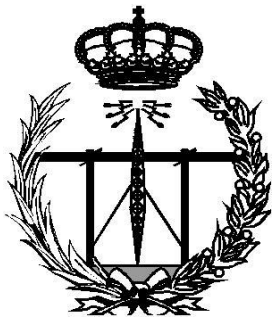
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Carrera

Gestión de datos para aplicaciones generales usando tecnologías .NET y metodologías de tipo 'Design Pattern' (General data management applications using .NET technologies and 'Design Pattern' methodologies)

Para acceder al Titulo de

Ingeniero de Telecomunicación

Autor: Pedro Bezanilla Torre

Septiembre - 2015

INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE CARRERA

Realizado por: Pedro Bezanilla Torre

Director del TFG: Tomás Fernandez Ibañez

Título: "Gestión de datos para aplicaciones generales usando tecnologías .NET y metodologías de tipo 'Design Pattern'"

Title: "General data management applications using .NET technologies and 'Design Pattern' methodologies**"**

Palabras clave: Aplicaciones Full-Stack, Desarrollo Web, Testing, Buenas prácticas, Patrones de Diseño.

Presentado a examen el día: 29 de Septiembre de 2015

para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente: Alicia Casanueva Lopez
Secretario: Tomás Fernández Ibáñez
Vocal: José Angel García García

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente	Fdo.: El Secretario
Fdo.: El Vocal	Fdo.: El Director del TFG (sólo si es distinto del Secretario)
Vº Bº del Subdirector	Trabajo Fin de Grado Nº (a asignar por Secretaría)

Agradecimientos

Quiero agradecer a Tomás Fernandez Ibañez por su tiempo y dedicación tanto en este proyecto como a lo largo de la carrera. Aprovecho también para agradecer la amistad y humildad de mis antiguos compañeros Miroslav, Daniel, Oscar, Jaime, Marta, Mihail, y todos los que han hecho posible una buena convivencia en estos 5 últimos años.

Y por supuesto a mi padre, a mi fallecida abuela, a mi madrina y como no a mi querida Andressa. Gracias por 'soportarme' y por acompañarme en todos mis logros y fracasos.

Índice

1. Introducción

¿Qué es una aplicación web?

Según el Diccionario Oxford, una aplicación es un "programa o conjunto de programas que ayudan al usuario de un ordenador a procesar una tarea específica". Una aplicación web es básicamente una forma de facilitar el logro de una tarea específica. Por ejemplo, si queremos buscar un libro de la BUC (Biblioteca de la Universidad de Cantabria), es mucho más óptimo y productivo hacerlo mediante su aplicación Web (www.buc.unican.es → buscador 'ÚniCo') que hacerlo manualmente.

Un poco de historia

Las aplicaciones Web interactivas han ido revolucionando paulatinamente la forma de utilizar Internet pasando de tener páginas estáticas (con texto estático, páginas que muestran información permanente, HTML puro) a páginas dinámicas con contenidos interactivos, aplicaciones integradas como por ejemplo encuestas, pedidos on-line, foros de soporte, etc.

Uno de los primeros lenguajes de aplicaciones web fue el 'Perl', inventado en 1987 por Larry Wall antes de que Internet fuera accesible para el público en general. En 1991 nace la primera página web, que no era más que una pequeña referencia sobre qué era y qué se podía hacer en la World Wide Web. Pero fue a partir de 1995 cuando todo el desarrollo de aplicaciones web realmente despegó gracias al Ingeniero Rasmus Lerdorf, creador del lenguaje PHP y a que en este año se empezó a comercializar Internet. Actualmente muchas aplicaciones importantes están hechas en PHP en cuanto a BackEnd se refiere, por ejemplo Facebook, Google y Wikipedia.

Unos meses más tarde, Netscape, el navegador más antiguo y popular, anunció una nueva tecnología responsable de la transición de páginas estáticas a páginas dinámicas, el famoso JavaScript (FrontEnd). Por ejemplo en la página de Google, cuando se muestran los resultados de búsqueda a medida que escribimos, se hace un uso intensive de JavaScript que es el responsable de iteractuar con la parte BackEnd de la aplicación, y por tanto con el Servidor.

En 1996, tras la divulgación de Rasmus Lerdorf, dos desarrolladores, Sabeer Bhatia y Jack Smith lanzaron Hotmail, uno de los primeros SaaS (Software as Service) del Mercado. Luego, en 1997 vino la conocida plataforma Flash utilizada para añadir animaciones y objetos interactivos en sitios Web. Al año siguiente, en 1998 el sitio web "The Drudge Report" anunció por primera vez una serie de noticias antes de que se difundieran por los medios televisivos y prensa. Este evento fue el detonante del periodismo online tal como lo conocemos hoy en día. En este mismo año Google desarrolló su primer motor de búsqueda online de páginas web. Tres años después, en 2001, poco después de la explosion de la burbuja de Internet, surgió la conocida enciclopédia online Wikipedia. En 2002 aparece Fotolog (la primera red social con el concepto de album fotográfico), también surge Friendster, Lastfm, etc. En 2003 aparece MySpace, en 2004 Facebook, y más adelante Youtube, Twitter, Tuenti, SoundCloud, Spotify, etc.

Como se puede ver, a lo largo de estos 20 últimos años han ido apareciendo numerosos servicios a través de Internet que son basicamente un conjunto de programas responsables de facilitar al usuario ciertas tareas específicas.

En los 45 años de la historia de Internet, los programadores han tratado de hacer estallar las barreras entre aplicaciones tradicionales y aplicaciones web. El progreso de los últimos años en tecnología, velocidad de descarga, así como herramientas de desarrollo cada vez más avanzadas (.NET, Eclipse, Oracle Suite, Grunt, etc) han facilitado el avance de aplicaciones cada vez más potentes.

Es evidente que el impacto de las aplicaciones Web en la productividad, en cómo operar un negocio, transmitir o recibir información, e incluso en la vida cotidiana de las personas es

importante.

Algunos conceptos del paradigma de la programación

- Golden Hammer (Martillo de Oro): Es un antipatrón cuya definición tiene mucho que ver con la siguiente frase: “Cuando la única herramienta que tienes es un martillo todo problema comienza a parecerse a un clavo”. Un Martillo de Oro es por tanto cualquier herramienta, tecnología, paradigma o similar cuyos partidarios ensalzan de manera exagerada. Predicen que resolverá multiples problemas, incluso aquellos para los que obviamente no es adecuada. De la misma manera forma que un martillo de oro físico sería bastante impresionante, pero practicamente inútil, puesto que el oro es un metal relativamente maleable.

- Silver Bullet (Bala de Plata): Es un antipatrón cuya definición la podríamos resumir con la siguiente frase: “Es posible que con un serrucho cortes un tronco, pero ¿puedes cortar una roca?”. La diferencia con el Martillo de Oro es que ese antipatrón se basa en una solución metodológica completa, la bala de plata trata de soluciones más concretas, de buenas practicas que pensamos que funcionan siempre, sin pararnos a pensar dónde nos ha funcionado, por qué nos ha funcionado (si realmente nos ha funcionado) y dónde nos encontramos ahora.

- Mito del hombre-mes: Es otro antipatrón que consiste en que si asignas más programadores a un proyecto atrasado sólo lo retrasarás aún más, debido al tiempo requerido por los nuevos programadores para aprender acerca del proyecto, así como al aumento en la sobrecarga de comunciaciones.

- SOA (Arquitectura Orientada a Servicios): Es un marco de trabajo conceptual que permite a las organizaciones unir los objetivos de negocio con la infraestructura de TI integrando los datos y la lógica de negocio de sus sistemas separados. El objetivo para una acompañia de contar con sistemas integrados, responde a que las empresas necesitan poder interconectar los procesos, personas e información, tanto con la misma organización como con subsidiarias y socios comerciales.

- Desarrollo AGILE: Son métodos de ingeniería del software basados en el desarrollo iterativo e incremental (1 iteración ≈ 1-4 semanas), donde los requisitos y soluciones evolucionan mediante la colaboración de grupos organizacionales y multidisciplinarios (ej.: Dpto. de Desarrollo). Cada iteración del ciclo de vida incluye: planificación, análisis de requisítos, diseño, codificación, revision y planificación. Un ejemplo de desarrollo AGILE podría ser SCRUM, muy presente en los requisitos de ofertas de trabajo sobre desarrollo software.

- Diagrama de uso: Transmite una idea de qué puede hacer un sistema y quienes intervienen. A la izquierda se colocan los actors que son en un caso de uso particular quienes lo arrancan o quienes son los actores principales. En la derecha se ponen los actores secundarios, los que intervienen en los casos de uso pero que no son los principales. La caja del medio significa el ámbito de nuestro sistema software (p.e. servidor + bd, etc). Las elipses de dentro describen una petición para dar una idea en general de qué hace el sistema.

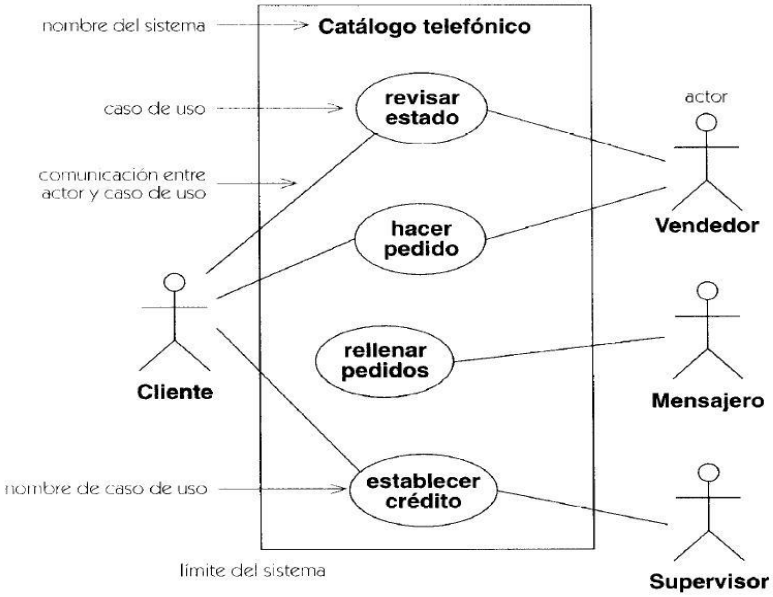


Imagen 1. Ejemplo de diagrama de casos de uso

- DRF: Se trata de un documento de requisitos funcionales. Un requisite funcional define una función del sistema de software o sus componentes. Una función es descrita como un conjunto de entradas, comportamientos y salidas.

- DevOps: Metodología de desarrollo de software que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de operaciones en tecnologías de la información (IT). DevOps es una respuesta a la interdependencia del desarrollo de software y las operaciones IT. Su objetivo es ayudar a una organización a producer productos y servicios software rápidamente.

Imagen 2. Diagrama DevOps

- Estructura típica del sistema de trabajo de una empresa desarrolladora de software:

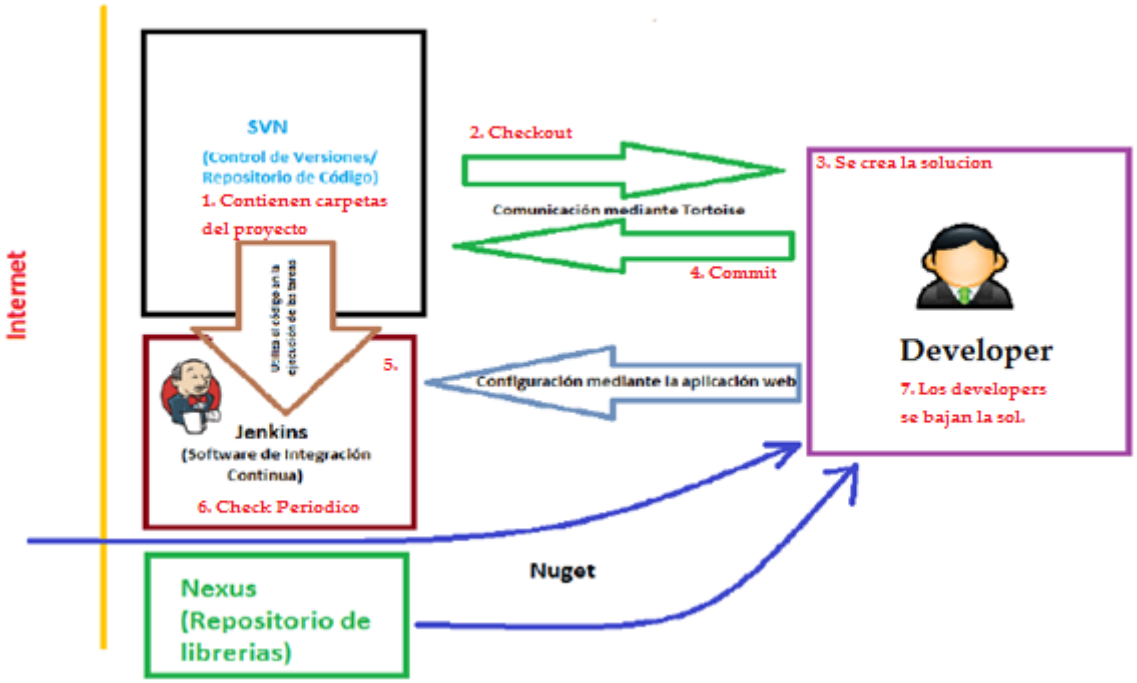


Imagen 3. Estructura del sistema de trabajo de una empresa de Software

- Curva de Gartner: También conocido como ciclo de sobreexcitación, es una representación gráfica de la madurez, adopción y aplicación comercial de una aplicación específica.

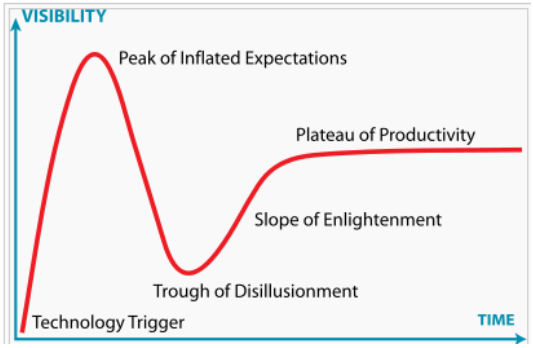


Imagen 4. Curva de Gartner

1.1. Objetivos del proyecto

Para estructurar este proyecto marqué una serie de objetivos definidos a continuación:

Definir una serie de casos de uso para la creación de una versión estable de la aplicación.

Entendiendo “casos de uso” como requisitos funcionales de la aplicación y sus interacciones.

Arquitectura

Dada la gran popularidad de la arquitectura por capas en las empresas desarrolladoras de software para crear productos informáticos, he optado por utilizar una arquitectura tipo REST dividida en 3 capas para las aplicaciones desarrolladas en este para que así se tenga una vision más realista de lo que se suele hacer en las empresas. Dicha

FrontEnd

Para las aplicaciones a desarrollar se decidió que la parte Front de las aplicaciones se desarrollaría mediante los Frameworks de JavaScript: AngularJS y jQuery. Una de las razones de elegir AngularJS se encuentra en la curiosidad por utilizarla ya que se trata una herramienta novedosa que está pegando un salto muy importante durante estos últimos años y está enfocado en el desarrollo de aplicaciones web en un tiempo relativamente corto.

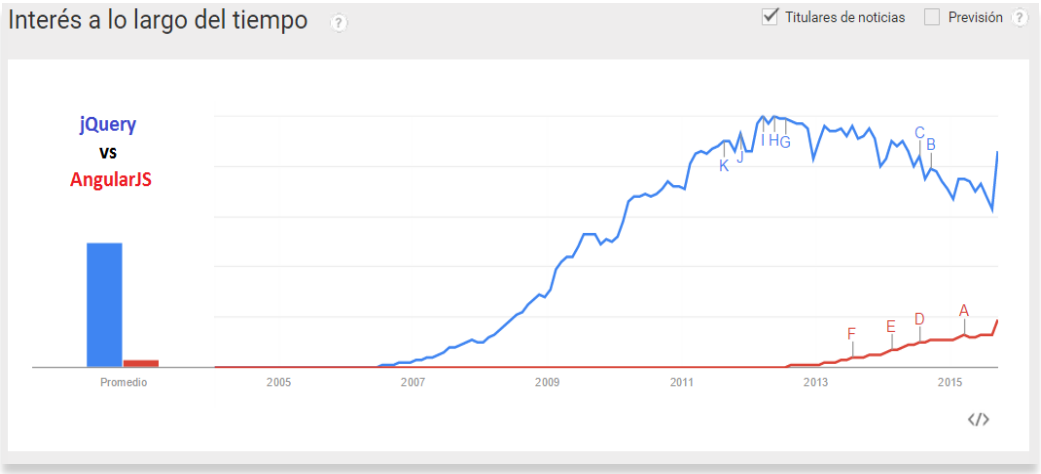


Imagen 5. Gráfica comparativa jQuery vs AngularJS

BackEnd

En cuanto al Back, se ha utilizado C# debido a la gran popularidad de los lenguajes orientados a objetos. Tambien se podría haber utilizado Java o PHP. En lo que se refiere a Base de Datos, se han gestionado las tablas de datos mediante SQLServer, y en algunas aplicaciones también se ha utilizado un framework de este: Entity Framework. Más adelante se explicará el porqué de dicho framework.

Patrones de programación

Los patrones en programación son muy interesantes para los desarrolladores, ya que

ofrecen soluciones a problemas communes y cotidianos a la hora de desarrollar una aplicación. Por tanto he decidido utilizar algunos en este proyecto.

Testeo.

Dado que en el mundo del desarrollo de Software siempre surgen bugs, se agregan más funcionalidades, se integran diferentes tecnologías, y se trabaja en grupo, es de vital importancia probar en la medida de lo posible todos los casos de uso de la aplicación. Esto nos va a permitir que cuando tengamos que tocar algo de la aplicación no interfiera con lo que se ha hecho anteriormente. Es decir, nos proporciona mantenibilidad y escalabilidad.

Posteriormente se explicarán las funcionalidades de las aplicaciones realizadas. Para finalizar, se detallarán posibles mejoras y nuevas funcionalidades que podrían resultar de utilidad, así como las conclusiones en las que explico cuáles considero que son los aspectos en los que este proyecto me ha hecho mejorar las habilidades y conocimientos que ya poseía y obtener nuevos.

2. Características técnicas

2.1. Herramientas y tecnologías utilizadas para las aplicaciones web

El desarrollo de la aplicación se ha llevado a cabo utilizando los siguientes lenguajes, entornos de desarrollo y tecnologías:

- Entorno de desarrollo: Visual Studio Community: Es el entorno estándar para trabajar con C# y con todas las herramientas .NET. Además es gratuito si se utiliza para fines académicos.

- Entorno del Software: El entorno sería más o menos así:

En paralelo a esto está la gestion del software: Depuración de bugs, desarrollo de nuevas versiones, documentación, repositorios, etc.
En cuanto a la estructura del código, por un lado tendremo los DATOS (estructuras) y por otro la LÓGICA.

- Lenguajes de programación: Antes de citar los lenguajes es importante tener claro los tecnicismos FrontEnd y BackEnd.

→ FrontEnd: Son todas aquellas tecnologías que corren del lado del cliente, es decir, todas aquellas tecnologías que corren del lado del navegador web, generalizandose más que nada en tres lenguajes, HTML, CSS y JavaScript. Normalmente en FrontEnd se de estilizar la página de tal manera que la página pueda quedar comoda para la persona que la ve, es decir, esta persona, debe de conocer tecnicas de User Experience para dar una experiencia de usuario comoda a la persona que visita la página, así mismo debe de saber de diseño de Interacción para que sepa colocar las cosas de tal manera que el usuario las pueda ubicar de manera rápida y comoda, es decir, el Backend posteriormente se encargará de llenar las páginas de información (en ocasiones) y de colocar la página en un servidor, pero esto de nada sirve si la página es fea, por tanto, el programador del FrontEnd debe de saber un poco de diseño, ya que como he dicho anteriormente, este se va a encargar de que la página no solo se vea bonita para el usuario, si no que sea comoda de utilizar, comoda de navegar e intuitiva, existen muchas tecnologías relacionadas a estos tres lenguajes que se utilizan en el FrontEnd, por ejemplo, para JavaScript tenemos jquery, angular.js y backbone.js, que son

tecnologías avanzadas que utiliza el frontend, este se puede apoyar en librerias de CS y de JavaScript como animate.css y JQuery para poder dar una solución amena y comoda, así mismo, debe de conocer lenguajes de transferencia de información como XML y JSON, y Ajax para hacer solicitudes sin necesidad de refrescar la página, en pocas palabras, la parte Front es la que se encargará de dejar bonita la página, en ver que los datos se muestren de manera comoda para el usuario, de que la interacción que realice sea llamativa y en la estética del sitio.
En este proyecto se ha utilizado HTML, CSS (Bootstrap Framework), JavaScript (AngularJS y jQuery Framework) y AJAX para comunicarse con la parte Back mediante el framework ASP .NET Web API.

→ BackEnd: El programador backend es aquel que se encuentra del lado del servidor, es decir, esta persona se encarga de lenguajes como PHP, Python, .Net (C#), Java, etc, es aquel que se encarga de interactuar con bases de datos, verificar manejos de sesiones de usuarios, montar la página en un servidor, y desde este "servir" todas las vistas que el FrontEnd crea, es decir, uno como backend se encarga mas que nada de la manipulación de los datos, que en muchas ocasiones suele ser lo más tedioso, pero al mismo tiempo, un Backend no sirve de mucho si no existe un FrontEnd de por medio que se haya encargado de que la página se vea estetica, el programador de Backend normalmente debe de conocer Bases de datos, Frameworks y Librerias que le permitan desenvolverse mejor en la manera en la que sirve las páginas, ya que el se va a encargar de que todos los datos que llegan desde el FrontEnd, lleguen a una base de datos, por tanto, debe de conocer un poco de seguridad, para mantener los datos cuidados y tratar de protegerse de todo tipo de inyecciones que se puedan tratar de hacer al servidor para que no sea vulnerable, así mismo, se encarga de crear API's para que sus datos puedan consumirse de manera comoda para el frontend y que pueda mejorar la experiencia del usuario, teniendo en cuenta los cuidados necesarios que debe considerar para que su servidor pueda mantenerse seguro.
En este proyecto se ha utilizado **C#** y **SQLServer** (Entity Framework).

- Frameworks:

→ Unity Framework: Es un patron de diseño responsable de la inyección de dependencias. Se trata de una técnica que ayuda a I nyectar objetos dependientes de una clase. Con esto Unity nos permite el control automático de las dependencias.

→ ASP .NET WEBAPI: Framework de la familia .NET que tiene como objetivo el facilitarnos en gran medida la construcción de aplicaciones REST orientadas a ofrecer servicios, como podría ser Twitter, Facebook, Amazon, YouTube, etc. Además, incluye referencias para utilizar Entity Framework y construye el proyecto testing adecuado para el desarrollo en TDD (Test Driver Development)

→ Entity Framework: Es un ORM (Object Relational Mapping) que facilita las tareas básicas de acceso a los datos (CRUD) a través de una transformación de las tablas de una base de datos en una serie de entidades. Se ha optado por utilizar este framework para ahorrar tiempo en todo el desarrollo correspondiente a la gestión de la base de datos.

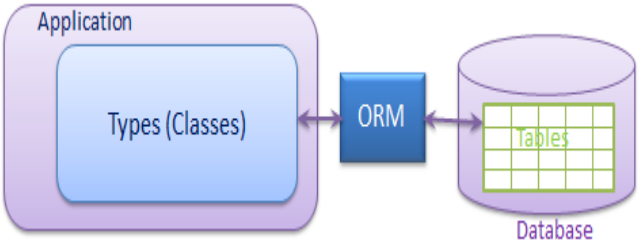


Imagen 6. Relación ORM

→ Selenium: Se ha optado por utilizar Selenium para el realizar los Tests Funcionales de las aplicaciones. De esta manera es posible automatizar todos los flujos de navegación necesarios.

- Repositorios: Como se trata de un proyecto individual, no tenia mucho sentido utilizar un control de versiones como por ejemplo Tortoise SVN. Sin embargo, es importante saber utilizar este tipo de herramientas pues los proyectos de software desarrollados por las empresas son casi en su totalidad desarrollos en equipo y este tipo de herramientas ayuda a mantener un control de ficheros, históricos, guardados y organización o sincronización del equipo.

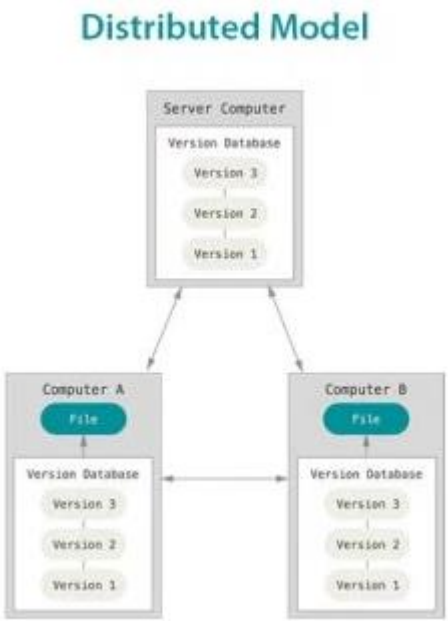


Imagen 7. Modelo Distributivo en SVN

3. Desarrollo de las aplicaciones

3.1. Metodología de desarrollo

El modo en que se ejecuta la creación del software puede depender de muchos factores como los requisitos definidos, la disponibilidad de recursos económicos o capital humano disponible o el tiempo requerido. Es necesario plantearse como llevar a cabo un proyecto de este tipo teniendo en cuenta los factores que pueden intervenir durante la gestación del mismo para poder llevarlo de una manera óptima.

Imagen 8. Ilustración Calidad VS Precio

Según se considere, un proyecto puede realizarse siguiendo varias metodologías. Entre ellas se consideraron para el desarrollo de las aplicaciones las siguientes:

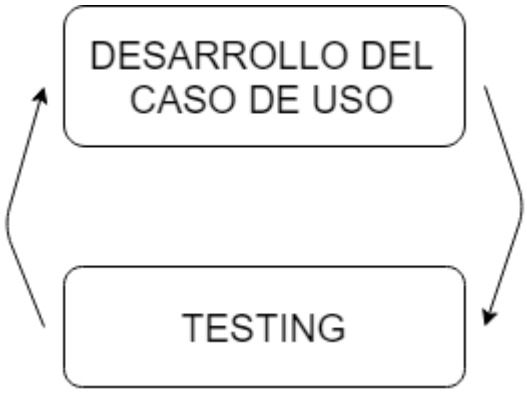
> Una metodología en cascada, en la cual se realiza un análisis y un diseño del sistema, se programa por completo y se procede a realizar las pruebas necesarias.
>
> Una metodología iterativa, en la que el proyecto se divide en grupos de tareas denominados iteraciones. En cada una de ellas se analiza el trabajo a realizar, se ejecuta y se prueba el resultado.

Ell principal problema de este trabajo estaba en el tiempo. Debido a compaginar la realización del proyecto con los demás proyectos de mi jornada laboral, el número posible de horas semanales a dedicar planteaba un problema si se decidía realizar toda la funcionalidad a la vez.

La decisión consistió en seguir el método iterativo dividiendo el proyecto en las etapas necesarias para ir consiguiendo secciones completas hasta la creación de una version cerrada de la aplicación.

A medida que se han ido realizando las tareas, se ha realizando testing en todos los ámbitos (unitario, de integración y funcional).

Las etapas del desarrollo se pueden resumir con el siguiente esquema:



3.2. Arquitectura

A la hora de llevar a cabo un desarrollo, es de vital importancia realizar una estructura del código ordenada y que éste sea agrupado según diferentes características específicas. Por ejemplo puede resultar de utilizar realizar una agrupación que incluya lo necesario para presentar el interfaz gráfico, otra que una toda la gestión de acceso a una fuente de datos, una tercera que tome unos datos y los manipule según las necesidades existentes...

Una adecuada estructura del código aporta muchas ventajas. Algunas de ellas son las siguientes:

Permite una mejor comprensión del código y un aprendizaje más rápido de su manejo cuando no se ha intervenido desde el inicio de su creación. - Este punto es especialmente útil cuando intervienen varios desarrolladores o el software se crea y mantiene en un largo periodo de tiempo.

Facilita el mantenimiento y actualización de la aplicación. - Al dividirlo en capas, es posible sustituir más rápidamente partes del programa que se quieran mejorar o corregir sin afectar al resto.

Mejora el proceso de testeo del software al organizar el código en función de sus características. - Separando la parte visual de la aplicación de su funcionalidad es posible realizar pruebas unitarias, de ingregración y funcionales que simulen el funcionamiento de la aplicación e identifiquen posibles errores.

Reduce el tiempo de creación de software al poder reutilizarse el código entre aplicaciones según las necesidades que surgan.

A lo largo de la historia de la programación se han ido ideando muchas formas para realizar esta estructura y la comunicación entre las partes que la componen. Estas formas son denominadas patrones de arquitectura de software y tratan de aportar soluciones a los diferentes problemas que puede presentar el crear una estructura que organice el código que compone un programa.

Existe una gran variedad de patrones de diseño y su estudio y comprensión sería digno de un trabajo fin de grado independiente por lo que simplemente se detallará a continuación la solución utilizada para el software objeto de este trabajo.

3.2.1. Estructura de la aplicación

Antes del inicio del proyecto se buscó información sobre cuáles son los modos más comunes a la hora de estructurar una aplicación web en un entorno laboral y cómo de fácil sería encontrar explicaciones y tutoriales que pudiesen ser de ayuda para aprender lo necesario para acometer la tarea.

Se observó que había bastante información y ejemplos para el patron arquitectural de 3 capas en conjunto con el patron de presentación MVC, lo que era óptimo para un mejor aprendizaje, y además se consideró adecuada la separación del código que plantea, pudiendo resultar sencillo posteriormente agregar nuevas funcionalidades, por lo que se optó por esta opción.

Diseñar en 3 capas trata sobre no poner todo el código en las interfaces de usuario de tu sistema (UI). Para subsanar esto, la idea es tener 3 niveles de funcionalidad bien definidos y evitar problemas de concurrencia:

- Capa Controller: Recibe peticiones del exterior (Usuario) y decide a qué lógica de negocio de la capa de servicio debe dirigirse.
- Capa Service: En esta capa reside la lógica de negocio. Es decir, es la que gestiona la operación expuesta en el interfaz y según lo solicitado por el usuario en la petición, solicita unos datos a la la capa repository y al recibirlos, opera con ellos generando los objetos de respuesta que sean necesarios.
- Capa Repository: Su responsabilidad es la lógica de acceso a la base de datos.

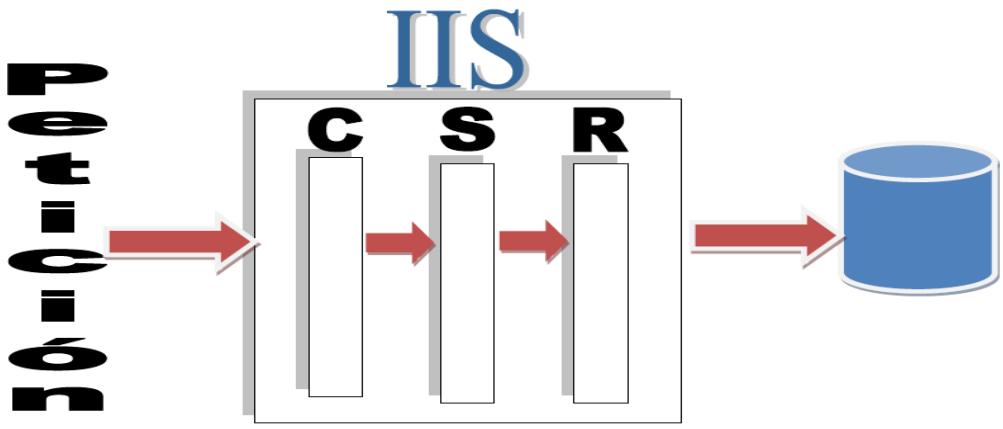


Imagen 9. Arquitectura en 3 capas

Estas 3 capas son de lógica y se engloban en un IIS (Servidor Web) encargado de hostear la aplicación web en un entorno Windows, además de ofrecer una serie de facilidades como por ejemplo la gestión de hilos, port listeners, cargar y descargar contenidos mediante FTP, HTTP, etc.

Cada capa tendrá una clase y se seguirá la siguiente convención: Tendrán como sufijos los nombres de las capas, por ejemplo:

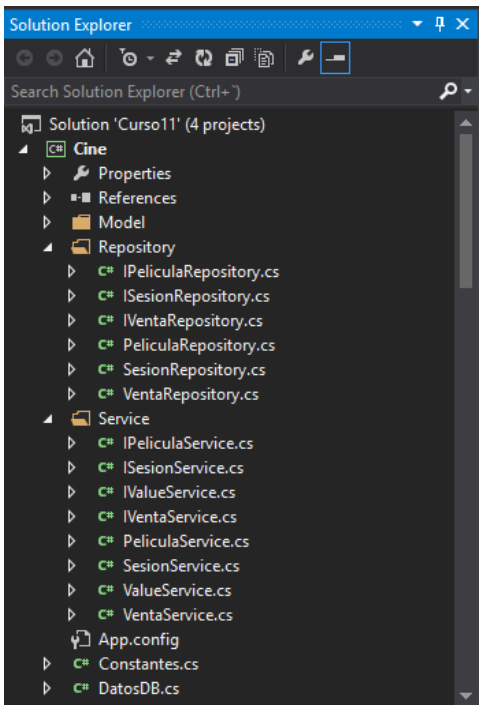


Imagen 10. Ejemplo de sufijos

El uso de este patron arquitectural tiene numerosas ventajas, entre ellas cabe destacar:

Mejoran la seguridad ya que es posible otorgar permisos a los procedimientos para obtener acceso o manipular determinaros objetos de la base de datos, y denegárselo para otros casos. Además al no enviarse la consulta completa se evita que un posible usuario externo revise nombre de tablas o campos de manera maliciosa para realizar posteriormente operaciones no deseadas en la base de datos.

Mayor posibilidad de reutilización del código, al poder utilizarlo desde múltiples puntos de un programa con solo realizar una llamada por su nombre al procedimiento. Esto evita el tener que escribir toda la consulta cada vez que se necesite acceder a los mismos datos.

Mejora en el tráfico, al enviar a través de la red solo el nombre del procedimiento y los parámetros de filtrado si los tiene, en oposición a mandar todo el texto de la consulta.

Mejor mantenimiento del código al tener la posibilidad de cambiar las consultas en un único lugar si hay una modificación en el modo de listado de datos.

En cuanto al patrón de presentación MVC, consiste en tres partes bien diferenciadas:

18

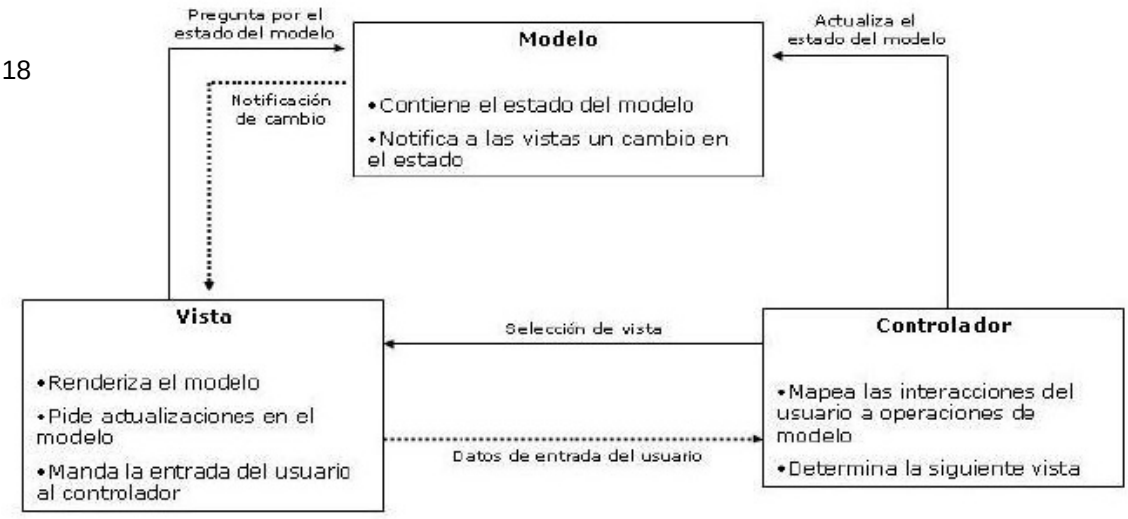


Imagen 11. Ejemplo Arquitectura MVC

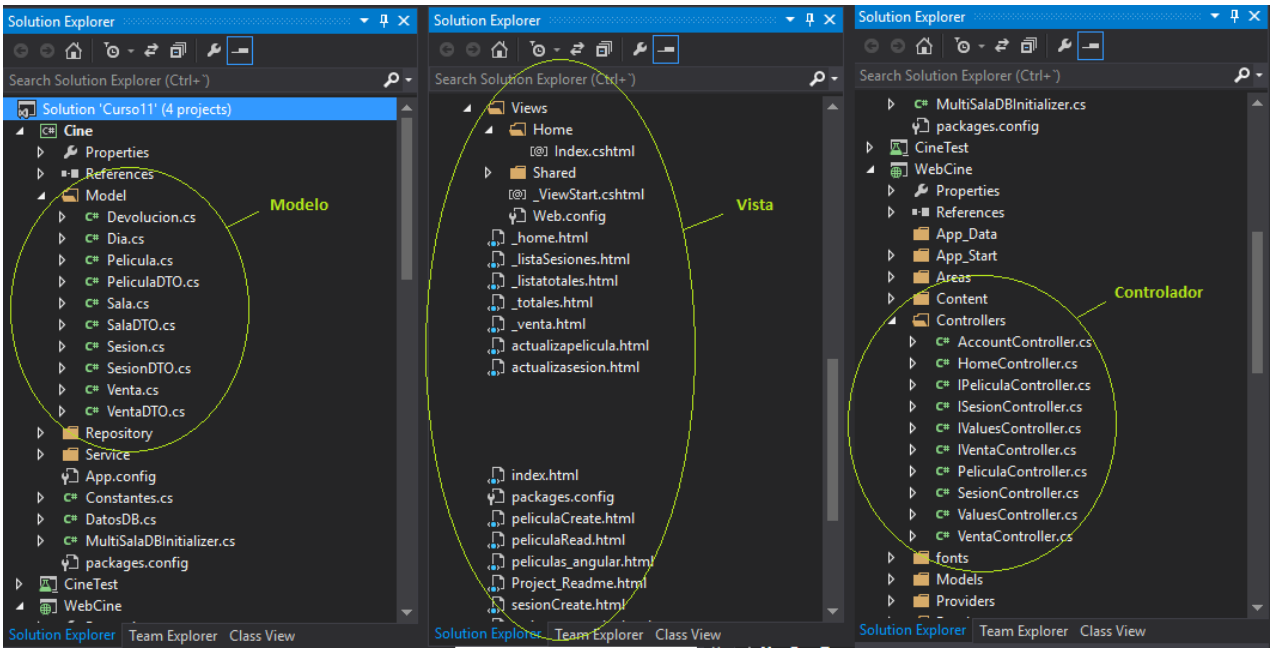


Imagen 12. Archivos MVC

En cuanto a los principales beneficios de utiliar el patron de diseño MVC son:

Convierte una aplicación en un paquete modular fácil de mantener y mejora la rapidez del desarrollo.

La separación de las tareas de la aplicacion en modelos, vistas y controladores hace que la aplicación sea además muy ligera de entender.

Las nuevas características se añaden fácilmente y agregar cosas nuevas a código viejo se hace muy sencillo.

El diseño modular tambien permite a los desarrolladores y a los diseñadores trabajar simultáneamente, incluyendo la capacidad de hacer prototipos rápidos.

4. Las aplicaciones

4.1. Gestión de Cines

Esta aplicación trata de cubrir algunos de los casos de uso típicos de los lugares encargados de gestionar la venta de entradas de cine.

4.1.1 Interfaz

Se trata de una aplicación de una sola página (Single Page Application), una técnica de desarrollo web muy utilizada hoy en día que consiste en brindar al usuario una experiencia de aplicación desktop, con rapidez y fluidez sin recargas innecesarias gracias a AJAX.

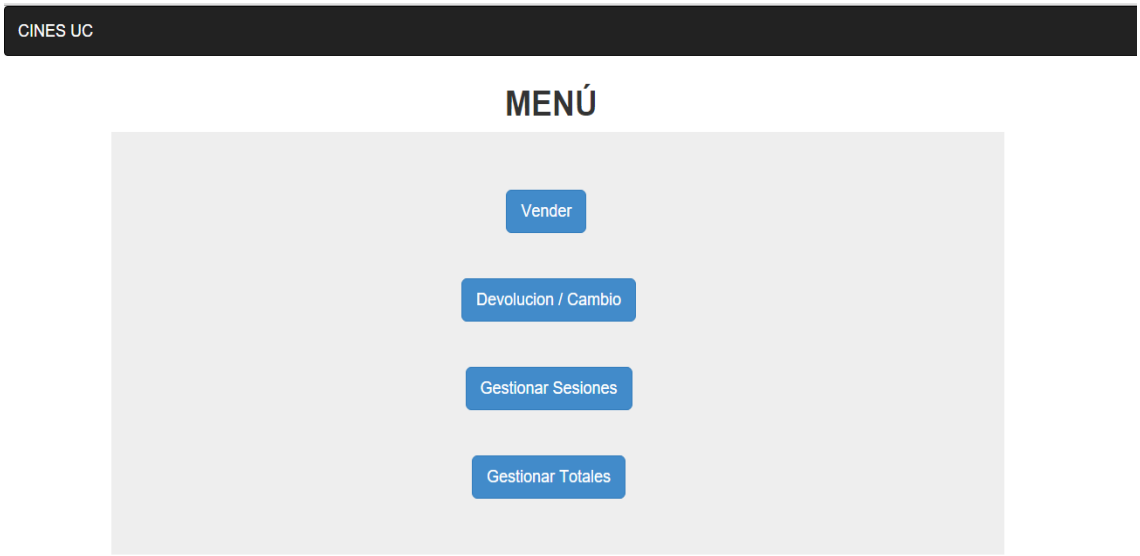


Imagen 13. Pantalla principal CINES UC

4.1.1.1. Casos de uso

VENDER

Imagen 14. Pantalla de Venta

Imagen 15. Pantalla de confirmación de venta

Imagen 16. Pantalla de venta realizada

DEVOLUCIÓN / CAMBIO

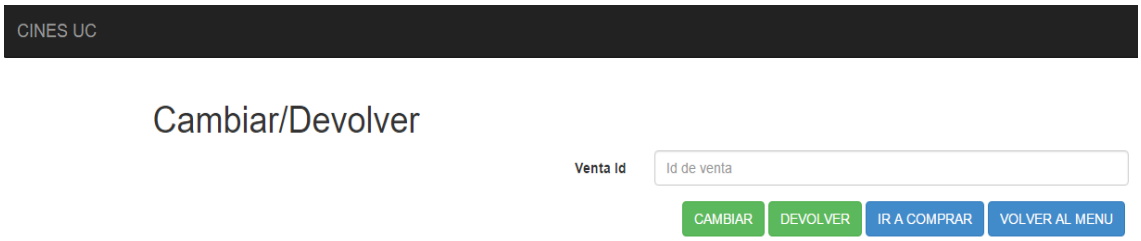


Imagen 17. Pantalla de cambio/devolución de entrada(s)

GESTIONAR SESIONES

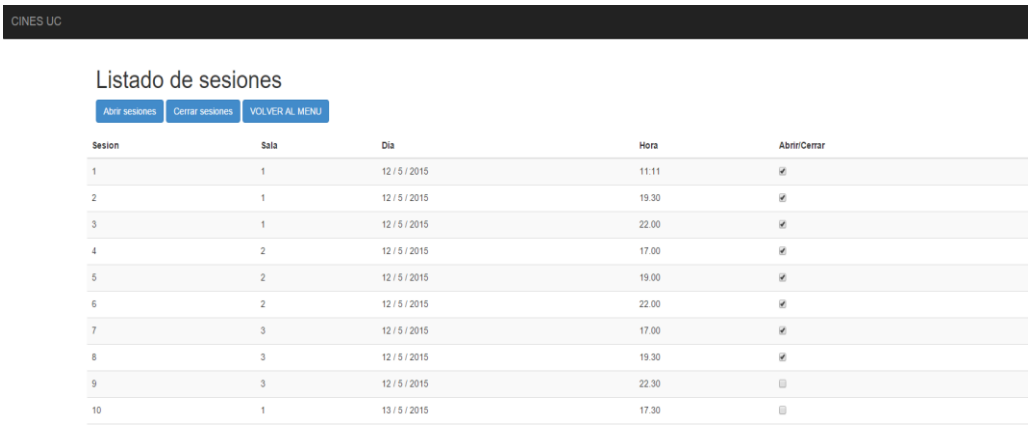


Imagen 18. Pantalla de gestión de las sesiones

GESTIONAR TOTALES

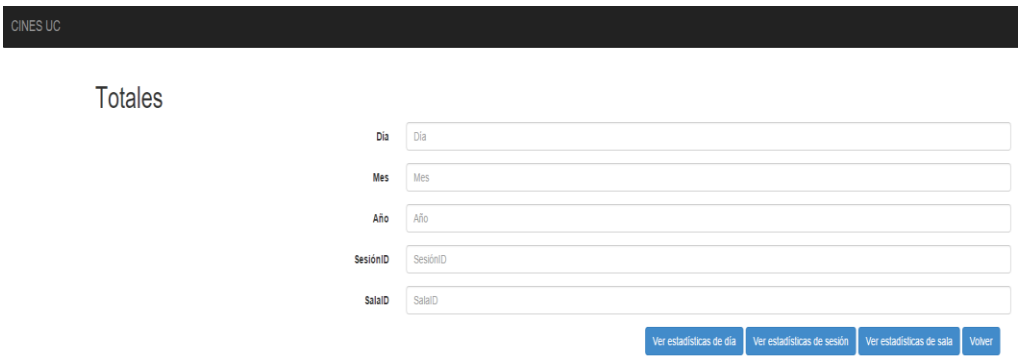


Imagen 19. Pantalla de gestión de totales recaudados

4.1.2. Parte BackEnd

Los objetos clases de la propia aplicación: Venta, Sesion, Sala, etc. son inicializados y añadidos a tablas de base de datos mediante Entity Framework:

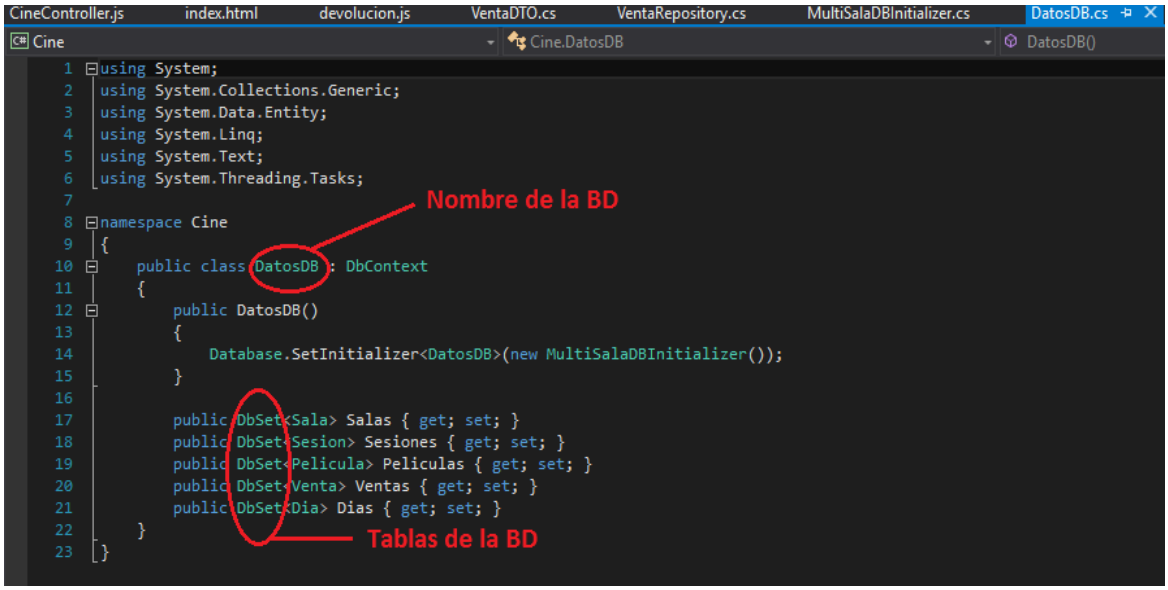


Imagen 20. Clase de definición de las tablas de la BD

A su vez, la última capa (Repository) se encarga de registrar las nuevas ventas, así como de listarlas, leerlas, borrarlas, actualizarlas, …

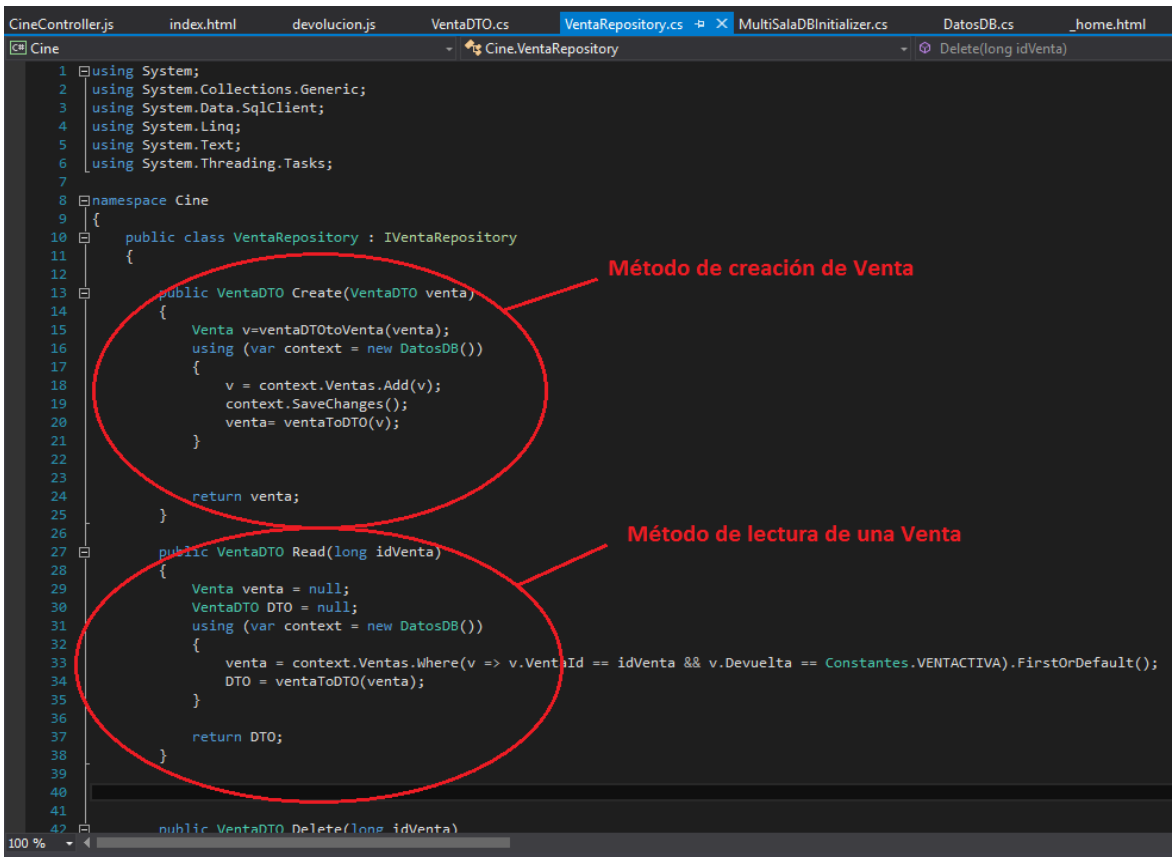


Imagen 21. Trozo de código de la clase VentaRepository

En cuanto a la capa intermedia, la de Service, se encarga de la lógica del modelo de negocio de la aplicación, como por ejemplo el calculo del precio de la(s) entrada(s).

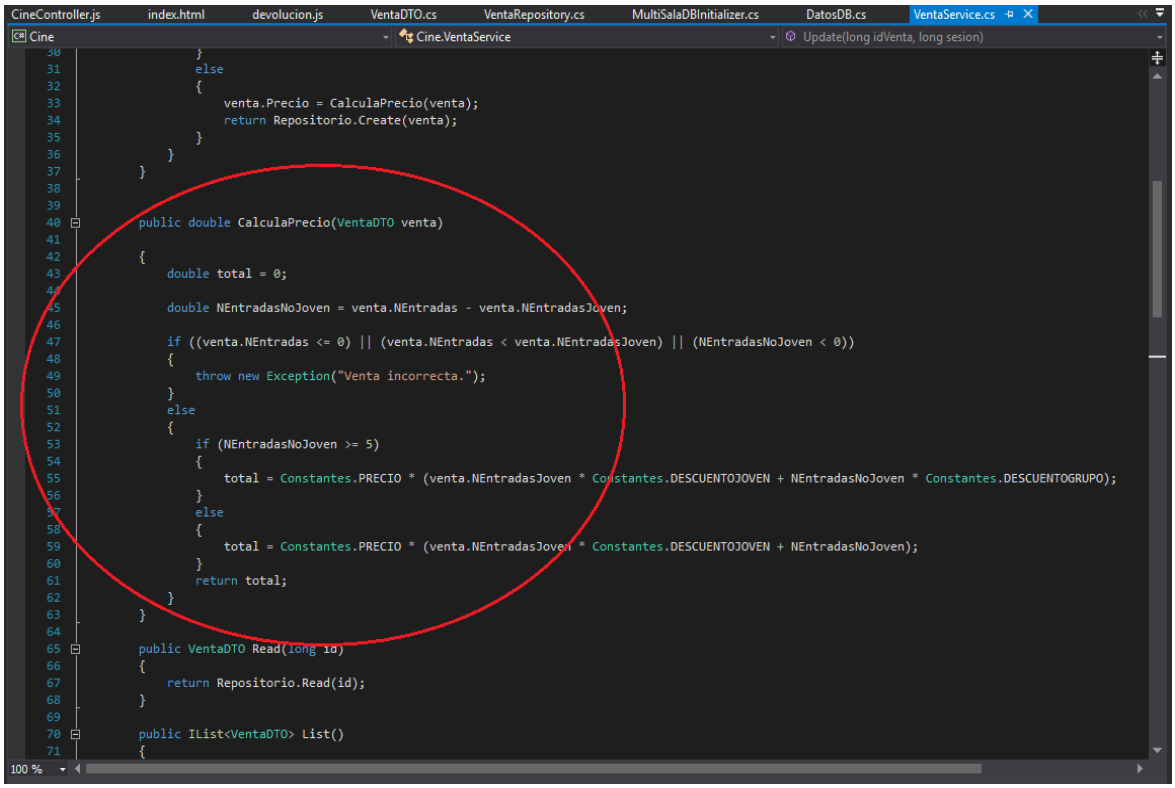


Imagen 22. Trozo de código de la clase VentaService responsable de calcular el precio de la(s) entrada(s)

Respecto a la primera capa, Controller, es la reponsable de recibir las peticiones del exterior, en este caso del fichero JavaScript mediante las llamadas AJAX.

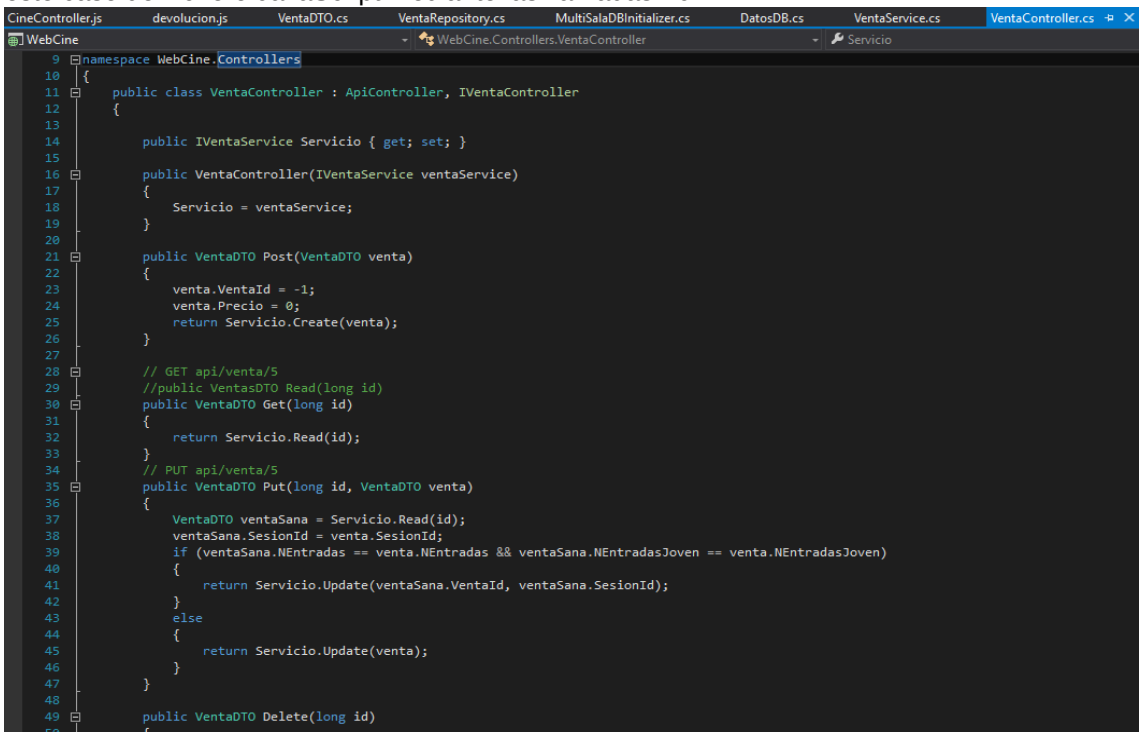


Imagen 23. Trozo de código de la clase VentaService responsable de calcular el precio de la(s) entrada(s)

4.1.3. Parte FrontEnd

CineController.js | devolucion.js | VentaDTO.cs | VentaRepository.cs | MultiSalaDBInitializer.cs | DatosDB.cs | VentaService.cs | VentaController.cs

```
392
393     }
394     // renders
395     _my.render = function (view, container, action, data) {
396         console.log("Render Call: view=" + view + "; container=" + container + "; action=" + action + "; data=" + JSON.stringify(data)
397         var actualAction = states[action] || null;
398         var responseCallback = function (result) {
399             if (actualAction) {
400                 if (typeof (actualAction.partialData) != 'undefined' && actualAction.partialData != null) {
401                     result = _.template(result)(actualAction.partialData);
402                 }
403             }
404             var source = $(result);
405             if (actualAction) {
406                 $(".site-title").text(actualAction.title);
407
408                 var inputs = source.find("input");
409                 if (actualAction.disabled != null && actualAction.disabled.length != 0) {
410                     for (var i = 0; i < actualAction.disabled.length; i++) {
411                         inputs.eq(i).attr("disabled", actualAction.disabled[i]);
412                     }
413                 }
414                 if (actualAction.buttons != null && actualAction.buttons.length != 0) {
415                     for (var i = 0; i < actualAction.buttons.length; i++) {
416                         source.find("#actions").append(actualAction.buttons[i]);
417                     }
418                 }
419             } else {
420                 $(".site-title").text("");
421             }
422             $("#" + container).html(source.html());
423             if (data) {
424                 data.Precio = Number(data.Precio).toFixed(2);
425                 descargarVenta(data);
426             }
427         };
428         $.ajax({
429             url: "/_" + view + ".html",
430             success: responseCallback,
431             dataType: "html",
432         });
433     };
434
```

Llamada AJAX (comunica con el Controller)

Imagen 23. Trozo de código responsable del renderizado de las pantallas y de de las llamadas AJAX al servidor

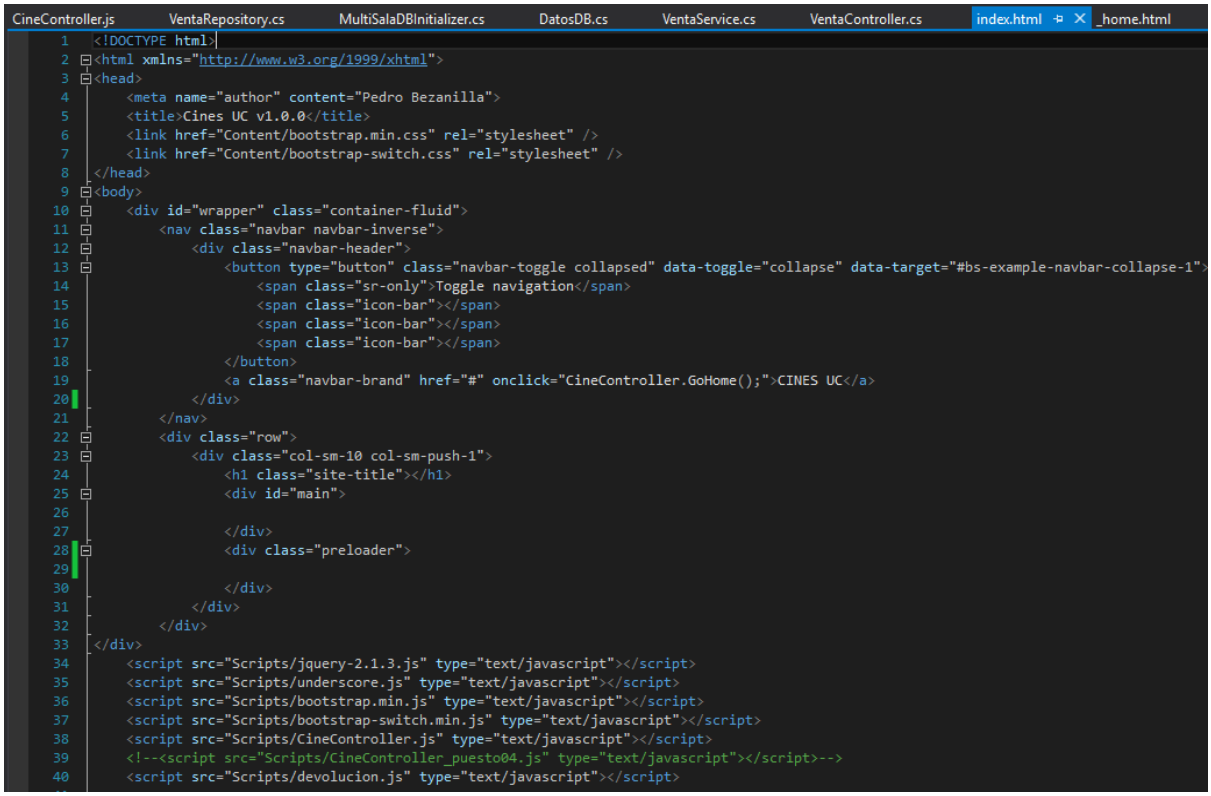


Imagen 24. Código del HTML principal

4.1.4. Testing

Los casos de uso de la aplicación han sido testeados durante todo el desarrollo de la misma además de otros casos de uso secundarios como por ejemplo el descuento joven mostrado a continuación.

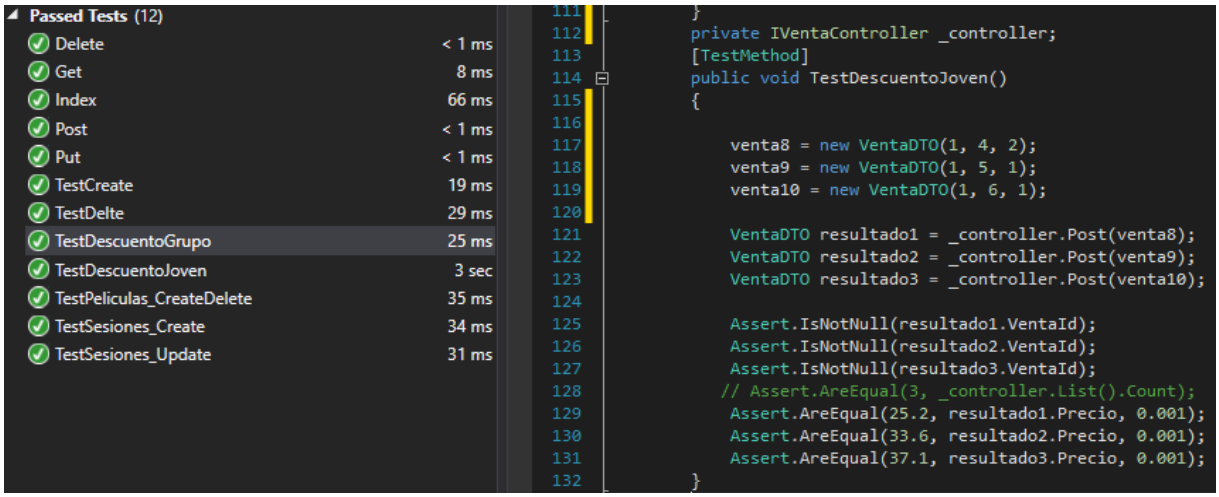


Imagen 25. Trozo de código responsable del testeo de la funcionalidad del decuento joven.

4.2. Motor de reglas

Esta aplicación está enfocada en el sentido de poder construir un circuito lógico sencillo, pues un motor de reglas no es más que la evaluación de condiciones IF-THEN y la ejecución de sus acciones correspondientes.

4.2.1. Interfaz

Se trata tambien de una aplicación de una sola página (Single Page Application).

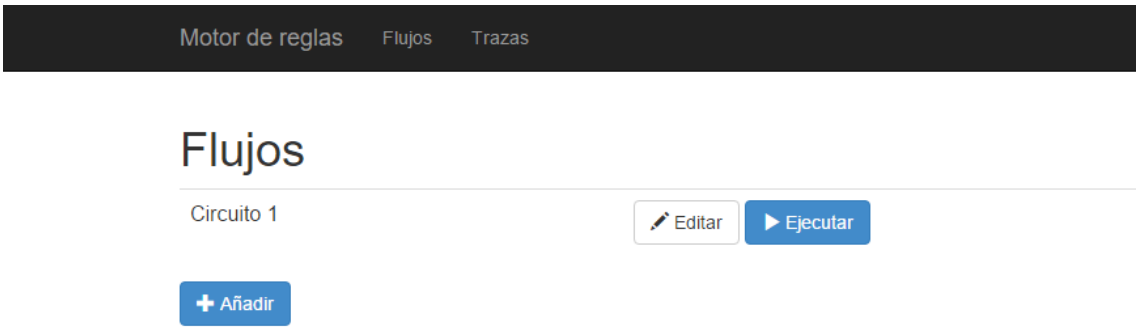


Imagen 26. Pantalla principal de la aplicación.

4.2.1.1. Casos de uso

AÑADIR FLUJO

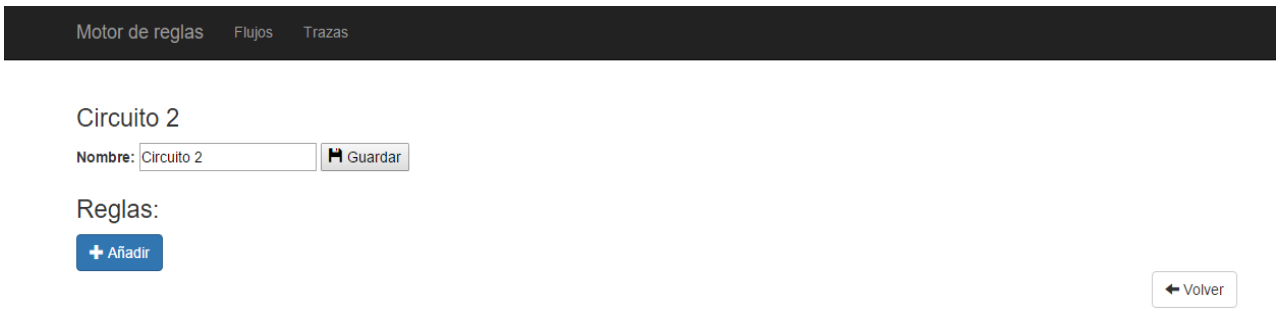


Imagen 27. Pantalla de añadir flujo

AÑADIR Y EDITAR REGLA

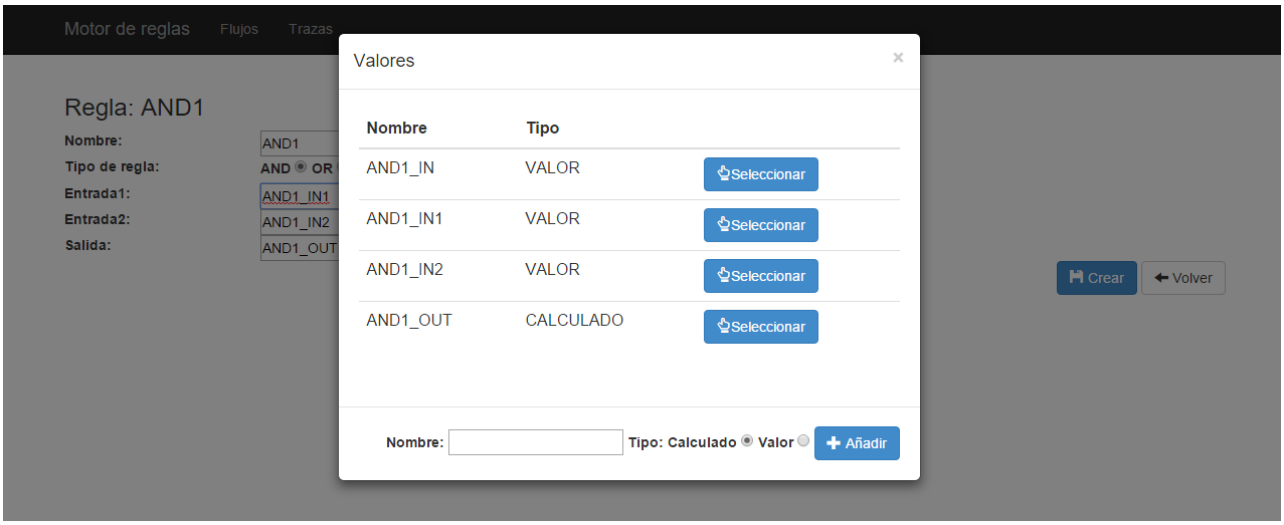


Imagen 28. Pantalla de edición de la regla

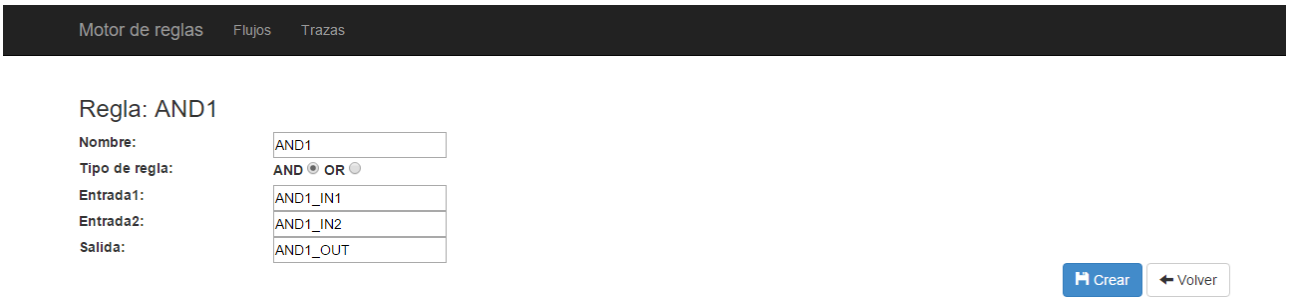


Imagen 29. Pantalla de añadir regla con los valores ya estabelecidos.

BORRAR REGLA

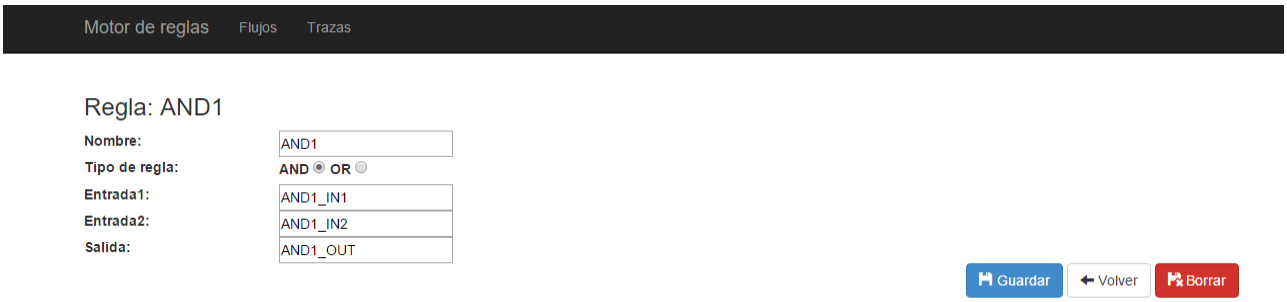


Imagen 30. Pantalla del borrado de la regla.

EDITAR O EJECUTAR FLUJO

Imagen 31. Pantalla del listado de flujos disponibles

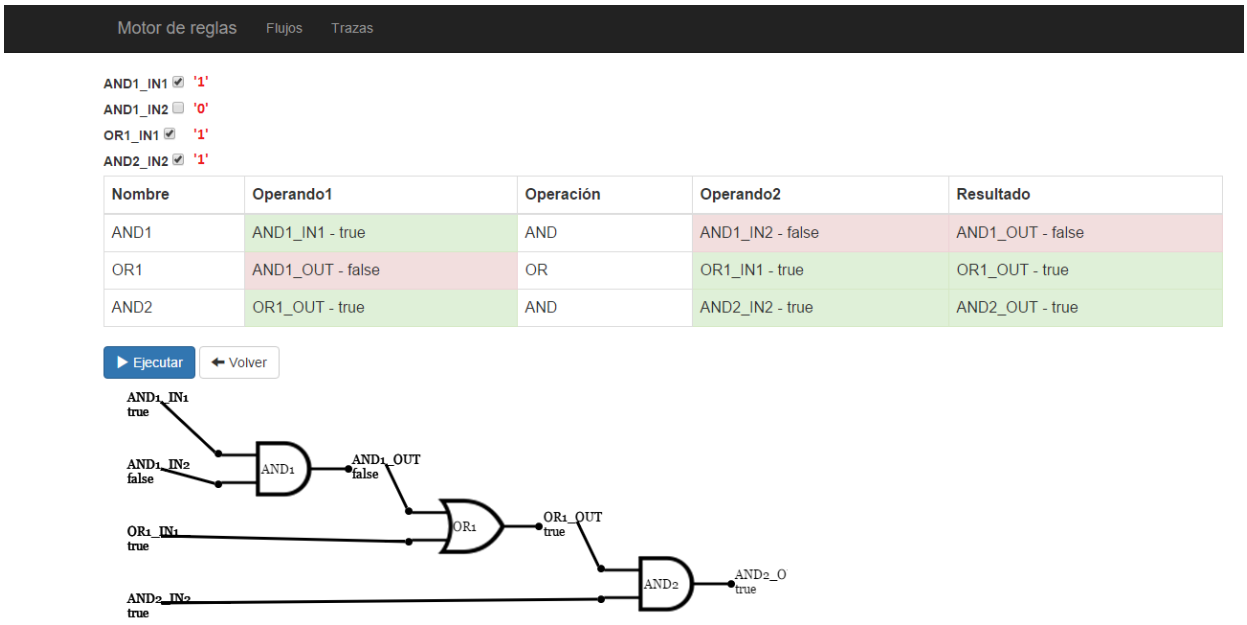


Imagen 32. Pantalla posterior a la ejecución del flujo

VER TRAZAS DE LAS EJECUCIONES

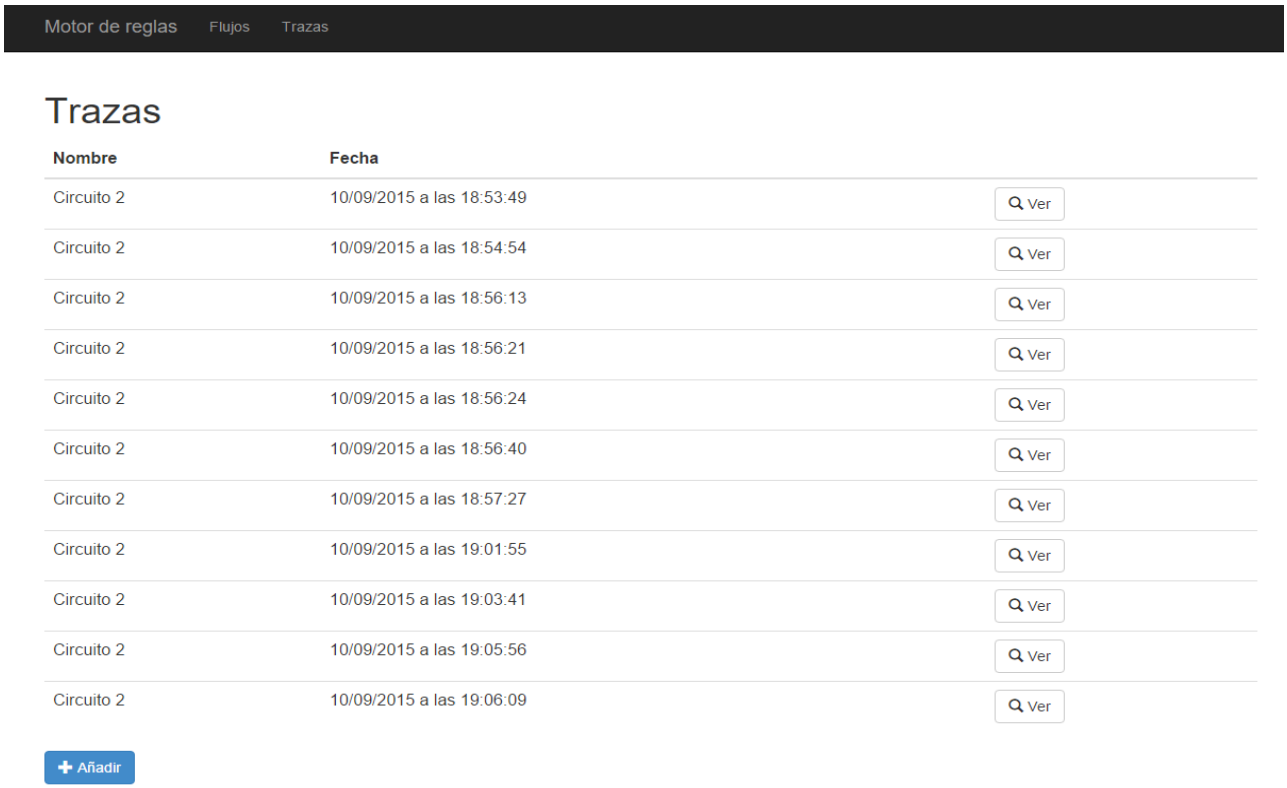


Imagen 33. Pantalla del listado de trazas

4.2.2. Parte BackEnd

Los objetos clases de la propia aplicación: Variable, Regla, Flujos, etc. son inicializados y añadidos a tablas de base de datos mediante las expresiones de Entity Framework ‘DbContext’ y ‘DbSet’, propias de Entity Framework:

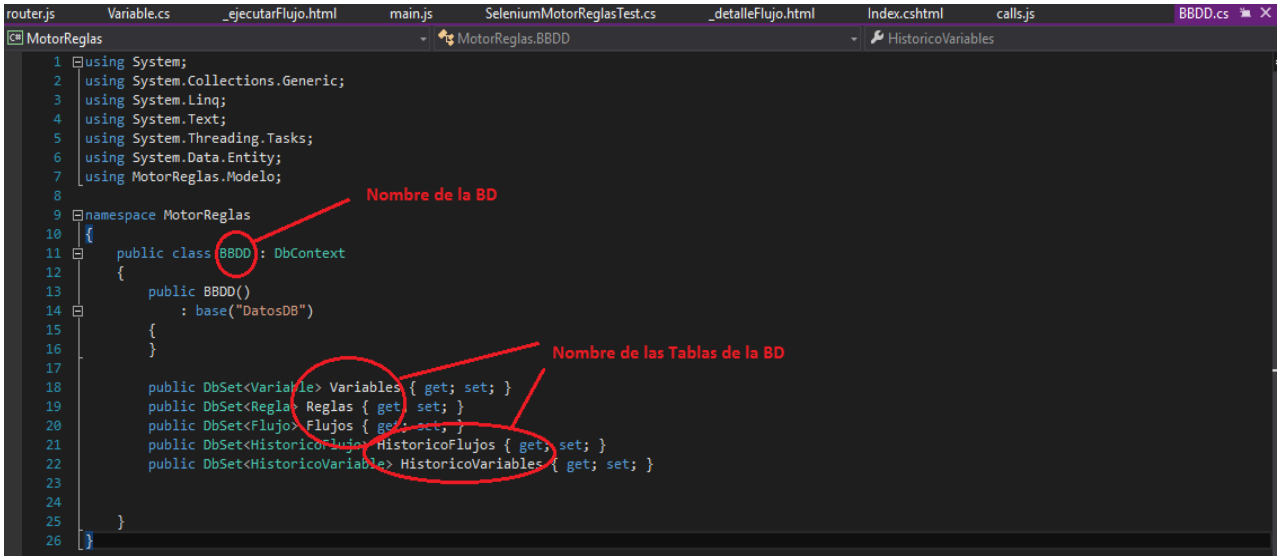


Imagen 34. Código de la definición de tablas de la BD

A su vez, la última capa (Repository) se encarga de registrar las nuevas reglas, flujos, históricos, etc. así como del listado, lectura, borrado y actualización.

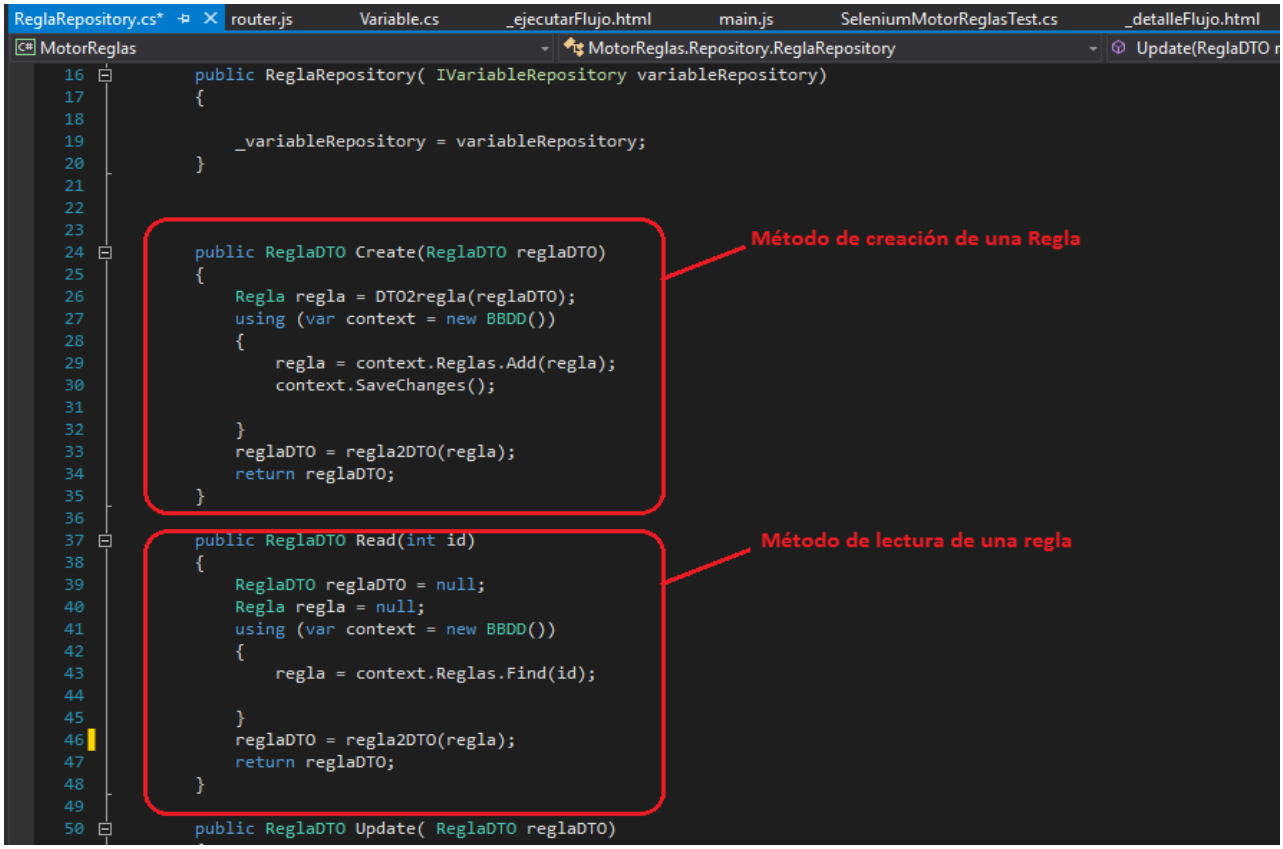


Imagen 35. Trozo de código de la clase ReglaRepository (Capa Repository)

En cuanto a la capa intermedia, la de Service, como ya se ha dicho, se encarga de la lógica del modelo de negocio de la aplicación, como por ejemplo el calculo del las salidas de las puertas lógicas:

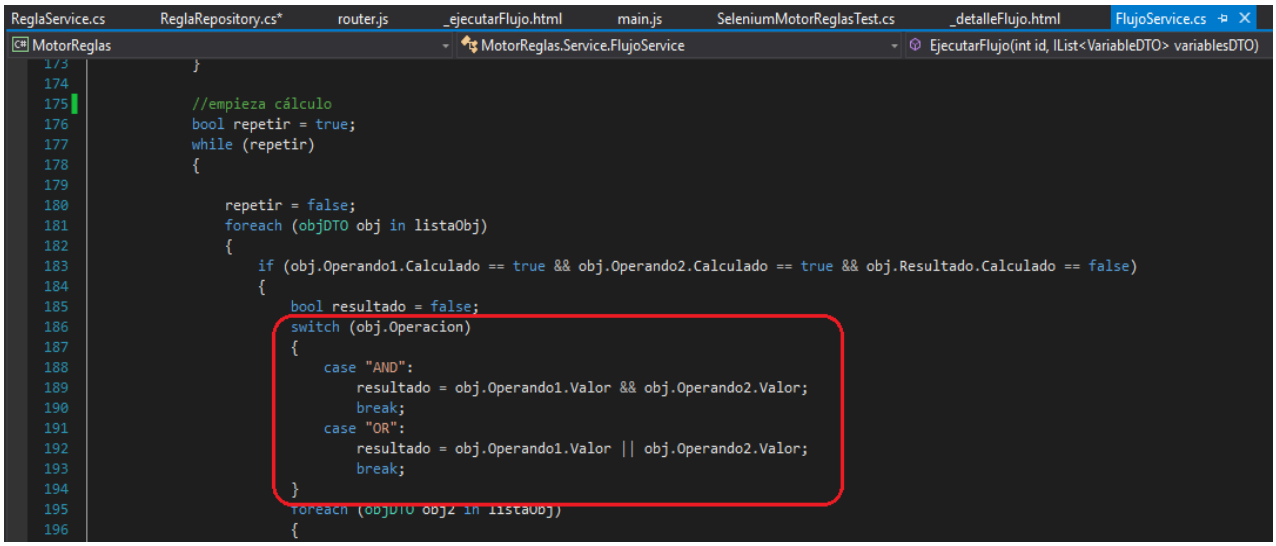


Imagen 36. Trozo de código de la evaluación de la salida de las puertas lógicas (Capa Service)

En cuanto a la primera capa, Controller, como también se ha dicho antes, es la reponsable de recibir las peticiones del exterior, en este caso del fichero JavaScript mediante las llamadas AJAX.

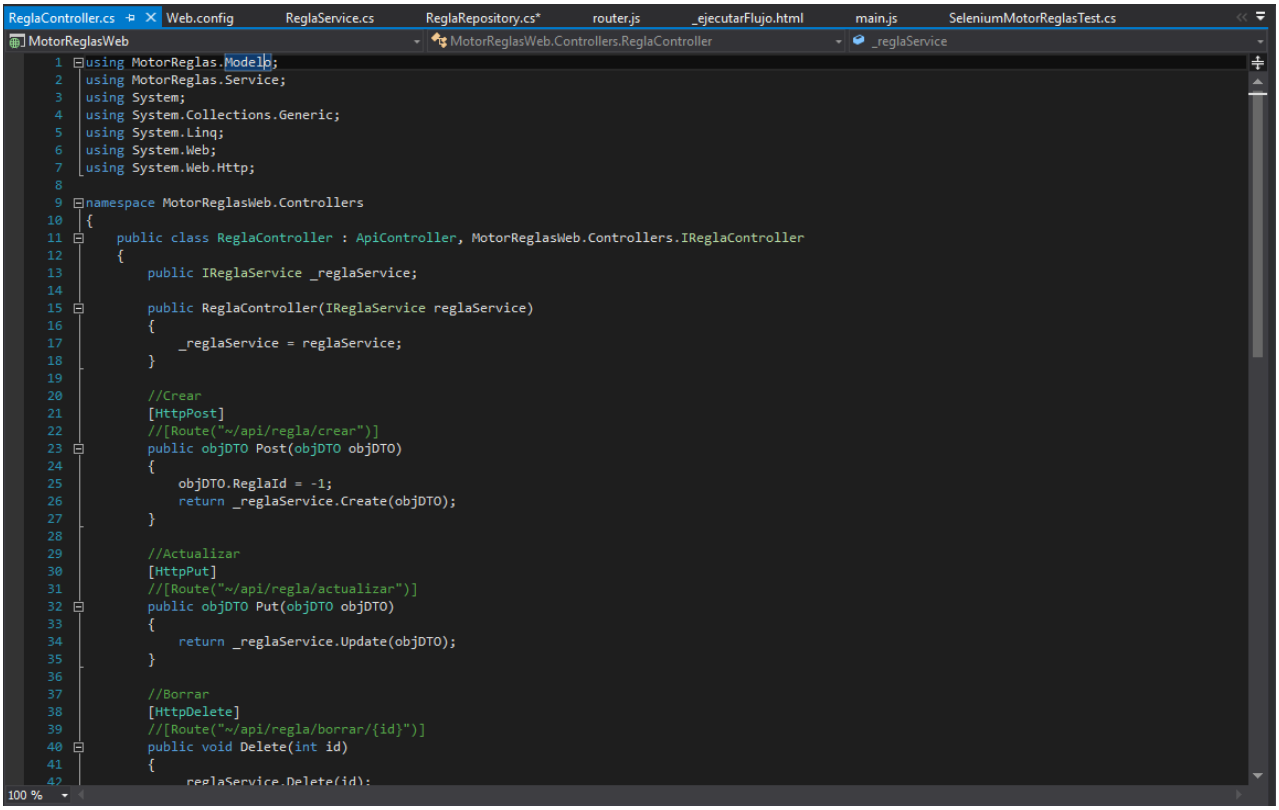
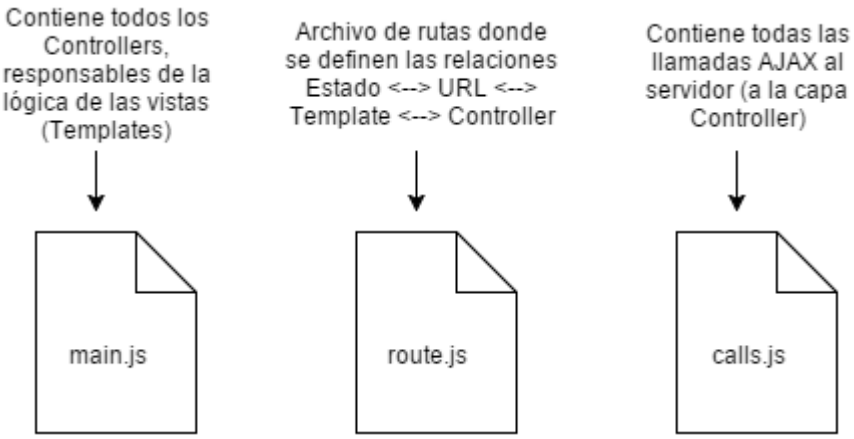


Imagen 37. Trozo de código de la clase ReglaController (Capa Controller)

4.2.3. Parte FrontEnd

La parte FrontEnd de esta aplicación está desarrollada mediante el framework AngularJS. Como se trata de un framework bastante modular, se ha dividido los archivos .JS en 3 partes:



_modalloading.html router.js main.js calls.js Index.cshtml _detalleFlujo.html _detalleRegla.html _ejecutarFlujo.html _listaFlujos.html

```
<html ng-app="reglasApp">
<head>
    <title></title>
    <link rel="stylesheet" type="text/css" href="Content/bootstrap.css">
    <script type="text/javascript" src="Scripts/angular.js"></script>
    <script type="text/javascript" src="Scripts/angular-ui-router.js"></script>
</head>
<body>

    <!-- modal que aparecera a la hora de cargar con gif de loading -->
    <div ng-show="loading" ng-include="'_modalloading.html'">
    </div>


    <div class="navbar navbar-inverse">
        <div class="container">
            <div class="navbar-header">
                <a class="navbar-brand" href="#/">Motor de reglas</a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a href="/#/">Flujos</a></li>
                    <li><a href="/#/trazas">Trazas</a></li>
                </ul>
            </div>
        </div>
    </div>

    <div ui-view class="container">
    </div>


    <script type="text/javascript" src="Scripts/app/main.js"></script>
    <script type="text/javascript" src="Scripts/app/trazas.js"></script>
    <script type="text/javascript" src="Scripts/app/router.js"></script>
    <script type="text/javascript" src="Scripts/app/calls.js"></script>
</body>
</html>
```

Imagen 38. Código HTML principal

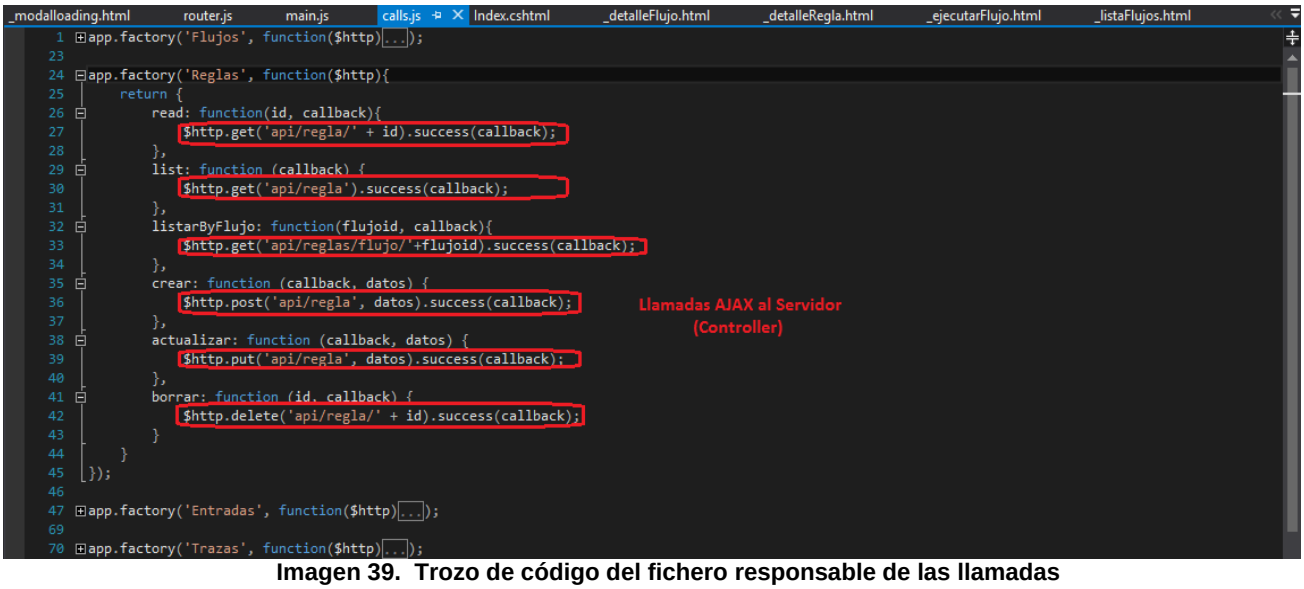


Imagen 39. Trozo de código del fichero responsable de las llamadas asíncronas (AJAX)

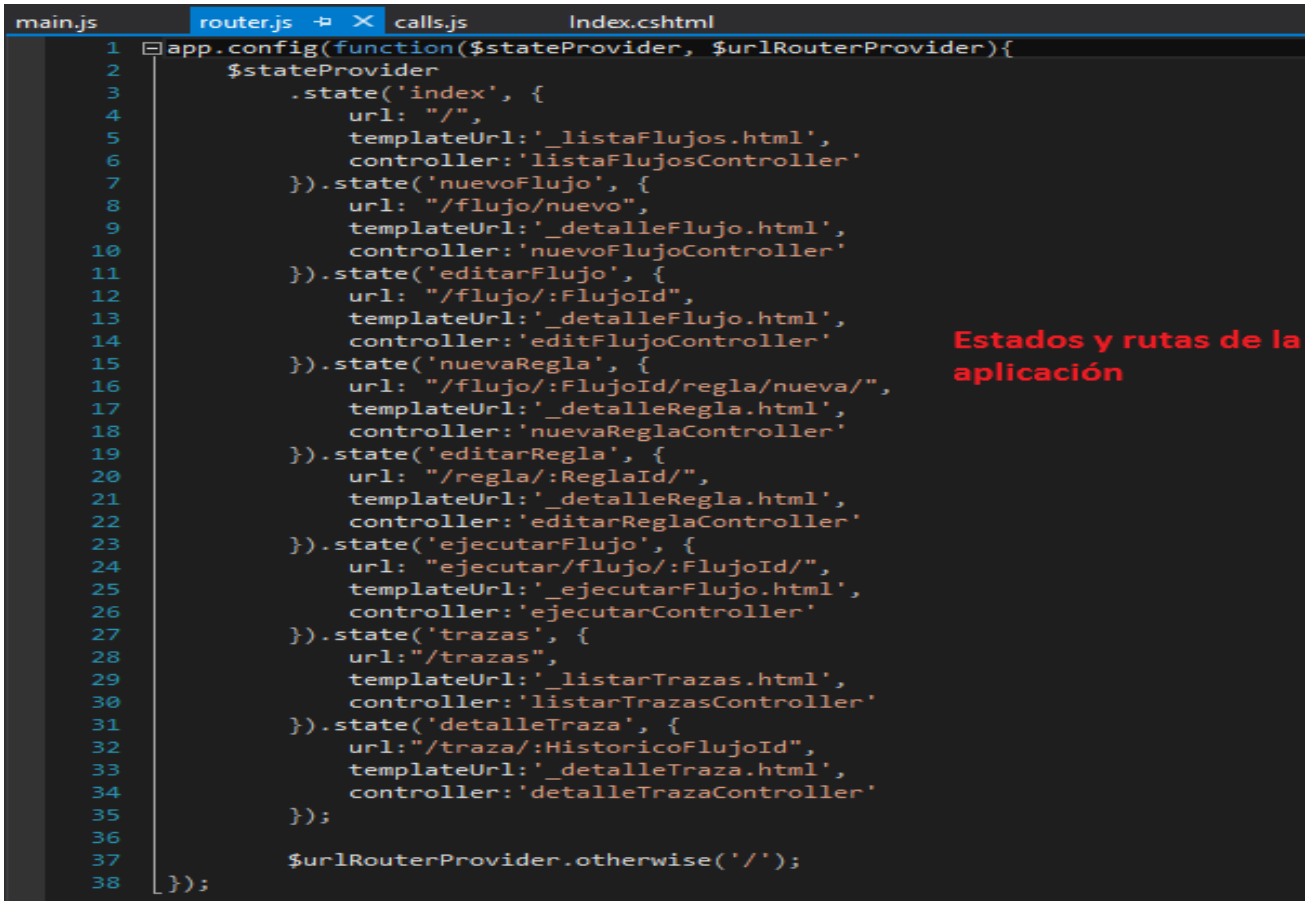
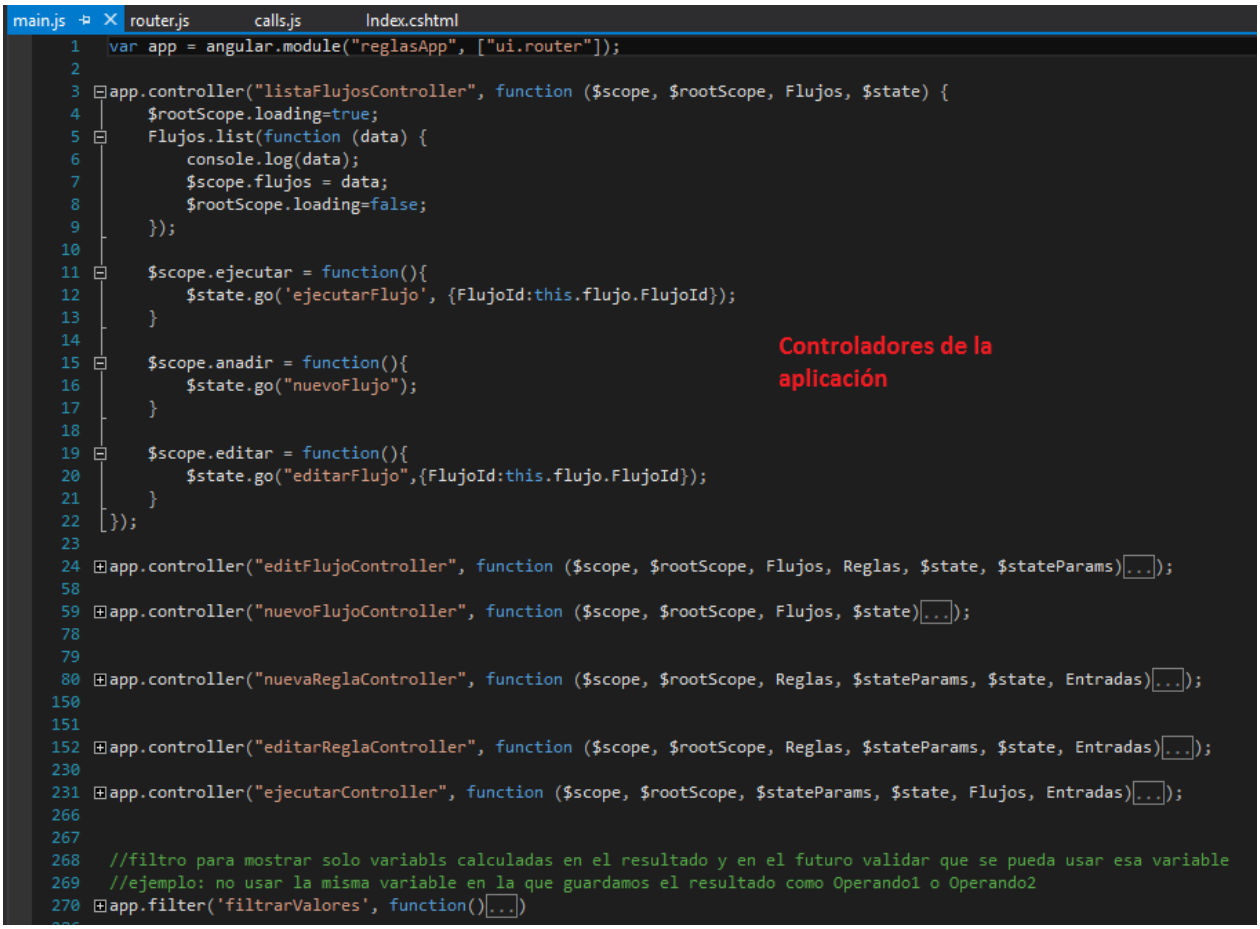


Imagen 40. Código de la tabla de rutas y estados




```
var app = angular.module("reglasApp", ["ui.router"]);

app.controller("listaFlujosController", function ($scope, $rootScope, Flujos, $state) {
    $rootScope.loading=true;
    Flujos.list(function (data) {
        console.log(data);
        $scope.flujos = data;
        $rootScope.loading=false;
    });

    $scope.ejecutar = function(){
        $state.go('ejecutarFlujo', {FlujoId:this.flujo.FlujoId});
    }

    $scope.anadir = function(){
        $state.go("nuevoFlujo");
    }

    $scope.editar = function(){
        $state.go("editarFlujo",{FlujoId:this.flujo.FlujoId});
    }
});

app.controller("editFlujoController", function ($scope, $rootScope, Flujos, Reglas, $state, $stateParams)...);

app.controller("nuevoFlujoController", function ($scope, $rootScope, Flujos, $state)...);


app.controller("nuevaReglaController", function ($scope, $rootScope, Reglas, $stateParams, $state, Entradas)...);


app.controller("editarReglaController", function ($scope, $rootScope, Reglas, $stateParams, $state, Entradas)...);

app.controller("ejecutarController", function ($scope, $rootScope, $stateParams, $state, Flujos, Entradas)...);


 //filtro para mostrar solo variabls calculadas en el resultado y en el futuro validar que se pueda usar esa variable
 //ejemplo: no usar la misma variable en la que guardamos el resultado como Operando1 o Operando2
app.filter('filtrarValores', function()...)
```

Imagen 41. Trozo de código de los controladores (Parte Front)

4.2.4. Testing

Los casos de uso de la aplicación han sido testeados durante todo el desarrollo de la misma además de otros casos de uso secundarios como por ejemplo el de creación de regla, mostrado a continuación.

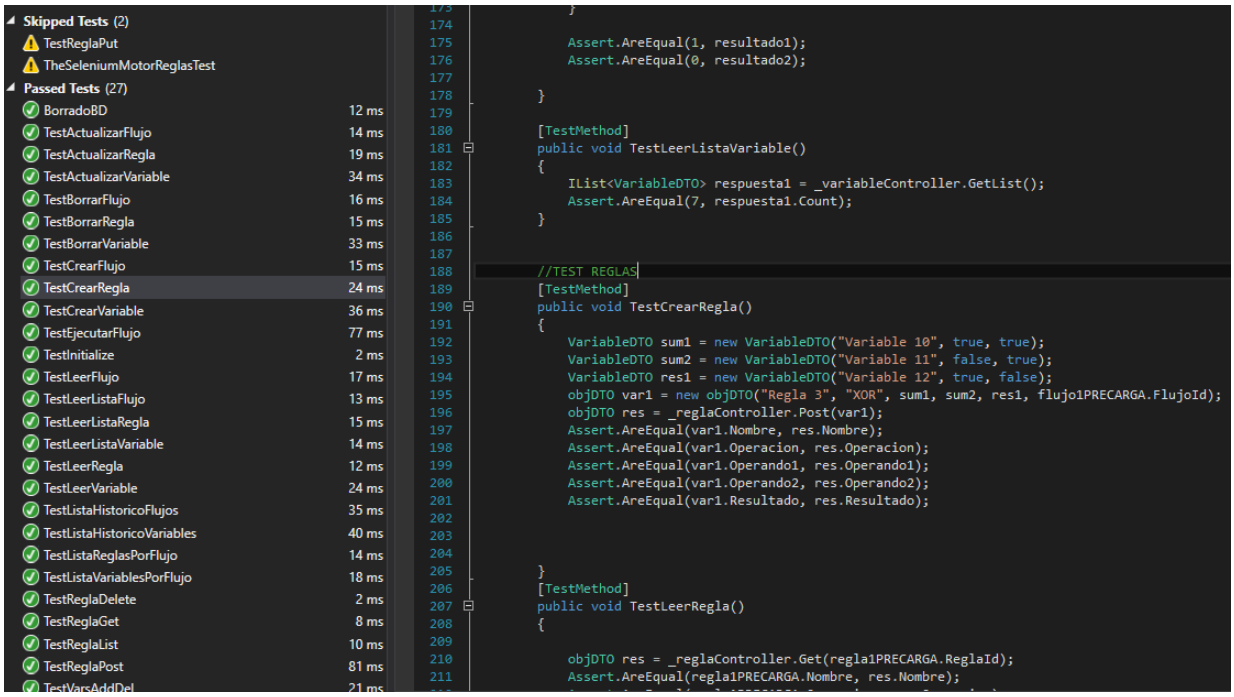


Imagen 42. Trozo de código del testeo de la funcionalidad de creación de regla

4.3. Gestión de Categorias de Atletismo

Se trata también de una aplicación SPA (Simple Page Application) para gestionar los datos correspondientes a las categorías de atletismo. En ella se va a poder crear, borrar, editar, ordenar y listar todas las categorías de atletismo que el usuario desee.

4.3.1. Interfaz

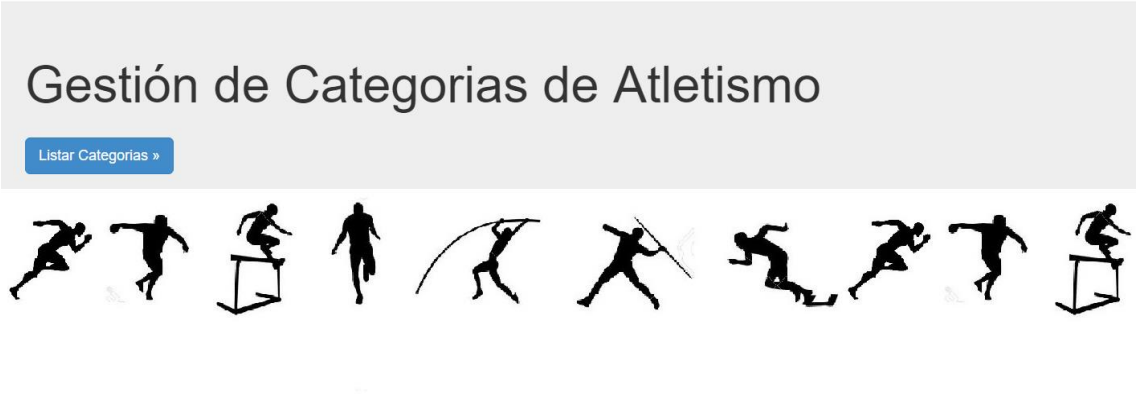


Imagen 43. Pantalla principal de la aplicación

4.3.2. Casos de uso

LISTAR/BORRAR CATEGORIAS

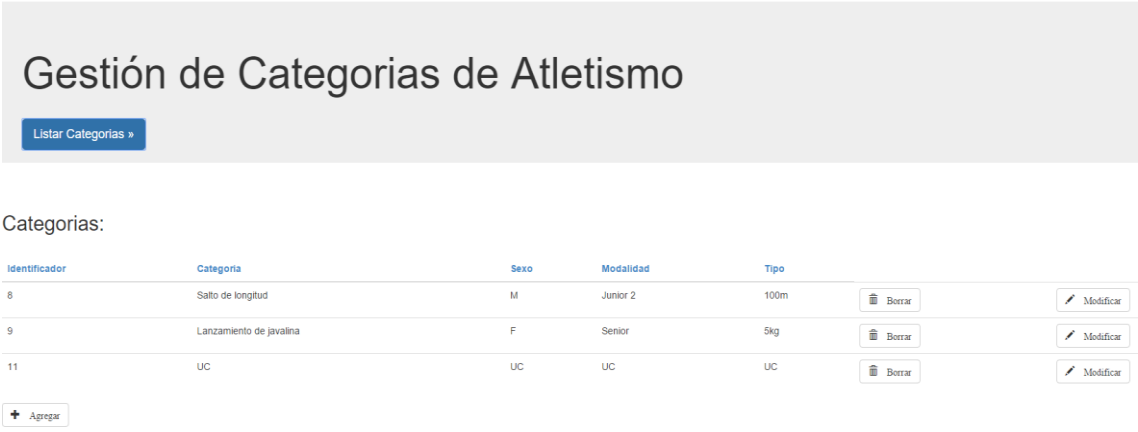


Imagen 44. Pantalla del listado de categorias

CREAR/EDITAR CATEGORIA

Imagen 45. Pantalla de creación de categoría

ORDENAR DATOS

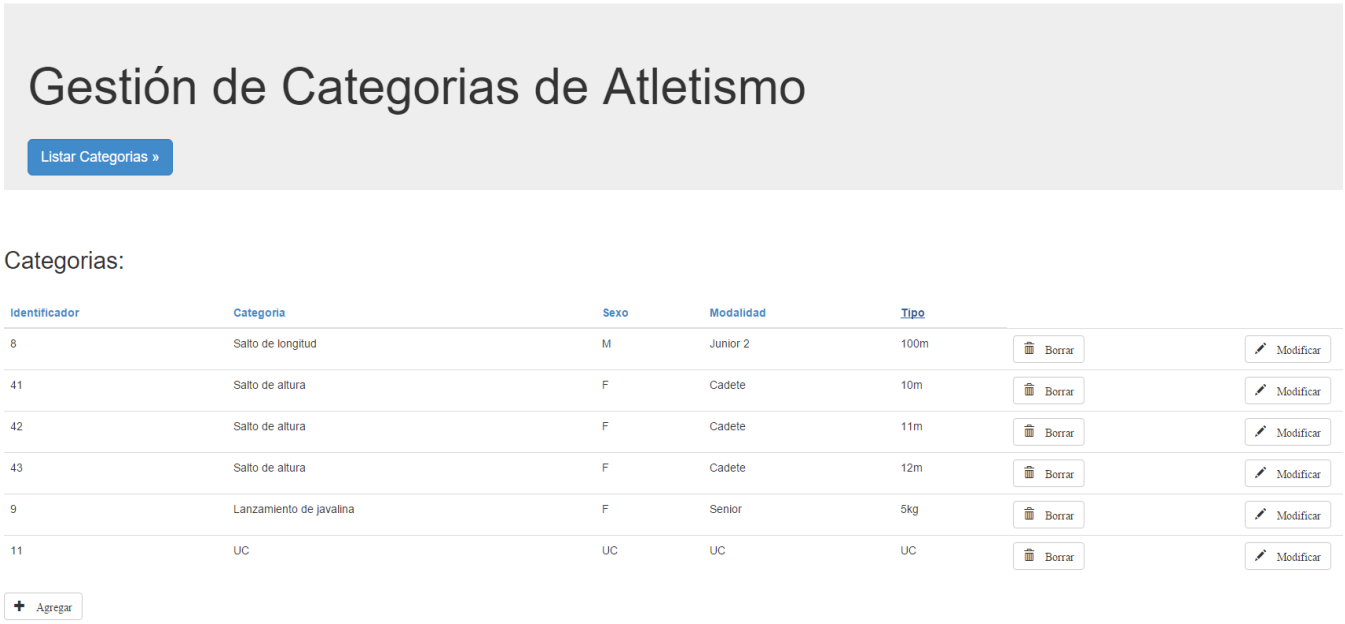
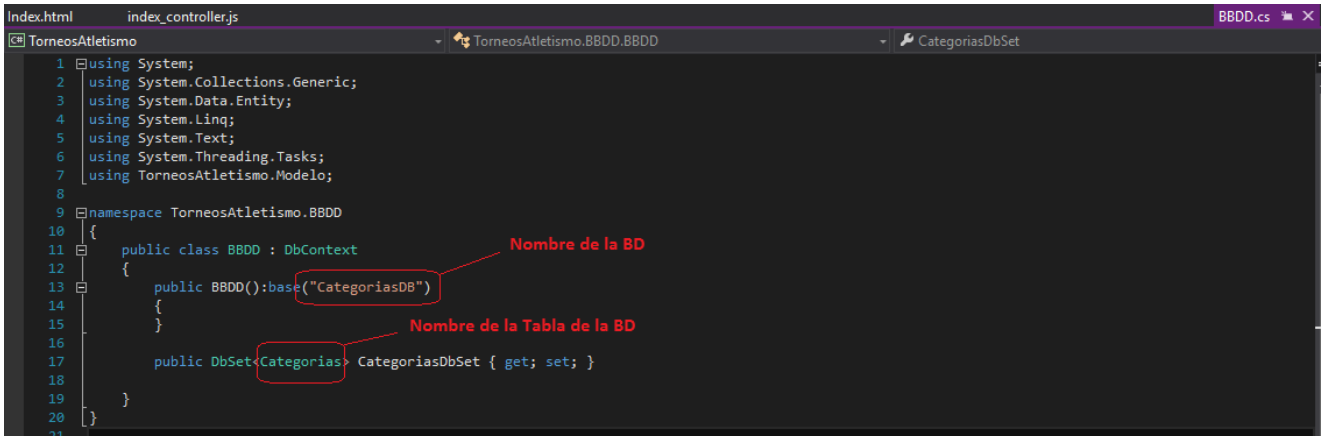


Imagen 46. Pantalla de ordenación del as categorías

4.3.3. Parte BackEnd

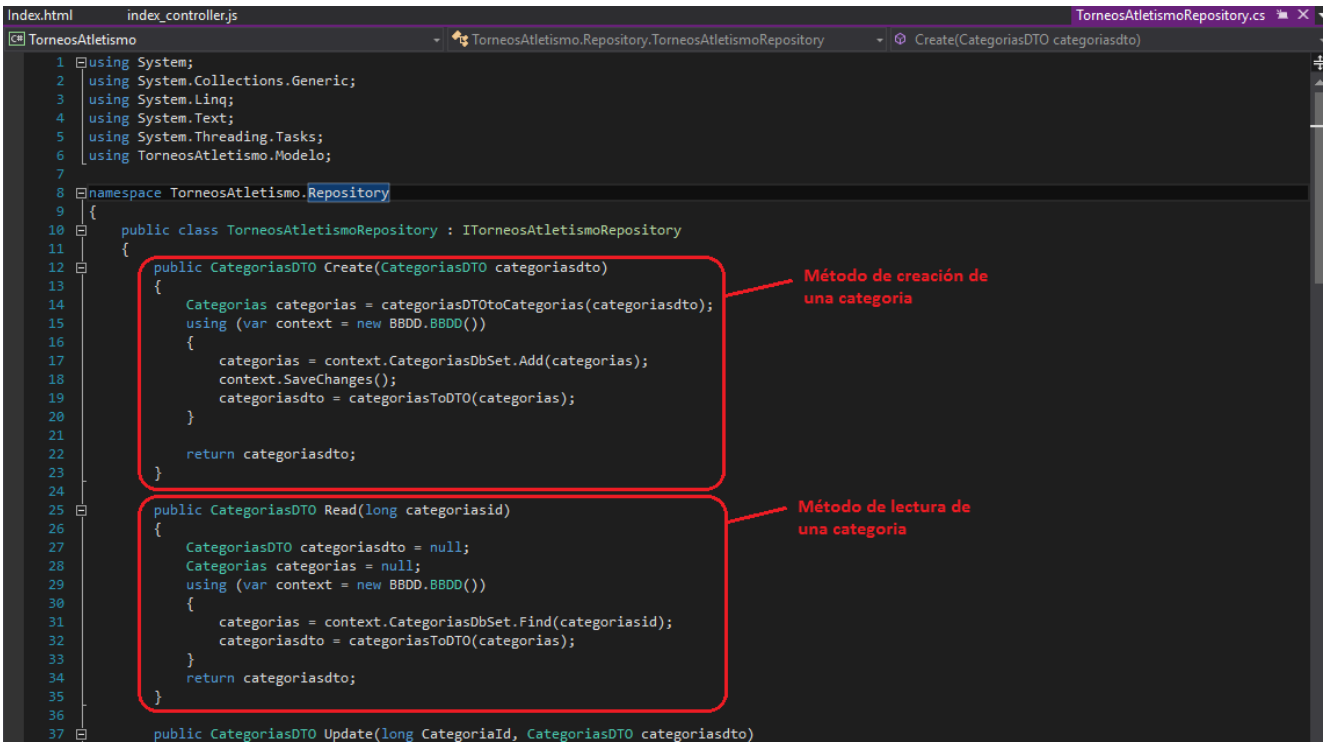
El objeto clase de la propia aplicación: Categorias es inicializado y añadidos a tabla de base de datos mediante las expresiones de Entity Framework ‘DbContext’ y ‘DbSet’, propias de Entity Framework:




```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TorneosAtletismo.Modelo;

namespace TorneosAtletismo.BBDD
{
    public class BBDD : DbContext
    {
        public BBDD():base("CategoriasDB")
        {
        }

        public DbSet<Categorias> CategoriasDbSet { get; set; }

    }
}
```

Imagen 47. Código de inicialización de las tablas de la BD

A su vez, la última capa (Repository) se encarga de registrar las nuevas categorías, leer, listar, actualizar y borrar directamente con la BD.




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TorneosAtletismo.Modelo;

namespace TorneosAtletismo.Repository
{
    public class TorneosAtletismoRepository : ITorneosAtletismoRepository
    {
        public CategoriasDTO Create(CategoriasDTO categoriasdto)
        {
            Categorias categorias = categoriasDTOtoCategorias(categoriasdto);
            using (var context = new BBDD.BBDD())
            {
                categorias = context.CategoriasDbSet.Add(categorias);
                context.SaveChanges();
                categoriasdto = categoriasToDTO(categorias);
            }

            return categoriasdto;
        }

        public CategoriasDTO Read(long categoriasid)
        {
            CategoriasDTO categoriasdto = null;
            Categorias categorias = null;
            using (var context = new BBDD.BBDD())
            {
                categorias = context.CategoriasDbSet.Find(categoriasid);
                categoriasdto = categoriasToDTO(categorias);
            }
            return categoriasdto;
        }

        public CategoriasDTO Update(long CategoriaId, CategoriasDTO categoriasdto)
```

Imagen 48. Trozo de código de la clase TorneosAtletismoRepository (capa R)

En cuanto a la capa intermedia, la de Service, como no hay ninguna lógica de negocio en esta aplicación, su única función va a ser la de hacer de 'bypass', es decir, delegar todo lo recibido a la siguiente capa.

TorneosAtletismoService.cs | Index.html | index_controller.js | TorneosAtletismoRepository.cs

TorneosAtletismo | TorneosAtletismo.Service.TorneosAtletismoService | List()

gestión de categorias | Entire Solution

```
using System.Threading.Tasks;
using TorneosAtletismo.Modelo;
using TorneosAtletismo.Repository;

namespace TorneosAtletismo.Service
{
    public class TorneosAtletismoService : ITorneosAtletismoService, ITorneosAtletismoRepository
    {
        private ITorneosAtletismoRepository _torneoatletismoRepository;

        public TorneosAtletismoService(ITorneosAtletismoRepository torneoatletismorepository)
        {
            _torneoatletismoRepository = torneoatletismorepository;
        }

        public CategoriasDTO Create(CategoriasDTO categoriasdto)
        {
            return _torneoatletismoRepository.Create(categoriasdto);
        }

        public CategoriasDTO Read(long categoriasid)
        {
            return _torneoatletismoRepository.Read(categoriasid);
        }

        public CategoriasDTO Update(long CategoriaId, CategoriasDTO categoriasdto)
        {
            return _torneoatletismoRepository.Update(CategoriaId, categoriasdto);
        }

        public void Delete(long categoriaid)
        {
            _torneoatletismoRepository.Delete(categoriaid);
        }

        public IList<CategoriasDTO> List()
        {
            return _torneoatletismoRepository.List();
        }
    }
}
```

Imagen 49. Código de la clase TorneosAtletismoService (Capa Service)

En cuanto a la primera capa, Controller, como también se ha dicho antes, es la reponsable de recibir las peticiones del exterior, que pasarán por un JavaScript y se conectarán al Controller mediante llamadas AJAX.

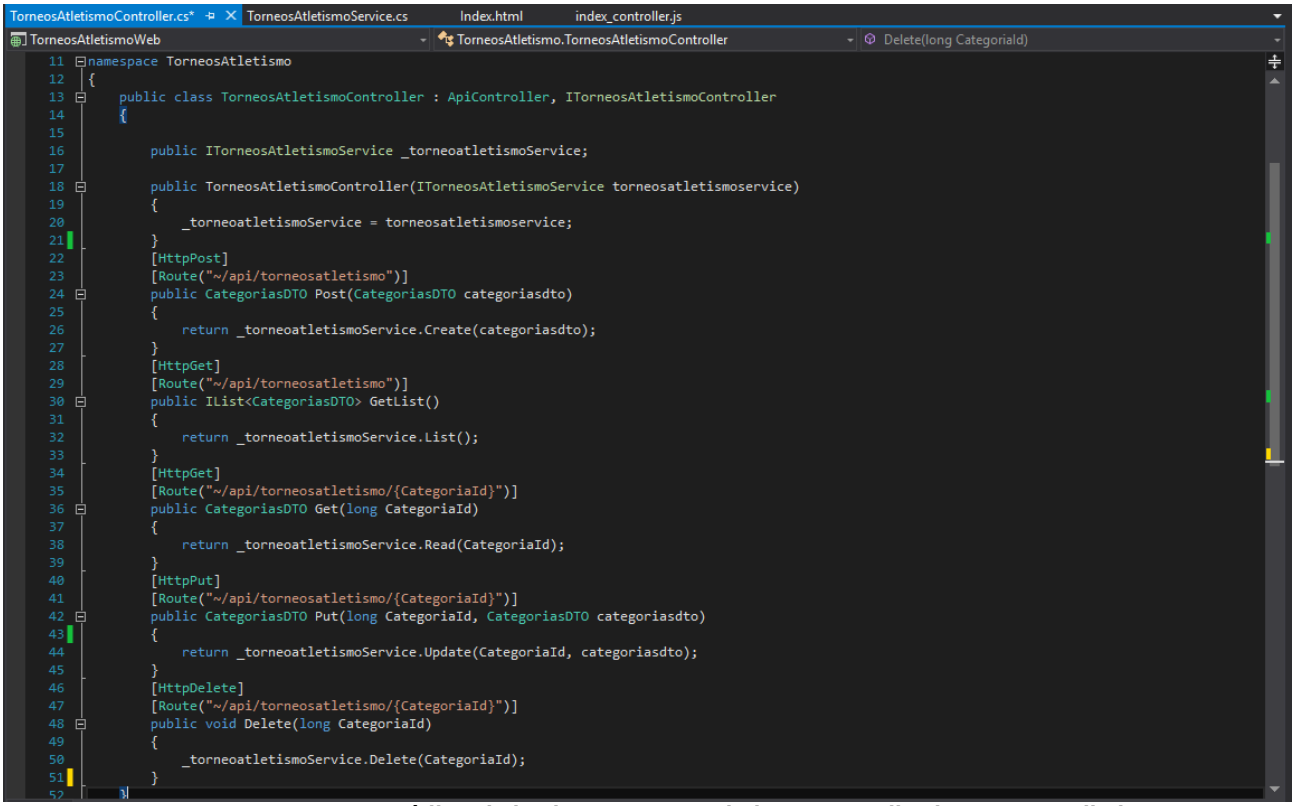
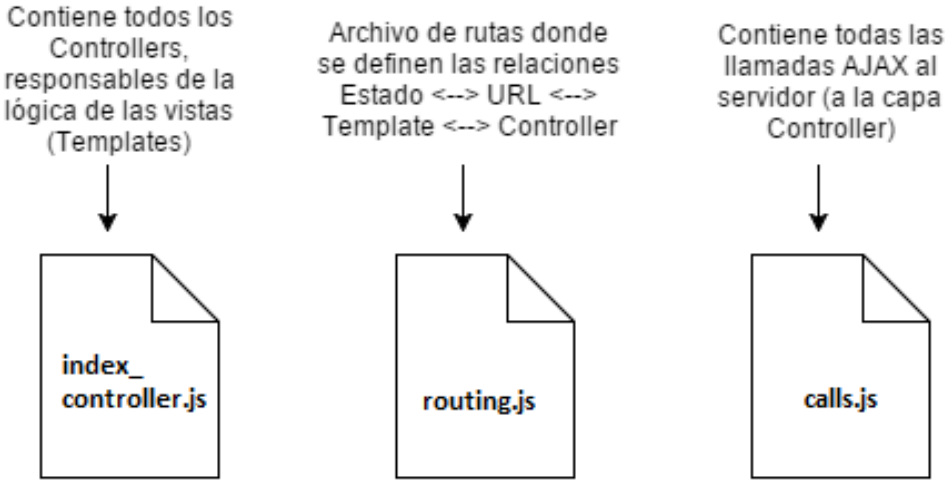


Imagen 50. Código de la clase TorneosAtletismoController (Capa Controller)

4.3.4. Parte FrontEnd

Así como en la aplicación anterior, la parte FrontEnd de este programa está desarrollada mediante el framework AngularJS, y los archivos se han clasificado de la misma manera que en el motor de regas.



routing.js | calls.js | index_controller.js

```
1  appTorneosAtletismo.factory('crud', function ($http) {
2      return {
3          //metodos CRUD
4          lee: function (CategoriaId, callback) {
5              $http.get('api/torneosatletismo/' + CategoriaId).success(callback);
6          },
7
8          list: function (callback) {
9              $http.get('api/torneosatletismo').success(callback);
10         },
11
12         crear: function (callback, categoriasdto) {
13             $http.post('api/torneosatletismo', categoriasdto).success(callback);
14         },
15
16         actualizar: function (CategoriaId, callback, categoriasdto) { //(CategoriaId, callback, categoriasdto)
17             $http.put('api/torneosatletismo/' + CategoriaId, categoriasdto).success(callback);
18         },
19         borrado: function (CategoriaId, callback) { //function(categoriaid, callback)
20             $http.delete('api/torneosatletismo/' + CategoriaId).success(callback); //success(callback)
21         }
22     };
23
24 });
```

Llamadas AJAX responsables de conectar con la capa Controller

Imagen 51. Código del fichero responsable de realizar las llamadas asíncronas AJAX

routing.js* calls.js index_controller.js

```
appTorneosAtletismo.config(function ($stateProvider) {
    $stateProvider
        .state('indexstate', {
            url: "",
            views: {
                "vista_index": {
                    templateUrl: "Index.html",
                    controller: "indexController"
                }
            }
        })
      .state('liststate', {
          url: "/list",
          views: {
              "vista_listado": {
                  templateUrl: "Templates/listaTorneosAtletismo.html",
                  controller: "indexController"
              }

          }
      })
         .state('liststate.addliststate', {
             url: "/addlist",
             views: {
                 //OJO LA @ => dependencias de vistas/estados para q me cargue cuando aprete el boton
                 "liststate@vista_addlistado": {
                     templateUrl: "Templates/addlistaTorneosAtletismo.html",
                     controller: "indexController"
                 }
             }
         })
     .state('liststate.updtliststate', {
         url: "/updtlist/{CategoriaId}",
         views: {
             //OJO LA @ => dependencias de vistas/estados para q me cargue cuando aprete el boton
             "liststate@vista_modificarlistado": {
                 templateUrl: "Templates/updtlistaTorneosAtletismo.html",
                 controller: "ModificarController"
             }
         }
     })
});
```

Estados y rutas de la aplicación

Imagen 51. Código de la tabla de estados

routing.js* calls.js index_controller.js*

```
1  var appTorneosAtletismo = angular.module('appTorneosAtletismo', ['ui.router']);
2
3  appTorneosAtletismo.controller('indexController', function ($scope, crud, $stateParams, $state) {
4      $scope.cargandomsg = "Cargando, por favor espere...";
5
6      // borrar
7      $scope.Borrar = function (listado)...;
19
20     // listar
21     crud.list(function (crud)...);
28
29     // crear
30     var categoriasdto;
31     $scope.creaCategoria = function (listado) {
32         $scope.state_action = false;
33         categoriasdto = {
34
35             //recogo valores de los inputs de addlistaTorneosAtletismo.html
36             categoria: $scope.categoria,
37             sexo: $scope.sexo,
38             modalidad: $scope.modalidad,
39             tipo: $scope.tipo
40         };
41             crud.crear(function (data) {
42
43                 alert("Elemento agregado correctamente");
44                 $state.go("liststate", {}, { reload: true });
45             }, categoriasdto);
46     };
47
48     // modificar
49     var categoriasdto;
50
51     $scope.modificaCategoria = function (categoriasdto)...;
58
59 });
60
61 appTorneosAtletismo.controller('ModificarController', function ($scope, crud, $stateParams, $state)...);
82
```

Controladores de la aplicación

Imagen 52. Trozo de código de los controladores de la aplicación

4.4.4. Testing

Test Explorer

Run All | Run... | Playlist : All Tests

Passed Tests (11)

Delete	< 1 ms
Get	5 ms
GetById	< 1 ms
Index	80 ms
Post	< 1 ms
Put	< 1 ms
TestCreate_TorneosAtletismoRepository	4 sec
TestDelete_TorneosAtletismoRepository	7 ms
TestList_TorneosAtletismoRepository	10 ms
TestRead_TorneosAtletismoRepository	13 ms
TestUpdate_TorneosAtletismoRepository	16 ms

routing.js calls.js index_controller.js TorneosAtletismoRepositoryTest.cs

TorneosAtletismoTest | TorneosAtletismoTest.TorneosAtletismoRepositoryTest | TestCreate_TorneosAtletismoRepository()

```
1  using System;
2  using Microsoft.VisualStudio.TestTools.UnitTesting;
3  using TorneosAtletismo;
4  using TorneosAtletismo.Repository;
5  using TorneosAtletismo.Modelo;
6
7
8  namespace TorneosAtletismoTest
9  {
10     [TestClass]
11     public class TorneosAtletismoRepositoryTest
12     {
13
14         private TorneosAtletismoRepository repo;
15         [TestInitialize()]
16         public void MyTestInitialize()
17         {
18             repo = new TorneosAtletismoRepository();
19         }
20
21         [TestMethod]
22         public void TestCreate_TorneosAtletismoRepository()
23         {
24             CategoriasDTO cat1 = new CategoriasDTO("benjamin", "masculino", "carrera de velocidad", "60 metros" );
25
26             CategoriasDTO creacat1 = repo.Create(cat1);
27
28             Assert.IsNotNull(creacat1.CategoriaId, "el ID se incrementa cuando se realiza una creacion");
29
30             repo.Delete(creacat1.CategoriaId);
31         }
32
33         [TestMethod]
```

Imagen 53. Trozo de código del testeo de la funcionalidad de creación de categoría

5. Futuras mejoras y nuevas funcionalidades

Durante el proceso de identificación de las necesidades a cubrir por la aplicación, la decisión sobre el formato de las diferentes interfaces, nuevos casos de uso y otras metodologías de desarrollo, se barajaron varias posibilidades que por diversas razones no han sido llevadas a cabo aún. Algunas de estas ideas han sido descartadas y no llegarán a realizarse pero varias de ellas podrían resultar útiles. Estas mejoras, novedades o nuevas versiones son las siguientes:

5.1. Subir la aplicación a un web hosting

A efectos prácticos, es mucho más sencillo acceder a una aplicación web mediante una simple URL que mediante Visual Studio. Para ello, es necesario un lugar donde alojarla, ese "lugar" (espacio web) nos puede ofrecer un servidor de Hosting (alojamiento de páginas web) bien de forma gratuita o bien pagando cierta cantidad. Por otro lado, una vez tenemos el alojamiento web para nuestra página, los visitantes necesitan escribir algo en sus navegadores para dirigirse a nuestra nueva web. Esto es la URL o dirección de la web, que puede ser o bien un dominio elegido (y pagado) por ti, como por ejemplo www.mipfc.com o bien un subdominio (gratuito) que tienen una forma algo más larga por ejemplo www.mipfc.eu.org o www.mipfc.tk.
Un lugar muy popular tanto para alojar páginas web como para comprar dominios es www.arvixe.com. Como las aplicaciones de este proyecto se han realizado con Windows y ASP.NET, elegiremos un servidor compatible: http://www.arvixe.com/asp_net_web_hosting y por apenas 4 dolares al mes tendremos un hosting dedicado que además nos proporcionará un dominio elegido por nosotros.
Una vez obtenidos host y dominio, es necesario subir los archivos de nuestra aplicación al servidor alquilado, para ello hace falta un cliente FTP como por ejemplo FileZilla(gratuito):

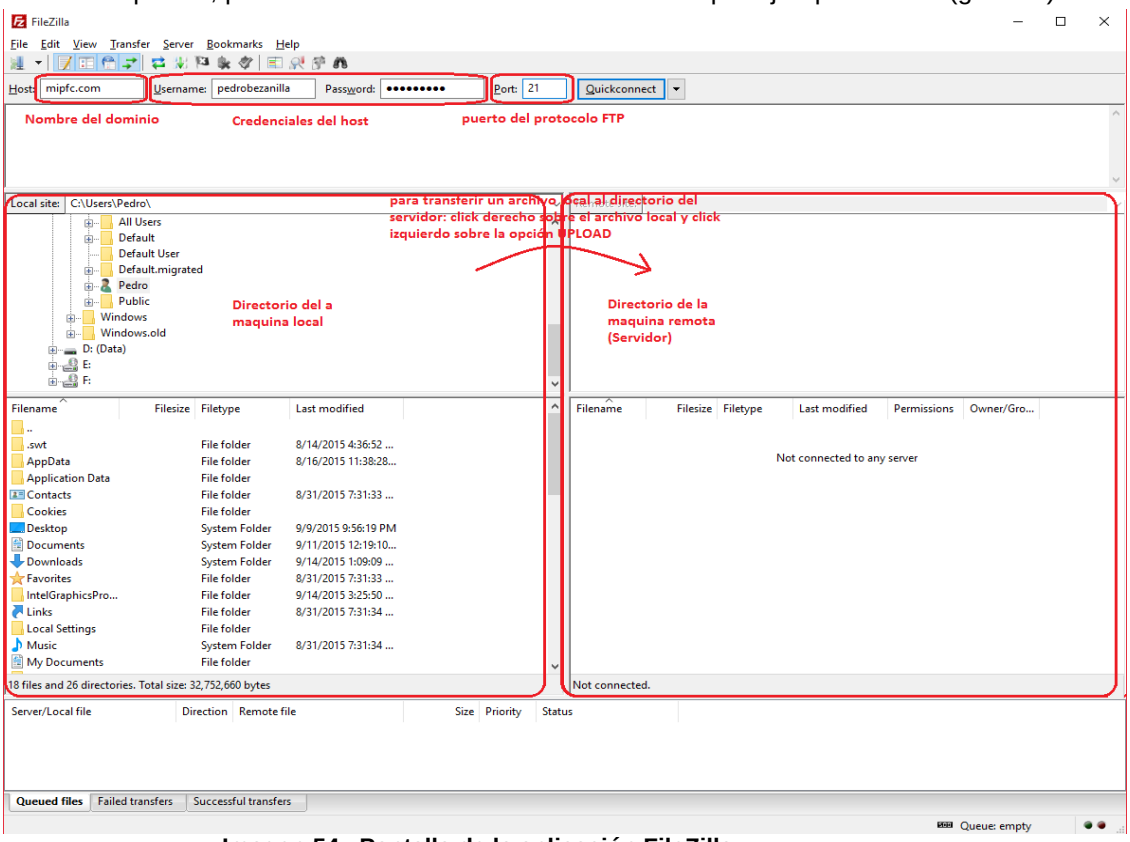


Imagen 54. Pantalla de la aplicación FileZilla

5.2. Nuevos casos de uso

5.2.1. Gestión de cines

El caso de uso “Gestión de totales” está incompleto y faltaría poder calcular el dinero recaudado a lo largo de un dia, mes o año para 1 o más sesiones.

También sería interesante añadir una imagen de la película vigente en ese momento junto a la sesión elegida.

5.2.2. Motor de reglas

Se podrían haber añadido nuevas reglas como por ejemplo puertas NOR, NAND, XOR, INVERSORES, etc. pues únicamente se evalúan las reglas AND y OR.

También se podría haber mejorado el aspecto de la aplicación, cuidar mejor el diseño, hacerlo más intuitivo para el usuario y añadir animaciones por ejemplo entre las transiciones de página.

5.2.3. Gestión de categorias de atletismo

Además de gestionar las categorías, sería interesante poder gestionar las modalidades, torneos vigentes, filtrado de datos, etc.

5.2.4. Control de autenticación

Se podría haber añadido también una etapa de autenticación en la capa Controller

7. Conclusiones

Durante la finalización del último curso de Ingeniería Superior de Telecomunicaciones, mis pasos profesionales se encaminaron hacia el campo de la programación trabajando en la empresa privada, lugar en el que me encuentro en la actualidad.

En estos pocos meses dedicándome al desarrollo he trabajado sobre algunos proyectos bastante prácticos y he querido plasmar en este proyecto algunos conceptos del entorno de la empresa privada.

Este desarrollo me ha permitido llevar a cabo ese interés personal por la programación de aplicaciones y ampliar mis conocimientos como programador al tratar con varias plataformas, utilizar diferentes lenguajes y Frameworks y manejar distintos entornos de desarrollo. Los principales puntos en los que realizar este proyecto me ha permitido mejorar como profesional y cubrir tareas en las que no me había visto inmerso son los siguientes

Ha sido mi primera incursión con el framework AngularJS. Para ello, he tenido que ver algunos video tutoriales sobre conceptos básicos del framework y

He podido profundizar en el lenguaje Java, algo en lo que también tenía gran interés ya que es bastante solicitado por las empresas al realizar ofertas de empleo. Aunque ya lo conocía anteriormente, con él simplemente había realizado pequeños programas de prueba llevado por la curiosidad de aprender un lenguaje nuevo. Este desarrollo en Java me ha permitido además familiarizarme con el entorno de programación Eclipse, que no lo había utilizado ya que las prácticas y pruebas anteriores los había llevado a cabo utilizando NetBeans.

Desde el punto de vista de la planificación del proyecto, en los desarrollos en los que he intervenido en mi carrera profesional, aunque si he realizado planificaciones y algunas tomas de requisitos puntuales, no había tenido la oportunidad de realizar un estudio de las necesidades existentes y el modo de resolverlas desde cero. No me había sido posible estar desde el inicio en el proceso de plantear ideas, observar otras aplicaciones del estilo que pudiesen aportar funcionalidades interesantes o participar en reuniones en las que se definan los requisitos...

Este proyecto, al haberse creado desde la simple idea de mostrar unas noticias, me ha permitido realizar todo el proceso de definición de requisitos, análisis y desarrollo del mismo, cosa que considero de vital importancia para el crecimiento profesional dentro de la rama de la programación, en la que estoy en estos momentos.

Otro de los puntos novedosos para mí está en la futura versión para Iphone y Ipad. Aunque esta versión de la aplicación no entra dentro del alcance de este proyecto ya que no estaba previsto desarrollarla en un futuro inmediato, desde el inicio se tuvo en mente la importancia de realizarla debido al gran peso que estos dos dispositivos tienen en el mercado. En la actualidad he comenzado su elaboración y que me está sirviendo para familiarízame con otro entorno de desarrollo, el Xcode y un nuevo lenguaje que tampoco había utilizado hasta ahora, Objective-C.

8. Referencias

[1] «Introduction to AngularjJS in 50 Examples» Agosto 2015. [En línea]. Disponible en: https://www.youtube.com/watch?v=TRrL5j3MIvo o https://github.com/curran/screencasts/tree/gh-pages/introToAngular [Último acceso: 5 Septiembre 2015].

[2] «Historia del desarrollo de aplicaciones Web» Agosto 2015. [En línea]. Disponible en: http://www.um.es/docencia/barzana/DIVULGACION/INFORMATICA/Historia-desarrollo-aplicaciones-web.html [Último acceso: 15 Agosto 2015].

[3] «Learn and understand AngularJS» [En línea]. Disponible en: https://www.youtube.com/watch?v=ejBkOjEG6F0 [Último acceso: 20 Agosto 2015].

[4] «Antipatrón de diseño» [En línea]. Disponible en: https://es.wikipedia.org/wiki/Antipatr%C3%B3n_de_dise%C3%B1o [Último acceso: 19 Agosto 2015].

[5] «Patrón de arquitectura Modelo Vista Controlador (MVC)» [En línea]. Disponible en: http://www.lab.inf.uc3m.es/~a0080802/RAI/mvc.html [Último acceso: 19 Agosto 2015].

[6] «Hosting y Dominios» 15 Septiembre 2015. [En línea]. Disponible en: http://www.comocreartuweb.com/hosting-y-dominios.html [Último acceso: 15 Septiembre 2015].

[7] «Apuntes del Curso en Tecnologías .NET en frameworks de programación y metodologías propios de CIC para dispositivos móviles impartido por Consulting Informático de Cantabria S.L. a través del EMCAN (Servicio Cántabro de Empleo)» [No disponible en línea]. [Último acceso: 23 Septiembre 2015].