UNIVERSIDAD REY JUAN CARLOS

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER UNIVERSITARIO EN INFORMÁTICA INTERACTIVA Y MULTIMEDIA

Curso Académico 2010/2011

Trabajo Fin de Máster

TÍTULO DEL PROYECTO

Integrando las tecnologías .NET (MVC - EF) y jQuery UI para la generación automática de aplicaciones Web

Autor: Luis Miguel Rivera Cumbicus

Tutor: Paloma Cáceres García de Marina

RESUMEN

El objetivo de este Trabajo Fin de Máster es facilitar el desarrollo de aplicaciones Web de alta calidad, mediante la generación automática del código y las vistas, aprovechando la integración del patrón MVC (Modelo Vista Controlador), del patrón Code First de Entity FrameWork 4.1, del conjunto de bibliotecas de funciones jQuery UI, y de la nueva metodología para el diseño Web llamada Responsive Web Design (RWD).

El patrón MVC proporciona la arquitectura o infraestructura para desarrollar las aplicaciones, ofreciendo la "Separación de Responsabilidades o Soc (Separation of Concerns)", es decir separa la aplicación en tres capas: Modelo (Lógica del Negocio), Vista (Interfaz de usuario) y el Controlador (conecta el modelo con las vistas).

El patrón Code First de Entityframork4.1, facilita la creación de los objetos necesarios que representan la base de datos de la aplicación, generando de manera casi automática la representación en objetos las distintas tablas que componen la base de datos.

Las bibliotecas de funciones jQuery UI (JavaScript) son un conjunto de plug-ins que incluyen código de fácil utilización que permite la creación de interfaces de usuario mucho más interactivas y amigables, de manera eficiente y rápida.

Responsive Web Design (RWD) es una nueva metodología para desarrollar aplicaciones Web que se adaptan a los distintos dispositivos en los que pueden ser visualizados (smartphone, tablets, PC, portátiles, TV, etc). Se hace uso de diseños flexibles y fluidos que se adaptan a casi cualquier dispositivo.

Para integrar estos patrones junto con jQuery UI se ha desarrollado una aplicación práctica de un sitio Web para el grupo de investigación "Estudio Técnico de la Obra de Sorolla" [ETOS] de la Universidad Rey Juan Carlos, mediante el IDE (Entorno de Desarrollo Integrado) Visual Studio 2010 con ASP.NET MVC 3 Tools y SQL Server 2008 R2 como servidor de la base de datos.

Tabla de contenido

Indice de tablas

Indice de figuras

CAPÍTULO 1 INTRODUCCION

1.1 Presentación del problema

En este Trabajo de Fin de Máster se pretende demostrar que con la integración de los patrones MVC y Code First de Entityframework mas el conjunto de plug-ins de jQuery UI se permite desarrollar de manera rápida y eficiente sitios Web como aplicaciones de negocios con acceso a datos de manera más dinámica y amigable.
Tambien se pretende aplicar la nueva metodología de Responsive Wed Design para demostar que la creación de un sitio Web se adapate a los distintos dispositivos, pantallas y orientación en las que van a ser visualizados.

1.2 Motivación

Actualmente el desarrollo de aplicaciones Web con acceso a datos implica cierta complejidad. Las dificultades surgen fundamentalmente por la arquitectura cliente-servidor del protocolo HTTP, lo cual obliga a una división en el desarrollo; por un lado, el código que se ejecuta en el servidor, y por otro el que se ejecuta en el cliente (navegador) una vez que se descarga la página web. Ambas partes tienen que estar en sintonía para generar una aplicación que contenga una interfaz de usuario funcional y amigable (al estilo de la aplicaciones de escritorio) junto a un comportamiento (generalmente guiado por el servidor) que esconda la complejidad del protocolo HTTP.

Por esta razón han surgido las RIA (Rich Internet Application), tales como Microsoft Silverlight, Adobe Flex, etc. El problema con las RIA radica en que se exige instalar componentes adicionales en el navegador para poder ejecutar la aplicación (Silverlight, Flash).

El patrón MVC es ampliamente conocido y se aplica en el desarrollo de aplicaciones de escritorio, su utilizacion en la construcción de aplicaciones de negocios orientados a la Web facilita en gran medida su desarrollo e implementación con la separación de responsabilidades en las tres capas de la aplicación: Modelo, Vista y Controlador.

ASP.NET MVC 3 utiliza **Razor,** una nueva forma de implementar código de servidor en las etiquetas de HTML/ASP.NET mediante sintaxis de C# o de VB.NET para generar paginas dinámicas.

Microsoft ADO.NET Entity Framework (EF) simplica el acceso a los datos permitiendo no trabajar directamente con la base de datos, sino que lo realiza a través de unas instancias que actúan como intermediarios entre esta y el contexto del modelo del dominio.

EF ofrece tres maneras para describir el modelo de las entidades. Se puede comenzar con una base de datos existente (Data First) para crear un modelo. Se puede crear el modelo directamente en el diseñador (Model First) o definir las clases (Code First) para generar el modelo.

Muchos desarrolladores utilizan JavaScript que es un lenguaje pequeño y limitado en comparación con los otros lenguajes de programación (Java, ASP.NET, etc.), pero que es suficiente para desarrollar aplicaciones web interesantes gracias a la disponibilidad de un conjunto de bibliotecas de funciones, como jQuery y en especial jQuery UI, que incluyen mucho código de fácil utilización para desarrollar interfaces amigables, interactivas y de mucha calidad y que se ejecutan en cualquier navegador web o sistema operativo.

1.2 Objetivos

Este trabajo pretende proporcionar un marco de trabajo basado en la integración de los patrones ASP.NET MVC 3 Tools y Code First EF v4.1, junto con la biblioteca jQuery UI del framework jQuery, para la generación automática de todos los componentes de una aplicación Empresarial Web completa: acceso a datos, controladores y vistas dinamicas.
La aplicación generada puede usarse como un prototipo para su posterior desarrollo rápido por parte de desarrolladores y diseñadores de distinto nivel de experiencia, ganando con ello simplicidad y productividad.

Además al desarrollo de la aplicación, también se desea aplicar la nueva manera de diseñar y crear una aplicación web, independientemente del dispositivo, pantalla y orientación de estas en las que van a ser visualizadas. De esta forma en vez de crear distintas plantillas o sitios webs (escritorio, tablet, Smartphone, etc), hacemos que se adapte a los distintos dispositivos.

Para la implementación de esta propuesta se ha desarrollado el sitio Web del grupo de investigación "Estudio técnico de la Obra de Sorolla" [ETOS] de esta universidad [WebETOS].

1.3 Método de trabajo

1.3.1 Estudios previos

En este apartado se han estudiado los diferentes frameworks para el desarrollo de aplicaciones Web con acceso a datos: Desarrollo con ASP.NET WebForm y tecnología AJAX, junto con ASP.NET MVC 3 con el framework ASP.NET 4.

Se estudia el nuevo marco de acceso a datos desarrollado por Microsft: Entity FrameWork 4.1 con su patrón Code First.

Se realiza una comparativa entre el desarrollo con ASP.NET WebForm y ASP.NET MVC 3.

Se hace un repaso de los patrones, sus orígenes, clasificación y aplicaciones.

Se estudia el nuevo conjunto de plug-ins de jQuery UI, su estado actual y su aplicación en aplicaciones de sitios Web con acceso a datos para aumentar su usabilidad.

Se estudia el Responsive Web Design para ser implentados en el diseño y creación de sitios Web.

1.3.2 Desarrollo de la propuesta

Una vez estudiados y analizadas las diversas tecnologías explicados en los apartados anteriores, se ha optado por utilizar el nuevo paradigma de desarrollo para sitios Web: ASP.NET MVC 3, Code First. Con el fin de realizar las vistas más amigables y dinámicas se decide utilizar jQuery UI.
Para que el sitio Web se adapte a los diferentes dispositivos desde los que se puede ser accedido se utiliza Responsive Web Design (Diseño flexible).

1.3.3 Validacion de la propuesta

Una vez realizadas las pruebas de desarrollo e implementación de manera local en un ordenador portátil, se subió la aplicación a un proveedor de servicio host llamado WinHost [Winhost] alojando la base de datos en su servidor SQL Server 2008 R2.
Una vez realizada la implementacion en este proveedor host se realizó la interaccion real de los usuarios con la aplicación Web, realizando los ajustes necesarios requeridos por estos usuarios, la aplicación Web se ajusto a sus requerimientos, no obstante la mejora es continua a través del tiempo.

1.4 Tecnologias: Hardware y Software

1.4.1 Hardware

- Ordenador portátil Asus X52J
- S.O Windows 7, RAM 4 Gb., disco duro de 500 Gb.

1.4.2 Software

Se utilizarón las siguientes tecnologías:

- IDE: Visual Studio 2010 Ultimate.
- ASP.NET 4
- ASP.NET MVC3 Tools.
- ADO Entity Framework 4.1 con CodeFirst y la generacion de las clases POCO que sirven de puente entre las vistas y la base de datos.

- SQL Server 2008 R2 como servidor de la base de datos.
- Jquery UI 1.7 para crear vistas más dinámicas y aumentar su usabilidad.
- Razor como nueva sintaxis para incrustar código del servidor en las vistas.
- Responsive Web Design (fluid grid (cuadriculas o redes flexibles), media queries (consulta de medios), fluid images (imágenes fluidas))

1.5 Estructura del documento

El presente capitulo presenta los objetivos y motivaciones que ha llevado a desarrollar este trabajo.

El capítulo 2 hace un repaso del Estado del Arte de los patrones, desde su definición, historia, propiedades, clasificación, del patrón MVC, estructura, ventajas, propiedades, enrutamiento, habla también del patrón Code First de Entity Framework, su historia hasta la última versión 4.1. Trata también de jQuery y el conjunto de bibliotecas de jQuery UI para jQuery. En este capitulo también se estudia el Responsive Web Design para sitios Web.

El capítulo 3 presenta la implementación práctica de toda la teoría tratada en los capitulo anteriores. Indicando los pasos para el desarrollo del sitio Web del grupo de investigación ETOS. Desde la creación de la base de datos, modificación de las vistas y clases generadas automáticamente.

El capítulo 4 resume las principales conclusiones y trabajos futuros a realizar

CAPÍTULO 2: ESTUDIOS PREVIOS

2.1 Historia de los Patrones

La notación formal de los patrones nació con los patrones arquitectónicos de Christopher Alexander [Alexander,1979]. Da la siguiente definición de patrón: "Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma".

Propone, así, un paradigma para la arquitectura basado en tres conceptos: la cualidad, la puerta y el camino.

- La Cualidad (la Cualidad Sin Nombre): la esencia de todas las cosas vivas y útiles que nos hacen sentir vivos, nos da satisfacción y mejora la condición humana.
- La Puerta: el mecanismo que nos permite alcanzar la calidad. Se manifiesta como un lenguaje común de patrones. La puerta es el conducto hacia la calidad.
- El Camino (El Camino Eterno): siguiendo el camino, se puede atravesar la puerta para llegar a la calidad.

En 1987 - Ward Cunningham y Kent Beck usaron varias de las ideas de Alexander para desarrollar un pequeño lenguaje de patrones para programadores nóveles de Smalltalk.

En 1991 - Jim Coplien publica Advanced C++ Programming Styles and Idioms que compila un catálogo de idioms para C++.

De 1991 a 1995, Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides (conocidos como la "Banda de los cuatro" GoF) utilizaron su experiencia combinada para escribir el libro Design Patterns: Elements of Reusable Object-Oriented Software [GoF,1995].

2.2 Definición de Patrón

El patrón es una descripción de un problema y su solución en un contexto especifico, que recibe un nombre y se puede emplear en otros contextos; en teoría indica la manera de utilizarlo en circunstancias diversas.
Es por ello que los patrones de diseño benefician a los desarrolladores de sistemas al:

- Ayudar a crear software confiable utilizando arquitecturas comprobadas y la experiencia acumulada en la industria.
- Promover la reutilización del diseño en sistemas posteriores.
- Ayudar a identificar los errores y obstáculos comunes que ocurren al crear sistemas.
- Ayudar a diseñar sistemas, independientemente del lenguaje en el que se vaya a implementar.
- Establecer un vocabulario de diseño común entre los desarrolladores.
- Acortar la fase de diseño en un proceso de desarrollo de software.

Los patrones son una disciplina de resolución de problemas reciente en la ingeniería del software que ha emergido en mayor medida de la comunidad de orientación a objetos, aunque pueden ser aplicados en cualquier ámbito de la informática y las ciencias en general.

Los patrones son el medio idóneo para compartir experiencia y conocimiento [Welicki,2006]. Cada patrón es una relación entre un contexto, un sistema de fuerzas que ocurren repetidamente en ese contexto y una configuración espacial que permite que esas fuerzas se resuelvan entre sí [Alexander,1979]. Los patrones permiten codificar el conocimiento en forma uniforme. Compartir patrones es compartir conocimiento.

"Una patrón es una pieza de conocimiento que incluye información sobre un problema y su solución en un contexto, con sus ventajas e inconvenientes y toda la información necesaria para poder tener un buen entendimiento de los aspectos relacionados con él. Eventualmente puede contener información específica sobre su comportamiento, la cual permite generar artefactos de software" [Welicki,2006].

Definición de patrón según Alexander: contexto – problema – solución [Alexander,1979]

Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma.

Los miembros del mentado GoF [GoF,1995] destacan que los patrones no son un destino sino un punto de partida.

A tener en cuenta:

- Los patrones son un punto de partida, no un destino
- Los modelos no están bien o mal, sino que son más o menos útiles
- Un patrón de diseño no es una solución en sí mismo, sino la documentación de la forma en que se construyeron soluciones a problemas similares en el pasado, lo cual permite una mejor gestión de la experiencia y transferencia de conocimientos
- Los patrones de diseño no son dogmas que deben ser aceptados sin cuestionarlos.
- Qué es y qué no es un patrón de diseño es una cuestión que depende del punto de vista de cada uno y del nivel de abstracción en que se trabaje.
- Una de las heurísticas para establecer un patrón es que existan al menos 3 implementaciones en aplicaciones reales.
- No son abstracciones teóricas, sino que están fundados sobre una fuerte base práctica y pragmática, producto de la experiencia

2.3 Patrones y gestión del Conocimiento

Sumando la cantidad de patrones publicados en tres de los libros más influyentes de la literatura de patrones, "Design Patterns: Elements of Reusable Object Oriented Software" [GoF,1995], "Pattern Oriented Software Architecture, Volume 1" [POSA,1996] y "Patterns of Enterprise Application Architecture" [Fowler,2002], el número total es 100 [Welicki,2006].

Los patrones permiten establecer un vocabulario común de diseño, cambiando el nivel de abstracción a colaboraciones entre clases y permitiendo comunicar experiencia sobre dichos problemas y soluciones.

Son también un gran mecanismo de comunicación para transmitir la experiencia de los ingenieros y diseñadores experimentados a los más nóveles, convirtiéndose en unas de las vías para la gestión del conocimiento.

Lo que se pretende con un catálogo de patrones no es favorecer al diseñador experto (que quizás no necesite en absoluto los patrones), sino más bien ayudar al diseñador inexperto a adquirir con cierta rapidez las habilidades de aquél, como también comunicar al posible cliente, si es el caso, las decisiones de diseño en forma clara y autosuficiente.

2.4 Propiedades de los Patrones

Los patrones SI

- Proveen un vocabulario común
- Proveen un "atajo" para comunicar en forma sencilla principios complejos.
- Ayudan a documentar arquitecturas de software
- Capturas las partes esenciales de un diseño en forma compacta
- Muestran más de una solución
- Describen abstracciones de software

Los patrones NO

- Proveen una solución exacta
- Resuelven todos los problemas diseño
- Se aplican solamente al diseño orientado a objetos

2.5 Elementos de un Patrón

Nombre: permite describir, en una o dos palabras, un problema de diseño junto con sus soluciones y consecuencias. Al dar nombres a los patrones estamos incrementando nuestro vocabulario de diseño, lo cual nos permite diseñar y comunicarnos con un mayor nivel de abstracción (en lugar de hablar de clases individuales nos referimos a colaboraciones entre clases)

Problema: describe cuando aplicar el patrón. Explica el problema y su contexto. A veces incluye condiciones que deben darse para que tenga sentido la aplicación del patrón

Solución: describe los elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño o implementación en concreto, sino que es más bien una plantilla que puede aplicarse en muchas situaciones diferentes.

Consecuencias: son los resultados, así como ventajas e inconvenientes de aplicar el patrón.

2.6 Clasificación de los Patrones

En [GoF,1995] se describen 23 patrones de diseño, cada uno de los cuales proporciona una solución para un problema común de diseño de software en la industria. Este libro también los agrupa en tres categorías:

- **Patrones de diseño de creación**: Describen técnicas para instanciar objetos (o grupos de objetos).
- **Patrones de diseño de estructura**. Permiten a los diseñadores organizar clases y objetos en estructuras más grandes.
- **Patrones de diseño de comportamiento**. Asignan responsabilidades a los objetos.

El libro de la Banda de los cuatro [GoF,1995] demostró que los patrones de diseño evolucionan con el tiempo, a través de años de experiencia en la industria. En su

artículo Seven Habits of Successful Pattern Writers [2], John Vlissides declara que "la actividad más importante en la escritura de patrones es la reflexión". Esto implica que, para crear patrones, los desarrolladores deben reflexionar sobre sus éxitos, errores y documentarlos. Los desarrolladores utilizan patrones de diseño para capturar y emplear la experiencia colectiva en la industria para evitar cometer los mis errores dos veces.

En la tabla 1 se muestra los 23 patrones de diseño de la Banda de los cuatro.

Patrones de diseño de creación	**Patrones de diseño de estructura**	**Patrones de diseño de comportamiento**
Singleton	Proxy	Memento
Método de fábrica	Adaptador	Estado
Fábrica abstracta	Puente	Cadena de responsabilidad
Prototipo	Compuesto	Comando
Constructor	Decorador	Observador
	Fachada	Estrategia
	Flyweight	Método de plantilla
		Iterador
		Intérprete
		Mediador
		Visitante

Tabla 1: Los 23 patrones de diseño de la GoF (Banda de los cuatro)

Según Frank Buschman los patrones se pueden clasificar en:

- **Arquitectura**: Expresan una estructura fundamental de organización de los sistemas SW. Proveen un conjunto de subsistemas predefinidos, especificando sus responsabilidades y relaciones.
- **Diseño**: Proporcionan un esquema para refinar subsistemas o componentes. Resuelven problemas específicos de diseño.
- **Idiomas**: Son específicos de un lenguaje de programación. Describen cómo implementar ciertos aspectos de un problema utilizando las características de un lenguaje de programación.

Teniendo en cuenta el nivel de detalle según se puede apreciar en la figura 1

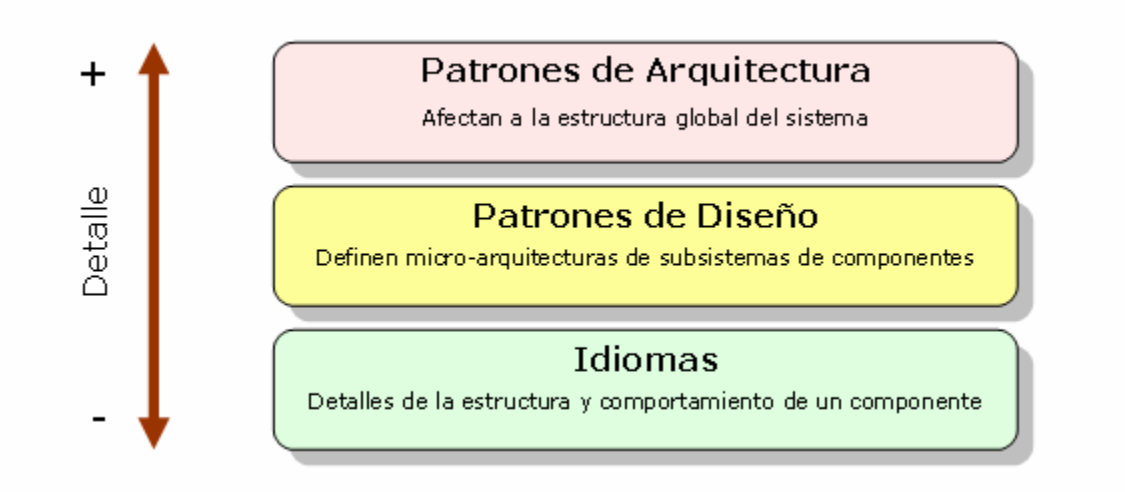


Figura 1: Clasificación de los patrones según su nivel de detalle, adaptado de [POSA,1996].

Desde la aparición del libro de la Banda de los cuatro han aparecido muchos patrones de diseño, entre los que se encuentran los siguientes:

- **Patrones de diseño de concurrencia**: Son muy útiles en el diseño de sistemas con subprocesamiento múltiple.
- **Patrones de arquitectura**: Especifican la manera en que los subsistemas interactúan entre sí.

En la tabla 2 se muestran dichos patrones.

Patrones de diseño de concurrencia	**Patrones de arquitectura**
Ejecución de un solo proceso Suspensión precavida Frustración Bloqueo de lectura/escritura Término en dos fases	Modelo-Vista-Controlador (MVC)

Tabla 2: Patrones de diseño de concurrencia y de arquitectura

Categorías de Patterns of Software Arquitecture (POSA)

En la tabla 3 a continuación se muestra la distribución de los patrones definidos en Patterns of Software Architecture [POSA1996]:

From Mud to Structure	Distributed Systems	Interactive Systems	Adaptable Systems
Layers Pipes and Filtres Blackboard	Broker	Model-View-Controller Presentation-Abstraction-Control	Microkernel Reflection

Tabla 3: Patrones de diseño de concurrencia y de arquitectura de POSA [POSA96

- **Layers**: Permite estructurar aplicaciones que se pueden descomponer en grupos de subtareas, donde cada grupo está en un determinado nivel de abstracción.
- **Pipes & Filters**: Provee una estructura para sistemas que procesan un flujo de datos. Cada etapa del proceso es encapsulada como un filtro. Los datos se pasan entre filtros adyacentes mediante Pipes. Recombinando filtros obtenemos familias de sistemas relacionados.
- **Blackboard**: Útil para sistemas en que no se conoce una solución o estrategia determinista. Varios subsistemas especializados ensamblan su conocimiento para construir una posible solución parcial.
- **Broker**: Permite estructurar sistemas distribuidos desacoplados que interaccionan mediante invocación de servicios remotos. Un componente Broker es responsable de coordinar la comunicación, así como de transmitir resultados y excepciones.
- **Model-View-Controller**: Divide una aplicación interactiva en 3 componentes. El Model contiene la información y funcionalidad principal. Views muestran información al usuario. Controllers gestionan la entrada de usuario. Un mecanismo de propagación de cambios asegura la consistencia entre el modelo y la interfaz de usuario.
- **Presentation-Abstraction-Control**: Estructura una aplicación interactiva como una jerarquía de agentes que cooperan. Cada agente es responsable de un determinado aspecto de la funcionalidad y consta de tres componentes: Presentación, Abstracción y Control, que separan la interacción con el usuario de la funcionalidad central y la Comunicación con otros agentes.

- **Microkernel:** Separar un mínimo núcleo funcional de funcionalidad extendida y partes específicas del cliente. El Microkernel también sirve como punto donde engarzar estas piezas y coordinar su colaboración.
- **Reflection:** Proporciona un mecanismo para cambiar la estructura y el comportamiento del sistema dinámicamente. La aplicación se divide en dos partes: una meta-nivel y un nivel-base. La meta-nivel hace al software autoconsciente.

Categorías de PEAA

En el libro "Patterns of Enterprise Application Architecture" [Fowler,2002] se definen las siguientes categorías de patrones de arquitectura:

• **Layering:** patrones para dividir un sistema en capas.
• **Organización de la lógica del dominio**: formas de organizar los objetos del dominio.
• **Mapping to Relational Databases**: se relaciona con la comunicación entre la lógica del dominio y los repositorios de datos. Incluye el mapeo entre modelos de objetos y bases de datos relacionales. En la actualidad, se consume mucho tiempo de desarrollo en la realización de estas tareas debido a las diferencias de impedancia entre SQL y los lenguajes orientados a objetos como Java, C#, C++, etc.
• **Presentación Web**: la presentación Web es uno de los desafíos que han tenido que sortear en los últimos años las aplicaciones empresariales. Los clientes delgados Web proveen muchas ventajas, siendo una de las principales la facilidad de distribución (no es necesario instalar software en los equipos cliente). Esta categoría incluye una serie de patrones para gestionar la presentación Web.
• **Concurrencia**: manejo de la concurrencia. Las aplicaciones actuales basadas en tecnologías Web tienen grandes necesidades de gestión de la concurrencia.
• **Estado de Sesión**: patrones para el manejo de la sesión de servidores Web.
• **Estrategias de Distribución**: distribución de objetos en múltiples emplazamientos, basado en conocimientos empíricos en clientes.

En la tabla 4 se muestra la distribución de los patrones definidos en Patterns of Enterprise Application Architecture (PEAA) [Fowler,2002] en las categorías mencionadas arriba:

Domain Logic Pattern	**Mapping to Relational Databases**	**Web Presentation Patterns**	**Distribution Patterns**	**Offline Concurrency Patterns**	**Session State Pattern**	**Base Patterns**
Transaction Script	Data Source Architectural Patterns	Model View Controller	Remote Façade	Optimistic Offline Lock	Client Session State	Gateway
Domain Model	Table Data Gateway	Page Controller	Data Transfer Object	Pesimistic Offline Lock	Server Session State	Mapper
Table Module	Row Data Gateway	Front Controller		Coarse-Grained Lock	Database Session State	Layer Supertype
Service Layer	Active Record	Template View		Implicit Lock		Separated Interface
	Data Mapper	Transform View				Registry
	Object Relational Behavioral Patterns	Two Step View				Value Object
	Unit of Work	Application Controller				Money
	Identity Map					Special Case
	Lazy Load					Plugin
	Object Relational Structural Patterns					Service Stub
	Identity Field					Record Set
	Foreign Key Mapping					
	Association Table					
	Mapping Dependent Mapping					
	Embedded Value					
	Serialized LOB					
	Single Table Inheritance					
	Class TableInheritance					
	Table Inheritance Concrete Inheritance					
	InheritanceMappers					
	Object-Relational Metadata Mapping Patterns					
	Metadata Mapping					
	Query Object					
	Repository					

Tabla 4: Clasificación de patrones de arquitectura de aplicaciones empresariales de [Fowler,2002].

2.7 Patrones de Arquitectura

- Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software.
- Proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos.
- Representan el nivel más alto en el sistema patrones propuesto en "Pattern Oriented Software Architecture, Volume1" [POSA,1996].
- Ayudan a especificar la estructura fundamental de una aplicación
- Cada actividad de desarrollo es gobernada por esta estructura, por ejemplo, el diseño detallado de los subsistemas, la comunicación y colaboración entre diferentes partes de los sistemas, etc.
- Cada patrón de arquitectura ayuda a conseguir una propiedad específica en el sistema global, como por ejemplo la adaptabilidad de la interface de usuario. Los patrones que dan soporte a características similares se agrupan en una misma categoría.

2.8 Papel de los patrones en el ciclo de vida del desarrollo

En el desarrollo software se utilizan fases para el análisis, el diseño, la implementación, las pruebas de la aplicación en desarrollo y en el caso del desarrollo de aplicaciones interactivas, también se incluye una fase de evaluación de la utilidad y la usabilidad. Cada fase crea una imagen más detallada del sistema que la anterior. Muchos desarrolladores tienen como motivación reducir el esfuerzo y el tiempo de este ciclo de vida del desarrollo. Cuando estamos desarrollando una aplicación, nos encontramos con problemas que ya hemos visto antes y nos preguntamos: "¿Cómo voy a resolver este problema esta vez?". Para muchos problemas, se busca la solución a un problema similar en el pasado para intentar reaplicarla. Los patrones son esa herramienta para capturar y comunicar este entendimiento y experiencia, creando conocimiento de diseño persistente entre profesionales de una comunidad. Un diseño de software efectivo requiere considerar asuntos que pueden no ser visibles hasta más allá de su

implementación. Reutilizar los patrones de diseño ayuda a prevenir asuntos sutiles que pueden causar problemas mayores.

Aunque los patrones no son un método o proceso de desarrollo, puede complementarlos. Los patrones fueron inicialmente introducidos para ser reutilizados durante la fase de diseño [los patrones de diseño presentados en Gamma et al (1994)], pero muchos profesionales se dieron cuenta de su utilidad en otras fases del desarrollo. La amplia aceptación de los patrones de diseño ha permitido cubrir todo el ciclo de vida del desarrollo software, creando nuevos tipos de patrones encargados de registrar el conocimiento de cada una de las fases según muestra la figura 2, acelerando así cada una de ellas

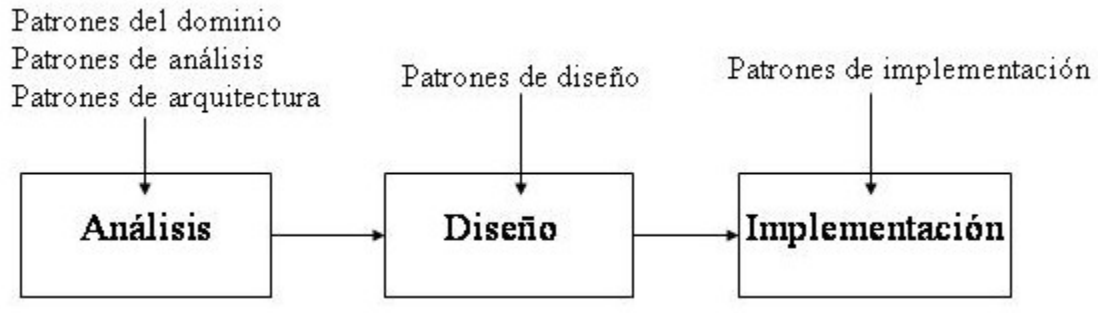


Figura 2: Tipos de patrones para cada una de las fases del desarrollo

A continuación, se describen los tipos de patrones que dan soporte a las diferentes fases del desarrollo:

- **Patrones del dominio**

Cuando se desarrolla un sistema que se encuadra dentro de cierto tipo o género, es muy útil consultar patrones específicos de ese dominio en el que estamos. Estos patrones nos servirán como referencia conceptual del dominio del problema, ayudándonos a identificar cuáles son los requisitos o características más relevantes de ese dominio.

Por ejemplo, en el dominio web para aplicaciones de subastas *on-line*, el lenguaje de patrones para la administración de un sitio de subastas (Re et al, 2001) permite identificar las principales características de estos tipos de sistemas: tratan con recursos (billetes de avión, libros, cds, etc.) que son intercambiados entre clientes, que pueden ser personas u organizaciones. Estos recursos pueden ser clasificados en categorías y

pueden existir diferentes tipos de subastas, que tienen diferentes reglas. En cada subasta hay negociaciones entre las partes implicadas, hay pujas que hay que administrar. Todas estas tareas tienen que ser controladas por la casa de subastas.

- **Patrones de Análisis**

Describen un conjunto de prácticas destinadas a elaborar modelos de los conceptos principales de la aplicación que se va a construir (Fowler, 1997). La intención principal de estos patrones es ayudar a las personas que realizan el trabajo de modelado, pues no siempre tienen experiencia al respecto y, en la mayoría de los casos, construyen sus modelos sin referencia alguna. Los patrones de análisis se diferencian de los de diseño en que se centran en aspectos sociales, de organización y económicos, los cuales son primordiales en el análisis de requisitos y la aceptación y usabilidad del sistema final.
Por ejemplo el patrón *Negociación* (Hamza y Fayad, 2004) proporciona un modelo que analiza los principales conceptos de la negociación, siendo conceptos que podemos encontrar en diferentes aplicaciones relacionadas con la compra o venta de propiedades, subastas y compras online, aplicaciones y dispositivos web. La solución propuesta está enfocada en el concepto de la negociación, intentando extraer sus principales componentes, de manera independiente del dominio de aplicación, para que los desarrolladores lo puedan utilizar según sus necesidades. Esos componentes son: *Acuerdo*, que representa el resultado de la negociación; *Parte*, que representa a los participantes de la negociación (persona, grupo u organización); *Medio*, que representa el medio a través el cual la negociación tiene lugar; *Contexto*, que representa las cuestiones que tienen que ser negociadas.

- **Patrones de Arquitectura**

Expresan un paradigma fundamental para estructurar u organizar un sistema software. Proporcionan un conjunto de subsistemas o módulos predefinidos, con reglas y guías para organizar las relaciones entre ellos. Por ejemplo, el patrón Broker se utiliza para sistemas software distribuidos con componentes desacoplados que interaccionan mediante invocaciones de servicios remotos. Un componente Broker es responsable de coordinar la comunicación. Este patrón podría complementar al patrón Negociación, visto en el punto anterior, proporcionando una arquitectura a la aplicación, ya que

posiblemente las partes de la negociación estén distribuidas, y el componente Broker puede actuar como medio de la negociación.

- **Patrones de Diseño**

Proporcionan un esquema para refinar los subsistemas o componentes de un sistema software y sus relaciones. Describen estructuras recurrentes de comunicar componentes que resuelven un problema de diseño en un contexto particular. Son patrones de un nivel de abstracción menor que los patrones de arquitectura, que están, por lo tanto, más próximos a lo que sería el código fuente final. Su uso no se refleja en la estructura global del sistema.

Por ejemplo, el patrón Proxy (Gamma et al., 1994) permite acceder a un recurso mediante un intermediario, con varios propósitos, tales como diferir la carga, controlar el acceso, hacer caché, etc. Los recursos a los que se puede querer acceder mediante un intermediario podrían ser archivos, conexiones de red, objetos muy grandes, etc. Este patrón se utiliza para diseñar los componentes que actúan como clientes y servidores en el patrón Broker. Relacionado con el patrón Negociador estaría el diseño de las partes, que actuarían como clientes, y el diseño del negociador y el medio, que actuaría como servidor, para así controlar el acceso al contexto de la negociación.

- **Patrones de Implementación**

Patrones de bajo nivel, específicos de un lenguaje de programación determinado. Describe cómo implementar aspectos particulares de los componentes de un patrón de diseño usando las características y potencialidades de un lenguaje de programación concreto.

Por ejemplo, los patrones J2EE representan soluciones expertas para problemas recurrentes en aplicaciones web de negocio basadas en la tecnología java. El patrón Business Delegate (Sun Microsystems, 2002) reduce el acoplamiento entre los clientes de la capa de presentación y los servicios de negocio y serviría para implementar el patrón Proxy en aplicaciones J2EE. Así, si estamos utilizando el patrón Negociación en el dominio de la web para un sistema de subastas, las partes que entrarían en la negociación (clientes) tendrían un acceso común y previo al medio en el que se está

produciendo la negociación, aislándolo en caso de que éste fallase y tuviese que ser reemplazado el servidor que gestiona la subasta.

2.9 LOS ANTIPATRONES

Los antipatrones son soluciones negativas que presentan más problemas que los que solucionan. Son una extensión natural a los patrones de diseño. Comprender los antipatrones provee el conocimiento para intentar evitarlos o recuperarse de ellos [BMMM,1998] y [Welicki,2006].

El estudio de los antipatrones permite conocer los errores más comunes relacionados con la industria del software. La obra más popular en este campo es "Antipatterns: Refactoring Software, Architectures and Projects in Crisis", publicada en 1998 [BMMM,1998].
En la figura 3 muestra las diferienias entre Patrón y Antipatrón

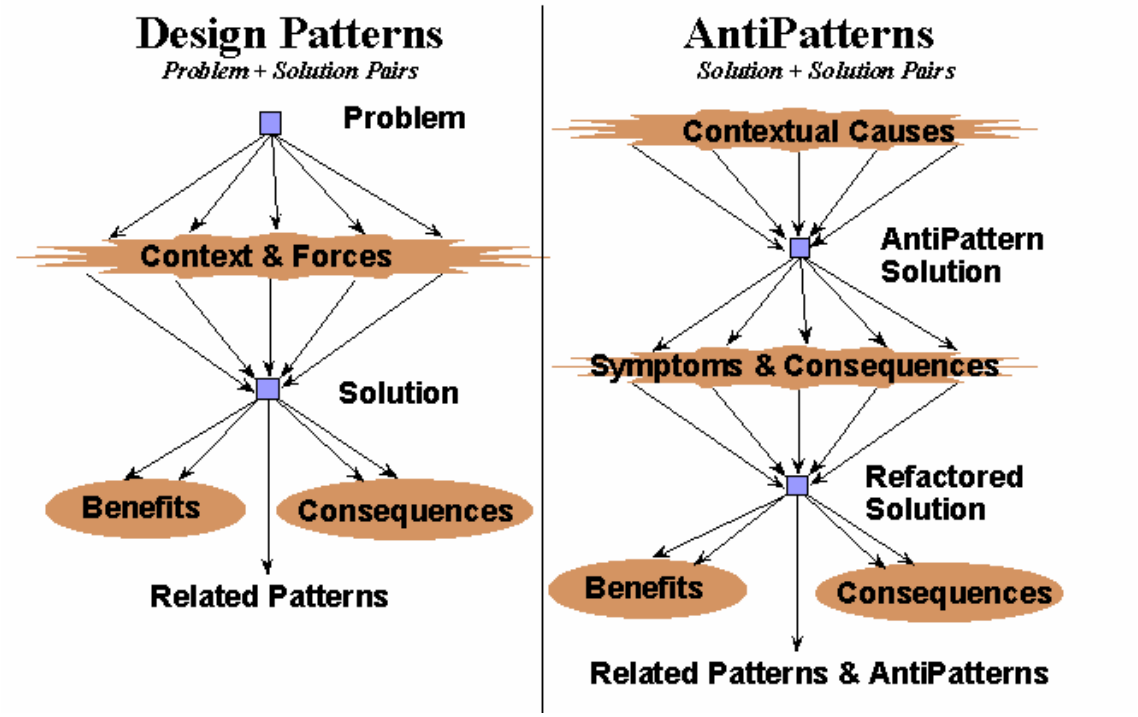


Figura 3: Patrones y antipatrones [McCormick,1998]

Los antipatrones se documentan con cierto cinismo, lo cual los hace bastante graciosos y fáciles de recordar. Los nombres siempre aluden al problema que tratan con humor.

Un buen antipatrón explica por qué la solución original parece atractiva, por qué se vuelve negativa y como recuperarse de los problemas que ésta genera. Según Jim Coplien, un antipatrón es "algo que se ve como una buena idea, pero que falla en malamente cuando se implementa" [Hillside].

En los antipatrones se parte desde una solución (que es la que genera el problema), mientras que en los patrones se parte de un problema a resolver.

2.10 El Patrón MVC

El patrón MODEL-VIEW-CONTROLLER (en adelante MVC) fue introducido inicialmente en la comunidad de desarrolladores de Smalltalk-80 [Smalltalk] por Trygve Reenskaug [Trygve]. MVC divide una aplicación interactiva en tres áreas: procesamiento, salida y entrada [POSA,996].

El corazón del MVC es que lo se llama "Separated Presentation". La idea detrás de Separated Presentation (Presentaciones Separadas) es realizar una clara división clara entre los objetos del modelo de dominio de la percepción real, y los objetos de presentación que son los elementos GUI que vemos en la pantalla [MartinFowler].

Este patrón utiliza las siguientes abstracciones:

• **Modelo (Model)**: Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada.

• **Vista (View)**: Muestra la información al usuario. Obtiene los datos del modelo. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.

• **Controlador (Controller)**: Reciben las entradas, usualmente como eventos que codifican los movimientos o pulsación de botones del ratón, pulsaciones de teclas, etc. Los eventos son traducidos a solicitudes de servicio21 para el modelo o la vista. El usuario interactúa con el sistema a través de los controladores.

La figura 4 nos permite visualizar de manera gráfica este patrón.

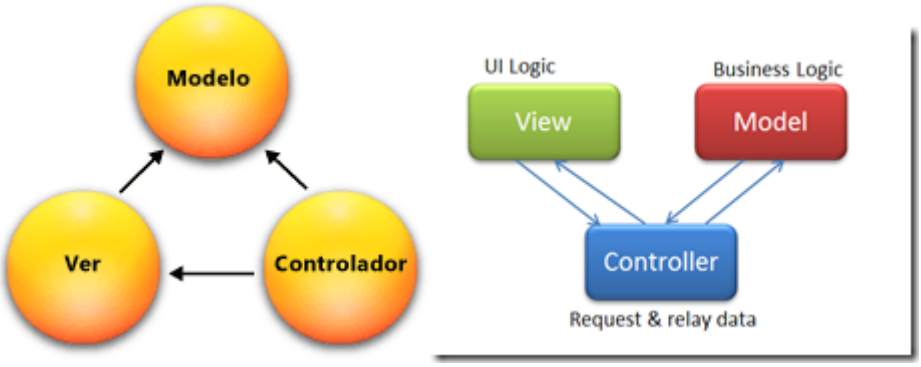


Figura 4: El patrón MVC [MSDN]

La Vistas y los Controladores conforman la interfaz de usuario. Un mecanismo de propagación de cambios asegura la consistencia entre la interfaz y el modelo.

La separación del modelo de los componentes vista y del controlador permite tener múltiples vistas del mismo modelo. Si el usuario cambie el modelo a través del controlador de una vista todas las otras vistas dependientes deben reflejar los cambios. Por lo tanto, el modelo notifica a todas las vistas siempre que sus datos cambios. Las vistas, en cambio, recuperan los nuevos datos del modelo y actualizan la información que muestran al usuario. [POSA,1996]

En la figura 5 a continuación se muestra la estructura del patrón MVC.

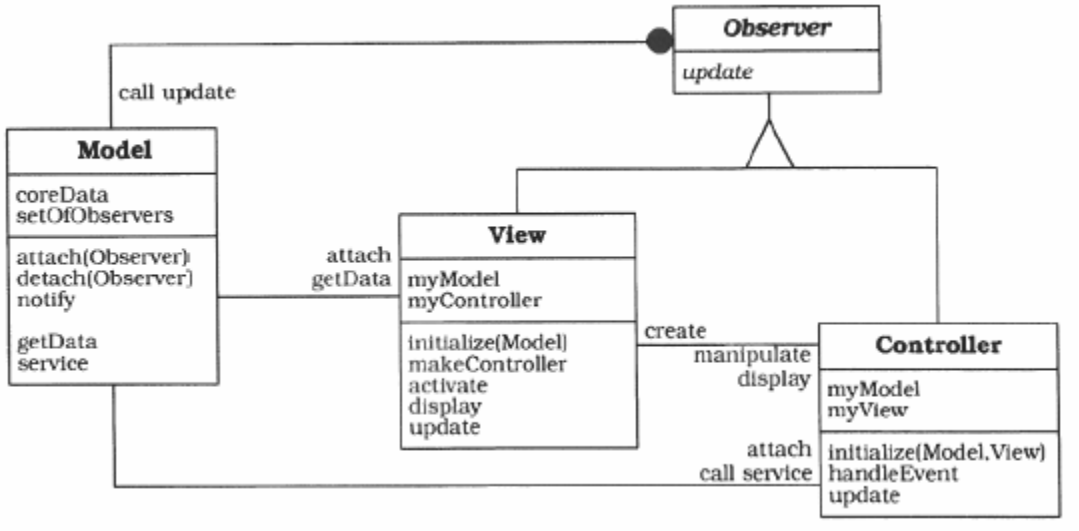


Figura 5: Diagrama de clases de MVC [POSA,1996]

Este patrón es muy popular y ha sido portado a una gran cantidad de entornos y marcos de trabajo como Struts [Struts], Swing [SJFC], WinForms [MSWF], ASP.NET

[MSAN], etc. Las herramientas de programación visual como Visual Basic [MSVB], Delphi [BID], JBuilder [BJB], etc. siguen también este esquema.

El MVC es un patrón ampliamente utilizado en múltiples plataformas y lenguajes. Algunos de sus principales beneficios son:

- Menor acoplamiento
  - Desacopla las vistas de los modelos
  - Desacopla los modelos de la forma en que se muestran e ingresan los datos

- Mayor cohesión
  - Cada elemento del patrón está altamente especializado en su tarea
  - (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio)

- Las vistas proveen mayor flexibilidad y agilidad.
  - Se pueden crear múltiples vistas de un modelo.
  - Se pueden crear, añadir, modificar y eliminar nuevas vistas
  - dinámicamente.
  - Las vistas pueden anidarse.
  - Se puede cambiar el modo en que una vista responde al usuario sin
  - cambiar su representación visual.
  - Se pueden sincronizar las vistas.
  - Las vistas pueden concentrarse en diferentes aspectos del modelo.

- Mayor facilidad para el desarrollo de clientes ricos en múltiples dispositivos y canales
  - Una vista para cada dispositivo, que puede varias según sus
  - capacidades
  - Una vista para la Web y otra para aplicaciones de escritorio.

- Más claridad de diseño.
  - Facilita el mantenimiento.

- Mayor escalabilidad.

2.11 LOS PATRONES DE DISEÑO EN EL MVC

El patrón MVC contiene o puede contener los siguientes patrones de diseño:

- **Observer** para el mecanismo de publicación y suscripción que permite la notificación de los cambios en el modelo a las vistas.
- **Composite** para la creación de vistas compuestas. Utilizando este patrón podemos crear una jerarquía de vistas y tratar a cada vista compuesta igual que una a una vista normal.
- **Strategy** en la relación entre las vistas y los controladores. Utilizando este patrón podemos cambiar dinámicamente (o en tiempo de compilación) los algoritmos del controlador mediante los cuales responde a su entorno.
- **Factory method** parar especificar la clase controlador predeterminada de una vista.
- **Decorator** para añadir capacidades adicionales a una vista (como por ejemplo scroll).
- **Proxy** para distribuir la arquitectura (Modelo y Vista-Controlador) en diferentes emplazamientos.

2.12 ASP.NET MVC 3

La primera versión ASP.NET MVC 1 apareció a mediados de 2009.

Con el lanzamiento de Visual Studio 2010, hace su aparación el MVC 2

MVC 3 es el último paradigma incorporado a la plataforma de desarrollo Web de Microsoft. Frente al modelo tradicional de ASP.NET Webforms, MVC 3 ofrece una forma totalmente diferente de programar para la Web, y es el modelo de futuro por el que apuesta Microsoft.

ASP.NET MVC 3 es una plataforma gratuita, Open Source, y está incluida de serie en .NET 4.0, pudiendo descargarse para su uso también con versiones anteriores (.NET

3.5). MVC 3, aparecida en Enero de 2011, añade nuevas e interesantes características a la colección de herramientas del programador Web. En el capítulo 3 detallaré las diferentes formas de desarrollar con esta tecnología.
El marco de ASP.NET MVC 3 proporciona una alternativa al modelo de formularios Web Forms de ASP.NET para crear aplicaciones web. El marco de ASP.NET MVC es un marco de presentación de poca complejidad y fácil de comprobar que (como las aplicaciones basadas en formularios Web Forms) se integra con las características de ASP.NET existentes, como páginas maestras y la autenticación basada en pertenencia. El marco de MVC se define en el ensamblado System.Web.Mvc. [MSDN]

ASP.NET MVC 3 es un modelo de diseño estándar con el que están familiarizados muchos desarrolladores. Algunos tipos de aplicaciones web salen beneficiadas con el marco de MVC. Otras seguirán utilizando el modelo de la aplicación ASP.NET tradicional que está basado en formularios Web Forms y postbacks. Otros tipos de aplicaciones web combinarán las dos estrategias; una no excluye a la otra.

La figura 6 nos muestra las Arquitecturas de Tecnologías de N Layer de Microsoft

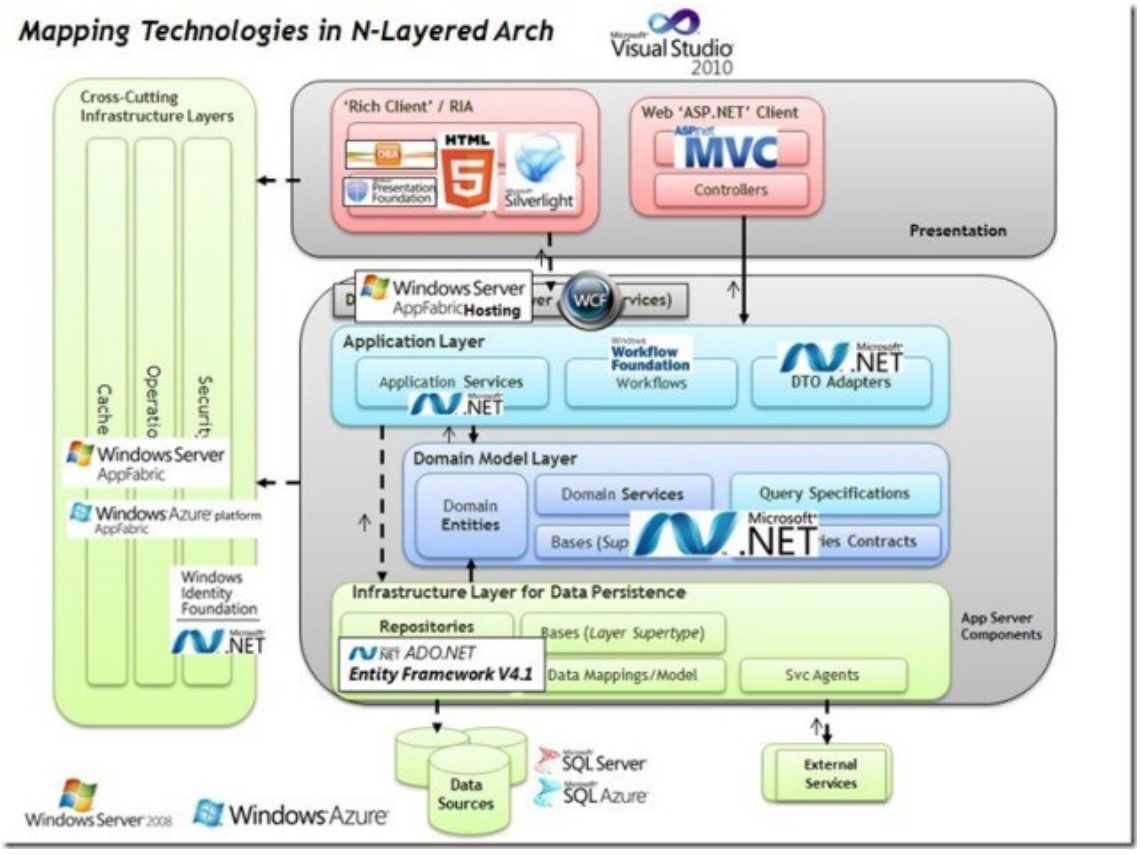


Figura 6: Arquitecturas de Tecnologías N-Layer de Microsoft [DeLaTorre,2010]

.

2.13 Características del marco de ASP.NET MVC

ASP.NET MVC ofrece las características siguientes:

- Separación de las tareas de la aplicación (lógica de entrada, lógica comercial y lógica de la interfaz de usuario), facilidad para pruebas y desarrollo basado en pruebas (TDD). Todos los contratos principales del marco de MVC están basados en interfaz y se pueden probar utilizando objetos ficticios, esto es, objetos ficticios que imitan el comportamiento de objetos reales en la aplicación. Puede hacer una prueba unitaria de la aplicación sin tener que ejecutar los controladores en un proceso de ASP.NET, lo cual hace que las pruebas unitarias sean rápidas y flexibles. Puede utilizar cualquier marco de pruebas unitarias que sea compatible con .NET Framework.
- Un marco extensible y conectable. Los componentes del marco de ASP.NET MVC están diseñados para que se puedan reemplazar o personalizar con facilidad.

Puede conectar su propio motor de vista, directiva de enrutamiento de URL, serialización de parámetros de método y acción y otros componentes. El marco de ASP.NET MVC también admite el uso de los modelos de contenedor Inyección de dependencia (DI) e Inversión de control (IOC). DI permite insertar objetos en una clase, en lugar de depender de que la clase cree el propio objeto. IOC especifica que si un objeto requiere otro objeto, el primer objeto debe obtener el segundo objeto de un origen externo como un archivo de configuración. Esto facilita las pruebas.

- Amplia compatibilidad para el enrutamiento de ASP.NET, un eficaz componente de asignación de direcciones URL que le permite compilar aplicaciones que tienen direcciones URL comprensibles y que admiten búsquedas. Las direcciones URL no tienen que incluir las extensiones de los nombres de archivo y están diseñadas para admitir patrones de nombres de direcciones URL que funcionan bien para la optimización del motor de búsqueda (SEO) y el direccionamiento de transferencia de estado representacional (REST, Representational State Transfer)..
- Compatibilidad para utilizar el marcado en archivos de marcado de páginas de ASP.NET existentes (archivos .aspx), de controles de usuario (archivos .ascx) y de páginas maestras (archivos .master) como plantillas de vista. Puede utilizar las características de ASP.NET existentes con el marco de ASP.NET MVC, como páginas maestras anidadas, expresiones en línea (<%= %>), controles de servidor declarativos, plantillas, enlace de datos, localización, etc.
- Compatibilidad con las características de ASP.NET existentes. ASP.NET MVC permite utilizar características como autenticación de formularios y autenticación de Windows, autorización para URL, pertenencia y roles, almacenamiento en caché de resultados y datos, administración de estados de sesión y perfil, seguimiento de estado, el sistema de configuración y la arquitectura de proveedor.

2.14 Ventajas de una aplicación web basada en ASP.NET MVC

ASP.NET MVC ofrece las ventajas siguientes:

- Facilita la administración de la complejidad, al dividir una aplicación en el modelo, la vista y el controlador.

- No utiliza el estado de vista ni formularios basados en servidor. Esto hace el marco de MVC sea ideal para los desarrolladores que deseen un control completo sobre el comportamiento de una aplicación.
- Utiliza un modelo Controlador frontal que procesa las solicitudes de la aplicación web a través de un controlador único. Esto permite diseñar una aplicación que admite una infraestructura de enrutamiento avanzada. Para obtener más información.
- Proporciona una mayor compatibilidad con el desarrollo basado en pruebas (TDD).
- Funciona bien para las aplicaciones web en las que trabajan equipos grandes de desarrolladores y para los diseñadores web que necesitan un alto grado de control sobre el comportamiento de la aplicación.
- Facilidad para crear prototipos rápidos.
- Reutilización de los componentes
- Simplicidad en el desarrollo y en mantenimiento de los sistemas.
- No existe el postback.
- No hay eventos.
- No existe el viewstate.
- No hay controles de servidor.

2.15 Estructura de una aplicación MVC

ASP.NET MVC incluye una plantilla de proyecto de Visual Studio que ayuda a crear aplicaciones web estructuradas para admitir el modelo de MVC. Esta plantilla crea una nueva aplicación web de MVC que se configura para tener las carpetas, plantillas de elementos y entradas de archivo de configuración necesarias.

Al crear una nueva aplicación web de MVC, Visual Studio ofrece la opción de crear dos proyectos al mismo tiempo. El primer proyecto es un proyecto web donde se implementa su aplicación. El segundo proyecto es un proyecto de prueba unitaria donde puede escribir las pruebas unitarias para los componentes de MVC del primer proyecto.

Nota: Microsoft Visual Studio Standard Edition y Microsoft Visual Web Developer Express Edition no permiten crear los proyectos de prueba unitaria. Por consiguiente, no ofrecen la opción de crear un proyecto de prueba al crear una aplicación de MVC.

Puede usar cualquier marco de pruebas unitarias que sea compatible con .NET Framework para probar las aplicaciones ASP.NET de MVC. Visual Studio Professional Edition incluye compatibilidad de pruebas de proyectos para MSTest.

Cuando se crea un proyecto de MVC de aplicación web ASP.NET, los componentes de MVC se separan en función de las carpetas de proyecto que se muestran en la siguiente figura 7

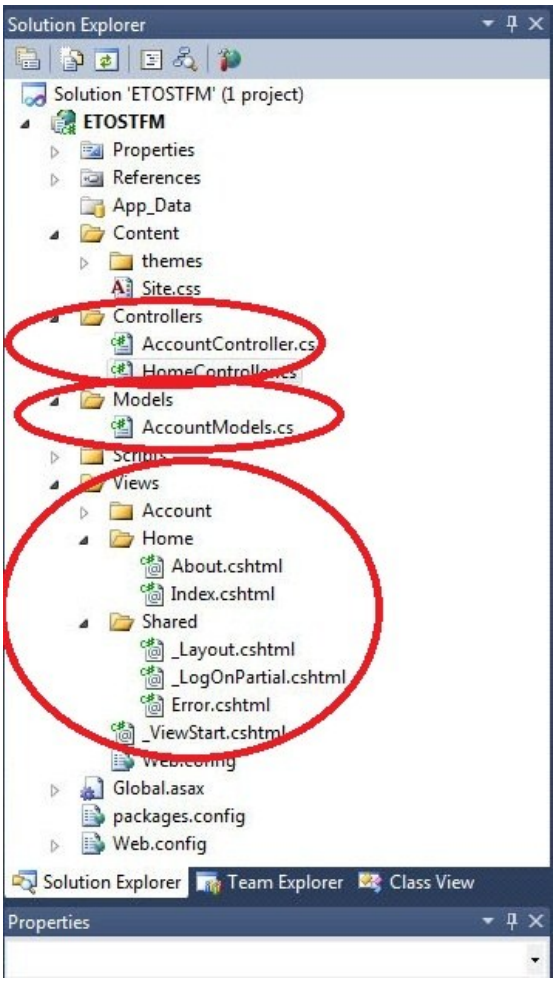


Figura 7: Estructura creada por defecto de un proyecto MVC

Podemos observar que la plantilla del proyecto ya nos genera dos Controllers (Home y Account), también nos genera la página Maestra (_Layout.cshtml), la página de Error

(Error.cshtml), dos Vistas (Home.cshtml e Index.cshtml). Esto es la infraestructura básica que me genera la plantilla del proyecto.

De forma predeterminada, los proyectos de MVC incluyen las carpetas siguientes:

- **App_Data**, que es el almacén físico de los datos. Esta carpeta tiene el mismo rol que en los sitios web ASP.NET que usan páginas de formularios Web Forms.
- **Content**, que es la ubicación recomendada para agregar los archivos de contenido, como los archivos de hoja de estilos en cascada, imágenes, etc. En general, la carpeta Content es para archivos estáticos.
- **Controllers**, que es la ubicación recomendada para los controladores. El marco de MVC requiere que los nombres de todos los controladores terminen con "Controller", como HomeController, LoginController o FichaController.
- **Models**, disponible para las clases que representan el modelo de aplicaciones de su aplicación web de MVC. Normalmente, esta carpeta incluye código que define los objetos y la lógica para la interacción con el almacén de datos. Normalmente, los objetos del modelo real estarán en bibliotecas de clases independientes. Sin embargo, al crear una nueva aplicación, podría colocar las clases aquí y, a continuación, moverlas a bibliotecas de clases independientes en un momento posterior del ciclo de desarrollo.
- **Scripts**, que es la ubicación recomendada para los archivos de script que respaldan la aplicación. De forma predeterminada, esta carpeta contiene archivos de base AJAX de ASP.NET y la biblioteca de jQuery.
- **Views**, que es la ubicación recomendada para las vistas. Las vistas usan archivos ViewPage (.aspx), ViewUserControl (.ascx) y ViewMasterPage (.master), además de otros archivos relacionados con la representación de vistas. La carpeta Views contiene una carpeta para cada controlador; el nombre de la carpeta es el prefijo del nombre del controlador. Por ejemplo, si tiene un controlador denominado HomeController, la carpeta Views contiene una carpeta denominada Home. De forma predeterminada, cuando el marco de MVC de ASP.NET carga una vista, busca un archivo ViewPage (.aspx) con el nombre de la vista solicitada en la carpeta Views*nombreControlador*.

- **Shared** en la carpeta Views, que no corresponde a ningún controlador, se utiliza para las vistas que se comparten entre varios controladores. Por ejemplo, puede colocar la página maestra de la aplicación web en la carpeta Shared.

Además de las carpetas enumeradas anteriormente, una aplicación web de MVC usa el código del archivo **Global.asax** para establecer los valores predeterminados de enrutamiento global de direcciones URL, y usa el archivo Web.config para configurar la aplicación.

2.16 Enrutamiento de direcciones URL para ASP.NET MVC

ASP.NET MVC usa el motor de enrutamiento de ASP.NET, que proporciona la flexibilidad de asignar direcciones URL a las clases de controlador. Se puede definir las reglas de enrutamiento que el marco de MVC de ASP.NET usará para evaluar direcciones URL entrantes y seleccionar el controlador adecuado. También puede hacer que el motor del enrutamiento analice automáticamente las variables definidas en la dirección URL, y hacer que el marco de MVC de ASP.NET pase los valores al controlador como argumentos de parámetro.

Las rutas se inicializan en el método Application_Start del archivo Global.asax. En la figura 8 se muestra un archivo Global.asax típico que incluye la lógica del enrutamiento predeterminado.

```
public class MvcApplication : System.Web.HttpApplication
{
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
filters.Add(new HandleErrorAttribute());
}
public static void RegisterRoutes(RouteCollection routes)
{
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

routes.MapRoute(
"Default", // Route name
"{controller}/{action}/{id}", // URL with parameters
        new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter defaults
);
```

```
}
protected void Application_Start()
{
AreaRegistration.RegisterAllAreas();
RegisterGlobalFilters(GlobalFilters.Filters);
RegisterRoutes(RouteTable.Routes);
}
}
```

Figura 8: Archivo Global.asax de enrutamiento por defecto

El enrutamiento ASP.NET permite usar direcciones URL que no es necesario asignar a archivos específicos de un sitio web. Dado que la dirección URL no tiene que asignarse a un archivo, se pueden usar direcciones URL que describan la acción [Enrutamiento] del usuario y, por tanto, sean más fáciles de comprender.

El marco de MVC de ASP.NET y los datos dinámicos de ASP.NET extienden el enrutamiento para proporcionar características que se usan únicamente en las aplicaciones de MVC y aplicaciones de datos dinámicos. Para obtener más información sobre ASP.NET MVC.

En una aplicación ASP.NET que no utiliza el enrutamiento, una solicitud entrante de una dirección URL normalmente se asigna a un archivo físico que controla la solicitud, como un archivo .aspx. Por ejemplo, una solicitud de http://server/application/Products.aspx?id=4 se asigna a un archivo denominado Products.aspx que contiene código y marcado para representar una respuesta al explorador. La página web utiliza el valor de cadena de consulta id=4 para determinar el tipo de contenido que se va a mostrar.

En el enrutamiento de ASP.NET, se pueden definir modelos de dirección URL que se asignen a archivos de controlador de solicitudes pero que no necesariamente incluyan los nombres de esos archivos en la dirección URL. Además, se pueden incluir marcadores de posición en un modelo de dirección URL de modo que se puedan pasar datos variables al controlador de solicitudes sin necesidad de una cadena de consulta.

Por ejemplo, en la solicitud de http://server/application/Products/show/beverages, el analizador del enrutamiento puede pasar los valores Products, show y beverages a un

controlador de páginas. En este ejemplo, si se define la ruta mediante el modelo de dirección server/application/{controlador}/{action}/{category}, el controlador de páginas recibirá una colección de diccionarios donde el valor asociado a la clave área es Products, el valor de la clave action es show y el valor de la clave category es beverages. En una solicitud no administrada por el enrutamiento de direcciones URL, el fragmento /Products/show/beverages se interpretaría como la ruta de acceso a un archivo de la aplicación.

Una ruta es un modelo de dirección URL que está asignado a un controlador. El controlador puede ser un archivo físico, como un archivo .aspx en una aplicación de formularios Web Forms. También puede ser una clase que procesa la solicitud, como un controlador en una aplicación de MVC. Para definir una ruta, se crea una instancia de la clase Route especificando el modelo de dirección URL, el controlador y, de forma opcional, un nombre para la ruta.

Se agrega la ruta a la aplicación agregando el objeto Route a la propiedad estática Routes de la clase RouteTable. La propiedad Routes es un objeto RouteCollection que almacena todas las rutas de la aplicación.

En la tabla 5 siguiente se muestran modelos de ruta válidos y ejemplos de solicitudes URL que coinciden con los modelos.

Definición de la ruta	**Ejemplo de dirección URL coincidente**
{controller}/{action}/{id}	/Products/show/beverages
{table}/Details.aspx	/Products/Details.aspx
blog/{action}/{entry}	/blog/show/123
{reporttype}/{year}/{month}/{day}	/sales/2008/1/5
{locale}/{action}	/US/show
{language}-{country}/{action}	/en-US/show

Tabla 5: Modelos de rutas válidos y ejemplos de solicitudes URL

Los modelos de dirección URL para las rutas en las aplicaciones de MVC suelen incluir los marcadores de posición {controller} y {action}.

Cuando se recibe una solicitud, se enruta dicha solicitud al objeto UrlRoutingModule y, a continuación, al controlador HTTP MvcHandler. El controlador HTTP MvcHandler especifica el controlador que se va a invocar agregando el sufijo "Controller" al valor de controlador en la dirección URL para determinar el nombre de tipo del controlador que va a controlar la solicitud. El valor de acción en la dirección URL determina el método de acción que se va a invocar.

Por ejemplo, una dirección URL que incluye la ruta de acceso /Products está asignada a un controlador denominado ProductsController. El valor del parámetro action es el nombre del método de acción que se invoca. Una dirección URL que incluye la ruta de acceso /Products/show daría lugar a una llamada al método Show de la clase ProductsController. La figura 9 podemos apreciar las secuencias de una aplicación MVC.

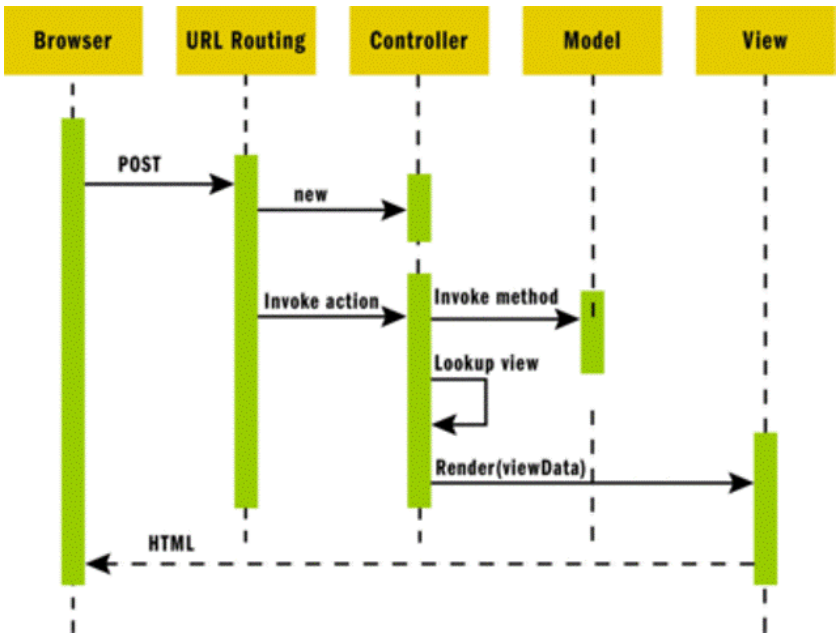


Figura 9: Diagrama de secuencias de una aplicación MVC [DinoEsposito]

2.17 Descripción de la ejecución de una aplicación MVC

Las solicitudes a una aplicación web basada en ASP.NET MVC [EjecucionMDSN] en primer lugar pasan a través del objeto UrlRoutingModule, que es un módulo HTTP. Este módulo analiza la solicitud y realiza la selección de la ruta. El objeto UrlRoutingModule selecciona el primer objeto de ruta que coincide con la

solicitud actual. (Un objeto de ruta es una clase que implementa RouteBase y normalmente es una instancia de la clase Route). Si ninguna ruta coincide, el objeto UrlRoutingModule no hace nada y permite que la solicitud recurra al procesamiento de solicitudes de ASP.NET o IIS normal.

A partir del objeto Route seleccionado, el objeto UrlRoutingModule devuelve un objeto que implementa la interfaz IRouteHandler y que está asociada al objeto Route. Normalmente, en una aplicación MVC, esta será una instancia de la clase MvcRouteHandler. Lainstancia MvcRouteHandler crea un objeto MvcHandler que implementa la interfaz IHttpHandler. A continuación, el objeto MvcHandler selecciona el controlador que administrará la solicitud finalmente.

Las clases UrlRoutingModule y MvcRouteHandler son los puntos de entrada principales del marco ASP.NET MVC. Realizan las siguientes acciones:

- Seleccionar el controlador adecuado en una aplicación web MVC.
- Obtener una instancia del controlador concreta.
- Llamar al método Execute del controlador.

En la tabla 6 se ofrece una lista de las fases de ejecución para un proyecto web de MVC.

Fase	Detalles
Recibir la primera solicitud para la aplicación	En el archivo Global.asax, los objetos Route se agregan al objeto RouteTable.
Realizar el enrutamiento	El módulo UrlRoutingModule utiliza el primer objeto Route coincidente de la colección RouteTable para crear el objeto RouteData, que a continuación utiliza para crear un objeto RequestContext.
Crear el controlador de solicitudes de MVC	El objeto MvcRouteHandler crea una instancia de la clase MvcHandler y la pasa instancia RequestContext al controlador de eventos.
Crear el controlador	El objeto MvcHandler utiliza la instancia RequestContext para identificar el objeto IControllerFactory (normalmente una instancia de la clase DefaultControllerFactory) para crear la instancia del controlador.
Ejecutar el controlador	La instancia MvcHandler llama al método Execute del controlador.
Invocar la acción	Para los controladores que heredan de la clase ControllerBase, el objeto ControllerActionInvoker que está asociado al controlador determina qué método de acción de la clase de controlador se llama y, a continuación, lo llama.
Ejecutar el resultado	El método de acción recibe los datos proporcionados por el usuario, prepara los datos

	de respuesta adecuados y, a continuación, ejecuta el resultado devolviendo un tipo de resultado. Los tipos de resultado integrados que se pueden ejecutar incluyen los siguientes: ViewResult (que representa una vista y es el tipo de resultado utilizado con mayor frecuencia), RedirectToRouteResult, RedirectResult, ContentResult, JsonResult, FileResulty EmptyResult.

Tabla 6: Fases de ejecución de un proyecto ASP.NET MVC

En la siguiente figura 10 se puede ver el Flujo de Solicitudes de MVC

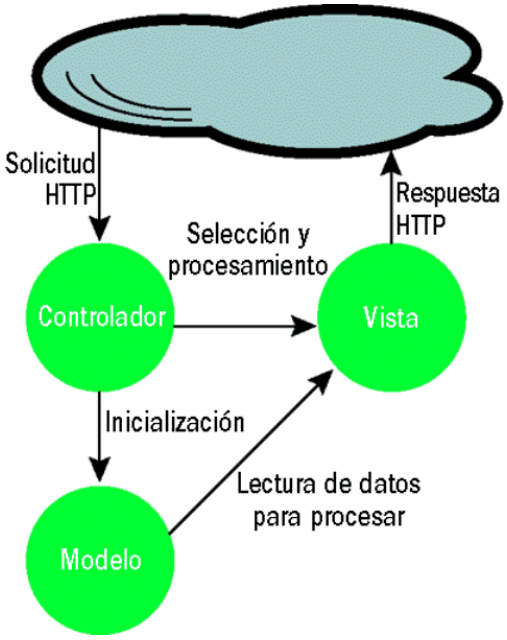


Figura 10: Flujo de solicitudes MVC [MsdnMagazine0308]

En la siguiente figura 11 se muestra la diferencia entre las aplicaciones Web Forms y MVC.

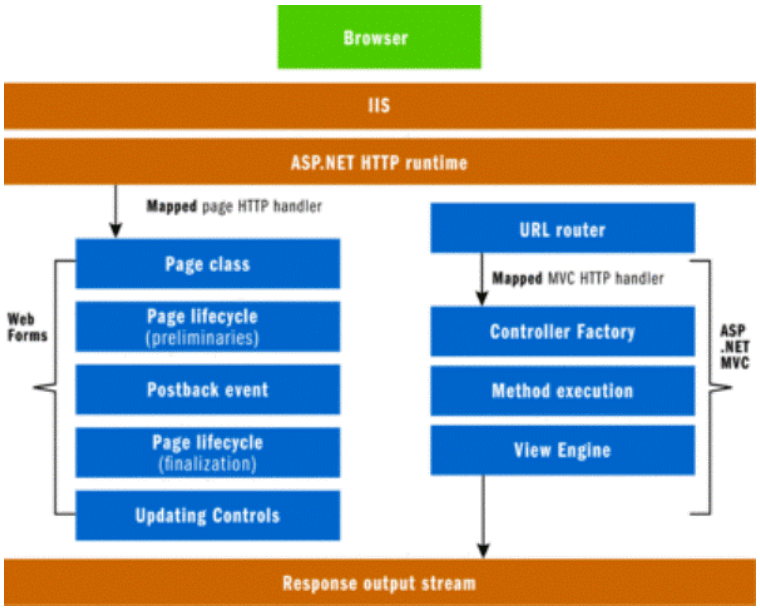


Figura 11: Ciclo de Vida de las aplicaciones Web Forms y MVC [DinoEsposito]

2.18 ASP.NET MVC 3 Tools

En junio de 2011 fue el lanzamiento de la actualización de ASP.NET MVC 3 Tools, actualización que incluye las siguientes mejores con respecto a ASP.NET MVC 3:

- El **lanzamiento final de EF 4.1** (que incluye EF Code First) está incluido por defecto en todos los proyectos nuevos. Antes se tenía que descargar esta librería de forma separada usando NuGet. Ahora esta referenciada por defecto.
- Soporte integrado para **data scaffloding** en el diálogob Add →Controller de Visual Studio. Esto nos permite construir una clase Controlador (incluyendo todo el código de acceso a datos necesario) y todas las vistas requeridas para crear una solución CRUD (Create, Ready, Update y Delete) sobre clases del modelo EF (usando tanto Database First, Model First o Code First). Esto facilita enormemente la creación de sitios Webs con acceso a datos.
- **Soporte de plantillas de proyectos HTML 5** para hacer más fácil el uso de etiquetas de semántica HTML 5 cuando se crea nuevos sitios Webs (<head>, <footer>, <section>, etc). También se incluye la librería **Modernizr 1.7 de JavaScript**. Modernizr es una librería OSS que hace muy fácil comprobar las características de las capacidades para HTML 5 de los navegadores, y también nos permite el uso de CSS estándar para elementos HTML 5.

- Se ha introducido la nueva plantilla de proyectos de Intranet para crear proyectos nuevos que usen autenticación de Windows para identificar a usuarios.
- Nuevas versiones del núcleo de jQuery, jQuery UI y jQuery Validation se han incluido en esta actualización.

2.19 Code First en ASP.NET Entity Framework 4.1

.NET Framework 4 viene con una versión muy mejorada de Entity Framework (EF) una biblioteca de acceso a los datos que vive en el espacio de nombres System.Data.Entity.

ASP.NET Entity Framework se puede utilizar en las siguientes bases de datos:

SQL Server, SQL CE, Oracle, MySQL, SQLLite y PostgreSQL.

Algunas de las grandes mejoras en EF4.1 incluyen [BlogScottGu]:

- **Soporte POCO:** Ahora puede definir las entidades sin necesidad de clases base o atributos de datos de persistencia.
- **Apoyo Lazy Loading (carga diferida):** Ahora puede cargar sub-objetos de un modelo bajo demanda en lugar de cargarlos en la delantera.
- **Entidades de N niveles de apoyo y auto-seguimiento:** Manejar escenarios donde el flujo de las entidades en todos los niveles o llama apátridas web.
- **Mejor soporte de SQL Generación y procedimientos almacenados (sprocs)** : EF4 ejecuta mejor SQL, e incluye una mejor integración con sprocs.
- **Apoyo a la pluralización automática** : EF4 incluye soporte automático de la pluralización de las tablas (Categorías-> por ejemplo, Categoría).
- **Capacidad de pruebas mejoradas** : el contexto EF4 de objetos ahora pueden ser más fácilmente falsificado el uso de interfaces.
- **Respaldar la mejora del operador LINQ** : EF4 ofrece ahora soporte completo para los operadores de LINQ.

Entity Framework 4.1 incluye dos características principales nuevas: la API de DbContext y Code First. La API de DbContext es una abstracción simplificada del tipo ObjectContext, además de otros tipos que se incluyeron en versiones anteriores de Entity Framework. La superficie de la API de DbContext está optimizada para las tareas

comunes y los patrones de código. La funcionalidad común se muestra en el nivel raíz y se presentan funcionalidades más avanzadas a medida que se profundiza en la API

En la siguiente figura 12 muestra el nuevo EF4.1 con Code First y DbContext

Figura 12: Entity Framework 4.1 con Code First y DbContext [MDSNDataEF4_1]

Visual Studio 2010 también incluye mucho más rica diseñador EF y soporte de herramientas. El diseñador de EF en VS 2010 es compatible con una "primera base de datos" estilo de desarrollo - donde se construye la capa del modelo en una superficie de diseño de una base de datos existente. También es compatible con un "primer modelo de" estilo de desarrollo - en la que definir en primer lugar la capa del modelo utilizando la superficie de diseño, y luego se puede utilizar para generar el esquema de base de datos de la misma.

ADO.NET Entity Framework permite a los desarrolladores crear aplicaciones de acceso a datos programando con un modelo de aplicaciones conceptuales en lugar de programar directamente con un esquema de almacenamiento relacional. El objetivo es reducir la cantidad de código y el mantenimiento necesarios para las aplicaciones orientadas a datos [MSDNEF]. Las aplicaciones de Entity Framework ofrecen las siguientes ventajas:

- Las aplicaciones pueden funcionar en términos de un modelo conceptual más centrado en la aplicación, que incluye tipos con herencia, miembros complejos y relaciones.

- Las aplicaciones están libres de dependencias de codificación rígida de un motor de datos o de un esquema de almacenamiento.

- Las asignaciones entre el modelo conceptual y el esquema específico de almacenamiento pueden cambiar sin tener que cambiar el código de la aplicación.

- Los desarrolladores pueden trabajar con un modelo de objeto de aplicación coherente que se puede asignar a diversos esquemas de almacenamiento, posiblemente implementados en sistemas de administración de base de datos diferentes.

- Se pueden asignar varios modelos conceptuales a un único esquema de almacenamiento.

- La compatibilidad con Language Integrated Query (LINQ) proporciona validación de la sintaxis en el momento de la compilación para consultas en un modelo conceptual.

ADO.NET Entity Framework admite aplicaciones y servicios centrados en datos, y proporciona una plataforma para la programación con datos que eleva el nivel de abstracción del nivel lógico relacional al nivel conceptual. Al permitir a los programadores trabajar con datos en un nivel de abstracción superior, Entity Framework admite código que es independiente de cualquier motor de almacenamiento de datos o esquema relacional determinados.

Entity Framework admite Entity Data Model (EDM) para definir datos en el nivel conceptual. Cuando se usa ADO.NET Entity Data Model Designer, la información sobre la asignación, el modelo conceptual y el de almacenamiento se almacena en un archivo .edmx. Entity Framework también permite a los desarrolladores programar directamente con los tipos de datos definidos en el nivel conceptual como objetos de Common Language Runtime (CLR). Entity Framework proporciona herramientas para generar un archivo .edmx y los objetos de CLR relacionados basándose en una base de datos existente. Esto reduce en gran medida el código de acceso a datos que se solía necesitar para crear aplicaciones y servicios de datos basados en objetos, y agiliza la creación de servicios y aplicaciones de datos orientadas a objetos a partir de una base de datos existente. Las herramientas también permiten compilar en primer lugar un modelo

conceptual y, a continuación, generar automáticamente los objetos de CLR relacionados y una base de datos auxiliar.

ADO.NET Entity Framework 4.1 se lanzó en abril de 2011 e incluye una gama de nuevas características creadas a partir de la funcionalidad de Entity Framework 4 existente que se lanzó en Microsoft .NET Framework 4 y Visual Studio 2010.

Code First es un nuevo patrón de desarrollo para Entity Framework que proporciona una alternativa a los patrones Database First y Model First existentes [RowanMiller]. Code First le permite definir su modelo mediante clases CLR. Después esas clases se pueden asignar a una base de datos existente o se pueden usar para generar un esquema de base de datos. Se puede realizar una configuración más detallada usando anotaciones de datos o mediante una API fluída. En la Figura 13 se aprecia la utilización de la autorización y autenticación de EFMembreshipProvide.

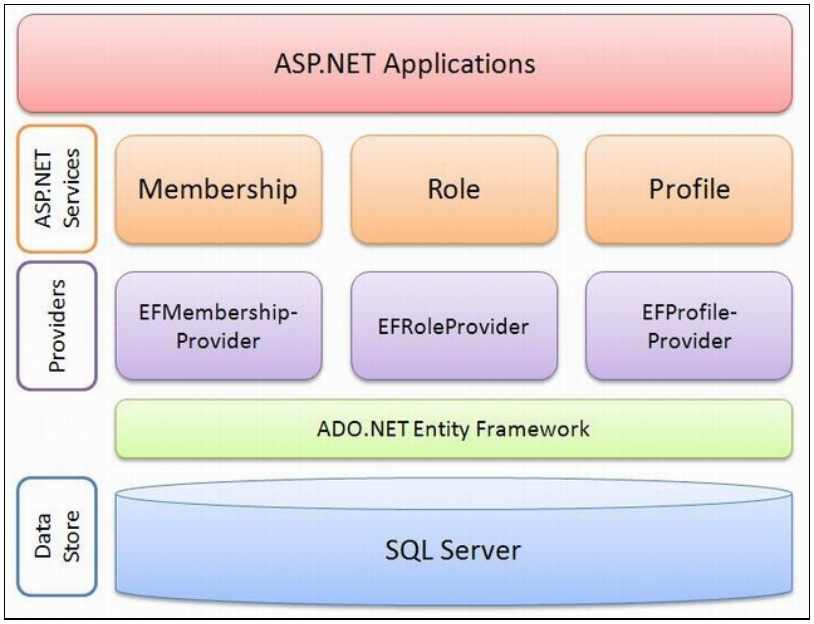


Figura 13 Entity Framework y EFMembershipProvider [CodeProjectEFProvider]

2.20 Utilizando Code First EF con una base de datos existente

La biblioteca de Code First de EF trabaja muy bien con bases de datos ya existentes, permitiendo que podamos usar un tipo de desarrollo centrado en código. Específicamente, nos deja usar clases simples y tradicionales (conocidas como clases

POCO) para el modelo de la aplicación, y crear correspondencias a la base de datos ya sea mediante convenciones predeterminadas, o configurando nuestras propias reglas para la creación del esquema de datos.

Este nuevo componente permite un eficiente flujo de desarrollo centrado en código. Nos permite:

- Trabajar con datos sin nunca tener que abrir un diseñador o lidiar con correspondencias en XML
- Definir objetos del modelo usando simples clases tradicionales sin tener que derivar de clases base específicas
- Usar la modalidad de convención en vez de configuración para el almacenaje de datos, evitándonos tener que configurarlo

En la siguiente figura 14 se muestra las tres maneras de trabajar con datos Entity Framework 4.1: Database First, Model First y Code First.

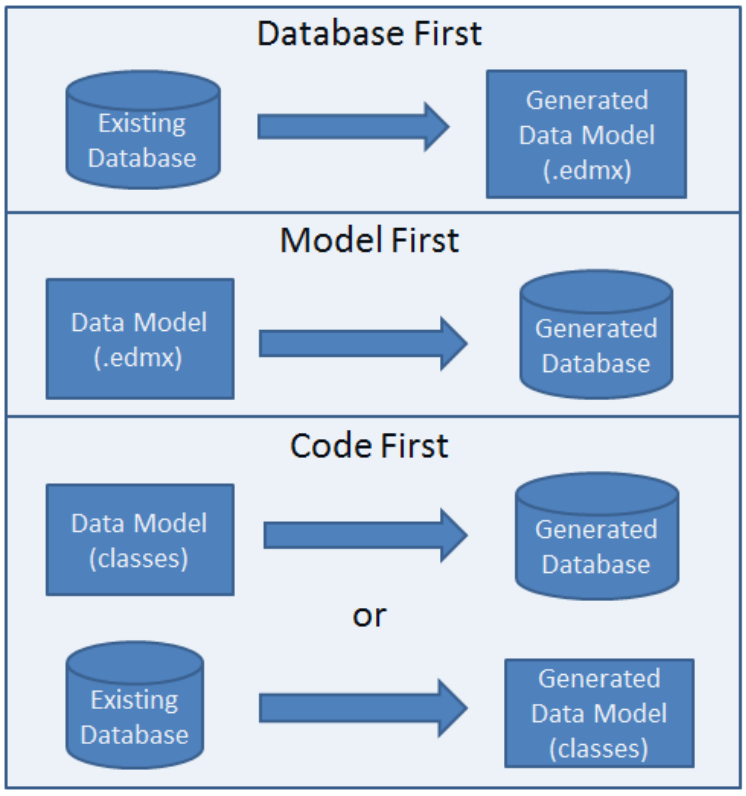


Figura 14: Entity Framework 4.1 para trabajar con datos [ContosoUniversityMVC]

2.20.1 Database First (Base de datos primero)

Si ya tenemos una base de datos, Entity Framework 4.1 **puede generar automáticamente un modelo de datos** que consta de clases y propiedades que corresponden a objetos de base de datos existentes, tales como tablas y columnas. La información sobre la estructura de su base de datos (esquema de almacenamiento), el modelo de datos (modelo conceptual), y la correspondencia entre ellos es almacenada en XML en un archivo **.edmx**. Visual Studio proporciona el diseñador de Entity Framework, que es una diseñador gráfico que se puede utilizar para visualizar y editar el archivo .edmx.

2.20.2 Model First (Modelo primero)

Si no tenemos una base de datos, podemos comenzar por la creación de un modelo con el diseñador de Entity Framework en Visual Studio. Cuando el modelo esté terminado, el diseñador puede generar DDL (Data Definition Language) para crear estados de la base de datos. Este enfoque también utiliza una archivo .edmx para almacenar el modelo y la información cartográfica.

2.20.3 Code First (Código primero)

Si tenemos una base de datos existente o no, podemos codificar nuestras propias clases y propiedades que corresponden a las tablas y columnas y el uso con Entity Framework 4.1 sin un archivo .edmx. Es por eso que a veces se ve este enfoque llamado Code Only, aunque el nombre oficial es **Code First**. El mapeo entre el esquema de almacenamiento y el modelo conceptual representado por su código está a cargo de la convención y por una API especial de cartografía. Si no tenemos una base de datos, Entity Framework 4.1 puede crear automáticamente la base de datos para usted, o quitar y volver a crearlo si el modelo cambia.

La API de acceso a datos que se ha desarrollado para Code First EF se basa en la clase DbContext. Esta API también se puede utilizar con los flujos de trabajo de desarrollo de Database First y Model First.

2.20.3.1 POCO (Plain Old CLR Object)

Entity Framework 4.1 permite utilizar clases de datos personalizadas junto con su modelo de datos sin realizar ninguna modificación en las clases de datos. Esto significa que podrá utilizar objetos CLR "antiguos" (POCO), tales como objetos de dominio existentes, con el modelo de datos. Estas clases de datos POCO (también conocidos como objetos que ignoran la persistencia), asignadas a entidades definidas en un modelo de datos, admiten la mayoría de los mismos comportamientos de consulta, inserción, actualización y eliminación (CRUD) que los tipos de entidad generados por las herramientas Entity Data Model [MSDNLibraryPOCO].

Para utilizar entidades POCO con un modelo de datos, el nombre del tipo de entidad debe ser igual que la clase de datos personalizada, y cada propiedad del tipo de entidad debe asignarse a una propiedad pública de la clase de datos personalizada. Los nombres de los tipos y de cada una de las propiedades asignadas deben ser equivalentes.

Por defecto, cuando se utiliza la base de datos de los enfoques Database First o el Model First, las clases de entidad en el modelo de datos heredan de la EntityObject de clase, que les proporciona la funcionalidad de Entity Framework. Esto significa que estas clases no son técnicamente "persistence ignorance" y por lo tanto no se ajusten plenamente a uno de los requisitos de Domain Driven Design. Todos los enfoques de desarrollo de Entity Framework también pueden trabajar con las clases POCO, lo cual significa básicamente que son ignorantes de persistencia, ya que no heredan de la EntityObject clase.

La biblioteca Code First EF nos permite usar objetos simples y llanos del CLR (o POCO) para representar las entidades en la base de datos. Por lo tanto, no necesitamos usar clases bases especiales ni implementar una interfaz específica o agregar atributos de ningún tipo. De esta manera nuestras clases se mantienen en ignorancia de su mecanismo de almacenaje.

Propiedades de asociación y cargado diferido

Code First facilita el uso de relaciones de claves primarias y externas en la base de datos y expone propiedades en las clases del modelo que nos ayudan a navegar las clases usando esas relaciones.

```
01 public class Product
02 {
03     public int ProductId { get; set; }
04     public string ProductName { get; set; }
05     public decimal? UnitPrice { get; set; }
06     public bool Discounted { get; set; }
07     public int CategoryID { get; set; }
08
09     public virtual Category Category { get; set; }
    }
10
```

```
1  public class Category
2  {
3     public int CategoryID { get; set; }
4     public string CategoryName { get; set; }
5     public string Description { get; set; }
6     public byte[] Picture { get; set; }
7
8     public virtual ICollection<Product> Products { get; set; }
9 }
```

```
1 public class Northwind : DbContext
2 {
3     public DbSet<Product> Products { get; set; }
4     public DbSet<Category> Categories { get; set; }
5 }
```

En el código arriba, la clase Product expone la propiedad Category, y la clase Category expone Products. Estas propiedades permiten el uso de relaciones de clave primaria y externa entre las dos tablas para obtener instancias del modelo. Las propiedades de asociación marcadas como virtuales son, por defecto, cargadas de modo diferido. Es decir, si uno obtiene una entidad Product, la información sobre la categoría no es cargada hasta que sea leída la propiedad Category (a menos que uno cargue la propiedad explícitamente en la consulta LINQ).

En el ejemplo arriba, las clases *Product* y *Category* son POCO que usaremos para representar las tablas *Products* y *Categories* en la base de datos Northwind. Las propiedades de las clases corresponden a las columnas de las respectivas tablas mientras que cada instancia es una fila de la tabla.

Columnas con valores nulos

Noten que algunas de las propiedades en la clase Product admiten valores nulos (la declaración *Decimal?* indica que puede tener valor nulo). Si la columna en la base de datos admite valores nulos, entonces la propiedad correspondiente en la clase también debe aceptarlos:

```
01 public class Product
02 {
03     public int ProductId { get; set; }
04     public string ProductName { get; set; }
05     public decimal? UnitPrice { get; set; }
06     public bool Discounted { get; set; }
07     public int CategoryID { get; set; }
08
09     public virtual Category Category { get; set; }
10 }
```

Clase de contexto de EF (DbContext)

Luego de crear las clases Product y Category, usamos el componente Code First EF 4.1 para **crear una clase que sirve como contexto de datos y que crea la asociación entre nuestros modelos de clases POCO y la base de datos**:

```
1 public class Northwind : DbContext
2 {
3     public DbSet<Product> Products { get; set; }
4     public DbSet<Category> Categories { get; set; }
5 }
```

La clase Northwind es el medio usado para asociar las clases Product y Category con la base de datos. Deriva de la clase **DbContext** provista por la biblioteca Code First EF y expone dos propiedades que corresponden a tablas en la base de datos. En este ejemplo estamos aprovechando las reglas predeterminadas de convención en vez de configuración para crear las correspondencias entre tablas y clases.

Configurar la cadena de conexión

Ya hemos escrito todo el código necesario para definir la capa del modelo. Nuestro último paso, ante de poder usarlo, es configurar una cadena de conexión para la base de datos.

En nuestro caso específico, no queremos que la base de datos sea regenerada de manera automática, de manera que vamos a configurar la cadena de conexión a la base de datos de Northwind que ya tenemos. Esto se hace en el archivo web.config:

```
1 <connectionStrings>
2  <add name="Northwind"
3      connectionString="data source=.\SQLEXPRESS;
4              Integrated Security=SSPI;
5              AttachDBFilename=|DataDirectory|\Northwnd.mdf;
6              User Instance=true"
7      providerName="System.Data.SqlClient"/>
8 </connectionStrings>
```

Code First EF se adhiere a la convención de que las clases del contexto de datos tienen el mismo nombre que las cadenas de conexión. Por llamarse Northwind, nuestra clase buscará una cadena de conexión con el nombre Northwind. La cadena en sí configura una conexión a una base de datos local en SQL Express. También sería posible que señale a un servidor SQL remoto.

2.21 jQuery

JQuery es una librería de JavaScript creada en el año 2006 por John Resig, permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas Web, siendo compatible con la mayoría de los navegadores. Esto permite crear sin grandes esfuerzos páginas Web muy vistosas y dinámicas [jQuery].

Ventajas de utilizar jQuery

- Es de código abierto, y esta licenciado bajo un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libre y Privativos.
- Es pequeño (18 KB minified) y comprimido (114 KB, sin comprimir).
- Es muy popular, es decir, tiene una gran comunidad de usuarios y una buena cantidad de colaboradores que participan como promotores y evangelistas.
- Su base de plugins es muy amplia y ha tenido un crecimiento constante desde el lanzamiento de jQuery.
- Su API está completamente documentada, con ejemplos de código en línea, lo cual hace muy fácil su implementación.
- Es fácil, es decir, que ofrece formas útiles para evitar conflictos con otras bibliotecas JavaScript.
- Es abiertamente desarrollado, lo que significa que cualquiera puede contribuir con correcciones de errores, mejoras, y contribuir a su desarrollo.
- Su desarrollo es constante y consistente, es decir, siempre tenemos las actualizaciones disponibles.
- Su adopción por las grandes organizaciones contribuye a su estabilidad (por ejemplo, Microsoft, Dell, Bank of America, Digg, CBS, Netflix).
- Incorpora las especificaciones del W3C antes que los navegadores. A modo de ejemplo, jQuery es compatible con una buena mayoría de los selectores CSS3.
- En la actualidad es probado y optimizado para el desarrollo de los navegadores modernos (Chrome 1, Chrome noche IE, 6, IE 7, IE 8, Opera 9.6, Safari 3.2, WebKit noche, Firefox 2, Firefox 3, Firefox noche).
- Es muy poderosa en manos de los diseñadores, así como de programadores.
- jQuery no discrimina.
- Se ha mantenido una biblioteca de JavaScript (en lugar de un marco) en el corazón, mientras que en mismo tiempo que proporciona un proyecto hermano de widgets de interfaz de usuario y la aplicación desarrollo (jQuery UI).
- Su curva de aprendizaje es accesible, ya que se basa en conceptos que la mayoría de los desarrolladores y los diseñadores ya comprender (por ejemplo, CSS y HTML).

La filosofía de jQuery es "escribir menos, hacer más". Esta filosofía se puede dividir en tres conceptos [Lindley,2010]:

- Búsqueda de algunos elementos (a través de los selectores CSS) y hacer algo con ellos (a través de los métodos de jQuery).
- Múltiples métodos de encadenamientos de jQuery en un conjunto de elementos.
- Uso de la envoltura de jQuery y la iteración implícita

jQuery en Visual Studio 2010

Cuando se crea un proyecto nuevo con Visual Studio 2010, se añade automáticamente en la carpeta Scripts la librería de jQuery muestra la figura 15

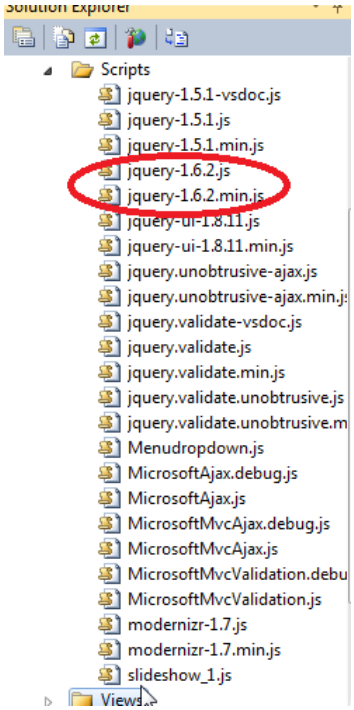


Figura 15: Librerías jQuery 1.6.2 importadas

Para este proyecto he agregado la última versión 1.6.2.js (para desarrollo) y la 1.6.2.min.js (producción), ya que VS2010 viene con la versión 1.5.1. En el capítulo de implementación muestro como agregar las librerías de jQuery UI.

Para utilizar jQuery en una página Web simple o maestra es necesario agregar un script en la cabecera para que haga referencia a la librería instalada. En la siguiente figura 16 se muestra este script.

```
<head>
    .........
        <script src="@Url.Content("~/Scripts/jquery-1.6.2.min.js")"
        type="text/javascript"></script>
    .........
</head>
```

Figura 16: Script que referencia a la librería jquery-1.6.2.min.js

Para aplicar JavaScript tras cargar la página, generalmente se utiliza el evento **ready** del objeto **document**, como se muestra en la figura 17,

```
document.onready = function () {
// Código que se ejecuta al cargar la página
};
```

Figura 17: Evento ready del objeto document

jQuery nos facilita una sintaxis abreviada de esta operación, usando un alias **$**, la figura 18 nos muestra esta síntesis.

```
$(function () {
// Código que se ejecuta al cargar la página
});
```

Figura 18: Sintaxis abreviada del evento ready

En la tabla 7 [DotNetMania,76/2010] muestra algunos de los selectores básico que se pueden utilizar con jQuery, para saber más sobre su manejo se recomienda visitar su sitio oficial [jQuery], manual en internet [DesarrolloWeb] y el sitio Web [w3schools].

Tabla de algunos selectores básicos	
$(".laClase")	Selecciona todos los elementos que tiene **class = "laClase"**
$("#elId")	Selecciona todos los elementos que tienen **id = "elId"**
$("div")	Selecciona todos los elementos de tipo ** **
$("p a")	Selecciona todo los elementos de tipo **<a>** que se encuentren dentro de un **<p>**
$("img[alt]")	Selecciona todos los elementos de tipo **<img>** que tengan e1 atributo **alt**
$("p#miId")	Selecciona todos los elementos de t ipo **<p>** que tengan **id = "miId"**
$("li > div")	Selecciona todos los elementos de tipo ** ** que estén contenidos directamente dentro de un ** **
$("a[Href$ = .xls]")	Selecciona todos los hiper enlaces (**<a>**) cuyo atributo **href** tiene un valor que termina en **.xIs**

Tabla 7: Algunos selectores básicos de jQuery

En la tabla 8 [DotNetMania76/2010] se muestra una selección de selectores basados en su posición relativa.

Tabla de algunos selectores basados en su posición	
$("p : first")	Selecciona el primer **<p>** de la página.
$("div.laClase:last")	Selecciona el último ** ** que exista en la página que tenga la clase **laClase**.
$("li : first-child")	Selecciona todos los elementos ** ** que sean el primer hijo de su contenedor (es decir, el primer ítem de cada lista).
$("li : nth-child(2)")	Selecciona todos los elementos ** ** que sean el hijo número 2 dentro de su contenedor (es decir, el segundo ítem de cada lista).
$("ul : only-child")	Selecciona todos los elementos ** ** que sean los únicos elementos dentro de su contenedor.
$("table > tr : odd")	Selecciona todos los elementos **<tr>** que ocupen una posición impar dentro de una tabla. Se utiliza para crear efectos de "papel pijama".
$(".laClase : lt(4)")	Selecciona los cuatro primeros elementos de la página que tengan la c1ase **laClase** (es decir, los que ocupan las posiciones 0, 1, 2 y 3).

Tabla 8: Selectores de jQuery basados en su posición

La librería jQuery es fácilmente extensible mediante plugins, permitiendo añadir funciones y métodos no existentes en la librería original, incorporando nuevas capacidades a la misma.

En el sitio Web oficial [jQuery] existen muchas de ellas listas para incorporar a nuestros proyectos.

2.22 jQuery UI

jQuery UI es una librería de componentes Open Source para el marco jQuery que le añaden un conjunto de plug-ins, widgets y efectos visuales para la creación de aplicaciones Web.

jQuery UI se compone de 4 módulos [WikipediaUI]:

- **Núcleo**

  Contiene las funciones básicas para el resto de módulos.

- **Interacción**

  Añade comportamientos complejos a los elementos:

  - **Draggable:** Hace al elemento arrestable.
  - **Droppable:** Permite que el elemento responda a elementos arrastables.
  - **Resizable:** Permite redimensionar el elemento.
  - **Selectable:** Permite seleccionar entre una lista de elementos.
  - **Sortable:** Ordena una lista de elementos.

- **Widgets**

  Es un conjunto completo de controles UI. Cada control tiene un conjunto de opciones configurables y se les pueden aplicar estilos CSS.

  - **Acordeón:** Menú con efecto acordeón.
  - **Dialog:** Ventanas con contenido.
  - **Slider:** Elemento para elegir en un rango de valores.
  - **Tabs:** Pestañas.
  - **Datepicker:** Calendario gráfico.
  - **Progressbar:** Barra de progreso.
  -

- **Efectos**
  - Una API para añadir transiciones animadas y facilidades para interacciones

jQuery UI es un paquete de plugins de validez oficial de la biblioteca jQuery de JavaScript que nos ayudan a reducir de bastante rápida el diseño de la interfaz de usuario para nuestras aplicaciones Web modernas, haciendo de ellas más amigables e interactivas. La máxima compatibilidad, estabilidad y el mínimo tiempo y esfuerzo, hace que estos plugins nos faciliten el desarrollo de la interfaz de usuario.
En su sitio oficial Web [jQueryUI] existen muchos ejemplos y documentación lista para agregar a nuestros proyectos.
En el capítulo 2 del Desarrollo e Implementación se explica cómo incorporar estos plugins a la aplicación.

2.23 Responsive Web Desig

Un artículo escrito por Ethan Marcotte [MarcotteEtahn] el 25 de mayo de 2010 hace referencia a esta nueva metodología para abordar una manera de diseñar y crear, de acuerdo a los distintis dispositivos (smartphone, tablets, PC, portátiles, TV), navegadores, tamaños de pantalla y orientaciones mediante la creación de sitios Web flexibles, fluido y adaptables. En vez de responder a las necesidades actuales de una versión web de escritorio adaptado a la resolución de pantalla más comunes, junto con una versión móvil particular (a menudo específica para un único dispositivo móvil), la idea es abordar el tema al revés: el uso flexible y diseños de fluidos que se adaptan a casi cualquier dispositivo.

Esta metodología recomienda que empezemos diseñando nuestra aplicación pensando en pantallas pequeñas e ir avanzando hacia pantallas más grandes, es decir empezar de abajo a arriba, "En lugar de comenzar con un diseño del sitio de sobremesa e ir ampliando hacia abajo y volver a fluir el contenido para las pequeñas ventanas, se debe diseñar para la ventana pequeña y luego progresivamente mejorar el diseño y el contenido hacia grandes ventanas [Frain,2012]".

Para realizar esta adaptación me modifica la estructura html del sitio Web a través de su CSS, se aplica una formula que convierte en porcentajes en lugar de pixeles los distintos tamaños de estos elementos.

De esta manera, en vez de crear distintas plantillas o webs, se hacen que se adapte a los distintos dispositivos.

En la siguiente figura 19 se muestra un ejemplo:

Figura 19: http://www.xacobe.net/blog/diseno-sensible-responsive-design

En la figura anterior se puede apreciar como cambia la dispocion, tamaño y area visible de las imágenes.
En la vista tableta las imágenes se han reducido en comparación con las de escritorio.
En la vista móvil, además de reducirse más el tamaño de las imágenes la composición ha cambiado de dos a una columna, haciendo que los bloques laterales se intercalen con el contenido, además el menú ha cambiado de disposición.

El Responsive Web Design, cuya traducción en castellano es "**Diseño Web flexible**", "**Diseño Web Adaptable**" o "**Diseño Web Responsable**", "**Diseño Plastico**", "**Diseño Elástico**", "**Diseño Adaptativo**", "**Diseño Líquido**" [Frain,2012], utiliza las siguientes tecnicas:

- Meta Tag

  Se usa la etiqueta meta tag "viewport" para adaptar el sitio Web a las diferentes resoluciones de los dispositivos móviles (Android, IOS, Blackberry OS, etc.), tables, PC, portátiles, etc. De esta manera se le dice al navegador, que fuerze el ancho de la web al ancho de la pantalla:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1" />
```

  Es importante entender que "viewport" y el tamaño de la pantalla no son lo mismo. Viewport se relaciona con el área de contenido dentro de la ventana del navegador, con exclusión de las barras de herramientas, pestañas, etc. De manera más sucinta, se refiere a la zona donde un sitio web realmente se muestra. El tamaño de la pantalla se refiere para el área de visualización física de un dispositivo, incluyendo elementos tales como el navegador, la barra de direcciones, fichas y cuadros de búsqueda, etc.

- Fluid grid (Grid Fluidas)

  A través del uso de valores proporcionales (porcentajes y ems) y de las propiedades max-width o max-height en nuestro css3, se hace que los distintos elementos se adapten al tamaño del navegador.

  Grid flexibles es uno de los pilares del RWD, se deja de utilizar los pixeles para utilizar porcentajes o ems (unidad de medida basada en la altura de la fuente).

Se aplica la siguiente formula:

target / context = result

Si tenemos una pagina Web con la siguiente estructura:

```
<div id="wrapper">
      <!–cabecera y navegación -->
      <div id="header">
            <div id="logo">
                  Mi Logo
            </div>
            <div id="navigation">
                  <ul>
                        <li><a href="#">Enlace 1</a></li>
                        <li><a href="#">Enlace 2</a></li>
                        <li><a href="#">Enlace 3</a></li>
                  </ul>
            </div> <!–fin id navegación -->
      </div> <!–fin id cabecera -->
……..
……..
</div> <!–fin id wrapper -->
```

El CSS de la página anterior es la siguiente:

```
#wrapper {
      margin-right: auto;
      margin-left: auto;
      width: 960px;
}

#header {
      …….
      …..
      width: 940px;
}

#navigation {

      width: 940px;

}
```

Para aplicar el RWD se tiene que aplicar la formula arriba mencionada, para este caso, se tiene que transformar este CSS a porcentajes de la siguiente manera:

```
#wrapper {
```

```
        margin-right: auto;
        margin-left: auto;
        width: 96%; /* se convierte a % */
}

#header {
        …….
        …..
        width: 97.9166667%; /* 940 ÷ 960 */
}

#navigation {

        width: 97.9166667%; /* 940 ÷ 960 */

}
```

Supongamos que el contexto normal para el tamaño de la fuente del cuerpo es de 16 píxeles. Si el diseñador especifica que el H1 debe ser de 24 píxeles, se puede calcular lo siguiente:

24 ÷ 16 = 1.5

Esto da como resultado en la siguiente estilo CSS:

```
h1 {

    font-size: 1.5em;

}
```

Siempre se debe de tener en cuenta el contexto. Continuando con el ejemplo anterior, si tenemos un elemento dentro de la H1 que tiene que ser de 12 píxeles, se utiliza el H1 actual como el contexto. El contexto es ahora 24 píxeles, de modo que el cálculo del contexto para "H1 a" es:

12 ÷ 24 = 0.5

Y el estilo CSS es el siguiente:

```
h1 {

    font-size: 0.5em;

  }
```

- Media Queries de CSS3

  El W3C ha mejorado los media queries en CSS3 [MediaQueries], dando un gran paso adelante para su utilización.
  Hoy en día, puede utilizar las consultas de los media queries (medios de comunicación), las nuevas características incluyen la anchura, altura, ancho, altura máxima, la altura del disposito, la orientación, relación de aspecto, resolución, etc.

  En los siguientes ejemplos de media queries se utilizan los @media screen de css3:

  a) Para resoluciones inferiores a 980px:

```
@media screen and (max-width: 980px) {
   #contenedor { width: 100%; }
   #contenido { width: 70%; }
   #lateral {width: 30%; }
}
```

  b) Para resoluciones inferiores a 700px:

```
@media screen and (max-width: 700px) {
   #contenido, #lateral {
      width: auto;
      float: none;
   }
}
```

  En el caso a) adaptamos el diseño a resoluciones inferiores de 980 pixeles, donde nuestros div principales se ajustan basados en porcentajes de ancho. En el caso b) se adpata las resoluciones inferiores a 700 pixeles, y lo que se hace es quitar el flotamiento entre divs para que se coloquen uno encima del otro.

- Fluid images

  Básicamente, esta característica nos permite adaptar las imágenes u otros medios para cargar de manera diferente en función del dispositivo, ya sea por la escala o mediante el uso de la propiedad de desbordamiento de CSS.

La escala en el CSS es bastante simple de implementar, tanto para imágenes y video. Se puede configurar el elemento de los medios de comunicación max-width al 100 por ciento, y el navegador hará que las imagenes se encogan y expanden en función de su contenedor. Se debe proporcionar a la imagen en la mejor calidad y tamaño posible y luego dejar CSS adaptar la imagen al tamaño correcto.

Si se tiene una etiqueta que muestra una imagen, su diseño debe ser el siguiente:

```
<div id="sidebar">
        <div class="BloqueLateral">
                <h4>Mis Imagenes...</h4>
                <img src="img/MiImagen1.jpg" altI="Mi imagen 1" />
                ……
                ……
        </div>
</div>
```

El CSS de esta etiqueta se muestra a continuación:

```
img{
        max-width: 100%;
}
.BloqueLateral img{
        max-width: 45%
}
```

Con estas modificaciones anteriores se logra que las imagines seba fluidas y flexibles, adaptandose al dispositivo donde va ser visualizado.

CAPÍTULO 3: DESARROLLO e IMPLEMENTACIÓN

3.1 Marco de trabajo

La siguiente implementación está enmarcada en el desarrollo que he realizado para el grupo de investigación del **"Estudio Técnico de la Obra de Sorolla (ETOS")** de la universidad Rey Juan Carlos y la Comunidad de Madrid.

3.2 Descripción de la aplicación

ETOS (Estudio técnico de la obra de Sorolla) nace de la necesidad sentida desde distintos ámbitos por un grupo de reconocidos profesionales ligados desde hace tiempo al estudio, análisis y restauración de la producción del pintor Joaquín Sorolla Bastidas (Valencia 1863 – Cercedilla 1923) [WikipediaSorolla]. Entendiendo por estudio técnico, el análisis cromático de su paleta, del lienzo y marcos, de manera que se pueda disponer de los suficientes criterios de atribución científica para enmarcar y deslindar la obra del pintor con parámetros objetivos alejados de la especulación. En este campo no se parte de cero, antes bien, son ya numerosos los trabajos que se han llevado a cabo. Para coordinar y reunir estos esfuerzos y avances es de trascendental importancia reunirlos en una base de datos apropiada y, desde ese trabajo previo, orientar la toma de datos futura siguiendo unas mismas pautas [ETOS11].

En la siguiente figura 20 se muestra la página inicial del sitio Web.

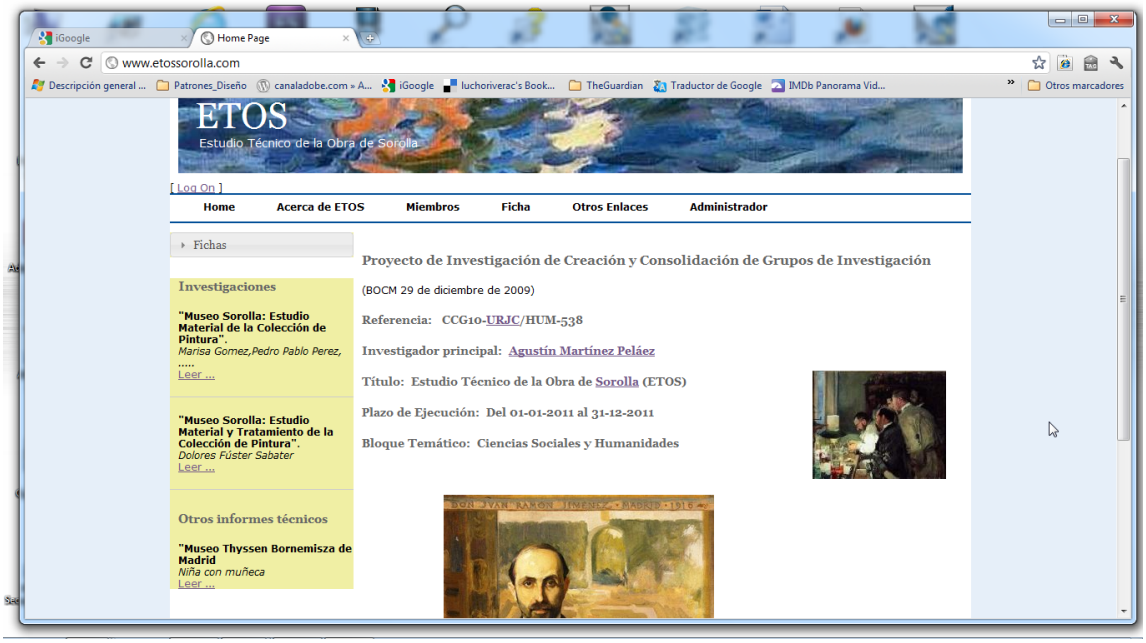


Figura 20: Página inicial de sitio Web de ETOS

3.3 Tecnologías y herramientas empleadas

ASP.NET MVC 3 es una plataforma gratuita, Open Source, que está incluida de serie en .NET 4.0, pudiendo descargarse para su uso también con versiones anteriores (.NET 3.5). MVC 3, aparecida en Enero de 2011, añade nuevas e interesantes características a la colección de herramientas del programador Web.

La siguiente figura 21 nos da una idea de las herramientas gratuitas para desarrollar una aplicación ASP.NET MVC

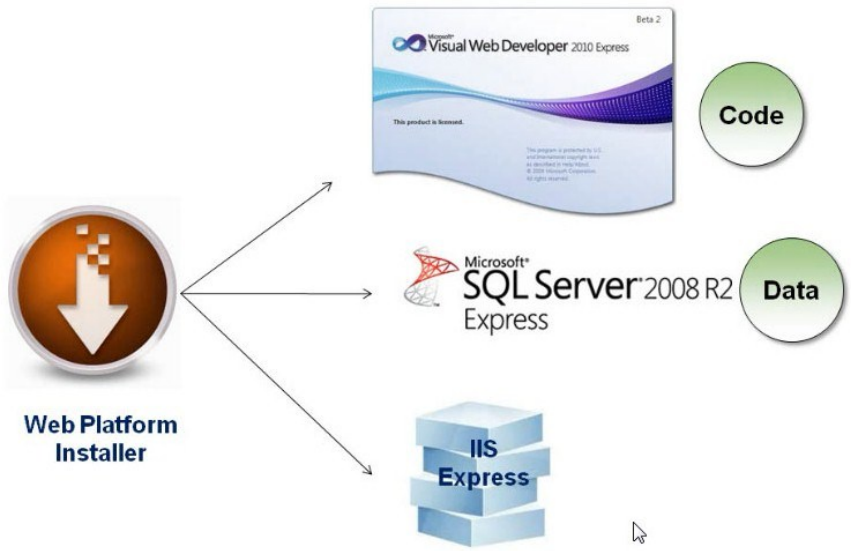


Figura 21: Herramientas Open source para desarrollar una aplicación MVC

Todas estas herramientas se pueden instalar desde Web Platform Installer 3.0 disponible en [WebPlatformInstaller].

También se puede bajar ASP.NET MVC 3 desde [ASPNETMVC].

Para el desarrollo de esta aplicación se a utilizar:

- Visual Studio 2010 ULTIMATE.
- SQL Server 2008 R2

3.4 Modelado de la base de datos

ETOS es el estudio de 23 obras del pintor Joaquín Sorolla Bastidas.
Cada Obra está compuesta por la siguiente estructura:

- Información básica.
- Documentación Antigua Relacionada.
- Restauración.
- Técnicas Analíticas.
- Estudio Radiográfico.
- Resultado de los Análisis.
- Tratamiento de Conservación.

- Galería de Imágenes.
- Bibliografía Específica.

Cada estructura contiene muchos campos para mostrar información muy variada de cada uno de ello.

La figura 22 muestra el esquema del modelo de datos de la aplicación.

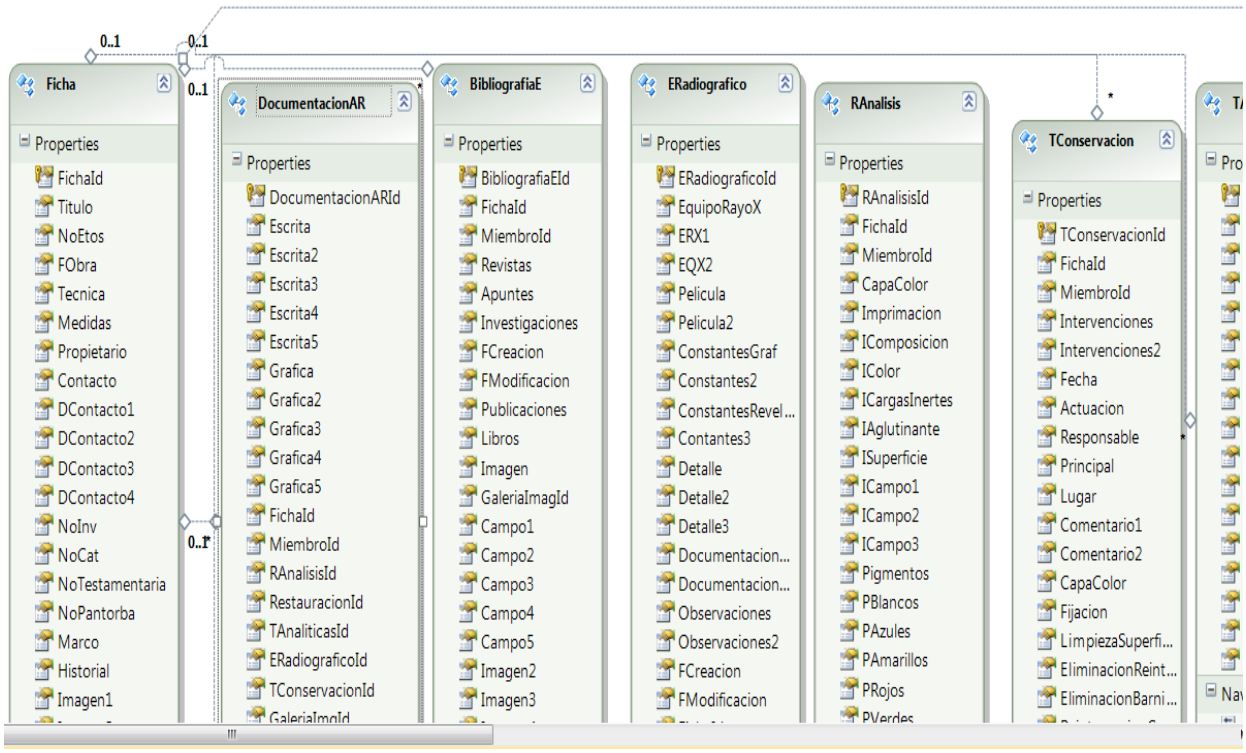
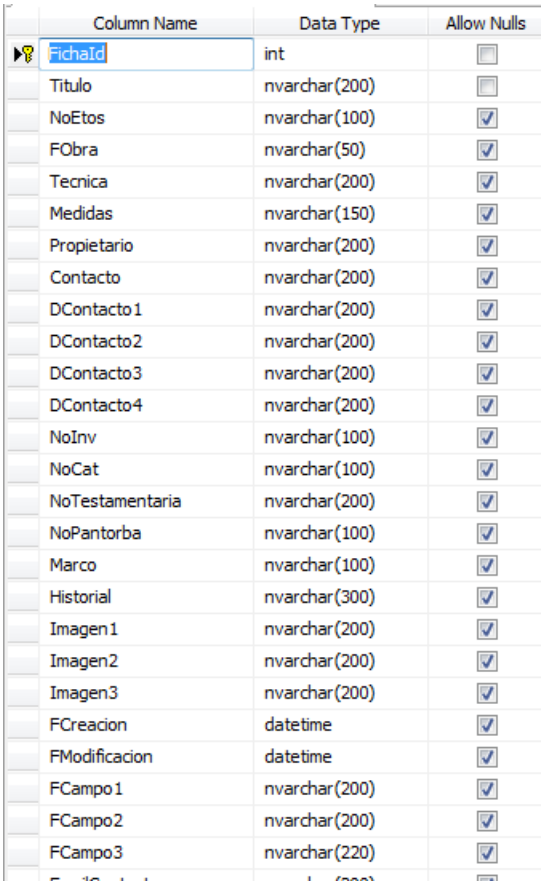


Figura 22: Esquema del Modelo de datos

Se debe observar con mucho detalle los nombre de cada una de las tablas y su clave primaria, ya que van a resultar de gran importancia a la hora de generar el codigo automático, se genera todo el código siguiendo las Convenciones de Code First [CodeFirstConvenciones].

En la siguiente figura 23 se muestra el detalle de la tabla Ficha.



Column Name	Data Type	Allow Nulls
FichaId	int	☐
Titulo	nvarchar(200)	☐
NoEtos	nvarchar(100)	☑
FObra	nvarchar(50)	☑
Tecnica	nvarchar(200)	☑
Medidas	nvarchar(150)	☑
Propietario	nvarchar(200)	☑
Contacto	nvarchar(200)	☑
DContacto1	nvarchar(200)	☑
DContacto2	nvarchar(200)	☑
DContacto3	nvarchar(200)	☑
DContacto4	nvarchar(200)	☑
NoInv	nvarchar(100)	☑
NoCat	nvarchar(100)	☑
NoTestamentaria	nvarchar(200)	☑
NoPantorba	nvarchar(100)	☑
Marco	nvarchar(100)	☑
Historial	nvarchar(300)	☑
Imagen1	nvarchar(200)	☑
Imagen2	nvarchar(200)	☑
Imagen3	nvarchar(200)	☑
FCreacion	datetime	☑
FModificacion	datetime	☑
FCampo1	nvarchar(200)	☑
FCampo2	nvarchar(200)	☑
FCampo3	nvarchar(220)	☑

Figura 23: Detalle de la tabla Ficha

Esta tabla se llama Ficha y la clave primaria es FichaId, con estos dos parámetros: nombre de la tabla y clave primaria se genera automaticamente el código para realizar las operaciones CRUD de la base de datos de la aplicación para cada una de las tablas que la siguiendo las Convenciones de Code First [CodeFirstConvenciones] y [ConvencionesCoC].

3.5 Generación automática del código

Entity Framework utiliza plantilla T4 [PantillasT4] para generar de manera automática nuestras clases, permitiendo tener cierto grado de control sobre su proceso de generación [MSDNT4].

Hacemos click derecho en un espacio en blanco del diseñador de Entity Framework y seleccionamos **Add Code Generation ítem…**.

La figura 24 nos muestra este paso.

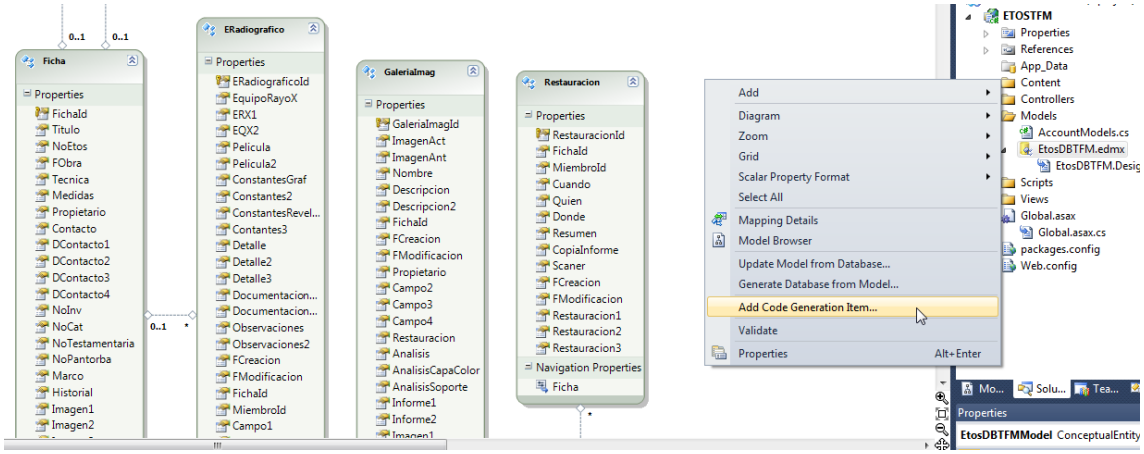


Figura 24: Esquema del Modelo generado

De las plantillas instaladas seleccionamos Code → ADO.NET DbContext Generator.

Dejamos el nombre por defecto (Model1.tt) → Add.

Nos genera dos plantillas T4, generando de manera automática las clases que necesitamos:

- Model1.Context.tt (Clase DbContext)
- Model1.tt (Clases POCO)

En las siguientes figuras 25 y figura 26 podemos apreciar este proceso.

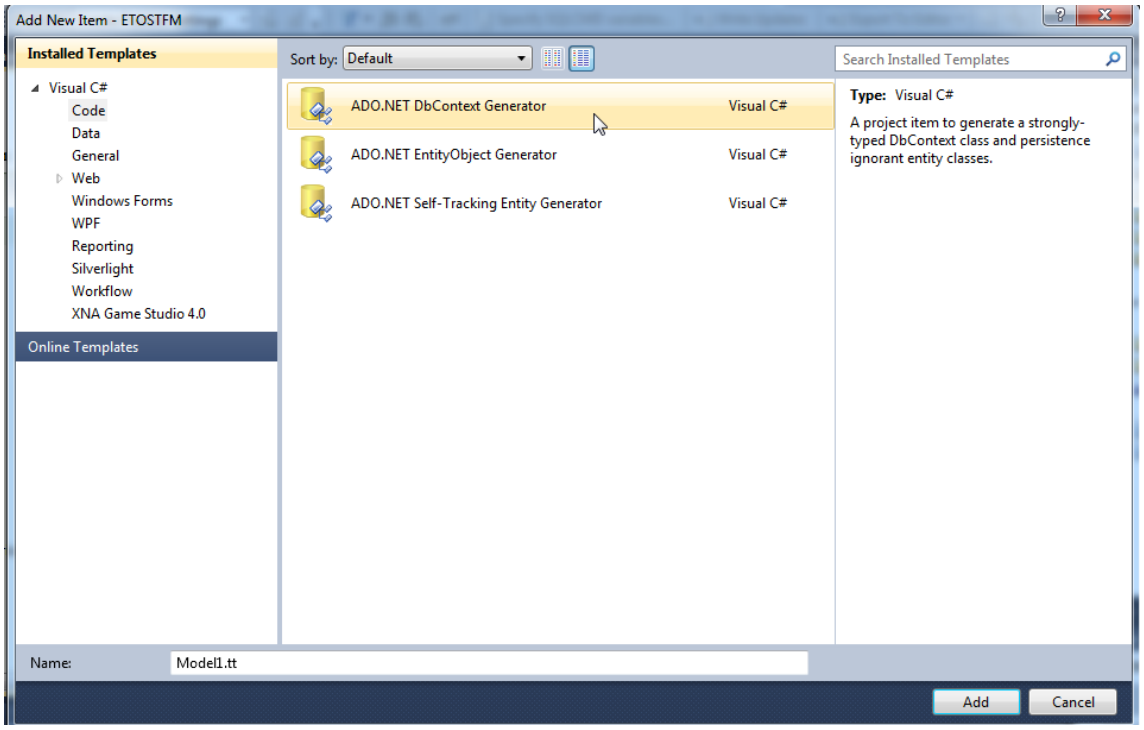


Figura 25: Esquema del Modelo generado

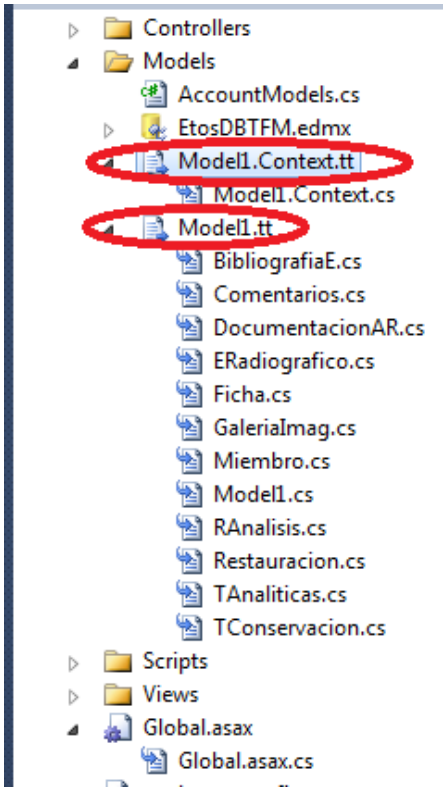


Figura 26 Plantilla T4 con las clases creadas

La plantilla Model1.Context.cs creó una clase **EtosDbTFMEntities** que deriva de **DbContext**, es la que contiene las colecciones de entidades de nuestra base de datos. Contiene un **DbSet** por cada entidad. Sirve como un contexto de datos y crea la asociación entre nuestro modelo de las clases **POCO** y la base de datos.
La figura 27 nos muestra el código generado automático.

```
namespace ETOSTFM.Models
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    public partial class EtosDBTFMEntities : DbContext
    {
        public EtosDBTFMEntities()
            : base("name=EtosDBTFMEntities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public DbSet<BibliografiaE> BibliografiaE { get; set; }
```

```
        public DbSet<Comentarios> Comentarios { get; set; }
        public DbSet<DocumentacionAR> DocumentacionAR { get; set; }
        public DbSet<ERadiografico> ERadiografico { get; set; }
        public DbSet<Ficha> Ficha { get; set; }
        public DbSet<GaleriaImag> GaleriaImag { get; set; }
        public DbSet<Miembro> Miembro { get; set; }
        public DbSet<RAnalisis> RAnalisis { get; set; }
        public DbSet<Restauracion> Restauracion { get; set; }
        public DbSet<TAnaliticas> TAnaliticas { get; set; }
        public DbSet<TConservacion> TConservacion { get; set; }
    }
}
```

Figura 27 Código generado por la plantilla Model1.Context.cs

La plantilla Model1.tt creó las clases por cada una de las tablas seleccionadas anteriormente.

La biblioteca de Code First (código primero) de EF 4.1 nos permite usar objetos simples y llanos del CLR (o POCO) para representar las entidades en la base de datos. Por lo tanto, no necesitamos usar clases bases especiales ni implementar una interfaz específica o agregar atributos de ningún tipo. De esta manera nuestras clases se mantienen en ignorancia de su mecanismo de almacenaje.

La Figura 28 muestra la clase generada automáticamente para la tabla de la base de datos, en ella se puede apreciar que las clase Ficha es clase POCO y se utiliza para representa las tablas Ficha de nuestra base de datos EtosDbTFM.

Las propiedades de esta clase representan las columnas de esta tabla mientras que cada instancia es una fila de su tabla respectiva.

```
namespace ETOSTFM.Models
{
    using System;
    using System.Collections.Generic;

    public partial class Ficha
    {
        public Ficha()
        {
            this.BibliografiaE = new HashSet<BibliografiaE>();
            this.DocumentacionAR = new HashSet<DocumentacionAR>();
            this.ERadiografico = new HashSet<ERadiografico>();
            this.GaleriaImag = new HashSet<GaleriaImag>();
            this.RAnalisis = new HashSet<RAnalisis>();
            this.Restauracion = new HashSet<Restauracion>();
            this.TAnaliticas = new HashSet<TAnaliticas>();
            this.TConservacion = new HashSet<TConservacion>();
        }

        public int FichaId { get; set; }
        public string Titulo { get; set; }
        ……..
```

```
        ……..
        public virtual ICollection<BibliografiaE> BibliografiaE { get; set; }
        public virtual ICollection<DocumentacionAR> DocumentacionAR { get; set; }
        public virtual ICollection<ERadiografico> ERadiografico { get; set; }
        public virtual ICollection<GaleriaImag> GaleriaImag { get; set; }
        public virtual ICollection<RAnalisis> RAnalisis { get; set; }
        public virtual ICollection<Restauracion> Restauracion { get; set; }
        public virtual ICollection<TAnaliticas> TAnaliticas { get; set; }
        public virtual ICollection<TConservacion> TConservacion { get; set; }
    }
}
```

Figura 28: Código generado para la tabla Ficha

Code First de EF 4.1 facilita el uso de relaciones de claves primarias y externas en la base de datos y expone propiedades en las clases del modelo que nos ayudan a navegar las clases usando esas relaciones.

En el código arriba, la clase Ficha expone la propiedad TConservacion, y la clase TConservacion expone Ficha. Estas propiedades permiten el uso de relaciones de clave primaria y externa entre las dos tablas para obtener instancias del modelo. Estas propiedades siguen siendo de tipo POCO sin requerir un tipo especial de colección en su definición.
Las propiedades de asociación marcadas como virtuales son, por defecto, cargadas de modo diferido. Es decir, si uno obtiene una entidad Ficha, la información sobre la tabla TConservacion no es cargada hasta que sea leída la propiedad TConservacion (a menos que carguemos la propiedad explícitamente en la consulta LINQ).

3.6 Controladores y Vistas automáticas

Al agregar los controladores, se nos genera de manera automática todo el código necesario, tanto para los Controladores como para las Vistas necesarias para realizar las operaciones CRUD (Create, Read, Update y Delete) desde nuestra aplicación hacia nuestra base de datos.

A continuación se detalla el proceso de creación del Controladores Ficha, ya que todos siguen el mismo método, solo hay que cambiar algunos parámetros a la hora de su creación. Todo ello se explicar con más detalle de forma grafica

Creación del Controlador Ficha

En la carpeta Controllers hacer click derecho → Add → Controller.

La siguiente figura 29 nos muestra como agregar un nuevo controlador

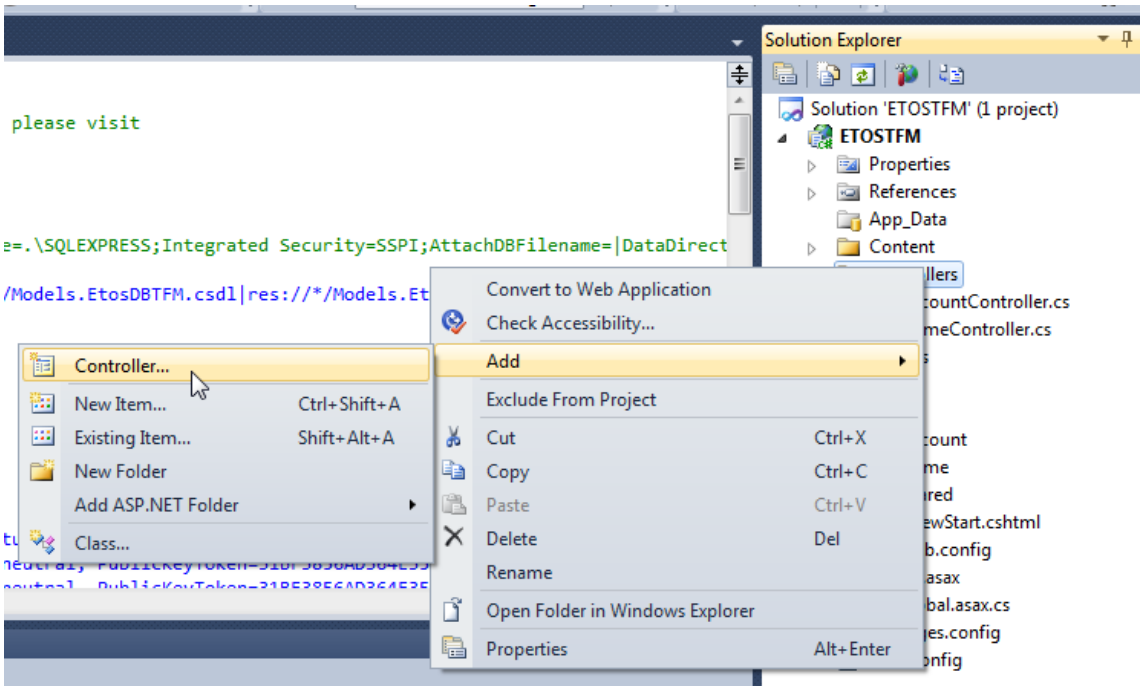


Figura 29: Creación del controlador Ficha

Nombra el Controlador como: **FichaController** (Ficha es el nombre de la tabla de la base de datos)

En Scaffolding options, en Template elegir:

Controller with read/write actions and views, using Entity Framework

En Model Clases, elegir :

Ficha(ETOSTFM.Models)

En **Data context** class hay que tener mucho cuidado en este punto, debemos escoger el nombre de nuestra cadena de conexión que elegimos para nuestro archivo Web Config :

EtosDBTFMEntities(ETOSTFM.Models)

En Views elegimos Razor(CSHTML).

La figura 30 nos muestra con más detalle este paso muy importante.

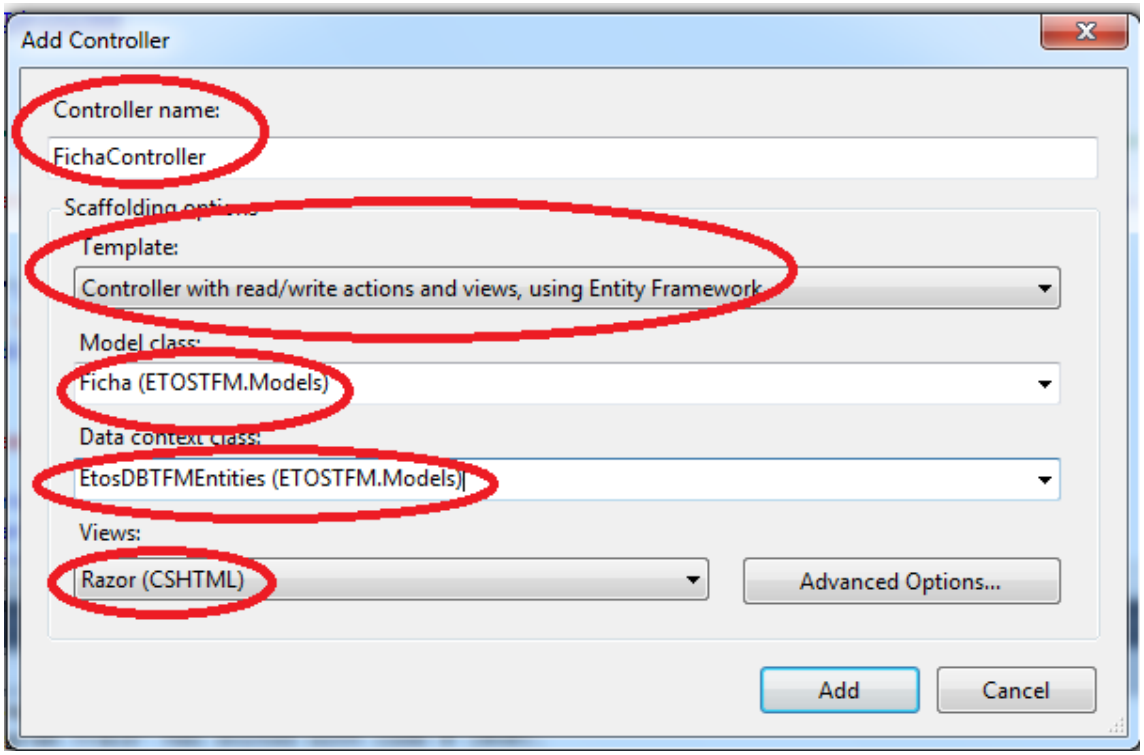


Figura 30: Agregando el Controlador Ficha

Como se puede ver en la siguiente figura 31, automáticamente se nos genera todo el código del Controlador Ficha así como sus respectivas Vistas necesarias para poder trabajar inmediatamente con nuestra base de datos

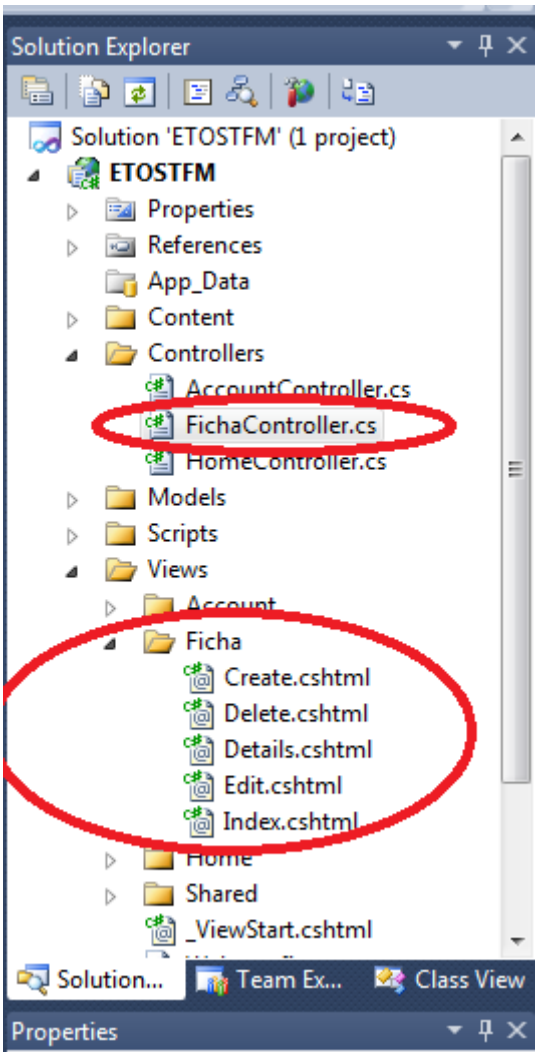


Figura 31: Controlador y Vistas de la tabla Ficha

El código automático generado para el Controlador Ficha (ControllerFicha) se muestra en la siguiente figura 32.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using ETOSTFM.Models;

namespace ETOSTFM.Controllers
{
    public class FichaController : Controller
    {
        private EtosDBTFMEntities db = new EtosDBTFMEntities();

        // GET: /Ficha/
```

```
public ViewResult Index()
{
    return View(db.Ficha.ToList());
}

// GET: /Ficha/Details/5

public ViewResult Details(int id)
{
    Ficha ficha = db.Ficha.Find(id);
    return View(ficha);
}

// GET: /Ficha/Create

public ActionResult Create()
{
    return View();
}

// POST: /Ficha/Create

[HttpPost]
public ActionResult Create(Ficha ficha)
{
    if (ModelState.IsValid)
    {
        db.Ficha.Add(ficha);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(ficha);
}

// GET: /Ficha/Edit/5

public ActionResult Edit(int id)
{
    Ficha ficha = db.Ficha.Find(id);
    return View(ficha);
}

// POST: /Ficha/Edit/5

[HttpPost]
public ActionResult Edit(Ficha ficha)
{
    if (ModelState.IsValid)
    {
        db.Entry(ficha).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(ficha);
}

// GET: /Ficha/Delete/5
```

```
        public ActionResult Delete(int id)
        {
            Ficha ficha = db.Ficha.Find(id);
            return View(ficha);
        }

        // POST: /Ficha/Delete/5

        [HttpPost, ActionName("Delete")]
        public ActionResult DeleteConfirmed(int id)
        {
            Ficha ficha = db.Ficha.Find(id);
            db.Ficha.Remove(ficha);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        protected override void Dispose(bool disposing)
        {
            db.Dispose();
            base.Dispose(disposing);
        }
    }
}
```

Figura 32: Código automático generado para el Controlador Ficha

Vista Ficha creada automáticamente

La figura 33 nos muestra la Vista Ficha/Index (Controller/Action)

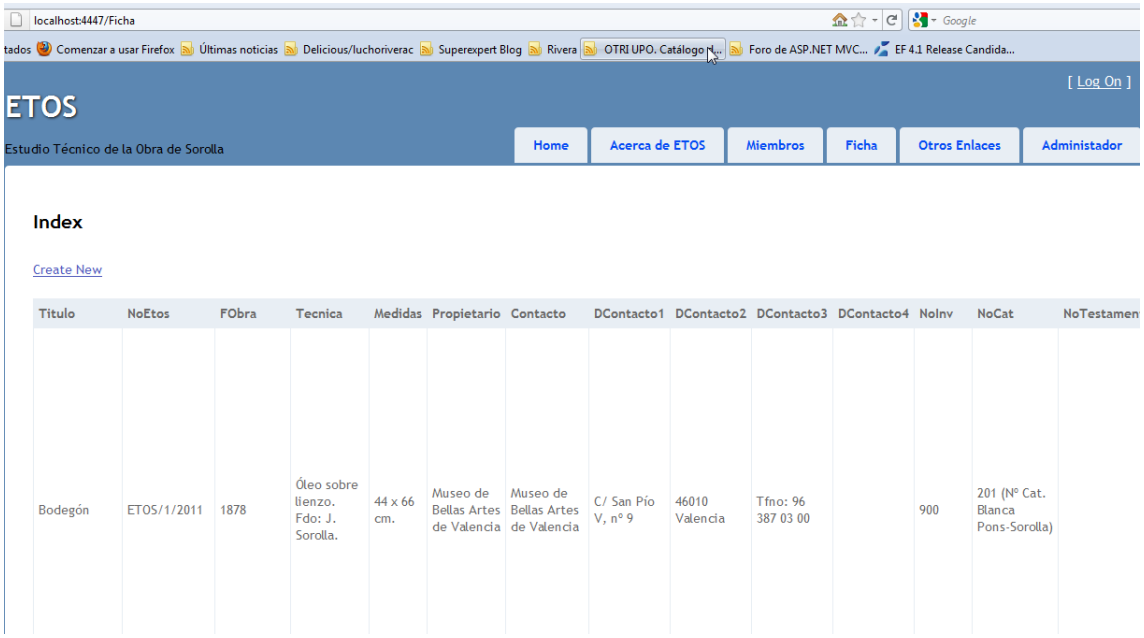


Figura 33: Vista Ficha generada automáticamente

La figura 34 nos muestra el código generado automáticamente para la Vista Ficha/Index, como se puede apreciar esta generado en HTML puro en forma de tabla y el código Razor (@) para generarlo de forma dinámica.

```
@model IEnumerable<ETOSTFM.Models.Ficha>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table>
    <tr>
        <th>
            Titulo
        </th>
        <th>
            NoEtos
        </th>
        <th>
            FObra
        </th>
        <th>
            Tecnica
        </th>
        <th>
            ……
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Titulo)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.NoEtos)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.FObra)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Tecnica)
        </td>
        ……

        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.FichaId }) |
            @Html.ActionLink("Details", "Details", new { id=item.FichaId }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.FichaId })
        </td>
    </tr>
}
```

```
</table>
```

Figura 34: Código HTML generado automáticamente para la Vista Ficha/Index

.

La figura 35 nos muestra la Vista Ficha/Edit/1 (Controller/Action/Id)

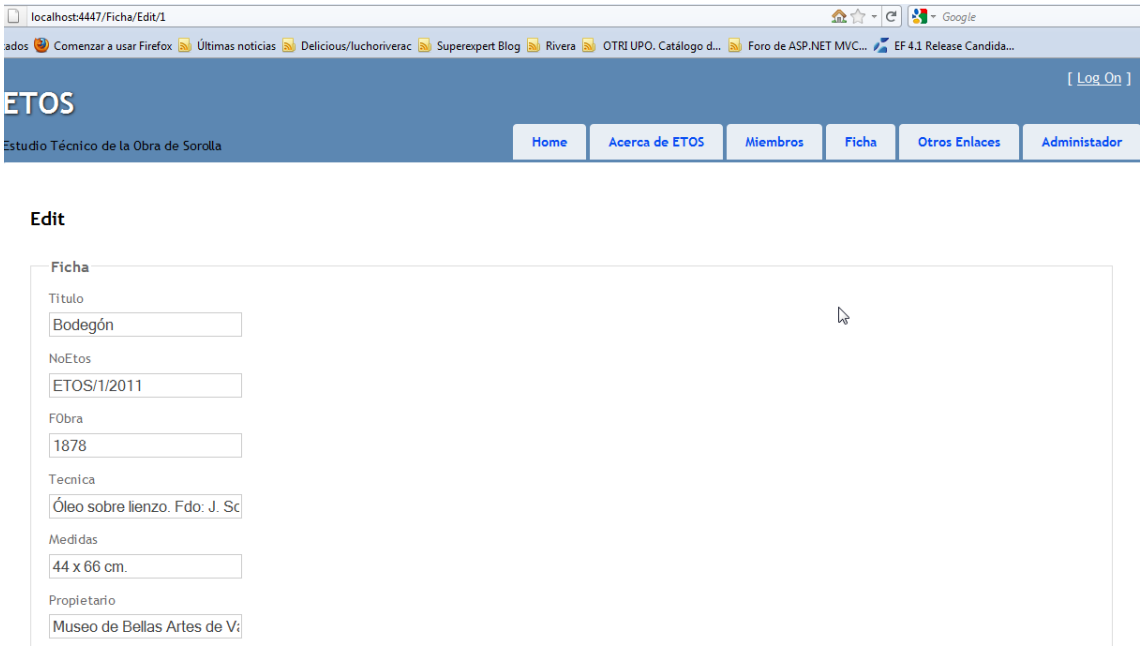


Figura 35: Vista Ficha/Edit/1 generada automáticamente

La siguiente figura 36 nos permite observar el código HTML generado para esta Vista.

```
@model ETOSTFM.Models.Ficha

@{
    ViewBag.Title = "Edit";
}

<h2>Edit</h2>

<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")"
type="text/javascript"></script>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Ficha</legend>

        @Html.HiddenFor(model => model.FichaId)

        <div class="editor-label">
            @Html.LabelFor(model => model.Titulo)
        </div>
        <div class="editor-field">
```

```
            @Html.EditorFor(model => model.Titulo)
            @Html.ValidationMessageFor(model => model.Titulo)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.NoEtos)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.NoEtos)
            @Html.ValidationMessageFor(model => model.NoEtos)
        </div>
            . . . . . .

        <p>
            <input type="submit" value="Save" />
        </p>
    </fieldset>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

Figura 36: Vista Ficha/Edit/1

La siguiente figura 37 nos muestra la Vista Ficha/Create.

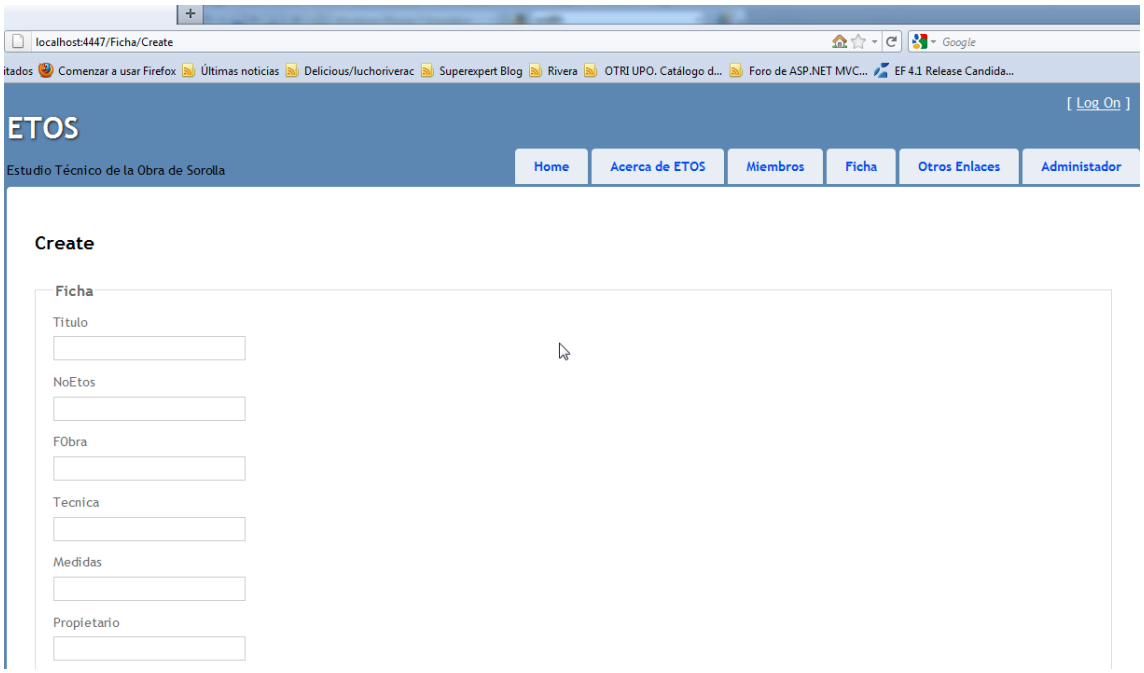


Figura 37: Vista autogenerada para Ficha/Create

En la siguiente figura 38 nos muestra en detalle el código HTML autogenerado para esta Vista.

```
@model ETOSTFM.Models.Ficha

@{
```

```
    ViewBag.Title = "Create";
}

<h2>Create</h2>

<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")"
type="text/javascript"></script>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Ficha</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Titulo)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Titulo)
            @Html.ValidationMessageFor(model => model.Titulo)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.NoEtos)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.NoEtos)
            @Html.ValidationMessageFor(model => model.NoEtos)
        </div>
            . . . . .

        <div class="editor-label">

        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

Figura 38: Código HTML para la vista Ficha/Create

La siguiente figura 39 nos muestra la Vista Ficha/Delete/1.

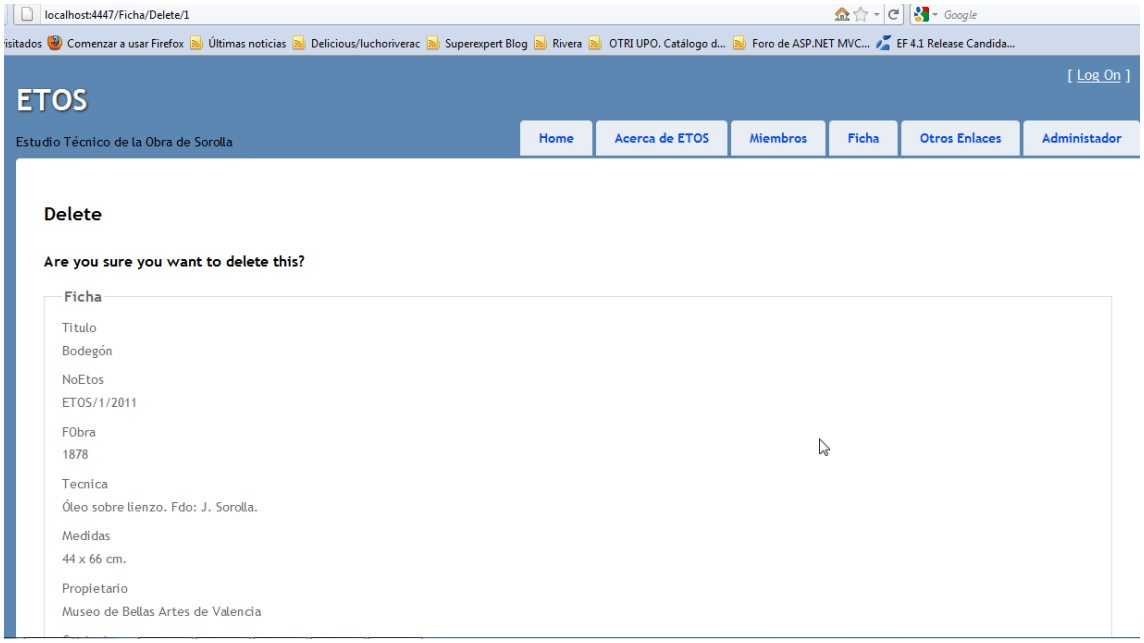


Figura 39: Vista Ficha/Delete/1 autogenerada

La siguiente figura 40 nos muestra en detalle el código autogenerado para la Vista Ficha/Delete/1.

```
@model ETOSTFM.Models.Ficha

@{
    ViewBag.Title = "Delete";
}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>
<fieldset>
    <legend>Ficha</legend>

    <div class="display-label">Titulo</div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Titulo)
    </div>

    <div class="display-label">NoEtos</div>
    <div class="display-field">
        @Html.DisplayFor(model => model.NoEtos)
    </div>
    . . . . . .

</fieldset>
@using (Html.BeginForm()) {
    <p>
        <input type="submit" value="Delete" /> |
        @Html.ActionLink("Back to List", "Index")
    </p>
```

}

Figura 40: Código autogenerado para la Vista Ficha/Delete/1

Todas las vistas con sus respectivas etiquetas HTML de los demás controladores se autogeneran de manera similar a los pasos explicados anteriormente.

3.7 Modificación de las Vistas con jQuery UI

Agregando las bibliotecas jQuery v1.6.2 y jQuery UI 1.8.11

En la carpeta Scripts click derecho →Add → Existing Item.. → buscar en el explorador de Windows la carpeta donde está el archivo bajado de desde el sitio Web oficial de jQuery [jQuery] y escoger el archivo jquery-1.6.2.min.js (para producción).

La figura 41 nos muestra este proceso.

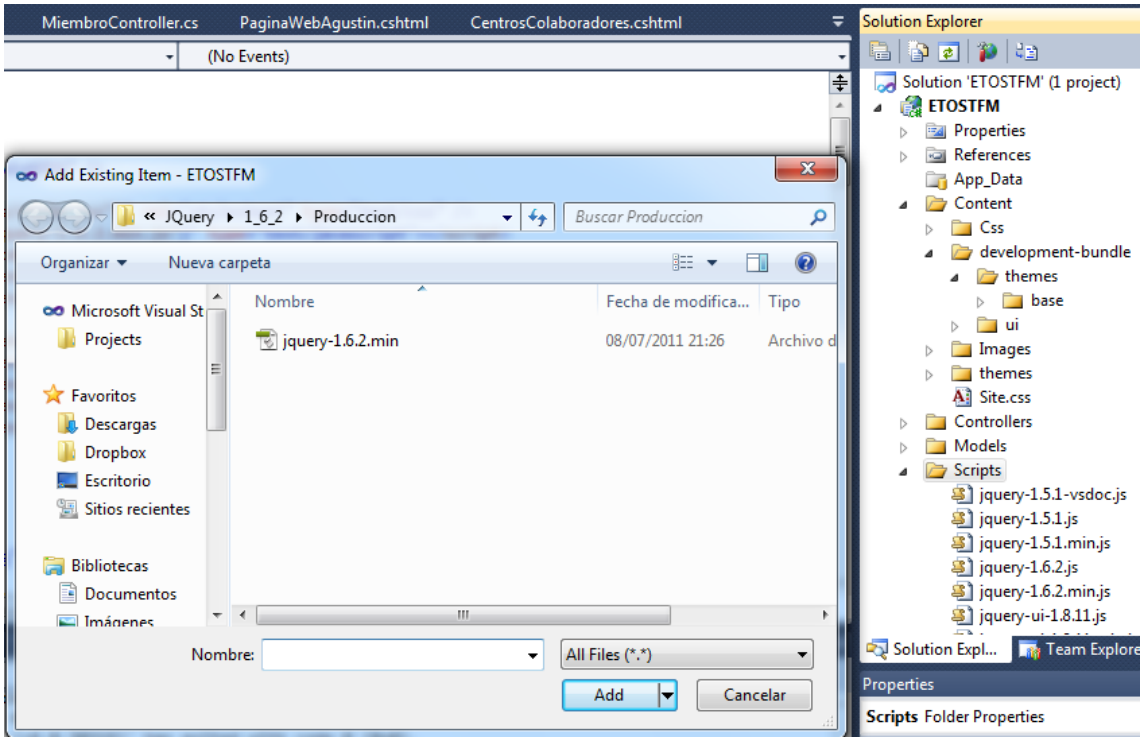


Figura 41: Agregar la jquery-1.6.2.min.js

Se crea en la carpeta Content la subcarpeta development-bundle y dentro de ellas las subcarpetas themes e ui, como muestra la figura 42.

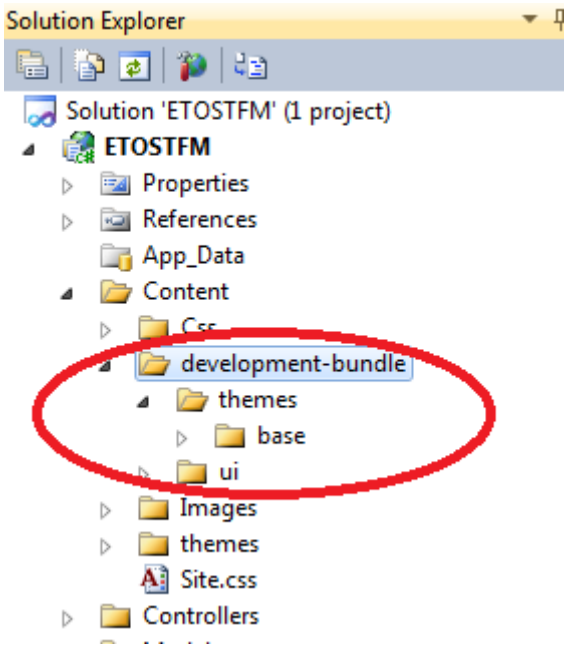


Figura 42: Creación de las carpetas development-bundle y subcarpetas themes e ui.

Se Agrega en cada subcarpeta los elementos existente. Este paso es muy similar al paso anterior, solo que en este caso se tiene que bajar desde el sitio Web oficial de jQuery UI [jQueryUI]. La figura 43 detalla este proceso.

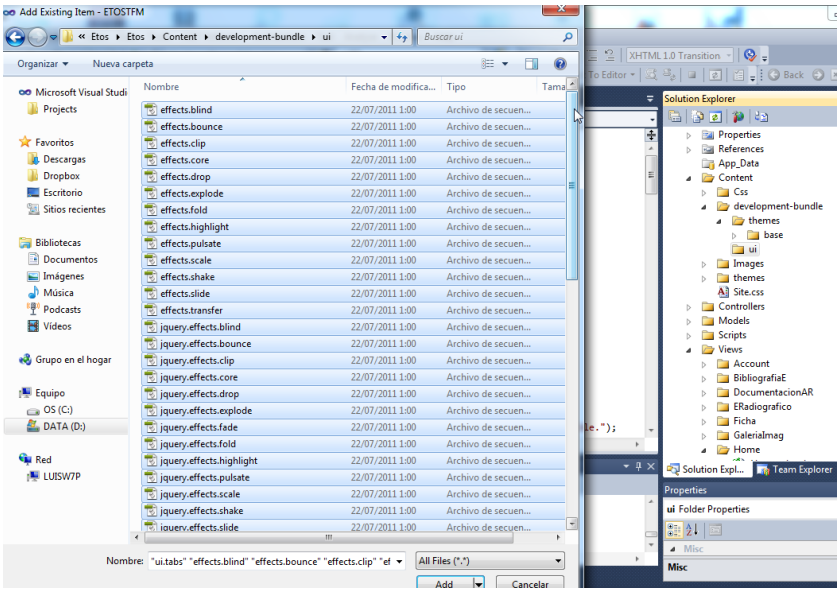


Figura 43: Instalación de las bibliotecas de jQuery UI 1.8.11

Creación de acordeones Ficha y Técnicas Artísticas:

Para la creación del acordeón Ficha de la columna izquierda de la página maestra, he agregado el componente Accordion de jQuery UI [DanWellman,2011], para ello he añadido la biblioteca jquery.ui.accordion.js en la cabecera de esta página. También he agregado el script necesario para poder activar este acordeón. La Figuras 44 y 45 se ven con más detalle lo descrito anteriormente.

```
……
        <script src="@Url.Content("~/Content/development-bundle/ui/jquery.ui.tabs.js")"
        type="text/javascript"></script>
           <script src="@Url.Content("~/Content/development-bundle/ui/jquery.ui.accordion.js")"
        type="text/javascript"></script>

           <!-- Para el acordeón de la izquierda:Ficha -->
            <script>
               $(function () {

                  //defino las opciones del acordeón
                  var accOpts = {
                     active: false,
                     collapsible: true
                     //event: "mouseover"
                  };
                  $("#accordion, #accordion2").accordion(accOpts);
               });
        </script>
```

Figura 44: Agregando jquery.ui.accordion.js y el script en la página maestra

```
……
        <div class="izq">
                    <div class="izq-box">
                    <div id="accordion">
                           <h2><a href="#">Fichas</a></h2>
                    <div>
                       <!--Acordeón para mostrar las fichas-->
                      @{Html.RenderAction("MenuFichas", "Ficha");}
                    </div>
                    </div>
                    </div>

        …..
        </div >
```

Figura 45: Creación del elemento div para el acordeón

La figura 46 muestra esta Vista parcial MenuFicha con el acordeón

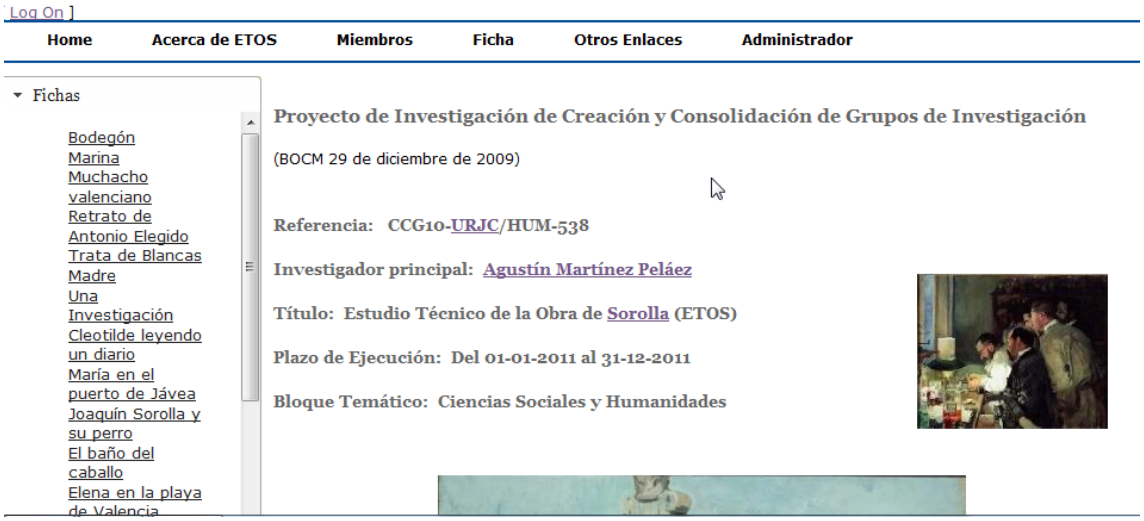


Figura 46: Vista parcial MenuFicha con la biblioteca accordion.js de jQuery UI

Cuando hacemos click en un elemento de este acordeón nos muestra en la parte central un Widget Tab de jQuery UI con los distintos estudios de esa ficha seleccionada.
La Figura 47 se puede ver en más detalle.

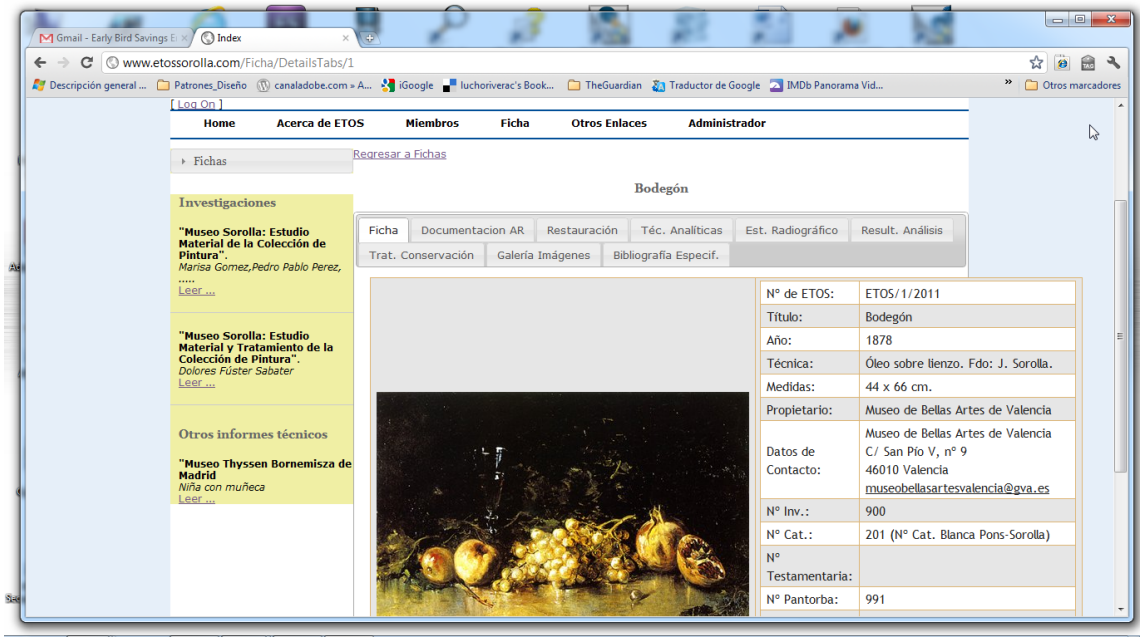


Figura 47: Vista mostrada al seleccionar una ficha del acordeón

El acordeón de Técnicas Artísticas se construye de manera similar al anterior

Creación de la Vista Ficha:

Otra manera de acceder a cada una de las fichas de ETOS, es creando una Vista llamada Obras, para ello se agrega una nueva acción en el controlador Ficha. La figura 48 muestra esta acción.

```
//Para la vista Obras
      public ViewResult Obras()
      {
        return View(db.Ficha.ToList());
      }
```

Figura 48: Acción Obras agregada al controlador Ficha.

La Vista para esta acción se muestra en la figura 49

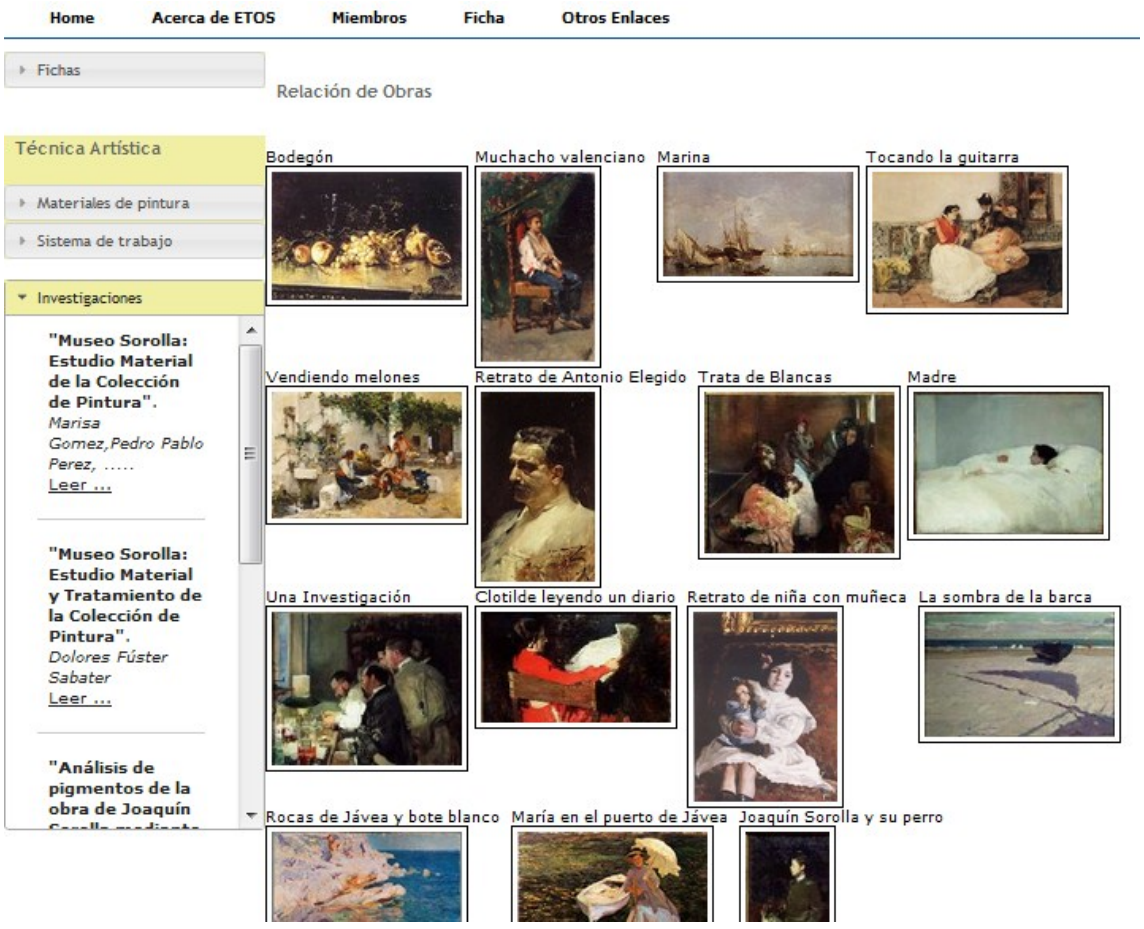


Figura 49: Vista Obras al hacer click en el menú Ficha

Cuando hacemos click en alguna de las Obras se muestra la vista de la figura 50.

Creación de la Vista Ficha Maestra

Se ha modificado la vista Ficha\Details para que sirva de página maestra para el acceso a los diferentes estudios de cada una de las obras. La figura 51 muestra esta vista.

La Figura 51 muestra el estudio "Resultado de Análisis" y la Figura 52 muestra el estudio "Tratamiento de Conservación" en detalle.

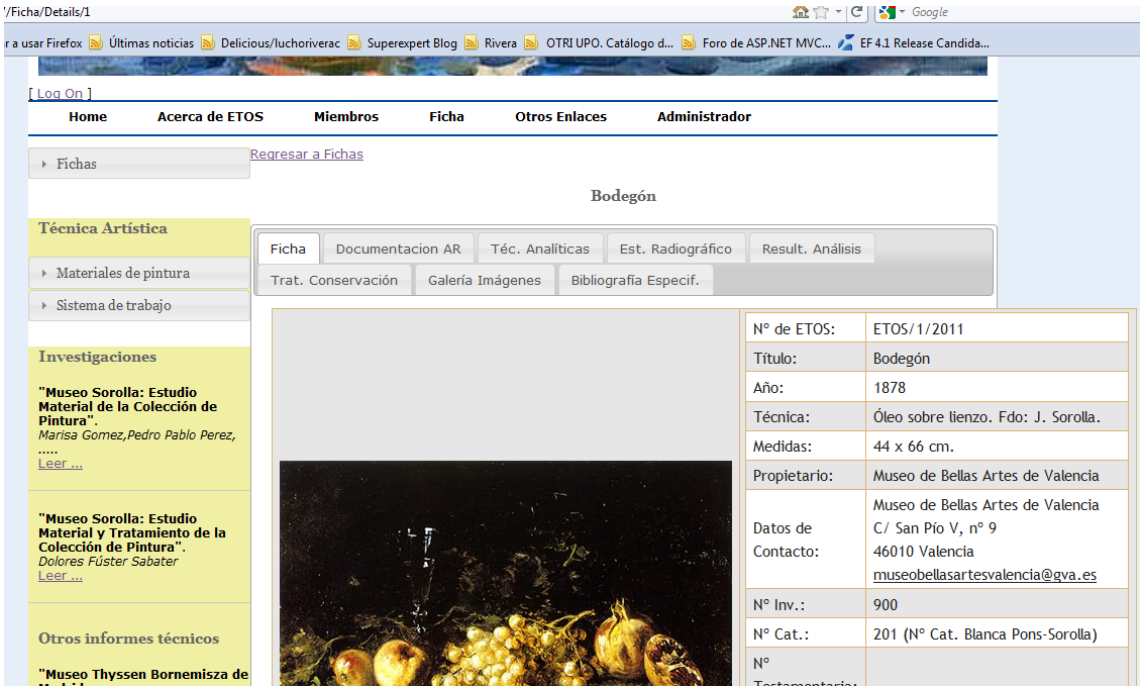


Figura 50: Vista Ficha Details como página maestra.

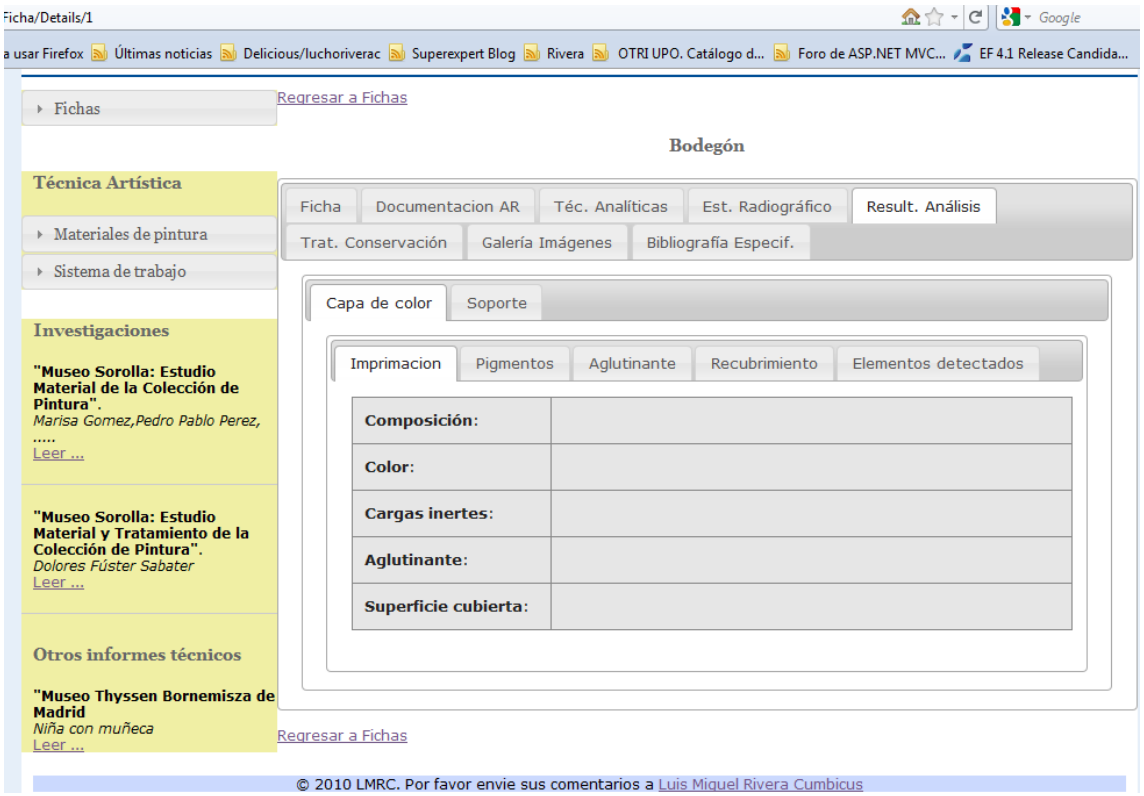


Figura 51: Vista Ficha con el estudio "Resultado de Análisis" activado

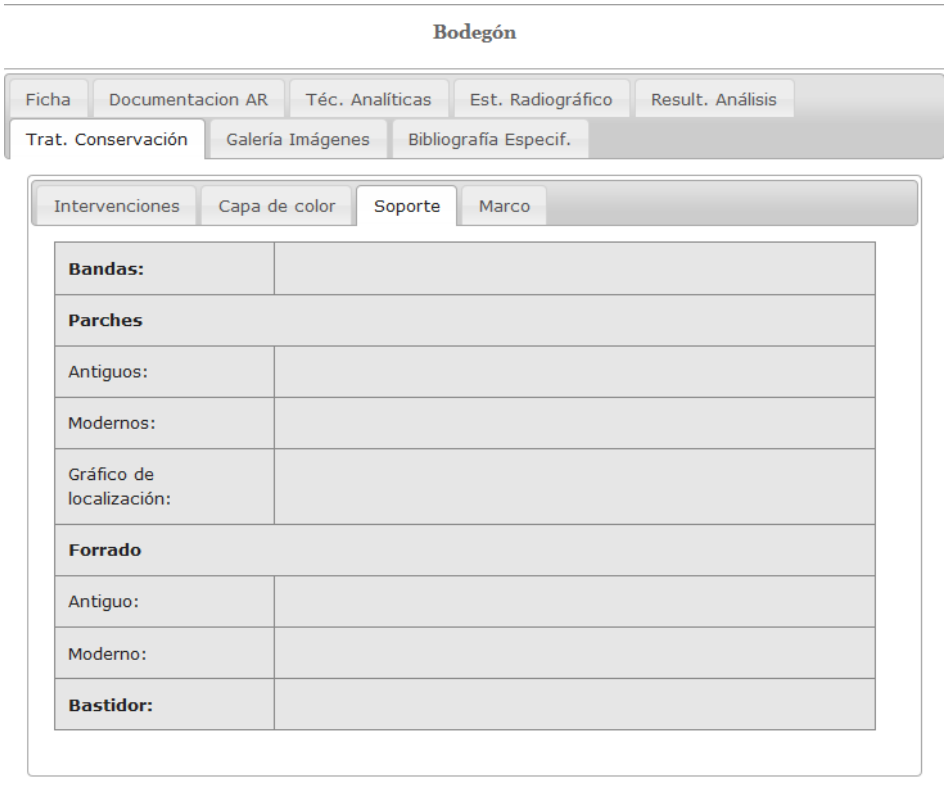


Figura 52: Estudio "Tratamiento de Conservación"

Para realizar el cambio de las tres vistas anteriores, se muestra en la figuras 53 siguientes los cambios realizados:

Modificación de la Vista Ficha\Details:

```
@model ETOSTFM.Models.Ficha


*@ Fin de Código Comentado, creado automáticamente *@

@{
    ViewBag.Title = "Lista de fichas";
    Layout = "~/Views/Shared/_Layout.cshtml";

*@script creado para el componente tabs de jQuery*@

            <script>
                $(function () {
                    $("#tabs").tabs({
                        collapsible: true,
                        //event: "mouseover",
                        ajaxOptions: {
                            error: function (xhr, status, index, anchor) {
                                $(anchor.hash).html(
                                                                "No se puede cargar esta página, vamos a
tratar de solucionar este problema lo más pronto posible.");
                            }
                        },
                        cookie: {
                            // store cookie for a day, without, it would be a session cookie
                            expires: 30
                        }

                    }).find(".ui-tabs-nav").sortable({ axis: "x" }); *@para arrastar los tabs*@
                });
            </script>

*@Estilo para la Tabla *@
    <style type="text/css">
        .tabs
        {
            /*width:600px;*/
            text-align:center;

        }
        #fichaTabla {
            font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;
            width:660px;
            height:auto;
            border-collapse: collapse;
            }

            #fichaTabla td {
                                font-size: 1.1em; /* 1.0em significa: 1.0 veces el tamaño de la letra
                    actual q se está usando */
                border: 1px solid #DDB575;
                padding: 3px 7px 2px 7px;
            }
```

```
        #fichaTabla tr.alt td {
        background-color: #FFF;
        }
        .datos
        {
           width:380px;
        }

        .ranalisis{
           width:640px;  /* ancho de las tablas de Resultado de análisis*/
        }
        .pigmentos{
           width:150px;  /* ancho de las etiquetas de las tablas de resultado de análisis*/
        }

        /* Para la altura de los div del acordeón */
        #myAccordion div {
           height:600px;
        }

    </style>

}
   @Html.ActionLink("Regresar a Fichas", "Obras", "Ficha")
   <h2 style="text-align:center;">@Model.Titulo</h2>

<div id="tabs" class="tabs">
          <ul>
                    <li><a href="#tabs-1">Ficha</a></li>
      <li><a href="@Url.Action("DocumentacionARFicha", "DocumentacionAR", new { id =
@Model.FichaId })">Documentacion AR</a>
      </li>
      <li><a href="@Url.Action("TAnaliticasFicha", "TAnaliticas", new { id = @Model.FichaId
})">Téc. Analíticas</a>
      </li>
      <li><a href="@Url.Action("ERadiograficoFicha", "ERadiografico", new { id =
@Model.FichaId })">Est. Radiográfico</a>
      </li>
      <li><a href="@Url.Action("RAnalisisFicha", "RAnalisis", new { id = @Model.FichaId
})">Result. Análisis</a>
      </li>
      <li><a href="@Url.Action("TConservacionFicha", "TConservacion", new { id =
@Model.FichaId })">Trat. Conservación</a>
      </li>
      <li><a href="@Url.Action("GaleriaImagFicha", "GaleriaImag", new { id =
@Model.FichaId })">Galería Imágenes</a>
      </li>
       <li><a href="@Url.Action("BibliografiaEFicha", "BibliografiaE", new { id =
@Model.FichaId })">Bibliografía Especif.</a>
      </li>
          </ul>

          <div id="tabs-1">
          <table id="fichaTabla">
           <tr>
              <td style="width:300px;">
                                  <img src="@Model.Imagen2" alt="@Model.Titulo"
                                  title="@Model.Titulo" />
```

```
                </td>

                <td style="padding-left:5px; border:solid 1px #DDB575;">
                  <table class="datos">
                    <tr class="alt">
                      <td>Nº de ETOS:</td>
                      <td>@Model.NoEtos</td>
                    </tr>
                    <tr>
                      <td>Título:</td>
                      <td>@Model.Titulo</td>
                    </tr>
                    <tr class="alt">
                      <td>Año:</td>
                      <td>@Model.FObra</td>
                    </tr>
                    <tr>
                      <td>Técnica:</td>
                      <td>@Model.Tecnica</td>
                    </tr>
                    <tr class="alt">
                      <td>Medidas:</td>
                      <td>@Model.Medidas</td>
                    </tr>
                    <tr>
                      <td>Propietario:</td>
                      <td>@Model.Propietario</td>
                    </tr>
                    <tr class="alt">
                      <td>Datos de Contacto:</td>
                      <td>@Model.Contacto<br />
                        @Model.DContacto1<br />
                        @Model.DContacto2<br />
                       <a href="mailto:@Model.EmailContacto">
                                          @Model.EmailContacto
                                          </a>

                       </td>
                    </tr>
                    . . . .
                    <tr class="alt">
                      <td>Historial:</td>
                      <td>@Model.Historial</td>
                    </tr>

                  </table>
                </td>
              </tr> <!-- fin 1er miembro-->
            </table>
          </div>
        </div>
        <br />
          @Html.ActionLink("Regresar a Fichas", "Obras", "Ficha")
        <br />
```

Figura 53: Modificación de la Vista Ficha\Details

Modificando la Vista Miembros

El menú Miembros tiene cuatro vistas:

- Investigador principal
- Investigadores
- Colaboradores
- Centros Colaboradores

La vista creada automáticamente muestra en una tabla un listado de todos los miembros con todos los campos que la componen. Esta lista la tengo que modificar para que me muestre tres vistas adicionales por categorías: Investigador principal, Colaborador, Centros Colaboradores. En la Figura 54 sde puede apreciar el código para crear estas vistas.

En la carpeta Controller/MiembroController, agrego las siguientes acciones para que me muestre las cuatro vistas que necesito mostrar:

```
        //Para la Vista Investigador Principal
        public ViewResult InvestigadorPrincipal()
        {
            return View(db.Miembro.Where(m => m.MCampo2 == "Investigador
principal").ToList());
        }

        //Para la Vista Investigadores
        public ViewResult Investigadores()
        {
            return View(db.Miembro.Where(m => m.MCampo2 == "Investigadores").ToList());
        }

        //Para la Vista Colaboradores
        public ViewResult Colaboradores()
        {
            return View(db.Miembro.Where(m => m.MCampo2 == "Colaboradores").ToList());
        }

        //Para la Vista Centros Colaboradores
        public ViewResult CentrosColaboradores()
        {
            return View(db.Miembro.Where(m => m.MCampo2 ==
            "CentrosColaboradores").ToList());
        }
```

Figura 54: Creando Vistas por tipo de Investigador

Para crear la vista de Investigador, hacemos click derecho en InvestigadorPrincipal →

Add →Add View….

La figura 55 nos muestra este proceso

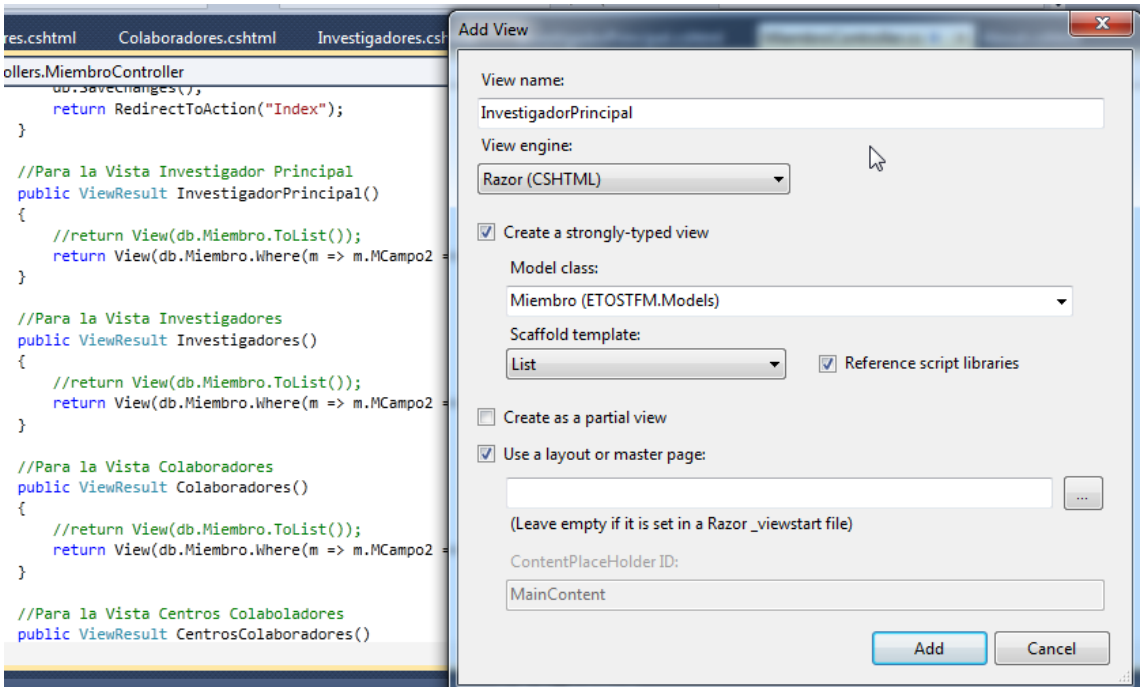


Figura 55: Agregar una nueva vista desde el Controlador

La nueva vista automáticamente generada, según podemos apreciar en la figura 56

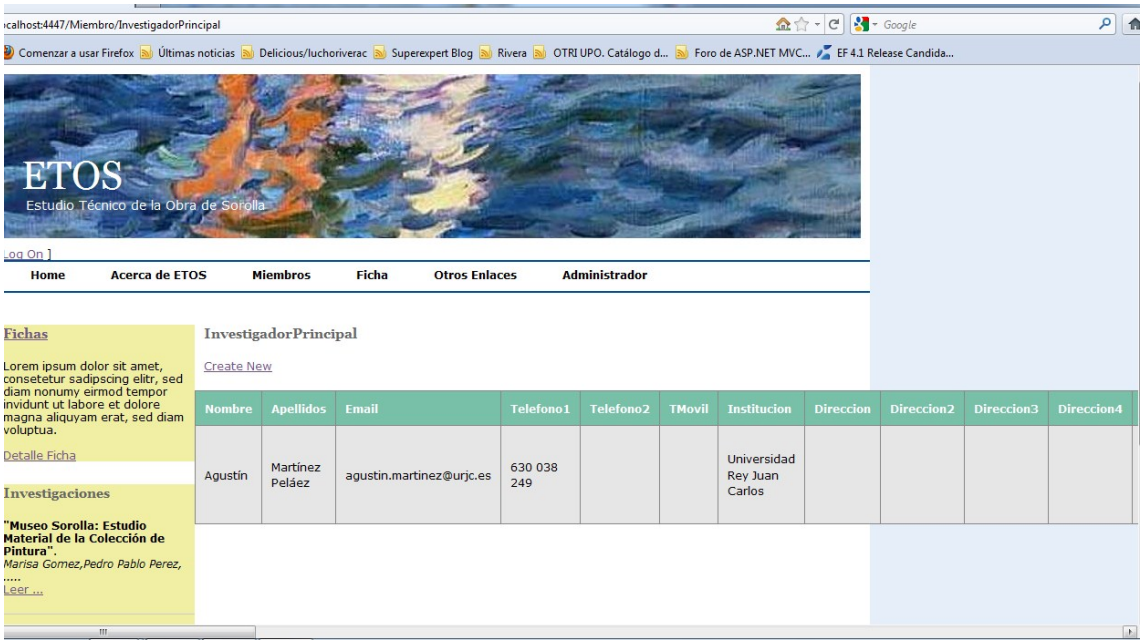


Figura 56: Nueva Vista de Investigador Principal autogenerada

Para cambiar la Vista Investigador Principal autogenerada, elimino el código HTML y agrego el siguiente código. La Figura 57 muestra el nuevo código agregado.

```
@{
    ViewBag.Title = "ListaTabla";
   <style type="text/css">
        #inventory {
            width:400px;
            margin-top: 12px;
            margin-left:auto;
            margin-right:auto;
        }
        #inventory_datos
        {
           width:300px;
        }
         #inventory td
         {
                    font-size: 1.0em; /* 1.0em significa: 1.0 veces el tamaño de la letra actual q se
está usando */
                    border: 1px solid #DDB575;
            padding: 3px 7px 2px 7px;
         }
         #inventory tr.alt td {
           background-color: #FFF;
         }
</style>
}
```

```
<table id="inventory">
@foreach (var item in Model)
{
   <tr>
      <td>
        <a href="@Url.Action("DetailsTab", "Miembro", new { id = item.MiembroId })">
         <img alt="@item.Nombre" src="@item.Imagen1" title="@item.Nombre
@item.Apellidos" />
        </a>
      </td>
      <td style="padding-left:15px; border:solid 1px #DDB575;">
                  <table id="inventory_datos">
                     <tr class="alt">
                        <td> @Html.DisplayFor(modelItem => item.Nombre)</td>
                     </tr>
                     <tr>
                        <td>@Html.DisplayFor(modelItem => item.Apellidos)</td>
                     </tr>
                      <tr class="alt">
                        <td>@Html.DisplayFor(modelItem => item.Institucion)</td>
                     </tr>
                     <tr class="alt">
                        <td><a href="mailto:@Html.DisplayFor(modelItem =>
item.Email)">@Html.DisplayFor(modelItem => item.Email)</a></td>
                     </tr>
                     <tr>
                        <td><a href="@Html.DisplayFor(modelItem => item.MCampo3)">Página
Web personal</a></td>
                     </tr>
                     <tr>
                        <td>
                              @Html.ActionLink("Ver más información....", "DetailsTab", new { id
= item.MiembroId })
                         </td>
                      </tr>
                  </table>
      </td>  <!-- fin 1er miembro-->
   </tr>
}
</table>
```

Figura 57: Nuevo código agregado para la Vista Investigador Principal

En la siguiente figura 58 se muestra la nueva Vista Investigador Principal.

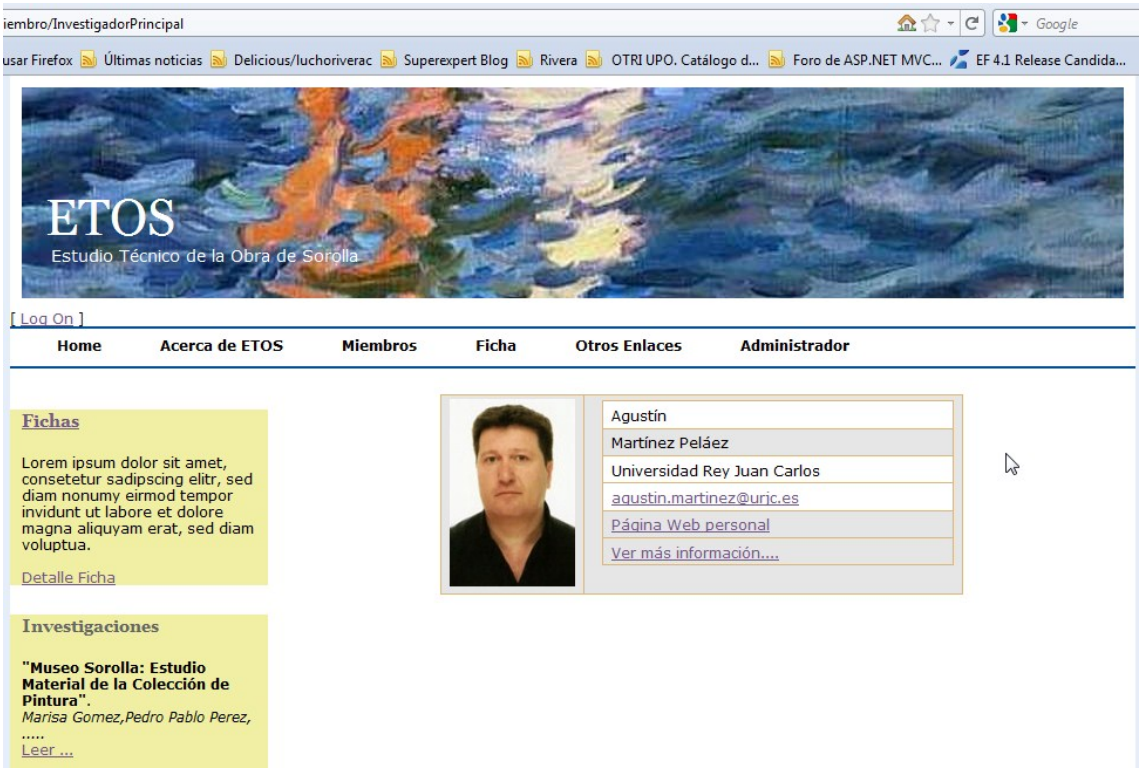


Figura 58: Nueva Vista Investigador Principal modificada

Todas las demás vista del menú Miembros se modifican de forma similar a la del Investigador Principal.

Creación de las Páginas Web Personales:

Para la creación de la página Web personal del Investigador Principal voy a hacer uso de la ayuda de las bibliotecas de jQuery UI, en este caso voy a utilizar el Widget **Tabs.**

La Figura 59 nos muestra esta página.

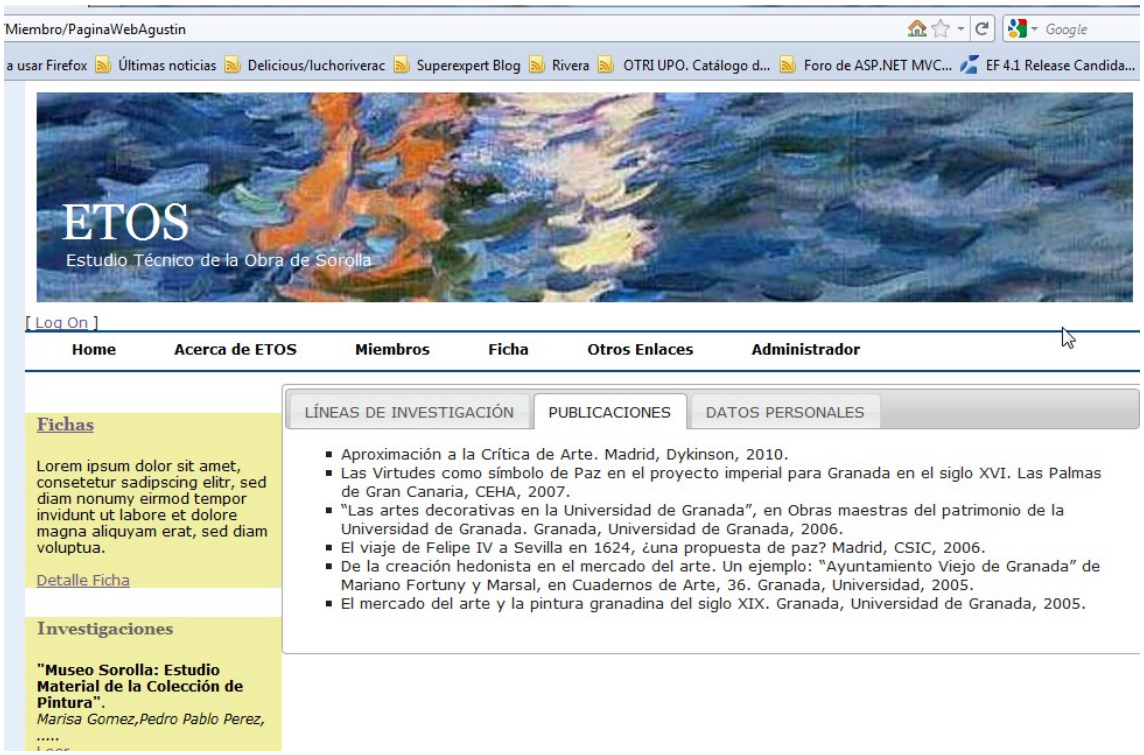


Figura 59: Creacion de páginas personales

Como se puede apreciar, esta vista es de forma tabular, haciendo click en su tab respectivo se puede ocultar su contenido. También se puede arrastrar los tabs.

En esta modificación también he agregado un script: jquery.cookie.js, creación externa que no pertenece a la biblioteca jQuery UI, pero que es open source y se puede bajar de [jQueryCookie], este script mantiene el estado del último tab abierto.

3.8 Aplicación de la metodología "Responsive Web Design"

Se debe de resaltar que la aplicación se había creado solo para ser vista en ordenadores de escritorio y en portaliles mas no así para dispositivos como smartphone, tables, etc.

Con el fin de poder aplicar la metodología del Responsive Web Design (RWD) y que se pueda adapta al dispositivo en la que va a ser visulaizado, se ha tenido que realizar ciertos cambios en la estructura y en la hoja de estilo de la aplicación.

Para adaptar esta aplicación a RWD, se ha tenido que aplicar la siguiente formula [Frain,2012] explica en la sección 2.23 :

target / context = result

Los cambios que se realizaron a nuestra aplicación, fueron los siguientes:

- En la plantilla principal _Layout.cshtml, se agregó la siguiente línea:

```
<meta name="viewport" content="width=device-width,
    initial-scale=1.0,maximum-scale=1" />
```

- Se modificó la hoja de estilo style.css en los siguientes elementos;

```
#container {
…
width:22.9166666666%; /*260/960*/
min-width:20%;
……
}
……

#navTop
        {
            …….
            height: 3.64583333%; /* 35/960 */
            /*width: 960px;*/
            width: 96%;
        }
……….

#container .izq {
       ……..
        width:22.9166666666% /* 260/960  */

       }
…….
#container #content {
        /* width:740px;  /*690*/
       width:77.083333%;  /* 740/960 */
       ……….
       }
       ……
fieldset {
       width: 57.291666666px;  /*550/960*/
       }
       ……

/********************
*   Mobile Styles   *
********************/
/*

@media screen and (min-width:480px) and (max-width:1024px)
{
       body {
              min-width: 480px;
              max-width: 800px;
       }
```

```
   img {
          width: 500px;
          height: 375px;
          align: center;
   }

   ul.thumbnails li {
   width: 515px;
   height: 390px;
   line-height: 200px;
   }
   }*/
   @media only screen and (max-width: 850px) {

       /* header
       ------------------------------
       */
       header .float-left,
       header .float-right {
           float: none;
       }

       --------------------------------

       /* menu */
       nav {
           margin-bottom: 5px;
       }

      #navTop #dropdown li ul li a {
           margin: 0;
           padding:0;
           text-align: center;
       }
      -----

/* main layout
-----------------------------------------*/
---------
#container {
    padding-right: 10px;
    padding-left: 10px;
}

/* page content */
#container #content {
    float: none;
    width: 100%;
}
#container .izq {
    clear:left;
   /*width:220px;*/
   /*border-top: 2px solid #004e97; /**/
   width:22.9166666666%; /*260/960*/
   min-width:150px;
}
 ------

/* forms */
fieldset input[type="text"],
fieldset input[type="password"] {
```

```
    width: 90%;
}

/* login page */
#loginForm {
    border-right: none;
    float: none;
    /*width: auto;*/
    width: 30%;
}
```

Las siguientes figuras 60 al 63 muestran las visualizaciones de este sitio Web a través del Emuladador de Windows Phone y con ventanas del navegador Firefox simulando distintos tamaños de ventanas.

Figura 60: Visualizacion con el Emulador de Windows Phone aplicando "viewport"

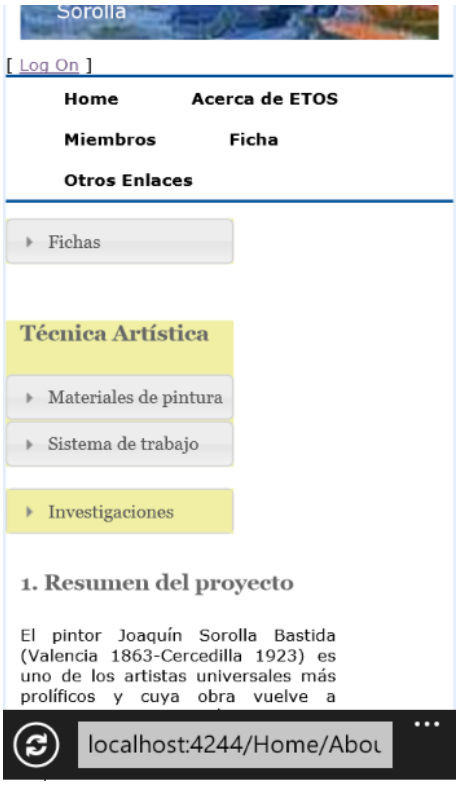


Figura 61: Visualizacion con WP aplicando RWD

Figura 62: Visualizacion WP emulando un tablet

Figura 63: Visualización con Firefox con ventana 441x768

CAPÍTULO 4: CONCLUSIONES Y TRABAJOS FUTUROS

4.1 Conclusiones

El principal objetivo de este trabajo ha sido demostrar la generación automática de aplicaciones Web basadas en datos, aprovechando los patrones ASP.NET MVC 3 y Code First de Entity Framework 4.1 y junto con las bibliotecas jQuery UI.

Después de haber realizado la parte práctica de la aplicación del sitio Web del grupo de investigación he llegado a las siguientes conclusiones:

1. Se pueden diseñar y desarrollar aplicaciones Web Empresariales apoyándonos en patrones que son ampliamente utilizados el desarrollo del software:
   El patrón estructural MVC se apoya en los Patrones de Diseño (Observer, Strategy, Composity, Factory, Decorator, Proxy, etc).
   El patrón Code First de EF 4.1 utiliza los Patrones de Arquitectura de Aplicaciones Empresariales (PEAA) (Domain Model, Unit of Work, Lazy Load, etc).
2. Estos patrones nos permiten desarrollar todo tipo de aplicaciones Web de alta calidad, fácil mantenimiento ahorrando muchas horas de desarrollo y ganado con ello más productividad.
3. Las bibliotecas de jQuery UI nos permiten ahorrar muchísimas horas de trabajo en el desarrollo de las interfaces de usuario, además que estas interfaces con más dinámicas, interactivas y amigables.
4. Esta aplicación de ejemplo es solo un ejemplo de la utilización de estos patrones, se puede utilizar en todo tipo de aplicaciones Web, desde la más sencillas hasta las más complejas.
5. Tal como he demostrado en este trabajo, desarrollar aplicaciones Web con base de datos manera automática, no es necesario saber sofisticados programas de desarrollo de la Ingeniería del Software o de la Ingeniería Web, si podemos aprovechar los nuevos paradigmas que nos ofrecen desarrollara casi de manera rápida, eficiente y segura estas aplicaciones. Tenemos que aprovecharnos de la reutilización del software que ya han sido probar en muchas aplicaciones

teniendo como base los unos de los principios del concepto de los patrones: **Transmitir el Conocimiento**

6. Con la aplicación de la nueva metodología de Responsive Web Design, se diseñan y creaan aplicaciones Web flexibles que se adaptan a los dispositivos, ventanas, orientaciones en las que son visualizadas.

4.2 Trabajos futuros

Dentro de los trabajos futuros a realizar, me gustaría mencionar los siguientes:

1. Desarrollar esta misma aplicación para un dispositivo móvil.
2. Desarrollar este prototipo en Java 2EE.
3. Desarrollar este prototipo de PHP con MySQL
4. Aplicar más la biblioteca jQuery UI para que el usuario interactúe creando sus propias vistas.
5. Desarrollar toda la aplicación Web con HTML 5 y CSS3, para asi aplicar Responsive Web Design en toda su dimensión.

BIBLIOGRAFIA

[Alexander,1979]	Alexander, Christopher. The Timeless Way of Building. Oxford University Press. 1979.
[BMMM,1998]	Brown, William; Raphael Malveau; Hays Mc Cormick III; Tom Mowbray. Antipatterns: Refactoring Software, Architectures and Project in Crisis. Wiley and Sons. 1998.
[DanWellman,2011]	Wellman,Dan, jQuery UI 1.8 : The User Interface Library for jQuery, Agosto 2011
[DeLaTorre,2010]	De la Torre LL,Cesar;Zorrila C.,Unai;Calvario N.,Javier;Ramos B.,Miguel: Guia de Arquitectura N-Capas otrientada al Dominio con .NET 4.0 Noviembre 2010
[DotNetMania,76/2010]	Programación con jQuery en Visual Studio 2010
[Fowler,1996]	Fowler, Martin. Analysis Patterns: Reusable Object Models. Adisson-Wesley Professional. 1996.
[Fowler,2002]	Fowler, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley Professional. 2002.
[Frain,2012]	Frain, Ben, Responsive Web Design with HTML 5 and CSS3, Packt Publishing, 11-April-2012
[GoF,1995]	Gamma, Erich; Richard Helm, Ralph Johnson, John Vlissides. Design Patters: Elements of Reusable Object Oriented Software. Addison Wesley. 1995.
[Lindley,2010]	Libdley Cody. jQuery Coolbook. O'Reilly 2010
[PedroJMolina,2003]	Tesis Doctoral: Especificación de interfaz de usuario: De los requisitos a la generación automática
[POSA,1996]	Buschmann, Frank; Regine Meunier; Hans Rohnert; Peter Sommerland; Michael Stal. Pattern Oriented Software Architecture, Volume 1: A System of Patterns. Wiley & Sons. 1996.
[Welicki,2006]	Tesis Doctoral: Meta-Especificación y Catalogación de Patrones de Software con Lenguajes de Dominio Específico y Módulos de Objetos Adaptatativos: Una Vía para la Gestión del Conocimiento en la Ingeniería del Software.

DIRECCIONES DE INTERNET

[15PreguntasMVC]	http://www.variablenotfound.com/2010/05/aspnet-mvc-2-quince-cuestiones-que.html
[AprendaMVCin7Dias]	http://www.codeproject.com/KB/aspnet/LearnMVC.aspx
[ASPNETMVC]	http://www.asp.net/mvc/mvc3
[BlogEFDesign]	http://blogs.msdn.com/b/efdesign/
[BlogScottGuthrie]	http://weblogs.asp.net/scottgu/archive/2010/07/16/code-first-development-with-entity-framework-4.aspx
[CodeFirstConvenciones]	http://blogs.msdn.com/b/efdesign/archive/2010/06/01/conventions-for-code-first.aspx
[CodeProjectEFProvider]	http://www.codeproject.com/KB/aspnet/AspNetEFProviders.aspx?display=Print
[ContosUniversityMVC]	http://www.asp.net/entity-framework/tutorials/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application
[ConvencionesCoC]	http://es.wikipedia.org/wiki/Convenci%C3%B3n_sobre_Configuraci%C3%B3n
[DateTDecimalDouble]	http://www.variablenotfound.com/2010/05/bug-sin-importancia-en-la-plantilla-de.html
[DesarrolloWeb]	http://www.desarrolloweb.com/manuales/manual-jquery.html
[DinoEsposito]	http://msdn.microsoft.com/es-es/magazine/dd942833.aspx
[EjecucionMDSN]	http://msdn.microsoft.com/es-es/library/dd381612%28v=VS.98%29.aspx
[Enrutamiento]	http://msdn.microsoft.com/es-es/library/cc668201.aspx
[ETOS]	http://www.etossorolla.com
[jQuery]	http://jquery.com/

[jQueryCookie]	http://plugins.jquery.com/project/Cookie
[jQueryDatepicker]	http://www.variablenotfound.com/2010/06/edicion-elegante-de-fechas-con-jquery.html
[jQueryUI]	http://jqueryui.com/ http://docs.jquery.com/UI
[LibroGratis]	http://nerddinnerbook.s3.amazonaws.com/Intro.htm
[MarcotteEthan]	http://www.alistapart.com/articles/responsive-web-design/
[MartinFowler]	http://martinfowler.com/eaaDev/uiArchs.html
[McCormick98]	McCormick, Hays. *Antipatterns Tutorial*. 1998. http://www.antipatterns.com/briefing/sld001.htm
[MDigitales1]	http://maromasdigitales.net/2011/03/desarrollando-con-codigo-primero-en-entity-framework-4/
[MDigitales2]	http://maromasdigitales.net/2011/04/correspondencias-personalizadas-con-codigo-primero-en-entity-framework-4/
[MDigitales3]	http://maromasdigitales.net/2011/04/usando-codigo-primero-de-ef-con-una-base-de-datos-ya-existente/
[MDigitales4_MusicSt]	http://maromasdigitales.net/2011/05/generando-un-modelo-de-clases-en-ef-a-partir-de-una-base-datos-usando-codigo-primero/
[MDigitalesCodeFirst]	http://maromasdigitales.net/2011/05/generando-un-modelo-de-clases-en-ef-a-partir-de-una-base-datos-usando-codigo-primero/
[MDigitalesEF]	http://maromasdigitales.net/2011/03/desarrollando-con-codigo-primero-en-entity-framework-4/
[MDSNDataEF]	http://msdn.microsoft.com/es-es/data/ef
[MDSNDataEF4_1]	http://msdn.microsoft.com/es-es/data/hh134816
[MediaQueries]	http://www.w3.org/TR/css3-mediaqueries/

[MSDN]	http://msdn.microsoft.com/es-es/library/dd381412%28v=VS.98%29.aspx
[MSDNBlog]	http://blogs.msdn.com/b/adonet/archive/2011/03/15/ef-4-1-code-first-walkthrough.aspx
[MSDNEF]	http://msdn.microsoft.com/es-es/library/bb399572.aspx
[MSDNLibraryPOCO]	http://msdn.microsoft.com/es-es/library/dd456853.aspx
[MsdnMagazine0308]	http://msdn.microsoft.com/es-es/magazine/cc337884.aspx?pr=blog
[MSDNT4]	http://msdn.microsoft.com/en-us/gg558520
[MVC4]	http://www.asp.net/mvc/mvc4
[Novedades MVC4]	http://www.variablenotfound.com/2011/09/aspnet-mvc-4-developer-preview-un.html#more
[PantillasT4]	http://msdn.microsoft.com/es-es/library/bb126445.aspx
[RowanMiller]	http://msdn.microsoft.com/es-es/magazine/hh126815.aspx
[ScottGuthrie_jQuery]	http://weblogs.asp.net/scottgu/archive/2008/09/28/jquery-and-microsoft.aspx
[Trygve]	http://heim.ifi.uio.no/~trygver/
[TutorialMusicStore]	http://www.asp.net/mvc/tutorials/mvc-music-store-part-1
[VideosTutoriales]	http://www.dotnetfunda.com/articles/article1297-how-to-create-a-simple-hello-world-aspnet-mvc-tutorial-no-1-.aspx
[w3schools]	http://www.w3schools.com/jquery/default.asp
[WebETOS]	http://www.etossorolla.com
[WebPlatformInstaller]	http://www.microsoft.com/web/gallery/install.aspx?appid=VWD2010SP1Pack
[Wikipedia]	http://es.wikipedia.org/wiki/JQuery

[WikipediaSorolla]	http://es.wikipedia.org/wiki/Joaqu%C3%ADn_Sorolla
[WikipediaUI]	http://es.wikipedia.org/wiki/JQuery_UI
[Winhost]	http://www.winhost.com/