

UNIVERSITAT OBERTA DE CATALUNYA

TRABAJO FINAL DE CARRERA – DESARROLLO EN .NET  
Ingeniería Técnica en Informática de Sistemas  
Consultor: Juan Carlos González Martín

# PGPWebmail

---

## Memoria

Ricardo Vázquez Almagro

13/06/2011



## Contenido

---

Tabla de diagramas.....	4
I. Introducción .....	5
II. Justificación de la necesidad del desarrollo software .....	5
1. ¿Por qué un cliente de correo <i>web</i> ? .....	5
2. ¿Por qué correo electrónico cifrado?.....	5
3. ¿Por qué correo electrónico firmado? .....	6
4. ¿Por qué criptografía asimétrica o de clave pública? .....	6
5. ¿Por qué PGP? .....	7
III. Objetivos del proyecto.....	8
1. Objetivos generales.....	8
2. Objetivos específicos.....	8
3. Objetivos didácticos .....	8
IV. Planificación inicial y real .....	9
V. Productos obtenidos.....	11
VI. Análisis .....	12
1. Análisis de Necesidades.....	12
2. Análisis de Requisitos .....	13
2.1. Requisitos no funcionales.....	13
2.2. Requisitos funcionales .....	17
3. Casos de uso .....	19
3.1. Casos de uso de configuración de cuentas.....	19
3.2. Casos de uso de redacción de emails.....	20
3.3. Casos de uso de lectura de emails.....	21
4. Modelo conceptual del sistema.....	22
VII. Diseño .....	23
1. Arquitectura del sistema .....	24
1.1. Arquitectura lógica MVC .....	24
1.2. Arquitectura lógica en tres capas .....	26
1.3. Arquitectura física: Cliente-Servidor.....	26
1.4. Arquitectura de red: Intranet.....	27
2. Diagramas de secuencia .....	28
2.1. Acceso y registro de usuarios: <i>PerfilController</i> .....	28
2.2. Configuración de cuentas de correo: <i>CuentaCorreoController</i> .....	30
2.3. Redacción de correos electrónicos: <i>RedaccionController</i> .....	30
2.4. Lectura de correos electrónicos: <i>LecturaController</i> .....	32
3. Diagrama de Clases de Diseño .....	32
4. Diseño de la Base de Datos. Modelo Relacional .....	34
5. Interfaz gráfica de usuario.....	35
VIII. Implementación .....	38
1. Proyectos incluidos en la Solución.....	38
2. Proyecto PGPWebmail: Modelo .....	38
2.1. Entity Framework .....	38

2.2. Recepción POP y envío SMTP .....	40
2.3. Modificaciones necesarias para cifrar y descifrar con adjuntos.....	41
2.4. Proveedor de membresía.....	41
2.5. El Repositorio.....	42
3. Proyecto PGPWebmail: Controladores .....	43
3.1. Routing hacia las Actions de los Controllers.....	43
3.2. Lo que devuelven las Actions .....	44
3.3. Evitar que el Internet Explorer haga caché del resultado de acciones .....	46
4. Proyecto PGPWebmail: Vistas .....	46
4.1. Renderizado especial para iPhone como prueba de concepto .....	46
4.1. jQuery: Código aliado del Ajax que corre en cliente.....	46
4.2. Marcadores <% %> en las .aspx: Código que corre en servidor .....	47
5. Proyecto PGPWebService.....	49
5.1. Servicio Web WCF.....	49
5.2. Implementación criptográfica .....	50
5.3. El cifrado de adjuntos .....	50
6. Instaladores .....	50
7. Unit Tests.....	51
IX. Objetivos conseguidos.....	52
X. Evaluación de costes.....	53
XI. Trabajo futuro y recomendaciones de mejora.....	54
1. Ayuda en línea .....	54
2. Retomar el proyecto de <i>Unit Testing</i> .....	54
3. Embellecer la interfaz gráfica y <i>renderizado</i> para más dispositivos .....	54
4. Solucionar ciertos agujeros de confidencialidad .....	55
5. Completar funcionalidades de cliente de correo.....	55
6. Completar funcionalidades como aplicación web .....	55
7. Completar funcionalidades de llavero PGP .....	55
XII. Conclusiones .....	56
XIII. Bibliografía .....	58
1. Publicaciones .....	58
2. Cursos en línea.....	59
3. Artículos en línea .....	59

## Tabla de diagramas

---

Ilustración 1: Planificación inicial .....	9
Ilustración 2: Planificación real del Análisis y Diseño .....	10
Ilustración 3: Planificación real de la Implementación.....	10
Ilustración 4: Casos de uso de Configuración de cuentas.....	20
Ilustración 5: Casos de uso de Redacción de emails.....	21
Ilustración 6: Casos de uso de Lectura.....	22
Ilustración 7: Modelo conceptual del sistema .....	23
Ilustración 8: Arquitectura lógica MVC.....	25
Ilustración 9: Arquitectura lógica en tres capas .....	26
Ilustración 10: Diagrama de arquitectura física .....	26
Ilustración 11: Diagrama de arquitectura de red .....	27
Ilustración 12: Diagrama de secuencia de Registro de nuevo usuario.....	29
Ilustración 13: Diagrama de secuencia de asociación de nueva cuenta de correo.....	30
Ilustración 14: Diagrama de secuencia de redacción de email .....	31
Ilustración 15: Diagrama de secuencia de carga de lista de recibidos .....	32
Ilustración 16: Diagrama de Clases de Diseño .....	33
Ilustración 17: Estructura de la base de datos: Modelo relacional .....	34
Ilustración 18: GUI - Pantalla de acceso a la aplicación.....	35
Ilustración 19: GUI - Asociación y configuración de cuenta de correo.....	35
Ilustración 20: GUI - Lista de cuentas de correo asociadas.....	36
Ilustración 21: GUI - Redacción de un correo.....	36
Ilustración 22: GUI - Bandeja de entrada.....	36
Ilustración 23: GUI - Lectura de email firmado .....	37
Ilustración 24: GUI - Renderizado especial para dispositivo móvil.....	37
Ilustración 25: Tablas correspondientes al diagrama EF en base de Datos.....	38
Ilustración 26: Diagrama de entidades del modelo generado por Entity Framework .....	39
Ilustración 27: Ejemplo de propiedad añadida a una clase EF de código autogenerado .....	40
Ilustración 28: Utilización en vista de propiedad añadida al código autogenerado EF .....	40
Ilustración 29: Modificaciones en Web.config para personalizar el proveedor de membresía	42
Ilustración 30: Routing entre URLs y Acciones de los Controladores.....	43
Ilustración 31: Ejemplo de jQuery: Habilitar un botón según la longitud de la contraseña .....	47
Ilustración 32: Marcadores <% %>: Código que se ejecuta en servidor para generar la página	48
Ilustración 33: Service Reference a PGPWebService (Cifrado) .....	50
Ilustración 34: Propuestas de mejora gráfica mediante hojas de estilo en CodePlex.....	55

## I. Introducción

---

El objetivo de este Trabajo Final de Carrera sobre desarrollo en .NET ha sido la toma de contacto con alguna de las últimas tecnologías de desarrollo de software introducidas por Microsoft dentro del vasto conjunto que se engloba dentro de la extraordinaria plataforma .NET.

La elección finalmente recayó en el desarrollo de una aplicación web con ASP NET MVC 2 que a su vez hiciese uso de un servicio web WCF.

La aplicación web implementa un cliente de correo y el servicio web brinda funcionalidades de criptografía asimétrica PGP. De la interacción entre ambos obtenemos un cliente web de correo electrónico con capacidad de cifrado y firma PGP para comunicación segura.

## II. Justificación de la necesidad del desarrollo software

---

Éstas son las razones que nos han llevado a desarrollar un cliente web de correo electrónico seguro mediante criptografía PGP:

### 1. ¿Por qué un cliente de correo web?

---

Existen clientes de correo para PC que soportan el cifrado y firma PGP, como el *Claws Mail* o el *plugin FireGPG* para *Mozilla Firefox*, pero no los hay para dispositivos móviles.

Y sin embargo con el actual boom de los dispositivos de acceso a Internet (*smart-phones*, tabletas, web-TV, etc.) que no tienen arquitectura ni sistema operativo de PC (iOS, Android, etc.) cada vez es más frecuente no encender nuestro PC (¡cuánto tarda en arrancar!) para leer nuestro correo y muchas veces tampoco para redactarlo.

Por lo tanto si sólo tengo un cliente que soporte PGP en mi PC, no voy a utilizar cifrado: No me interesa: ¿Cómo lo leo luego desde el móvil, si no puedo descifrarlo? (es un ejemplo).

Necesitaría para ello poder usar como cliente de correo una aplicación universalmente accesible.

Lo que garantiza la universalidad del acceso a una aplicación es que ésta sea web.

En concreto, si de verdad no quiero tener barreras de acceso, necesito una aplicación web basada en HTML, como las que se generan con ASP. *Silverlight*, -como *Flash*-, permite muy ricas interfaces gráficas, pero no todos los navegadores lo soportan, sobre todo los de los dispositivos de menor potencia, como los teléfonos.

Por lo tanto si sé que mediante una aplicación web universalmente accesible voy a poder utilizar PGP, no me va a importar cifrar y/o firmar mis correos como una práctica habitual, puesto que posteriormente voy a poder acceder a ellos desde cualquier lugar y sin ningún problema y así también podrán hacerlo sus destinatarios.

### 2. ¿Por qué correo electrónico cifrado?

---

En cuanto a la necesidad de cifrado del correo electrónico, baste con recordar que el secreto de las comunicaciones es un derecho constitucionalmente protegido. Que como todos los Derechos Fundamentales y Libertades Públicas ha costado mucho esfuerzo y mucha lucha el conseguirlo, y que merece la pena trabajar –también desde la ingeniería informática- por preservarlo, facilitarlo y protegerlo.

*Hace unas semanas me sorprendió que el abogado que representaba a la familia que me demandó pareciera saber de antemano la estrategia que habíamos planeado mi abogado y yo para aquel juicio.*

*La habíamos diseñado a través de diversos correos electrónicos, puesto que no vivimos en la misma ciudad.*

*Pero su dirección de email era una dirección genérica para su asesoría.*

*Sus empleados tienen libre acceso a esta cuenta.*

*Al parecer uno de estos empleados hizo un dinero proporcionándole al abogado contrario copia de nuestros correos.*

*Si nuestros correos hubieran estado cifrados, aunque hubiera podido acceder a su contenido no hubiera podido entender una palabra.*

*Nuestras comunicaciones hubieran estado protegidas y tal vez yo no habría perdido en ese juicio.*

En cualquier caso, tenemos derecho a que ningún intermediario entre nosotros y nuestros destinatarios tenga acceso al contenido de nuestras comunicaciones electrónicas. Da igual que ese intermediario sea legal (el Sr. Google) o ilegal (un *hacker*).

Nuestra aplicación es una ayuda en esta línea.

### 3. ¿Por qué correo electrónico firmado?

En cuanto a la necesidad de firmar un correo electrónico, la firma permite verificar que la identidad del remitente es la de quien dice ser, basándose en la red de confianza que constituye PGP.

En un mundo en el que la suplantación de identidad digital es cada vez más frecuente, tomar por costumbre firmar criptográficamente nuestros correos es una excelente costumbre.

Nada es tan fácil como hacernos pasar por otra persona en un correo electrónico.

*Hace unas semanas, en una lista de correo en la que participo, uno de los miembros decía que si alguien tenía la película tal. Y otro alguien le respondía que sí, que él la tenía bajada, que le hacía una copia y se la dejaba en casa de cual.*

*Yo entonces cambié la configuración de mi GMail para que en vez de mi nombre al destinatario le apareciera “Ángeles G. Sinde (M. Cultura)”, y escribí a la lista diciendo que habían sido descubiertos por el Ministerio de Cultura y que se enfrentaban a penas de hasta 3 años de prisión... Ni qué contar el revuelo que armé. Luego aclaré la broma, claro.*

Y es que nadie cuestiona que el remitente de un email es quien ahí pone. Pero lo que “ahí pone” lo configura quien envía el mail. Y puede poner lo que quiera sin que esto sea verificable de ninguna forma... salvo mediante firma criptográfica.

### 4. ¿Por qué criptografía asimétrica o de clave pública?

Si pensamos en criptografía tenemos dos opciones principales: La criptografía de clave compartida (o simétrica) y la de clave pública (o asimétrica).

Para el proyecto desarrollado necesitamos clave pública por dos razones:

- 1) Sólo la criptografía de clave pública permite firma digital, que es uno de los objetivos.
- 2) Nuestro otro objetivo es permitir el uso de cifrado en los correos de forma generalizable. Para ello es muy poco conveniente que mi destinatario tenga que conocer la clave secreta que yo uso para cifrar los correos que escribo y yo conocer la suya.

*Lo más adecuado es que yo cifre los correos que envíe a Francisco Toledano, mi abogado, con la clave pública de Francisco Toledano, que puedo obtener fácilmente de un repositorio público de claves sin tener siquiera que llamarle por teléfono. Cuando le llegue mi correo, ese correo sólo podrá descifrarlo la clave privada de Francisco, a la que sólo él*

*tiene acceso. Sólo él puede descifrarlo pero no necesito que comparta para ello su clave secreta conmigo. Si el abogado tuviera que compartir su clave de encriptación con todos sus clientes, mala privacidad tendríamos.*

Por tanto, el modo de cifrado que necesita nuestro proyecto es el llamado de clave pública.

## 5. ¿Por qué PGP?

Como decíamos en el epígrafe anterior, el cifrado y firma mediante clave pública requiere que los intervinientes en la comunicación tengan acceso a un repositorio de claves públicas en el que encontrarán la clave pública de su interlocutor, mediante la que cifrarán su mensaje o mediante la cual verificarán la identidad de la firma de los mensajes que reciban.

El acceso a estas claves públicas requiere de una infraestructura, la llamada PKI (*Public Key Infrastructure*), que tenga esos repositorios y que dé fe de la identidad de la persona que dice ser titular de una clave pública.

*De poco me sirve escribir a mi abogado un correo cifrado con su clave pública si en el repositorio de claves su identidad ha sido suplantada, y su clave no es suya en realidad.*

Esta labor de certificación de identidades pueden hacerla Autoridades de Certificación tales como *beTRUSTed*, *Buypass*, *Certplus*, *certSIGN*, *Comodo*, *Cybertrust*, *Entrust*, *GlobalSign*, *Microsoft Internet Authority*, *Network Solutions*, *RSA Security*, *USERTRUST*, *Verisign*, *Visa*, *Wells Fargo*, etc. Estas autoridades forman estructuras piramidales: autoridades de más alto nivel certifican a autoridades de más bajo nivel, hasta llegar al nivel más bajo que es el usuario final.

Pero los servicios de certificación de estas autoridades no son servicios gratuitos.

En consecuencia no parece la opción ideal para ayudar a la generalización del uso del cifrado en las comunicaciones *entre particulares*.

Frente a este modelo de confianza jerárquico, existe una alternativa: el modelo de confianza distribuido, en el que se basa el criptosistema llamado “PGP” (*Pretty Good Privacy*) y diversas variantes que sobre el mismo se han hecho posteriormente: *OpenPGP*, *GnuPG*, etc.

La idea de PGP es crear una “red de confianza”. Veámoslo con un ejemplo:

*Yo tengo acceso a la clave pública de mi abogado en un repositorio público; si puedo confirmar con él que efectivamente esa es su clave, puedo “firmarla con la mía”; de este modo, criptográficamente, afirmo estar seguro de que esa clave pública es de quien dice ser.*

*Cuando un amigo mío quiera escribir al abogado, verá que su clave está firmada por mí. Él, que ya ha tenido comunicaciones cifradas conmigo, sabe que mi clave es mía; ve que mi clave “firma” la clave del abogado. Pues ya directamente, como confía en que yo habré verificado la veracidad de la identidad de esa clave, y confía en mí, puede, transitivamente, confiar él en que esa es la clave del abogado, y usarla para escribirle.*

De este modo se va tejiendo una “red de confianza”, -como una telaraña, frente a la estructura piramidal de las Autoridades de Certificación-, que no requiere de autoridades, y que por tanto es suficientemente confiable (“pretty good”) y al mismo tiempo gratuita.

Creemos que esta idea es muy buena, que hay que potenciarla.

### III. Objetivos del proyecto

---

#### 1. Objetivos generales

---

- a) Hacer fácilmente accesible el cifrado de correos electrónicos mediante PGP, para garantizar la privacidad de la comunicación mediante este medio.
- b) Hacer fácilmente accesible la firma criptográfica de correos electrónicos mediante PGP, para hacer verificable la identidad del remitente que lo firma y para poder tener comunicaciones mediante e-mail que permitan el no-repudio (el emisor del mensaje firmado no puede negar que el mensaje es efectivamente suyo, y por tanto esto le permite usar el medio electrónico para prestar válidamente su voluntad).
- c) Que ese acceso sea lo más universal posible, desde cualquier dispositivo con acceso a Internet (ordenador, tableta, *smart-phone*, etc.), con cualquier sistema operativo (Mac OS, iOS, Android, Windows, Linux, etc.), para lo cual crearemos un cliente web basado en HTML.

#### 2. Objetivos específicos

---

- a) Desarrollar en lenguaje C# un cliente de correo electrónico web mediante ASP.NET MVC, de interfaz gráfica sencilla pero que funcione bien no sólo en ordenadores sino también en *smart-phones*.
- b) Desarrollar en lenguaje C# un Servicio Web WCF que exponga una máquina que permita el cifrado, descifrado, firma y verificación de mensajes mediante cifra asimétrica PGP.

#### 3. Objetivos didácticos

---

Aprendizaje o profundización en las siguientes tecnologías de desarrollo de software:

- a) Lenguaje de programación C#, en la versión que brinda la versión 4 de .NET Framework.
- b) LINQ, lenguaje de consulta “tipo SQL”, pero integrado en el núcleo mismo del lenguaje C#, en su versión *LINQ to Entities*.
- c) ADO.NET Entity Framework, como potente sistema de acceso a la base de datos que utiliza un mapeado automatizado entre la estructura relacional de la base de datos y la estructura orientada a objetos del código en C#.
- d) ASP.NET MVC, versión de ASP.NET adaptada al patrón de arquitectura MVC (Modelo – Vista – Controlador) especialmente orientado al desacople de capas, lo cual es ideal para aplicaciones web cuya capa de presentación está muy desacoplada por su propia naturaleza.
- e) Servicios Web WCF, un paso más allá de los Servicios Web ASP.NET.
- f) Ajax, para, mediante la interacción con el código que se ejecuta en el navegador cliente (JavaScript) cargar parcialmente las páginas, para lograr una experiencia de usuario más suave y fluida.



## IV. Planificación inicial y real

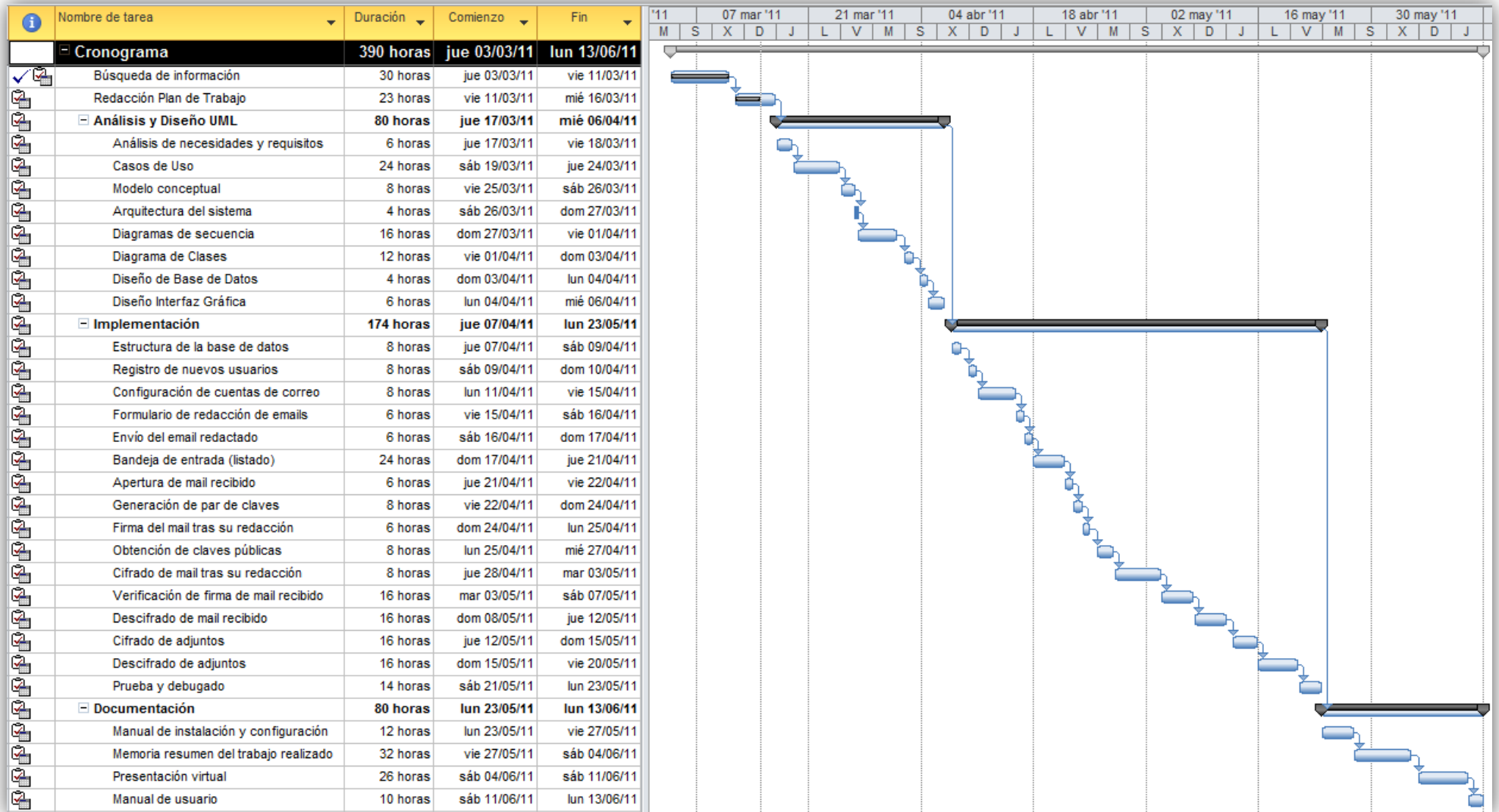


Ilustración 1: Planificación inicial

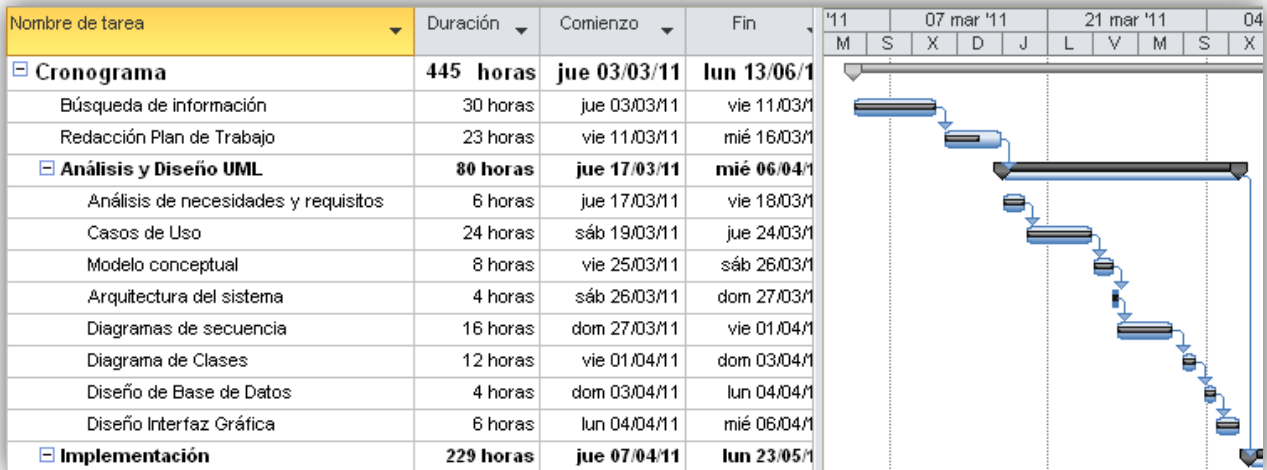


Ilustración 2: Planificación real del Análisis y Diseño

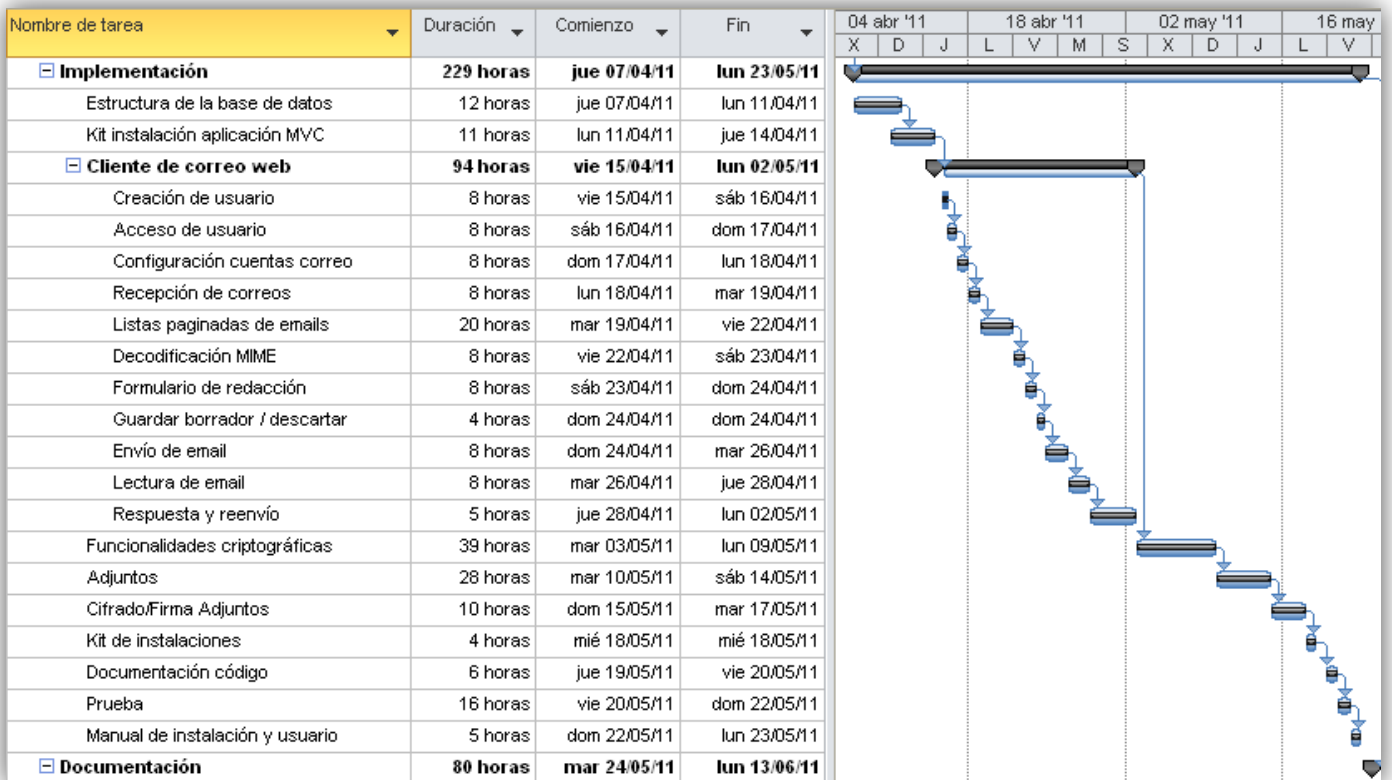


Ilustración 3: Planificación real de la Implementación

Como podemos concluir de la observación de la planificación inicial y de su comparación con la planificación final, la fase de análisis estaba perfectamente prevista, pero hemos necesitado sacar 55 horas más para la fase de implementación, que inicialmente estaba pensada en 174, pero que finalmente ha consumido 229 horas.

Los días empleados son los mismos, pero se ha trabajado más horas diarias.

También cambia el desglose en tareas de la implementación.

Las principales tareas de las que consta la implementación son: cliente de correo web, funcionalidades criptográficas y adjuntos.

En la planificación inicial pensamos que en 66 horas podríamos tener desarrollado un cliente de correo web; sin embargo han sido necesarias 94. Para la implementación del servicio web de

criptografía y su consumo por el cliente web estimamos que necesitaríamos 62 horas, y hemos utilizado sólo 39. Para los adjuntos, tanto en claro como cifrados, planificamos 32 horas, y hemos utilizado 38.

En general podemos concluir que la planificación inicial era bastante precisa, si bien se necesitaban bastantes más horas para desarrollar el cliente de correo web, bastantes menos para dar funciones criptográficas, y no habíamos contado con el tiempo de hacer los instalables, los manuales y el documento de comentarios a la implementación, que creíamos que venían en la fase posterior, de documentación, pasado el 23 de mayo; todo lo cual nos ha incrementado en 55 horas el trabajo de desarrollo. Por lo tanto, sobre una estimación inicial total de 390 horas, supone un incremento del 14% del tiempo previsto.

## V. Productos obtenidos

---

Los entregables que se han obtenido durante la realización del TFC son:

### **Plan de Trabajo:**

Con la descripción y justificación del proyecto, sus objetivos, una breve descripción de las funcionalidades que debería incluir, tecnologías propuestas y organización temporal.

### **Documento de Análisis y Diseño:**

Que incluye, como productos destacados:

- Análisis de necesidades
- Análisis de requisitos funcionales
- Análisis de requisitos no-funcionales
- Descripción textual y diagramas de los casos de uso
- Diagrama del modelo conceptual
- Diagramas de arquitectura, lógica, física y de red
- Diagramas de secuencia que ilustran los casos de uso
- Diagrama de clases de diseño
- Diagrama del modelo relacional de la base de datos
- *Mock-ups* de diseño de la interfaz gráfica
- Diagrama de Gantt de planificación del proyecto

### **Solución Visual Studio:**

Que consta a su vez de:

- Dos proyectos independientes:
  - *PGPWebmail*: aplicación web ASP.NET MVC cliente de correo con soporte PGP
  - *PGPWebService*: servicio web WCF que brinda funcionalidad de criptografía PGP, gracias a cuyo uso, *PGPWebmail* permite cifrado y firma
- Dos proyectos de instalación para cada una de las aplicaciones anteriores
- Un proyecto de biblioteca de enlace dinámico para la personalización de las instalaciones anteriores

### **Instalables:**

Ejecutables de instalación del servicio web WCF *PGPWebService* y de la aplicación web ASP.NET MVC *PGPWebmail*.

### **Documento de comentarios a la implementación:**

Con una reflexión sobre los puntos más destacados o de especial dificultad o interés del código de la solución Visual Studio anteriormente referida.

**Documento de pruebas:**

Que documenta someramente las pruebas funcionales que pasó la solución software.

**Manual de instalación:**

Que explica detalladamente los pasos a seguir para, utilizando los sencillos instalables arriba referidos, tener la solución software corriendo correctamente sobre el IIS de una máquina con Windows 7.

**Manual de usuario:**

Que refiere detalladamente el uso de las diversas funcionalidades de la solución software.

**Presentación virtual:**

Vídeo explicativo y demostrativo de la solución software *PGPWebmail*.

**Memoria:**

El presente documento.

## VI. Análisis

---

Partiendo de las necesidades que quiere cubrir el cliente, analizamos los requisitos, funcionales y no-funcionales que debe cumplir el sistema informático para que aquéllas sean satisfechas.

### 1. Análisis de Necesidades

---

La necesidad básica que hemos detectado en el cliente –y que nuestro proyecto debe satisfacer- es la de mantener comunicaciones secretas mediante correo electrónico y, a la vez, poder acceder a este medio de comunicación desde dispositivos *no-PC*.

Para desglosar esta necesidad básica, debemos intentar pensar en lo que el usuario va a esperar de la aplicación, para lo cual analizo el servicio que prestan aplicaciones análogas: web-mails y *front-ends* de PGP:

Los web-mails ofrecen:

- Gestión del usuario: creación, edición del perfil, log-on y log-off.
- Bandeja de entrada: con el listado de correos recibidos, dividido por tantas páginas como sea necesario y desde donde podremos borrar, reenviar, abrir o mover a carpeta cada correo (no nos parece una necesidad de primer nivel implementar un sistema de carpetas para los correos, podría ser una mejora futura).
- Lectura del correo: Con posibilidad de responderlo, responderlo a todos, reenviarlo o borrarlo.
- Redacción de nuevo email: Con posibilidad de guardar borrador, adjuntar archivos y enviar. También dan prestaciones de texto enriquecido; pero tampoco lo vemos una necesidad de primer nivel y queda para posible mejora futura.

Los *front-ends* de PGP ofrecen:

- Creación de par de claves criptográficas
- Subir la clave pública a repositorios
- Buscar en repositorios claves públicas
- Exportar claves desde el llavero privado (no es necesidad de primer nivel: para futura mejora)

- Importar claves al llavero privado (tampoco es necesidad de primer nivel: para futura mejora)
- Firmar claves públicas que sabemos confiables (tampoco es necesidad de primer nivel: para futura mejora)
- Cifrar texto o archivo
- Descifrar texto o archivo
- Firmar texto o archivo
- Verificar la firma de un texto o archivo

El usuario va a esperar de *PGPWebmail* que ofrezca la unión de todas estas funcionalidades.

## 2. Análisis de Requisitos

Identificadas las necesidades del cliente, las traducimos en los requisitos que deberá cumplir la aplicación para que aquéllas sean satisfechas.

### 2.1. Requisitos no funcionales

Lo primero que nos planteamos son los requisitos no funcionales, es decir, el conjunto de requisitos que la aplicación debe cumplir pero que no se refieren a ninguna funcionalidad concreta, sino que forman parte de la “experiencia de usuario”. Representan cualidades “intangibles” del sistema.

A este respecto identificamos:

#### **Factores humanos:**

(NF-01) Máximo 200 usuarios: Se trata de una aplicación web que potencialmente podría ser usada por miles de usuarios de forma concurrente. Pero el cliente sólo necesita para un primer despliegue una aplicación intranet que asegure el uso concurrente de varias decenas de usuarios, con un total máximo de 200 usuarios del sistema en total.

(NF-02) Con experiencia en el uso de correo web: Los usuarios de nuestra aplicación tendrán ya experiencia previa en el uso de correo web. Sin embargo no tendrán experiencia en criptografía. Por lo tanto todo lo relacionado con el cifrado y firma debe ser de lo más transparente para el usuario; y cuando esto no fuera posible, debe estar muy claramente explicado.

(NF-03) Interfaz web similar a la de los web mails: Facilidad de comprensión y uso del sistema: La interfaz web debe ser lo más similar posible a las de los web mails más comunes, simplificándola. La parte relativa al cifrado y firma debe ser de lo más transparente para el usuario; y cuando esto no fuera posible, debe estar muy claramente explicado.

#### **Ambiente físico:**

(NF-04) Despliegue en un solo servidor ASP: Número de lugares donde se va a instalar la aplicación: En el despliegue inicial del que hablamos la aplicación se instalará en un solo servidor ASP. A ella se accederá desde otras máquinas de la intranet a través del navegador web.

(NF-05) En Intranet: Todos los usuarios se encuentran dentro del mismo dominio de red, en una intranet: Si bien potencialmente podría ser una aplicación accesible desde cualquier lugar a través de Internet, dado que la seguridad es crítica, el despliegue que nos pide el cliente es dentro de una LAN: el navegador web puede estar en una máquina, el servidor ASP en otra, pero dentro de una misma LAN. Lógicamente se puede acceder remotamente por VPN. En un despliegue futuro podría pensarse en servir esta aplicación de forma universal a través de HTTPS.

#### **Seguridad:**

(NF-06) Acceso con autenticación modo “Forms”: Definición del control de acceso a la aplicación: Es necesario un acceso seguro de los usuarios y las adecuadas protecciones para que nadie pueda acceder a datos a los que no esté autorizado.

### **Hardware:**

**(NF-07) Servidor único:** Definición de hardware que se va a utilizar: Puesto que el despliegue es sólo para una LAN y con no más de varias decenas de usuarios concurrentes, sólo necesitaremos un servidor para el *Internet Information Server (IIS7)* con el motor de ASP.NET y el mismo u otro igual para el *SQL Server 2008*.

Este servidor deberá tener:

- como mínimo 2GB de RAM (recomendado 4GB);
- tarjeta de red recomendada de 1Gigabit;
- procesadores recomendados, *Intel Xeon, AMD Opteron* o *AMD Athlon* (especialmente debe buscarse un procesador con el máximo tamaño posible de memoria caché, L1, L2 Y L3);
- recomendado doble o cuádruple procesador (interesa sobre todo si IIS se ejecuta en la misma máquina que *SQL Server* y los servicios de encriptación);
- para el almacenamiento se recomienda un *Redundant Array of Independent Disks (RAID 1 –discos espejo- o RAID 5 –fragmentación de discos con paridad)* de dispositivos SCSI (un RAID nos permitirá seguir dando servicio aunque falle un disco duro) que permita 1TB de almacenamiento (con 5GB de almacenamiento por usuario es suficiente, lo que permitiría más de 200 usuarios, suficiente para el despliegue inicial del que hablamos);
- finalmente se recomienda tener el servidor conectado a un Sistema de Alimentación Ininterrumpida (SAI) configurado para que el servidor se apague limpiamente por sí mismo en caso de corte del fluido eléctrico para evitar daños en el hardware.

En cuanto a los clientes, no tienen ningún requisito de hardware, ni de arquitectura, tan sólo deberán tener acceso a Internet y a la LAN donde se realice el despliegue y poder ejecutar un navegador web.

### **Software:**

**(NF-08) Software base: Windows 7:** Definición del software base: Aunque lo ideal para ejecutar servidores de base de datos y de aplicaciones web sería un sistema operativo Microsoft® Windows® de servidor, como el Windows Server 2008 o Windows Server 2008 R2, el cliente nos impone que nuestra aplicación debe poder ejecutarse sobre Windows 7, ya que son las licencias que tienen adquiridas. Realizaremos por tanto el desarrollo orientado a que nuestra aplicación funcione sobre Windows 7.

**(NF-09) Software complementario: IIS7, .NET 4:** Definición de software complementario: Será necesario que el servidor que albergue nuestra solución tenga instalado este software:

- El servidor web Microsoft® Internet Information Services (el incluido en Windows 7 es su versión 7.5, que será entonces sobre la que realizaremos el desarrollo y prueba).
- Microsoft .NET Framework 4.

No es necesaria la instalación de módulos criptográficos, de la que se encargará la propia instalación del módulo *servicio web de cifrado PGP* de nuestra aplicación.

### **Base de datos:**

**(NF-10) Sistema Gestor de Base de Datos: SQL Server® 2008 Express:** Para el despliegue previsto de hasta 200 usuarios, no más de varias decenas concurrentes, no es necesaria la versión de pago, Microsoft® SQL Server® 2008 o SQL Server® 2008 R2, bastando con la versión *Express* que es gratuita.

**(NF-11) Acceso a BD por autenticación Windows y en modo local:** Definición de control de acceso a la Base de Datos: Para conseguir máxima seguridad, nuestra base de datos debe deshabilitar protocolos de acceso remoto, permitiendo sólo el protocolo de memoria

compartida<sup>1</sup>. Para ello el módulo *cliente webmail* debería instalarse en la misma máquina que el SQL Server. Si el cliente desea instalarlo en máquina aparte, habrá que habilitar un protocolo de red, como las *canalizaciones con nombre*, ideales para entornos LAN, manteniendo por seguridad el TCP/IP y el VIA deshabilitados.

Dicho esto, la forma de autenticación en el SQL Server será Windows, es decir, que el acceso a la misma vendrá dado por los privilegios de los que esté dotado el usuario IIS ASP.NET (en Windows 7 el *DefaultAppPool*<sup>2</sup>). Por otra parte, en base de datos guardaremos las contraseñas de los usuarios cifradas. El contenido de los emails cifrados se almacena cifrado PGP y no en su versión “en claro”.

(NF-12) Copias de seguridad: Definición de frecuencia de backups y responsables: Consultado el cliente, nos informa de que el Administrador de Sistemas realiza una copia de seguridad semanal de todos sus servidores, y que estas copias se almacenan en otro edificio. Los servidores sobre los que se despliegue *PGPWebmail* deben estar incluidos entre el conjunto de servidores sobre los que se hace este backup semanal. Además se recomienda programar la tarea de un backup diario específico de la base de datos durante la noche, backups que se mantendrán durante una semana. Si a todo ello unimos que el almacenamiento es en RAID, creemos que hay suficiente seguridad para el despliegue intranet solicitado.

(NF-13) Almacenamiento de datos históricos hasta 5GB por usuario: Siguiendo la filosofía de los *webmails* que hemos usado como fuentes de información, nos parece interesante que el usuario pueda mantener sus emails de manera indefinida en la web. Por lo tanto la limitación no será de tiempo, sino sólo de espacio: 5GB de almacenamiento por cuenta de usuario. De este modo será el propio usuario el que se encargará de borrar lo más antiguo o menos útil cuando vaya llegando a este límite. Por lo tanto no se necesita la intervención del administrador del sistema, sino que la función descansa en el propio usuario de *PGPWebmail*.

(NF-14) Procesos SQL Server de mantenimiento: Definición del mantenimiento de la base de datos: Ya hemos indicado que se programará un backup nocturno diario de la base de datos, backups que se mantendrán durante una semana. Además será necesario que el administrador del sistema programe una vez alasemanaun procesoSQL Server que recalcule las estadísticas sobre la base de datos, reorganice índices y datos, compruebe integridad y elimine espacio no utilizado dentro de la base de datos, todo lo cual permitirá mantener el óptimo rendimiento.

### **Comunicaciones:**

(NF-15) Comunicación TCP/IP HTTP sobre LAN con la aplicación: Formas de comunicación hacia la aplicación: En el despliegue intranet que nos solicita el cliente, el acceso a nuestra aplicación será web, pero dentro de una misma LAN o WLAN segura, si bien, lógicamente, a dicha LAN se puede acceder remotamente por VPN. En un segundo paso de despliegue, si el cliente lo solicitara, podría dársele acceso universal a través de HTTPS; (el HTTPS sería aquí necesario porque es imprescindible dar seguridad al tramo que va desde el navegador web y el servidor ASP.NET, ya que se trata de un tramo no protegido por PGP).

### **Restricciones de diseño:**

(NF-16) Idioma castellano: Definición de idioma de la aplicación: El idioma de la aplicación será el castellano. La plataforma .NET tiene muy interesantes recursos de localización de aplicaciones multi-idioma, pero utilizarlos excedería las posibilidades de tiempo que tenemos para el proyecto.

---

<sup>1</sup> Microsoft (2011) “Elegir un protocolo de red. SQL Server 2008 R2”. MSDN [artículo en línea]. [Fecha de consulta: 21 de marzo de 2011].

<http://msdn.microsoft.com/es-es/library/ms187892.aspx>

<sup>2</sup> Microsoft (2011) “Application Pool Identities”. IIS [artículo en línea]. [Fecha de consulta: 17 de mayo de 2011].

<http://learn.iis.net/page.aspx/624/application-pool-identities/>

(NF-17) Hasta 25.000 mensajes por semana, máximo 2 MB por mensaje: La aplicación debe diseñarse para soportar unas 25.000 transacciones por semana. El tamaño máximo por mensaje de correo electrónico, incluidos adjuntos, será de 2MB.

(NF-18) 50 correos por pantalla: El máximo número de correos que deben listarse por pantalla son 50. No configurable.

(NF19) Varias decenas de usuarios concurrentes: La concurrencia que debe soportar la aplicación en el despliegue solicitado es de varias decenas de usuarios simultáneos.

### **Usabilidad / Capacitación:**

(NF-20) Manual de usuario y de instalación: Será responsabilidad del propio cliente realizar el entrenamiento y formación de usuarios. Sin embargo nosotros deberemos proveerle con un manual de uso sencillo dirigido a los usuarios finales, así como las instrucciones de instalación dirigidas al administrador del sistema. Además, tras la entrega de todo el sistema a cliente antes del 24 de mayo de 2011, entre el 20 y el 23 de junio estaremos a disposición del cliente para resolver remotamente todas las dudas de uso o instalación que pudiera tener.

### **Interfaces:**

(NF-21) No hay importación ni exportación de datos: Input/Output del sistema: El sistema almacenará todos los datos en una base de datos específica no accesible desde otros programas. No se importarán ni exportarán configuraciones ni listas de correos desde ni hacia otros programas.

(NF-22) El sistema no se conectará a aplicaciones externas: Salvo la lógica conexión con los servidores de correo entrante y saliente que configure cada usuario, y que se realizará exclusivamente mediante los protocolos POP3 y SMTP.

### **Instalación:**

(NF-23) Instalación: La responsabilidad de la instalación del sistema será del cliente. Nosotros nos comprometemos a facilitar a su administrador los kits de instalación necesarios así como un manual de instalación adecuado, y a prestarle soporte técnico remoto entre el 20 y el 23 de junio de 2011.

(NF-24) Horario de instalación, libre: Definición de horarios de instalación: No hay diferencia horaria entre desarrollador y cliente; y puesto que la instalación es responsabilidad del cliente, éste podrá llevarla a cabo cuando mejor le convenga. Duda sobre instalación o uso de la aplicación se responderá remotamente en el plazo de 24 horas entre los días 20 y 23 de junio de 2011.

(NF-25) No se necesita personal de apoyo en la instalación en cliente.

(NF-26): No será precisa la instalación ni utilización de aplicaciones de control remoto para auxiliar en la instalación. Las dudas se responderán por correo electrónico.

(NF-27) Tablas de base de datos ya cargadas por nuestra instalación: No será responsabilidad del cliente la carga de tablas base; esta carga se hará por nuestra propia aplicación o su instalación.

### **Soporte:**

(NF-28) Fecha final de soporte: Pasado el 1 de julio no se dará servicio de mantenimiento sobre la aplicación.

(NF-29) No es necesario acceso a cliente para prestar soporte: Hasta esa fecha, nuestro servicio de mantenimiento no necesitará para prestar su soporte acceso a ambientes de test ni de producción en el cliente.

### **Confiabilidad y Rendimiento:**

(NF-30) Confiabilidad y rendimiento "web": *PGPWebmail* deberá tener una confiabilidad y rendimientos medios, acordes al funcionamiento de otras aplicaciones web similares. Los



interfaces web sabemos que pueden tener algunos fallos que pueden salvarse (recargar la página, volver a intentar tal o cual acción); fallos que asumimos a cambio de la ubicuidad y accesibilidad universal de lo “web”, que es lo que queremos alcanzar.

### **Documentación:**

(NF-31) Manuales de uso e instalación, comentarios de implementación y código abierto: Junto con el producto software se entregarán manuales de uso y de instalación. Además todo el desarrollo estará también documentado a nivel técnico, permitiendo al cliente entender su arquitectura, estructura y código, que también se le entregará y estará abierto.

(NF-32) Sin ayuda en línea: En su primer despliegue *PGPWebmail* no tendrá ayuda en línea, pero sus funciones de cliente de correo serán lo más intuitivas posible y sus funciones criptográficas estarán comentadas sobre la misma interfaz gráfica o mediante *tooltips* cuando se considere necesario.

(NF-33) El idioma de la documentación será el castellano.

## 2.2. Requisitos funcionales

Son todas aquellas funciones que la aplicación debe cumplir para alcanzar sus objetivos y cubrir las necesidades para las que se desarrolla.

### **Cliente web: Configuración de cuentas**

A diferencia de las aplicaciones de escritorio, al tratarse de una aplicación web, lo primero con lo que tendrá que lidiar es con la existencia de múltiples usuarios, cada uno de los cuales deberá tener su propio perfil, sus cuentas de correo asociadas y configuradas y sus claves criptográficas propias.

Por lo tanto un primer grupo de necesidades funcionales son las relativas a la gestión y configuración de usuarios y cuentas.

(U-01) Crear usuario: La primera pantalla de acceso a la aplicación web debe permitir registrarse a quien aún no tenga creado un usuario. Como la aplicación funcionará dentro de una intranet, no será necesario el uso de *catcha* para protegerse contra la creación automatizada de usuarios por bots, ni otras precauciones similares.

(U-02) Hacer *log-on*/ *log-off* de usuario: La primera pantalla de acceso a la aplicación web debe permitir hacer *log-on* (autenticarse) al usuario que ya tiene un perfil registrado.

En todo momento el usuario tendrá en la interfaz gráfica la posibilidad de salir de la sesión (*log-off*).

(U-03) CRUD de cuentas de correo: A diferencia de otros servicios web de correo, nuestra aplicación, como cliente de correo que es, permitirá configurar tantas cuentas de correo como el usuario quiera. Las cuentas quedan asociadas al usuario; serán usadas todas ellas en recepción y el usuario podrá elegir cualquiera de ellas para el envío.

Por tanto, el usuario autenticado podrá crear, ver, modificar y borrar de cuentas de correo saliente y entrante. Para poder enviar correos necesita tener al menos una.

Existen varios protocolos posibles en servidores de correo; pero sólo daremos soporte a SMTP para saliente y POP para entrante con diversos métodos de autenticación. El objetivo funcional es lograr enviar y recibir con GMail y Yahoo.

(U-04) Gestión de claves criptográficas: El usuario autenticado tendrá interfaz para crear su par de claves PGP, que quedarán guardadas localmente en el servidor de la aplicación; la privada protegida mediante cifrado simétrico (el usuario deberá proporcionar una *passphrase* de protección); el usuario tendrá también interfaz para subir su clave pública a repositorios públicos universalmente accesibles.

**Cliente web: redacción de correos**

Otro grupo de funcionalidades del cliente serán las relativas a la creación y edición de nuevos correos electrónicos, así como su envío.

(W-01) Redactar email: El usuario autenticado tendrá interfaz para poder editar texto que será el contenido del nuevo correo electrónico. En esta primera versión de la aplicación, no se dará interfaz de texto enriquecido, que se identifica como una necesidad secundaria.

El usuario podrá añadir tantos archivos adjuntos como precise, con un peso máximo de 2 MB.

(W-02) Guardar borrador: En cualquier momento el usuario podrá guardar copia del mensaje que está redactando en “borradores”.

(W-03) Enviar email: Terminado el email, el usuario podrá cifrarlo y/o firmarlo y a continuación enviarlo. Sólo daremos soporte a protocolo SMTP de envío de emails.

(W-04) Gestión de borradores: El usuario podrá acceder a la lista (bandeja) de borradores guardados, visualizando un máximo de 50 por pantalla (NF-18).

Podrá seleccionar uno para continuar con su edición (y eventual envío posterior).

Y también podrá borrar uno o varios de estos borradores.

**Cliente web: lectura de correos**

La otra cara de la moneda de la creación y envío es la recepción y lectura de mails.

(R-01) Gestión de bandeja de entrada: El usuario autenticado puede ver la lista de los correos que ha recibido (remitente, asunto y fecha). Éstos estarán ordenados de forma descendente por fecha, y en cada página no se visualizarán más de 50 (NF-18).

Sólo daremos soporte al protocolo de correo entrante POP.

Desde la bandeja de entrantes podrá borrar uno o varios correos.

También podrá seleccionar en esta lista para visualizar el contenido de un correo.

En esta primera versión no se implementará la posibilidad de clasificación de los correos recibidos por carpetas.

(R-02) Visualizar el contenido de un correo: Desde la bandeja de entrada podrá seleccionar un correo para visualizar su contenido. Si está cifrado, tendrá interfaz para descifrarlo, si está firmado, tendrá interfaz para verificar esa firma. También podrá responder al remitente o al remitente y demás destinatarios. También podrá reenviarlo, lo cual le llevará a la interfaz de redacción de emails (W-01).

También podrá abrir archivos adjuntos, incluso si estos están cifrados.

En caso de que el correo sea de texto enriquecido, se presentará una versión del mismo en texto plano adecuada para su lectura; si la entidad MIME correspondiente al cuerpo es HTML se visualizará como tal.

(R-03) Gestión de la bandeja de enviados: El usuario autenticado podrá visualizar la lista de correos que ha enviado desde nuestra aplicación (destinatario, asunto y fecha). Ordenados de forma descendente por fecha, no más de 50 por página (NF18).

Podrá seleccionar en esta lista para visualizar el contenido de un correo, con las funcionalidades de lectura de correos indicadas más arriba (R-02).

**Servicio Web de criptografía PGP:**

El otro módulo de software que compondrá *PGPWebmail* será un Web Service que se encargará de encapsular las funcionalidades de criptografía necesarias para alcanzar nuestros objetivos. Estas funcionalidades ya se han mencionado desde la perspectiva del cliente web: U-04, W-03 y R-02. Pero para que el cliente pueda brindarlas, será necesario a su vez que a él se las brinde el servidor de criptografía mediante protocolo SOAP:

(C-01) Creación de par de claves criptográficas: Que serán guardadas localmente en el servidor y protegidas mediante cifrado simétrico, por lo que será necesario proporcionar al servidor una *passphrase*.

Para evitar abusos, sólo se crearán claves para una dirección de email con la que *PGPWebmail* pueda probar envío.

(C-02) Subida de clave pública a repositorios: La “etiqueta” que identifica a un par de claves PGP es un correo electrónico. Dado un correo electrónico, se podrá solicitar del servidor que suba la clave pública correspondiente a servidores de repositorios.

Para evitar abusos, sólo se subirá clave para una dirección de email con la que *PGPWebmail* pueda probar envío.

(C-03) Cifrado PGP: Dada una cadena de texto o un fichero y una o varias direcciones de correo electrónico, el servidor de criptografía podrá cifrarlo utilizando para ello las claves públicas PGP que se correspondan con esas direcciones de correo, buscándolas para ello si fuera necesario en los servidores públicos de claves.

(C-04) Descifrado PGP: Dada una cadena de texto o un fichero cifrados, una dirección de correo electrónico y una *passphrase*, el servidor de criptografía intentará descifrarlo utilizando para ello la clave privada que se corresponda a esa dirección de correo, que tendrá necesariamente que tener almacenada localmente.

(C-05) Firma PGP: Dada una cadena de texto o un fichero, una dirección de correo electrónico y una *passphrase*, el servidor de criptografía intentará firmarlo utilizando para ello la clave privada que se corresponda a esa dirección de correo, que tendrá necesariamente que tener almacenada localmente.

(C-06) Verificación de firma PGP: Dada una cadena de texto o un fichero firmados y una dirección de correo electrónico, el servidor de criptografía podrá verificar dicha firma electrónica, utilizando para ello la clave pública PGP que se corresponda con ese correo electrónico, buscándola para ello si fuera necesario en los servidores públicos de claves.

### 3. Casos de uso

---

#### 3.1. Casos de uso de configuración de cuentas

---

Engloba las acciones relativas a la administración del perfil del usuario.

Algunas de las historias a las que responde este diagrama son las siguientes:

*“El usuario desea iniciar sesión.”*

*“El usuario autenticado introduce en su perfil en el cliente de correo una nueva cuenta de correo”*

*“El usuario autenticado crea sus claves criptográficas”*

*“El usuario autenticado sube a servidores públicos su clave pública”*

*“El usuario autenticado desea finalizar la sesión”*

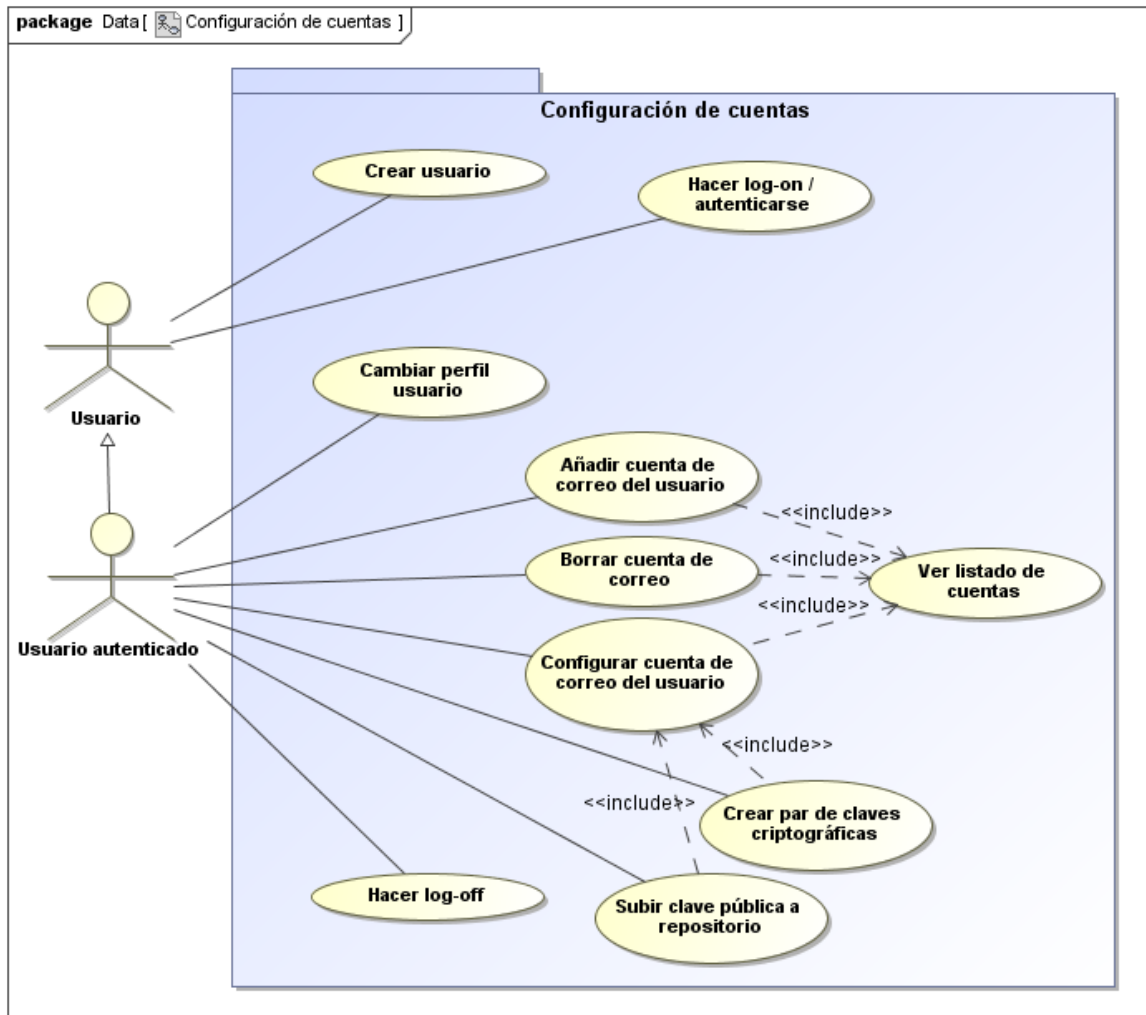


Ilustración 4: Casos de uso de Configuración de cuentas

### 3.2. Casos de uso de redacción de emails

Se trata de una colección de casos de uso relativos a la redacción de los correos electrónicos.

La idea central es la interfaz de edición de texto que permite al usuario autenticado crear un nuevo mail. Podrá añadirle adjuntos, guardarlo en borradores o enviarlo. Antes de enviarlo podrá cifrarlo y/o firmarlo.

Para poder gestionar borradores tendremos también un caso de uso de bandeja de borradores, que nos permitirá listarlos, borrarlos y editarlos.

Algunas de las historias a las que responde este diagrama son las siguientes:

*“El usuario autenticado redacta un email.”*

*“El usuario autenticado añade un adjunto al mail que está editando.”*

*“El usuario autenticado guarda el mail que está editando en borradores.”*

*“El usuario autenticado cifra y firma y envía un mail.”*

Todo ello queda expresado en el siguiente diagrama de casos de uso.

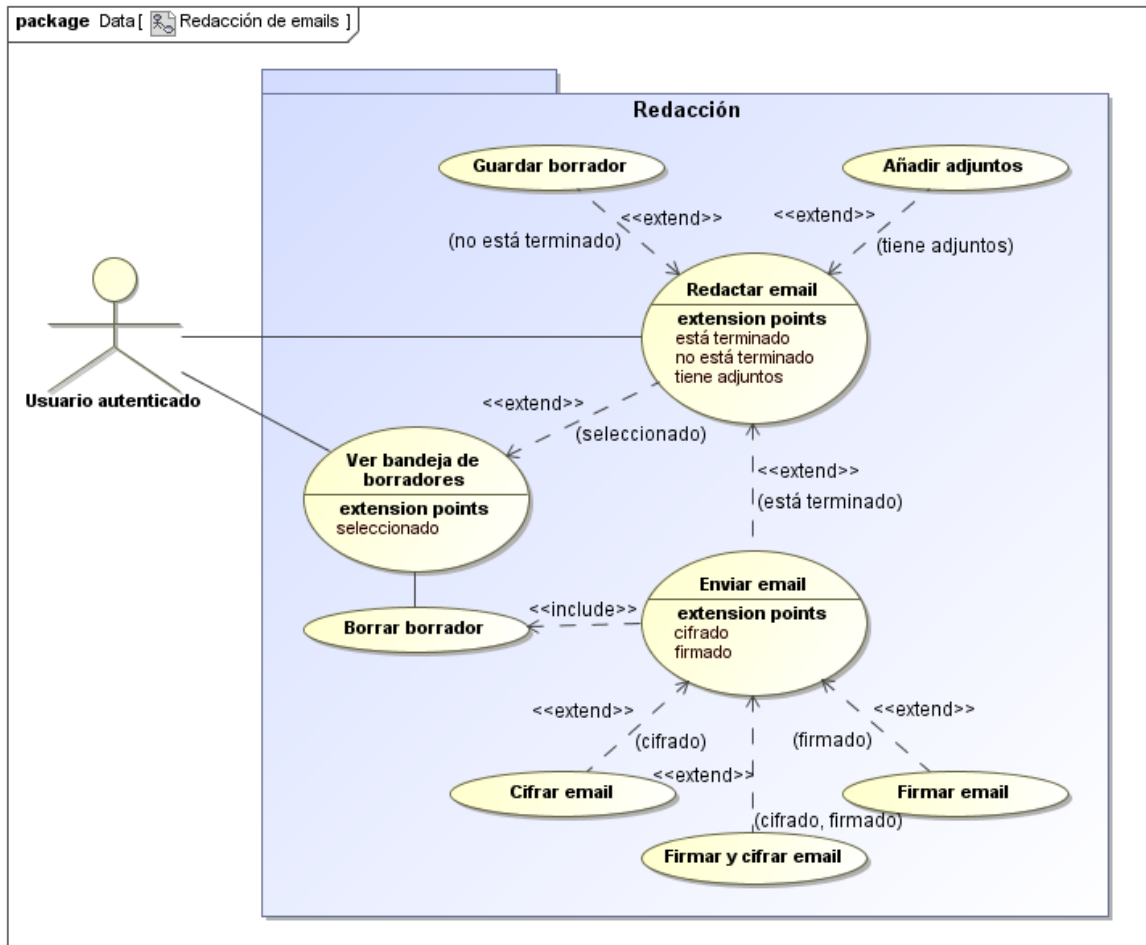


Ilustración 5: Casos de uso de Redacción de emails

### 3.3. Casos de uso de lectura de emails

Se trata de una colección de casos de uso relativos a la recepción y lectura de correos electrónicos.

La idea central es la bandeja de entrada, si bien también podrá accederse a leer los correos de la bandeja de enviados.

La lectura implicará poder descifrar los correos cifrados y verificar la firma de los firmados.

En caso de que el correo sea de texto enriquecido, se mostrará una versión legible en texto plano o en HTML si una de las entidades MIME lo lleva.

Desde la lectura del mail, podrá responderse, responderse a todos o reenviarse, lo que nos llevará al caso de uso de redacción, que estaba en el diagrama anterior.

Algunas de las historias a las que responde este diagrama son las siguientes:

*“El usuario autenticado abre la bandeja de recibidos.”*

*“El usuario autenticado abre, descifra y lee un correo que acaba de recibir.”*

*“El usuario autenticado reenvía un email a otra persona.”*

Todo ello queda expresado en el siguiente diagrama de casos de uso.

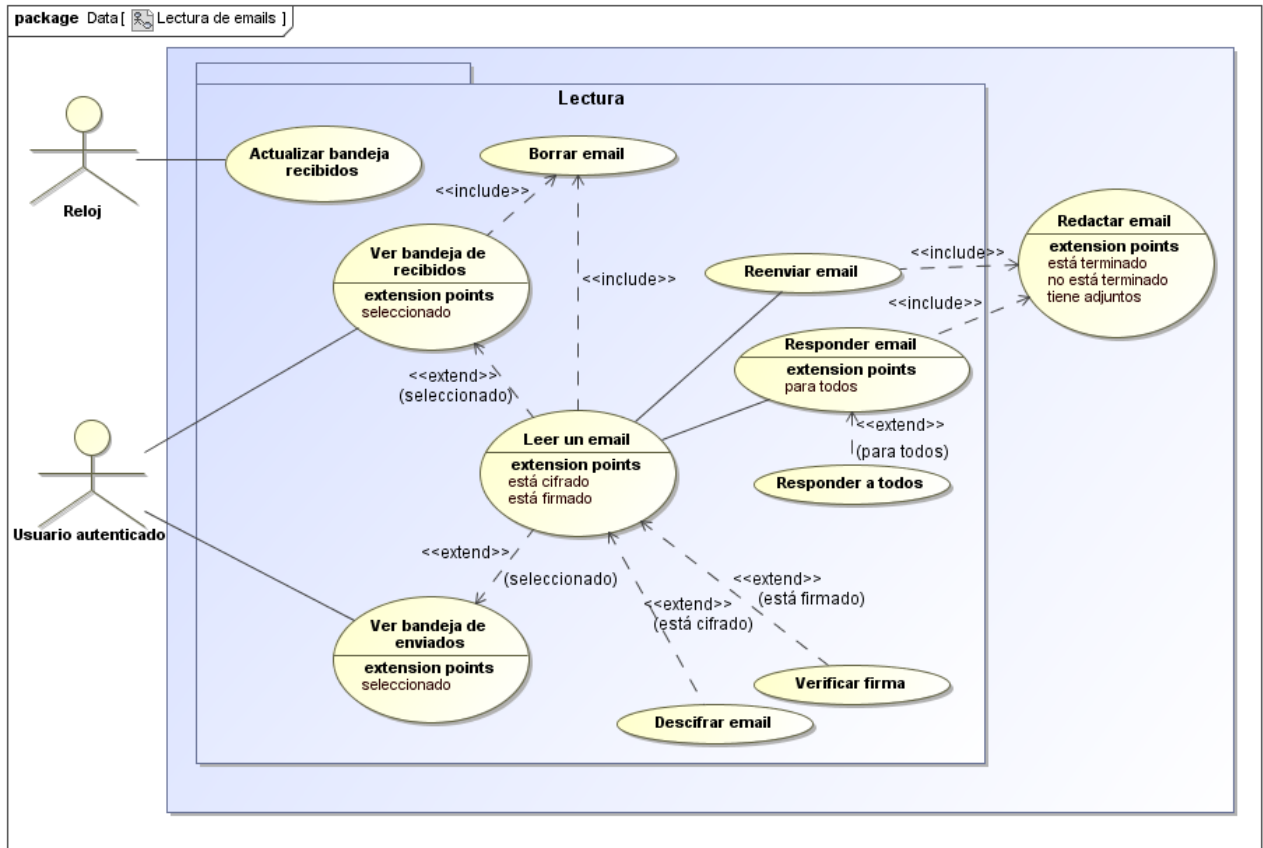


Ilustración 6: Casos de uso de Lectura

#### 4. Modelo conceptual del sistema

El modelo conceptual gira en torno a dos entidades principales: *Email* y *Usuario*.

Cada *Usuario* tiene tres *Bandejas*: una de tipo “recibidos”, otra de tipo “enviados” y finalmente una de tipo “borradores”.

Cada *Usuario* puede tener “n” *CuentaDeCorreos* configuradas (como en los clientes de correo de escritorio).

Cada *CuentaDeCorreo* se compone de una configuración de correo saliente, *SMTPServer*, entrante, *POPServer*, y una dirección de correo, *EmailAddr*, con su usuario y contraseña, a la que podemos haber asociado un par de claves criptográficas PGP, *ClavePGP*.

Cualquier dirección de email, *EmailAddr*, puede tener claves públicas, que también se recogen en *ClavePGP*.

Un *Email* tiene un *EmailAddr* en el campo “de”, “n” *EmailAddren* el campo “para”, “n” *EmailAddren* el campo “copia”, “n” *EmailAddren* el campo “copia oculta”. También puede tener cualquier número de *FicheroAdjuntos* (aunque recordemos por diseño hemos limitado el tamaño total de un correo electrónico a 2MB).

Y como ya hemos visto, los *Emails* están agrupados en alguna de las tres bandejas de algún *Usuario*.

Con el código real del proyecto, Visual Studio nos ha generado el diagrama que vemos en la Ilustración 7, que finalmente ha resultado más plano y simplificado que el de nuestro análisis original:

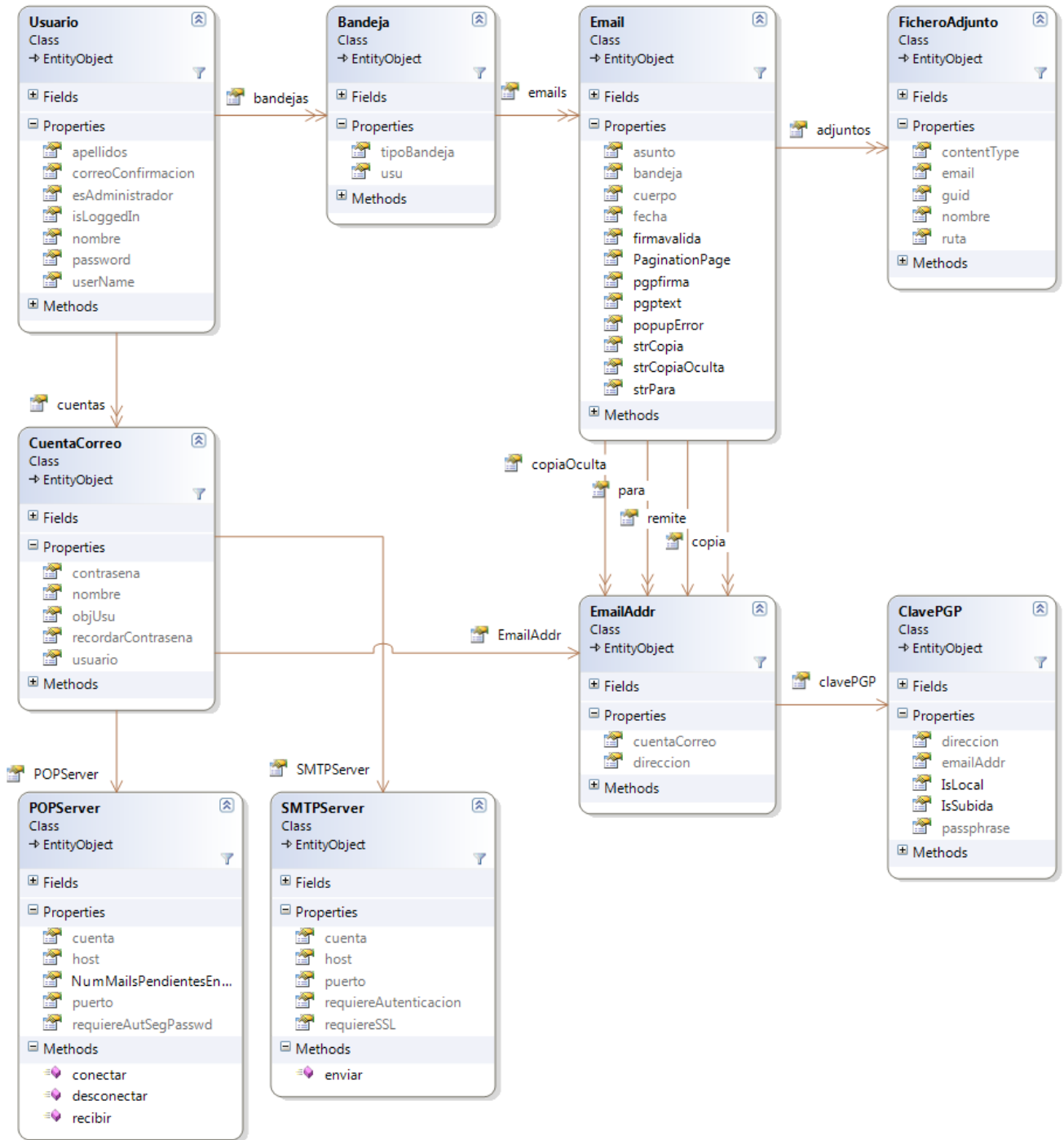


Ilustración 7: Modelo conceptual del sistema

## VII. Diseño

Dentro de la fase de diseño se han evaluado los siguientes puntos:

- Arquitectura Lógica del Sistema: patrón MVC y en tres capas
- Arquitectura Física del Sistema: cliente-servidor
- Arquitectura de Red propuesta para el despliegue
- Diagramas de Secuencia para los Casos de Uso
- Diagrama de Clases de Diseño sobre el que poder iniciar directamente el desarrollo
- Modelo relacional de la estructura para la Base de Datos
- Diseño de la Interfaz Gráfica.

## 1. Arquitectura del sistema

---

El objetivo principal de este punto es ofrecer una visión simplificada del sistema, de forma que se pueda obtener una visión global del mismo.

Hemos generado cuatro diagramas de arquitectura en función del punto de vista utilizado para generarlos:

- Diagrama lógico MVC: Esa es la estructura lógica que guía nuestro desarrollo: Los módulos “Modelos”, “Vistas” y “Controladores”. El código está estructurado en capas que siguiendo este patrón de arquitectura quedan desacopladas, lo cual permite la distribución y especialización del trabajo de desarrollo y facilita el posterior mantenimiento y reutilización de componentes.
- Diagrama lógico en tres capas: La clásica distribución de componentes en capa de presentación, de lógica y de datos. Puesto que en nuestra estructura MVC el Modelo no se comunica directamente con la Vista (es MVC “pasivo”), los componentes pueden reagruparse también para componer una *tree-tier architecture*.
- Diagrama Físico: Donde se ve cómo se distribuyen los distintos módulos de software dentro de cada una de las máquinas, cada módulo brindando una funcionalidad e interrelacionándose con otros para alcanzar una funcionalidad conjunta que cubre las necesidades y requisitos analizados.
- Arquitectura de red: Una visión gráfica de una propuesta de instalación posible dentro del entorno de red del cliente.

### 1.1. Arquitectura lógica MVC

---

Hemos optado por el patrón de arquitectura software llamado MVC (Modelo-Vista-Controlador).

Se trata de una arquitectura orientada a desacoplar la interfaz de usuario (de entrada y presentación) aislándola de la lógica de negocio de la aplicación, permitiendo así que cada una sea desarrollada, probada y mantenida de forma independiente<sup>3</sup>.

En aplicaciones web tiene especial sentido optar por este modelo, puesto que lo natural de la presentación de estas aplicaciones es estar muy desacoplada del modelo, hasta el punto de que la presentación se renderiza en otra máquina distinta de aquella en la que corre la aplicación web, y dentro de otro programa, el navegador. Lo ideal es entonces intentar mantener su capa de presentación lo más ligera (para no sobrecargar cliente) y lo más desacoplada (para no sobrecargar red) que sea posible. MVC es ideal para alcanzar estos objetivos.

Tanto es así, que el propio Microsoft ha creado plantillas específicas para la creación de aplicaciones web que sigan este modelo mediante ASP.NET, de modo que, junto con ASP.NET *WebForms*, que era la plantilla original de desarrollo de aplicaciones ASP.NET, a partir de marzo de 2009 lanza “ASP.NET MVC”<sup>4</sup> para su entorno de desarrollo *Visual Studio*, así como los nuevos espacios de nombres *System.Web.Mvc* en la biblioteca de clases .NET desde su versión 4.

---

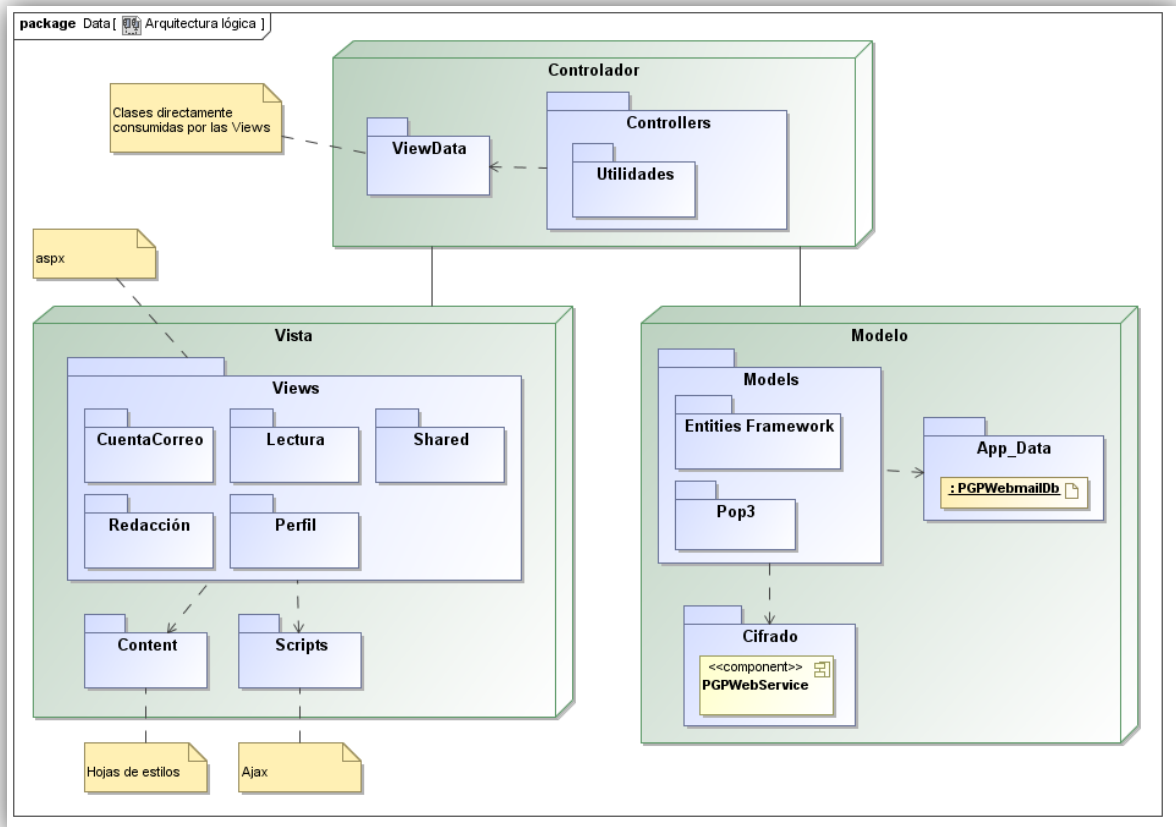
<sup>3</sup> Wikipedia (2011, abril). “Model–view–controller” [artículo en línea]. [Fecha de consulta: 3 de abril de 2011].

<http://en.wikipedia.org/wiki/Model-view-controller>

<sup>4</sup> Wikipedia (2011, marzo). “ASP.NET MVC Framework” [artículo en línea]. [Fecha de consulta: 3 de abril de 2011].

[http://en.wikipedia.org/wiki/Asp.net\\_mvc](http://en.wikipedia.org/wiki/Asp.net_mvc)





Ilustraci3n 8: Arquitectura l3gica MVC

Vemos en el esquema del diagrama que dentro de la “Vista” tenemos varios paquetes de c3digo: Las vistas ASPX propiamente dichas; los *scripts* necesarios para darles funcionalidad *Ajax*, JS, imprescindible para efectos y transiciones suaves en el navegador (al evitar innecesarias recargas completas de la p3gina); y hojas de estilos, CSS, que determinar3n propiedades gr3ficas de los elementos mostrados.

Dentro de “Controlador” encontramos el paquete de los *Controllers* propiamente dicho, pero tambi3n hemos incluido las clases *ViewData*: son clases espec3ficamente preparadas por los controladores para ser directamente consumidas por las vistas, que las renderizan; son el *output* de los controladores cuando no basta con “alimentar” a la vista directamente con una entidad del modelo.

Finalmente dentro del “Modelo” encontramos todas las clases que representan a las entidades que modelizan el negocio as3 como los paquetes relativos a su persistencia (base de datos); y en paquete aparte hemos incluido el m3dulo de criptograf3a PGP, ya que f3sicamente lo hemos sacado a un *Web Service*.

Siguiendo la definici3n de Ramiro Lago<sup>5</sup>, el Modelo contiene los datos y las reglas de negocio. La Vista muestra la informaci3n al usuario y le brinda interfaz para interactuar: botones, links, cajas de texto, etc. Y en el Controlador “saltan” (*routing*) funciones a las que llamamos “Actions” ante las entradas del usuario (pinchar en un link o pulsar un bot3n, etc.) y, aplicando las reglas de gesti3n de esos eventos, interactuando si es necesario con el Modelo, genera una salida (el *return* de esas Actions) que es el nuevo paquete de informaci3n que renderizar3 la Vista.

<sup>5</sup> Lago, Ramiro (2007, abril). “Patr3n Modelo-Vista-Controlador”. *Patrones de dise1o software* [art3culo en l3nea]. [Fecha de consulta: 3 de abril de 2011].

<http://www.proactiva-calidad.com/java/patrones/mvc.html>

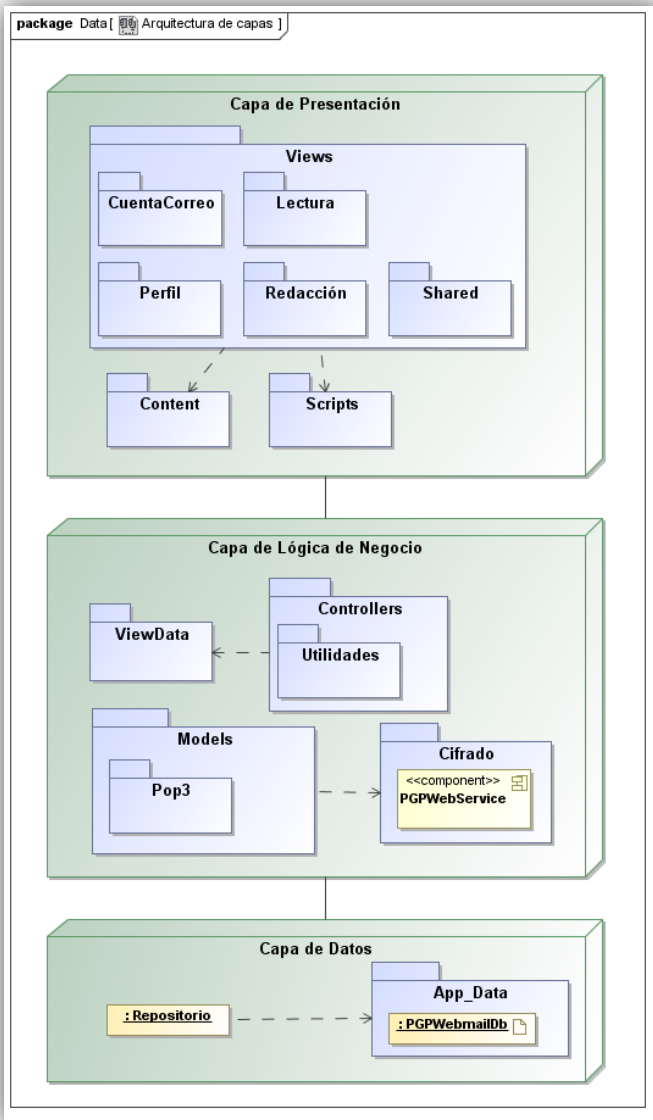


Ilustración 9: Arquitectura lógica en tres capas

## 1.2. Arquitectura lógica en tres capas

La forma clásica de distribuir componentes para que estén desacoplados es la llamada arquitectura en tres capas: Presentación, Lógica y de Datos.

En realidad la verdadera estructura de desacoplamiento de nuestro proyecto es la referida más arriba: MVC.

Sin embargo, puesto que nuestro modelo es un MVC “pasivo”, en el que el Modelo no notifica nada a la Vista, las relaciones entre los tres bloques se hacen lineales (y no triangulares), y por lo tanto podríamos reformular la arquitectura en los términos de Presentación, Lógica y Datos, según vemos en el diagrama de la Ilustración 9.

## 1.3. Arquitectura física: Cliente-Servidor

Nuestra solución sigue una arquitectura cliente-servidor: Al tratarse de una aplicación web, la aplicación reside y se ejecuta en una máquina servidor, pero su interfaz de usuario se visualiza en otra máquina, desde la que el usuario interactúa con la aplicación. Entre ambas máquinas hay una relación de cliente-servidor, en la que la máquina cliente inicia una conexión a nivel de transporte TCP hacia la máquina servidor, y una vez aceptada y establecida, sobre ella, HTTP a nivel aplicación.

A su vez, dentro del nodo servidor, nuestra solución requiere de algunas otras relaciones cliente-servidor para, fruto de todas ellas, terminar prestando el servicio necesario para la satisfacción de las necesidades y requisitos:

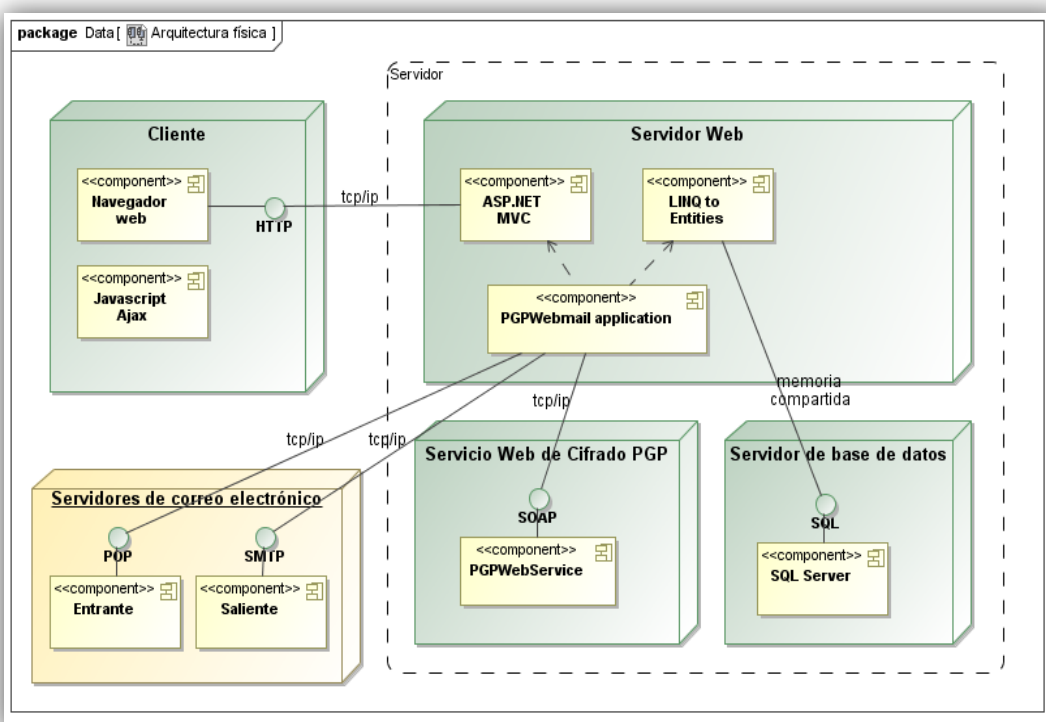


Ilustración 10: Diagrama de arquitectura física

- Nuestro servidor *PGPWebmail* es cliente del servicio de gestión de base de datos SQL Server, sobre memoria compartida o canalizaciones con nombre, comandos SQL.
- Nuestro servidor *PGPWebmail* es cliente de los servidores de correo electrónico que los usuarios configuren en sus cuentas, que pueden ser múltiples. Sobre transporte TCP, aplicación POP y SMTP.
- Finalmente, dentro del núcleo de nuestra aplicación web hay también otra estructura cliente-servidor, ya que hemos sacado la funcionalidad de criptografía PGP a un módulo software aparte, un *Web Service WCF*, que hace de servidor hacia el que el módulo principal se conecta. Sobre transporte TCP, aplicación SOAP.

#### 1.4. Arquitectura de red: Intranet

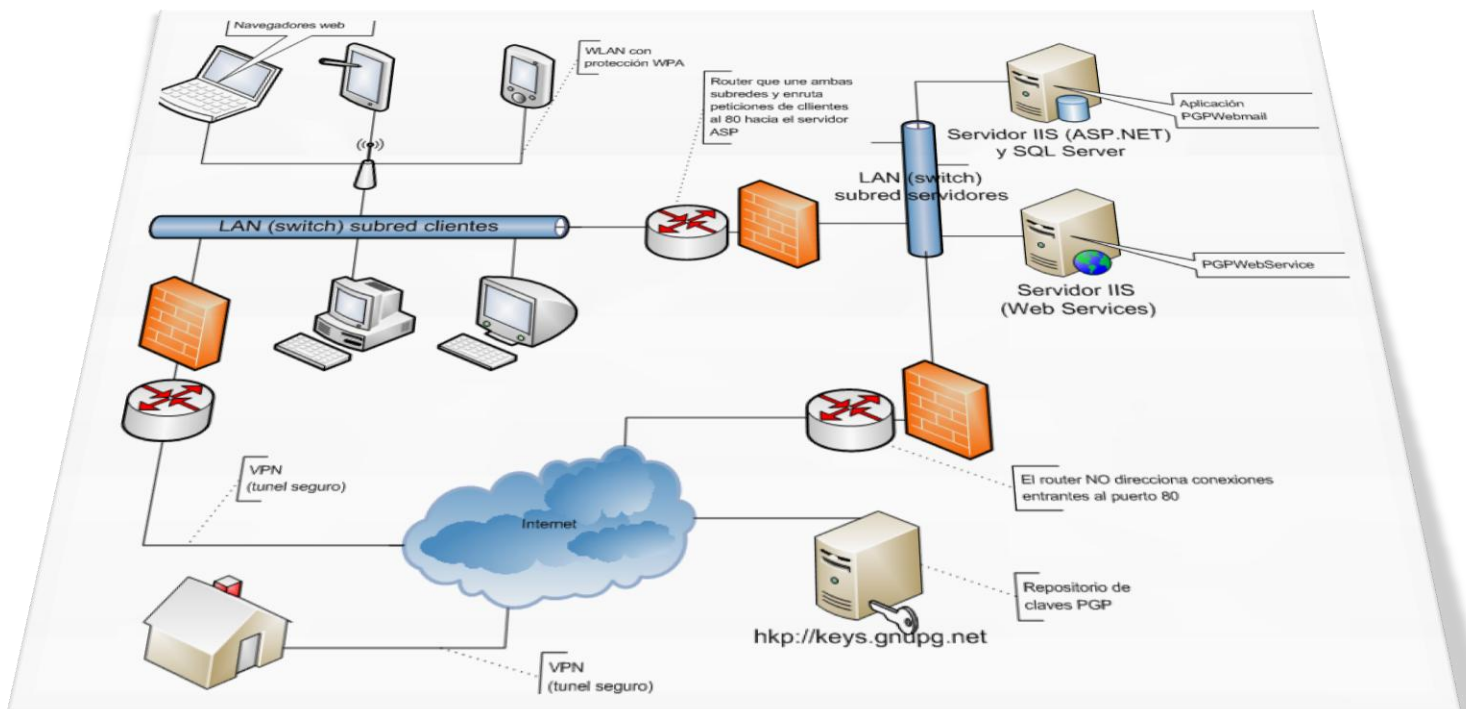


Ilustración 11: Diagrama de arquitectura de red

El objetivo es dar acceso a la aplicación a los dispositivos de una organización, nuestro cliente. Son dispositivos conectados a una LAN, bien directamente bien a través de VPN. Los dispositivos inalámbricos tienen una conexión WiFi protegida con cualquiera de las variantes de WPA. Podrán ser dispositivos cualesquiera (ordenadores, tabletas, smart-phones, etc.), con sistemas operativos cualesquiera (iOS, Mac OS, Windows, Linux, Android, etc.) con la única condición de que puedan conectarse a una LAN TCP/IP y que puedan correr un navegador web.

Nuestro planteamiento, de entre los varios posibles para alcanzar este objetivo, es configurar una LAN separada de la LAN de clientes de la organización, que contendrá los servidores que intervengan en nuestra solución. Ambas estarán interconectadas a través de un encaminador (*router*) que direccionará las llamadas al puerto 80 hacia el servidor en el que se encuentre alojado el IIS sirviendo nuestra aplicación ASP.NET.

Es importante que la LAN cableada en la subred de los clientes esté constituida mediante un conmutador (*switch*), y no mediante un concentrador (*hub*), para evitar que las máquinas puedan acceder de forma promiscua al tráfico de paquetes que corresponde a otras máquinas de la misma LAN.

En el diagrama proponemos, aunque no es imprescindible, que nuestro módulo *PGPWebmail* se encuentre en el mismo servidor que el SQL Server, con comunicación mediante protocolo de *memoria compartida*. Y que nuestro módulo *PGPWebService* se aloje en otro servidor aparte, dentro de esta subred de servidores protegida<sup>6</sup>.

Esta subred de servidores NO es una DMZ, puesto que el encaminador que la comunica con Internet NO permite el acceso desde el exterior. Como vimos en el análisis de requisitos, el despliegue que nos pide nuestro cliente es como aplicación de *intranet* para su organización; por tanto, pudiendo evitarlo, no nos interesa exponerla al acceso público.

El único acceso posible desde el exterior será ingresando en la LAN *de clientes* a través de VPN.

Sin embargo los servidores sí tendrán acceso a Internet mediante tráfico separado de la subred de clientes, para poder subir y bajar claves públicas PGP necesarias para las operaciones criptográficas que realizará nuestro módulo *PGPWebService*; éstas se guardan en el repositorio público `hkp://keys.gnupg.net`.

## 2. Diagramas de secuencia

---

Los diagramas de secuencia que ilustran los casos de uso y nos ayudan a descubrir clases de diseño y los métodos que debe incluir cada clase, los habíamos diseñado en abstracto mediante la herramienta CASE *MagicDraw UML* y aportado en el documento de *Análisis y Diseño*.

Pero ahora que ya hemos codificado, podemos generar estos diagramas fácilmente desde el propio código a través de Visual Studio; -lo cual es una gran herramienta para refactorizar-.

Como pueden autogenerarse desde el código, y el código está abierto y entregado, no voy a incluir a continuación todos los diagramas de secuencia posibles por mor de la brevedad, sino sólo alguno representativo de cada *Controller* a modo ilustrativo.

### 2.1. Acceso y registro de usuarios: *PerfilController*

---

Los diagramas de secuencia sobre configuración de cuentas tienen como controlador a la clase *PerfilController*. Que deriva de *ControladorBase*, -la que le brinda funciones auxiliares comunes para hacer seguimiento de usuarios "loggedon"-, que a su vez deriva de la clase *System.Web.Mvc.Controller*, que le presta toda la funcionalidad genérica como controlador desde la biblioteca de clases .NET.

Usamos la clase *ServicioDeMiembros* (dudé si llamarla *ServicioDeAfiliacion*, *ServicioDePertenencia* o *ProveedorDeMembresía*, que es una palabra del español centroamericano), que encapsula toda la funcionalidad relativa a determinar la pertenencia al sistema de un determinado usuario (user + password). Hereda de *System.Web.Security.MembershipProvider*, rescribiendo sus métodos para que utilice nuestra base de datos, *PGPWebmailDb*, y la tabla *UsuarioSet* como forma de determinar la validez de un usuario.

Hacemos también uso de la clase *AutenticacionPorFormulario*, que usará a su vez de los métodos estáticos que le brinda la clase abstracta *System.Web.Security.FormsAuthentication*, también de la biblioteca .NET, para dejar mediante *cookies* autenticada la sesión del usuario mientras esta dure.

---

<sup>6</sup>Este despliegue "en doble servidor" requiere modificar, una vez terminadas las instalaciones, el `web.config` de *PGPWebmail*, para cambiar la IP que indica el endpoint "producción", dentro de `system.serviceModel.client`, que por defecto es 127.0.0.1, y que apunte a la IP del servidor donde hayamos instalado *PGPWebService*.

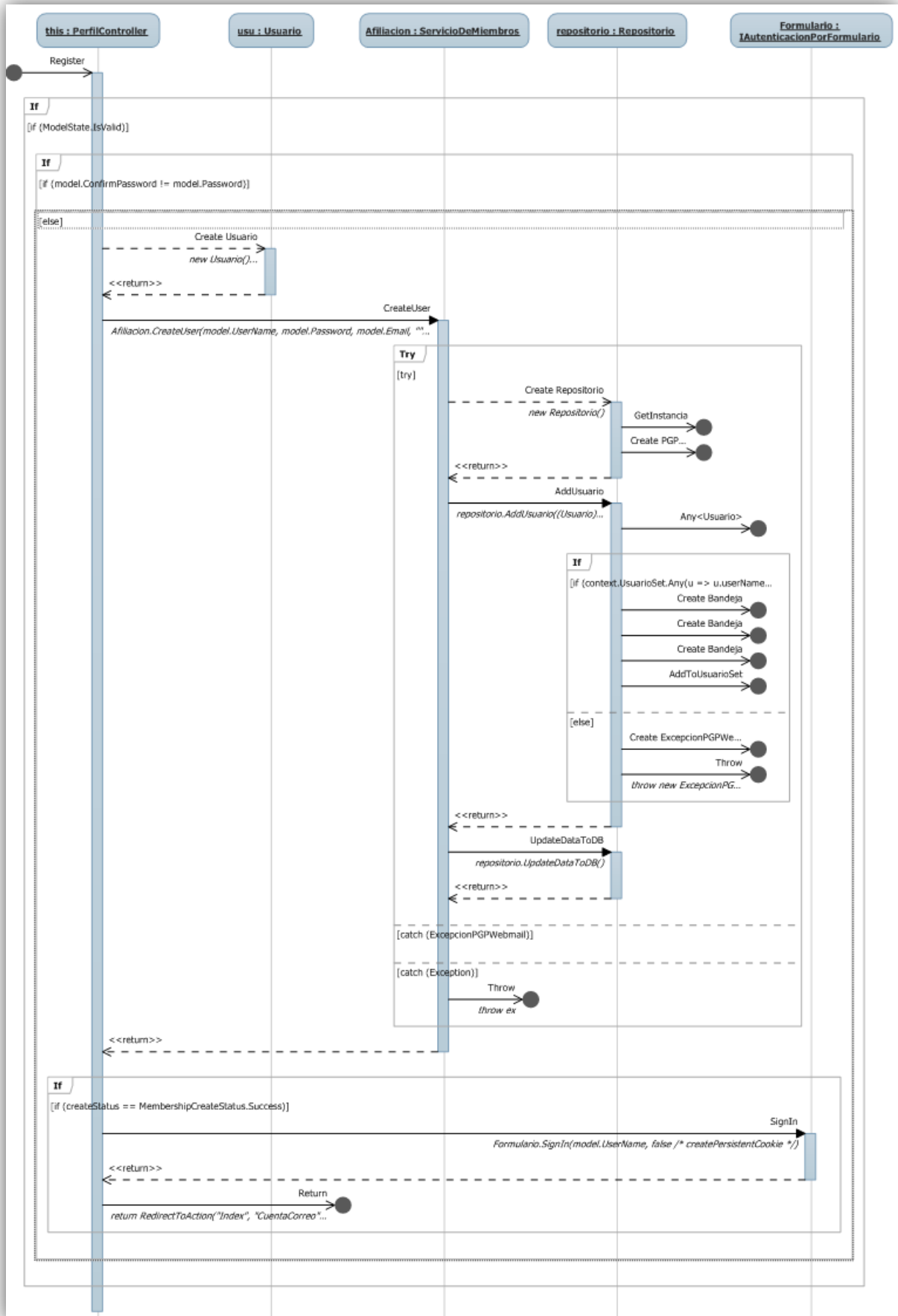


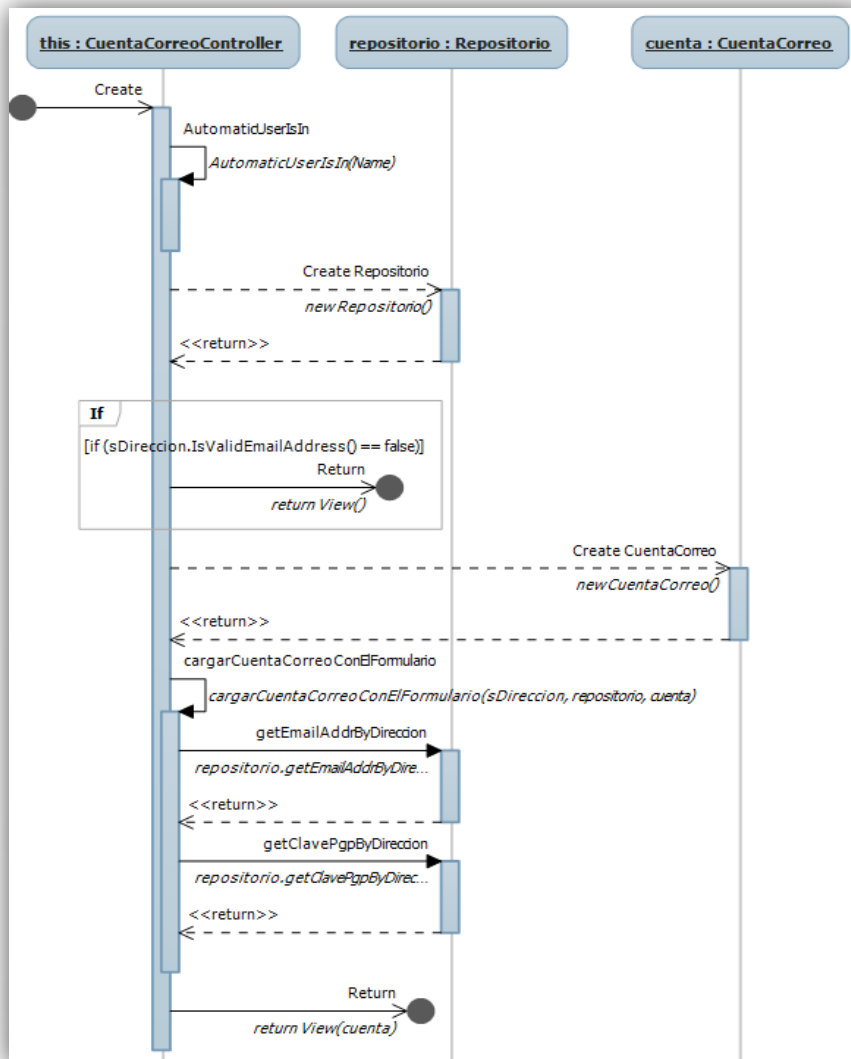
Ilustración 12: Diagrama de secuencia de Registro de nuevo usuario

## 2.2. Configuración de cuentas de correo: *CuentaCorreoController*

Interviene un nuevo controlador, *CuentaCorreoController*, y las clases *CuentaDeCorreo*, *EmailAddr* y *ProxyCifrado*, que encapsula nuestra relación con *PGPWebService*, que nos sirve funcionalidades de cifrado.

*CuentaDeCorreo* se cargará con los servidores POP y SMTP (se crearán sus clases correspondientes) configurados en el formulario web, cuya información nos llegará a través del parámetro *FormCollection* del método post correspondiente.

Configurada una *CuentaDeCorreo*, tendremos un *EmailAddr*, para el que podremos solicitar un par de claves PGP pública/privada, y a continuación subir la pública al servidor de claves.



Dentro de “cargaCuentaCorreo ConElFormulario”, MVC, mediante la función *UpdateModel* heredada de *Controller*, instancia y carga de contenido las clases asociadas a *CuentaCorreo*, como *POPServer*, *SMTPServer* o *ClavesPGP*.

Ilustración 13: Diagrama de secuencia de asociación de nueva cuenta de correo

## 2.3. Redacción de correos electrónicos: *RedaccionController*

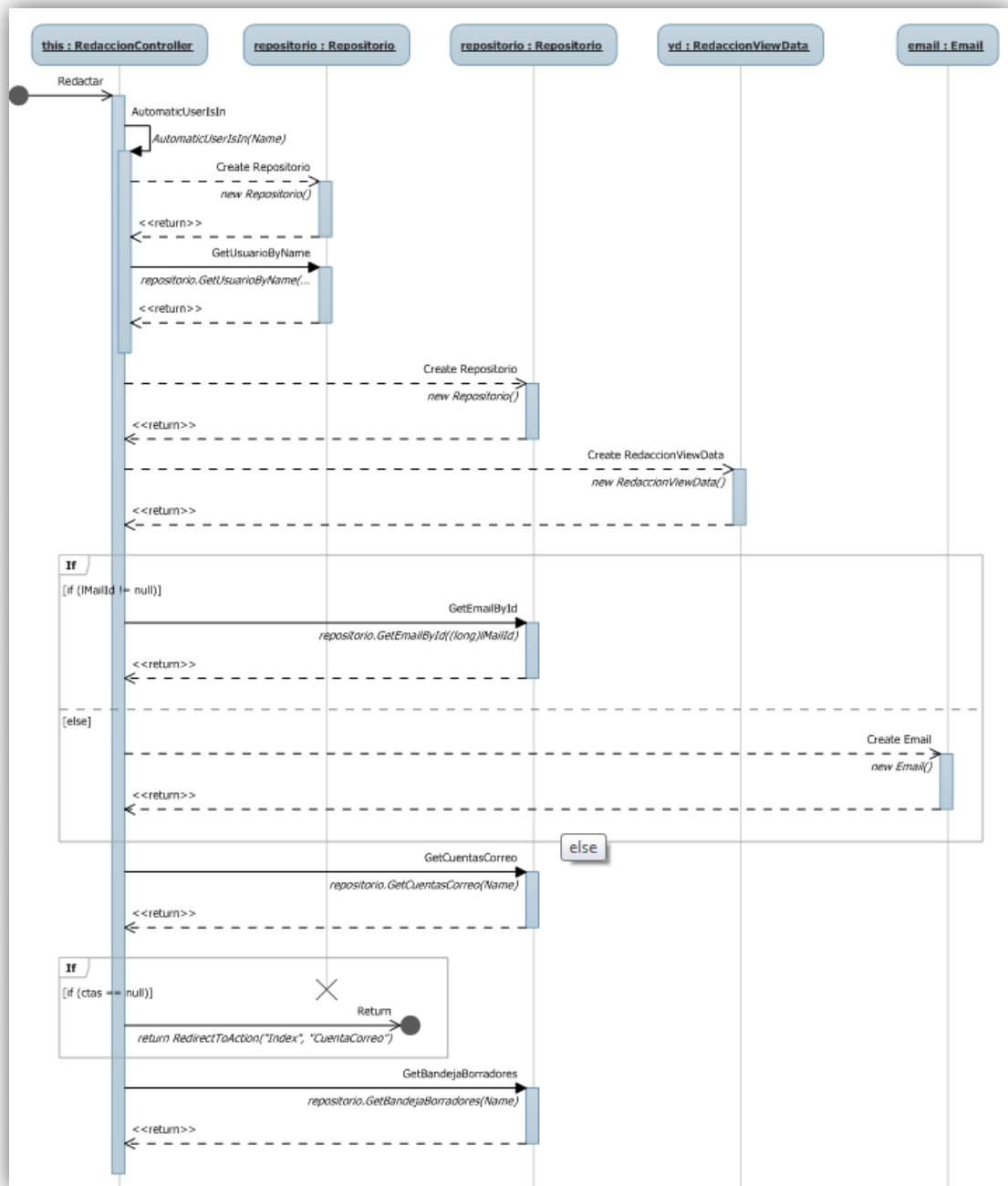
En los casos de uso relativos a la redacción de correos electrónicos el controlador es *RedaccionController*.

Naturalmente todo gira en torno a la clase *Email*.

Interviene la colección *Bandeja*, de tipo “borradores” y de tipo “enviados”; *SMTPServer* para el envío; y *ClavePGP* para el cifrado y firma.

Para mostrar listas de emails “paginadas” (puesto que no deben aparecer más de 50 emails por pantalla, según los requisitos NF-18, R-01, R-03 y W-04) es necesario crear una clase del paquete *ViewData*, *ListaCorreosViewData*, para ser directamente consumida por la vista (el

listado paginado de emails), que contendrá un subconjunto –según la página visualizada- de hasta 50 de los mails de la bandeja, así como referencia a cuál es el número de página anterior y siguiente.



**Ilustración 14: Diagrama de secuencia de redacción de email**

Hay que tener en cuenta que la *Action* de redacción de email no sólo entra en acción cuando clicamos sobre “redactar nuevo mail”, sino también cuando retomamos un email que ya estaba siendo redactado, como un borrador, o el mail autogenerado que se produce cuando se responde o cuando se reenvía otro email.

Por esta razón vemos que el diagrama de secuencia, aunque refleja una *Action* tipo “get”, contempla la posibilidad de que se le pase un *EmailId* con el que haya que recuperar de la base de datos un objeto *Email* ya existente.

En caso contrario, se crea ahora.



También observamos que si el usuario no tiene cuentas asociadas, se le redirecciona a la lista de cuentas, para que pueda ver que está vacía y entender (leer) que tiene que configurar al menos una.

### 2.4. Lectura de correos electrónicos: *LecturaController*

En los casos de uso relativos a la lectura de correos electrónicos el controlador es *LecturaController*.

Vuelve a intervenir la colección *Bandeja*, esta vez de tipo “enviados” y “recibidos”. *ClavePGP* para verificar firmas y descifrar los correos cifrados recibidos. Y de nuevo para mostrar listas de emails “paginadas” usaremos *ListaCorreosViewData*.

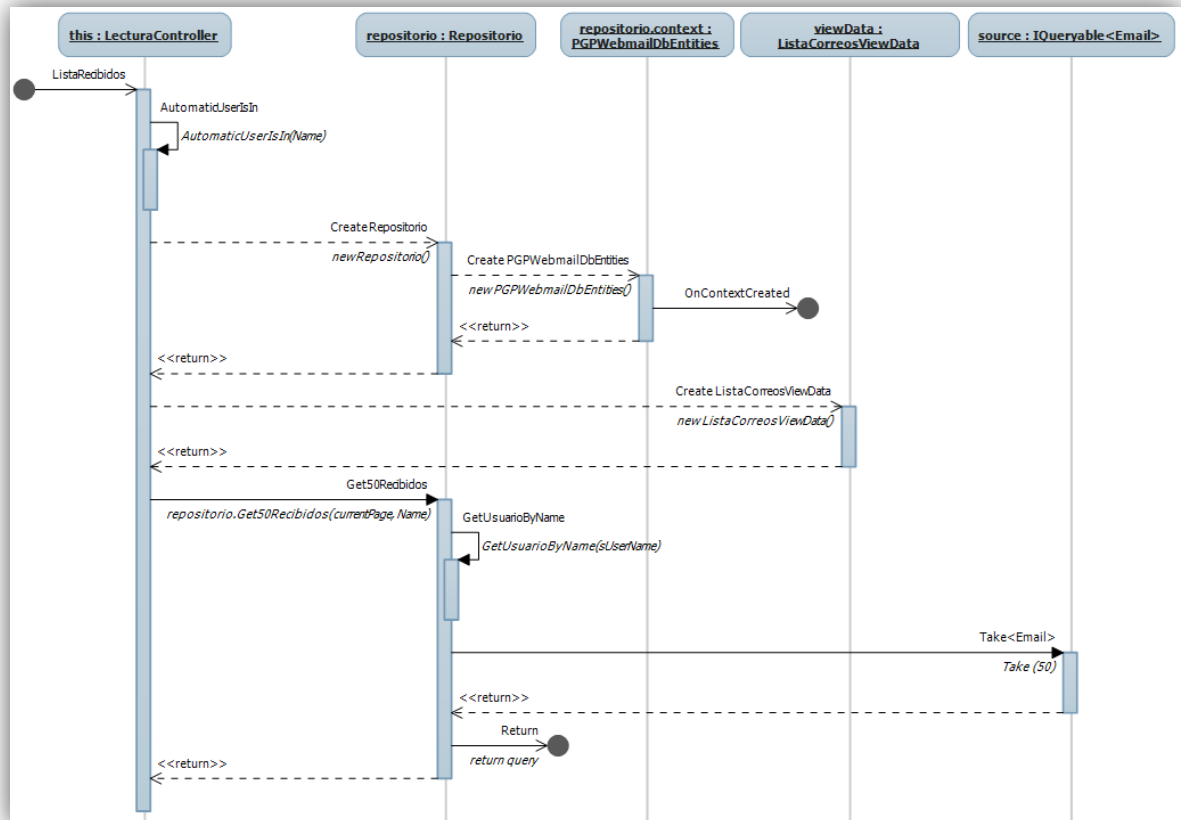


Ilustración 15: Diagrama de secuencia de carga de lista de recibidos

## 3. Diagrama de Clases de Diseño

Mediante Visual Studio podemos diseñar el diagrama de clases partiendo del código real. De este modo estamos completamente seguros de que el código coincide con el diseño que lo documenta.

Si lo comparamos con el diagrama del documento *Análisis y Diseño*, diseñado mediante la herramienta CASE *MagicDraw UML*, vemos que el resultado final del código real no dista mucho del diseño inicial del que partimos:

Aparte de cierta simplificación de las clases del Modelo, quizás lo más reseñable sea que *Repositorio* ya no es una propiedad de *ControladorBase*, como diseñamos inicialmente, sino que crearemos instancias del mismo cada vez que lo necesitamos. Y ello por el comportamiento que hemos descubierto que tienen los *ObjectContext* del *Entity Framework*, de lo que hablaremos más adelante en el epígrafe sobre implementación *El Repositorio*, que está en la página 42.



## Diagrama de Clases de Diseño

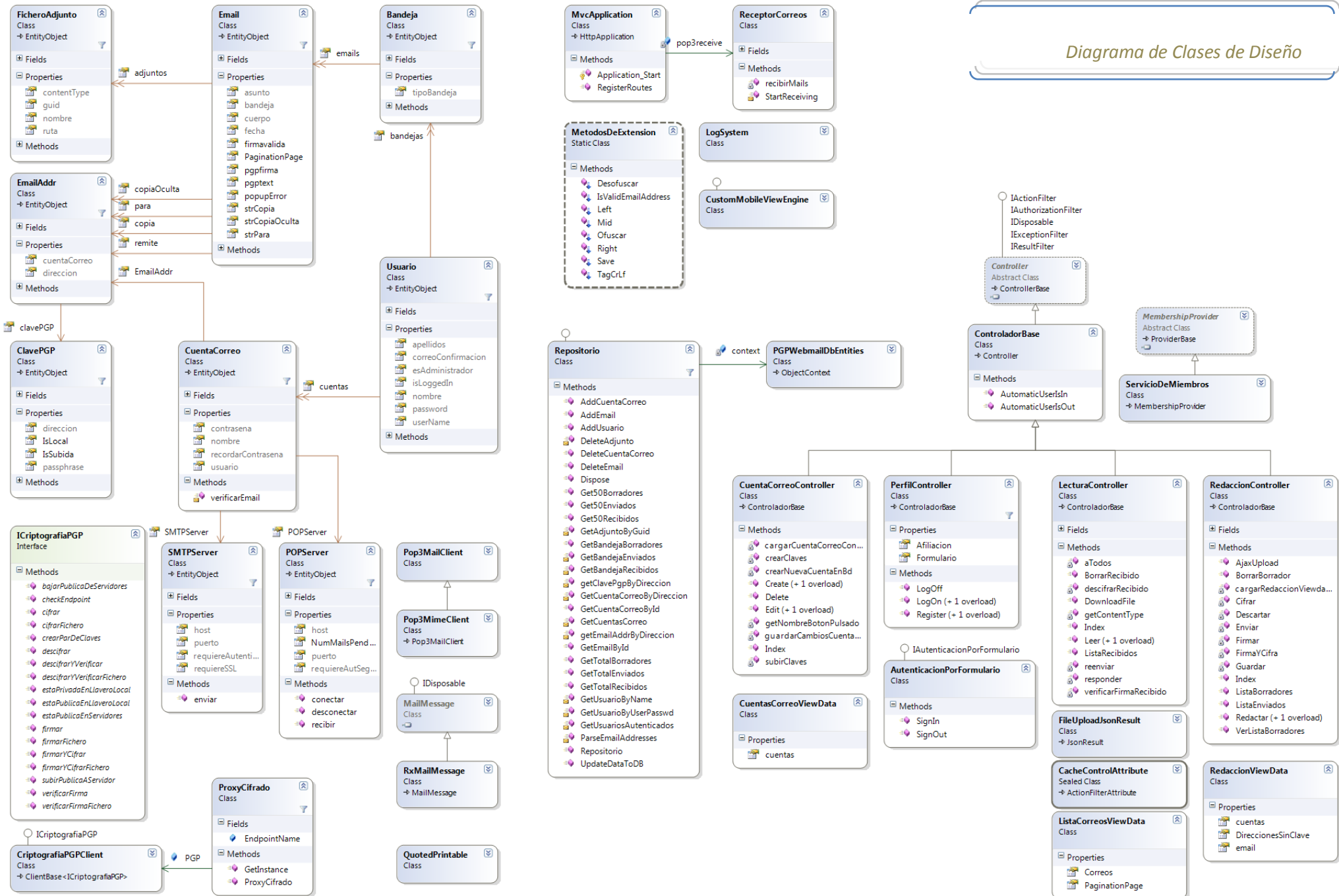


Ilustración 16: Diagrama de Clases de Diseño

#### 4. Diseño de la Base de Datos. Modelo Relacional

En fase de diseño llegamos a un Modelo Relacional. Y fue precisamente por ahí por donde comencé la implementación: Creé un proyecto MVC 2 con Visual Studio 2010, le añadí como nuevo ítem una *SQL Server Database* en la que creé la estructura relacional de tablas que había diseñado. A continuación añadí al proyecto un *ADO.NET Entity Data Model*, señalándole la base de datos desde la que debía generar las entidades. Lo que obtuve es la estructura que puede verse más adelante, en la página 39, Ilustración 26.

Me sorprendió que en esta estructura de entidades EF desaparecían las tablas cuyo único objetivo era sustentar relaciones n:m. Aunque entendí su lógica.

Y a continuación, para ver si el sistema era consistente, lo que hice fue pedirle a Visual Studio que me generase la base de datos partiendo de las entidades EF que acababa de obtener.

El resultado fue muy parecido a la estructura original de partida, lógicamente, si bien le añadió a los nombres de las tablas el sufijo "Set" y los nombres de las claves ajenas los reformuló a su manera.

La estructura de base de datos resultante es la que finalmente quedó y que recojo en la Ilustración 17.

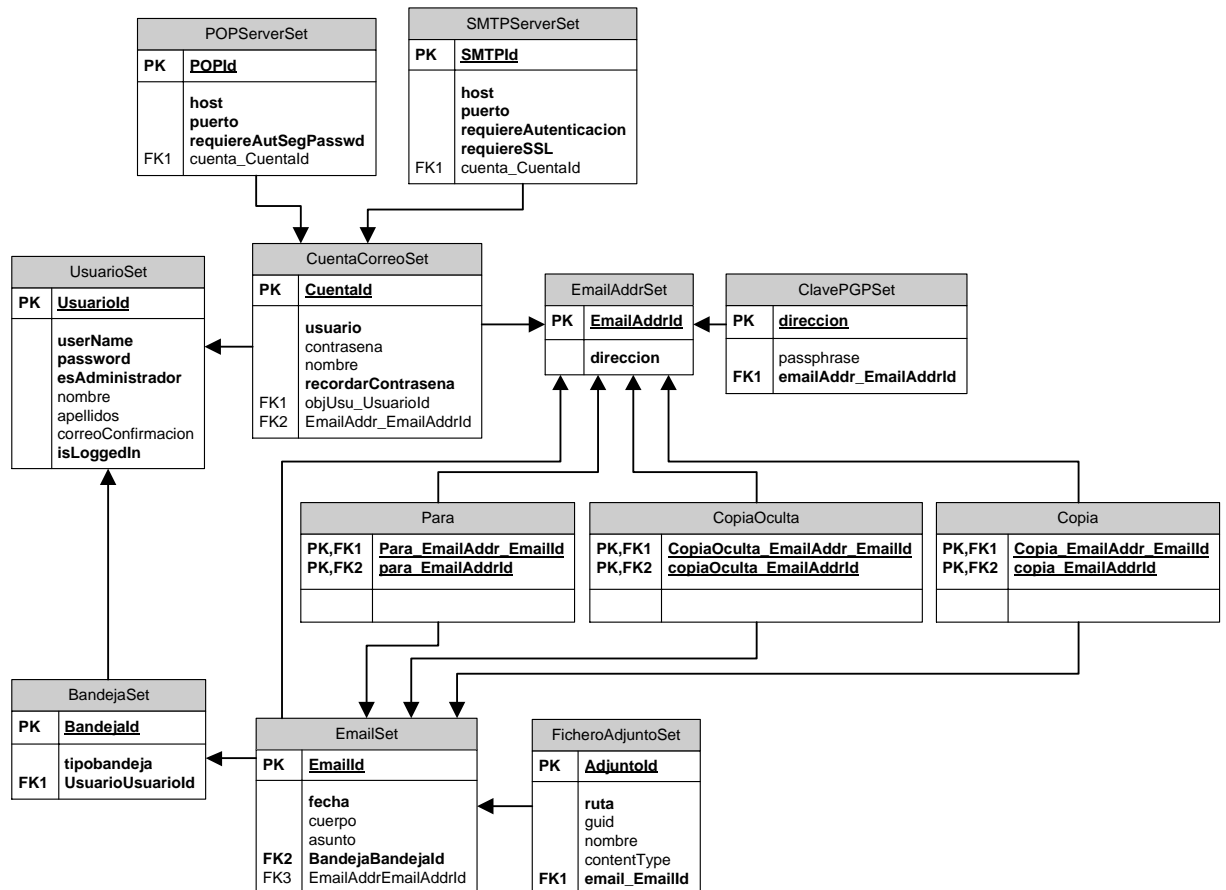


Ilustración 17: Estructura de la base de datos: Modelo relacional

La estructura es muy similar al diseño original (el anterior a la implementación), cambiando algunos nombres de campo, de tabla, y añadiendo algunos campos cuya necesidad descubrí durante la implementación (como un "guid" o un "content-type" para los adjuntos).

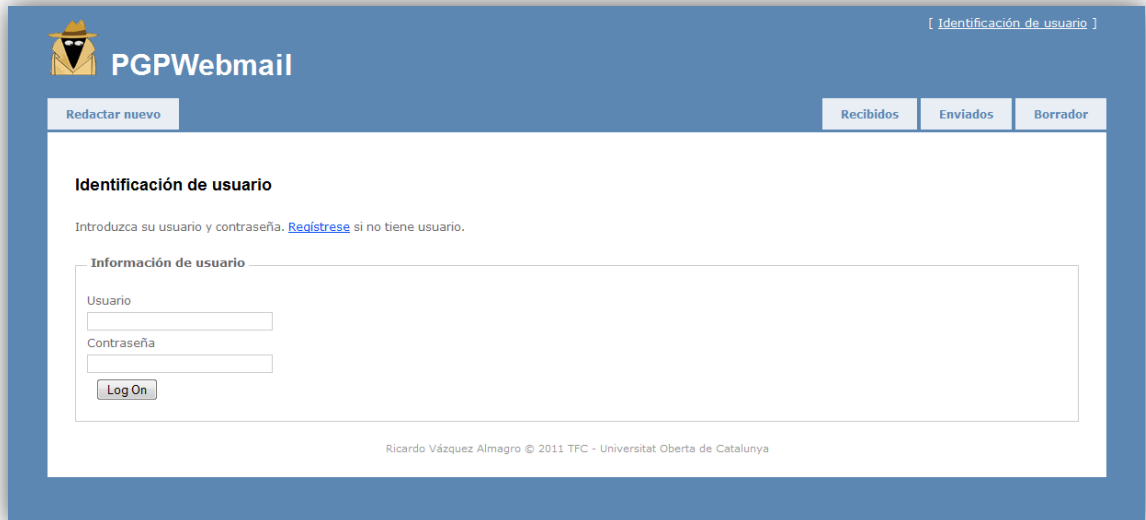
Como podemos observar, partiendo de la tabla de usuarios, cada usuario puede tener  $n$  bandejas (si bien deben restringirse a 3 por usuario: borradores, recibidos y enviados). Cada

bandeja podrá tener un número indeterminado de emails. Cada email tendrá una dirección remitente y  $n$  direcciones destinatarias ( $n$  en su campo para,  $n$  en su campo copia y  $n$  en su campo copia-oculta). Cada dirección puede estar asociada a una clave PGP (sólo a pública si tiene el campo *passphrase* a nulo, o a pública/privada si tiene dicho campo relleno).

Cada usuario tiene además una o más cuentas de correo, compuestas por la configuración de su servidor POP, la de su servidor SMTP y la dirección de correo asociada (una por cuenta).

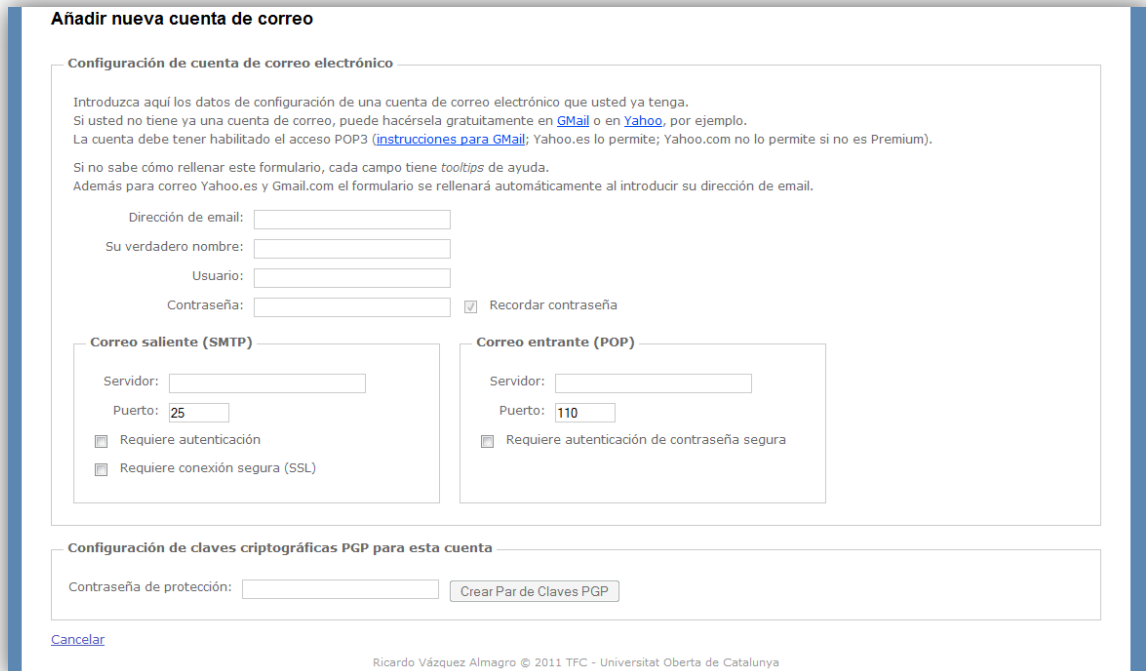
## 5. Interfaz gráfica de usuario

Hemos mantenido la GUI que por defecto genera Visual Studio para proyectos MVC. Reproduzco capturas de algunas de las pantallas principales:



The screenshot shows the 'Identificación de usuario' (User Identification) page of PGPWebmail. The page has a blue header with the PGPWebmail logo and a navigation bar with buttons for 'Redactar nuevo', 'Recibidos', 'Enviados', and 'Borrador'. The main content area is titled 'Identificación de usuario' and contains a form with fields for 'Usuario' and 'Contraseña', a 'Log On' button, and a 'Regístrate' link. The footer of the page reads 'Ricardo Vázquez Almagro © 2011 TFC - Universitat Oberta de Catalunya'.

Ilustración 18: GUI - Pantalla de acceso a la aplicación



The screenshot shows the 'Añadir nueva cuenta de correo' (Add new email account) page. The page is titled 'Añadir nueva cuenta de correo' and contains a form for configuring an email account. The form is divided into several sections: 'Configuración de cuenta de correo electrónico' (Email account configuration), 'Correo saliente (SMTP)' (Outgoing mail (SMTP)), 'Correo entrante (POP)' (Incoming mail (POP)), and 'Configuración de claves criptográficas PGP para esta cuenta' (PGP cryptographic keys configuration for this account). The 'Configuración de cuenta de correo electrónico' section includes fields for 'Dirección de email', 'Su verdadero nombre', 'Usuario', and 'Contraseña', along with a 'Recordar contraseña' checkbox. The 'Correo saliente (SMTP)' section includes fields for 'Servidor', 'Puerto' (set to 25), and checkboxes for 'Requiere autenticación' and 'Requiere conexión segura (SSL)'. The 'Correo entrante (POP)' section includes fields for 'Servidor', 'Puerto' (set to 110), and a checkbox for 'Requiere autenticación de contraseña segura'. The 'Configuración de claves criptográficas PGP para esta cuenta' section includes a 'Contraseña de protección' field and a 'Crear Par de Claves PGP' button. The footer of the page reads 'Ricardo Vázquez Almagro © 2011 TFC - Universitat Oberta de Catalunya'.

Ilustración 19: GUI - Asociación y configuración de cuenta de correo

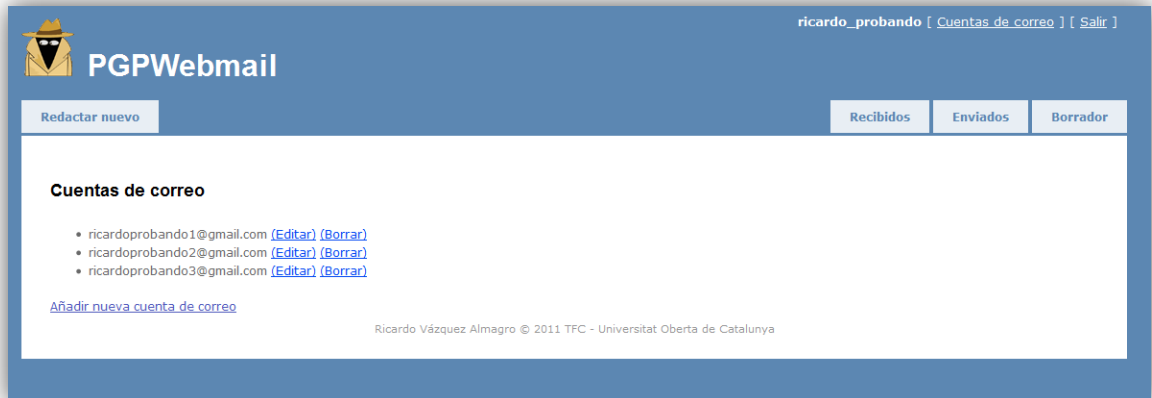


Ilustración 20: GUI - Lista de cuentas de correo asociadas

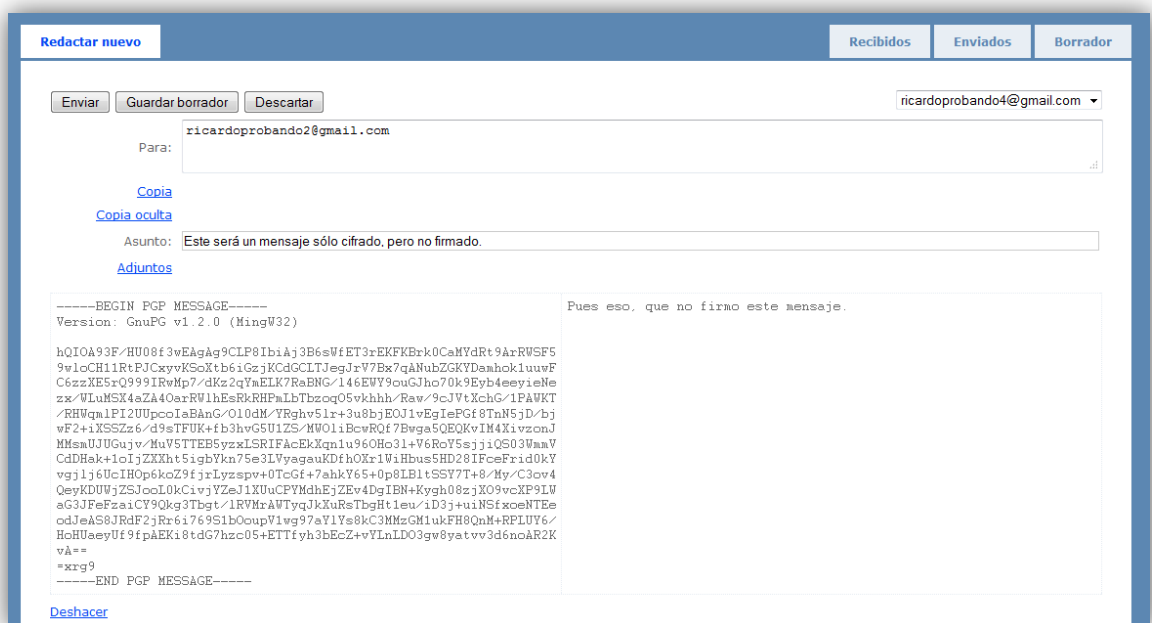


Ilustración 21: GUI - Redacción de un correo



Ilustración 22: GUI - Bandeja de entrada



Ilustración 23: GUI - Lectura de email firmado



Ilustración 24: GUI - Renderizado especial para dispositivo móvil

## VIII. Implementación

### 1. Proyectos incluidos en la Solución

Si abrimos la solución PGPWebmail.sln nos encontramos en la vista *Solution Explorer* que está formada por cinco proyectos:

- CustomInstall
- PGPWebmail
- PGPWebService
- PGPWebmailSetup
- PGPWebServiceSetup

De todos ellos, PGPWebmail es el que aparece marcado como *Startup Project*, es el proyecto central: Es la aplicación web ASP NET MVC cliente de correo electrónico.

Pero toda la funcionalidad de criptografía del cliente de correo está sacada a un Servicio Web WCF: es el proyecto PGPWebService.

A continuación vemos dos proyectos de instalación de aplicaciones, uno para la aplicación web ASP NET MVC y otro para el Web Service WCF: PGPWebmailSetup y PGPWebServiceSetup.

Finalmente el proyecto CustomInstall contiene una Installer Class que permite programar acciones personalizadas de instalación, ya que por defecto las instalaciones de aplicaciones web con Visual Studio no permiten dar permisos de escritura en carpetas de destino. Este proyecto es pues auxiliar de las dos instalaciones PGPWebmailSetup y PGPWebServiceSetup.

### 2. Proyecto PGPWebmail: Modelo

*PGPWebmail* se trata de la aplicación web cliente de correo.

Desplegando el contenido de PGPWebmail vemos las carpetas típicas de una aplicación MVC: Controllers, Models y Views.

Comencé su desarrollo por el modelo: Ya tenía diseñadas las tablas de la base de datos: Reproduciría el diseño en la vista de entidades del Entity Framework (en adelante EF) y ése sería mi modelo inicial.

#### 2.1. Entity Framework

Mi primera sorpresa con el EF fue ver que no tenía que incluir las tablas que en el Modelo Relacional me permitían asociaciones n:m, porque esas tablas ya me las generaba el EF de forma transparente: En el diseño en Visual Studio no se veían, sólo se veían líneas que representaban asociaciones, y sin embargo sí estaban como tablas en la base de datos.

Podemos ver el diseño EF en la Ilustración 26, y las tablas autogeneradas en SQL Server en la Ilustración 25.

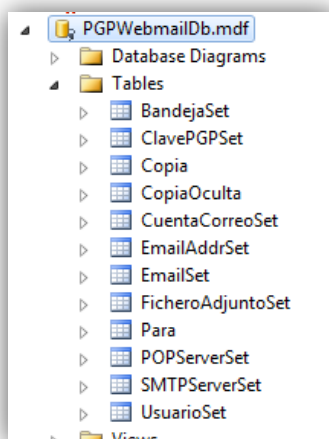
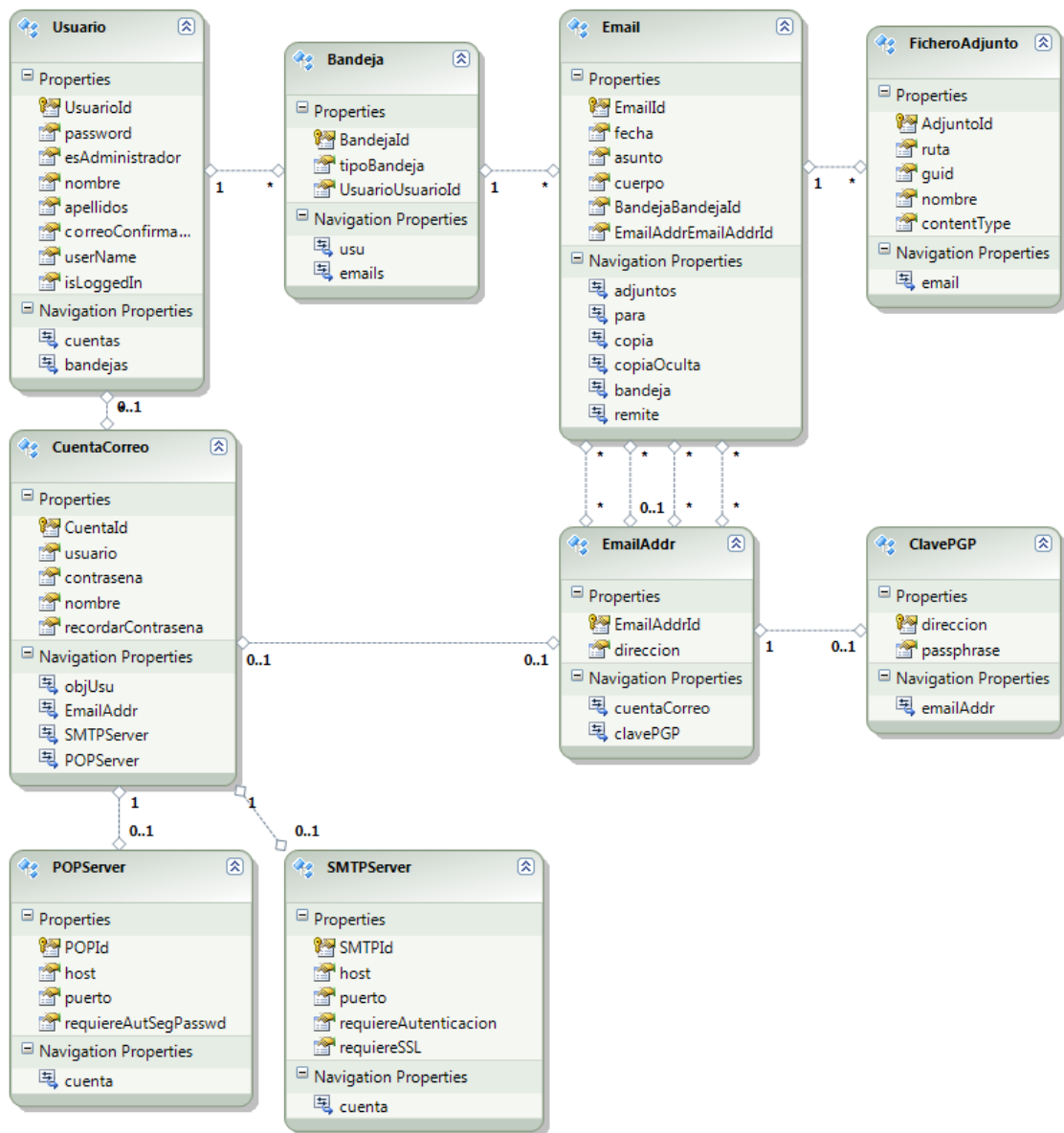


Ilustración 25: Tablas correspondientes al diagrama EF en base de Datos



**Ilustración 26: Diagrama de entidades del modelo generado por Entity Framework**

Ese diseño EF es la base de mi Modelo: EF produce código autogenerado para las clases que se reflejan en el diagrama.

Como el código autogenerado no debe tocarse, pero frecuentemente vamos a necesitar ampliar la funcionalidad de las clases autogeneradas, nos aprovechamos de que EF las genera como *partial*, lo que nos permite crear nuevos archivos .cs en los que sigamos añadiéndoles implementación.

Por ejemplo, como vemos en la Ilustración 27, hemos añadido al proyecto un fichero ClavePGP.cs (clase para la que ya el EF nos había dado implementación) para añadir a la clase parcial dos propiedades, *IsSubida* e *IsLocal*, que, mediante invocación al nuestro WCF Web Service de criptografía PGP, consultan el llavero local o los repositorios remotos para decirnos si están o no las claves correspondientes a la dirección de correo de la instancia. Son propiedades que no tienen ni deben tener reflejo en base de datos, que por tanto no se encuentran en el código autogenerado por EF, pero que sí necesitaremos utilizar, por ejemplo en diversas vistas .aspx (Ilustración 28).

```

/// Esto es una enorme ventaja a la hora de poder añadirles funcionalidad
/// sin tocar el código autogenerated.
/// </remarks>
public partial class ClavePGP
{
    private bool _IsSubida = false;
    /// <summary>
    /// True si para la dirección de correo de esta ClavePGP existe clave p
    /// La verificación la realiza a través de llamada a nuestro servicio d
    /// </summary>
    public bool IsSubida
    {
        get
        {
            if (string.IsNullOrEmpty(direccion)) return false;

            Cifrado.CriptografiaPGPClient pgp = new ProxyCifrado().PGP;

            _IsSubida = pgp.estaPublicaEnServidores(direccion);
            return _IsSubida;
        }
    }
}

```

Ilustración 27: Ejemplo de propiedad añadida a una clase EF de código autogenerated

```

<legend>Configuración de claves criptográficas PGP para esta cuenta</legend>
<% if (Model != null && Model.EmailAddr != null && Model.EmailAddr.clavePGP != null)
{
    if (Model.EmailAddr.clavePGP.IsSubida == true && Model.EmailAddr.clavePGP.IsLocal == true)
    { %>
        <div style="float:left;margin:5px;">
            El Par de Claves PGP para esta cuenta ya han sido creadas y subidas a servidor.
        </div>
    }
    else if (Model.EmailAddr.clavePGP.IsSubida == true && Model.EmailAddr.clavePGP.IsLocal == false)
    { %>
        <div style="float:left;margin:5px;">
            PGPWebmail <b>no</b> podrá utilizar esta cuenta de correo para operaciones PGP.<br />
            Ya existe clave pública PGP subida a repositorios para esta dirección de email
            pero PGPWebmail no tiene la correspondiente privada en su llavero local.
        </div>
    }
}

```

Ilustración 28: Utilización en vista de propiedad añadida al código autogenerated EF

## 2.2. Recepción POP y envío SMTP

Otro ejemplo de completar implementación de las clases del EF son las funciones de enviar o recibir correos electrónicos con las que teníamos que dotar a las clases *POPServer* y *SMTPServer*.

La funcionalidad de envío de correos electrónicos a través de protocolo SMTP viene incorporada ampliamente a la misma plataforma .NET, a través de las clases de los *namespaces* System.Net y System.Net.Mail. Mediante ellas implementé fácilmente la función “enviar” de *SMTPServer*.

Sin embargo y sorprendentemente, .NET no tiene soporte *built-in* para POP.

Por lo tanto, para conseguir la funcionalidad POP sin reinventar la rueda, tuve que seleccionar e incorporar una implementación ya existente publicada en *CodeProject* por Peter Huber<sup>7</sup>, que podemos ver incorporada a nuestro modelo en la carpeta /Models/Pop3.

<sup>7</sup> Huber SG, Peter (2006, octubre) “POP3 Email Client with full MIME Support (.NET 2.0)”. The Code Project [artículo en línea]. [Fecha de consulta: 16 de abril de 2011].

<http://www.codeproject.com/KB/IP/Pop3MimeClient.aspx>



Es reseñable que lo más difícil no es bajar correo electrónico siguiendo el protocolo POP3, que es un protocolo muy sencillito, sino convertir en texto legible el contenido *raw* de un email cualquiera; porque las formas de codificar los textos, en cualquier alfabeto, para poder enviarlos sobre ASCII 127, 7 bits, que es lo que soporta un email, son muy variadas.

Así me sorprendió que, aunque Peter Huber manifestaba que su código era “full MIME”, muchos de los asuntos de los mails me aparecían mal decodificados.

Tuve entonces que añadirle una vieja clase mía, *QuotedString*, con la que decodifico los asuntos de los mails; mientras que la decodificación de sus cuerpos la dejo a *QuotedPrintable*, de Huber, y toda la dificultad del formato MIME para los adjuntos la dejo igualmente en sus clases.

Tiene Huber una clase muy interesante: *RxMailMessage*. Es muy interesante porque hereda de la clase .NET System.Net.Mail.MailMessage, con lo cual es totalmente compatible con las clases de envío SMTP de las que ya habíamos hablado.

El bucle de recepción efectiva de correos reside en mi clase *ReceptorCorreos*. Se trata de un bucle infinito que se arranca en hilo aparte en el *Application\_Start* y que va bajándose en background los correos de las cuentas de los usuarios que estén logged-in en cada momento dado.

### 2.3. Modificaciones necesarias para cifrar y descifrar con adjuntos

Retorciendo un poco *RxMailMessage*, he conseguido extraer la codificación MIME de un email que el usuario haya redactado para ser enviado.

¿Para qué, si decimos que el SMTP está completamente soportado por .NET?

Para poder *cifrar* un email con adjuntos:

Partimos de un *Models.Email* (entidad del EF). Mediante el operador de conversión explícita que hemos implementado, *public static explicit operator RxMailMessage(Email mail)*, sacamos de él un *RxMailMessage*. En esa conversión hemos añadido los adjuntos con la simplísima instrucción de .NET *.Attachments.Add*. Pues bien, a continuación sacamos de él el MIME que .NET enviaría, mediante el método de extensión *public static void Save(this RxMailMessage Message, string FileName)*, basado en código de Allan Eagle<sup>8</sup>. Ahora ciframos este MIME mediante nuestro servicio web de cifrado, y eso es lo que enviamos.

También he tenido que retorcer un poco la clase de Huber *Pop3MimeClient*, de tal modo que su *StreamReader* de entrada pudiera ser no sólo la conexión POP, sino también un fichero.

De este modo podemos pasar a esta clase un correo que hayamos recibido cifrado, después de descifrarlo (lo que tenemos es un cuerpo MIME en un fichero), y *Pop3MimeClient* en colaboración con *RxMailMessage* se encargarán de sacar de él los adjuntos. A continuación, mediante la función *toEmail*, que le hemos añadido a *RxMailMessage*, obtenemos la instancia de nuestro *Models.Email* que necesitábamos.

### 2.4. Proveedor de membresía

Otras dos clases que podemos encontrar en *Models* son *ServicioDeMiembros* y *AutenticacionPorFormulario*. Ambas son necesarias para adaptar a nuestro modelo el control de usuarios que el proyecto MVC traía por defecto: Se trata de que un usuario pertenece al sistema y puede autenticarse si existe en nuestra base de datos, PGPWebmailDb, en la tabla *UsuariosSet* (mapeada a la entidad EF *Usuarios*, parte de nuestro modelo).

Para ello *ServicioDeMiembros* hereda de *System.Web.Security.MembershipProvider* y sobrescribe funciones como *CreateUser* o *ValidateUser* para que hagan uso de nuestra base de datos, mientras que *AutenticacionPorFormulario* hace uso de *System.Web.Security.Forms-*

<sup>8</sup> Eagle, Allan (2009, enero) "Adding Save() functionality to Microsoft.Net.Mail.MailMessage". The Code Project [artículo en línea]. [Fecha de consulta: 16 de abril de 2011].

<http://www.codeproject.com/KB/IP/smtplclientext.aspx>

*Authentication* para marcar mediante las cookies correspondientes al usuario como ya autenticado.

Además, esta personalización del servicio de membresía necesita de modificaciones en el Web.config, como podemos ver en la Ilustración 29.

```
<authentication mode="Forms">
  <forms loginUrl="~/Perfil/LogOn" timeout="2880" />
</authentication>
<membership defaultProvider="PGPWebmailMembershipProvider">
  <providers>
    <clear />
    <add name="PGPWebmailMembershipProvider" type="PGPWebmail.Models.ServicioDeMiembros" />
  </providers>
</membership>
```

Ilustración 29: Modificaciones en Web.config para personalizar el proveedor de membresía

## 2.5. El Repositorio

Piedra angular en el proyecto es la clase *Repositorio*.

Toda relación con la base de datos, para extraer información, añadirla o modificarla, pasa por las funciones de esta clase.

Dentro de Repositorio, a su vez, la pieza clave es su propiedad *PGPWebmailDbEntities*.

Se trata delObjectContext autogenerado por EF.

Cómo funciona esteObjectContext me ha dado bastantes quebraderos de cabeza; pero finalmente me he enamorado de él y he empezado a usarlo en proyectos en mi empresa.

La magia (y también el infierno) delObjectContext es que cada hilo de ejecución puede crear el suyo propio, de modo que haya *n* instancias del mismo funcionando simultáneamente dentro de la aplicación, sin que lo que pase en unas tenga reflejo ninguno en las demás. Unas con otras sólo pueden relacionarse a través de la propia base de datos: Cuando una instancia delObjectContext haga persistente su información al escribirla en la base de datos, sólo entonces, podrán tener conocimiento de la misma el resto de instanciasObjectContext que haya en la aplicación.

Como yo esto no sabía, mi idea inicial fue un Repositorio Singleton (instancia única en toda la aplicación) que automáticamente hace que también sea instancia única elObjectContext de EF, propiedad de Repositorio.

Pero, lógicamente, una aplicación web tiene múltiples hilos corriendo, los programas tú expresamente o no. Y el acceso a un mismoObjectContext desde múltiples hilos, aunque lo protejas con un *lock*, choca frontalmente con su filosofía. En consecuencia, con frecuencia no conseguía escribir en base de datos porque me saltaba una excepción: "New transaction is not allowed because there are other threads running in the session".

Cuando comprendí de qué se trataba unObjectContext cambié el enfoque:

*Repositorio* ya no sería una instancia única, sino que crearía uno por cada "transacción":

Por ejemplo: Cuando el usuario que está redactando un email le da al botón de guardar, la Action correspondiente en el Controller creará una nueva instancia de Repositorio (la que a su vez crea una nueva instancia delObjectContext EF). A continuación la Action creará una nueva instancia de Models.Email, la cargará con los datos del formulario que la vista ha entregado al controlador, e invocará a la función de su instancia de Repositorio encargada de añadir este Email alObjectContext. Cuando finalmente invocamos la función del Repositorio UpdateDataToDB, lo que estamos pidiendo alObjectContext es que haga persistentes sus modificaciones en la base de datos, y ya tendremos insertado nuestro nuevo Email, debidamente asociado a los EmailAddr de su remitente y de sus destinatarios, (EmailAddr que también habrán sido insertados en BD en caso de que no existieran ya), y debidamente asociado a sus FicheroAdjunto en caso de que los tuviese, que también se insertarán automáticamente en BD.

¡Magia!

Como hice que Repositorio implementase la interfaz `IDisposable`, cuando la instancia de Repositorio se destruye, automáticamente llama a la función sobrescrita `Dispose`, que a su vez invoca a la función `SaveChanges` del `ObjectContext` para asegurarnos de que todo cambio producido en un Repositorio va a terminar, siquiera sea implícitamente, escribiéndose en la base de datos.

Por esta razón a veces utilizo Repositorio dentro de una sentencia `using`, para que al salir de ella automáticamente se invoque el `Dispose` y se guarden cambios en BD.

Pero no siempre la utilizo: Porque si los objetos del `ObjectContext` (por ejemplo, el `Email` que guardábamos en el ejemplo de arriba) van a ser necesitados por la vista que devuelve el controlador (el `.aspx`, como se ve en la captura de la Ilustración 28, por ejemplo) y Repositorio ya ha llamado a `Dispose`, entonces su `ObjectContext` ya no existe, y sus objetos (el `Email` del ejemplo) ya no pueden accederse sin que salte una `NullReferenceException`.

Que esto no ocurra es tan sencillo como no crear Repositorio en un `using` cuando sus objetos vayan a ser usados por la vista. Ya se encargará el *garbage collector* de .NET de borrarlo cuando ya nada haga referencia a él.

En este caso, en vez del `SaveChanges` del `Dispose`, podemos usar una llamada explícita a `SaveChanges` a través de la función `UpdateDataToDB` del Repositorio, de la que ya he hablado.

Lógicamente, para todo el acceso a la información, *Repositorio* utiliza el poderoso LINQ to Entities. LINQ introduce toda la potencia de la sintaxis del lenguaje de consultas (tipo SQL) en el corazón mismo del lenguaje de programación, C#. El uso que hago del mismo es sin embargo sencillo y autoexplicativo, por lo que no tengo nada que comentar al respecto.

### 3. Proyecto PGPWebmail: Controladores

Para mí el concepto de Controlador ha sido todo un descubrimiento y, por más que leía sobre ellos, hasta que no me he puesto “manos a la masa” con ellos no lo terminaba de entender:

#### 3.1. Routing hacia las Actions de los Controllers

Simplificando, las vistas (las páginas `.aspx`) tienen “puntos de acceso” a los controladores, que son lo que llamamos “Actions”. Estos “puntos de acceso” pueden dispararse a través de pinchar en un enlace (Acciones GET) o a través de pulsar un botón (que pertenece a un Form HTML, Acciones POST).

Que al pinchar en la página “salte” la acción en el controlador es lo que llamamos “enrutamiento”, cuya configuración es precisamente lo primerísimo que hacemos en una aplicación MVC, en el `Application_Start` de la instancia de `HttpApplication` (Ilustración 30).

```
public class MvcApplication : System.Web.HttpApplication
{
    Pop3.ReceptorCorreos pop3receive = new Pop3.ReceptorCorreos();

    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
        routes.MapRoute(
            "Default", // Route name
            "{controller}/{action}/{id}", // URL with parameters
            new { controller = "Perfil", action = "LogOn", id = UrlParameter.Optional }
        );
    }

    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();

        RegisterRoutes(RouteTable.Routes);

        ViewEngines.Engines.Clear();
    }
}
```

Ilustración 30: Routing entre URLs y Acciones de los Controladores

Todos mis Controller heredan de mi clase ControladorBase, para que todos tengan las funciones “AutomaticUserIsIn” y “AutomaticUserIsOut”, que me permiten control sobre qué usuario está dentro del sistema y qué usuario no:

Me engancho al evento de página *window.onbeforeunload*, de javascript, para que siempre que salgamos de una página de mi aplicación invoquemos, mediante Ajax, AutomaticUserIsOut.

Pero a la entrada de cada Action invoco AutomaticUserIsIn.

De este modo, sólo si el usuario sale de una página de mi aplicación para entrar en otra que no sea de mi aplicación, quedará realmente marcado como “out”.

Esta información se persiste en base de datos, en la tabla UsuarioSet. Y nos es muy útil para saber de qué usuarios tenemos que estar “bajando correo”<sup>9</sup> y de qué usuarios no es necesario porque no están logged-in.

Los controladores siguen la misma estructuración que las carpetas de vistas: CuentaCorreo, Lectura, Perfil y Redacción; que a su vez es lo que propusimos en nuestro documento de Análisis y Diseño.

El controlador de Lectura se encarga de el listado de recibidos, de la vista de lectura de un mail concreto, de dar cauce hacia el controlador de redacción a las acciones “responder”, “reenviar” y “responder a todos”, y de gestionar las acciones de descifrar mail y de verificar su firma (para lo que hará uso de nuestro servicio web WCF de criptografía PGPWebService, del que hablaremos más adelante).

El controlador de Redacción se encarga de los listados de mails enviados y borradores, de la vista de edición de emails, del envío y del cifrado.

Las listas de mails, recibidos, enviados y borradores, son paginadas mediante Ajax, de tal forma que no los cargamos todos, sino que según la página solicitada por el usuario, esos mails pedimos a la base de datos, y esos reflejamos en la vista, de 50 en 50.

El controlador de CuentaCorreo lista las cuentas configuradas por el usuario, y permite añadir, editar o modificar las mismas. Es un ejemplo bonito y sencillo de funcionamiento maestro-detalle con MVC y de utilización del Repositorio.

El controlador Perfil es el encargado de la página de LogOn así como de la de registro de nuevo usuario.

### 3.2. Lo que devuelven las Actions

Hemos hablado de la “entrada” en las funciones Action de los controladores, pero me parece también interesante hablar de su “salida”: La salida que devuelva la Action del controlador será la que determine cuál es la página siguiente que verá el usuario después de que ha pinchado en ese vínculo o ha pulsado en ese botón.

#### **Return View**

En los controladores del proyecto podemos encontrar frecuentemente la salida más normal, *return View()*, que devuelve la página asociada a esa Action. Si bien frecuentemente encontramos que como parámetro de *return View* incluimos un objeto.

Este objeto, al que se denomina ViewData, será el que pueda manejar y utilizar la vista para componer la página de vuelta.

Así, si por ejemplo acabamos de redactar un email y hemos pulsado ‘cifrar’, el flujo del código ha enrutado hacia la Action *Cifrar*, el controlador ha actualizado el mail con los datos que por parámetros le vienen en esa Action, a continuación ha modificado el contenido de ese mail (ha cifrado su cuerpo), y ahora queremos que ese mail *modificado* sea lo que muestre la vista.

---

<sup>9</sup> Bajar correo es una tarea que se realiza en thread aparte que arrancamos en el arranque mismo de la aplicación, en el Application\_Start. Este hilo reside en la clase *ReceptorCorreo*, de la que ya hemos hablado más arriba.

Y para ello la vista deberá tener acceso a este objeto, el mail modificado de cuerpo ahora cifrado, para poder componer la página que muestre los distintos componentes de ese mail.

Eso es lo que frecuentemente hay que pasar como parámetro en el return de la Action: *return View(mail)*, por ejemplo.

### **Return RedirectToAction**

Otras veces tras la ejecución de esa Action por el controlador lo que queremos es que se renderice una vista que no es la asociada a esa Acción. Por eso devolvemos, en vez de “View”, “RedirectToAction”, que redirige a otra Acción que es la propia de la vista que queremos mostrar, esté o no en el mismo controlador.

Por ejemplo, después de enviar un email no queremos que se vuelva a ver el mail que acabamos de enviar, puesto que eso le daría la sensación al usuario de que el mail, en realidad, no se ha enviado.

Es más adecuado devolverle a la lista de emails enviados, en la que podrá ver que el último es precisamente el que acaba de enviar, entendiéndose así que se envió con éxito.

Pero la vista de esta lista es la que se corresponde con la acción *Index* del controlador de Redacción. Y por tanto saldremos de la acción Enviar con un *return RedirectToAction("Index", new { Name = Name });* en caso de éxito en el envío.

Otras interesantes aunque menos frecuentes respuestas de Actions que podemos encontrar en el proyecto son:

### **Return Json:**

En respuesta a la solicitud de borrar un email de una lista, por ejemplo. Se trata de una petición Ajax y su respuesta la encapsulamos en notación Json, para que la vista al recibirla sepa si debe eliminar el email de la lista que visualiza o por el contrario marcarlo como que no ha podido ser borrado (lo enmarca en rojo por medio de jQuery).

También tenemos una clase derivada de Json, *FileUploadJsonResult*, de John Rudolf Lewis<sup>10</sup>, para permitir subir ficheros a servidor (los adjuntos a los correos) mediante Ajax y sin uso de Flash (la solución Flash para esta función es muy popular, pero Flash no está soportado en el Safari para iPad ni iPhone).

### **Return File:**

Es la respuesta que da la acción de bajar fichero (bajarnos el adjunto de un email). Provoca el típico cuadro de diálogo del navegador sobre si deseamos guardar y dónde.

Haber puesto un link que directamente hubiera referenciado al fichero (como si se tratase de un recurso HTML más), supondría un gran agujero de seguridad; hasta el punto de que Internet Explorer ni reacciona (fue lo primero que probé; Chrome sí abría las imágenes).

### **Return PartialView:**

Las listas de correo se cargan paginadas y mediante Ajax: Como podemos ver en Index.aspx de Lectura, por ejemplo, inyectamos mediante jQuery en un DIV que inicialmente está vacío el resultado de la acción “ListaRecibidos”. Por tanto este resultado no debe ser una página web HTML completa, sino contenido DOM parcial que pueda entrar dentro de un DIV. Este contenido es ListaRecibidos.ascx, que es precisamente la vista asociada a la acción “ListaRecibidos”.

Cuando las acciones no están asociadas a aspx (System.Web.Mvc.ViewPage) sino a ascx (System.Web.Mvc.ViewUserControl) no devuelven View (un cuerpo HTML completo y consistente) sino PartialView (un contenido DOM parcial).

---

<sup>10</sup> Lewis, John Rudolf (2009, agosto) “jQuery File Upload in ASP.NET MVC without using Flash”. ASP Zone [artículo en línea]. [Fecha de consulta: 10 de mayo de 2011].

<http://aspzone.com/tech/jquery-file-upload-in-asp-net-mvc-without-using-flash/>

### 3.3. Evitar que el Internet Explorer haga caché del resultado de acciones

Relacionado con la devolución de las Actions quiero señalar un efecto perverso que se produce en Internet Explorer (y no así en Firefox ni en Chrome), y que es que el resultado de determinadas acciones se guarda en la caché del navegador cliente.

Cuánto me dolió la cabeza con esto.

Para evitarlo, una vez que diagnosticué el problema, lo que me llevó muchas horas, incorporé la clase *CacheControlAttribute*, de Craig Stuntz<sup>11</sup>, que permite marcar una acción para que su resultado no se guarde nunca en caché.

Esto resultó especialmente necesario para las invocaciones Ajax, como la paginación de las distintas bandejas de correos.

## 4. Proyecto PGPWebmail: Vistas

Siguen la misma estructura que los controladores, que ya hemos explicado.

### 4.1. Renderizado especial para iPhone como prueba de concepto

Sin embargo, como puede verse en el Solution Explorer, tenemos carpetas específicas /Mobile/iPhone.

Tal y como expresé en mi declaración de intenciones, tanto en mi Plan de Trabajo como en mi documento de Análisis y Diseño, la idea era crear una aplicación que pudiera ser accedida desde dispositivos no-PC, y por eso proponíamos una aplicación web.

Sin embargo, cuando el tamaño de una pantalla es pequeño y la orientación natural es vertical en vez de apaisada, parece interesante adecuar la interfaz gráfica.

Además ése es uno de los grandes logros de desacoplar totalmente la vista de la lógica: Que podemos crear fácilmente tantas vistas distintas cuantas queramos sin que eso altere nada esencial en la aplicación.

Por eso, a modo de ejemplo, he querido incluir un renderizado alternativo de las vistas de PGPWebmail y he preparado uno sencillito (e insisto, como prueba de concepto) para iPhone4.

Para conseguir esta funcionalidad, además de crear esas vistas específicas, hemos incluido la clase *CustomMobileViewEngine*<sup>12</sup>, y hemos registrado las ViewEngines correspondientes en el *Application\_Start*.

### 4.1. jQuery: Código aliado del Ajax que corre en cliente

Otro gran descubrimiento para mí en este trabajo ha sido jQuery.

jQuery es una biblioteca de JavaScript libre y de código abierto tan potente que su núcleo ha sido incluido por Microsoft en la plantilla misma de los proyectos MVC.

En prácticamente todas mis vistas hago uso de jQuery. Es omnipresente.

Por ejemplo vemos en *CuentaCorreo/Create.aspx* que cuando el usuario va a configurar una cuenta de gmail.com o de yahoo.es o de yahoo.com, automáticamente se le rellenan los campos con los valores adecuados (nombre de los servidores, puertos que usan, etc.)

Para ello, mediante jQuery nos “suscribimos” al evento *keyup* del `<input id = “EmailAddr_direccion”>`, de tal modo que tecla a tecla vamos comprobando en cliente lo que va

---

<sup>11</sup> Stuntz, Craig (2010, junio) “How to stop MVC caching the results of invoking and action method?”. Stack Overflow [artículo en línea]. [Fecha de consulta: 23 de abril de 2011].

<http://stackoverflow.com/questions/2969450/how-to-stop-mvc-caching-the-results-of-invoking-and-action-method>

<sup>12</sup> Hanselman, Scott (2010, noviembre) “A Better ASP.NET MVC Mobile Device Capabilities ViewEngine”. Scott Hanselman's ComputerZen.com [artículo en línea]. [Fecha de consulta: 24 de marzo de 2011].

<http://www.hanselman.com/blog/ABetterASPNETMVCMobileDeviceCapabilitiesViewEngine.aspx>



escribiendo el usuario, y si lo que ha escrito tiene el dominio de gmail o de yahoo... modificamos mediante jQuery el contenido de los <input> adecuados.

En general jQuery hace magia encontrando elementos de la página y, una vez encontrados, modificándolos. Un interesante tutorial de jQuery tiene un título así de significativo: *How to Get Anything You Want*<sup>13</sup>.

También lo usamos para situar el cursor en determinada caja de texto cuando se abre una página; o para que aparezca un mensaje de paciencia que deje mientras griseada y bloqueada la página (mientras un email se está enviando, por ejemplo, en Redactar.aspx); o para que una parte de la página aparezca o desaparezca con efecto acordeón (como el <div> que contiene el <input type="file"> para subir adjuntos, que sólo aparece si el usuario lo pide).

Me parece destacable (ay, cuánto me costó) la paginación Ajax de las listas de correos que podemos ver en Lectura/Index.aspx, por ejemplo, con su correspondiente ViewUserController. Ya hemos hablado de ellas al hablar de las PartialView.

También es muy destacable su uso para subir archivos a servidor (los adjuntos a un email) mediante Ajax, que también está en Lectura/Index.aspx.

En la Ilustración 31 podemos ver un ejemplo de uso de jQuery: En la creación de cuentas de correo, mientras la contraseña de protección de las claves PGP que vamos a crear y asociar a la misma no tenga al menos 6 letras, no se activa el botón de crearlas.

```
<script type="text/javascript">
    $(document).ready(function () {

        $("#EmailAddr_clavePGP_passphrase").keyup(function () {
            if ($("#EmailAddr_clavePGP_passphrase").val().length < 6)
                $("#btnCrearClave").attr('disabled', 'disabled');
            else
                $("#btnCrearClave").removeAttr('disabled');
        });
    });
```

Ilustración 31: Ejemplo de jQuery: Habilitar un botón según la longitud de la contraseña

#### 4.2. Marcadores <% %> en las .aspx: Código que corre en servidor

Finalmente me gustaría señalar cómo podemos utilizar marcado en las .aspx para introducir y utilizar dentro de él código que correrá en servidor:

Antes de enviar la página respuesta a una Action hacia el navegador del cliente, ASP NET “monta” el HTML que va a enviar. Las aspx que vemos en nuestro proyecto son como las plantillas que ASP NET utilizará para realizar este montaje. Y en dichas plantillas es posible usar código C#.

¿Y qué datos puede manejar este código?

Sólo (no es poco) los que le pasemos a la vista como parámetro de las respuestas de la Action (View, PartialView, etc.). Es lo que se llama el ViewData, que puede estar fuertemente tipado o no (a la vista le entra un objeto genérico o sólo un objeto de un tipo específico).

Se utiliza en todas las vistas, pero puede ser un buen ejemplo el ViewUserController ListaRecibidos.ascx.

<sup>13</sup> Swedberg, Karl (2006, noviembre) “How to Get Anything You Want”. The jQuery Project [artículo en línea] [Fecha de consulta: 19 marzo 2011].

[http://docs.jquery.com/Tutorials:How\\_to\\_Get\\_Anything\\_You\\_Want](http://docs.jquery.com/Tutorials:How_to_Get_Anything_You_Want)

Esta vista no es más un una tabla de 50 emails, con cuatro columnas para cada uno: el remitente, el asunto, la fecha y un checkbox para poder marcar y borrarlo.

Desde una perspectiva HTML sólo tenemos que escribir `<table>` y dentro de ella repetir 50 veces `<tr> <td> </td> <td> </td> <td> </td> <td> </td> </tr>`. Pero, ¿con qué contenido dentro de cada `<td>`?

Ese contenido habrá de generarse dinámicamente en servidor: Lo crea la acción `ListaRecibidos` de `LecturaController` buscando en base de datos; lo empaqueta en un `ListaCorreosViewData`; `ListaRecibidos.ascx` lo recibe y con él, con código marcado entre `<%>`, monta la tabla que necesitamos, recorriendo en bucle cada `Email` de la propiedad `Correos` del `ViewData` cargado, tomando de cada uno la información necesaria y escribiéndola dentro de cada `<td>`.

Podemos verlo en la Ilustración 32.

```
<%@ Control Language="C#"
Inherits="System.Web.Mvc.ViewUserControl<PGPWebmail.ViewData.ListaCorreosViewData>" %>

<table class="tableMails" style="background-color:#E0E0E0;width:100%;">
  <%
  int n = 0;
  if (ViewData.Model.Correos != null)
  {
    try
    {
      foreach (var correo in ViewData.Model.Correos)
      {
        n++;
        string sColor = "\"background-color:#E0E0E0;\"";
        if (n % 2 == 0) sColor = "\"background-color:White;\"";
        string sAsunto = "(vacío)";
        if (string.IsNullOrEmpty(correo.asunto) == false)
          sAsunto = correo.asunto;
        string sRemite = "(vacío)";
        if (correo.remite != null &&
            string.IsNullOrEmpty(correo.remite.direccion) == false)
          sRemite = correo.remite.direccion;
        <%
        <tr id="row<%= correo.EmailId %>" style=<%= sColor %> >
          <td style="width:250px;border-width:thin;border-color:White;">
            <%= sRemite %>
          </td>
          <td style="border-width:thin;border-color:White;">
            <%= Html.ActionLink(sAsunto, "Leer", "Lectura",
              new { Name = Page.User.Identity.Name, id = correo.EmailId,
                page = ViewData.Model.PaginationPage }, null)%>
          </td>
          <td style="width:130px;border-width:thin;border-color:White;text-align:right;">
            <%= correo.fecha.ToShortDateString() + " "
              + correo.fecha.ToShortTimeString()%>
          </td>
          <td style="width:65px;border-width:thin;border-color:White;text-align:right;">
            <input type="checkbox" id="<%= correo.EmailId %>"
              name="<%= correo.EmailId %>" class="chkDel" />
          </td>
        </tr>
        <%
      }
    }
    catch { }
  }%>
</table>
```

Ilustración 32: Marcadores `<%>`: Código que se ejecuta en servidor para generar la página



## 5. Proyecto PGPWebService

---

Se trata de un servicio web WCF que da servicios criptográficos PGP que permiten al cliente de correo PGPWebmail cifrar, descifrar, firmar o verificar firmas PGP.

### 5.1. Servicio Web WCF

---

Su contrato podemos verlo en ICriptografiaPGP.cs:

```
[OperationContract]
bool checkEndpoint();

[OperationContract]
bool estaPublicaEnLlaverLocal(string sAddr);

[OperationContract]
bool estaPrivadaEnLlaverLocal(string sAddr);

[OperationContract]
bool estaPublicaEnServidores(string sAddr);

[OperationContract]
void bajarPublicaDeServidores(string sAddr);

[OperationContract]
void crearParDeClaves(string sName, string sAddr, string sPassphrase);

[OperationContract]
void subirPublicaAServidor(string sAddr);

[OperationContract]
string cifrar(out string sError, string sPlainText, string sDestinatarios);

[OperationContract]
string descifrar(out string sError, string sPlainText, string sPassphrase);

[OperationContract]
string firmar(out string sError, string sPlainText, string sRemitente, string sPassphrase);

[OperationContract]
string verificarFirma(out string sError, string sPlainText, string sRemitente);

[OperationContract]
string firmarYcifrar(out string sError, string sPlainText, string sRemitente, string sPassphrase,
string sDestinatarios);

[OperationContract]
string descifrarYverificar(out string sError, string sPlainText, string sPassphrase, string
sRemitente);

[OperationContract]
string cifrarFichero(out string sError, string sPlainFile, string sDestinatarios);

[OperationContract]
string firmarFichero(out string sError, string sPlainFile, string sRemitente, string sPassphrase);

[OperationContract]
string verificarFirmaFichero(out string sError, string sSignedFile, string sRemitente);

[OperationContract]
string firmarYcifrarFichero(out string sError, string sPlainFile, string sRemitente, string
sPassphrase, string sDestinatarios);

[OperationContract]
string descifrarYverificarFichero(out string sError, string sPlainFile, string sPassphrase, string
sRemitente);
```

Su implementación se encuentra en CriptografiaPGP.svc.cs.

Para poderlo usar desde PGPWebmail incluimos en este proyecto una Service Reference, como vemos en la Ilustración 33.

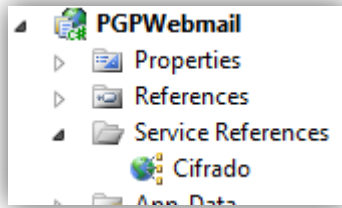


Ilustración 33: Service Reference a PGPWebService (Cifrado)

## 5.2. Implementación criptográfica

En cuanto a la implementación criptográfica propiamente dicha, hemos tomado un proyecto de Emmanuel Kartmann<sup>14</sup> en CodeProject, *Gnu Privacy Guard (GPG/PGP) for .NET [v1.0]*. Hemos tenido que añadirle funciones, pero la estructura básica ha sido perfecta.

Se trata de un *wrapper*, es decir, una envoltura en torno al ejecutable de consola `gpg.exe`.

`Gpg.exe` es una implementación probada, completa y libre del estándar OpenPGP tal y como se describe en el RFC 2440.

Aunque en el mundo Windows pueda sonar extraño implementar un programa que lo que hace es “tirar” de un ejecutable de consola, en realidad esto es algo normal, es lo que se llaman “front-ends”, y para Linux hay muchísimos: Front-ends para LaTeX, para IPTables, etc.

Se trata de interfaces gráficas y amables que facilitan el uso de programas de consola altamente confiables y probados.

La habilidad de Emmanuel Kartmann ha sido saber engancharse bien al *stdin*, *stdout* y *stderr* de la aplicación de consola. Hecho esto, pasarle a la aplicación los comandos correctos para que realice diversas operaciones criptográficas es juego de niños. Yo he tenido que ampliar bastantes de estas operaciones, que no estaban originalmente soportadas en el proyecto de Kartmann.

Al no poder probar un uso concurrente masivo (sí he probado la concurrencia, pero dentro de mis limitados recursos, tres PCs), “por si acaso”, he preferido proteger el uso del wrapper de Kartmann con instrucciones `lock` en la entrada a cada función del servicio.

## 5.3. El cifrado de adjuntos

Cuando en el último estadio del desarrollo del proyecto le “hiqué el diente” a los adjuntos, comprendí que no era razonable pasarle al servicio por parámetros una cadena que contuviese dentro de sí el contenido completo de un archivo.

Vi que el *transfer mode* de un servicio podía cambiarse de *Buffered* a *Streamed*, pero, puesto que ambas aplicaciones corren dentro del mismo PC, preferí la opción de pasar simplemente la ruta completa al fichero que queremos cifrar, firmar, descifrar o verificar.

Por eso el contrato contiene operaciones específicas sobre ficheros además de las de cadenas, que son las que PGPWebmail utiliza cuando los emails contienen adjuntos.

## 6. Instaladores

La aplicación web ASP NET MVC y el servicio web WCF tienen cada uno su propia instalación, como podemos ver en el Solution Explorer.

Lo único destacable al respecto es el uso que ambas instalaciones hacen de la DLL que genera el proyecto `CustomInstall`, también nuestro y también parte de la solución.

Microsoft es muy celoso de todo el tema de permisos y seguridad.

<sup>14</sup> Kartmann, Emmanuel (2003, septiembre) “Gnu Privacy Guard (GPG/PGP) for .NET [v1.0]”. The Code Project [artículo en línea]. [Fecha de consulta: 16 de marzo de 2011].

<http://www.codeproject.com/KB/security/gnupgdotnet.aspx>

Quizás por esto, aunque en el proyecto de despliegue de una aplicación web indiques a Visual Studio las determinadas carpetas sobre las que deseas tener permisos de escritura, no te hace caso y simplemente no te los da: Terminada la instalación todas las carpetas tendrán los mismos permisos, y no de escritura.

Sin embargo no había ninguna manera de soslayar esta necesidad de nuestro proyecto: Tenemos que poder escribir los mails que nos llegan y sus adjuntos, y no queremos “petar” una base de datos que, al ser gratuita (versión Express) tiene una limitación de espacio. También hemos visto que queremos usar archivos para operaciones criptográficas sobre mails de gran tamaño, para evitar pasárselos como parámetro al servicio en una cadena.

Por ello fue necesario pelear mucho para conseguir una instalación personalizada que concediese estos permisos.

La instalación personalizada para proyectos de despliegue de aplicaciones con Visual Studio está bien documentada en Internet:

Se trata de añadir un proyecto que contendrá una clase derivada de System.Configuration.Install.Installer y que generará una DLL; el output de este proyecto debe añadirse a Install y a Commit en la vista de Custom Actions del proyecto de despliegue.

Dentro de la clase derivada de Installer podremos sobrescribir diversas funciones: Las que nos interesan son OnAfterInstall y Commit.

En Commit simplemente abriremos la página web de la aplicación web, lo que ayuda a la persona que acaba de instalarla a saber “por dónde empiezo a usarla”.

Y en OnAfterInstall es donde cambiamos los permisos de las carpetas que nos interesan.

Es un muro duro el de los permisos en las carpetas NTFS. Sorprendentemente la plataforma .NET tiene poca ayuda al respecto. Es más, parece que un desarrollador de Microsoft, Renaud Paquay, hizo un namespace de clases que encapsularan toda esta funcionalidad, Microsoft.Win32.Security, pero todos los links a este proyecto en Internet terminan indefectiblemente en una página de “recurso no encontrado”<sup>15</sup>: Sospechoso, ¿no?

Y si utilizar la API de Windows para conseguir esto ya es difícil, hacerlo en C# en vez de C++, por mucha ayuda que nos preste “pinvoke.net”, se hace muy difícil.

Así que finalmente utilicé la solución a la que se apuntaba en este Foro:

<http://forums.asp.net/p/1255778/2354823.aspx#2354823>

Completada con los datos que obtuve de este otro:

<http://learn.iis.net/page.aspx/624/application-pool-identities/>

Se trata, una vez más, de llamar a un ejecutable de consola, el cacls.exe y el icacls.exe, con los parámetros adecuados. Y parece que funciona.

Nos parecía muy importante, por mor de la simplicidad de la instalación, que no fuera necesario indicar al usuario instalador que tenía que ir a tal y cual carpeta y dar permisos a tal y cual usuario.

## 7. Unit Tests

Entiendo que una enorme ventaja de desacoplar vista de lógica es que eso nos permite programar tests que automáticamente prueben una por una las funciones que serían invocadas desde la vista.

---

<sup>15</sup>Paquay, Renaud. Microsoft.Win32.Security

<http://www.gotdotnet.com/Community/UserSamples/Details.aspx?SampleGuid=e6098575-dda0-48b8-9abf-e0705af065d9>

En el caso de ASP NET MVC, bastaría con tests que invocasen cada una de las Action de los Controller y verificasen que el ActionResult de salida (sea éste del subtipo que sea) es el esperado. Limpísimo.

Lo he intentado. La solución original llevaba un proyecto de Unit tests. Pero me he topado con las dificultades del “mocking” y sencillamente no me ha dado tiempo.

Ya en mi documento de Especificación y Diseño apuntaba a que intentaría los Unit Tests, pero que si no me era posible, haría y documentaría la prueba de forma manual, que es lo que he hecho (está documentada en “Documento de pruebas.pdf” de la PEC 3).

## IX. Objetivos conseguidos

---

Hemos logrado desarrollar una aplicación plenamente funcional que actúa como cliente de correo electrónico permitiendo a la vez firmar y/o cifrar el cuerpo y adjuntos del mensaje mediante PGP, y cuyo acceso es web.

Con ello hemos cubierto los tres objetivos generales del proyecto software: Hacer fácilmente accesible el cifrado y firma de correos electrónicos mediante PGP y que el acceso a ello fuera lo más universal posible, desde cualquier dispositivo con acceso a Internet.

Y hemos cubierto los dos objetivos específicos: Desarrollar en lenguaje C# un cliente de correo electrónico web mediante ASP.NET MVC, de interfaz gráfica sencilla pero que funcione bien no sólo en ordenadores sino también en *smart-phones*, (como hemos probado sobre iPhone, Ilustración 24); y desarrollar en lenguaje C# un Servicio Web WCF que exponga una máquina que permita el cifrado, descifrado, firma y verificación de mensajes mediante cifra asimétrica PGP, que es el PGPWebService, de cifrado, del que hace uso el cliente web de correo.

Han quedado cubiertos todos los requisitos funcionales (pág. 17) y no-funcionales (pág. 13) que planteamos en el documento de Análisis y Diseño, salvo dos:

(U-03) Cambiar perfil de usuario: Era sencillo porque el mismo proyecto que la plantilla MVC te genera ya incluye esta funcionalidad; pero como en una primera fase del uso del producto no sería necesaria, la eliminé, dejándola para una segunda fase.

(U-06) Borrado de usuarios: Puesto que el despliegue se planteaba dentro de una misma organización, pequeña, para algunas decenas de usuarios, no me pareció necesario para una primera fase dar una interfaz a un administrador para que borrara usuarios; en caso de necesidad siempre podría hacerse mediante la ejecución de una sentencia SQL; darle interfaz gráfica de administración lo dejo pues para una segunda fase .

Además las listas de correos, que estaban definidas para ser visualizadas de 100 en 100 (NF-18 del documento original de diseño), han pasado a 50, por mejor usabilidad.

Pero quizás lo más importante son los objetivos de carácter didáctico alcanzados a través de todo el proceso, profundizando como estudiante en algunas de las más interesantes tecnologías .NET, entre otras:

- Lenguaje de programación C#, en la versión que brinda la versión 4 de .NET Framework: métodos de extensión, tipos anónimos, propiedades automáticas, inicialización rápida de objetos, etc.
- LINQ, lenguaje de consulta “tipo SQL”, pero integrado en el núcleo mismo del lenguaje C#, en su versión *LINQ to Entities*.
- ADO.NET Entity Framework, un sistema de acceso a la base de datos muy potente que utiliza un mapeado automatizado entre la estructura relacional de la base de datos y la estructura orientada a objetos del código en C#. Este mapeado nos permite obtener de la estructura relacional de nuestro modelo de datos en SQL Server, las clases del modelo de nuestra aplicación, o viceversa.

- ASP.NET MVC, versión de ASP.NET adaptada al patrón de arquitectura MVC (Modelo – Vista – Controlador) especialmente orientado al desacople de capas, lo cual es ideal para aplicaciones web cuya capa de presentación está muy desacoplada por su propia naturaleza; gracias a MVC la información que en cada momento la vista necesita conocer sobre el estado de ejecución de la aplicación subyacente es la mínima. La utilización de patrón de arquitectura ha dado como resultado un código con un nivel de estructuración, claridad y limpieza que yo mismo he resultado sorprendido.
- Servicios Web WCF, un paso más allá de los Servicios Web ASP.NET, permitiendo una interacción mucho más rica con el cliente que los consume, mayor seguridad, transaccionalidad, etc. Haberlos descubierto podrá ayudarme en un futuro para aplicaciones para las que intuya que pueda ser interesante una arquitectura SOA.
- Ajax, para, mediante la interacción con el código de ejecución en el navegador cliente (Javascript, mediante las librerías jQuery) cargar parcialmente las páginas, para lograr una experiencia de usuario más suave y fluida.

Y a todos estos objetivos alcanzados, que me resultaban “esperables”, se suma un logro inesperado: el “gusanito” que toda la investigación a la que me ha llevado la ejecución de este proyecto, me ha metido en el cuerpo:

Al usar MVC, he descubierto la importancia de los patrones de arquitectura y de diseño; MVC no es el único, y se me ha despertado la curiosidad por indagar otros, cuáles son sus fines y objetivos, y cuáles y por qué pueden serme útiles en mi trabajo diario como desarrollador. De pronto se me ha despertado el interés por la arquitectura del software -con la que profesionalmente he tenido muy poco contacto- y por la Ingeniería del Software: Ahora quiero saber qué hay detrás de acrónimos como TDD, o DDD, o SOA; donde yo hasta ahora sólo tenía en mente que existiera la OOP. He usado y me ha encantado el patrón *Singleton*, del que ya había oído hablar en IS; pero, siguiendo en esta línea, ¿podría serme de utilidad en algunas situaciones de mi trabajo diario el *Multiton*? ¿Y qué es eso de las *Factory*, de las que he oído hablar bastante en mis investigaciones sobre Unit Testing? He usado en el proyecto la clase *CacheControlAttribute* de Craig Stuntz que funciona como un *Decorator* de la Action de un Controller: ¿No podría ser interesante aprender a utilizar esta técnica para extender la funcionalidad de un objeto sin tener que derivarlo?

Y LINQ o el Entity Framework me han abierto al mundo de las “tecnologías”: Yo hasta ahora vivía en un mundo lenguaje-céntrico, donde saber programar era saber usar un lenguaje de programación y unas APIs y el resto me parecían oropeles de marquetin. De pronto entiendo la importancia de abrirme a descubrir toda la funcionalidad que pueden brindarme tecnologías que están al alcance de un clic mediante Visual Studio (como EF o WCF).

Y para mantenerme “al cabo de la calle” en todo esto, después de haber experimentado su importancia, me he suscrito a varios *newsletters* y a una revista en papel sobre desarrollo en .NET.

## X. Evaluación de costes

---

Para evaluar económicamente los costes que hubiera tenido el proyecto de no haber sido de carácter educativo, he considerado los siguientes honorarios por hora, según perfil profesional:

- Director de proyecto: 80€/hora
- Analista: 55 €/hora
- Programador: 30 €/hora
- Téster: 25 €/hora
- Diseñador gráfico: 25 €/hora
- Documentador: 25 €/hora

Puesto que nuestro plan de trabajo se elaboró usando como unidad de trabajo la hora y no el día, podemos fácilmente traducir lo trabajado en dinero:

Ha habido:

- ✓ 53 horas de trabajo de Director del proyecto: 4.240€
- ✓ 80 horas de Analista: 4.400€
- ✓ 197 horas de Programador: 5.910€
- ✓ 16 horas de Téster: 400€
- ✓ 11 horas de Documentador: 275€.

Todo lo cual arroja un total de: 15.225€, frente al presupuesto inicial de 13.575€.

Para documentación, *testing* y desarrollo la estimación inicial de horas era de 55 horas menos. Por lo tanto ha habido un desfase entre el costo inicial previsto y el final de 1.650€, un 12% más de lo inicialmente presupuestado.

## XI. Trabajo futuro y recomendaciones de mejora

---

### 1. Ayuda en línea

---

Para la PEC 3 ya hemos producido un documento muy completo y claro de ayuda al usuario, el “Manual de Usuario.pdf”. Convertir este documento a HTML es extremadamente sencillo, y eso permitiría añadirle a la página Site.master del proyecto un link permanente a “ayuda en línea” para dotar al usuario con un sistema de ayuda que le facilite saber rápidamente como trabajar con la aplicación.

### 2. Retomar el proyecto de *Unit Testing*

---

Como ya he indicado antes, la solución inicial incluía un proyecto de *Unit Testing*. De hecho lo utilicé mucho al principio hasta que me encontré con dificultades que no lograba salvar y que estaban distrayendo demasiado mi atención y mi (escaso) tiempo del proyecto principal.

Quizá lo primero que haría si volviera a tener 400 horas para continuar el proyecto sería volver a incluirle ese proyecto de test y ponerlo a funcionar, profundizando en conceptos como el *mocking* y el TDD.

### 3. Embellecer la interfaz gráfica y *renderizado* para más dispositivos

---

Lo siguiente en lo que invertiría tiempo sería en una mejora de la interfaz gráfica. Contemplé en su momento la posibilidad de hacerlo usando la Mvc Template Gallery de CodePlex<sup>16</sup>, pero también me parecieron bastante feas las propuestas, así que no me animé por ninguna. Ahora que la funcionalidad está terminada, podría probar unas y otras plantillas CSS hasta dar con algo un poco más atractivo.

En la Ilustración 34 muestro algunas posibilidades, pero es que son poco sencillas y no muy atractivas, por lo que no habría más remedio que trabajar bastante en ellas.

También me gustaría mejorar el *renderizado* para dispositivos móviles, que pasara de ser una prueba de concepto a algo realmente atractivo y adaptado al look&feel de los dispositivos.

Por ejemplo podríamos seguir las indicaciones de Scott Hanselman para usar el *IUI Project*<sup>17</sup>.

---

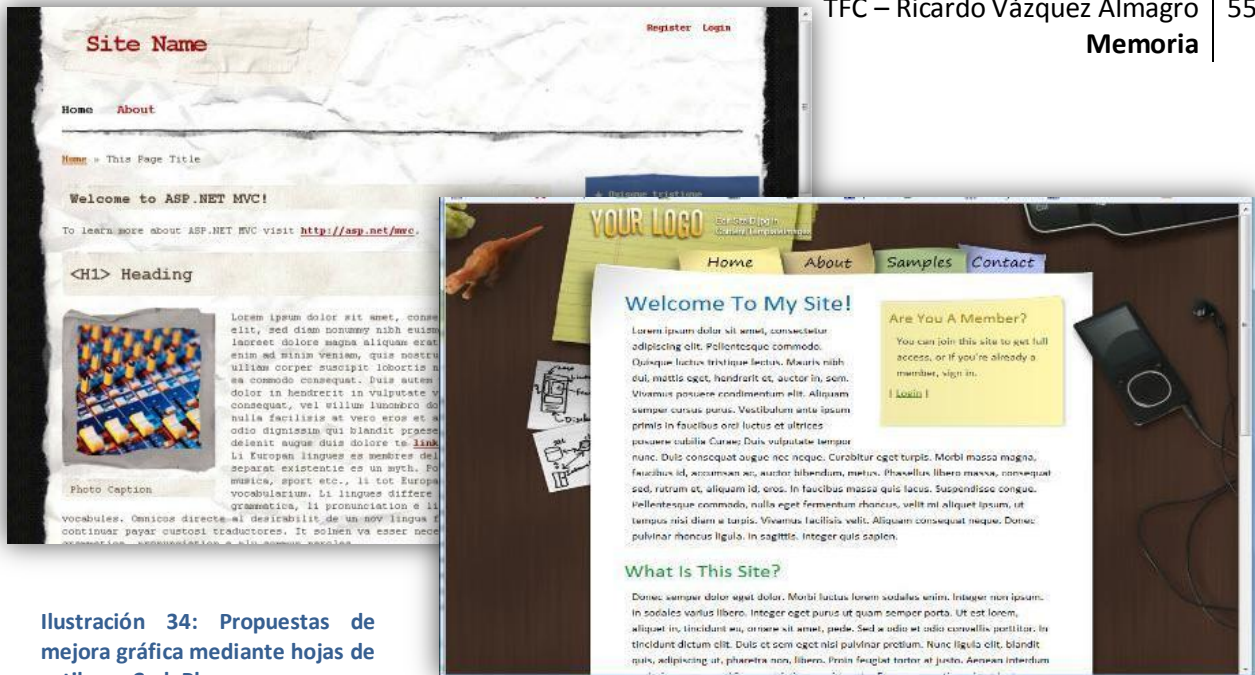
<sup>16</sup> Hexter, Eric y otros (2010, agosto) “MvcContrib Template Gallery”. CodePlex [artículo en línea]. [Fecha de consulta: 1 de abril de 2011].

<http://mvcontribgallery.codeplex.com/>

<sup>17</sup> Hanselman, Scott (2008, septiembre) "The Weekly Source Code 28 - iPhone with ASP.NET MVC Edition". Scott Hanselman's ComputerZen.com [artículo en línea]. [Fecha de consulta: 8 de junio de 2011].

<http://www.hanselman.com/blog/TheWeeklySourceCode28iPhoneWithASPNETMVCEdition.aspx>





**Ilustración 34: Propuestas de mejora gráfica mediante hojas de estilo en CodePlex**

#### 4. Solucionar ciertos agujeros de confidencialidad

A continuación perfeccionaría la seguridad de la funcionalidad ya existente: Los adjuntos llegan cifrados; pero una vez descifrados se guardan en claro y en disco duro en el servidor. Esto habría que arreglarlo, porque el administrador que pueda entrar al servidor puede acceder a los adjuntos descifrados.

Para ello investigaría la posibilidad de devolver el Controller.File de la Action “DownloadFile” en LecturaController cargado desde un *MemoryStream* en vez de cargarlo mediante la ruta de un fichero realmente existente en disco duro.

También ha quedado inseguro que la aplicación almacene las passphrase de los usuarios, aunque sea ofuscadas; debería solicitarla al usuario cuando la necesite, sin almacenar nada.

Y sería interesante probar un despliegue sobre HTTPS. Parece que no es difícil configurar el IIS para que nos brinde esta funcionalidad. Esto nos abriría a la posibilidad del acceso WAN.

#### 5. Completar funcionalidades de cliente de correo

En cuanto a las funcionalidades propias de un cliente de correo, implementaría una agenda de contactos a la que se pudiera acceder “en caliente” en el momento de introducir las direcciones de email de nuestros destinatarios.

E igualmente sería interesante implementar una página de búsqueda de correos, por fecha, destinatarios, remitente, asunto...

En relación con esta misma idea, podía también ser interesante implementar la posibilidad de aplicar etiquetas a los mails, que funcionarían como una especie de carpetas, como hace GMail.

#### 6. Completar funcionalidades como aplicación web

En cuanto a las funcionalidades como aplicación web, habría que implementar la posibilidad de “recuperación de contraseña”, para el caso de haberla olvidado (se enviaría al correo de recuperación, que ya lo guardamos en el perfil de usuario); así como la de poder cambiar los datos del perfil, entre ellos poder cambiar la contraseña.

#### 7. Completar funcionalidades de llavero PGP

Finalmente, en cuanto a las funcionalidades PGP, habría que implementar interfaz para poder importar y exportar las claves del llavero. Esto permite al usuario (siempre que tenga su passphrase) llevarse sus claves a otra máquina, para poder usar otras aplicaciones PGP; o

viceversa: traerse sus claves a PGPWebmail para poder usarlas en nuestra solución web además de en otras aplicaciones con soporte PGP en las que ya las esté usando.

También es necesario dar interfaz para que el usuario pueda firmar con su clave privada las claves públicas bajadas de repositorios que ya ha podido comprobar y en las que por tanto confía (para ayudar a la constitución de la “red de confianza” en la que reside la fuerza del criptosistema PGP; para ayudar a “tejer la telaraña”).

## XII. Conclusiones

---

Aunque llevo años trabajando como programador, la UOC me ha empujado mediante este TFC a zambullirme en el “state of the art”, la punta de lanza, la vanguardia de las tecnologías de desarrollo para entornos Microsoft. Y eso ha marcado la diferencia: Desde ahí se entiende todo el crecimiento profesional que he experimentado a lo largo de estos meses de aplicación práctica de los conocimientos adquiridos en la carrera, así como de intensa investigación, descubrimiento, aprendizaje, sorpresa y disfrute.

Quizás la piedra angular de este proceso haya sido descubrir los *patrones de arquitectura* de la mano de MVC.

Me animé a usar esa variante de ASP.NET porque sonaba bien eso de un desacoplamiento máximo de la capa de presentación en un tipo de aplicaciones, las aplicaciones web, cuya naturaleza hace evidente que la capa de presentación “va aparte” (se renderiza en otra máquina y dentro de otro programa, el navegador).

Pero al usar el patrón de arquitectura, he alcanzado otro objetivo que no esperaba y que me ha resultado sorprendente: una limpieza y una claridad en la estructuración del código que me parece un gran logro en sí mismo.

Este logro ha ido también de la mano del esfuerzo que he realizado por el adecuado análisis y el cuidadoso diseño de la solución software. Para ello he tenido que pelearme con el UML, que nunca había usado “de verdad”, fuera de ejercicios de clase, con lo que también he aprendido muchísimo.

Para mi gusto, el exitoso resultado de todo este esfuerzo se aprecia en el que yo considero mi “diagrama estrella”: el Diagrama de Clases de Diseño, Ilustración 16, en la página 33, que sintetiza realmente la implementación completa, de un golpe de vista.

El resultado de usar MVC me ha sorprendido muy positivamente. Pero además ha logrado otro efecto inesperado: Al usar MVC, he descubierto la importancia de los patrones de arquitectura y de diseño; MVC no es el único, y se me ha despertado la curiosidad por indagar otros, cuáles son sus fines y objetivos, y cuáles y por qué pueden serme útiles en mi trabajo diario como desarrollador. De pronto se me ha despertado el interés por la arquitectura del software –con la que profesionalmente he tenido muy poco contacto– y por la Ingeniería del Software: Ahora quiero saber qué hay detrás de acrónimos como TDD, DDD o SOA (donde yo hasta ahora sólo tenía en mente que existiera la OOP). He usado y me ha encantado el patrón de diseño *Singleton*, del que ya había oído hablar en IS; pero, siguiendo en esta línea, ¿podría serme de utilidad en algunas situaciones de mi trabajo diario el *Multiton*? ¿Y qué es eso de las *Factory*, de las que he oído hablar bastante en mis investigaciones sobre Unit Testing? He usado en el proyecto la clase *CacheControlAttribute* de Craig Stuntz que funciona como un *Decorator* de la Action de un Controller: ¿No podría ser interesante aprender a utilizar esta técnica para extender la funcionalidad de una clase sin tener que derivarla?

Se me ha abierto un mundo.

Los otros dos grandes elementos de aprendizaje en este Trabajo han sido el *Entity Framework* y los Servicios WCF.

El Entity Framework es una manera poderosísima de acceder a datos.



Por una parte ayuda a eliminar el desajuste de impedancia entre aplicaciones y servicios de datos mediante la elevación del nivel de abstracción del nivel lógico (relacional) al nivel conceptual (entidad). Dicho en claro: Automatiza un mapeo entre tablas de una base de datos y clases del modelo de nuestra aplicación que nos permite a nosotros como programadores trabajar sólo teniendo en mente un paradigma OOP.

Esto en sí ya es precioso, limpísimo, poderosísimo y mágico.

Pero el otro gran logro que lleva aparejado el Entity Framework es elObjectContext, que tanto dolor de cabeza me ha dado y del que ahora estoy totalmente enamorado.

Yo hasta ahora sólo me había relacionado con bases de datos a través de ADO para VC++ 6.0.

En mis clases “repositorio” en C++ creaba una variable miembro privada “conexión”, que inicializaba y abría en el constructor mismo de “repositorio”, y que permanecía abierta a lo largo de toda la vida de la aplicación, para que las funciones de “repositorio” pudieran usarla.

Por analogía, mi primer impulso fue hacer para PGPWebmail un Repositorio Singleton con una propiedadObjectContext que sería la única que usaríamos para todo.

Los errores constantes a los que este enfoque me llevó (“new transaction is not allowed because there are other threads running in the session”) al principio me desquiciaron: “pues claro que hay otros hilos: ¡es una aplicación web, canastos!”; hasta que entendí la filosofía delObjectContext. Y entonces fue cuando me enamoré.

¿Qué necesidad había de tener unObjectContext permanentemente abierto, cargándose y cargándose con datos, cuando la estructura misma del MVC es tan “transaccional” (evento – FormCollection – Action – ViewData – return)?

Para cada necesidad de información que tengamos de la base de datos, creamos, usamos y destruimos unObjectContext.

Mucho más limpio, mucho más claro... Me enamoré.

Ya lo estoy aplicando en proyectos en mi trabajo.

En cuanto a los Servicios Web con WCF, después de seguir el curso<sup>18</sup> que me indicó Juan Carlos González, nuestro consultor, me quedé estupefacto al ver la facilidad con la que con un par de clics podíamos crear servicios, que luego pudieran alojarse en IIS o en la consola de servicios de Windows, que fácilmente pudieran soportar diversidad de protocolos de acceso, etc., así como la extraordinaria facilidad para crearles clientes consumidores.

Si a ello le sumamos mi nuevo interés por los patrones de arquitectura, que me ha llevado a leer sobre el SOA, creo que WPF va a empezar a formar parte de mis proyectos en el trabajo rápidamente.

EF y WCF me han abierto además al mundo más genérico de las “tecnologías”: Yo hasta ahora vivía en un mundo *lenguaje-céntrico*, donde saber programar era saber usar un lenguaje de programación y unas APIs y el resto me parecían “oropeles de márquetin”. De pronto haber usado EF y WCF me ha ayudado a entender la importancia de abrirme a descubrir toda la funcionalidad que pueden brindarme tecnologías que están al alcance de un clic mediante Visual Studio. Tengo que descubrir las ya existentes y estar al cabo de la calle sobre las nuevas que puedan salir, porque su incidencia en mi mejor trabajo cotidiano es real. Por eso me he suscrito a varios *newsletters* y a una revista en papel.

A lo largo de la implementación concreta ha habido muchos más descubrimientos emocionantes que ya voy trasladando a mi trabajo: las consultas sobre cualquier tipo de colecciones mediante LINQ, lo fácil y controlable que es lanzar Threads en .NET, la extraordinaria potencia de las expresiones regulares con Regex, la limpieza en la extensión de funcionalidad sin

---

<sup>18</sup> Microsoft (2011) “Curso de Servicios Web”. Microsoft World Wide Events [curso en línea]. [Fecha de consulta: 11 marzo 2011].

<https://msevents.microsoft.com/CUI/WebCastEventDetails.aspx?EventID=1032379565&Culture=es-ES>

derivación mediante los métodos de extensión, la emoción de poder programar tus propios “cast” (¡en C++ usamos tanto el casteo...!) mediante *operadores de conversión explícita*, la personalización del control de usuarios mediante la derivación del MembershipProvider y el uso de FormsAuthentication para autenticar mediante cookies, la magia del routing MVC, los distintos objetos que una Action puede devolver para que sea procesado por una View (View, RedirectToAction, JsonResult, FileResult, PartialViewResult, etc.), la brevedad y expresividad de Json para serializar objetos, la facilidad de servir vistas distintas según cuál sea el dispositivo desde el que se soliciten, el poder y la simplicidad de jQuery para manejar en cliente contenido DOM de la página visualizada, la potencia de las técnicas Ajax para suavizar la experiencia del usuario web y eliminar sobrecarga en el navegador y en la red, la simplicidad y potencia de las plantillas .aspx para la generación dinámica en servidor de HTML mediante marcado <% %>, la limpieza de que con MVC sólo pasemos a la vista la información imprescindible que necesitará para construir la página (el ViewData que le prepara la Action), la creación de *wrappers* en torno a aplicaciones de consola, el *workaround* para dar programáticamente permisos de escritura en carpetas NTFS, la posibilidad de personalizar instalaciones en Visual Studio mediante código en C# (con toda su potencia).

Ha sido un trabajo extenuante, utilizando tecnologías que me eran absolutamente desconocidas (mi perfil profesional es el de programador en C++, normalmente de aplicaciones sin interfaz gráfica), pero he aprendido muchísimas cosas y lo he pasado muy bien. Aprender es muy placentero.

### XIII. Bibliografía

---

#### 1. Publicaciones

---

- Bataller Díaz, Alfons** (2008). *Gestión y Desarrollo de Proyectos. Conceptos y sugerencias*. Barcelona: FUOC
- Campderrich Falgueras, Benet** (2004). *Recogida y documentación de requisitos*. Barcelona: FUOC
- Campderrich Falgueras, Benet; Recerca informàtica, S.L.** (2004). *UML: El modelo estàtico*. Barcelona: FUOC
- Campderrich Falgueras, Benet; Recerca informàtica, S.L.** (2004). *UML: El modelo dinámico*. Barcelona: FUOC
- Campderrich Falgueras, Benet; Recerca informàtica, S.L.** (2004). *Anàlisis orientado a objetos*. Barcelona: FUOC
- Campderrich Falgueras, Benet; Recerca informàtica, S.L.** (2004). *Diseño orientado a objetos*. Barcelona: FUOC
- Ceballos Villach, Jordi** (2009). *Introducción a .NET*. Barcelona: Universitat Oberta de Catalunya
- Domingo Ferrer, Josep** (2005). *Criptosistemas de clave pública*. Barcelona: FUOC
- Domingo Ferrer, Josep** (2005). *Firmas digitales*. Barcelona: FUOC
- Marguerie, Fabrice; Eichert, Steve; Wooley, Jim** (2008). *LINQ*. Madrid: Anaya Multimedia
- Rifà Pous, Helena** (2005). *Infraestructura de clave pública*. Barcelona: FUOC
- Sáenz Higuera, Nita; Vidal Oltra, Rut** (2008). *Redacción de documentos científico-técnicos*. Barcelona: FUOC

## 2. Cursos en línea

---

**Microsoft** (2011) “Curso de Servicios Web”. Microsoft World Wide Events [curso en línea]. [Fecha de consulta: 11 marzo 2011].

<https://msevents.microsoft.com/CUI/WebCastEventDetails.aspx?EventID=1032379565&Culture=es-ES>

**Microsoft** (2011, marzo) “Visual Studio 2010 and .NET Framework 4 Training Course”. MSDN [curso en línea]. [Fecha de consulta: 7 marzo 2011].

<http://msdn.microsoft.com/en-us/gg465334>

## 3. Artículos en línea

---

**Birke, Gabriel** (2011, junio) “jQuery Pagination Plugin”. GitHub [artículo en línea]. [Fecha de consulta: 23 abril 2011].

[https://github.com/gbirke/jquery\\_pagination](https://github.com/gbirke/jquery_pagination)

**Brett** (2010, mayo) “Encrypt/Decrypt string in .NET”. Stack Overflow [artículo en línea]. [Fecha de consulta: 17 abril 2011].

<http://stackoverflow.com/questions/202011/encrypt-decrypt-string-in-net>

**Eagle, Allan** (2009, enero) “Adding Save() functionality to Microsoft.Net.Mail.MailMessage”. The Code Project [artículo en línea]. [Fecha de consulta: 16 abril 2011].

<http://www.codeproject.com/KB/IP/smtplibtext.aspx>

**Gledhill, Mike** (2010, enero) “The installer was interrupted before Application could be installed...” [respuesta en foro]. [Fecha de consulta: 11 abril 2011].

<http://social.msdn.microsoft.com/Forums/en-US/winformssetup/thread/6bead7aa-cdc5-4b3d-af0c-82f246b2a3b8/>

**Hanselman, Scott** (2008, septiembre) “The Weekly Source Code 28 - iPhone with ASP.NET MVC Edition”. Scott Hanselman's ComputerZen.com [artículo en línea]. [Fecha de consulta: 8 junio 2011].

<http://www.hanselman.com/blog/TheWeeklySourceCode28iPhoneWithASPNETMVCEdition.aspx>

**Hanselman, Scott** (2010, noviembre) “A Better ASP.NET MVC Mobile Device Capabilities ViewEngine”. Scott Hanselman's ComputerZen.com [artículo en línea]. [Fecha de consulta: 24 marzo 2011].

<http://www.hanselman.com/blog/ABetterASPNETMVCMobileDeviceCapabilitiesViewEngine.aspx>

**Hexter, Eric y otros** (2010, agosto) “MvcContrib Template Gallery”. CodePlex [artículo en línea]. [Fecha de consulta: 1 abril 2011].

<http://mvcontribgallery.codeplex.com/>

**Huber SG, Peter** (2006, octubre) “POP3 Email Client with full MIME Support (.NET 2.0)”. The Code Project [artículo en línea]. [Fecha de consulta: 16 abril 2011].

<http://www.codeproject.com/KB/IP/Pop3MimeClient.aspx>

**Hui, Soon** (2008, noviembre) “Deploy ASP.NET MVC App on Windows XP (IIS 5.1)”. It's common sense, stupid [artículo en línea]. [Fecha de consulta: 25 marzo 2011].

<http://itscommonsensetupid.blogspot.com/2008/11/deploy-aspnet-mvc-app-on-windows-xp-iis.html>

**Kartmann, Emmanuel** (2003, septiembre) “Gnu Privacy Guard (GPG/PGP) for .NET [v1.0]”. The Code Project [artículo en línea]. [Fecha de consulta: 16 marzo 2011].

<http://www.codeproject.com/KB/security/gnupgdotnet.aspx>

**Lago, Ramiro** (2007, abril). “Patrón Modelo-Vista-Controlador”. Patrones de diseño software [artículo en línea]. [Fecha de consulta: 3 abril 2011].

<http://www.proactiva-calidad.com/java/patrones/mvc.html>

**Lewis, John Rudolf** (2009, agosto) “jQuery File Upload in ASP.NET MVC without using Flash”. ASP Zone [artículo en línea]. [Fecha de consulta: 10 mayo 2011].

<http://aspzone.com/tech/jquery-file-upload-in-asp-net-mvc-without-using-flash/>

**Microsoft** (2011) “101 LINQ Samples”. MSDN Visual C# Developer Center [artículo en línea]. [Fecha de consulta: 8 abril 2011].

<http://msdn.microsoft.com/en-us/vcsharp/aa336746>

**Microsoft** (2011) “Application Pool Identities”. IIS [artículo en línea]. [Fecha de consulta: 17 mayo 2011].

<http://learn.iis.net/page.aspx/624/application-pool-identities/>

**Microsoft** (2011) “Authenticating Users with Forms Authentication”. ASP.net [artículo en línea]. [Fecha de consulta: 28 marzo 2011].

<http://www.asp.net/mvc/tutorials/authenticating-users-with-forms-authentication-cs>

**Microsoft** (2011) “Cómo: Ejemplo de implementación del proveedor de pertenencia”. MSDN [artículo en línea]. [Fecha de consulta: 28 marzo 2011].

[http://msdn.microsoft.com/es-es/library/6tc47t75\(v=VS.100\).aspx](http://msdn.microsoft.com/es-es/library/6tc47t75(v=VS.100).aspx)

**Microsoft** (2011) “Elegir un protocolo de red. SQL Server 2008 R2”. MSDN [artículo en línea]. [Fecha de consulta: 21 marzo 2011].

<http://msdn.microsoft.com/es-es/library/ms187892.aspx>

**Microsoft** (2011) “Query Expression Syntax Examples: Ordering”. MSDN [artículo en línea]. [Fecha de consulta: 8 abril 2011].

<http://msdn.microsoft.com/en-us/library/bb738627.aspx>

**Microsoft** (2011) “Using login controls for an existing data source by creating a custom membership provider in ASP.NET 2.0”. Soporte Microsoft [artículo en línea]. [Fecha de consulta: 8 abril 2011].

<http://support.microsoft.com/kb/910440/en-us>

**Stuntz, Craig** (2010, junio) “How to stop MVC caching the results of invoking and action method?”. Stack Overflow [artículo en línea]. [Fecha de consulta: 23 abril 2011].

<http://stackoverflow.com/questions/2969450/how-to-stop-mvc-caching-the-results-of-invoking-and-action-method>

**Suttle, Ian** (2007, diciembre) “LINQ to SQL: Selecting a single row”. Ian Suttle's Blog [artículo en línea]. [Fecha de consulta: 8 abril 2011].

<http://www.iansuttle.com/blog/post/LINQ-to-SQL-Selecting-a-Single-Row.aspx>

**Swedberg, Karl** (2006, noviembre) “How to Get Anything You Want”. The jQuery Project [artículo en línea] [Fecha de consulta: 19 marzo 2011].

[http://docs.jquery.com/Tutorials:How\\_to\\_Get\\_Anything\\_You\\_Want](http://docs.jquery.com/Tutorials:How_to_Get_Anything_You_Want)

**Tomàs i Avellana, Eduard** (2009, junio) “ASP.NET MVC y Ajax: fácil no... facilísimo :)”. Burbujas en .NET [artículo en línea]. [Fecha de consulta: 15 marzo 2011].

<http://geeks.ms/blogs/etomas/archive/2009/06/30/asp-net-mvc-y-ajax-f-225-cil-no-facil-237-simo.aspx>

**Wikipedia** (2011, abril). “Model–view–controller” [artículo en línea]. [Fecha de consulta: 3 abril 2011].

<http://en.wikipedia.org/wiki/Model-view-controller>

**Wikipedia** (2011, marzo). “ASP.NET MVC Framework” [artículo en línea]. [Fecha de consulta: 3 abril 2011].

[http://en.wikipedia.org/wiki/Asp.net\\_mvc](http://en.wikipedia.org/wiki/Asp.net_mvc)