

# Memoria.

---

*Desarrollo de aplicaciones con ASP.NET MVC*

*Autor: Pablo Domínguez Cuevas*

*Consultor: Jordi Ceballos Villach*

*10 de enero de 2011*

*A mis petardos, Juan y Sofía.*

1.	Descripción del proyecto.....	7
1.1	Resumen.....	7
1.2	Alcance del proyecto.....	8
1.3	Objetivos del proyecto.....	8
1.4	Planificación del proyecto.....	9
1.4.1	Relación de actividades.....	9
1.4.2	Calendario de trabajo.....	14
1.4.3	Hitos principales.....	15
1.5	Resultados obtenidos.....	15
1.5	Productos entregados.....	16
1.6	Estructura del documento.....	16
2.	Arquitectura del proyecto: Análisis y diseño.....	17
2.1	Casos de uso.....	17
2.1.1	Nombre: Seleccionar tipo animal.....	17
2.1.2	Nombre: Seleccionar estado animal.....	17
2.1.3	Nombre: Listado animales.....	18
2.1.4	Nombre: Buscar animales.....	19
2.1.5	Nombre: Gestionar animales.....	19
2.1.6	Nombre: Enviar mensaje.....	21
2.1.7	Nombre: Gestionar mensajes.....	22
2.1.8	Nombre: Contestar mensaje.....	23
2.1.9	Nombre: Gestión reducida de animales.....	24
2.1.10	Nombre: Gestión usuarios.....	24
2.1.11	Nombre: Login.....	25
2.1.12	Nombre: Registrarse.....	26
2.2	Diagrama de casos de uso.....	27
2.3	Diseño del interfaz de usuario.....	27
2.3.1	Diagramas de actividad.....	28
2.3.1.1	Listado de animales.....	28
2.3.1.2	Gestión usuarios.....	29
2.3.1.3	Registrar.....	29
2.3.1.4	Gestionar mensajes.....	30

2.3.1.5 Gestionar animales.....	30
2.3.1.6 Contestar mensaje. ....	31
2.3.1.7 Enviar mensaje. ....	31
2.4 Diseño capa presentación. ....	31
2.4.1 Diseño externo. ....	31
2.4.2 Pantallas del interfaz de usuario. ....	32
2.4.2.1 Parte común a todas las pantallas. ....	32
2.4.2.2 Listado de animales.....	32
2.4.2.3 Pantalla de gestión de animales.....	34
2.4.2.4 Pantalla de edición del perfil del animal. ....	35
2.4.2.5 Pantalla para consultar el perfil del animal.....	36
2.4.2.6 Pantalla para enviar mensaje a otro usuario. ....	37
2.4.2.7 Pantalla de gestión de mensajes recibidos. ....	38
2.4.2.8 Pantalla de detalles del mensaje.....	39
2.4.2.9 Pantalla de registro de un nuevo usuario. ....	39
2.5 Diseño de la capa de dominio. ....	41
2.5.1 Diagrama de componentes. ....	41
2.5.1.1 Diagrama de componentes WebForms.....	41
2.5.1.2 Diagrama de componentes para MVC. ....	42
2.5.2 Modelo conceptual: diagrama de clases UML. ....	43
2.6 Arquitectura del proyecto.....	44
2.6.1 Arquitectura WebForms.....	45
2.6.2 Arquitectura MVC (Modelo2).....	45
3. Implementación. ....	46
3.1 Desarrollar web MVC. ....	46
3.1.1 Proyecto tibisit.Domain.....	46
3.1.2 Proyecto tibisit.Db.....	46
3.1.3 Proyecto tibisit.mvc.Web .....	46
3.1.4 Proyecto tibisit.test .....	46
3.1.4 Problemas con las clases derivadas. ....	47
3.1.5 Ninject: control de dependencias. ....	47
3.1.6 Primer controlador y vista. Numerar páginas. ....	47

3.1.6.1	Acción List del controlador.....	47
3.1.6.2	PagingInfo.....	48
3.1.6.3	HtmlHelpers: PageLinks.....	48
3.1.6.4	Modelo.....	48
3.1.6.5	Código final.....	48
3.1.6.6	Vista.....	48
3.1.7	Página master.....	49
3.1.7.1	MS Expression Web 4.....	49
3.1.7.2	Buscar página master.....	49
3.1.7.3	Modificar página master.....	49
3.1.8	Crear página de inicio.....	49
3.1.8.1	Menú derecha. EstadoController.....	50
3.1.9	Crear resto de controladores para visualizar animales.....	50
3.1.10	Añadir controlador para gestión de animales.....	51
3.1.10.1	Crear controlador.....	51
3.1.10.2	Menú derecha.....	51
3.1.10.3	Acción Edit y Create.....	51
3.1.10.4	AdoptarEditarListViewModel.....	51
3.1.10.5	Vista Edit.....	51
3.1.10.6	Errores inesperados.....	51
3.1.10.7	Validación.....	52
3.1.10.8	Subir imágenes al servidor.....	53
3.1.11	Mostrar imágenes de la base de datos.....	54
3.1.12	Definir el resto de controladores de gestión.....	55
3.1.13	Opción de ampliar la información del animal.....	55
3.1.13.1	Acción Show.....	55
3.1.13.2	Vista Show.....	55
3.1.13.3	Añadir botón que llama a la acción Show.....	55
3.1.14	Borrar animales.....	56
3.1.14.1	Añadir botón en vista.....	56
3.1.14.2	jQuery UI dialog.....	56
3.1.15	Gestión de mensajes.....	57

3.1.15.1	Posibilidad de enviar mensajes sobre un animal. ....	57
3.1.15.2	Acción EnviarMensaje. ....	57
3.1.15.3	jQuery UI dialog. ....	57
3.1.15.4	Controlador MensajeController. ....	57
3.1.15.5	Modelo MensajeListViewModel. ....	58
3.1.15.6	Vistas. ....	58
3.1.16	Autenticación. ....	59
3.1.16.1	Controladores. ....	59
3.1.16.2	Vistas. ....	59
3.1.16.3	Autorize. ....	59
3.2	Desarrollar web mediante WebForms. ....	60
3.2.1	Página master. ....	60
3.2.2	Página de inicio. ....	60
3.2.3	Página para mostrar animales. ....	60
3.2.4	Página de gestión de animales. ....	61
3.2.4.1	Validación. ....	61
3.2.5	Gestión de mensajes. ....	61
3.2.6	Autenticación. ....	61
3.2.7	Control de errores. ....	61
4.	Hosting: Problemas y soluciones. ....	62
4.1	Problemas. ....	62
4.1.1	Problemas de mapeo. ....	62
4.1.2	Problemas con el estado de sesión. ....	62
4.1.2.1	Guardar estado en SQL Server. ....	62
4.1.3	Problemas con la autenticación. ....	62
4.1.4	Error interno del servidor. ....	63
4.1.4.1	ErrorController. ....	63
4.2	Modificaciones para solucionar problemas. ....	63
4.2.1	Caché en disco. ....	63
4.2.2	Session State. ....	64
4.2.3	CDN. ....	64
4.2.4	Watermark. ....	64

4.2.5	Enviar email con errores. ....	64
4.2.6	Dispose. ....	64
4.3	Windows Azure Platform. ....	64
4.3.1	Principales modificaciones. ....	65
4.3.1.1	Modificaciones propias. ....	65
4.3.1.2	Guardar estado de sesión. ....	65
4.3.1.3	Subir imágenes al servidor. ....	65
4.3.1.4	SQL Azure. ....	65
5.	Diferencias MVC y WebForms. ....	65
5.1	MVC. ....	65
5.2	Ventajas MVC. ....	66
5.3	Inconvenientes MVC. ....	67
5.4	Ventajas WebForms. ....	67
5.5	Inconvenientes WebForms. ....	68
5.6	Principales diferencias. ....	68
5.7	Experiencia personal. ....	69
6.	Conclusiones. ....	70
7.	Bibliografía. ....	70
	Pro ASP.NET MVC 2 Framework, Second Edition [Kindle Edition] .....	70
	Professional Application Lifecycle Management with Visual Studio 2010 [Kindle Edition] .....	70
	Entity Framework 4.0 Recipes: A Problem-Solution Approach [Kindle Edition] .....	70
	jQuery Cookbook [Kindle Edition] .....	71
	Microsoft® .NET: Architecting Applications for the Enterprise [Kindle Edition] .....	71
	CODE Magazine - 2010 MarApr [Kindle Edition] .....	71
	Programming Windows Azure [Kindle Edition] .....	71
	Foundation Expression Blend 4 with Silverlight [Kindle Edition] .....	71

## 1. Descripción del proyecto.

### 1.1 Resumen.

Este proyecto tiene como objetivo principal el aprendizaje de las tecnologías .NET de Microsoft, y fundamentalmente el análisis del framework MVC disponible para ASP.NET, y para ello se detallarán las principales ventajas y desventajas del mismo, comparándolo con WebForms.

Para el aprendizaje de estas tecnologías se ha desarrollado una web cuya funcionalidad es que los usuarios puedan publicar referencias a animales que desean dar, para que otros puedan adoptarlos. Podríamos hablar de un portal destinado a evitar el abandono de los animales a favor de su adopción.

Entre las funcionalidades principales implementadas en esta web podríamos destacar:

La web desarrollada consiste en un portal donde los usuarios puedan realizar una serie de acciones, en favor de localizar un animal.

- Un usuario podrá, tras registrarse, dar de alta su perro o gato, para que otros puedan adoptarlo.
- Un usuario podrá, tras registrarse, dar de alta su perro o gato, informando que desea cruzarlo.
- Un usuario podrá, tras registrarse, dar de alta su perro o gato, informando que lo ha perdido.
- Un usuario podrá, tras registrarse, dar de alta un perro o gato, informando que lo ha encontrado perdido.
- Los usuarios anónimos podrán ver todos estos animales clasificados, de modo, que podrá buscar contenido según la situación del animal: para adoptar, para cruzar, perdido o encontrado perdido.
- Los usuarios podrán enviarse mensajes entre sí, una vez registrados, habrá un buzón, donde el usuario podrá ver todos sus mensajes recibidos.
- El usuario no autenticado podrá ver toda la información de perros y gatos, pero no podrá gestionar mensajes ni animales.

El proyecto ha ido cumpliendo una serie de entregas, hasta terminar con esta última, cuya fecha límite es el 10 de enero de 2011.

El proyecto ha sido realizado por Pablo Domínguez Cuevas y ha sido tutelado por el consultor Jordi Ceballos Villach.

Una vez finalizado el proyecto, lo pondré en producción, esperando sirva de ayuda a la comunidad de usuarios de internet con el encomiable fin de evitar el abandono y favorecer la adopción de animales, evitando así su maltrato y muerte.

### 1.2 Alcance del proyecto.

El proyecto abarca desde el aprendizaje de las tecnologías ASP.NET de Microsoft y el patrón MVC, hasta el desarrollo de la web utilizando WebForms y la misma pero utilizando MVC, de modo que se puedan analizar las ventajas y desventajas de cada uno de ellos. En un primer momento se establecieron las principales tareas como:

- Buscar bibliografía para el estudio de estas tecnologías.
- Analizar y comprender a éstas, para poder llevar a cabo el desarrollo de la web.
- Analizar y diseñar la aplicación web.
- Desarrollar la aplicación utilizando MVC.
- Desarrollar la aplicación utilizando WebForms.
- Estudiar las diferencias entre ambos modelos.
- Realizar memoria y presentación del proyecto realizado.

### 1.3 Objetivos del proyecto.

El objetivo principal de este PFC es el de aplicar los conocimientos adquiridos durante el estudio de la Ingeniería Informática, llevando a cabo para ello, el desarrollo de un proyecto elegido por el estudiante. En concreto, este proyecto va dirigido a aplicarlos mediante el uso de la tecnología .Net de Microsoft, y más en concreto, desarrollando una aplicación con ASP.NET MVC y otra con WebForms, analizando las ventajas e inconvenientes de cada uno de ellos.

Para poder cumplir estos objetivos, he procurado utilizar las máximas tecnologías de Microsoft y otras que me han servido de apoyo, como jQuery. Las bases para el desarrollo han sido:

- Como IDE, MS Visual Studio 2010 Ultimate y como sistema de gestión de base de datos MS SQL Server 2008.
- Una de las web utiliza el framework ASP.NET MVC 2 y la otra WebForms, siempre aplicando el nuevo framework 4. En ambos casos, para aumentar la usabilidad, he utilizado técnicas AJAX mediante jQuery UI.
- Para el acceso a datos he optado por el ADO.NET Entity Framework 4 junto a LINQ. Además para mantener la separación entre capas, he utilizado objetos POCO, que son los que se intercambian entre las distintas capas, de modo, que la capa de presentación, no tenga ninguna referencia a la de datos. Para los métodos CRUD, he

usado repositorios, que son los que implementan toda la funcionalidad necesaria para que la información de los objetos POCO se guarde en la capa de persistencia.

- Aun no siendo necesario para este PFC, también he aplicado técnicas TDD, es decir, realizar primero el test de prueba, y posteriormente mediante refactoring implementar el método que permita pasar el test correctamente. Pero por motivos de tiempo, no he podido desarrollarlo como hubiese querido.
- Respecto al punto anterior, también he usado librerías “mock”, para no acceder directamente al repositorio de datos en los test.
- Como repositorio de código y para el seguimiento del proyecto he usado TFS 2010 (Team Foundation Server 2010) con la plantilla MSF Agile 5.0, y Sharepoint 2010.
- Para el diseño de la página master me he ayudado de MS Expression Web 4. Podría haber seguido con esta herramienta para el resto del diseño, pero actualmente, tiene poco integración con aplicaciones MVC.
- Como programas auxiliares para la confección de la documentación y gestión del proyecto, he usado MS Project 2010, MS Visio 2010, MS PowerPoint 2010 y MS Word 2010.

## 1.4 Planificación del proyecto.

### 1.4.1 Relación de actividades.

<b>Realizar plan de trabajo</b>	
<b>objetivo</b>	Este documento inicial ayuda a tener una primera visión del proyecto a realizar.
<b>duración</b>	6 días
<b>entregables</b>	Documento de plan de trabajo.
<b>Observaciones</b>	Este plan de trabajo ha sido bastante exacto a pesar de que nunca había utilizado las tecnologías. Las principales variaciones se han debido a la publicación de la web en los distintos hosting: Winhost y Windows Azure Platform.

<b>Buscar bibliografía</b>	
<b>objetivo</b>	Las tecnologías a utilizar no eran todas conocidas, por lo que primero he buscado bibliografía sobre éstas, estudiado y analizado.

<b>duración</b>	6 días
<b>entregables</b>	
<b>Observaciones</b>	La mayor parte del tiempo de este proyecto se ha gastado en documentarme.

<b>Gestión del proyecto</b>	
<b>objetivo</b>	En general la previsión inicial se cumplió bastante bien.
<b>duración</b>	1 día
<b>entregables</b>	Plan de proyecto actualizado
<b>Observaciones</b>	La web desarrollada con WebForms la pude terminar antes de tiempo, por lo que pude dedicar tiempo a las distintas adaptaciones para publicar las web en el hosting.

<b>Requerimientos H/S</b>	
<b>objetivo</b>	Estudiar los requerimientos necesarios para llevar a cabo la web.
<b>duración</b>	1 día
<b>entregables</b>	Documento de requerimientos.
<b>Observaciones</b>	Tras el análisis decidí utilizar el framework 4 para la realización del proyecto.

<b>Análisis funcional</b>	
<b>objetivo</b>	Realizar modelos de caso de uso, diagrama estático o modelo conceptual, definir la interfaz gráfica del usuario.
<b>duración</b>	5 días
<b>entregables</b>	Documentos resultantes de estos análisis.
<b>Observaciones</b>	Revisé los apuntes de las asignaturas que tratan estos temas.

<b>Diseño de la capa de presentación</b>	
<b>objetivo</b>	Realizar diseño externo e interno de la aplicación.
<b>duración</b>	5 días
<b>entregables</b>	Pantallas de la aplicación y diseño de vistas, controladores y el modelo.
<b>Observaciones</b>	Revisé los apuntes de las asignaturas que tratan estos temas.

<b>Diseño de la capa de dominio</b>	
<b>objetivo</b>	Diseño de las clases del modelo conceptual.
<b>duración</b>	5 días
<b>entregables</b>	Modelo conceptual.
<b>Observaciones</b>	Revisé los apuntes de las asignaturas que tratan estos temas.

<b>Diseño del modelo lógico de la base de datos</b>	
<b>objetivo</b>	Realizar el modelo E/R para almacenar los datos
<b>duración</b>	5 días
<b>entregables</b>	Modelo lógico.
<b>Observaciones</b>	Revisé los apuntes de las asignaturas que tratan estos temas como BD I, BD II y SGBD.

<b>Programación de la web usando MVC</b>	
<b>objetivo</b>	Utilizar esta arquitectura para poder analizar las ventajas y desventajas del patrón MVC.
<b>duración</b>	25 días
<b>entregables</b>	Código fuente de la web.
<b>Observaciones</b>	Esta primera web ha consumido más recursos que la segunda, ya

que es mi primer contacto con esta tecnología.

#### Pruebas de la web usando MVC

<b>objetivo</b>	Asegurarse el correcto funcionamiento de la misma.
<b>duración</b>	2 días
<b>entregables</b>	Documento de errores y aceptación de la aplicación.
<b>Observaciones</b>	La web se abrió a internet en fase beta, publicándola en un proveedor de hosting, de este modo amigos y familiares pudieron colaborar en las pruebas de la misma.

#### Programación de la web usando WebForms

<b>objetivo</b>	Utilizar esta arquitectura para poder comparar con la MVC
<b>duración</b>	12 días
<b>entregables</b>	Código fuente de la web.
<b>Observaciones</b>	Se aprovechó gran parte del código de la web anterior, por lo que el tiempo de programación fue visiblemente más ajustado.

#### Pruebas de la web usando WebForms

<b>objetivo</b>	Asegurarse el correcto funcionamiento de la misma.
<b>duración</b>	2 días
<b>entregables</b>	Documento de errores y aceptación de la aplicación.
<b>Observaciones</b>	La web se abrió a internet en fase beta, publicándola en un proveedor de hosting, de este modo amigos y familiares pudieron colaborar en las pruebas de la misma.

#### Publicar ambas web en un proveedor hosting

<b>objetivo</b>	Hacer accesible la web a todos los usuarios y permitir al consultor y
-----------------	---

	al jurado un fácil acceso a la misma.
<b>duración</b>	8 días
<b>entregables</b>	
<b>Observaciones</b>	He tenido que realizar múltiples cambios en el proyecto para que funcionase en el hosting de Winhost debido a la limitación de memoria del proceso que me da este proveedor.

<b>Adaptar proyecto MVC a Windows Azure Platform</b>	
<b>objetivo</b>	Publicar la web en un hosting sin limitaciones y escalable.
<b>duración</b>	13 días
<b>entregables</b>	
<b>Observaciones</b>	Tras los problemas encontrados en Winhost, por limitaciones en la memoria de proceso, decido adaptar la web MVC con un proveedor fiable y sin limitaciones, como Windows Azure Platform, ya que se paga por uso, pudiendo escalar a las necesidades del cliente.

<b>Analizar las diferencias entre ambos modelos</b>	
<b>objetivo</b>	Conocer las ventajas y desventajas de un modelo sobre el otro.
<b>duración</b>	1 día
<b>entregables</b>	Documento de análisis de los modelos utilizados.
<b>Observaciones</b>	En este punto las tecnologías ya son conocidas.

<b>Realizar memoria del proyecto</b>	
<b>objetivo</b>	Describir el trabajo realizado para que pueda ser valorado por el consultor y el tribunal.
<b>duración</b>	2 días
<b>entregables</b>	Documento de memoria del proyecto.

<b>Observaciones</b>	Para su correcta realización estudié los materiales que se nos proporcionan en la asignatura para la elaboración de documentos técnicos.
----------------------	--

<b>Realizar presentación</b>	
<b>objetivo</b>	Describir el trabajo realizado para que pueda ser valorado por el consultor y el tribunal en una presentación.
<b>duración</b>	1 día
<b>entregables</b>	Presentación virtual del proyecto.
<b>Observaciones</b>	Opto por utilizar Demo Builder y MS PowerPoint 2007.

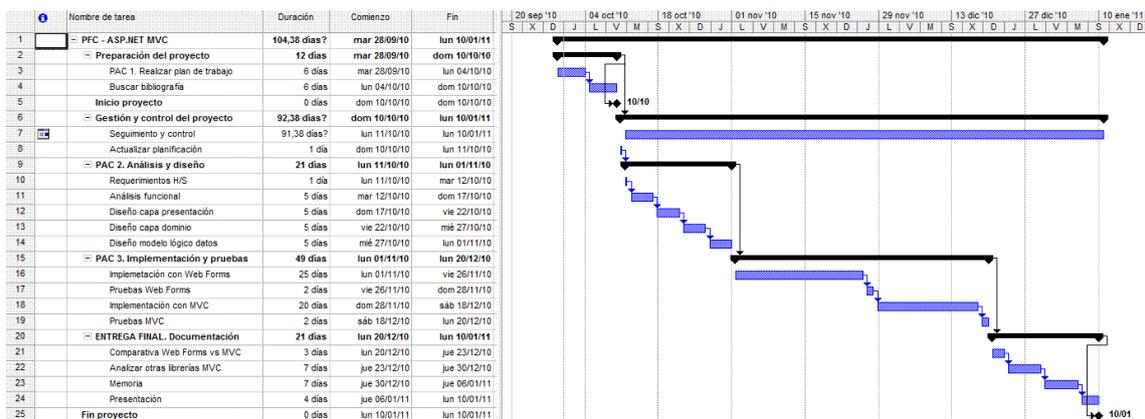
### 1.4.2 Calendario de trabajo.

El calendario para realizar el proyecto no ha tenido festivos y está definido para trabajar 8 horas diarias.

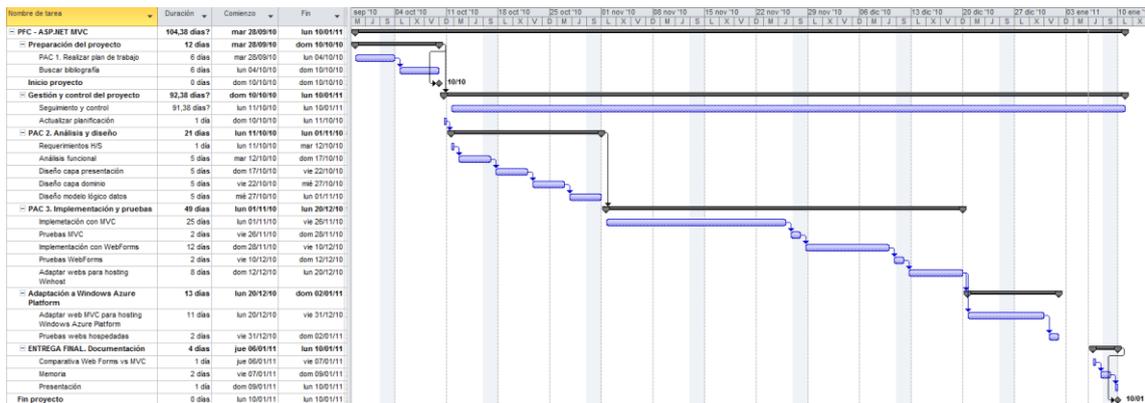
A pesar de que la asignatura comenzó el 21 de septiembre de 2010, he decidido establecer la fecha de inicio del proyecto en el momento en el que el consultor me explicó exactamente en qué iba a consistir mi PFC, y por tanto, cuando pude comenzar a planificar el trabajo.

La fecha límite para la entrega final ha sido el 10 de enero de 2011.

Planificación inicial:



Proyecto final:



### 1.4.3 Hitos principales.

A partir de la planificación del punto anterior, considero como hitos más relevantes los que han coincidido con la entrega de cada PAC.

Fecha	Descripción hito
28/09/2010	Preparación del proyecto.
04/10/2010	PAC 1. Entregar plan de trabajo.
10/10/2010	Inicio del proyecto.
01/11/2010	PAC 2. Entregar análisis y diseño de la aplicación web.
20/12/2010	PAC 3. Entregar aplicación web utilizando WebForms y MVC.
10/01/2011	ENTREGA FINAL. Entregar memoria y presentación virtual.
10/01/2011	Fin del proyecto.

### 1.5 Resultados obtenidos.

Una vez completado el proyecto se ha conseguido:

A nivel de formación académica: disponer de los conocimientos técnicos necesarios para la realización del PFC. Básicamente haber repasado todos los conocimientos aprendidos durante estos años en las distintas asignaturas de la UOC y principalmente haber utilizado las tecnologías .Net de Microsoft como ASP.NET utilizando WebForms y MVC.

A nivel TIC: se ha dispuesto del equipamiento necesario que contempla las diferentes tecnologías que se han utilizado en el proyecto. Se ha encontrado un proveedor de hosting como Winhost, capaz de albergar la web con sus requisitos técnicos, pero debido a las

limitaciones del mismo y el presupuesto, finalmente opté por Windows Azure Platform, mucho más abierta, profesional y ajustable a las necesidades de cada proyecto.

A nivel de negocio: aunque no forma parte de este proyecto, espero que la web dada su temática y a pesar de ser totalmente altruista, pudiera además de ayudar a los animales, ser una fuente de ingresos en publicidad.

## 1.5 Productos entregados.

Los entregables que se han realizado durante el PFC han sido:

- Documento de plan de trabajo.
- Documento de requisitos y análisis preliminar:
  - Casos de uso.
  - Diseño del interfaz de usuario.
- Documento de diseño de la capa de presentación:
  - Diseño externo.
- Documento de la capa de dominio:
  - Diagrama de componentes.
  - Modelo conceptual: diagrama de clases UML.
  - Modelo conceptual: diagrama de clases de Visual Studio.
- Documento de diseño de la capa de servicios técnicos: modelo lógico de la base de datos.
- Documento de arquitectura del proyecto: WebForms y MVC (modelo 2).
- Manual de instalación del PFC para el consultor, a pesar de que ambas web, han sido hospedadas en internet para que sean fácilmente accesibles. La web MVC se encuentra en Windows Azure Platform y la web WebForms en el proveedor de hosting Winhost.
- Memoria y presentación virtual del PFC.

## 1.6 Estructura del documento.

Este documento pretende recoger las principales tareas realizadas durante el desarrollo del proyecto, así como los problemas encontrados y las soluciones propuestas, sobre todo al hospedar las webs en distintos proveedores de hosting.

También se analizarán las principales diferencias encontradas entre WebForms y MVC.

## 2. Arquitectura del proyecto: Análisis y diseño.

### 2.1 Casos de uso.

El diagrama de casos de uso permite ver de modo gráfico la funcionalidad del sistema. Muestra quién usa el sistema y qué puede hacer en él. El diagrama no muestra detalles de los casos de uso por sí mismos, pero en su lugar muestra una vista de los casos de uso, los actores y los sistemas que participan.

#### 2.1.1 Nombre: **Seleccionar tipo animal.**

Funcionalidad: filtro previo, permite seleccionar el tipo de animal que vamos a tratar.

Acciones:

1. El usuario elegirá el tipo de animal: perros o gatos.
2. El sistema mostrará la pantalla “seleccionar estado animal”.

Pantallas:

- Seleccionar tipo animal.

Datos solicitados:

Seleccionar perros o gatos.

#### 2.1.2 Nombre: **Seleccionar estado animal.**

Funcionalidad: permite seleccionar el estado de los animales que queremos ver (se adoptan, perdidos, se cruzan, adoptados y encontrados).

Casos de uso relacionados:

- Está incluido en “Seleccionar tipo animal”.

Acciones:

1. El usuario elegirá el estado de animal: se adoptan, perdidos, se cruzan, adoptados y encontrados.
2. El sistema mostrará la siguiente pantalla “listado animales” mostrando solo los registros del tipo seleccionado.

Pantallas:

- Pantalla de selección de estado. Realmente será un menú que estará incluido en la pantalla de listado de animales y en la de gestión de animales.

Datos mostrados:

Se muestran los estados: se adoptan, perdidos, se cruzan, y encontrados.

Datos solicitados:

Seleccionar alguno de estos estados.

### 2.1.3 Nombre: Listado animales.

Funcionalidad: permite visualizar todos los animales con sus datos principales.

Acciones principales:

1. El usuario elegirá el estado de animal (caso de uso incluido “Seleccionar estado animal”), mostrándose sólo los animales de ese estado.

Acciones alternativas:

2. El usuario podrá realizar búsquedas sobre los animales por los campos que corresponden con cada uno de los estados del animal. Caso de uso incluido “Buscar animal”.
  - a. En caso de seleccionar la búsqueda, el sistema refrescará la pantalla “listado animales” con los que cumplan los filtros.
3. El usuario podrá seleccionar un animal para ver el detalle del mismo.
  - a. Si selecciona un animal, el sistema muestra la pantalla “Perfil del animal”.

Pantallas:

- Listado de animales.

Datos a mostrar:

Mostrar en una rejilla (grid) los datos principales de los animales según los filtros anteriores (tipo y estado), dependiendo del estado se mostrarán unos datos u otros:

Si estado es “se adopta” o “se cruza” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto y provincia y CP del usuario.

Si estado es “perdido” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto, fecha de pérdida, recompensa y provincia y CP donde se perdió.

Si estado es “encontrado” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto, fecha en que se encontró, y provincia y CP del usuario.

Los animales se mostraran ordenados por prioridad: máxima, alta, normal y baja (esta prioridad solo podrá asignarla el administrador) y por fecha de creación descendente. La prioridad se le asignará al usuario, de modo, que todos sus animales hereden dicha prioridad, aunque será posible cambiarla individualmente a cada animal.

- Búsqueda animales.

- Perfil del animal

Datos a mostrar:

Si estado es “se adopta” o “se cruza” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto y provincia y CP del usuario.

Si estado es “perdido” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto, fecha de pérdida, recompensa y provincia y CP donde se perdió.

Si estado es “encontrado” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto, fecha en que se encontró, y provincia y CP del usuario.

#### **2.1.4 Nombre: Buscar animales.**

Funcionalidad: permite visualizar en la pantalla de listado de animales todos los animales que cumplen con los filtros indicados.

Casos de uso relacionados:

- Está incluido en “Listado animales”.

Acciones principales:

1. El usuario rellenará los filtros que desee para ajustar su búsqueda.
2. El sistema refrescará los datos la pantalla “Listado animales” según los filtros introducidos.

Pantallas:

- Búsqueda animales

Datos solicitados:

Si estado es “se adopta”, “se cruza”, o “adoptado” poder filtrar por sexo, raza, tamaño, color, y provincia y CP del usuario.

Si estado es “perdido” poder filtrar por sexo, raza, tamaño, color, peso, y provincia y CP del lugar donde se perdió.

Si estado es “encontrado” poder filtrar por sexo, raza, tamaño, color, peso, y provincia y CP del lugar donde se encontró.

#### **2.1.5 Nombre: Gestionar animales.**

Funcionalidad: permite gestionar al usuario los datos de sus animales. Mostrará una rejilla con los datos principales del animal, pero sólo con los animales del usuario. Este caso de uso no está disponible para los animales cuyo estado sea “adoptado”.

Casos de uso relacionados:

- Está incluido en “Seleccionar estado animal”.

#### Acciones principales:

1. El sistema muestra la pantalla “gestión de animales”.
2. El usuario podrá seleccionar un animal para editar sus datos.
3. El usuario podrá dar altas de nuevos animales.

#### Acciones alternativas:

2a) el sistema muestra la pantalla de “perfil animal a gestionar”.

2a.1) Si el usuario “guarda” los cambios, mostramos mensaje de validación.

2a.2) si el usuario borra el animal, el sistema debe volver al punto 1, refrescando los datos.

3a) el sistema muestra la ventana “alta de un nuevo animal”.

3a.1) si el usuario selecciona el botón “adoptado”, el sistema rellenará la fecha de adopción que podrá ser modificada por el usuario, así como el alias del usuario que adopta.

3a.2) al “grabar”, mostraremos mensaje de validación y si todo es correcto volvemos al punto 1, refrescando los datos.

#### Pantallas:

- Listado de animales a gestionar

#### Datos a mostrar:

Mostrar en una rejilla (grid) los datos principales de los animales según los filtros anteriores (tipo y estado), dependiendo del estado se mostrarán unos datos u otros:

Si estado es “se adopta” o “se cruza” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto y provincia y CP del usuario.

Si estado es “perdido” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto, fecha de pérdida, recompensa y provincia y CP donde se perdió.

Si estado es “encontrado” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto, fecha en que se encontró, y provincia y CP del usuario.

- Perfil del animal a gestionar

#### Datos solicitados:

Si estado es “se adopta” o “se cruza” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto y provincia y CP del usuario.

Si estado es “perdido” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto, fecha de pérdida, recompensa y provincia y CP donde se perdió.

Si estado es “encontrado” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto, fecha en que se encontró, provincia y CP donde se encontró.

- Alta de un nuevo animal

Datos solicitados:

Si estado es “se adopta” o “se cruza” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto.

Si estado es “perdido” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto, fecha de pérdida, recompensa y provincia y CP donde se perdió.

Si estado es “encontrado” mostrar: Nombre, sexo, raza, tamaño, pelo, color, peso, foto, fecha en que se encontró, provincia y CP donde se encontró.

Actores:

- Usuario

### **2.1.6 Nombre: Enviar mensaje.**

Funcionalidad: permite a un usuario enviar un mensaje al cuidador del animal, indicándole que está interesado en el mismo.

Casos de uso relacionados:

- Extiende el caso de uso “Lista animales”.
- Extiende el caso de uso “Contestar mensaje”.
- Incluye el caso de uso “Login”, es decir, el usuario debe estar registrado en el sistema.

Acciones principales (“Lista animales”):

1. El sistema muestra la pantalla “Perfil del animal”.
2. Si el usuario selecciona “enviar mensaje” el sistema mostrará la pantalla “enviar mensaje”.

Acciones principales (“Contestar mensaje”):

1. El sistema muestra la pantalla “Contestar mensaje”.
2. Si el usuario selecciona “contestar mensaje” el sistema mostrará la pantalla “enviar mensaje”.

Acciones alternativas:

2a) Si el usuario “cancela” el caso de uso finaliza.

2b) Si envía volvemos al punto 1 indicando que se ha realizado el envío correctamente.

Pantallas:

- Perfil del animal

Datos a mostrar:

Además de los definidos en el caso de uso que extiende, tendrá un botón que permita el envío de mensajes.

- Enviar mensaje

Datos a mostrar:

Alias de quién envía y a quién va destinado, así como el animal al que hace referencia.

Datos solicitados:

Asunto y cuerpo del mensaje.

Actores:

- Usuario

### **2.1.7 Nombre: Gestionar mensajes.**

Funcionalidad: permite a un usuario gestionar todos sus mensajes recibidos.

Casos de uso relacionados:

- Incluye el caso de uso “Login”, es decir, el usuario debe estar registrado en el sistema.

Acciones principales:

1. El usuario visualiza la lista de sus mensajes recibidos.
2. El usuario podrá seleccionar un mensaje para abrirlo (leerlo).
3. El sistema mostrará la pantalla “detalles mensaje”

Acciones alternativas:

3a) Si el usuario “abre” el mensaje, el sistema lo marcará como leído.

3b) el usuario podrá eliminar el mensaje.

3b.1) al eliminar, el sistema debe volver al punto 1, refrescando los datos.

#### Pantallas:

- Gestión de mensajes

#### Datos a mostrar:

Mostrar en una rejilla (grid) todos los mensajes recibidos, indicando remitente (alias), fecha envío, id y nombre del animal, asunto y marca de leído.

- Detalles mensaje

#### Datos a mostrar:

Remitente (alias), fecha envío, id y nombre del animal, asunto y mensaje.

#### Actores:

- Usuario

### **2.1.8 Nombre: Contestar mensaje.**

Funcionalidad: permite responder a un mensaje.

#### Casos de uso relacionados:

- Extiende el caso de uso “Gestión mensajes” en el punto 3.

#### Acciones principales:

1. El sistema muestra la pantalla “contestar mensaje”.
2. El sistema rellenará los campos y en el asunto indicará “RE:” + el asunto del mensaje que se contesta.

#### Acciones alternativas:

2a) el usuario rellena los datos y si “envía”:

2a.1) el sistema validará el mensaje, comprobando que ni asunto ni cuerpo estén vacíos.

2b) si el usuario “cancela” el caso de uso finaliza.

#### Pantallas:

- Contestar mensaje

Datos a mostrar:

Se mostrará un nuevo mensaje con los campos remitente, destinatario y animal al que hace referencia el mensaje, el asunto indicará "RE:" + el asunto del mensaje que se contesta.

Datos solicitados:

Se solicitarán el asunto y cuerpo del mensaje.

Actores:

- Usuario

### **2.1.9 Nombre: Gestión reducida de animales.**

Funcionalidad: permite al administrador gestionar los animales que desee. Mostrará una rejilla con los datos principales del animal. El administrador sólo podrá eliminar el animal o modificar su prioridad.

Casos de uso relacionados:

- Incluye el caso de uso "Login", por lo que hay que estar registrado.

Acciones principales:

1. El sistema muestra la pantalla "gestión reducida animales".
2. El administrador podrá borrar desde la propia rejilla.
3. El administrador podrá modificar la prioridad.

Pantallas:

- Gestión reducida animales

Datos a mostrar:

Mostrar en una rejilla (grid) los datos principales de los animales: Id, Fecha alta, Foto, Nombre, Raza, Sexo y Cuidador (email y alias).

Actores:

- Administrador

### **2.1.10 Nombre: Gestión usuarios.**

Funcionalidad: permite al administrador gestionar los usuarios del sistema. Mostrará una rejilla con los datos principales del usuario. El administrador sólo podrá eliminar el usuario o modificar su prioridad.

Casos de uso relacionados:

- Incluye el caso de uso “Login”, por lo que hay que estar registrado.

Acciones principales:

1. El sistema muestra la pantalla “gestión usuarios”.
2. El administrador podrá borrar desde la propia rejilla.
3. El administrador podrá modificar la prioridad.

Acciones alternativas:

2a) el sistema deberá refrescar los datos tras el borrado.

3a) será necesario grabar tras modificar la prioridad, para guardar los cambios.

Pantallas:

- Listado de usuarios.

Datos a mostrar:

Mostrar en una rejilla (grid) email, alias, provincia, CP y zona.

Datos solicitados:

Prioridad.

Actores:

- Administrador

**2.1.11 Nombre: Login.**

Funcionalidad: permite al usuario identificarse en el sistema.

Acciones principales:

1. El sistema muestra la pantalla “login”.
2. El usuario introduce su nombre de usuario y contraseña.
3. El sistema comprueba los datos y si son correctos lo registra con su sesión.

Acciones alternativas:

2a) el usuario selecciona “Cancelar”

2a-1) el caso de uso finaliza.

3a) los datos no son correctos

3a.1) el sistema muestra la pantalla “login” con un mensaje de error.

3a.2) ir al punto 2

Pantallas:

- Login.

Datos solicitados:

Nombre de usuario (email) y contraseña.

Actores:

- Usuario y Administrador

**2.1.12 Nombre: Registrarse.**

Funcionalidad: permite al usuario registrarse en el sistema y poder acceder a otros casos de uso.

Casos de uso relacionados:

- Extiende el caso de uso “Login” en el punto 2.

Acciones principales:

4. El sistema muestra la pantalla “Registro de usuario”.
5. El usuario introduce los datos que se requieren.
6. El sistema comprueba los datos y si son correctos lo registra con su sesión.

Acciones alternativas:

2a) el usuario selecciona “Cancelar”

2a-1) el caso de uso finaliza.

3a) los datos no son correctos

3a.1) el sistema muestra la pantalla “Registro de usuario” con un mensaje de error.

3a.2) ir al punto 2

Pantallas:

- Registro de usuario.

Datos solicitados:

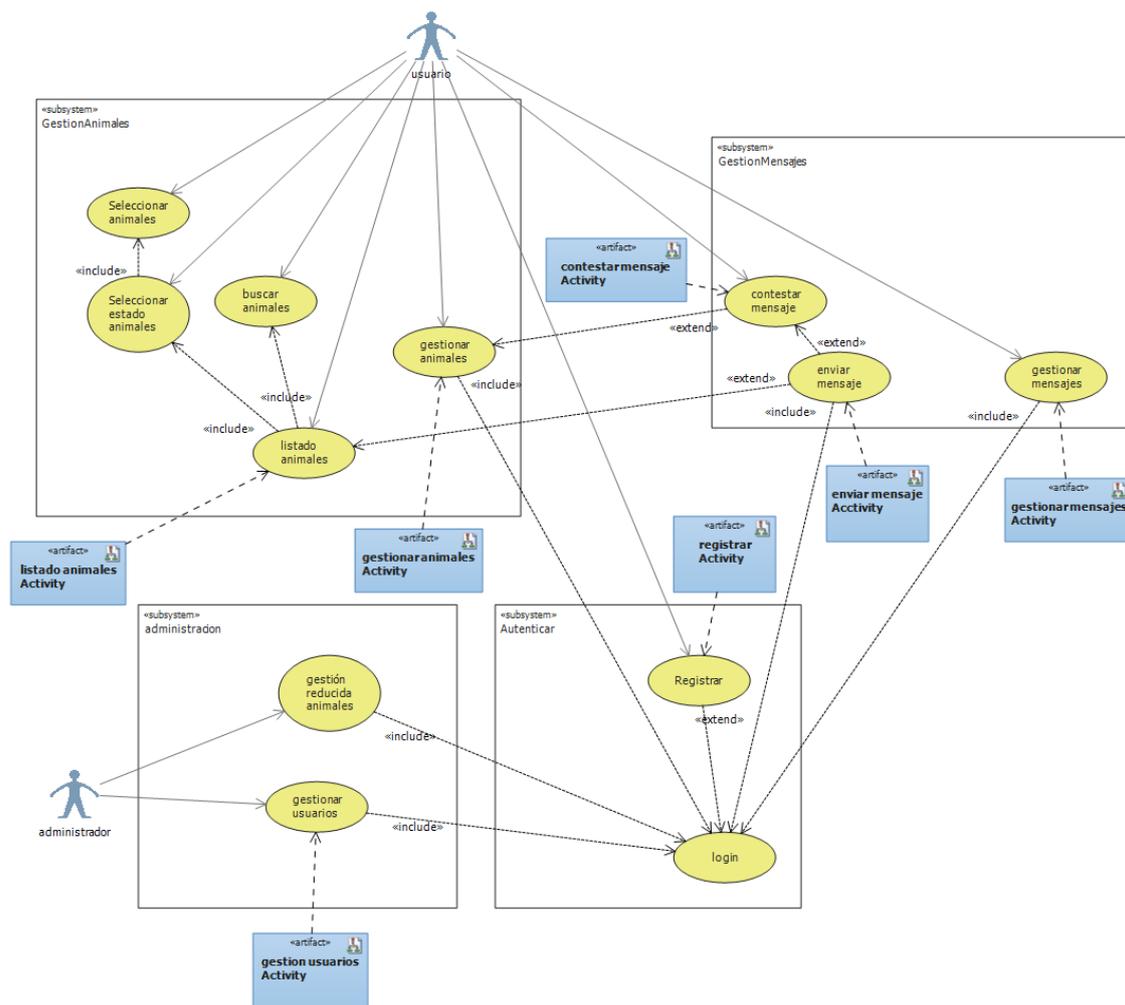
Nombre de usuario (email), contraseña, alias, provincia, CP y zona.

Actores:

- Usuario

### 2.2 Diagrama de casos de uso:

Los elementos etiquetados como "artifact" son los enlaces a los diagramas de actividad de cada caso de uso.



### 2.3 Diseño del interfaz de usuario.

Para diseñar el interfaz de usuario, nos basaremos en los casos de uso, a partir de los cuales generaremos los diagramas de estado o de actividad, que representan cada pantalla y las transacciones que se pueden dar durante la ejecución del caso de uso.

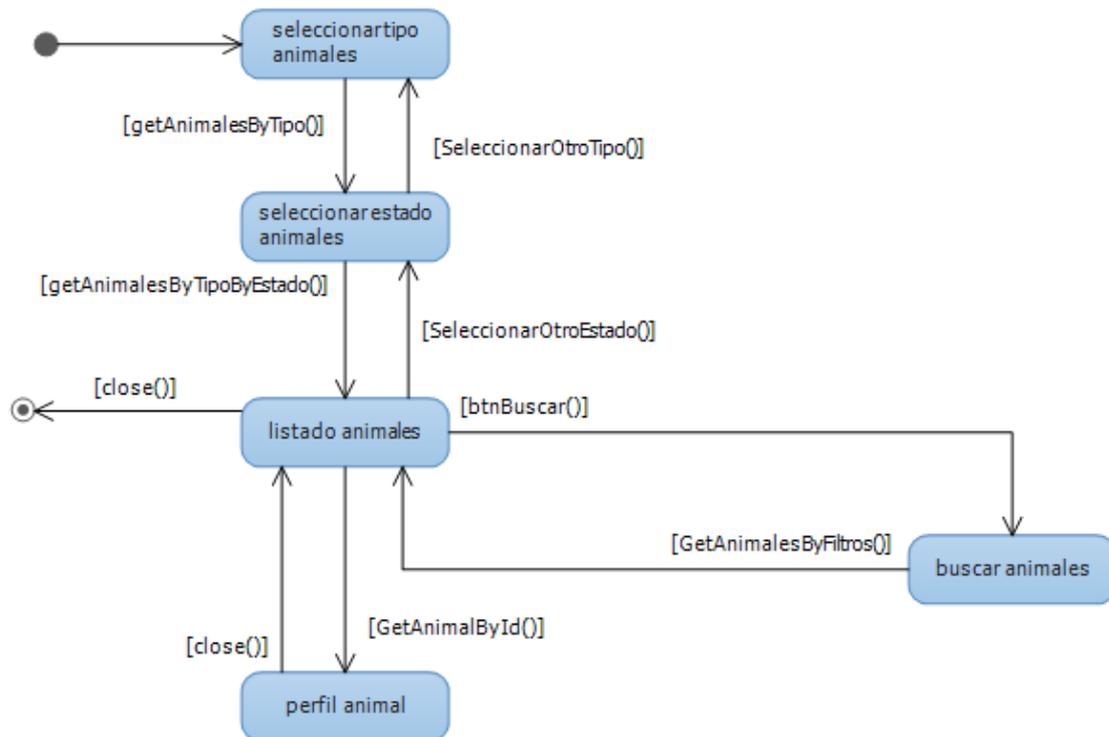
El diagrama de actividad es usado para mostrar los procesos que se realizarán al ejecutar una serie de acciones. Estas acciones las puede realizar cualquier tipo de objetos: personas,

software u ordenadores. La mejor manera de entender este diagrama es verlo con un diagrama de flujo.

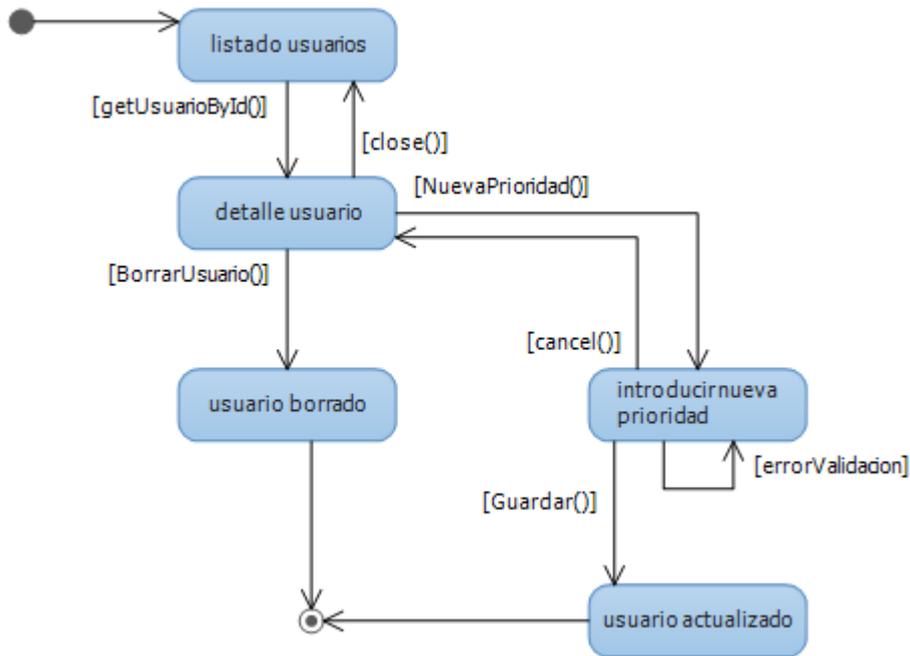
### 2.3.1 Diagramas de actividad.

Los diagramas de actividad desarrollados son:

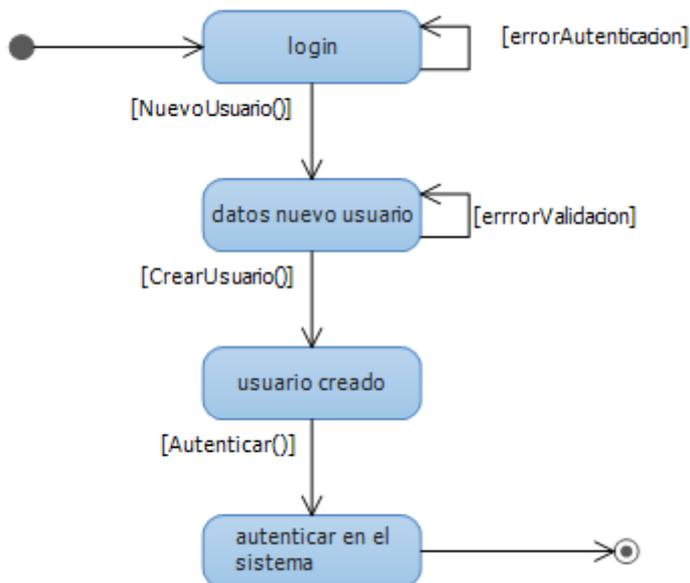
#### 2.3.1.1 Listado de animales.



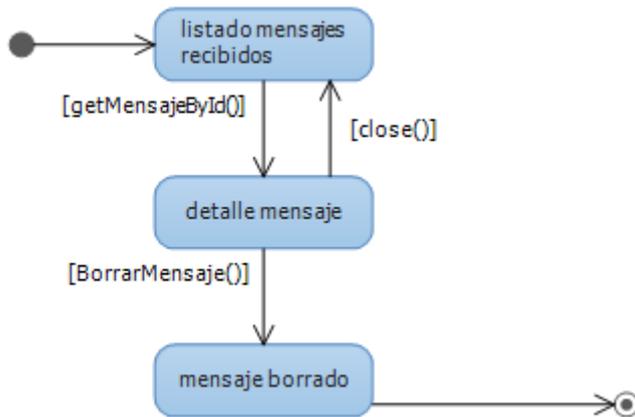
### 2.3.1.2 Gestión usuarios.



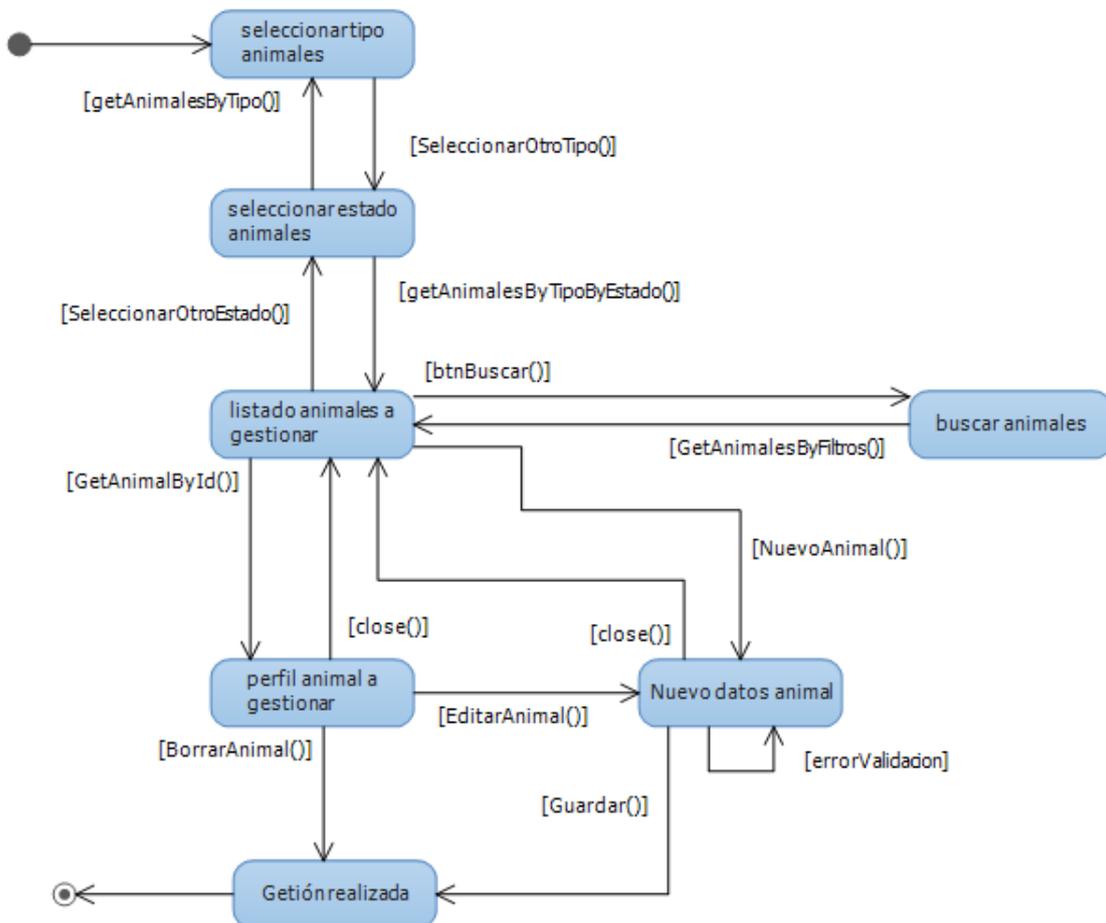
### 2.3.1.3 Registrar.



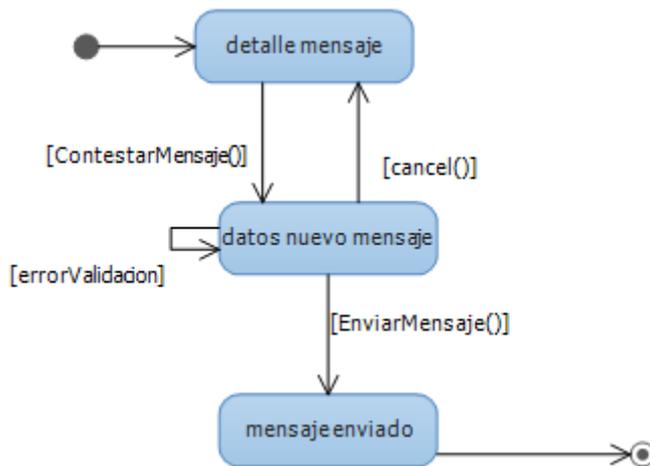
### 2.3.1.4 Gestionar mensajes.



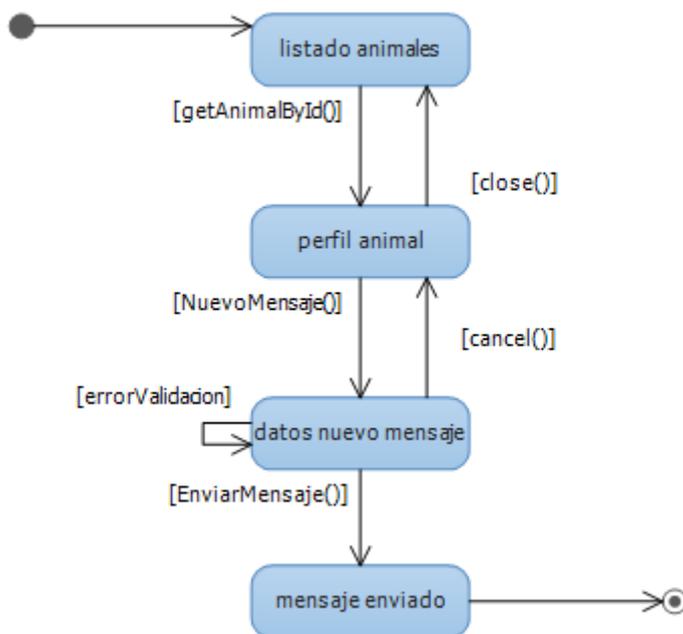
### 2.3.1.5 Gestionar animales.



### 2.3.1.6 Contestar mensaje.



### 2.3.1.7 Enviar mensaje.



## 2.4 Diseño capa presentación.

### 2.4.1 Diseño externo.

En el diseño externo del sistema nos limitaremos al diseño de las pantallas de la web: se diseña el aspecto gráfico del sistema, así como el de los controles de la interfaz gráfica del usuario (desplegables, botones, etc.) que se utilizarán para mostrar la información a los usuarios y recoger sus datos.

Para realizar este diseño, nos basaremos en los diagramas obtenidos hasta el momento: diagrama de casos de uso, diagramas de actividad y los esbozos de las pantallas realizadas a mano (que no se incluyen en este documento).

El uso de la tecnología web, para esta capa, nos impone una serie de limitaciones en el diseño de las pantallas que hay que tener en cuenta:

- Las pantallas las recibe el cliente en formato HTML y se mostrarán en su navegador. Por tanto los controles a utilizar estarán limitados a los que nos ofrece HTML. En caso de WebForms, podemos disponer de controles del servidor, sin embargo, en el modelo MVC, no existen este tipo de controles, por lo que las vistas deben ser creadas manualmente, obteniendo a cambio, mayor control sobre el HTML generado.
- El usuario interactúa con el sistema haciendo peticiones HTTP, cada una de las cuales puede tener parámetros que el usuario indicará rellenando los formularios HTML.

Las pantallas que se han definido son:

## **2.4.2 Pantallas del interfaz de usuario.**

### **2.4.2.1 Parte común a todas las pantallas.**

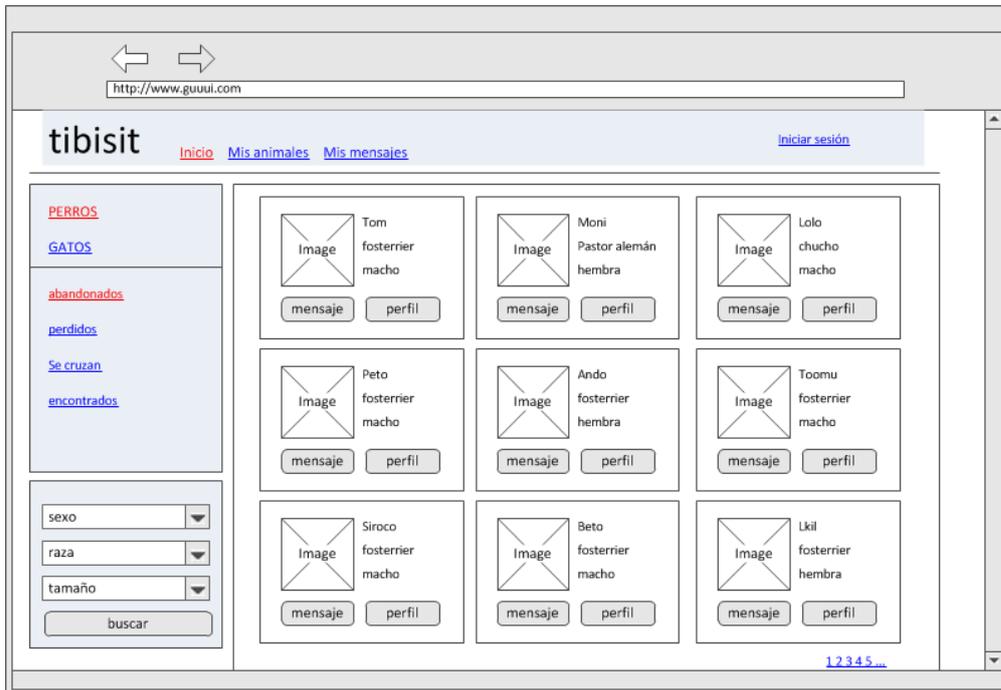
Se definirá una “master page”, es decir, una zona que será común a todas las pantallas, que darán acceso la pantalla inicial, a gestionar los animales del usuario, a gestionar los mensajes recibidos del usuario o a iniciar sesión en el sistema.

### **2.4.2.2 Listado de animales.**

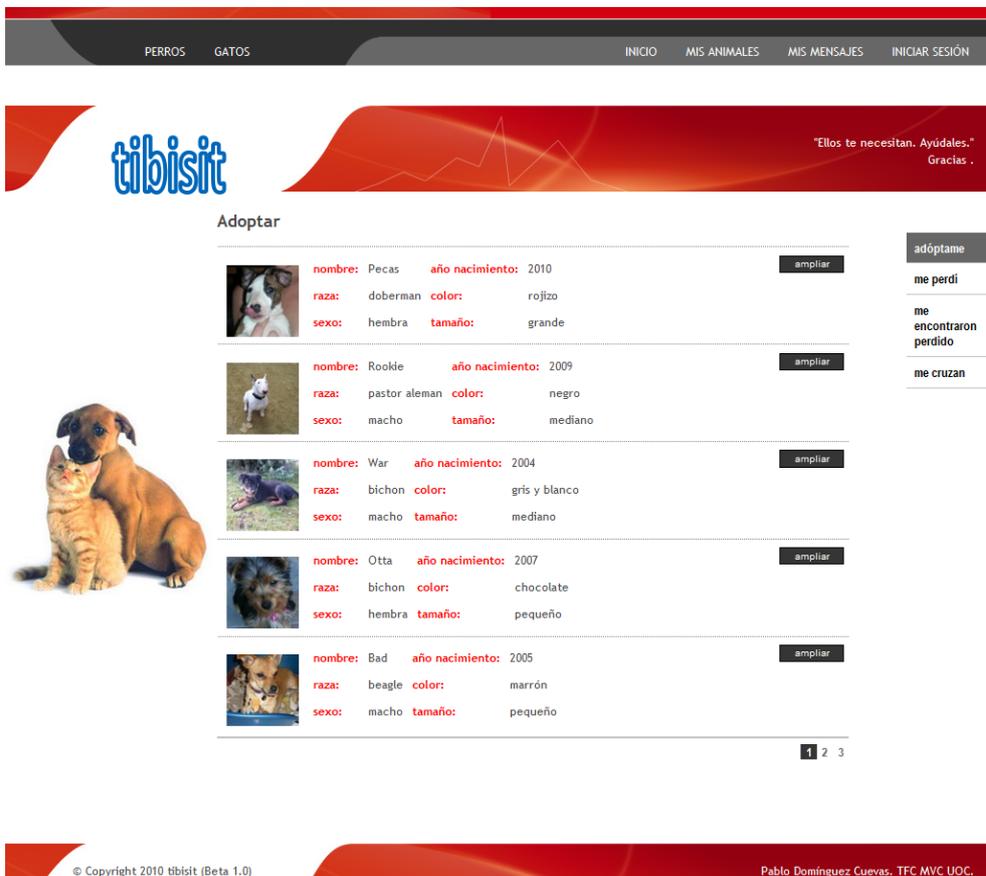
En esta pantalla elegiremos entre “perros” o “gatos”, posteriormente entre “abandonados”, “perdidos”, “se cruzan” y “encontrados”.

Se nos mostrará el listado de animales según nuestras preferencias, apareciendo una foto, unos datos básicos y los botones de ver el perfil completo o enviar un mensaje al cuidador del animal.

Además tendremos la opción de buscar un animal mediante una serie de filtros que podrá rellenar el usuario.

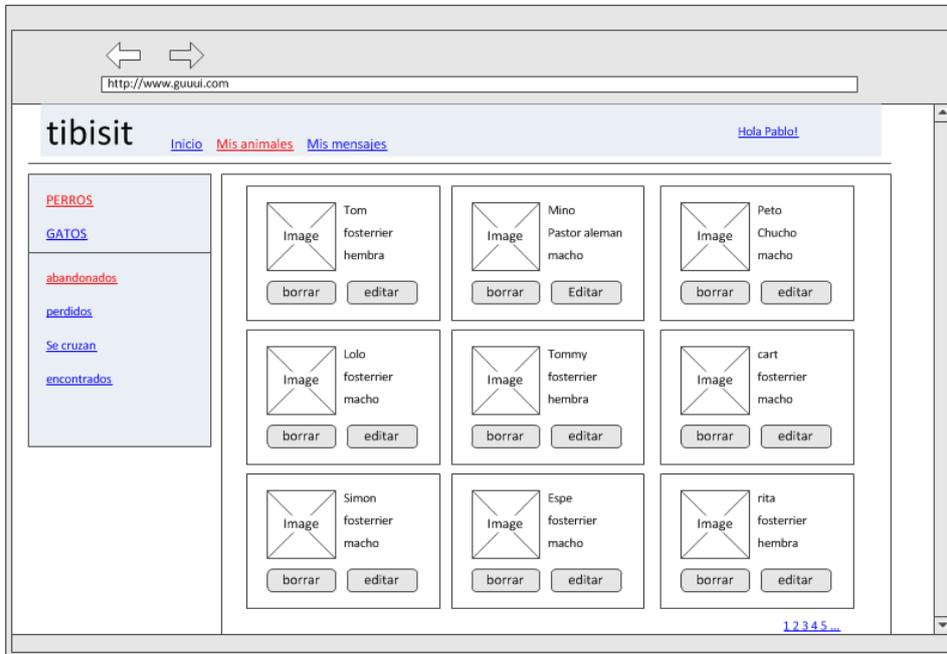


Pantalla final:

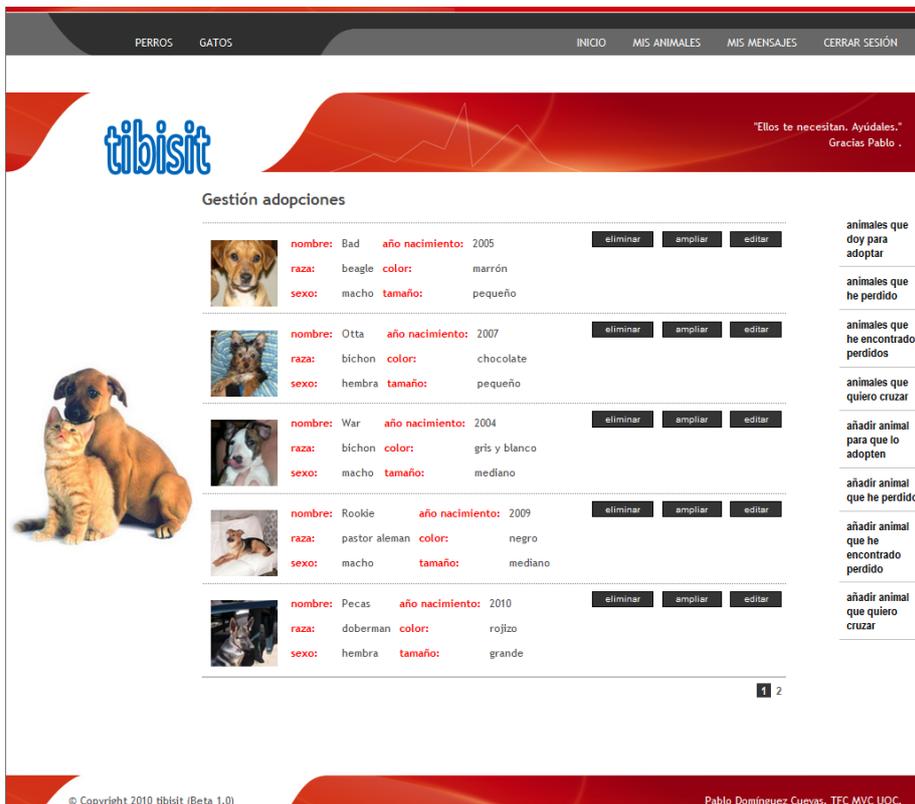


### 2.4.2.3 Pantalla de gestión de animales.

Desde esta pantalla el usuario podrá gestionar sus animales, se mostrarán en una rejilla la foto, los datos básicos del animal, y los botones que dan opción a borrarlo o editar la totalidad de sus datos. Como se puede ver arriba: *“Hola Pablo!”*, el usuario ya se ha autenticado.

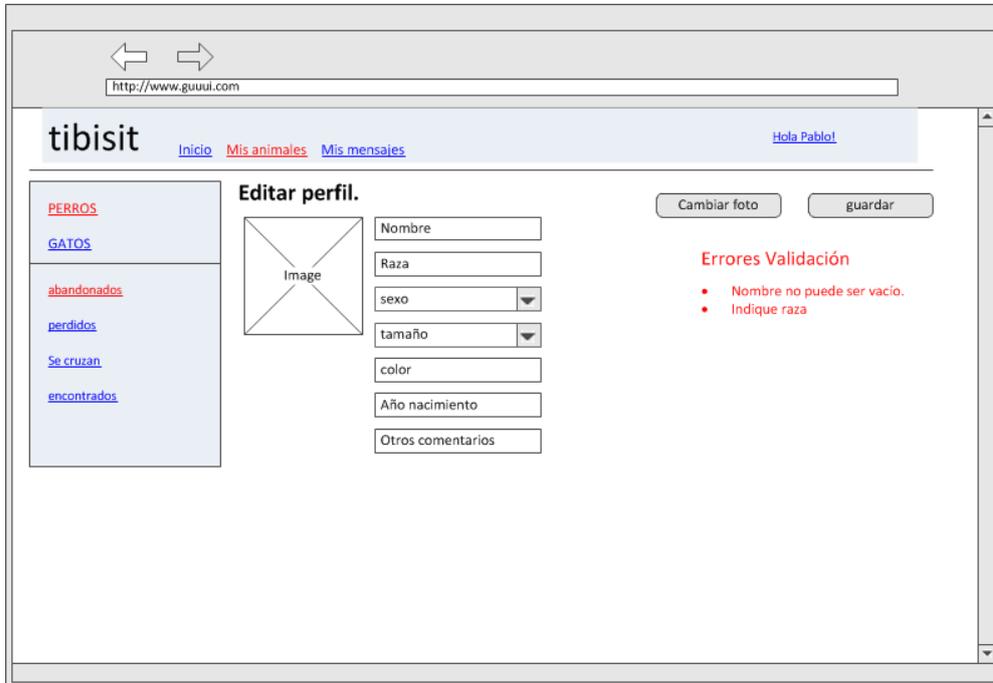


Pantalla final:



### 2.4.2.4 Pantalla de edición del perfil del animal.

Desde esta pantalla el usuario podrá editar todos los datos de su animal y cambiar la foto. También se mostrarán los errores de validación de los datos introducidos.



Pantalla final:



Editar adopción: Bad




PdCuevas,TFC U

Nueva imagen

nombre :  
Bad

año nacimiento :  
2005

sexo :  
macho

raza :  
beagle

tamaño :  
pequeño

peso (Kg) :  
2

pelo :  
corto

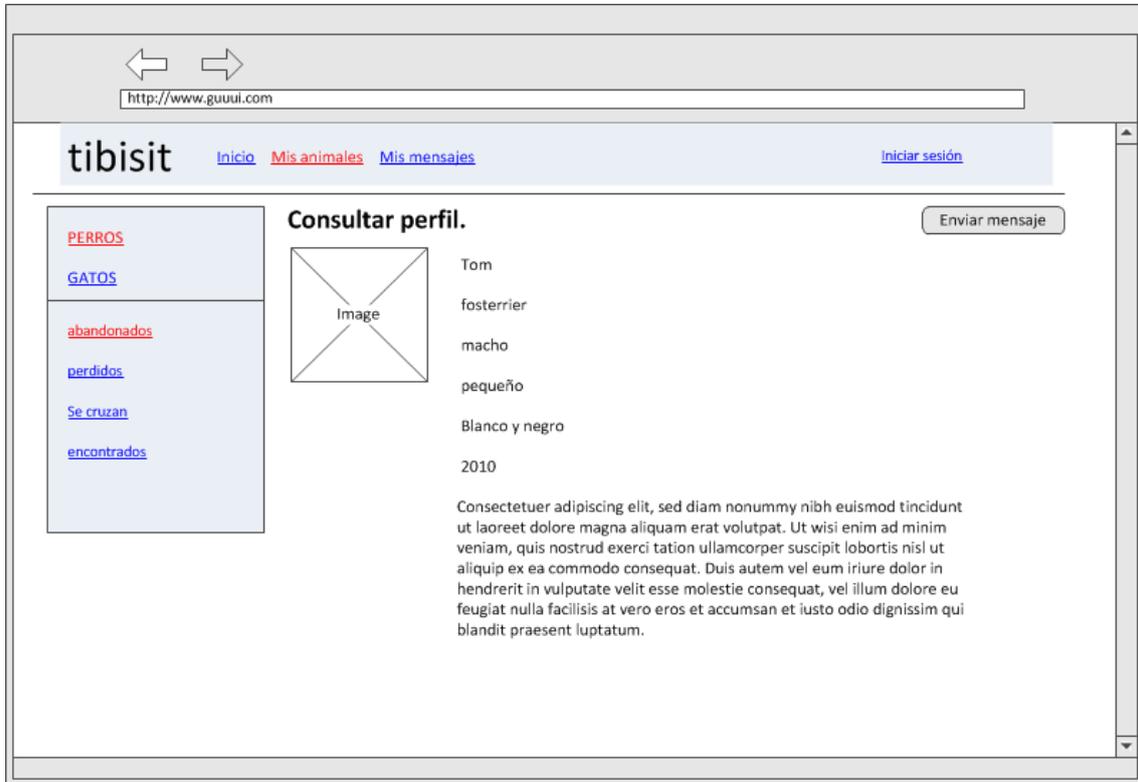
color :  
marrón

comentario :  
muy bonito

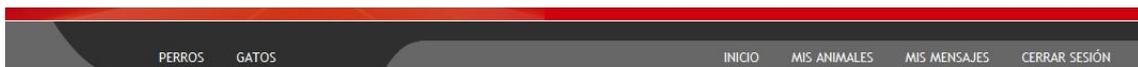
Guardar Volver

### 2.4.2.5 Pantalla para consultar el perfil del animal.

Desde esta pantalla el usuario podrá ver todos los datos de su animal y enviar un mensaje al cuidador del animal.



Pantalla final:



Ver animal para adoptar Bad



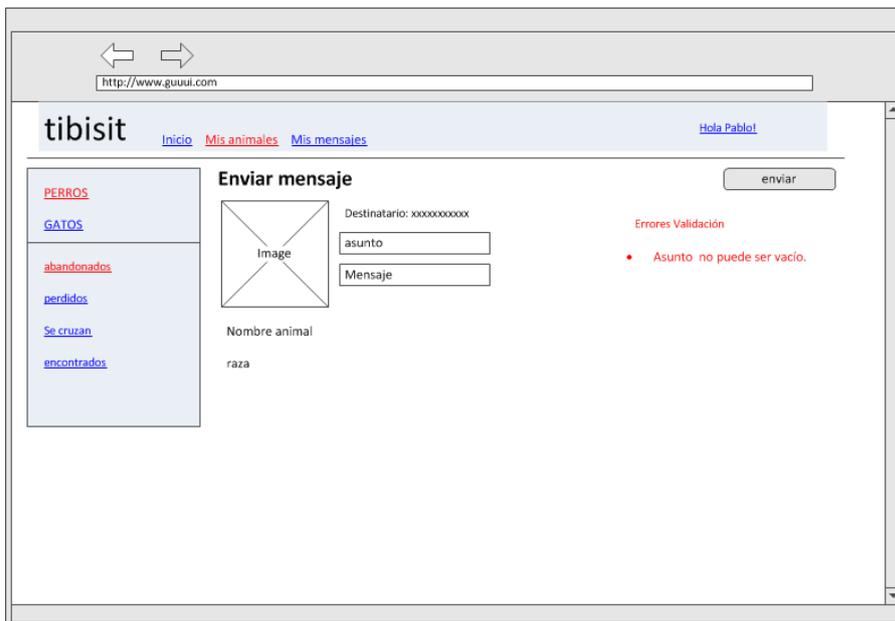
nombre:	Bad	año nacimiento:	2005
raza:	beagle	color:	marrón
sexo:	macho	tamaño:	pequeño
pelo:	corto	peso (Kg):	2
alta:	12/30/2010	tipo:	perro
alias:	Pablo		
provincia:	Castellón		
comentario:	muy bonito		

Volver

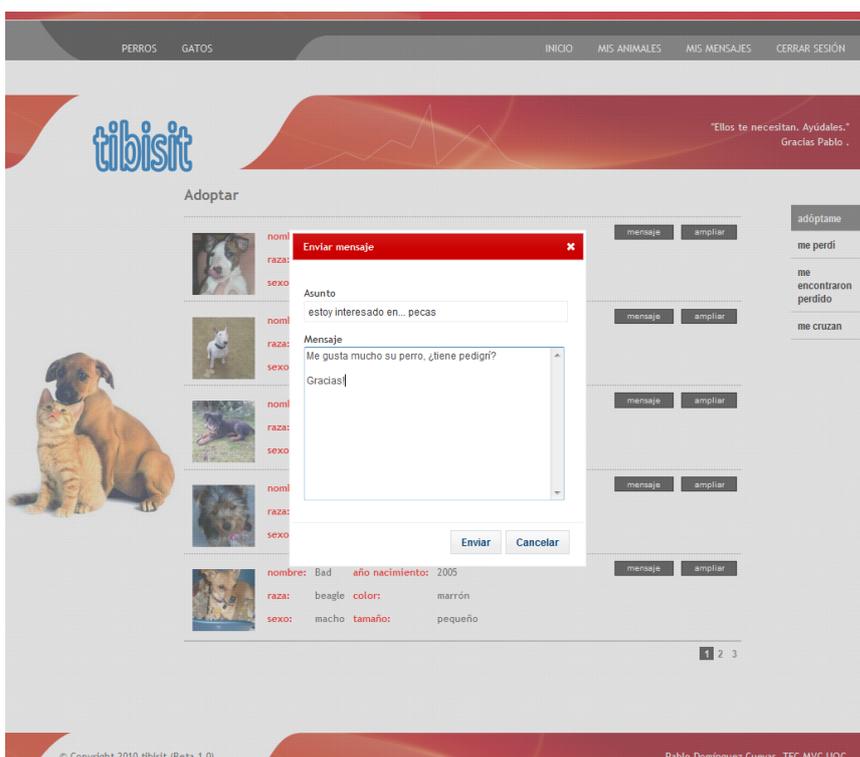
### 2.4.2.6 Pantalla para enviar mensaje a otro usuario.

Desde esta pantalla el usuario introducirá el “asunto” y “mensaje” del mensaje, que desea enviar a otro usuario, para indicarle que está interesado en un animal. Todos los mensajes van ligados a un animal.

También se mostrarán los errores de validación de los datos introducidos.



Pantalla final usando jQuery UI dialog:

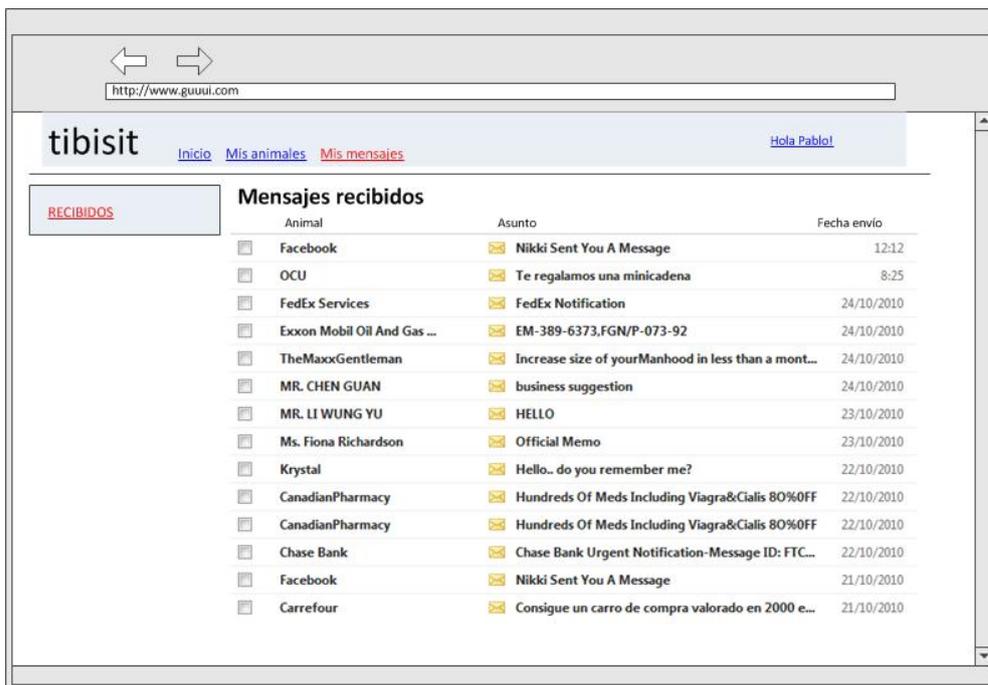


### 2.4.2.7 Pantalla de gestión de mensajes recibidos.

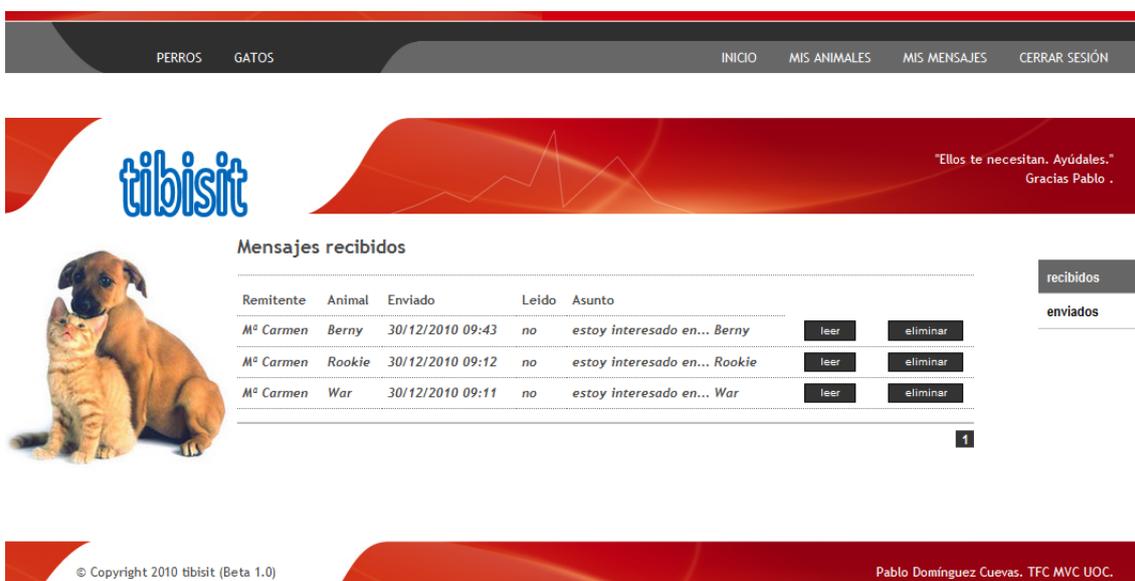
En esta primera versión de la web, se mostrarán los mensajes recibidos por los usuarios, en un futuro se ampliaría a los mensajes enviados, para ello se deja el diseño de la web preparado para una fácil actualización.

Se mostrará una rejilla indicando el animal al que va destinado el mensaje, el asunto y la fecha de envío del mismo.

Haciendo click sobre cada uno de ellos se podrá ver el mensaje completo.

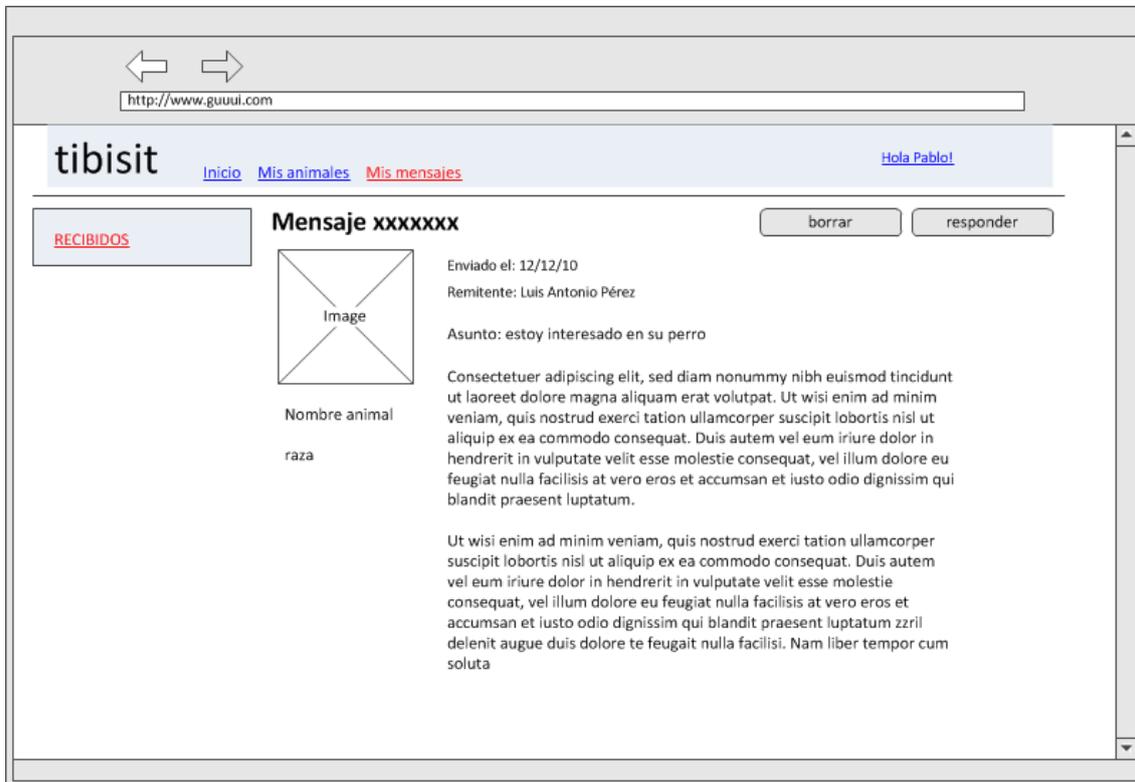


Pantalla final:

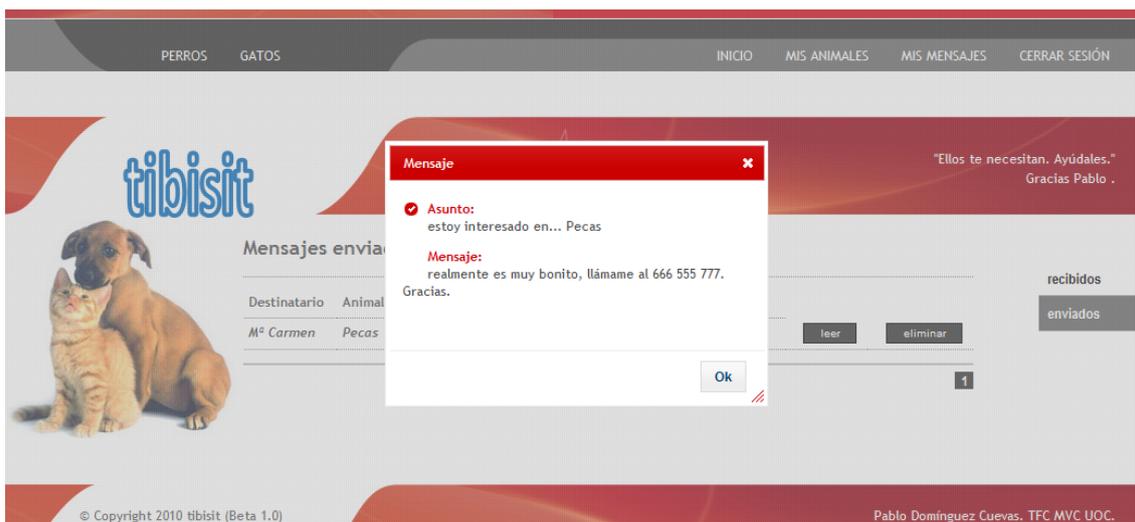


### 2.4.2.8 Pantalla de detalles del mensaje.

En esta pantalla el usuario verá el mensaje recibido con todos sus detalles, dándole la opción de borrarlo o responderlo al usuario que nos lo remitió.



Pantalla final usando jQuery UI dialog:

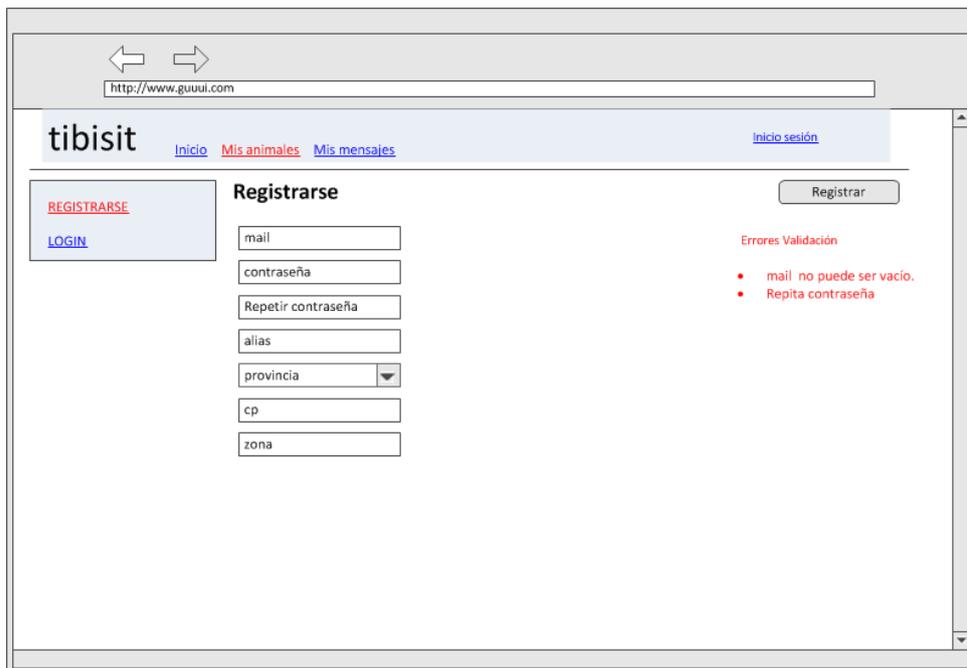


### 2.4.2.9 Pantalla de registro de un nuevo usuario.

Los usuarios para realizar distintas acciones sobre el sistema deberá autenticarse, para ello podrá, en esta pantalla darse de alta. Indicando su mail, su contraseña, el alias por el que

desea que se le identifique en el sistema, así como sus datos de localización, como la provincia, el código postal y la zona en la que reside.

También se mostrarán los errores de validación de la información introducida por el usuario. Una vez registrado, el usuario estará autenticado automáticamente en el sistema.



http://www.guuui.com

tibisit Inicio Mis animales Mis mensajes Inicio sesión

REGISTRARSE LOGIN

Registrar

mail

contraseña

Repetir contraseña

alias

provincia

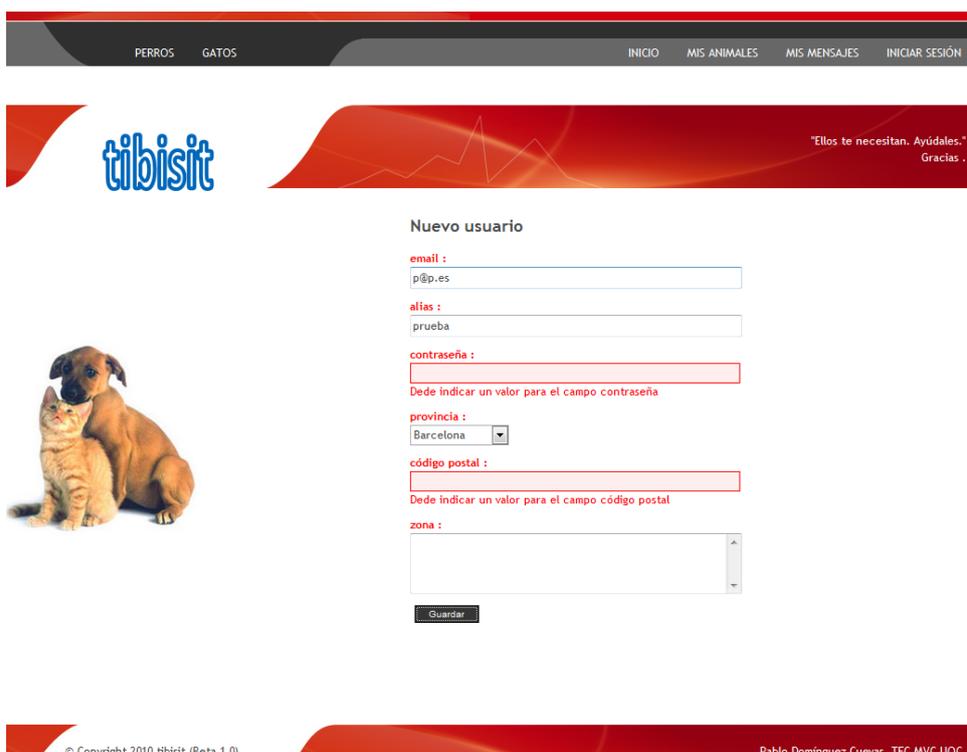
cp

zona

Errores Validación

- mail: no puede ser vacío.
- Repite contraseña

Pantalla final:



PERROS GATOS INICIO MIS ANIMALES MIS MENSAJES INICIAR SESIÓN

tibisit "Ellos te necesitan. Ayúdales." Gracias .

Nuevo usuario

email : p@p.es

alias : prueba

contraseña : Dede indicar un valor para el campo contraseña

provincia : Barcelona

código postal : Dede indicar un valor para el campo código postal

zona :

Guardar

© Copyright 2010 tibisit (Beta 1.0) Pablo Domínguez Cuevas. TFC MVC UOC.

## 2.5 Diseño de la capa de dominio.

### 2.5.1 Diagrama de componentes.

Este diagrama nos permite visualizar los componentes del sistema que implementan la funcionalidad del mismo, algo así como un puzle de piezas del sistema. Muestra las relaciones entre varios componentes de la aplicación o el sistema.

Su elaboración permite al equipo de desarrollo entender el diseño actual y ver potenciales fallos o mejoras. Además al mostrar las relaciones entre componentes, permite aplicar los cambios más fácilmente, si los requisitos cambian.

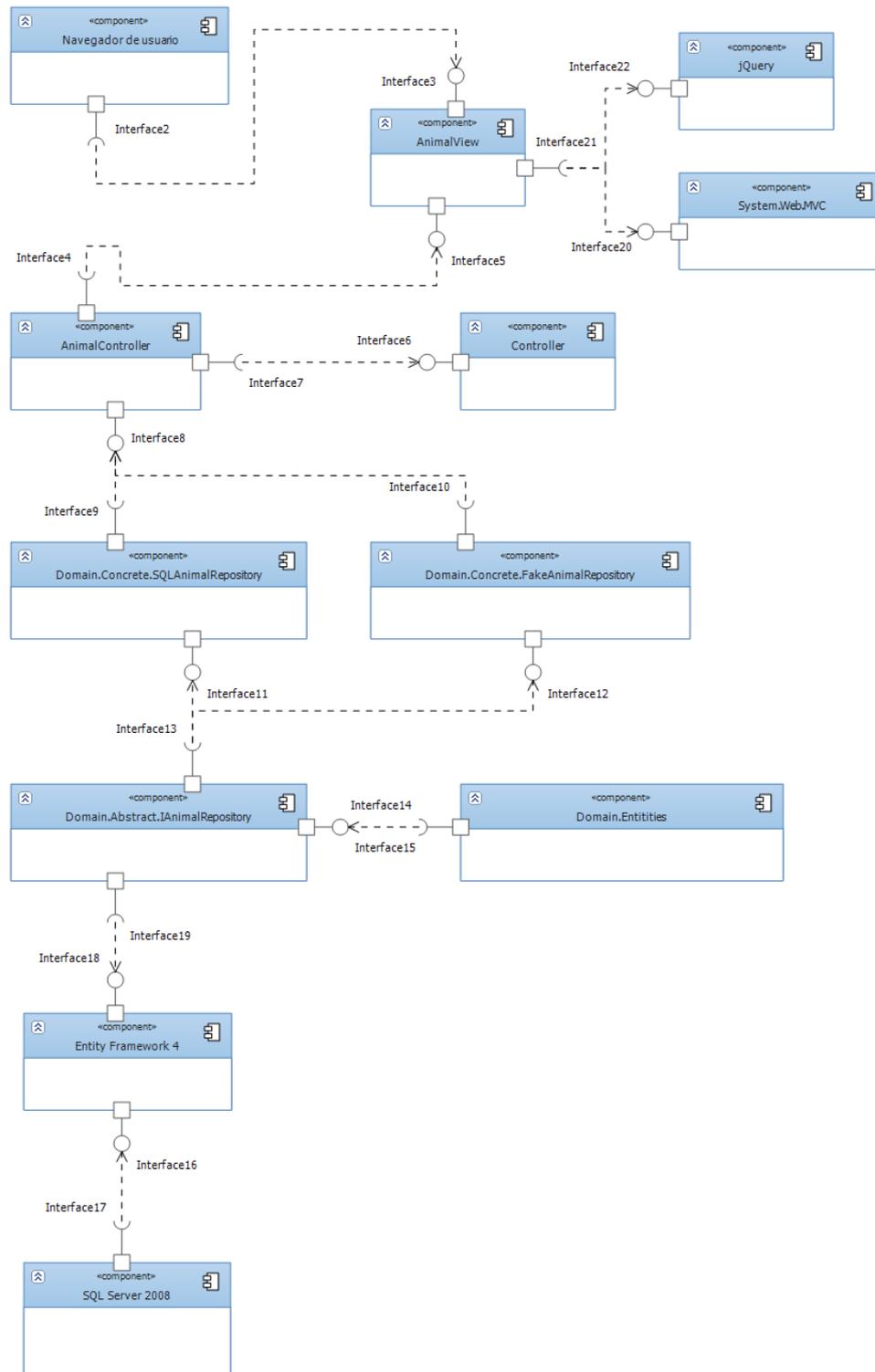
Dado que la aplicación se ha de desarrollar en dos arquitecturas diferentes, el diagrama de componentes desarrollado, será distinto para cada una de ellas.

El diagrama mostrado, en ambos casos, es el de la clase “animal”, sin embargo, para el resto de clases, sería el mismo, por tanto, no los publico.

#### 2.5.1.1 Diagrama de componentes WebForms.



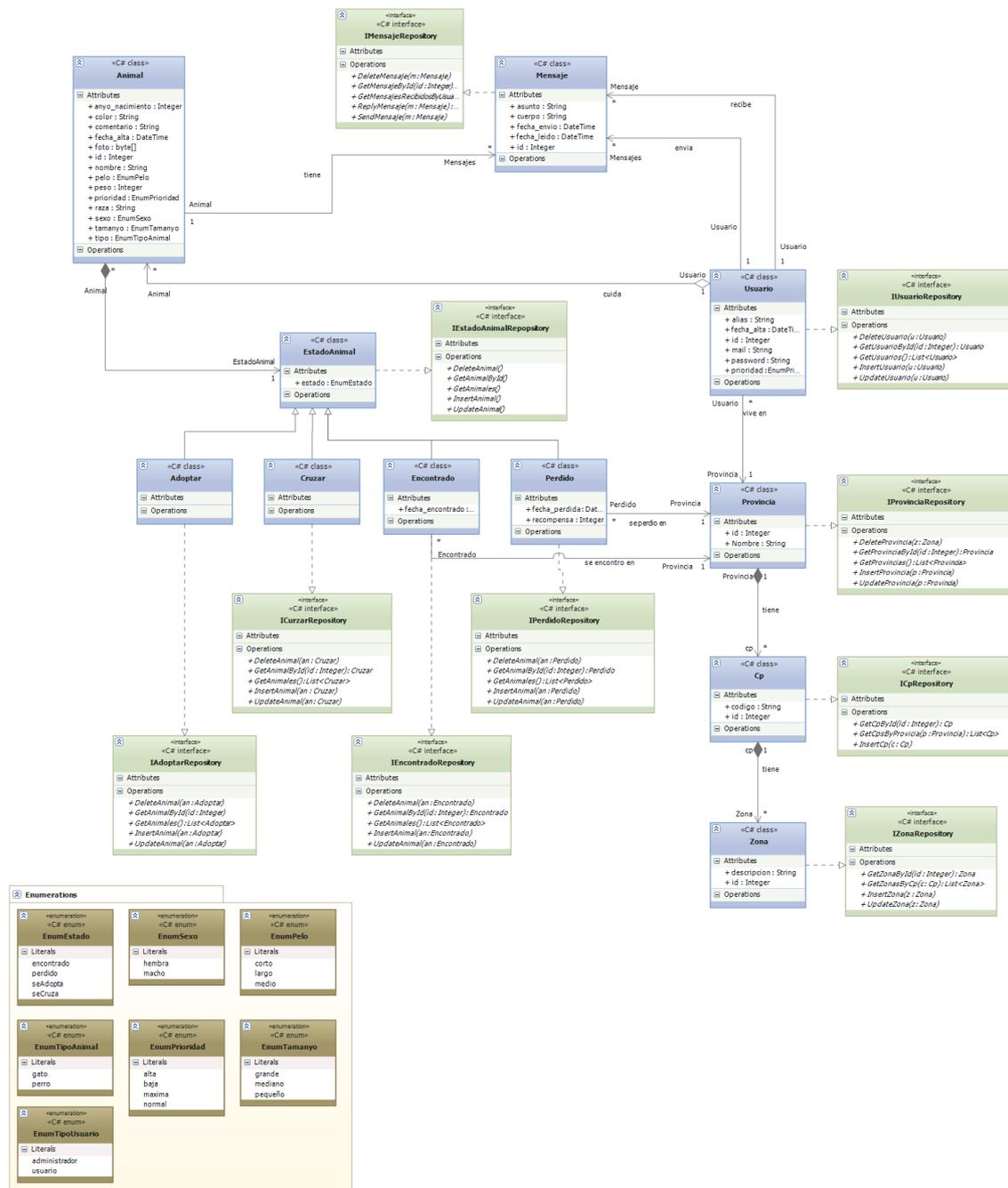
### 2.5.1.2 Diagrama de componentes para MVC.



Como ejemplo, he introducido un componente “FakeAnimalRepository”, cuya función es la de ayudar a realizar los test de la aplicación. Como se ve, con este desarrollo, es muy fácil tener distintas implementaciones para los repositorios.

### 2.5.2 Modelo conceptual: diagrama de clases UML.

El diagrama de clases muestra las clases de la aplicación o sistema, y las relaciones que existen entre ellas. Los diferentes símbolos representan las distintas relaciones que pueden tener entre ellas, como herencia o asociación. El propósito de este diagrama es centrarse en los aspectos lógicos de las clases, en lugar de como han de ser implementadas.



Lo más destacable de este modelo es la utilización del patrón de diseño: Estado. Para diferenciar los estados de un animal, se podrían haber creado directamente las clases derivadas “adoptar”, “cruzar”, “encontrado” y “perdido”, derivando de la clase “animal”. Sin

embargo aplico este patrón que nos facilita la extensión del diseño con nuevos estados y nos permite cambiar de estado durante la ejecución del sistema.

Otro patrón de diseño utilizado es el método factoría abstracta (c# interface). Los objetos de la base de datos, deben ser almacenados en algún tipo de almacenamiento persistente, sin embargo, no debemos mezclar el código de persistencia con el código de la capa de dominio, no se debe poner código de acceso a la base de datos directamente en los métodos de la capa de dominio. Por tanto, el modo de separarlo limpiamente es mediante los “repositorios”, que nos proporcionan una base para acceder a los datos de la base de datos, web services o cualquier otro soporte de datos, actuando como una fachada sobre la implementación real. Los repositorios solo los utilizaremos para leer o actualizar datos. En este proyecto, los métodos de cada repositorio contendrán el código LINQ para SQL, sin embargo se podría utilizar cualquier otra estrategia de acceso a datos, como usar directamente T-SQL con ADO.NET, NHibernate, etc.

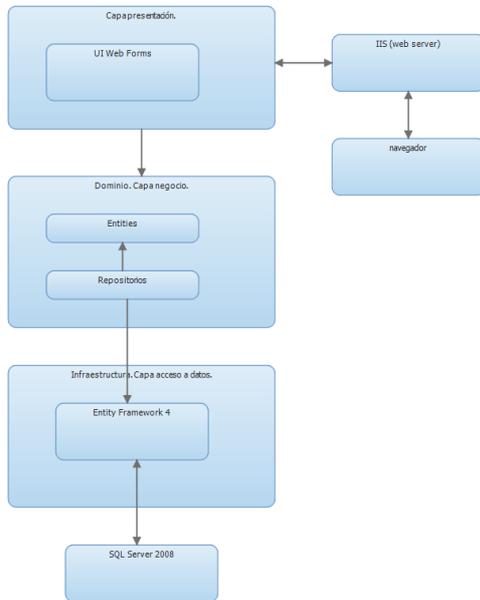
El modo de implementar estos repositorios en .NET es mediante una clase de tipo “interface”, esto facilitará sobre todo la realización de pruebas unitarias, pudiendo utilizar fácilmente “fakes” o “mocks” para éstas pruebas, y facilita además, el cambio, si queremos desarrollar sobre distintos tipos de bases de datos.

Las clases derivadas “cruzar” y “adoptar”, en un principio no tienen propiedades o métodos que las diferencien de “animal”, sin embargo, las creo, porque en un futuro pueden si tenerlos, y de este modo, el cambio será muy sencillo.

## 2.6 Arquitectura del proyecto.

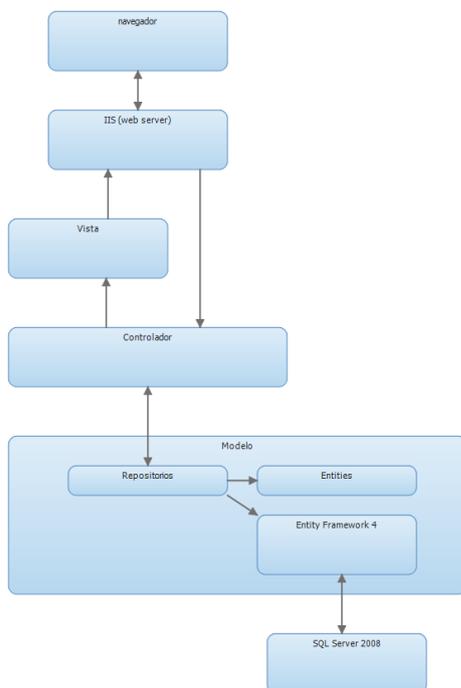
Como he mencionado, en este proyecto, además de desarrollar la aplicación MVC, se debe realizar otra con WebForms. Esto implica utilizar dos arquitecturas distintas. Para WebForms utilizaremos una arquitectura de 3 capas para aplicaciones web, mientras que para la otra, utilizaremos una arquitectura MVC (Modelo2) para la web.

### 2.6.1 Arquitectura WebForms.



Diferenciaremos entre capa de presentación, dominio (negocios) e infraestructura (acceso a datos).

### 2.6.2 Arquitectura MVC (Modelo2).



Lo más destacable de este gráfico, es que podemos ver, como a diferencia del modelo MVC original, en el que la “vista”, se comunicaba con el “controlador” y el “modelo”, en la revisión de este patrón, llamado MVC Modelo2, ahora la “vista” tiene un papel mucho más pasivo, ya que no se comunica con el “modelo”, y ahora es el “controlador”, y no la “vista”, el que se

comunica con el “modelo” y la “vista”. Ahora la “vista” recibe los datos a través del “viewModel”, pero no tiene acceso a la capa de negocio o “modelo”.

Este cambio, permite que la “vista” sea más simple en el Modelo2, que en el MVC original, por ello, es el utilizado en el framework de Microsoft para ASP.NET MVC.

### 3. Implementación.

#### 3.1 Desarrollar web MVC.

##### 3.1.1 Proyecto tibisit.Domain.

El primer paso fue crear la base de datos, para ello cree el modelo ADO.NET Entity Data definiendo en modo gráfico distintas entidades y sus relaciones de asociación y herencia. Una vez definido, generé la base de datos a partir del modelo (“Generate Database from Model...”). El siguiente paso fue generar las clases sin persistencia POCO (Plain old CLR object). Estas clases son las que compartiremos entre las distintas capas, ya que son objetos planos sin referencias al Entity Framework. Las clases se generan a partir del fichero T4 “poco.tt”.

El siguiente paso fue añadir y modificar el fichero T4 “IRepository.tt”, cuya finalidad es la de crear los repositorios para cada clase POCO. Por cada una de ellas se generan dos ficheros, uno que puede ser modificado posteriormente y en el que añadí los nuevos métodos que fui necesitando posteriormente. Y un fichero “xxx.Generated.cs”, que no se puede modificar y que contiene los métodos básicos con las operaciones CRUD de cada clase POCO.

##### 3.1.2 Proyecto tibisit.Db.

Este proyecto es de tipo base de datos SQL Server 2008, y que he utilizado en caso de necesitar modificar la base de datos y generar el script que gestiona dicho cambio.

##### 3.1.3 Proyecto tibisit.mvc.Web

Éste es el que define la capa de presentación del proyecto. En este caso es una aplicación web utilizando el patrón de diseño MVC. En concreto es una aplicación ASP.NET MVC 2.

##### 3.1.4 Proyecto tibisit.test

A la vez que creé este proyecto, también añadí el proyecto tibisit.test, cuya finalidad era la de programar mediante TDD, es decir, realizar primero el test de prueba, y posteriormente mediante refactoring implementar el método que permita pasar el test correctamente. Sin embargo, como he mencionado en el documento de instalación, por necesitar un tiempo aprendizaje y adaptación del que no disponía, no pude continuar con esta técnica. Para poder realizar los test, también implementé repositorio Mock, es decir, repositorios que trabajan con las clases POCO, pero no acceden a la base de datos, permitiendo crear datos de prueba que facilitan la realización de los test. Hay que tener presente que para mejorar el rendimiento de los test, es más rápido no acceder directamente a los registros de la base de datos, para mejorar el rendimiento de estos.

A la hora de implementar un mock opto por una librería (<http://code.google.com/p/moq/>) para .NET que permite de un modo más o menos simple implementar los casos de prueba. Un

ejemplo de esto se puede ver en el fichero "MockRepositories.cs" del proyecto tibisit.test. Debido al poco tiempo disponible tampoco puedo realizar todos los test que hubiera deseado, pero al menos, he aprendido a utilizarlo para futuros proyectos.

### 3.1.4 Problemas con las clases derivadas.

El primer problema grave con el que me encuentro es que las clases derivadas, es decir, las clases Adoptar, Cruzar, Perdido y Encontrado, que derivan de Animal, me fallan cuando intento crear un IObjectSet mediante el método CreateObjectSet(). Tras investigar el problema veo que este método solo acepta las clases base y no sus derivadas, por tanto, en lugar de hacer algo como: CreateObjectSet<T>(), tengo que hacer:

```
IObjectSet<T> Activator.CreateInstance(typeof(ObjectSetProxy<, >).MakeGenericType(typeof(T), BaseType), UnitOfWork.Context).
```

Este punto fue bastante delicado, ya que llegué incluso a pensar en eliminar las clases derivadas tras un par de días de infructuosa búsqueda, que finalmente encontré en <http://www.questionhub.com/StackOverflow/3597692>. Modifico el T4 "IRepository.tt" con esta corrección.

### 3.1.5 Ninject: control de dependencias.

Para el control de dependencias decidí utilizar Ninject (<http://ninject.org/>), una de las múltiples opciones open source disponibles para .NET y muy sencilla de utilizar. Básicamente es especialmente útil en las aplicaciones MVC para construir un "controller factory" que resuelve las dependencias automáticamente. Un "controller factory" en MVC, es un objeto al que llama el framework para instanciar el controlador, en MVC se llama, DefaultControllerFactory, y lo que se puede hacer es reemplazarla por una distinta que creamos nosotros, para ello solo debemos crear una clase que implemente IControllerFactory o que herede de DefaultControllerFactory. Esto lo podemos ver en el fichero "Infrastructure/NinjectControllerFactory.cs".

Para utilizar Ninject, solo hay que referenciar la dll, implementar la clase mencionada anteriormente e indicar su uso en el Global.asax en el método Application\_Start.

### 3.1.6 Primer controlador y vista. Numerar páginas.

#### 3.1.6.1 Acción List del controlador.

Creo el primer controlador, cuya funcionalidad es la de mostrar el listado de animales con estado "adoptar", es decir, el controlador AdoptarController y el acción List(). Lo primero fue ver que no hacía falta instanciar los repositorios, ya que lo hacía automáticamente la clase creada para Ninject.

El siguiente paso fue crear los nuevos métodos necesarios en el repositorio para acceder a la información requerida, por ejemplo: GetAnimales(Enumerations.EnumTipoAnimal tipo), que nos devuelve todos los animales en estado "adoptar" del tipo indicado como parámetro: "perro" o "gato". Otros métodos creados fueron: GetAnimalByUser() o GetAnimalById().

Posteriormente veo la necesidad de paginar los resultados obtenidos con el List(), por lo que lo modifiqué para que junto al parámetro del número de página a mostrar, este método devuelva

únicamente los registros necesarios para mostrar una página determinada. En este punto he podido observar como facilita LINQ algunas tareas que antes eran bastante más complejas. Por ejemplo la paginación utilizando T-SQL en SQL Server era mucho más compleja que mediante LINQ, que con Skip() y Take(), nos devuelven los registros que necesitamos.

### 3.1.6.2 PagingInfo.

Para poder paginar necesito una clase auxiliar en la que almacenar información, como el total de registros, registros por página y la página actual. Esta clase sería “/Models/PagingInfo.cs”.

### 3.1.6.3 HtmlHelpers: PageLinks.

Del mismo modo, en la página debía mostrar en html la información contenida en la clase PagingInfo, para lo que tuve que crear un html helper, que tras recibir como parámetro esta clase, me devuelve el html con los links a las distintas páginas que me permiten visualizar los registros solicitados. La clase “PageLinks” se encuentra en (“/Models/HtmlHelpers/PagingHelpers.cs”). El resultado sería algo como:

```
<a href="/?page=1">1</a>
<a class="selected" href="/?page=2">2</a>
<a href="/?page=3">3</a>
```

### 3.1.6.4 Modelo.

A la hora de intercambiar datos entre la vista y el controlador podría utilizar ViewData[], sin embargo, esto puede ser más difícil de mantener que utilizar una clase “view model” que nos permita intercambiar información entre la vista y el controlador. En este caso la clase AnimalesListViewModel que se encuentra en “/Models/AnimalesListViewModel.cs”.

### 3.1.6.5 Código final.

La acción List tras realizar estos pasos quedaría más o menos tal cual:

```
var adoptarToShow = _adoptarRepository.GetAnimales(tipo);
var viewModel = new AnimalesListViewModel<Adoptar>
{
    Animales = adoptarToShow.Skip((page - 1) * PageSize)
        .Take(PageSize)
        .ToList(),
    PagingInfo = new PagingInfo
    {
        CurrentPage = page,
        ItemsPerPage = PageSize,
        TotalItems = adoptarToShow.Count()
    }
};
return View(viewModel);
```

### 3.1.6.6 Vista.

Finalmente ya solo quedaba crear la vista, la cual se genera directamente seleccionando el método del controlador e indicando que cree la vista. Al crear la vista hay que indicar que sea

“tipada”, elegir la clase e indicar la página master. En este momento aun no tenía definida la master, por lo que posteriormente tuve que modificarlo.

La vista creada “/Adoptar/List.aspx” básicamente contiene un bucle para mostrar los animales y una etiqueta “div” que llama a la clase PageLinks, comentada en el punto anterior, para mostrar los links a las distintas páginas.

El bucle a su vez, hace una llamada a una vista parcial “/Shared/AdoptarSummary.ascx” que tiene el html necesario para mostrar la información de un solo animal. De este modo, lo que hacemos es ir construyendo el html necesario en forma de tabla, que va a mostrar los animales.

### **3.1.7 Página master.**

Un punto importante a la hora de diseñar una web, es contar con una página master y un estilo que sea común a toda la web y que sea lo más atractivo posible.

#### **3.1.7.1 MS Expression Web 4.**

Para conseguirlo, opto por instalar MS Expression Web 4. Este IDE está dirigido al diseñador de la web, sin embargo, tras varias horas de trabajo con él, veo que no es una herramienta preparada para trabajar con aplicaciones MVC. Por tanto, desecho la posibilidad de utilizarla en toda su funcionalidad, y básicamente la utilizo para modificar la página master a mis necesidades.

#### **3.1.7.2 Buscar página master.**

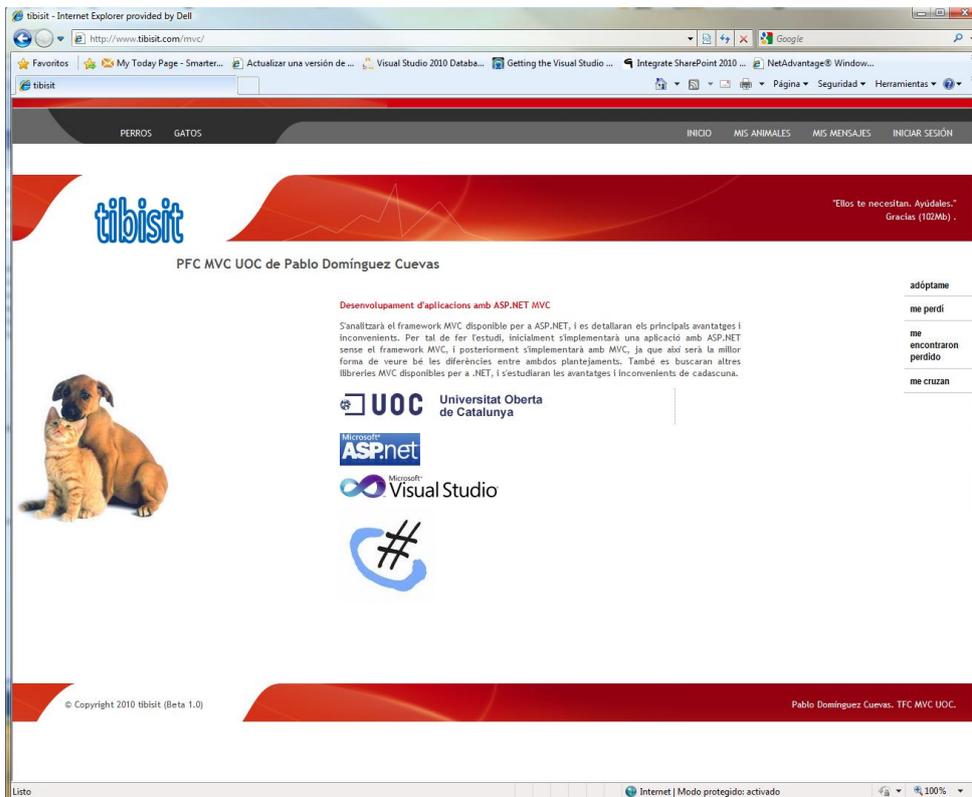
Tras varias búsquedas por internet, doy con la web de recursos del propio Expression Web. Allí opto finalmente por descargar y utilizar el template Red Style (<http://gallery.expression.microsoft.com/en-us/RedStyle>), diseñado por Helen (<http://helen.org.ua/blog/post/Red-Style.aspx>).

#### **3.1.7.3 Modificar página master.**

Tras instalar el template, utilizo Expression Web para modificarlo y ajustarlo a mis necesidades. Principalmente añadirle una columna para el menú de la derecha, y por supuesto, añadir toda la funcionalidad en MVC para que se muestren los menús o la información de usuario autenticado. Es importante observar que el template va acompañado del respectivo fichero css. A este fichero, le fui añadiendo posteriormente los estilos que fui necesitando.

### **3.1.8 Crear página de inicio.**

El siguiente paso fue crear la página de inicio basada en la master. Para ello definí el controlador HomeController y la vista correspondiente para la acción Index.



### 3.1.8.1 Menú derecha. EstadoController.

Para crear el menú que aparece a la derecha creo el controlador "EstadoController". Éste crea los distintos links que llamarán a la acción "List" de los diferentes controladores de cada estado del animal: "AdoptarController", "PerdidoController", "EncontradoController" y "CruzarController".

Como parámetro a la acción "Menu", le pasamos el estado en el que ha pinchado el usuario, de este modo, podremos mostrar resaltado el botón seleccionado por el usuario.

Para construir estos links, se definió la clase "PageLink" en "/Models/PageLink.cs". La clase guarda el texto que aparecerá en el menú, la ruta al controlador y la acción, y si el link ha sido el seleccionado.

Añado al fichero "/Content/styles.css" la definición para la clase "menú" para que se muestre resaltada la opción seleccionada.

### 3.1.9 Crear resto de controladores para visualizar animales.

Una vez definido el controlador "AdoptarController", la página master y la página de inicio, creo el resto de controladores, para cada uno de los estados del animal: "PerdidoController", "EncontradoController" y "CruzarController". Actualmente son prácticamente idénticos, pero en un futuro, cada estado de animal, podrá tener diferentes campos a mostrar, por lo que la aplicación ya está preparada para ello.

### 3.1.10 Añadir controlador para gestión de animales.

#### 3.1.10.1 Crear controlador.

El primer controlador que creo es “AdoptarGestionController”, su funcionalidad es similar a la de su anónimo “AdoptarController”, por lo que todas las clases auxiliares y modelos creados, me sirvieron para este nuevo controlador al implementar la acción “List”.

#### 3.1.10.2 Menú derecha.

El menú de la derecha se construye tal y como se hacía en la acción “Menu” del controlador “EstadoController”, pero en este caso, añadí las distintas opciones para crear nuevos animales, uno para cada estado. La acción que construye este nuevo menú es “MenuGestionar”.

#### 3.1.10.3 Acción Edit y Create.

Esta acción me permite dar de alta y editar los datos de un animal. Para dar un alta creo la acción “Create”, que lo que hace es llamar a la acción “Edit” con la información necesaria para crear el animal.

Definí dos acciones “Edit”. La primera “HttpGet” se utiliza para enviar información a la vista. Posteriormente cuando hacemos el “submit” desde la vista, ésta llama a la acción “Edit” “HttpPost”, que es donde se edita o da de alta el animal, con los datos rellenos en la vista, actualizando los datos en la base de datos.

#### 3.1.10.4 AdoptarEditarListViewModel.

Como en otros casos, tengo que crear este modelo que me permite intercambiar información entre el controlador y la vista. Este modelo está definido en “/Models/AnimalesListViewModel.cs”.

#### 3.1.10.5 Vista Edit.

Desde la propia acción “Edit” del controlador, creo la vista. En este punto lo más complicado fue crear los “dropdownlist” para los distintos tipos “enum”. Para ello tuve que crear distintos métodos que me convertían los tipo “enum” en “SelectListItems”. Estos métodos se definieron en la clase “/Models/EnumerationsToList.cs”.

#### 3.1.10.6 Errores inesperados.

Al crear el primer animal, me encuentro con un error inesperado. Al parecer hay un problema con el Entity Framework y la conversión de tipos entre “datetime” y “datetime2”, éste último específico de SQL Server 2008. Tras investigar el problema encuentro una referencia a este problema: <https://connect.microsoft.com/SQLServer/feedback/details/519863/datetime-parameters-getting-datetime2-values-from-net>.

Tras ver que es un problema que se produce “a veces”, opto por cambiar el tipo del campo de la base de datos, de modo, que el problema queda solucionado.

Otro problema con el que me encontré, fue que al instanciar los repositorios desde el “controller factory” de Ninject, utilizaba un nuevo “EUnitOfWork” para cada uno de ellos, por lo que al llamar al método “Save”, y tras modificar más de un repositorio, recibía un error

indicando que hacía referencia a varias instancias IEntityChageTracker. Para solucionarlo instancie un solo "EUnitOfWork", que utilicé para instanciar todos los repositorios.

### 3.1.10.7 Validación.

Cuando se permite al usuario introducir información en la base de datos, es preciso SIEMPRE, validar sus datos. Tras estudiar los distintos tipos de validación disponibles en MVC, opte por utilizar etiquetas "metadata" en las clases POCO.

#### 3.1.10.7.1 Modificar "poco.tt".

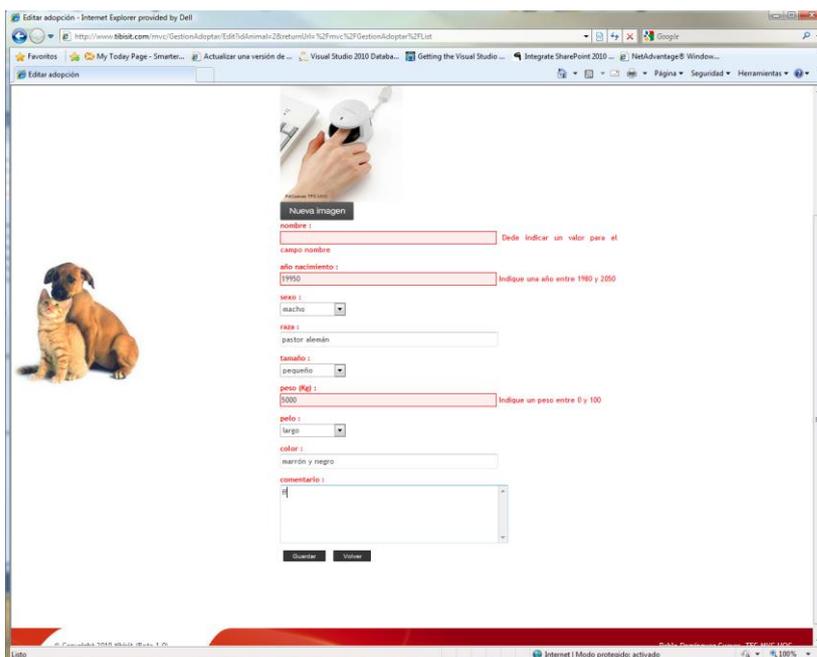
La primera tarea a realizar fue la de modificar el T4 "poco.tt", para que se crearan las nuevas clases "metadata", para cada una de las entidades del modelo. En esta clase es donde definí las nuevas etiquetas, de este modo, se separan las clases POCO, de las propiamente utilizadas únicamente para validación. Un ejemplo de estas etiquetas es:

```
[Required(ErrorMessage="Dede indicar un valor para el campo año_nacimiento")]
[Range(1980,2050,ErrorMessage="Indique una año entre 1980 y 2050")]
[DisplayName("año nacimiento")]
```

Hay que tener en cuenta que fue necesario añadir la referencia System.ComponentModel.DataAnnotations en el proyecto tibisit.Domain, para poder utilizar algunas de estas etiquetas.

#### 3.1.10.7.2 Ajax.

Para poder utilizar la validación en el lado del cliente, fue necesario añadir después del "body" las referencias a los scrips de ajax: MicrosoftAjax y MicrosoftMvcValidation. Actualmente no están disponibles los links CDN a estas librerías para MVC 2. Por lo que los referencia a la carpeta /Scripts.



### 3.1.10.8 Subir imágenes al servidor.

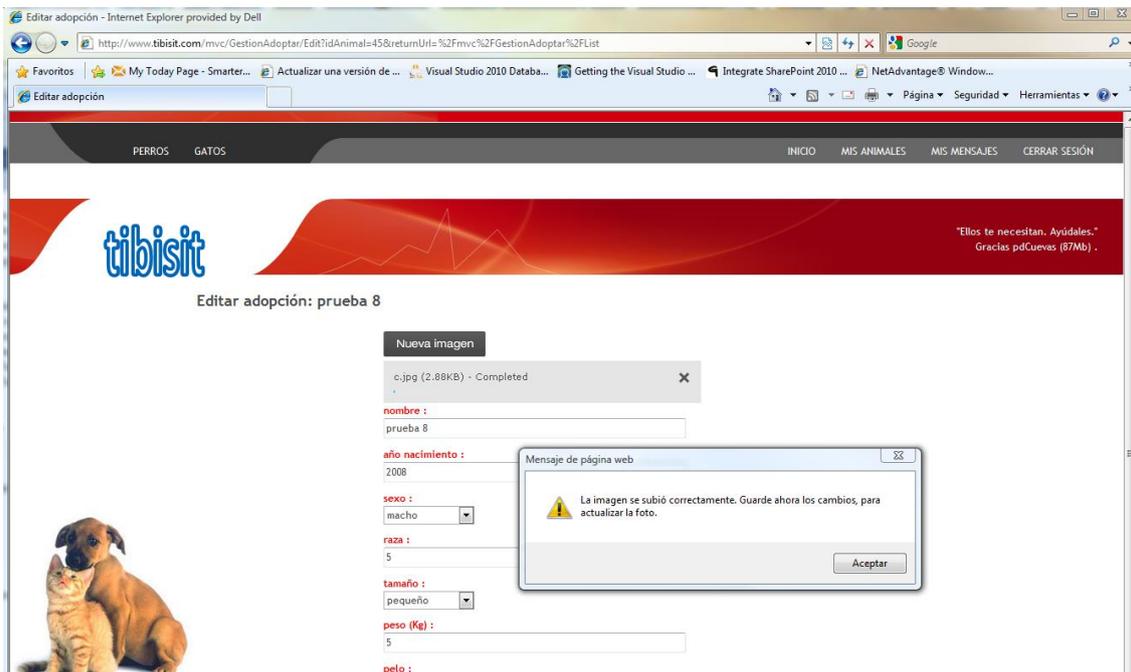
Para subir las imágenes al servidor opté por una librería JQuery gratuita: uploadify (<http://www.uploadify.com/>). Decidí utilizarla por permitir controlar en el lado del cliente, las extensiones de los ficheros a subir, así como el tamaño máximo del fichero (que limité a 100Kb). Su uso para proyectos MVC no está muy documentado, por lo que me costó un poco adaptarlo.

Como soporte a esta librería creo la clase ImageHelper (“/Models/ImageHelper.cs”), que me permite crear un nombre de fichero único para cada animal.

EL funcionamiento básico consiste en que, cuando el usuario sube una imagen al servidor, ésta se guarda en el directorio “/uploads”, con un nombre de fichero que depende del animal al que se asigna la imagen. Posteriormente la acción “Edit” se encarga de guardar esta imagen en la base de datos.

Es necesario modificar todas las vistas “Edit” añadiendo las referencias a los scripts de “uploadify”, así como para poner el código de subida de la imagen.

```
<link href="<%: Url.Content("~/Content/uploadify.css")%" type="text/css" rel="stylesheet" />
<script type="text/javascript" src="<%: Url.Content("~/Scripts/swfobject.js")%"></script>
<script type="text/javascript" src="<%:
Url.Content("~/Scripts/jquery.uploadify.v2.1.4.min.js")%"></script>
<script type="text/javascript">
    $(document).ready(function () {
        $('#file_upload').uploadify({
            'uploader': '<%= Url.Content("~/uploadify/uploadify.swf")%>',
            'script': '<%= Url.Action("UploadImagen", "GestionAdoptar", new {idAnimal=Model.Animal.Id})
%>',
            'cancelImg': '<%= Url.Content("~/images/cancel.png")%>',
            'folder': '<%= Url.Content("~/uploads")%>',
            'auto': true,
            'multi': 'false',
            'buttonText': 'Nueva imagen',
            'fileExt' : '*.jpg;*.gif;*.png',
            'fileDesc': 'Imágenes',
            'sizeLimit': 102400,
            'onAllComplete' : function(event,data) {
                alert('La imagen se subió correctamente. Guarde ahora los cambios, para actualizar la
foto.');
```



### 3.1.11 Mostrar imágenes de la base de datos.

Creo el controlador “ImagenController”, en el cual definí las acciones necesarias para obtener las imágenes desde la capa de datos.

Una opción muy interesante que implementé aquí fue la de poder añadir una “watermark” a las imágenes en el momento que se muestran. Es decir, en la base de datos, se guardan sin esta marca, pero antes de mostrarlas se les añade. Esto da una imagen profesional y evita que las imágenes puedan ser utilizadas por otros medios. La clase que genera esta marca es “ImagenWatermarkResult” que hereda de “ActionResult”.

Del mismo modo, añadí la posibilidad de leer las imágenes sin esta marca, sobre todos para cuando se presentan en modo miniatura, para reducir el tiempo de proceso.

Una vez creado el controlador, actualizo las vistas correspondientes donde quiero que aparezcan las imágenes, añadiendo algo como:

```
<div class="imgListado">
    
    %>
</div>
```



### 3.1.12 Definir el resto de controladores de gestión.

Una vez terminado el primer controlador de gestión "AdoptarGestionController", creo los otros tres, uno para cada estado, de modo que en la actualidad, como ya he mencionado, son básicamente iguales, pero en un futuro permitirán que se editen distintos datos, dependiendo del estado del animal.

### 3.1.13 Opción de ampliar la información del animal.

#### 3.1.13.1 Acción Show.

Esta acción la añadí a todos los controladores propios del animal, tanto de visualización como de gestión. La acción lee los datos de la capa de datos de un animal y los mete en un modelo, el cual, se envía a la vista.

#### 3.1.13.2 Vista Show.

Cree la vista desde la propia acción. Ésta muestra todos los datos del animal que pueden ser visibles para un usuario, incluida la foto.

#### 3.1.13.3 Añadir botón que llama a la acción Show.

Modifico las respectivas vistas parciales ("ascx") añadiendo un botón que va a llamar a la acción "Show" del correspondiente controlador:

```
<%:Html.ActionLink ("ampliar", "Show", new {idAnimal=Model.Id, returnUrl=Request.Url.PathAndQuery }) %>
```

También le pasé como parámetro la URL actual, de modo, que desde la vista "Show", el usuario pueda volver a la página de origen.

### 3.1.14 Borrar animales.

#### 3.1.14.1 Añadir botón en vista.

En los controladores de gestión añado la opción de “eliminar” animal. Para ello añadí en las vistas parciales (“ascx”) el botón que llama a la acción “Delete” del respectivo controlador.

```
<%using (Html.BeginForm("Delete", "GestionCruzar"))
{
    %>
    <%: Html.Hidden("idAnimal", Model.Id) %>
    <input type="submit" value="eliminar" class="button" id="<%: Model.Id %>" />
    <div id="dialog-confirm-<%: Model.Id %>" title="Eliminar animal" style="display:none" >Vas a
eliminar el animal '<%: Model.nombre %>', ¿estás seguro?</div>
```

```
<%} %>
```

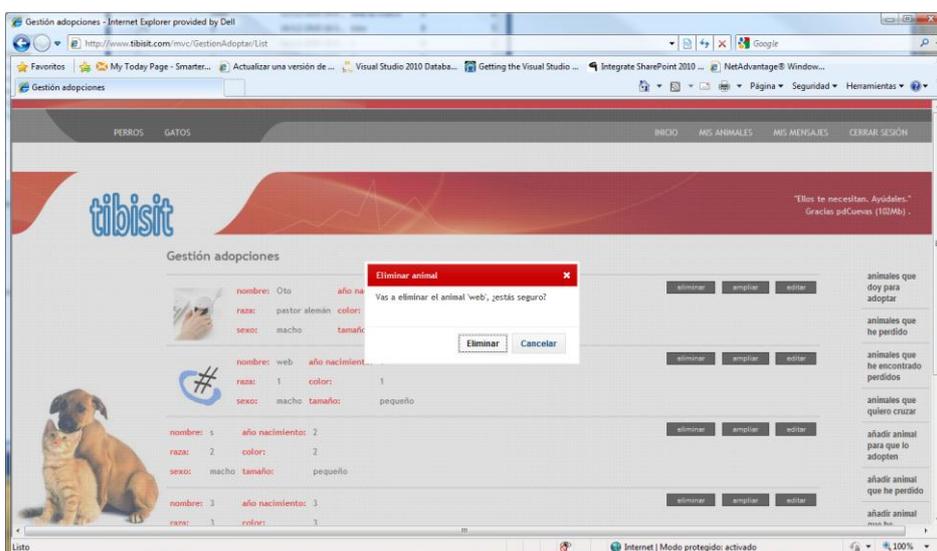
#### 3.1.14.2 jQuery UI dialog.

Para pedir confirmación de borrado, busco una opción más profesional que la de un simple mensaje de confirmación. Tras buscar información en la web de jQuery (<http://jquery.com/>) me encontré con que existía un UI que implementaba distintas funcionalidades e incluso con distintos temas.

Tras examinar la documentación al respecto y documentarme en distintos ejemplos, decidí utilizar el “widget dialog” para mostrar las ventanas de mensajes de confirmación.

Como se ve en el punto anterior, por cada animal cree un “div” oculto con un “id” distinto. En la vista “List” añadí el código jQuery que muestra este “div” y llama a la acción “Delete” del controlador de gestión correspondiente, en caso de que el usuario confirme el borrado. Además, oculta el “div” del animal borrado, de modo, que no es necesario volver a cargar la página.

Para poder utilizar estas librerías añadí los links CDN al UI de jQuery.



### 3.1.15 Gestión de mensajes.

#### 3.1.15.1 Posibilidad de enviar mensajes sobre un animal.

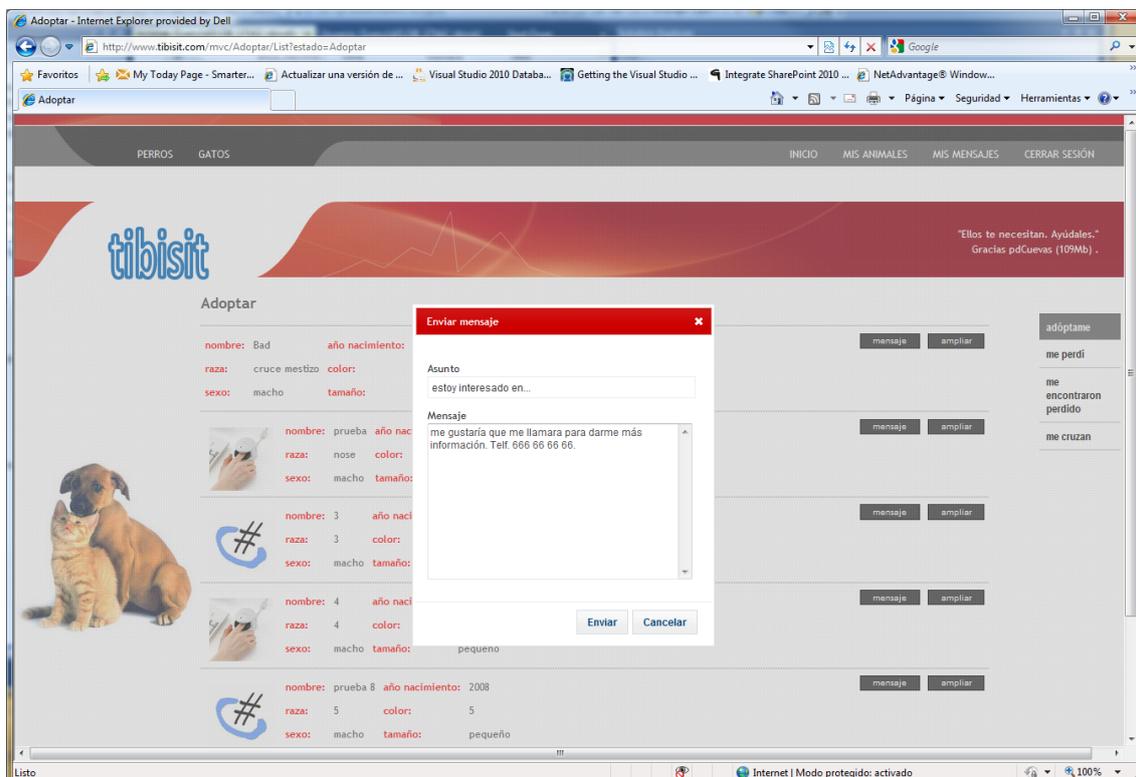
En la vista "List", añadí el botón enviar mensaje. Esto permite a un usuario enviar un mensaje al propietario de un animal, indicándole que está interesado en él y dándole los datos de contacto.

#### 3.1.15.2 Acción EnviarMensaje.

En los controladores "AdoptarController", "PerdidoController", "EncontradoController" y "CruzarController", añado la acción "EnviarMensaje" que envía un mensaje interno al dueño del animal. Además envía un email (mediante el servidor SMTP indicado en el "Web.Config") al dueño del animal, informándole de que ha recibido un mensaje en la web.

#### 3.1.15.3 jQuery UI dialog.

Una vez comprendido el funcionamiento de este UI, decido utilizarlo también para enviar mensajes. Su implementación fue más complicada que en el caso anterior, ya que no solo se trataba de un mensaje de confirmación, sino que el usuario iba a rellenar el asunto y el mensaje en una ventana del UI, y la información de estos campos, debía enviarse a la acción del controlador, para que se generara el mensaje.



#### 3.1.15.4 Controlador MensajeController.

Para gestionar los mensajes que se envían los usuarios, creé este controlador, el cual tiene las acciones necesarias para acceder a los registros de mensajes enviados y recibidos. También incluye las acciones propias para eliminar o leer un mensaje.

También incluye la acción que muestra el menú de la derecha, que permite al usuario elegir entre mensajes recibidos o enviados.

### 3.1.15.5 Modelo MensajeListViewModel.

Las acciones que comparten información con la vista lo hacen mediante este modelo, que guarda la información de los mensajes a mostrar, dependiendo de la página solicitada, y también guarda la información de la clase "PagingInfo" (que ya he descrito con anterioridad).

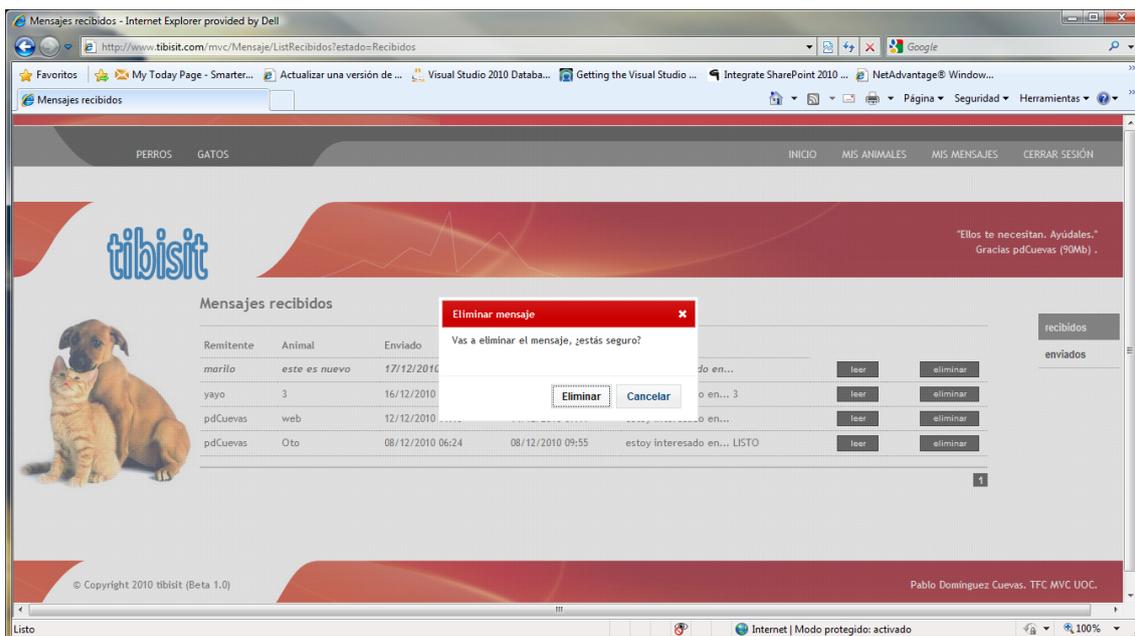
### 3.1.15.6 Vistas.

Creé una vista para las acciones que lo requerían, por ejemplo las de mostrar el listado de mensajes y le añadí funcionalidad mediante jQuery.

Las funcionalidades implementadas fueron:

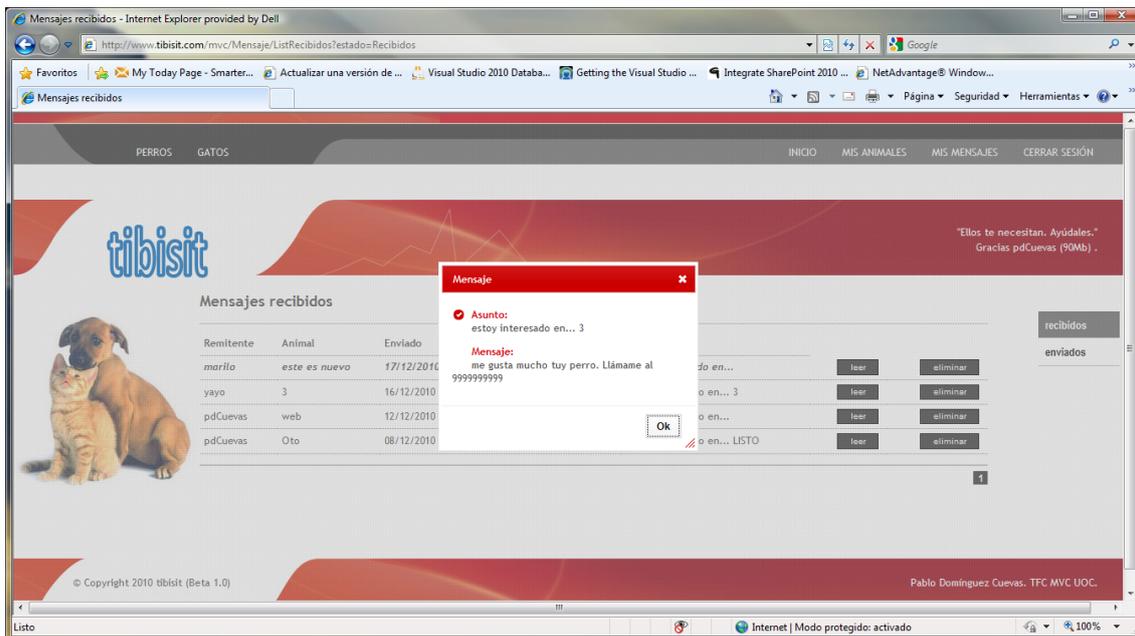
#### 3.1.15.6.1 Eliminar mensaje.

Añadí un botón para que el usuario pudiera borrar los mensajes (acción "Delete" del controlador "MensajeController"). Este botón implementa código jQuery que pide confirmación de borrado al usuario, mediante el jQuery UI dialog.



#### 3.1.15.6.2 Leer mensaje.

También añadí un botón que permitiera leer el mensaje (acción "Show" del controlador "MensajeController"). Este botón muestra una ventana jQuery UI dialog, con los detalles del mensaje. Además, si nos encontramos en el listado de mensajes recibidos, tras leerlo, se modifica el estilo del "div" (`$("#d" + id).removeClass("SinLeer")`) y se indica que el mensaje ha sido leído (`$("#fLeido" + id).text("SÍ")`), como se ve siempre mediante jQuery, por lo que no es necesario volver a cargar la página.



### 3.1.16 Autenticación.

#### 3.1.16.1 Controladores.

Para la autenticación de usuario creo el controlador "AccountController". Este dispone de las acciones necesarias para autenticar al usuario y para que éste se de de alta en el sistema.

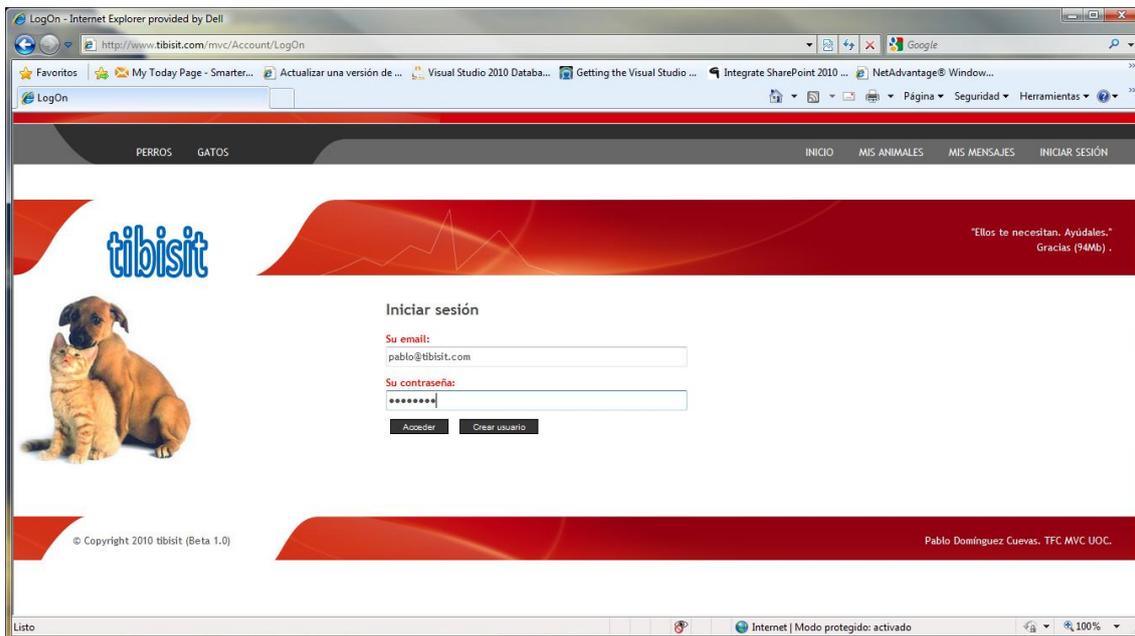
Otra acción importante es la de "UserLog", la cual permite mostrar en la página master la información del usuario conectado, así como la memoria que consume el proceso.

#### 3.1.16.2 Vistas.

Para cada acción se ha creado una vista. Para la acción "LogOn" se creó la vista correspondiente, que permite al usuario introducir su mail y contraseña. En caso de no estar dado de alta, le permite crear un nuevo usuario. Para ello se crea la vista "Create", donde el usuario indica sus datos principales.

#### 3.1.16.3 Autorize.

A la hora de indicar qué páginas se pueden visualizar, solo si se es un usuario autenticado, tan solo hay que añadir la etiqueta "[Authorize]" a los controladores que así lo requieran. Por tanto, la añadí a los controladores "MensajeController" y todos los de gestión del animal.



También cree la vista parcial (“ascx”) “UserLog”, la cual, es llamada desde la página master para mostrar la información de usuario.

## 3.2 Desarrollar web mediante WebForms.

La creación de esta nueva web, fue más rápida y sencilla de lo esperado. Para empezar he aprovechado por completo la capa de datos y de negocio, por tanto, tan solo he tenido que crear una nueva capa de presentación, en la que también he podido aprovechar gran parte de la funcionalidad ya creada para MVC.

*Las páginas que muestra esta web, son idénticas a las de la web MVC, por tanto, no mostraré capturas de pantallas.*

### 3.2.1 Página master.

El primer paso fue modificar la página master, eliminando toda referencia a llamadas a controladores y reemplazándolas por controles de servidor como “asp:Menu”, “asp:LoginView” y “asp:Image”.

### 3.2.2 Página de inicio.

Creé la página de inicio (“default.aspx”) que básicamente no difiere de la de MVC a excepción de que el menú de la derecha se construye con un control de servidor “asp:Menu”.

### 3.2.3 Página para mostrar animales.

Añadí al proyecto la página que muestra los animales (“List.aspx”). El código para cargar los registros se encuentra en la propia página, en lo que se llama ASP.Net Code-Behind. A mi parecer, esta forma es mucho más sencilla e intuitiva para los que alguna vez hemos programado para entornos Windows Forms, por lo que me ha sido más fácil de realizar.

La parte de diseño HTML también es más sencilla que en MVC, ya que usé los controles ListView y DataPager, que realizan todo el trabajo casi automáticamente. Se necesita mucha menos programación que en MVC.

La funcionalidad jQuery la mantuve con las modificaciones propias.

Añadí también la página “/Animales/Show.aspx” para mostrar todos los datos del animal que el usuario puede visualizar.

### 3.2.4 Página de gestión de animales.

La página de gestión es distinta a la anterior, por tener otras funcionalidades, por tanto añadí “MyList.aspx”. Básicamente el diseño es muy parecido a la página anterior, así como el Code-Behind. El resto de funcionalidades también eran muy parecidas a MVC, por lo que pude aprovechar parte del código.

El resto de funcionalidades jQuery, las mantuve con las modificaciones propias para su correcto funcionamiento.

Para editar o añadir nuevos animales cree la página “/Animales/Edit.aspx”, en la que algunas de las variaciones que hice respecto a MVC, fue la de usar el control de servidor “FileUpload”, en lugar del jQuery “uploadify”. La diferencia principal entre ellos, es que ahora la validación de la imagen se hace en el servidor.

#### 3.2.4.1 Validación.

La parte de validación de datos de usuario fue completamente distinta a MVC, por lo que tuve que rehacerla utilizando los controles propios que ofrece WebForms, como “RequiredFieldValidator” y otros.

### 3.2.5 Gestión de mensajes.

A la hora de gestionar mensajes, creé una página para mensajes recibidos (“/Mensajes/Recibidos.aspx”) y otra para enviados (“/Mensajes/Enviados.aspx”). El diseño HTML cambia del MVC, en que utilicé un “ListView” y un “DataPager”, que como ya he comentado es más sencillo que hacerlo todo manualmente, como ocurre en MVC.

La funcionalidad jQuery tampoco varía casi nada.

### 3.2.6 Autenticación.

Esta parte la tuve que rehacer por completo, aprovechando los controles asp que ofrece el modelo, como “asp:Login”. Para implementar esta funcionalidad creé la página “/Account/Login.aspx” y para añadir nuevos usuarios “/Account/Register.aspx”.

### 3.2.7 Control de errores.

Del mismo modo que en MVC, por problemas con el proveedor de hosting, en algunos casos la aplicación puede fallar, por ello, añadí la página que muestra el error “/Error/Unknown.aspx”.

## 4. Hosting: Problemas y soluciones.

Tal y como comenté al consultor, mi intención era aprovechar el PFC para en un futuro cercano, mejorar y publicar la web como un portal abierto a todos los usuarios, de modo que sirviera para ayudar a los animales.

Decidí, por tanto, buscar un hosting donde publicar la web y comprar un dominio “tibisit.com” (te necesitan). Tras una larga búsqueda encontré el que más se ajustaba a mi presupuesto, unos 5\$ al mes, [www.winhost.com](http://www.winhost.com).

Después de hacerme con la consola de administración, creé el directorio para la web y la base de datos. Posteriormente, publiqué desde el IDE de Visual Studio conectando con mi espacio hosting. Estos pasos iniciales fueron bastante rápidos. Sin embargo, los problemas comenzaron tras publicar la web.

### 4.1 Problemas.

#### 4.1.1 Problemas de mapeo.

La web funcionaba perfectamente en mi entorno de desarrollo, pero tras publicarla vi como la aplicación no encontraba las imágenes de la página master, o los scripts, etc. Este problema se corrigió fácilmente, tan solo fue necesario utilizar “Url.Content” para que la url se resolviera correctamente.

#### 4.1.2 Problemas con el estado de sesión.

Tras comenzar a utilizar la web, me di cuenta de que el estado de sesión se perdía constantemente. Tras ponerme en contacto con el soporte técnico, me informaron de que la web superaba los 100Mb de memoria de proceso, por lo que IIS reiniciaba mi aplicación para liberar recursos. Me dieron la opción de pasar a un hosting superior de 200Mb o guardar el estado en SQL Server. Por supuesto, pasar a un hosting superior, no era una opción dado mi presupuesto, por lo que decidí cambiar el almacenamiento del estado.

##### 4.1.2.1 Guardar estado en SQL Server.

Para guardar el estado en SQL Server, tuve que ponerme en contacto con el soporte técnico, ya que es necesario configurar la base de datos, con las tablas necesarias para ello. Una vez me informaron de que ya podía acceder a funcionalidad, modifiqué el “Web.Config” para que el estado de sesión pasase de “InProc” a “SQLServer”.

Con esta solución se corrigió el problema.

#### 4.1.3 Problemas con la autenticación.

El hecho de que IIS reiniciara mi aplicación cada minuto, produjo otro error inesperado. Las cookies de autenticación, expiran y no son válidas tras cada reinicio. Este problema era aun más grave, ya que un usuario autenticado podría estar dando de alta un animal o viendo sus animales y, repentinamente, el sistema volvería a pedirle que se identificara.

Para corregir este problema, tuve que hacer una “chapucilla”, pero que, al no ser un entorno real de producción, la utilizo para evitar este problema. Sin embargo, tengo claro que, si quiero poner la web en producción, no será factible que mi proveedor reinicie mi proceso en ningún momento.

La solución por la que opté, fue la de guardar el mail del usuario en una variable de sesión, de modo que, cuando la aplicación se reinicia y me redirige a la página de login, primero compruebo la variable de sesión, y si tiene algún valor, regenero la cookie, autentico al usuario y lo redirijo a la página original, evitando que se muestre la página de login. Pero quiero que quede claro al consultor, que sé que esto no es permisible en una web en producción, sobre todo por la inseguridad de las variables de sesión, que podría facilitar a un usuario malintencionado, hacerse pasar por un usuario que no es.

#### 4.1.4 Error interno del servidor.

Finalmente el error más grave, que no he podido resolver es que, tras el reinicio del proceso, la aplicación web puede quedar en un estado inconsistente, lo cual provoca errores aleatorios imposibles de controlar.

##### 4.1.4.1 ErrorController.

Para poder mostrar una página de error, en caso de que se produjera este problema, creé el consiguiente controlador y su vista, que únicamente muestra la página de error, indicando que se ha producido un error interno del servidor.

## 4.2 Modificaciones para solucionar problemas.

Todos estos problemas me hicieron intentar recortar la memoria que consumía el proceso, para ello realicé una serie de modificaciones:

### 4.2.1 Caché en disco.

El primer paso, ya comentado, fue implementar una caché en disco, de modo que ésta no se almacenara en memoria. La información la obtuve de <http://www.josecuellar.net/microsoft-asp-net/outputcache-asp-net-4-0-almacenamiento-en-cache-de-salida-2> y <http://www.asp.net/LEARN/whitepapers/aspnet4>. Básicamente se comenta que ASP.net 4.0 amplía las posibilidades de anteriores frameworks, con la posibilidad de guardar la caché en disco.

La caché se implementó en el proyecto tibisib.CacheProvider. Este proyecto guarda las páginas que llevan la etiqueta `<%@ OutputCache VaryByParam="*" Duration="15" %>` en el directorio “/Cache”, el cual se indica en el “Web.Config”.

El funcionamiento es sencillo, cada vez que se llama a una página, primero se busca en la caché, si existe, se carga ésta, y si no, se guarda en disco para su próxima llamada. Las páginas expiran a los 15 minutos (configurable), y se genera un fichero en caché por cada página dependiendo de los parámetros que se le pasen.

Para que funcionara la caché en el hosting, tuve que modificar el “Web.Config” para que la aplicación tuviera privilegios “trustLevel” como “Full”.

#### 4.2.2 Session State.

El paso siguiente fue, como ya he comentado, activar el estado en modo SQL Server.

#### 4.2.3 CDN.

También cambié todos los links posibles a CDN (<http://www.asp.net/ajaxlibrary/cdn.ashx>), se supone que esta acción debe mejorar el rendimiento de la aplicación, ya que las librerías se encuentran en caché de todo tipo de servidores alrededor del mundo.

Como curiosidad, me encontré con que las librerías ajax para MVC 2, todavía no están disponibles.

#### 4.2.4 Watermark.

Como he mencionado, antes de mostrar una imagen, le añadía un “watermark”. Opté por no apareciera en las imágenes en miniatura que se muestran en las acciones “List”. Dejé esta funcionalidad únicamente para la acción “Show”, que muestra la imagen en un tamaño mayor. Esperaba reducir de este modo, la memoria y el consumo de CPU que se empleaba para visualizar las imágenes de los animales.



#### 4.2.5 Enviar email con errores.

Para estar informado de cuándo la aplicación rompía por algún error, o era reiniciada por IIS, añadí al “Global.asax” la funcionalidad necesaria para que me enviara un correo con la información del error.

#### 4.2.6 Dispose.

Decidí añadir el método “Dispose” a los repositorios, para ello modifiqué el T4 “IRepository.tt” añadiendo este método, incluyendo incluso una llamada al GC (recolector de elementos) para que lleve a cabo una recolección de los elementos no utilizados mediante “GC.Collect()”. He podido observar que esta acción no está recomendada, pero no tuve otra opción, ya que no podía esperar a que ésta se realizara automáticamente.

Ahora en cada acción del controlador, hacía un “Dispose” de todos los repositorios.

### 4.3 Windows Azure Platform.

Para evitar los problemas del proveedor, decidí estudiar la migración a una plataforma sin limitaciones, totalmente escalable, de hecho, se paga por consumo, por tanto, la aplicación puede crecer casi sin limitaciones. Windows Azure Platform es un grupo de tecnologías en la nube que permiten que una aplicación corra en los data centers de Microsoft.

### 4.3.1 Principales modificaciones.

Para que la web funcione en esta plataforma he tenido que realizar algunas modificaciones, y al ser una plataforma tan reciente, todavía existe poca documentación, por lo que las pruebas y depuración de problemas, ha sido bastante costosa.

#### 4.3.1.1 Modificaciones propias.

Visual Studio dispone de un “template” propio para los proyecto en la nube. Por ello es necesario unir la web existente a un proyecto de este tipo, realizando las modificaciones oportunas para que la web pueda publicarse en Azure. Al no ser motivo de este proyecto no entraré en más detalles, pero se puede consultar la información en <http://blogs.msdn.com/b/jnak/archive/2010/02/08/migrating-an-existing-asp-net-app-to-run-on-windows-azure.aspx>.

#### 4.3.1.2 Guardar estado de sesión.

Inicialmente, no se permite guardar el estado de sesión en SQL, aunque no oficialmente, es posible hacerlo, pero dado el coste económico que también supondría esta opción, opté por guardarlo en el sistema mediante el servicio Table Storage, uno de los tres proporcionados por Azure, permite almacenar la información de forma masiva, en forma de entidades, las cuales a su vez tienen propiedades.

#### 4.3.1.3 Subir imágenes al servidor.

En esta plataforma no es posible establecer un directorio al que subir las imágenes, hay que entender que la aplicación corre en diferentes instancias, y cada una puede correr en distinto hardware, por tanto, lo que ocurre es que podemos almacenar objetos blob en un contenedor, al que nos referiremos mediante una URL. Esto implica que haya tenido que modificar este aspecto, por lo que en lugar de subir imágenes a un directorio del sistema de ficheros, debo utilizar el servicio Blob Storage. Para manejar estos ficheros blob podemos utilizar REST API o bien utilizar la API correspondiente proporcionada por Microsoft, que es la opción que yo elegí.

#### 4.3.1.4 SQL Azure.

En cuanto a la base de datos, la migré a este SGBD, que puede ser manejado con las mismas herramientas que SQL Server, con las limitaciones que implica que se ejecute en los data centers de Microsoft. Este cambio no necesitó de ninguna modificación en el Entity Framework, más allá de cambiar la cadena de conexión.

## 5. Diferencias MVC y WebForms.

### 5.1 MVC.

Podríamos definir MVC como un patrón arquitectural que describe una forma de desarrollar aplicaciones software separando los componentes en tres grupos (o capas):

- El Modelo que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.

- La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
- El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

MVC son las siglas de Modelo-Vista-Controlador, y se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

## 5.2 Ventajas MVC.

- Clara separación de responsabilidades entre interfaz, lógica de negocio y de control, que además provoca parte de las ventajas siguientes.
- Produce un código más limpio y estructurado, independizando totalmente la interfaz de la lógica de navegación y, por supuesto, de la de negocio.
- Facilidad para la realización de pruebas unitarias de los componentes, así como de aplicar desarrollo guiado por pruebas (TDD) y técnicas avanzadas de mocking.
- Simplicidad en el desarrollo y mantenimiento de los sistemas: el conjunto de convenciones en cuanto a la estructura de proyectos y de nombrado y disposición de elementos facilita el desarrollo una vez son asimiladas.
- Reutilización de los componentes.
- Facilidad para desarrollar prototipos rápidos.
- Sencillez para crear distintas representaciones de los mismos datos.
- Los sistemas son muy eficientes, y a la postre más escalables.
- Fácilmente se puede utilizar DI (dependency injection): es una técnica que permite realizar aplicaciones cuyos componentes se encuentran muy desacoplados entre sí, lo que flexibiliza el diseño y, por ejemplo, facilita la realización de pruebas unitarias.
- Se trata de un patrón muy fácilmente implementable y que nos puede aportar muchos beneficios.
- Integración sencilla del framework MVC con soluciones basadas en cliente, como jQuery y su interminable colección de plugins, en los que podemos encontrar elementos de interfaz para prácticamente cualquier necesidad.
- Las direcciones amigables, es un beneficio directo del uso del framework de Microsoft, estrictamente hablando no es mérito de la plataforma MVC, sino del juego de clases presentes en el espacio de nombres System.Web.Routing, incluidas en .NET, con las ventajas que ello conlleva (SEO, REST, claridad en direcciones, etc.).
- La ausencia de automatismos y persistencia de estado aligera en gran medida el peso y complejidad de las páginas, lo cual redundará en el rendimiento del sistema.
- Existen multitud de frameworks MVC para ASP.Net, como MonoRail, Maverick.Net, FubuMVC y muchos otros.
- Es un framework con licencia MS-PL (Microsoft Public License), por tanto es libre, y permite su uso en aplicaciones comerciales.

- Flexibilidad de la arquitectura de ASP.NET MVC framework en la utilización de motores de vistas distintos al estándar.

### 5.3 Inconvenientes MVC.

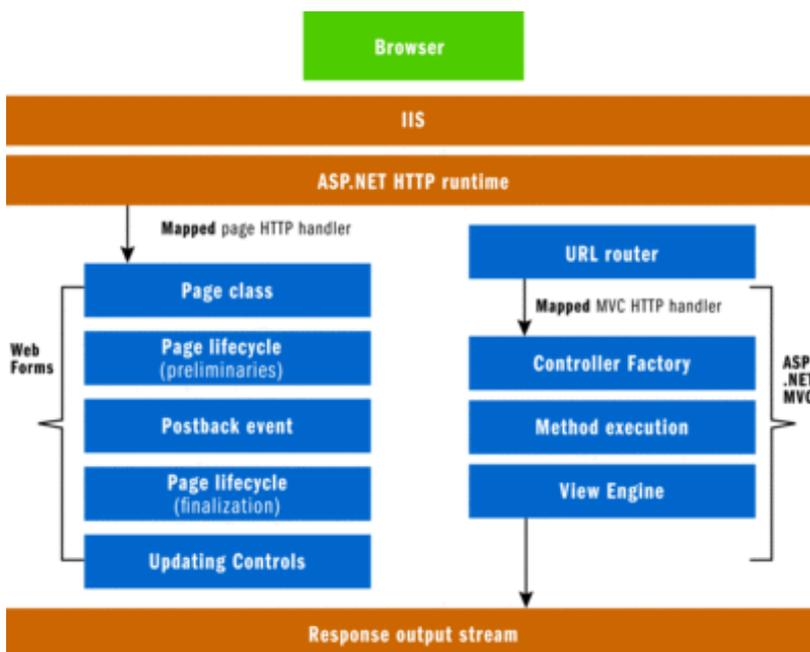
- Tener que ceñirse a una estructura predefinida, lo que a veces puede incrementar la complejidad del proyecto. De hecho, hay problemas que son más difíciles de resolver.
- Al principio cuesta cierto esfuerzo adaptarse a esta filosofía, sobre todo a desarrolladores acostumbrados a otros modelos más cercanos al escritorio, como Winforms.
- La distribución de componentes obliga a crear y mantener un mayor número de ficheros.
- En general requiere de más código que WebForms.
- Existe un número ingente de componentes y controles reutilizables disponibles para Webforms. Dado que no son compatibles con el framework MVC, se parte de una situación de clara desventaja frente a estos, aunque esto está ya cambiando y seguro que con el tiempo mejorará.
- Requiere un conocimiento más profundo del entorno web y sus tecnologías subyacentes, puesto que a la vez que ofrece un control mucho más riguroso sobre los datos que se envían y reciben desde el cliente, exige una mayor responsabilidad por parte del desarrollador, ya que deberá encargarse él mismo de mantener el estado entre peticiones, maquetar las vistas, crear las hojas de estilo apropiadas, e incluso los scripts.

### 5.4 Ventajas WebForms.

- La tecnología de formularios web permite el desarrollo rápido de aplicaciones (RAD) a través de diseñadores visuales con los que es posible componer una página compleja y definir el comportamiento del interfaz a golpe de ratón, puesto que el framework se encarga de realizar parte del trabajo duro, como el mantenimiento del estado entre peticiones, convertir propiedades de controles en código HTML y CSS, o incluso generar scripts que realicen determinadas tareas en cliente.
- Es posible crear aplicaciones para Internet sin tener apenas idea de las particularidades inherentes al desarrollo web, lo que permite que muchos programadores procedentes del mundo del escritorio puedan ser productivos muy rápidamente, aunque sea a costa de generar páginas mucho más pesadas y con un código de marcado complejo.
- Si el equipo de desarrollo tiene ya experiencia creando aplicaciones con WinForms y no posee grandes conocimientos sobre programación web de más bajo nivel ni experiencia previa trabajando con el patrón MVC, esta tecnología va a permitir una mayor productividad del equipo.
- La última versión del framework ya permite direcciones amigables.



- Control total del código de marcado generado.
- Podemos sustituir componentes internos del framework para adaptarlo a nuestras preferencias.
- Se integra con Ajax de forma natural, sin artificios como los UpdatePanels y similares.
- Favorece la introducción de buenas prácticas como la inversión de control o inyección de dependencias.
- Diferencias en la pila de ejecución:



- Diferencias entre code-behind y MVC controllers: aparentemente hay similitudes entre ambos, ya que estos contienen la lógica de la aplicación, sin embargo la página aspx no se puede separar del code-behind, ambas trabajan unidas para implementar la lógica de la aplicación y la lógica de presentación, pero en MVC es diferente, hay una clara separación entre la vista (UI) y los controladores, que son una representación abstracta de la interacción con el usuario. Esto permite un código más simple, por lo que la aplicación es más fácil de entender y mantener.

### 5.7 Experiencia personal.

Particularmente, tras realizar ambas web, las mayores diferencias que he encontrado ya se han comentado con anterioridad, pero enfatizaría la facilidad que proporciona WebForms, seguramente por su proximidad a WinForms, el uso de controles de servidor implica no tener que hacerlo prácticamente todo manual, como en MVC. La cantidad de código de la primera, refiriéndome solo a la capa de presentación, es casi la mitad que en MVC. Sin embargo, en ésta, al controlar exactamente el código HTML generado, no he tenido tantos problemas a la hora de presentar los datos en la página. La validación sin embargo, me ha parecido más simple en MVC gracias al uso de metadata en la definición de las clases.

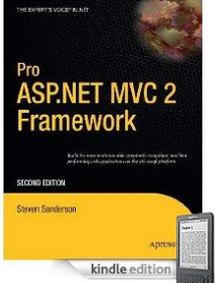
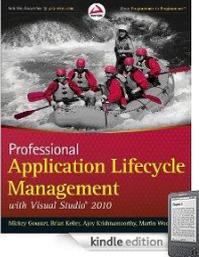
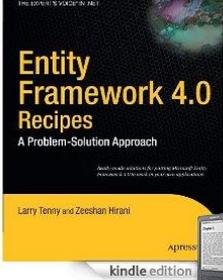
## 6. Conclusiones.

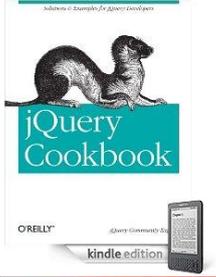
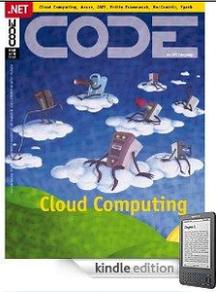
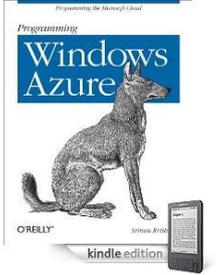
Desde el nacimiento de la World Wide Web hasta la actualidad, hay un verdadero abismo tecnológico y conceptual. No solo se ha evolucionado en la parte visual que utiliza el usuario, sino en la parte que no se ve. ASP.NET permite crear aplicaciones web de un modo muy productivo, proporcionando una visión moderna, interactiva y escalable de la red. Las mejoras ofrecidas por Visual Studio 2010, ASP.NET 4.0 y en general todas las funcionalidades que permite el framework 4, permiten desarrollar una web de calidad en poco tiempo.

Visual Studio 2010 me ha permitido desarrollar la aplicación desde las primeras fases, integrándose perfectamente con todas las herramientas de Microsoft para facilitar el trabajo.

A pesar de las facilidades que proporciona WebForms, pienso que para una aplicación de una determinada envergadura, la mejor opción es MVC, por todas las ventajas y pocas desventajas, que ya se han comentado. A pesar de que tiene una mayor curva de aprendizaje que WebForms y de que la productividad es menor en un primer momento, a la larga, MVC será más fácil de entender y mantener. Pienso que es el patrón con más futuro a corto plazo y medio plazo, aunque WebForms no desaparecerá, por su simplicidad.

## 7. Bibliografía.

	<p><b>Pro ASP.NET MVC 2 Framework, Second Edition [Kindle Edition]</b>  <a href="#">Steven Sanderson</a> (Author)</p>
	<p><b>Professional Application Lifecycle Management with Visual Studio 2010 [Kindle Edition]</b>  <a href="#">Mickey Goussset</a> (Author), <a href="#">Brian Keller</a> (Author), <a href="#">Ajoy Krishnamoorthy</a> (Author), <a href="#">Martin Woodward</a> (Author)</p>
	<p><b>Entity Framework 4.0 Recipes: A Problem-Solution Approach [Kindle Edition]</b>  <a href="#">Larry Tenny</a> (Author), <a href="#">Zeeshan Hirani</a> (Author)</p>

	<p><b>jQuery Cookbook [Kindle Edition]</b>  <a href="#">Cody Lindley</a> (Author)</p>
	<p><b>Microsoft® .NET: Architecting Applications for the Enterprise [Kindle Edition]</b>  <a href="#">Dino Esposito</a> (Author), <a href="#">Andrea Saltarello</a> (Author)</p>
	<p><b>CODE Magazine - 2010 MarApr [Kindle Edition]</b>  <a href="#">Rod Paddock</a> (Author), <a href="#">Paul Sheriff</a> (Author), <a href="#">Sahil Malik</a> (Author), <a href="#">Chris Williams</a> (Author), <a href="#">Donn Felker</a> (Author), <a href="#">Jeff Certain</a> (Author), <a href="#">John V Petersen</a> (Author), <a href="#">ANDREW(Intergen) TOKELEY</a> (Author), <a href="#">Chris Auld</a> (Author), <a href="#">CODE Magazine</a> (Editor)</p>
	<p><b>Programming Windows Azure [Kindle Edition]</b>  <a href="#">Sriram Krishnan</a> (Author)</p>
	<p><b>Foundation Expression Blend 4 with Silverlight [Kindle Edition]</b>  <a href="#">Victor Gaudio</a> (Author)</p>

Fuentes web principales:

<http://thinkingindotnet.wordpress.com/2008/02/13/actualizacion-del-road-map-del-aspnet-mvc-framework/>

<http://www.variablenotfound.com/2010/05/aspnet-mvc-2-quince-cuestiones-que.html>

<http://www.switchonthecode.com/tutorials/csharp-tutorial-image-editing-saving-cropping-and-resizing>

<http://www.variablenotfound.com/2008/04/combinando-aspnet-mvc-framework-y.html>

<http://dotnetslackers.com/articles/aspnet/Complexity-in-ASP-NET-MVC-Part-2-Plan-your-AJAX-Carefully.aspx>

<http://weblogs.asp.net/carloslone/default.aspx>

<http://geeks.ms/blogs/gtorres/archive/2010/03/11/los-controles-de-telerik-para-asp-net-mvc.aspx>

<http://code.msdn.microsoft.com/michelotti>

<http://code.google.com/p/swfupload/>

<http://haacked.com/archive/2010/07/16/uploading-files-with-aspnetmvc.aspx>

<http://www.asp.net/learn/whitepapers/aspnet4>

<http://www.asp.net/mvc>

<http://www.asp.net/web-forms>

<http://www.asp.net/ajax>

<http://jquery.com/>

<http://channel9.msdn.com/Blogs/VisualStudio/Test-Driven-Development-with-Visual-Studio-2010>

<http://msdn.microsoft.com/es-es/library/cc716683.aspx>

<http://msdn.microsoft.com/en-us/library/dd394709.aspx>

<http://msdn.microsoft.com/en-us/magazine/dd942833.aspx>

[http://www.highoncoding.com/Articles/689\\_Upload\\_and\\_Displaying\\_Files\\_Using\\_ASP\\_NET\\_MVC\\_Framework.aspx](http://www.highoncoding.com/Articles/689_Upload_and_Displaying_Files_Using_ASP_NET_MVC_Framework.aspx)

<http://stackoverflow.com/questions/1002680/how-do-i-get-jquerys-uploadify-plugin-to-work-with-asp-net-mvc>

<http://www.uploadify.com/>

[http://expression.microsoft.com/es-es/cc197140\(en-us\).aspx](http://expression.microsoft.com/es-es/cc197140(en-us).aspx)

<http://gallery.expression.microsoft.com/en-us/>

<http://blog.stevensanderson.com/2010/06/11/pro-aspnet-mvc-2-framework/>

<http://blog.maartenballiauw.be/post/2009/11/26/Supporting-multiple-submit-buttons-on-an-ASPNET-MVC-view.aspx>

<http://www.winhost.com/index.aspx>

<http://www.microsoft.com/windowsazure/>