éste ya no necesita desplazarse por la página hasta el punto en el que se encontraba antes de hacer clic sobre el botón “Enviar”.

A modo de conclusión podemos decir que gracias a Ajax, entre cliente y servidor sólo se transmiten los datos necesarios. Esto minimiza el uso de la red y el proceso en el lado cliente. Se busca mejorar la experiencia del usuario de forma que su iteración con la aplicación Web se asemeje a la iteración que realizaría con una aplicación de escritorio.

Apéndice B

APIS UTILIZADAS

B.1. GoogleMaps.Subgurim.NET

GoogleMaps.Subgurim.NET es el control de GoogleMaps más avanzado para ASP.NET. Ofrece todo el poder y funcionalidad del API oficial de GoogleMaps pero sin necesidad de código JavaScript, todo adaptado a la tecnología .NET.

Con este apéndice voy a tratar de guiar al lector en los primeros pasos para la utilización de esta API. Empezaré explicando como hacernos con el archivo DLL que contiene el API, a continuación indicaré como registrar dicho archivo en nuestra aplicación y finalmente, mediante sucesivos ejemplos, mostraré diferentes funcionalidades de esta potente API. Para una mayor profundización he añadido en el CD adjunto un archivo PDF donde pueden consultarse instrucciones y una variedad mayor de ejemplos.

Por último dedico un subapartado a explicar como superponer imágenes en nuestros mapas mediante la utilización de archivos KML, estándar utilizado por Google para construir sus propios mapas.

B.1.1. Cómo empezar

Para empezar a utilizar esta herramienta en primer lugar hay que descargarse el archivo gratuito de código compilado *GMaps.dll* disponible en el CD adjunto.

Una vez obtenido dicho archivo deberemos agregarlo a las referencias que utilizamos en nuestro proyecto de Visual Studio. Para ello, pinchamos con el botón derecho sobre la carpeta *References* y escogemos la opción *Agregar referencia…* En la ventana emergente seleccionamos la pestaña *Examinar* y desde ahí buscamos la ubicación donde hemos situado el archivo descargado. Mediante esta acción ponemos a disposición de Visual Studio toda la funcionalidad del API GoogleMaps.Subgurim.NET que encierra el ensamblado.

Tras haber agregado la referencia a nuestro proyecto ya estamos listos para empezar con nuestro primer ejemplo. En primer lugar, tras el encabezado del formulario .aspx, añadimos la siguiente línea de código para registrar el ensamblado en la aplicación:

```
<%@ Register Assembly="GMaps" Namespace="Subgurim.Controles" TagPrefix="cc1" %>
```

Figura B.1.1.

donde:

-**GMaps:** es el nombre del archivo de ensamblado sin la extensión (.DLL).

-**Subgurim.Controles:** es un espacio de nombres que se encuentra entre las clases del ensamblado.

-**cc1:** es un alias que se asocia al espacio de nombres.

Una vez registrado el ensamblado ya podemos colocar nuestro primer mapa en el formulario .aspx. Pero antes necesitamos conseguir una Key (clave), pues lo exige Google para poder utilizar su API. Esta clave puede encontrarse fácilmente en Internet, pero depende del dominio donde se utilice, por lo que deberá ser única para cada Web. Existen varias formas para dar de alta la clave: en el fichero *web.config*; en el archivo de código correspondiente al formulario o como muestro en el ejemplo, en el propio control. Mediante la siguiente instrucción podremos construir nuestro primer mapa:

```
<cc1:GMap ID="GMap1" runat="server" Key="keyobtenida" />
```

Figura B.1.2.

Con esta sencilla instucción, añadido a lo anteriormente explicado, ya tendremos un mapa en nuestra aplicación Web. Se trata de un simple mapa, sin funcionalidad alguna pero en el archivo de código asociado al formulario Web podremos definir y modificar todas las propiedades que deseemos del mapa. Para ello en primer lugar deberemos incluir la referencia a la biblioteca que vamos a utilizar mediante el correspondiente *using*, en este caso, *using Subgurim.Controles.*

B.1.2. Ejemplos prácticos

En este subapartado, mediante ejemplos reales que pueden probarse, voy a mostrar las funciones básicas que ofrece esta API. La intención de este subapartado no es la de exponer de forma exhaustiva toda la funcionalidad disponible, sino servir como tutorial mediante ejemplos para el programador que se inicia en esta materia, guiándolo en sus primeros pasos.

- **Básico:** en nuestro primer ejemplo, vemos que es increíblemente sencillo tener el mapa de Google en nuestra aplicación. Podemos decidir si el usuario puede mover el mapa con el ratón (***Dragging***); el idioma del mapa (***Language***); sus dimensiones (***Height***, ***Width***); el centro (***setCenter***) y el zoom del mapa (***GZoom***); el tipo de vista que se visualizará (***mapType***); permitir que el usuario pueda navegar por el mapa con las flechas del teclado (***enableGKeyboardHandler***).

Basico.aspx

```
<cc1:GMap ID="GMap1" runat="server" />
```

Basico.aspx.cs

```
GMap1.enableDragging = false;
GMap1.Language = "es";
GMap1.Height = 300;
GMap1.Width = 456;
GMap1.setCenter(new GLatLng(41, 2));
GMap1.GZoom = 6;
GMap1.mapType = GMapType.GTypes.Satellite;
GMap1.enableGKeyboardHandler = true;
```

Figura B.1.3.

- **Controles y overlays:** los controles prefabricados son aquellos que Google provee por defecto, y con ASP.NET es muy sencillo añadirlos a nuestro mapa. Para ello se utiliza **GControl**. Podemos añadir un pequeño mapa (que se puede minimizar) que permite navegar a grandes rasgos por los alrededores de nuestro mapa actual (***GControl.preBuilt.GOverviewMapControl***); también podemos añadir un menú desplegable para elegir entre un tipo de mapa y otro (***GControl.preBuilt.MenuMapTypeControl***) o cuatro flechas de dirección para navegar por el mapa, así como botones para aumentar y reducir el zoom (***GControl.preBuilt.SmallMapControl***), teniendo la posibilidad de escoger en que posición del mapa colocar cada control mediante la propiedad **position** del tipo **GControlPosition**.

Controles.aspx

```
<cc1:GMap ID="GMap1" runat="server" />
```

Controles.aspx.cs

```
GControl control = new GControl(GControl.preBuilt.GOverviewMapControl);
GControl control2 = new GControl(GControl.preBuilt.MenuMapTypeControl, new
GControlPosition(GControlPosition.position.Top_Left));

GMap1.addControl(control);
GMap1.addControl(control2);
GMap1.addControl(new GControl(GControl.preBuilt.SmallMapControl, new
GControlPosition(GControlPosition.position.Top_Right)));
```

Figura B.1.4.

- **Superposición de imágenes:** con el **GGroundOverlay** se pueden insertar imágenes a los mapas de forma muy sencilla basta con definir la imagen que deseemos superponer mediante ***imageUrl*** y su posición en el mapa mediante ***GLatLngBounds***.

Superposicion.aspx

```
<cc1:GMap ID="GMap1" runat="server" />
```

Superposicion.aspx.cs

```
GLatLng sw = new GLatLng(64,20);
GLatLng ne = new GLatLng(65,29);

GMap1.setCenter((sw / 2) + (ne / 2));

GGroundOverlay groundOverlay = new
GGroundOverlay("http://googlemaps.subgurim.net/images/logo.jpg", new
GLatLngBounds(sw, ne));

GMap1.addGroundOverlay(groundOverlay);
```

Figura B.1.5.

- **InfoWindows:** añadir un InfoWindow es tremendamente fácil. No hay más que instanciar a GInfoWindow y definir las coordenadas donde se ubicará (**point**) y el texto, con HTML incluido, que mostrará (**html**).

Infowindows.aspx

```
<cc1:GMap ID="GMap1" runat="server" />
```

Infowindows.aspx.cs

```
GLatLng latlon = new GLatLng(10.2, 22);
GMap1.setCenter(latlon, 4);

GInfoWindow window = new GInfoWindow(latlon, "Ejemplo de <b> infoWindow </b>");
GMap1.addInfoWindow(window);
```

Figura B.1.6.

- **Iconos:** una gran utilidad es la de colocar iconos en el mapa. Para ello utilizamos los **GMarker**. Para instanciar un icono deberemos definir las coordenadas donde deseemos colocarlo (**point**) y sus características (**options**) del tipo **GMarkerOptions** que nos permite definir propiedades del tipo: permitir clickear en un icono o no (**clickable**); definir si el icono puede ser arrastrado o no (**draggable**); la imagen que representará al icono (**icon**)…

Iconos.aspx

```
<cc1:GMap ID="GMap1" runat="server" />
```

Iconos.aspx.cs

```
GLatLng latlng = new GLatLng(42.12, -1.145);
GMap1.setCenter(latlng, 5, GMapType.GTypes.Hybrid);

GMarkerOptions markerOptions = new GMarkerOptions();
markerOptions.clickable = false;
markerOptions.draggable = true;
GIcon icono = new GIcon();
icono.image = "http://labs.google.com/ridefinder/images/mm_20_red.png";
icono.iconSize = new GSize(12,20);
markerOptions.icon=icono;
GMarker marker = new GMarker(latlng + new GLatLng(-1, 2.5), markerOptions);
GMap1.addGMarker(marker);
```

Figura B.1.7.

Como he dicho al principio del subapartado, no buscaba ofrecer una profunda explicación de la API descrita, sino pequeños ejemplos en los que apoyarse para familiarizarse con la herramienta. Para una mayor profundización se puede consultar el archivo PDF que existe en el CD adjunto.

B.1.3. Archivos KML

En este último subapartado voy a explicar la solución que encontré para la superposición de imágenes en Google Maps: los archivos KML

KML, o *Keyhole Markup Language* (lenguaje de marcas de Keyhole), es una gramática XML y un formato de archivo para la creación de modelos y el almacenamiento de funciones geográficas como puntos, líneas, imágenes, polígonos y modelos que se mostrarán en Google Earth, Google Maps y otras aplicaciones. KML puede ser utilizado para utilizar compartir lugares e información con otros usuarios de estas aplicaciones.

Google Earth procesa los archivos KML de una manera similar a como los navegadores Web procesan los archivos HTML y XML. Al igual que los archivos HTML, los KML cuentan con una estructura basada en etiquetas con nombres y atributos utilizados para poder visualizarlos. Por lo tanto, Google Earth actúa como un navegador de archivos KML, sin embargo, Google Maps sólo puede mostrar algunas características de KML, es por ello que para configurar la rotación de la imagen nos deberemos servir de un editor de imágenes convencional, como puede ser Adobe PhotoShop Elements.

El archivo KML, al igual que las imágenes que deseemos superponer, deben estar alojados en un sitio Web para que pueda ser accedido por Google Maps.

A continuación muestro un fichero KML de ejemplo, a partir del cual explicaré sus características principales:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2"
xmlns:gx="http://www.google.com/kml/ext/2.2"
xmlns:kml="http://www.opengis.net/kml/2.2"
xmlns:atom="http://www.w3.org/2005/Atom">
 <GroundOverlay>
    <name>planta3</name>
    <Icon>
        <href>http://jair.lab.fi.uva.es/~marsant/kml/planta3.png</href>
    </Icon>
    <LatLonBox>
        <north>41.66322055803502</north>
        <south>41.66203058944889</south>
        <east>-4.704443392678794</east>
        <west>-4.706185845650913</west>
    </LatLonBox>
  </GroundOverlay>
</kml>
```

Figura B.1.8.

donde:

-**href:** es la ruta donde se encuentra la imagen a superponer.

-**north:** especifica la latitud del borde norte del cuadrado delimitador en grados decimales desde 0 hasta ±90.

-**south:** especifica la latitud del borde sur del cuadrado delimitador en grados decimales desde 0 hasta ±90.

-**east:** especifica la longitud del borde este del cuadro delimitador en grados decimales desde 0 hasta ±180.

-**west:** especifica la longitud del borde oeste del cuadro delimitador en grados decimales desde 0 hasta ±180.

Para la superposición de los planos podría haber utilizado el control **GGroundOverlay** pero la existencia de los archivos KML y su consideración de estándar por parte de Google a la hora de realizar sus mapas me animó a utilizar estos archivos en lugar del control descrito.

B.2. ASP.NET Chart Control

ASP.NET Chart Control se trata de un componente de Microsoft con una versión específica para ASP.NET, válida para WebForms y MVC framework, y otra para Windows Forms, que permite generar gráficas estadísticas prácticamente de cualquier tipo, visualmente muy atractivas, realmente fáciles de utilizar en nuestras aplicaciones y, además, de forma gratuita.

Entre sus múltiples características cabe resaltar los 25 tipos diferentes de gráficas, muchas de ellas con vistas en tres dimensiones, en las que se puede modificar prácticamente todo: rotación, inclinación, sombras… Control total sobre los ejes en cuanto a escalado, visualización o etiquetado; soporta mapeo de imágenes, posibilidad de capturar clicks sobre áreas para establecer comportamientos personalizados, o combinarlo con Ajax, como he hecho en este proyecto para la actualización automática de las gráficas, para enriquecer la experiencia del usuario.

En los siguientes subapartados indicaré los archivos necesarios para la utilización de esta API, indicaré cuáles son los primeros pasos que se deben dar y finalmente mostraré una serie de ejemplos prácticos que pueden probarse con solo copiar y pegar.

B.2.1. Instalación

Antes de instalar, asegúrese que cumple el requisito previo básico, tener instalado Microsoft .NET Framework 3.5 SP1. Si no lo ha hecho antes, este es el primer paso que debe dar.

Una vez asegurado este punto, el siguiente paso es descargar Microsoft Chart Control, que incluye controles tanto para ASP.NET como para Windows Forms.

Existe también, como descarga opcional, el paquete de idioma para Microsoft Chart Control, que contiene la localización del producto para otros idiomas.

Después, es una buena idea instalar el Add-on para Visual Studio 2008 que os facilitará el trabajo con el control desde este entorno de desarrollo, a base de diseñadores integrados.

Y, por último, para tomar conciencia del tipo de resultados que se pueden obtener con este control, recomiendo descargar los proyectos de demostración, que os permitirán ver y tocar una auténtica batería de ejemplos muy útiles a la hora de usarlo en vuestros desarrollos, tanto ASP.NET como Winforms. Todos estos ficheros pueden obtenerse fácilmente de la Web, pero para evitar búsquedas problemáticas, están incluidos en el CD adjunto.

B.1.1. Cómo empezar

Tras instalar los archivos citados en el subapartado anterior estamos listos para empezar a trabajar. Para utilizar ASP.NET Chart Control debe crear una Aplicación Web. Una vez creada puede añadir una gráfica a un formulario Web arrastrando un Chart control desde el cuadro de herramientas.

Figura B.2.1.

Para configurar cualquier propiedad de la misma puede hacer click con el botón derecho sobre la gráfica en la vista de diseño con el botón derecho y escoger la opción *Propiedades*. Si una de las propiedades es una colección verá a la derecha de la celda un botón con puntos suspensivos. Puede pulsar sobre el botón para editar sus opciones en la ventana de edición de la colección.

Con esto y agregando un par de líneas en el archivo de código asociado, donde determinaremos el tipo de gráfica y los valores a representar, ya tenemos nuestra primera gráfica.

B.2.2. Ejemplos prácticos

En este subapartado, mediante ejemplos reales que pueden probarse, voy a mostrar las funciones básicas que ofrece esta API. La intención de este subapartado no es la de exponer de forma exhaustiva toda la funcionalidad disponible, sino servir como ayuda para el programador que se inicia en esta materia, guiándolo en sus primeros pasos.

En primer lugar mostraré los elementos que componen una gráfica:

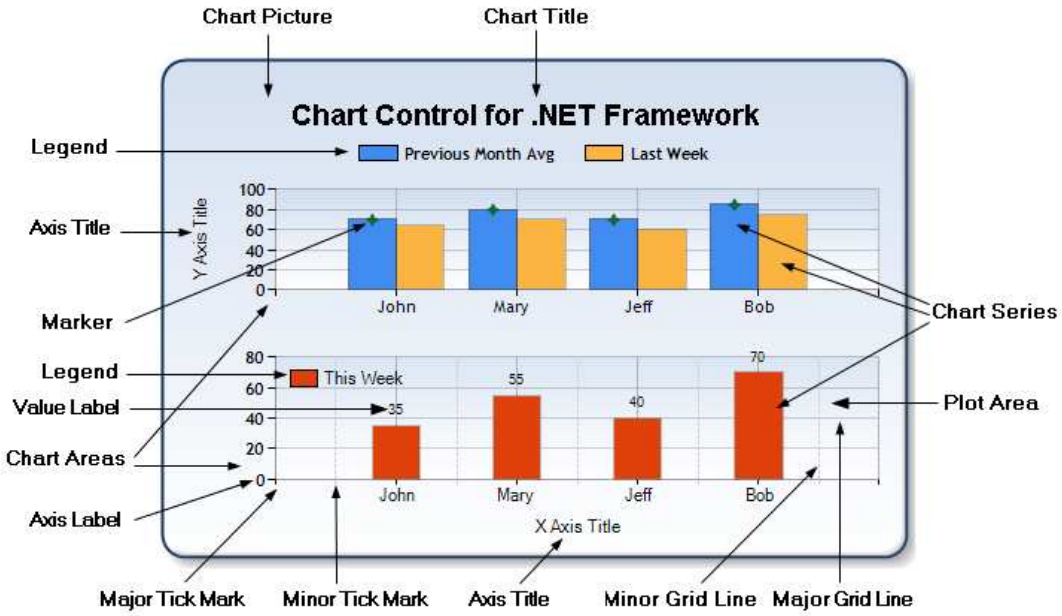

Figura B.2.2.

Cabe destacar la enorme versatilidad y potencia de esta herramienta que permite acceder a cada uno de los elementos que componen la gráfica, modificarlos a nuestro gusto, escoger su posición, su inclusión, utilizar diferentes tipos de gráficas e incluso entremezclarlos para conseguir composiciones.

Para la edición de las gráficas podemos optar por varias alternativas. Si éstas van a ser gráficas estáticas y no necesitan modificarse en tiempo de ejecución, podremos editar todas sus características en el HTML en tiempo de diseño. Si por el contrario, necesitamos gráficas dinámicas, nos serviremos de la tecnología *code-behind* que nos ofrece ASP.NET para definirlas. También está disponible la alternativa de mezclar ambas estrategias. Escoger una u otra dependerá de nuestras necesidades.

A continuación muestro una lista de ejemplos para ilustrar el manejo y la utilidad de esta herramienta:

- **Básico:** en el siguiente ejemplo muestro como crear una sencilla gráfica utilizando la tecnología “code-behind”. Se recomienda la utilización del “code-behind” en todos los desarrollos para separar la interfaz de la lógica de negocio. En primer lugar definimos el control en el HTML y en segundo lugar se modifican las propiedades que deseemos en el archivo de código asociado.

Basico.aspx

```
<asp:Chart ID="Chart1" runat="server">
    <series>
        <asp:Series Name="Series1">
        </asp:Series>
    </series>
    <chartareas>
        <asp:ChartArea Name="ChartArea1">
        </asp:ChartArea>
    </chartareas>
</asp:Chart>
```

Basico.aspx.cs

```
…
protected void Page_Load(object sender, EventArgs e)
{
      //Especifico el tipo de grafica
      Chart1.Series["Series1"].ChartType = SeriesChartType.Spline;

      //Añado los datos
      Chart1.Series["Series1"].Points.AddXY(1, 0.5);
      Chart1.Series["Series1"].Points.AddXY(2, 1.5);
      Chart1.Series["Series1"].Points.AddXY(3, 1);
      Chart1.Series["Series1"].Points.AddXY(4, 2);

      //Elimino los margenes
      Chart1.ChartAreas["ChartArea1"].AxisX.IsMarginVisible = false;
}
…
```

Figura B.2.3.

- **Varias gráficas:** en el siguiente ejemplo aparecen un par de gráficas en la misma imagen. Para esta ocasión he definido una mediante HTML, para mostrar un cómo se haría, y la otra gráfica está definida en el archivo de código asociado.

Varias.aspx

```
<asp:Chart ID="Chart1" runat="server">
    <Legends>
        <asp:Legend IsTextAutoFit="False" Name="Default"> </asp:Legend>
    </Legends>
     <BorderSkin SkinStyle="Emboss"></BorderSkin>
     <Series>
        <asp:Series Name="Primera" BorderColor="180, 26, 59, 105">
          <Points>
             <asp:DataPoint YValues="45" />
             <asp:DataPoint YValues="34" />
             <asp:DataPoint YValues="32" />
          </Points>
        </asp:Series>
    </Series>
    <ChartAreas>
        <asp:ChartArea Name="ChartArea1" BorderColor="64, 64, 64, 64"
BackColor="64, 165, 191, 228" ShadowColor="Transparent">
           <Area3DStyle Rotation="10" Perspective="10" Inclination="15"
IsRightAngleAxes="False"
                  WallWidth="0" IsClustered="False"></Area3DStyle>
          <AxisY LineColor="64, 64, 64, 64">
             <LabelStyle Font="Trebuchet MS, 8.25pt, style=Bold" />
             <MajorGrid LineColor="64, 64, 64, 64" />
          </AxisY>
          <AxisX LineColor="64, 64, 64, 64">
             <LabelStyle Font="Trebuchet MS, 8.25pt, style=Bold" />
             <MajorGrid LineColor="64, 64, 64, 64" />
          </AxisX>
        </asp:ChartArea>
    </ChartAreas>
</asp:Chart>
```

Varias.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
      //Creo, especifico y pueblo el tipo de grafica
      Series grafica2 = new Series("Segunda");
      grafica2.ChartType = SeriesChartType.Spline;
      grafica2.Points.AddY(37);
      grafica2.Points.AddY(57);
      grafica2.Points.AddY(20);
      //Incluyo la gráfica en el dibujo
      Chart1.Series.Add(grafica2);
}
```

Figura B.2.4.

- **Enlazar datos:** los datos representados en las gráficas pueden ser obtenidos directamente de una base de datos. Para ello deberemos abrir una conexión con una base de datos, crear un DataReader (dependiendo de nuestro proveedor de datos será: SqlDataReader, OleDbReader…) y utilizar el método *DataBindXY* que proporciona la API.

Enlazar.aspx

```
<asp:Chart ID="Chart1" runat="server">
    <series>
        <asp:Series Name="Series1">
        </asp:Series>
    </series>
    <chartareas>
        <asp:ChartArea Name="ChartArea1">
        </asp:ChartArea>
    </chartareas>
</asp:Chart>
```

Enlazar.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    Chart1.Series["Series1"].ChartType = SeriesChartType.Spline;
    //Creo la consulta
    string cadena = "SELECT x,y FROM datos;";
    //Defino la cadena de conexion
    SqlConnection conn = new SqlConnection("Data Source=USUARIO-
BDFE2C0; Initial Catalog=aivi; Integrated Security=True;");
     //Abro la conexion
    conn.Open();
    //Creo una instancia de DbCommand específica del proveedor actual
    SqlCommand cmd = new SqlCommand(cadena, conn);
    SqlDataReader reader =
cmd.ExecuteReader(CommandBehavior.CloseConnection);
    Chart1.Series["Series1"].Points.DataBindXY(reader, "x", reader, "y");
    reader.Close();
    //cierro la conexion
    conn.Close();
    Chart1.ChartAreas["ChartArea1"].AxisX.IsMarginVisible = false;
 }
```

Figura B.2.5.

La intención de este subapartado no era la de ofrecer una profunda explicación de la API, sino pequeños ejemplos en los que apoyarse para familiarizarse con la herramienta. Para una mayor profundización se pueden consultar los ejemplos en forma de aplicación Web que incluyo en el CD adjunto.