UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA EN SISTEMAS AUDIOVISUALES Y MULTIMEDIA

Curso Académico 2019/2020

Trabajo Fin de Grado

GLOBAL TREE: APLICACIÓN WEB PARA LA GESTIÓN DEL TELETRABAJO, BASADA EN .NET

Autor : Fernando Castro López

Tutor : Dr. Gregorio Robles

Trabajo Fin de Grado

GLOBAL TREE:
APLICACIÓN WEB PARA LA GESTIÓN DEL TELETRABAJO BASADA EN .NET

Autor : Fernando Castro López

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2020, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2020

Dedicado a

mis padres y a mi pareja.

II

Agradecimientos

Quiero agradecer a toda mi familia la ayuda que me han prestado en todo momento, en especial a mi madre que siempre me apoyó, me motivó y me ayudó en los momentos donde las fuerzas de uno le desaparecen y le piden abandonar. A mi padre por calmarme cuando más tensión tenía y quitar importancia en los momentos de más estrés, y que juntos son una pareja perfecta de la que siempre me sentiré muy orgullo, aunque nunca tendré palabras para agradecer todo lo que han significado, significan y significarán.

A mi pareja le debo todo, gracias por el apoyo, gracias por tu paciencia, gracias por ponerte en mi lugar tantas veces y levantarme el ánimo y gracias por ser la valiente que me hace arriesgarme y me da fuerzas en las decisiones importantes.

A mis compañeros, sé que no ha sido fácil muchas veces, pero los pocos que se han acercado a mi, se han convertido en compañeros de por vida.

Y a mi tutor, que me ayudó a hacer este proyecto sabiendo que la situación actual era muy complicada, causada por el COVID-19.

Resumen

Global Tree es una aplicación web de gestión de recursos para la organización interna de una empresa de tipo teletrabajo. El objetivo principal por el que se ha diseñado es para dar respuesta a la gestión remota de sus empleados y proyectos, permitiendo controlar y optimizar el tiempo empleado, y garantizando una mayor eficiencia en futuros trabajos similares.

Para el desarrollo de esta aplicación web se ha utilizado el entorno de trabajo de Visual Studio 2019 Community, que pertenece al grupo de herramientas de .NET. Además, se han añadido varios paquetes que automatizan procesos complejos, como Entity Framework, que agiliza el proceso de gestionar una base de datos de manera significativa. El lenguaje de programación utilizado para el *back end* es C#. Para la base de datos se ha utilizado Microsoft SQL Server 2014 Express, y en cuanto al *front end* las tecnologías utilizadas han sido HTML5, Bootstrap y JavaScript.

La pandemia global causada por el COVID-19 ha puesto de manifiesto la necesidad por parte de muchas empresas de que el teletrabajo sea la base del funcionamiento de estas. Si se ejecuta de manera correcta puede llegar a ahorrar grandes sumas de dinero. Sin embargo, su implantación puede suponer un gran reto. Con esta aplicación se pretende integrar todos los mecanismos necesarios para garantizar una correcta gestión de una empresa tipo.

Summary

Global Tree is a resource management web application for the internal organization of a telecommuting company. The main purpose for which it has been designed is to be able to remote management of its employees and projects, enabling to control and modify the time spent, and guaranteeing greater efficiency in future similar jobs.

For the development of this web application it has been used the Visual Studio 2019 Community working environment, which belongs to the .NET tool group. In addition, several packages that automates complex processes have been added, such as Entity Framework, which streamlines the process of managing a database in a very significant way. The programming language used for the back end is C#. Microsoft SQL Server 2014 Express has been used for the database, and HTML5, Bootstrap and JavaScript technologies have been used for the front end. fmáxi The global pandemic caused by COVID-19 has highlighted the need of many companies for teleworking to be the basis of their operation. If done correctly it can save large sums of money, but nevertheless its implementation can be a great challenge. With this application, the goal is to integrate all the necessary mechanisms to guarantee the correct management of a typical company.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Presentación de la aplicación

Global Tree es un proyecto para la organización interna de los recursos globales de una empresa que se ha diseñado para dar respuesta a la gestión remota de sus empleados. Esta aplicación web permite la administración de los trabajadores a través de un formulario de registro que servirá para controlar la incorporación y las bajas de personal. Además, se pueden asignar vacaciones y permisos. En cuanto a la organización del trabajo ofrece la posibilidad de iniciar y finalizar proyectos, dar de alta a los clientes y calibrar el tiempo dedicado a cada tarea. La aplicación permite el control y seguimiento de las horas invertidas en cada proyecto, lo que optimizará el tiempo empleado y garantizará una mayor eficiencia en los futuros trabajos de similares características, además de controlar las posibles incidencias de los trabajadores y el uso adecuado del tiempo destinado. Esta tarea se ve reflejada en un calendario de eventos donde cada trabajador asigna las labores realizadas y el tiempo invertido cada día. La aplicación ofrece resúmenes estadísticos con gráficas que recogen las horas trabajadas al mes, lo que permite controlar el ritmo de cada trabajador y la asignación del salario correspondiente. También evalúa la eficacia del trabajo en equipo entre varios trabajadores vinculados a un mismo proyecto. Por último, se han definido dos perfiles o roles: uno para la gestión de toda la plataforma (que será el administrador) y el otro para el usuario común (el empleado).

1.2. Contexto personal

Como Alumno del Grado en Ingeniería de Sistemas Audiovisuales y Multimedia (ISAM), he de reconocer que este trabajo se aleja algo de los conocimientos adquiridos durante mi grado. A pesar de que a lo largo de los cursos he abordado diferentes materias, siendo el campo de la creación de software y las aplicaciones web unas de las que menos carga lectiva he tenido, he de decir que desde el principio fueron las que me suscitaron mayor interés.

La realización de las prácticas en empresa en una pequeña empresa de desarrollo de software me ilusionó mucho en un principio, ya que como he mencionado antes este campo me llamaba mucho la atención. Pero al mismo tiempo supuso un enorme reto, ya que me obligó a tener que aprender mucho por mi cuenta y a marchas forzadas, puesto que en la empresa no había ni tiempo ni personal para ayudarme en mi formación.

Por lo tanto, pensé que el trabajo de fin de grado era una ocasión perfecta para seguir desarrollando ese proceso de autoaprendizaje en un proyecto que me obligase a integrar todos esos conocimientos adquiridos, profundizar en ellos y sumarle nuevos.

1.3. Análisis de entorno

A partir de la reforma del mercado laboral se comienza a regular de forma jurídica el trabajo a distancia o teletrabajo en el Artículo 13 del Estatuto de los Trabajadores de 2015 [3].
”El teletrabajo puede definirse como una forma de organización y/o ejecución del trabajo realizado a distancia, en gran parte o principalmente, mediante el uso intensivo de las técnicas informáticas y/o de telecomunicación”.

En nuestro país no es una forma de trabajo muy extendida, pero hoy en día resulta imprescindible ya que la pandemia causada por el COVID-19 ha provocado que sea la única vía para que muchas empresas no dejen de funcionar. Por lo tanto, esta manera de trabajar se ha tenido que instaurar de forma acelerada y en muchos casos sin control. Pero será la manera de trabajar a partir de estos momentos de muchos trabajadores y seguramente el medio que muchas empresas van a adoptar de ahora en adelante, ya que puede llegar a ahorrar grandes sumas de dinero si se ejecuta de manera correcta.

Este tipo de trabajo se realiza en un lugar distinto y alejado desde donde se solicita, y por

este motivo es difícil hacer un seguimiento y control de todos los procesos y tareas necesarias para lograr su consecución. Además, en muchas ocasiones hace falta transportarlo o, como ocurre en la mayoría de los casos, resulta imprescindible utilizar las nuevas tecnologías para su desarrollo. Esto supone organizarse de manera diferente a los modelos tradicionales.

Esta forma de trabajo exige considerar el tiempo que se emplea en desarrollar las tareas, la manera en la que se trabaja y controlar la producción final que se obtenga para garantizar la calidad del producto.

Javier Thibault Aranda [8] definió dos formas *a priori* de realizar el teletrabajo en función de la conexión a la red en cada momento en su trabajo de tesis doctoral:

a) Off-line o desconectado, donde se realiza el trabajo sin ninguna clase de enlace informático directamente con la empresa y el producto del trabajo se debe enviar físicamente a su destino a través de alguna empresa de mensajería.

b) On-line o conectado, donde se establece una conexión permanente o temporal y puede ser del lado del trabajador hacia el que encarga el trabajo o con comunicación bidireccional.

Para regular este tipo de trabajos, muchas veces realizados en el domicilio de las personas que los desarrollan, viene siendo necesario el empleo por parte de las empresas de programas que regulen estos servicios.
Existen en el mercado programas y aplicaciones que permiten aplicar la tecnología para gestionar mejor las tareas de los trabajadores como[1]:

- **Qlik Sense:** es una aplicación de visualización y descubrimiento de datos gobernada, basada en servidor, se adapta muy bien a las necesidades analíticas de grupos, departamentos o toda una organización. Es una herramienta con una carga estadística muy amplia que podría complementar *Global Tree* y viceversa.

- **Trello:** con esta aplicación se pueden crear tarjetas para organizar las tareas y ordenarlas en listas: con el clásico ≪To do≫, ≪Doing≫, ≪Done≫ se puede desplazar las tarjetas según el proceso, así como asignar responsables y fechas de entrega o crear checklists.

- **Global Tree**, la aplicación que se presenta en esta memoria, no asigna tareas por hacer, pero sí hechas y su mayor diferencia es el control horario de estas.

[1] `https://www.sesametime.com/es/las-6-herramientas-que-necesitas-para-teletrabajar/`

- **Slack:** es una herramienta que facilita la comunicación interna de la empresa para conocer al equipo con el que se trabaja, aunque no se esté presencialmente en la oficina. Esta herramienta integra correo electrónico, mensajería instantánea y otras aplicaciones como Skype, para que sea un punto de encuentro entre los trabajadores que pueden organizarse mediante grupos en los distintos apartados. Sería interesante añadir alguna de estas funcionalidades a *Global Tree* para mejorar la comunicación y así no depender de aplicaciones de terceros.

- **Sesame:** esta herramienta permite tener cierto control de horario para saber cuanto tiempo dedican los trabajadores a su jornada. Con un sistema de geolocalización, el trabajador puede realizar fichajes desde su propio móvil esté donde esté y así contabilizar su tiempo. Esta aplicación es, a día de hoy, la más completa y versátil, con múltiples funcionalidades; su finalidad es agilizar los procesos de una empresa presencial.

1.4. Estructura de la memoria

En esta sección se debería introducir la estructura de la memoria.
Así:

- En este primer capítulo se hace una introducción de la aplicación, se presenta el análisis del entorno y el contexto personal que me llevó a realizar este proyecto.

- En el capítulo 2 se muestran tanto el objetivo principal como los objetivos específicos que lo componen y una planificación temporal del proyecto.

- A continuación se presenta el estado del arte en el capítulo 3, donde se explican el concepto Modelo-Vista-Controlador (MVC) y las tecnologías y herramientas que he necesita en el desarrollo de la aplicación.

- El capítulo 4 está compuesto por la parte principal del proyecto, donde mostramos la estructura de la aplicación y explicamos su partes más importantes.

- Para finalizar en el capítulo 5 se ofrecen las conclusiones a las que me ha llevado este proyecto y posibles mejoras en la aplicación web.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo general de este proyecto es crear una aplicación que permita integrar todas las herramientas necesarias para que empresas, especialmente las que implementa total o parcialmente el teletrabajo, puedan agilizar sus procesos de gestión y administración.

2.2. Objetivos específicos

Para la consecución del objetivo general se han planteado los siguientes objetivos específicos:

- Mostrar la información vital de la empresa de manera clara y concisa. Esto es: creación de tablas de usuarios, proyectos y los clientes, y sus respectivas asociaciones. Esto permite controlar y manipular dicha información, la cual ayudará a una correcta organización de los recursos de la empresa.

- Facilitar su uso para que sea sencilla e intuitiva para que todos los trabajadores puedan entender y comprender su funcionamiento.

- Ser accesible desde cualquier ubicación, ya que sólo con la posibilidad de tener Internet podemos gestionar su contenido, y tener un diseño *responsive* lo que la hará más versátil puesto que desde cualquier *tablet* o dispositivo móvil se podrá utilizar.

- Gestionar los tiempos de trabajo mediante un calendario que contenga tanto tareas pasadas como futuras.

- Controlar el tiempo empleado por todos los trabajadores en cada tarea, función que a través de un sistema de roles sólo podrá ejecutar el administrador. De aquí se podrán extraer estadísticas útiles para poder extrapolar y organizar proyectos de características similares.

- Establecer algún tipo de herramienta de seguridad que proteja los datos de la empresa, mediante el uso de un usuario propio de cada empleado y una contraseña cifrada que se guardará en la base de datos.

2.3. Planificación temporal

La primera idea que tuve para realizar este Trabajo Fin de Grado (TFG) fue en octubre de 2017, cuando ya llevaba 4 meses de prácticas en empresa. Pero fue el año pasado cuando realmente empecé a dar forma a mi idea. Era difícil establecer un horario para dedicarle al TFG cuando trabajaba y tenía que estudiar las dos asignaturas de la carrera que me quedaban, pero el tiempo libre restante se lo dedicaba al proyecto. Durante este período se intercalaban semanas en las que ese tiempo era nulo, con semanas en las que la dedicación era total. Sin embargo, el problema después de una temporada sin trabajar en él era que tenía que invertir mucho tiempo en recordar lo hecho hasta ese momento. A continuación, detallaré todas aquellas fases en las que fui desarrollando el proyecto y en qué consistieron. En tiempo real le he dedicado unas 4 horas diarias todos los días menos los festivos.

- Primero diseñé la aplicación de manera muy esquemática, y empecé con el primer modelo de base de datos con sus campos y sus conexiones, la cual modificaría en múltiples ocasiones. Esta tarea me llevó un mes realizarla y a partir de ahí cree la plantilla MVC con la ayuda de Visual Studio. Una semana estuve estudiando e instalando el Entity Framework para conectar desde el inicio mi proyecto con la base de datos creada.

- Era hora del estilo de los *layout* de mi aplicación y la instalación del *bundle* de Bootstrap y el diseño inicial de la web, lo que me llevó 2 semanas.

- Tardé un mes más en crear los formularios de entrada, de edición y de eliminación de las tres tablas: usuarios, proyectos y clientes. Las dos primeras semanas las pasé diseñando y creando las funciones de la tabla de usuarios y las dos siguientes las otras dos, cada una con sus peculiaridades.

- La siguiente semana la dediqué a la realización de la parte de las imputaciones y su gráfica de barras dividida en semanas.

- Estuve una semana más creando la seguridad mediante el cifrado de los datos de entrada y de las cookies.

- Por último, el control de errores y pruebas de funcionamiento de la aplicación, así como ultimar detalles de diseño, fue el proceso más arduo y largo. Su concreción en tiempo es muy difícil de especificar ya que ha sido un proceso continuo desde la versión primera de la aplicación. Por ejemplo, llegué a cambiar y modificar parámetros de la base de datos, lo que supuso tener que retocar gran parte de la aplicación.

- Después de todo esto quedaba la tarea de trasladar al papel toda la aplicación. Aquí encontré varias dificultades. El hecho de no estar guiado por un tutor a lo largo de todo el proyecto, fue una dificultad en sí misma, tanto en la elaboración de la aplicación como sobre todo a la hora de redactar el trabajo escrito. Durante este periodo no ha sido fácil encontrar un tutor que realizara un seguimiento a mi trabajo. Sin embargo finalmente, al contactar con mi tutor del TFG tras varias reuniones para analizar y mostrarle el trabajo que ya tenía hecho, él enseguida me orientó sobre la forma en la que tenía que desarrollar la memoria en todos sus apartados. Esto supuso que tuviese que adaptar el contenido que ya había redactado a dichos apartados, al mismo tiempo tuve que componer los nuevos que no había tenido en cuenta.

Capítulo 3

Estado del arte

Comenzaré explicando el modelo en el que está basado esta aplicación, para así comprender mejor todas las tecnologías utilizadas.

3.1. Aplicación MVC

Este proyecto está basado en un modelo o plantilla de software conocido como Modelo Vista Controlador (MVC) [4], donde se fomenta la facilidad de mantenimiento, la reutilización de código mediante la separación en tres capas. Este modelo de desarrollo de software estructura de manera lógica un proyecto web facilitando su desarrollo tanto a la hora de crear, como a la de depurar errores. De esta manera, si hacemos un cambio en el código esta estructura permite que la aplicación entera no se vea afectada sino sólo una parte, pudiendo ser más intuitiva en caso de tener que localizar un error. Esta estructura también se anticipa a posibles cambios futuros, propiciando una gran adaptabilidad a los posibles cambios de tecnologías futuras.

En la Figura 3.1 observamos el esquema básico de la respuesta a una petición basándose en el modelo MVC. El navegador genera primero una petición (un HTTP request) y se lo envía al controlador (en el lado del servidor). El controlador genera una petición de datos según lo requerido y actualiza el modelo, p.ej., insertando dichos datos. El controlador envía el modelo actualizado a la vista y esta genera el interfaz de usuario que finalmente el controlador devuelve al navegador en una respuesta (un HTTP response).

En detalle, los componentes de MVC son:

- **Modelo:** es la representación de los datos de toda la aplicación y, por consiguiente, ma-

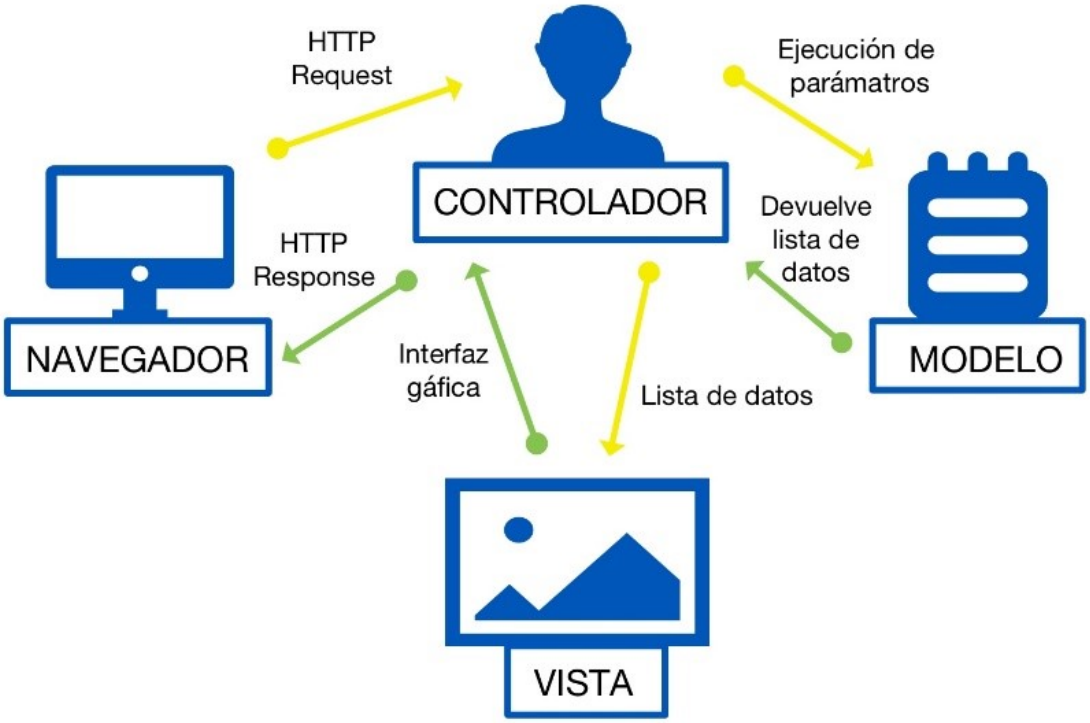


Figura 3.1: Diagrama de bloques MVC

neja las entradas y salidas de bases de datos. Es la información tanto a mostrar en las vistas como a recopilar y guardar. Además no sólo se obtendrá y mostrará, sino que se realizarán todas las transformaciones necesarias para cumplir con las funcionalidades de la aplicación.

- **Vista:** recibe los datos y los muestra al usuario final en una interfaz lógica. Ni el modelo ni el controlador se preocupan del diseño ni de la apariencia final de la aplicación, toda esa responsabilidad recae sobre las vistas.

- **Controlador:** es el responsable de responder a los eventos de entrada, ya sea respondiendo a un *click*, un cambio en un texto o un evento temporal, y realizar una serie de funciones establecidas previamente. Los controladores reciben el evento de entrada, interaccionan con el modelo si fuera necesario, se comunican con las vistas y responden al evento con una salida.

Esta estructura de desarrollo requiere de un lenguaje orientado a objetos, como es C#, para ser llevada a cabo de manera sencilla.

3.2. Tecnologías y materiales

En cuanto a las tecnologías haremos un repaso de las más importantes, su uso en la aplicación y por qué hemos elegido dichas tecnologías por encima de otras:

- La plataforma más importante, la base en la que se apoyan las demás tecnologías, es .NET Visual Studio Framework. Dicha plataforma de desarrollo de software fue lanzada por Microsoft para fusionar su catálogo de productos, que va desde sistemas operativos a herramientas de desarrollo. La alternativa estudiada fue la plataforma Java de Oracle Corporation, y sus diversos *framework* basado en PHP. La elección de .Net Framework se determinó por razones como la disminución de tiempos de desarrollo, la versatilidad o la centralización de todas las tecnologías necesarias para el desarrollo de aplicaciones web.

- Entorno de desarrollo: Visual Studio 2019 en el que se centra toda la aplicación, facilitando en gran parte tareas bastante tediosas con una cantidad de extensiones y plugins muy amplias. Esto facilita el desarrollo, con ayudas a la escritura o corrección al vuelo de posibles errores de compilación, y la depuración de errores, ya que permite conocer el valor de las variables añadiendo puntos de interrupción y evaluar el error cometido, ya sea del código propiamente dicho o de sintaxis, como de errores en tiempo de ejecución. Además, el lenguaje común conocido como CLR (Common Language Runtime) que es independiente de la plataforma, lo que acelera las posteriores ejecuciones del software. Las facilidades que ofrecen las plantillas creadas, en este caso una basada en el modelo vista controlador como se explicará más adelante, es uno de los motivos por los que el lenguaje elegido es C#.

3.2.1. Tecnologías básicas

- **El lenguaje:** C# es un lenguaje orientado a objetos multiparadigma desarrollado y estandarizado por Microsoft como parte de su plataforma .NET [2]. Cualquier tipo de dato

por simple que sea, se puede tratar como un objeto y llevará asociado unos métodos particulares. Así, por ejemplo, un *char* lleva aparejado el método GetNumericValue, que convertirá un objeto *char* que represente un número en un valor numérico. Además, es un lenguaje muy seguro y flexible, ya que permite que el código omita la verificación de seguridad del marco de datos y manipular la memoria directamente.

- **La parte Web:** utilicé HTML mezclado con algunas herramientas. En menor medida aparece jQuery y JavaScript para algunas de las funcionalidades que modifican el Document Object Model (DOM), que es el modelo de objetos del documento.

- **El estilo:** utilicé un estilo propio en cuanto a elección de colores, fuente, etc., pero basado en Bootstrap que es un marco de trabajo que nos facilita el trabajo de diseño y maquetación web, modificándolo a nuestro gusto.

- **Base de datos Microsoft SQL Server 2014 Express:** es la versión ligera de Microsoft SQL Server. Es una versión gratuita y de uso libre, en la que guardamos los datos de la aplicación web. Para manipular la base de datos utilizaremos la plataforma SQL Server Management Studio, que es la que usamos para crear las tablas, las claves y las restricciones de la base de datos.

- **Servidor Web:** Haremos uso de Internet Information Service (IIS) para alojar y desplegar el proyecto MVC. Asimismo, añadiremos el dominio y el Secure Socket Layer para el protocolo seguro de navegación. IIS nos permite tener el programa instalado en nuestra red interna en un servidor, o si lo instalamos en un servidor con acceso a Internet podríamos llevar a cabo sus funcionalidades en cualquier parte, ya que hoy en día económicamente hablando no supone un gran desembolso. Incluso en el caso de no querer ocupar espacio podríamos instalar la aplicación en un Virtual Personal Server, o servidor virtual privado, que podremos adquirir telemáticamente y a precios muy asequibles.

3.2.2. Herramientas más importantes dentro de Visual Studio

- Dentro de la aplicación usé `razor`, para que haga de nexo entre C# y HTML, basado en una sintaxis especial. Precedido de el símbolo @ podremos usar métodos como puede ser el de crear un menú dinámico dependiendo de las secciones introducidas en la base

de datos o usar los modelos o clases creados para, por ejemplo, el uso de formularios o el uso de diccionarios, útiles para modificar texto sin entrar en la aplicación, corregir errores o cambiar el idioma.

- Linq (Language Integrated Query) es un componente muy útil que añade funciones de consulta de datos, el cual he usado para ordenar, obtener y manipular listas. Aunque resulto un componente arduo de manejar una vez entendido su funcionamiento agilizo muchos procesos que de otra manera hubieran sido muy costos.

- La otra herramienta que sin duda es de las más importantes y una de las razones por la que Visual Studio ahorra tanto tiempo a la hora de desarrollar, es Entity Framework, que simplifica mucho todas las interacciones y sintaxis SQL. De esta manera, genera automáticamente un modelo, basado en clases de la base de datos, que podremos utilizar sin ningún tipo de problema como si fuese una clase más.

Capítulo 4

Diseño e implementación

4.1. Diseño y manual de usuario

La aplicación web, en cuanto al diseño consta de dos partes:

1. El acceso a la aplicación:

    - La primera ejecución de la aplicación, una vez instalada, mostrará un formulario de usuario que será el primer administrador de la aplicación (véase Figura 4.1).

Figura 4.1: Primera ejecución

    - Las posteriores ejecuciones empezarán con la página de login, tal y como se muestra en la Figura 4.2, en donde el usuario previamente creado, ya sea en la primera ejecución o añadido por el administrador, tendrá que introducir su usuario y contraseña.

2. Una vez se ha accedido a la aplicación:

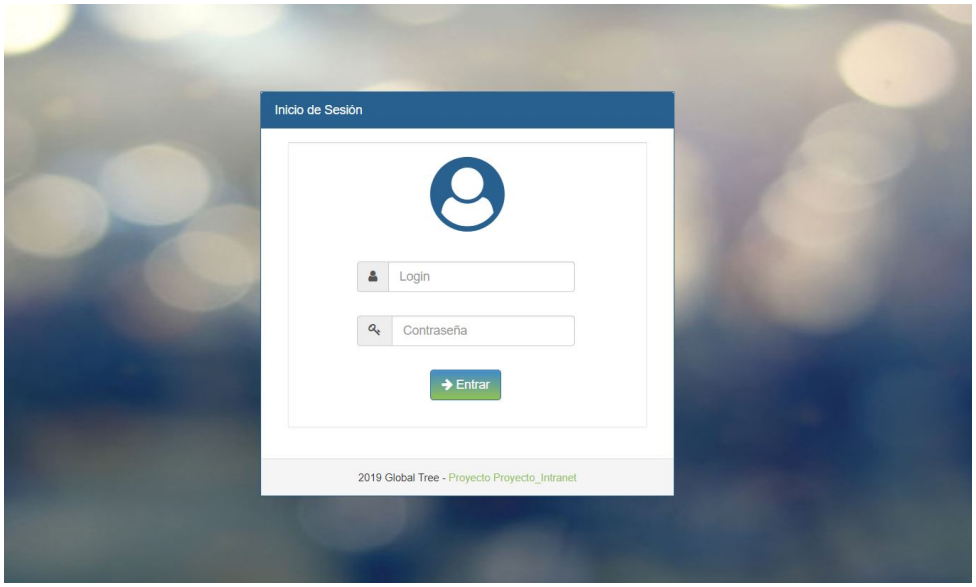


Figura 4.2: Plataforma de entrada o *login*

- Aparece el *home*, como se puede ver en la Figura 4.3, que es donde se muestra la aplicación por primera vez dando la bienvenida al usuario, siendo amigable y cercana y donde se puede ver el *layout*. Esta plantilla es el esquema que acompañará al usuario por toda la aplicación. Está compuesta por una barra superior con el logo y el nombre de la empresa a la izquierda. A la derecha, el botón que da acceso al cambio de perfil y un texto con el nombre del usuario registrado. En la parte izquierda aparecerá una columna con un menú informando sobre las distintas secciones o páginas disponibles. El contenido central es el que irá cambiando según accedamos a una sección u otra.

Figura 4.3: Página principal o *home*

- En el acceso al perfil, mostrado en la Figura 4.4, nos aparecerá un formulario con la información introducida en la base de datos de nuestro usuario y nos permitirá modificar algunos campos. Los campos modificables son: nombre, apellidos, nombre de usuario, dirección de correo electrónico, teléfono, dirección postal y, por último, dos campos en los que deberá introducir una nueva contraseña en el caso de querer cambiarla, teniendo en cuenta que ambos campos deberán coincidir.

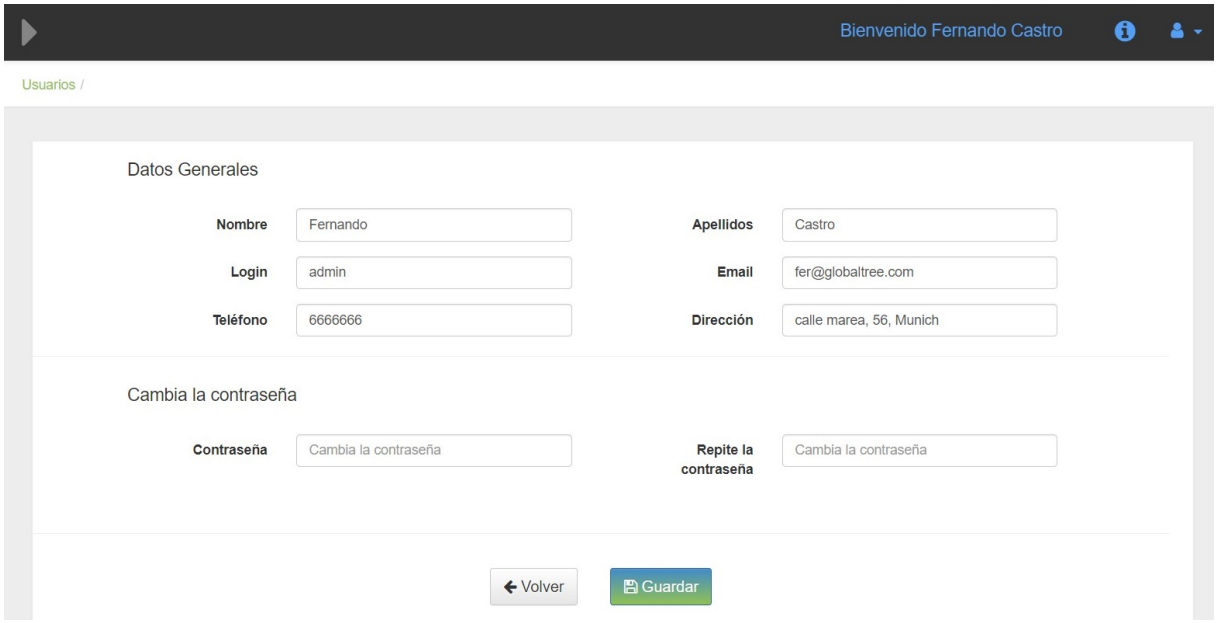


Figura 4.4: Formulario de edición del perfil

- Las secciones: usuarios y clientes, tal y como muestra la Figura 4.5, son secciones en las que se muestra el contenido de las bases de datos en una tabla. Se podrá buscar un cliente concreto en el campo de búsqueda y acceder a crear entradas a la base de datos mediante un formulario que para clientes necesitará rellenar los campos siguientes: nombre, CIF, representante, dirección de correo electrónico, teléfono y dirección postal. Para usuarios son: nombre, apellidos, nombre de usuario, dirección de correo electrónico, teléfono, dirección postal y, dependiendo del rol del usuario que visita la página, una serie de acciones, como son la de observar los datos, editar y borrar. Además, existe un sistema de paginación donde se podrá elegir cuántos registros de la base de datos se visualizan por página y la posibilidad de navegar por las diferentes páginas. La Figura 4.6 permite ver cómo es el formulario de introducción de datos.

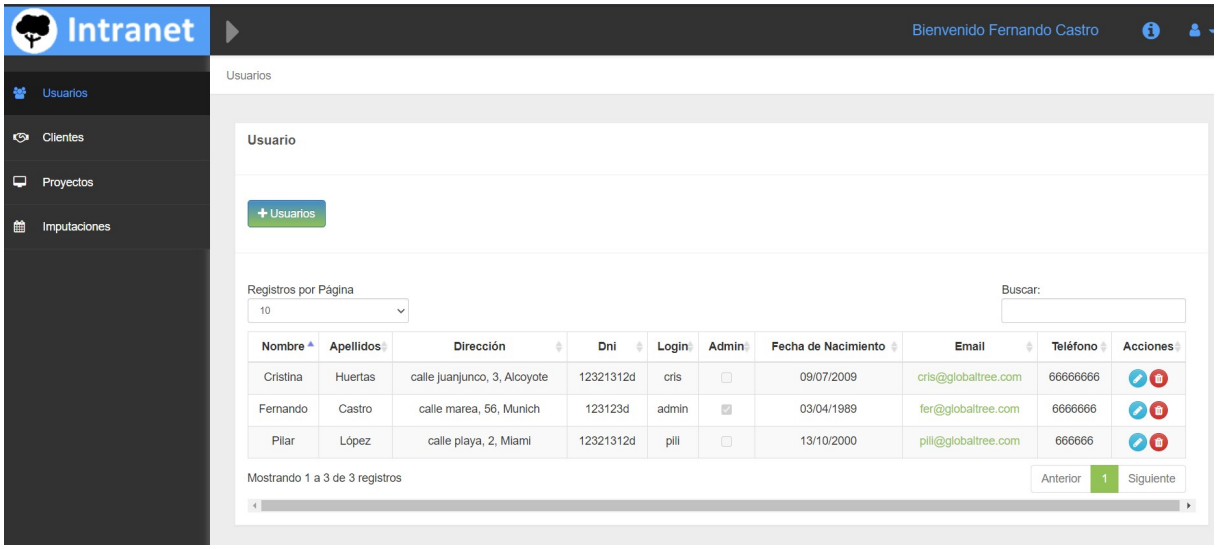


Figura 4.5: Tabla de la sección usuarios

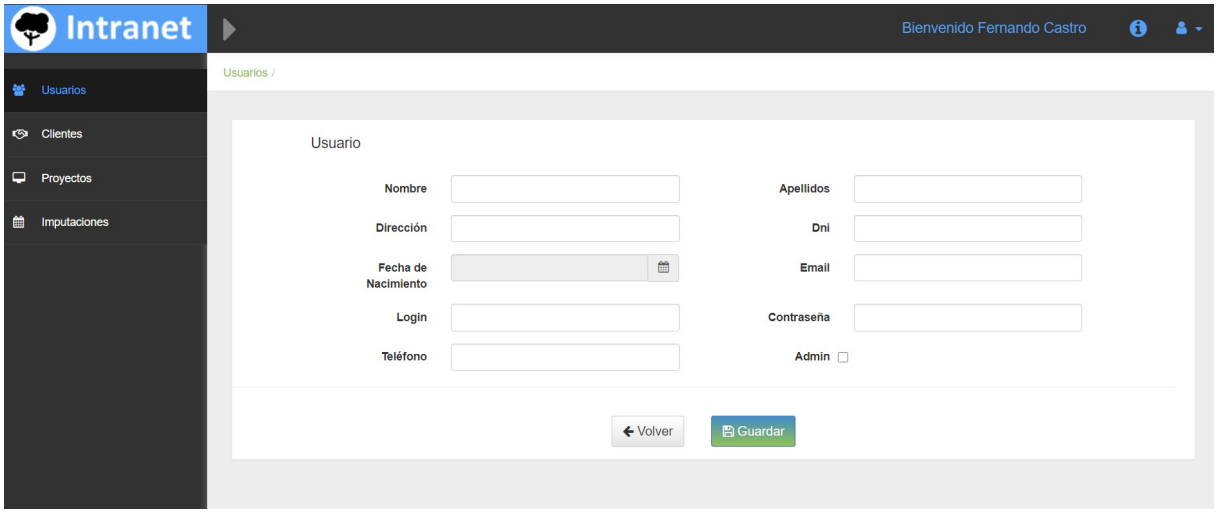


Figura 4.6: Formulario de la sección usuarios

- La última sección y una de las más importantes es la de imputaciones. Se trata de la razón de ser de toda la aplicación, ya que es donde aparecerá un calendario y la posibilidad de añadir imputaciones al mismo, que se guardarán en una base de datos. Dichas imputaciones se introducirán en una ventana emergente, como se puede ver en la Figura 4.7, en la cual mediante un formulario se puede introducir el día, tiempo de la tarea y un comentario explicativo de la misma. De esta manera, se podrá llevar un control de las tareas y el tiempo empleado en cada una de ellas.

- Por último, el usuario podrá mediante un botón llamado estadísticas tener acceso al

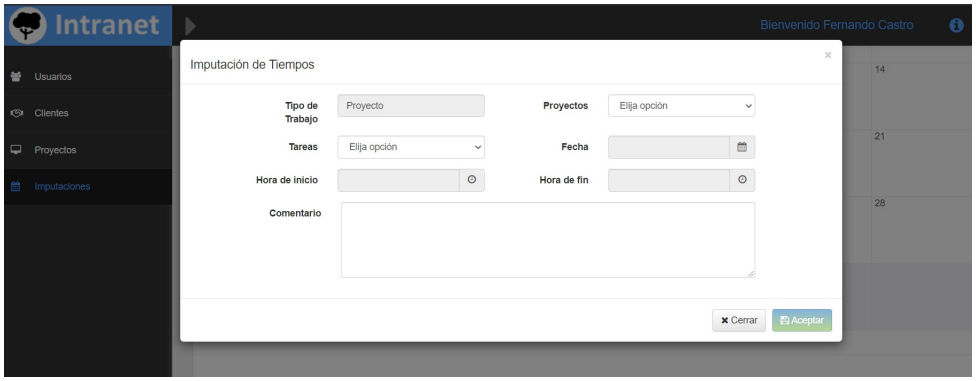


Figura 4.7: Calendario de la sección imputaciones

cómputo global de horas semanales imputadas, como se puede ver en la Figura 4.8.

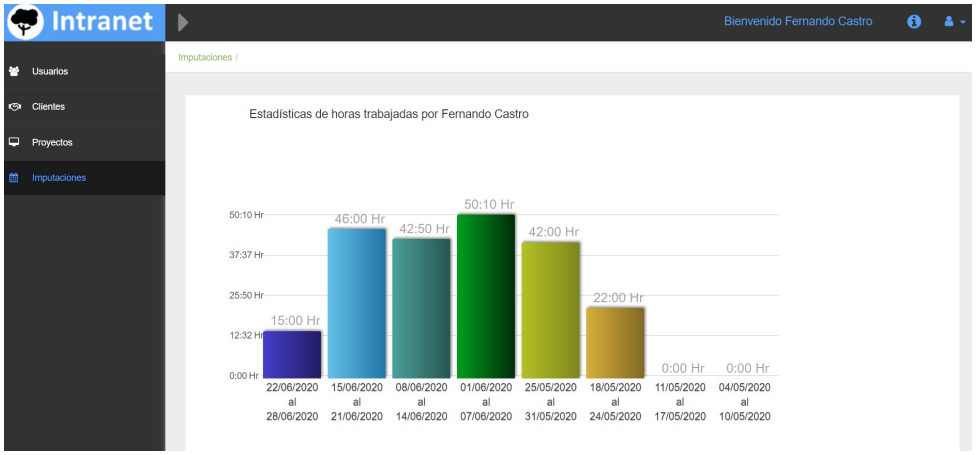


Figura 4.8: Estadísticas semanales de la sección imputaciones

4.2. Arquitectura general

Figura 4.9: Arquitectura cliente-servidor

En la Figura 4.9 mostramos el funcionamiento de la arquitectura cliente-servidor donde los proveedores de recursos, llamados *servidor* responden a las peticiones de los usuarios, los llamados *clientes*, mediante Internet.

4.3. Estructura del proyecto

Se ha utilizado la estructura propia de la plantilla de MVC generada por Visual Studio, que se ha ido modificando, organizando, añadiendo y cambiando a nuestro gusto. Ignoraremos algunas de las carpetas generadas para el correcto funcionamiento de la aplicación y que son menos relevantes.

El proyecto, como se puede ver en la Figura 4.10, se compone de:

- Carpeta de global *resources*, donde se incorporan una serie de diccionarios, *APP_Start* donde se encuentra archivos de configuración.
- *Content*, *Font*, *Images* y *Scripts* son carpetas/directorios donde se almacenan las distintas fuentes, el contenido jQuery, las imágenes del proyecto y todo lo relacionado con el contenido de JavaScript.
- En *controllers* se encuentra una de las tres partes más importantes de la aplicación: los controladores.

- La carpeta *models* contiene todas las clases relacionadas con los datos y es otra de las partes más importantes.

- *Services* es donde se gestiona las entradas y salidas de datos de usuarios necesarios para controlar las necesidades de los usuarios conectados.

- *Utils* es una carpeta creada para almacenar las clases que se usan a modo de ayuda en la aplicación y donde se estructuran las herramientas necesarias para facilitar el uso de la aplicación. Esta serie de clases han sido creadas por la empresa *Net&Compare*, empresa en la que realicé las prácticas en empresa, para que sirvan como comodín para todos los proyectos. En este TFG, solamente han sido modificadas para adecuarlas a las necesidades de mi proyecto.

- En *Views* se generan las vistas que supondrán la parte gráfica organizadas en carpetas, la mayoría relacionadas con su controlador.

- Por último, los archivos restantes, que se ven en la parte inferior de la Figura 4.10, son de configuración.

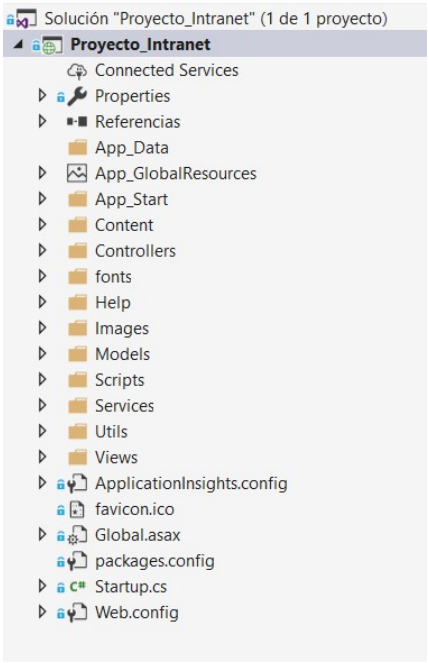


Figura 4.10: Estructura de la aplicación web MVC

4.4. Desarrollo de la aplicación

4.4.1. Configuración inicial

Se creó un nuevo proyecto en Visual Studio de tipo MVC, que genera automáticamente una plantilla con una estructura y una configuración, en la cual otorgaremos el nombre del proyecto. Una vez generado el proyecto desde la pestaña *extensiones*, en *administrar extensiones* buscaremos las herramientas y paquetes necesarios para nuestra aplicación. Las más importantes son `Razor` y `Entity Framework`.

Cuando se haya instalado todo lo necesario, y una vez creada la base de datos (tal y como se verá en la Sección 4.4.4), nos dispondremos a conectar la cadena de conexión que es uno de esos archivos que se generan automáticamente desde la plantilla. En concreto web.config donde modificaremos la siguiente línea con los datos para conectarnos a la base de datos local. Cuando ya está todo conectado, antes de ponernos a modificar nada, creamos las primeras pantallas mediante el *scaffolding*, procedimiento que generará desde un modelo de datos las vistas: index, crear, editar y borrar. Estas vistas las podremos modificar más tarde a nuestro gusto.

```
source=localhost\SQLEXPRESS;initial catalog=Proyecto_Intranet;
```

Esta cadena de conexión se usa para una base de datos en el mismo servidor donde alojamos la aplicación, con ello mejoraremos los tiempos de carga. Si usáramos un servidor externo tendríamos que indicar en el fichero de configuración aparte de los valores de *source* y *catalog*, *user* y *password*.

4.4.2. Inicio de la aplicación

En la capeta APP_Start se encuentran dos archivos de configuración que son de gran importancia:

- Por un lado tenemos *RouteConfig*, donde podemos observar los valores por defecto que va a tener el mapa de ruta, y cómo el controlador *home* nos dirigirá a la página principal *index*.

```
routes.MapRoute(
  name: "Default",
  url: "{controller}/{action}/{id}",
```

```
        defaults: new { controller = "Home", action = "Index",
                        id = UrlParameter.Optional });
```

- El otro archivo de configuración del que vamos a hablar es *BundleConfig.cs*. Si tenemos demasiadas interacciones de HTTP, podemos tener retrasos y múltiples hilos de conexión innecesarios, con todo lo que ello conlleva. El *bundling* es el proceso de agrupar varios recursos, CSS o JavaScript, en un único archivo descargable. De esta manera se descargan múltiples recursos con una sola solicitud.

En cuanto a la *minificación* lo que buscamos es atajar otro problema, que es reducir el ancho de banda, ya que elimina todos los caracteres innecesarios sin alterar la funcionalidad, cambiando identificadores, nombres de las funciones y eliminar todo aquello que se ha agregado con fines de legibilidad. Esta agrupación requiere otro paquete como es Microsoft ASP.NET Web Optimization.

```
bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
  "~/Scripts/jquery.validate*"));
```

4.4.3. Estilo y recursos globales de la aplicación

En cuanto al estilo, Visual Studio genera un CSS inicial. Este CSS se descartará y, en su lugar, para dar forma al proyecto en la parte visual y ahorrarnos mucho tiempo desarrollando un CSS propio desde cero, usaremos el *framework* de Bootstrap [7]. Bootstrap es una biblioteca CSS multiplataforma con licencia de software libre desarrollada por Twitter que contiene plantillas de diseño, maquetación, botones, cuadros de diálogo, tipografía, formularios, menús, etc. Existen alternativas a Bootstrap, como son Pure.css, W3.css, Materialize.

Se ha elegido este *framework* por ser de los más utilizados, con un gran soporte detrás, actualizaciones constantes y por contar con una documentación amplia [5, 1], haciendo sencilla la integración de cualquier parte en la aplicación. Todos estos archivos de configuración se cargan al iniciar la aplicación en el Global.asax.

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

```
        ModelBinders.Binders.Add(typeof(decimal), new RequestModelBinder<decimal>());
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
Adicionales Global resources/Script/fonts
```

App_GlobalResources:

Esta carpeta contiene una serie de diccionarios con varios fines. El primero es la rapidez con la que se puede cambiar cualquier valor sin tener que buscarlo por toda la aplicación, muy útil en futuros cambios o actualizaciones. Por otro lado, es de gran importancia a la hora de añadir nuevos idiomas en los que sólo necesitaríamos crear otros diccionarios y, usando el mismo campo de nombre, generar los nuevos valores y usar unos u otros según convenga.

Los diccionarios, se componen de tres campos: nombre, valor y comentario.

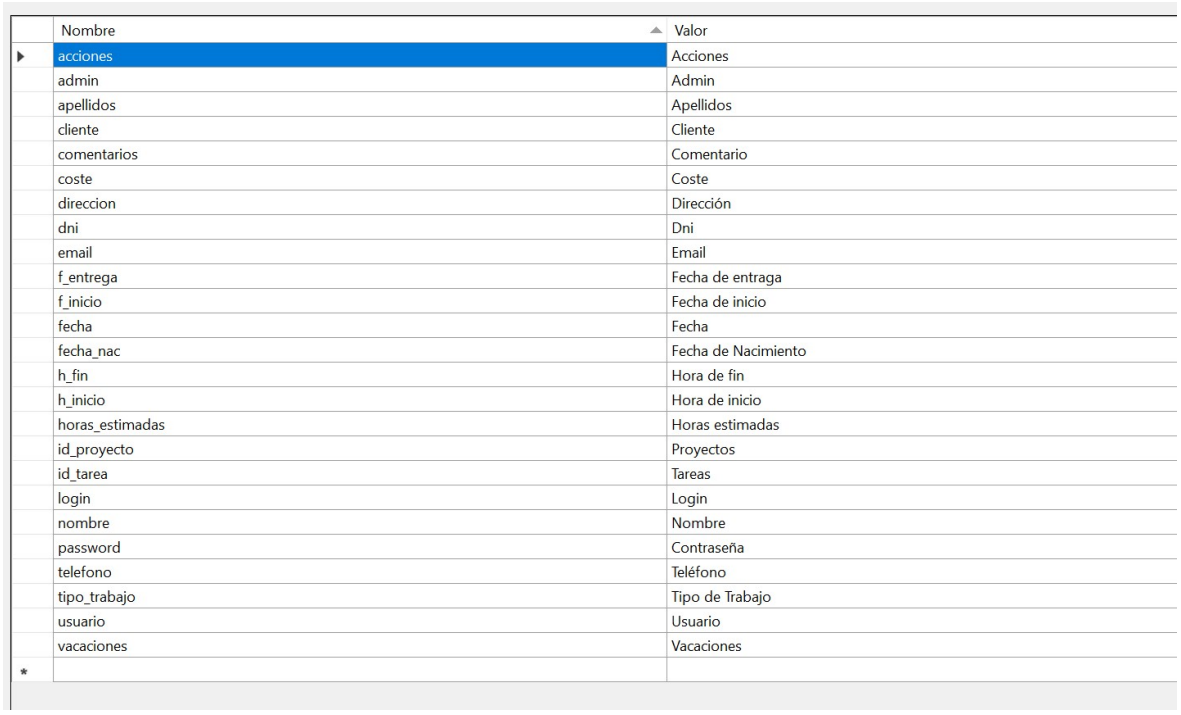
El **nombre** es la referencia para hallar el valor y es el que usaremos en la aplicación. El nombre llevará una serie de requisitos para evitar posibles confusiones o errores de compilación. Las restricciones son no llevar símbolos, mayúsculas, o espacios, y el único símbolo permitido será el guión bajo para sustituir a los espacios. **Valor y comentario** son campos internos. El **valor** es el campo que devuelve al pedir su nombre asociado y donde deberemos poner correctamente cómo se va a mostrar. El **comentario** es un campo que usamos para intercambiar información con los demás desarrolladores sobre ese registro en concreto.

Los diccionarios que hay son:

- Botones: donde introduciremos todos los botones de la aplicación, como guardar, cerrar, entre otros.
- DBFields: donde guardaremos todos los campos de las bases de datos que se usan en las vistas, como en la tabla I_USUARIOS, el campo admin, nombre, entre otros. Se puede ver un ejemplo de este diccionario en la Figura 4.11.
- Errores: serán errores que mostrará el programa al usuario final. Esta tabla se irá rellenando según vayan apareciendo más errores.

- Messages: diccionario de mensajes que aparecen en la aplicación, como el de bienvenido o cualquier información introducida en las vistas.

- Secciones: como su propio nombre indica son los apartados del propio menú.



Nombre	Valor
acciones	Acciones
admin	Admin
apellidos	Apellidos
cliente	Cliente
comentarios	Comentario
coste	Coste
direccion	Dirección
dni	Dni
email	Email
f_entrega	Fecha de entraga
f_inicio	Fecha de inicio
fecha	Fecha
fecha_nac	Fecha de Nacimiento
h_fin	Hora de fin
h_inicio	Hora de inicio
horas_estimadas	Horas estimadas
id_proyecto	Proyectos
id_tarea	Tareas
login	Login
nombre	Nombre
password	Contraseña
telefono	Teléfono
tipo_trabajo	Tipo de Trabajo
usuario	Usuario
vacaciones	Vacaciones

Figura 4.11: Diccionario DBFields con los campos de la base de datos.

4.4.4. Base de datos y metadata

Una de los aspectos más importantes de la aplicación, una vez hecha la configuración inicial, es el diseño y despliegue de la base de datos. Nos apoyaremos en la base de datos para almacenar todos los datos de la aplicación, que es a lo que llamamos el *modelo*. A la hora de diseñar una aplicación de este calibre se debe crear una estructura previa donde diseñaremos una base de datos inicial en función de las necesidades de cada proyecto. Esta base de datos puede ser modificada en cualquier momento, variando, añadiendo o suprimiendo las tablas en sí o su composición.

Para el diseño de las tablas se ha seguido una estructura según dos formatos: uno se ha nombrado con los prefijos *APP_* para indicar que el contenido de la tabla es necesaria para el

funcionamiento de la aplicación y dichas tablas no se podrán modificar desde la propia aplicación, como por ejemplo serían las secciones del menú o los datos de configuración; y el otro prefijo, mediante *I_*, indicando que la tabla está vacía y contendrá todos aquellos datos que vayas introduciendo mediante el uso, como por ejemplo la lista de clientes, usuarios, proyectos, entre otros.

APP_CCONFIG:

Key	clave principal, debe ser única	string
Value	valor	string

Cuadro 4.1: Tabla APP_CONFIG

APP_CONFIG (ver Tabla 4.1) es una tabla de dos campos tipo diccionario que contiene constantes y datos de configuración como el código de color por ejemplo: dias_vacaciones en key y en value 30.

APP_SECCIONES:

Id_seccion	clave principal	integer
Nombre	valor	string
Id_seccion_padre	valor	integer
Oreden_menu	valor	integer
Icono	valor	integer

Cuadro 4.2: Tabla APP_SECCIONES

APP_SECCIONES (ver Tabla 4.2) es una tabla que gestiona el menú y sus secciones. De la tabla se obtiene la información necesaria para ordenar la visualización del menú, el icono de cada sección, así como el nombre y por último si la sección en cuestión pertenece a otra que la llamaremos padre, para que el programa genere un menú desplegable automáticamente con el contenido.

I_ROLES_SECCIONES:

Admin	valor	boolean
Id_seccion	clave principal	integer
Solo_lectura	valor	boolean

Cuadro 4.3: Tabla I_ROLES_SECCIONES

I_ROLES_SECCIONES (ver Tabla 4.3) es una tabla que se usa para determinar el estado de la sección si es sólo lectura o se puede desplegar una nueva vista, además gestiona las secciones accesibles dependiendo de si tu rol es administrador o no.

I_CLIENTES:

Id_cliente	valor	integer
Nombre	valor	string
Mail	valor	string
Telefono	valor	string
Direccion	valor	string

Cuadro 4.4: Tabla I_CLIENTES

I_CLIENTES (ver Tabla 4.4) es la tabla que guarda los datos de los clientes de empresas, para que todos los trabajadores tengan acceso a ellos en caso de necesitarlos y para vincularlos a los proyectos.

I_PROYECTOS:

Id_proyecto	clave principal	integer
Nombre	valor	string
Horas_estimadas	valor	integer
F_inicio, f_entrega	valor	datetime
coste	valor	float

Cuadro 4.5: Tabla I_PROYECTOS

I_PROYECTOS (ver Tabla 4.5) es la tabla donde se guarda el proyecto de cada equipo donde guardamos las horas estimadas del proyecto, las fechas de inicio y de entrega, el nombre del proyecto, el coste que se ha pactado en la reunión de marketing. Esta tabla está vinculada a la tabla de clientes, donde se vincula el cliente a los diferentes proyectos.

I_TAREA:

Id_tarea	clave principal	integer
Nombre, color	valor	string

Cuadro 4.6: Tabla I_TAREA

La tabla I_TAREA (ver Tabla 4.6) guarda la tarea que utilizaremos para clasificar las imputaciones, como por ejemplo tarea de diseño, documentación, etc. También guarda el código HTML para el color con el que se va a guardar esa imputación. Esta tabla podrá ser modificada en cualquier momento y automáticamente se introduce en la lista de tareas.

I_IMPUTACIONES:

Id_imputacion	clave principal	integer
Id_proyecto, id_usuario, id_tarea	valor	integer
Horas_estimadas	valor	integer
Fecha	valor	datetime
Comentarios	valor	string

Cuadro 4.7: Tabla I_IMPUTACIONES

La tabla I_IMPUTACIONES (ver Tabla 4.7) es la más importante de la aplicación, ya que da sentido a la aplicación para que no sólo muestre tablas que enseñan y clasifican la información. En esta tabla se guarda la información necesaria para conocer la estimación de tiempos necesarios para cada proyecto dependiendo del usuario y el tipo de tarea que ha realizado.

I_USUARIOS:

Id_usuario	clave principal	integer
Nombre, apellido	valor	string
Dirección, dni	valor	string
Login, password	valor	string
Email, teléfono	valor	string
Admin	valor	boolean
Fecha_nac	valor	datetime

Cuadro 4.8: Tabla I_USUARIOS

La tabla I_USUARIOS (ver Tabla 4.8) contiene una lista de trabajadores de la empresa. A partir de la misma obtenemos la información personal de cada trabajador y sus datos de acceso a la aplicación, con *login* y *password*. Así también podremos conocer el nivel de acceso a la aplicación sabiendo si es el administrador del sistema o un trabajador.

En la Figura 4.12 se pueden observar todas las tablas de la aplicación, así como las relaciones entre las mismas.

Metadata:

Se utiliza una clase parcial de todas las vistas de formularios para añadir los datos a las tablas como son: cliente, imputación, proyectos y usuarios. Dicha clase parcial se usa para introducir los requisitos de cada formulario de entrada, como los campos imprescindibles o la cantidad de letras o números, usándolos para validar el modelo antes de guardarlo en la base de datos. Su otra función es la de utilizar la biblioteca System.ComponentModel.DataAnnotations; como por ejemplo en la nomenclatura:

```
[FieldName("h_inicio ")]
```

donde la palabra *h_inicio* de la tabla I_IMPUTACIONES se sustituye por el texto correspondiente del diccionario, que en este caso es *Hora de inicio*. Para este ejemplo los requisitos de validación serían tener el formato necesario para dar coherencia a una hora, como es que no contengan letras y que se separe las horas de los minutos mediante los dos puntos:

```
[DataType(DataType.Time), DisplayFormat(DataFormatString = "{0:HH:mm}"...
```

El segmento de código anterior corresponde a la validación de las *Horas de inicio*.

La base de datos es la que sustenta el modelo y nos ayuda a guardar todo la información de la aplicación. Además sería conveniente realizar periódicamente un *backup* o copia de seguridad. Con esto podemos evitar o subsanar errores como borrar registros, ataques de seguridad o problemas en el servidor que hagan perder toda la información.

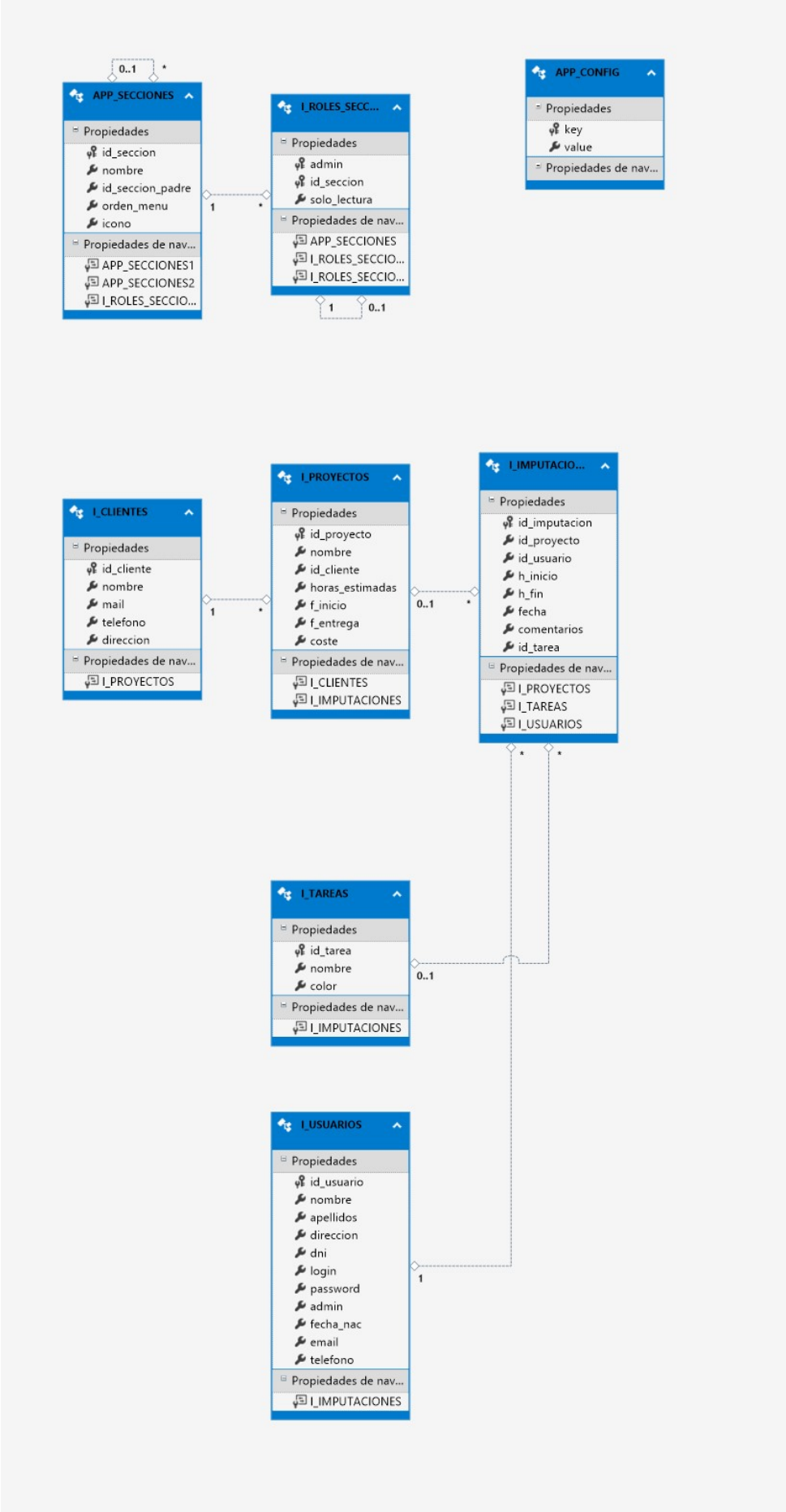


Figura 4.12: Base de datos con todas sus tablas y relaciones

4.4.5. Utils y Services

Utils y Services son herramientas que se usan de manera repetida en distintas partes de la aplicación con utilidades diversas. La diferencia con la carpeta *service* es que no es un servicio propio de la aplicación sino unos métodos agrupados en clases que dan utilidades diversas.

La funcionalidad que proporciona es la siguiente:

- **AppConfig:** gestiona las configuraciones guardadas en la tabla APP_CONFIG.
- **Constants:** guarda las constantes de la aplicación, para agilizar procesos en la aplicación y haciendo muy sencillo cualquier modificación.
- **CookieStore:** crea, obtiene y elimina la información necesaria para usar las cookies, necesarias para guardar la información de usuario.
- **DAUtil:** son un grupo de métodos que administran las entradas de datos. Comprueban que todos los requerimientos se cumplan para guardar los datos en la base de datos sin error, ya que son su validación, si no se cumple devuelve cuál está mal.
- **Parse:** se usa para realizar transformaciones complejas de datos, para agilizar estos procesos en la aplicación y generar nuevos tipos de datos.
- **RazorUtil:** puente entre el código HTML y C#. Se utilizan métodos para crear formularios, ya sean de texto como fechas, horas, entre otros. También se usan para métodos de validación de los formularios y creación de desplegables y botones.
- **Security:** se usa para controlar la seguridad, ya sea utilizando algoritmos de criptografía en una contraseña para guardarla en la base de datos, o para cifrar y descifrar cualquier información necesaria dentro de la aplicación a menor nivel acorde al uso, como el de guardar información del usuario en las cookies.
- **Session:** obtiene la información de la sesión creada en el momento de iniciar la aplicación para obtener el nombre del usuario y añadirlo dinámicamente en las vistas, o para identificarlo con su id y por último obtener su rol.

- **Tools:** herramientas diversas de uso interno como puede ser obtener el controlador antiguo, obtener las secciones, despachar mensajes de error o éxito, obtener la URL, escalar imágenes, obtener el mes, entre otros.

Service:

Este espacio se usa para introducir clases que harán de servicio de cualquier función necesaria que se use en un caso especial y tenga más de un método. Se compone de la interfaz, que es donde declaramos los métodos, y el otro es la implementación de la clase. En este caso sólo hay una clase:

AccountService:

Clase cuya misión es comprobar si es la primera ejecución, ya que en eso caso mostrará un formulario para registrar al primer usuario administrador y en caso contrario mostrará la pantalla *login*. La forma en la que realiza esta acción es con la comprobación del número de usuarios: si no hay ninguno registrado estaremos ante la primera ejecución. Además comprueba si el usuario es administrador para que tenga acceso o no a las funciones y secciones que le correspondan.

4.4.6. Controladores

Es otra de las partes importantes de la aplicación, ya que gestionan los eventos requeridos por el usuario y devuelven las vistas incluyendo los modelos. Los distintos controladores se diferencian por las secciones y los usos. En un primer nivel se dividen según hayas iniciado sesión o no, el controlador que controla el inicio de sesión y el que va por libre es AccountController, en cambio el resto parte de la BaseController donde este controla que controlador debe responder a que evento.

- **BaseController:** clase controlador del que heredan todos los controladores. Se genera automáticamente al generar la plantilla MVC. A partir de esta clase se generan varios métodos, como son la conversión de las vistas de Razor a string, u obtener el estado del usuario, es decir si se encuentra conectado o no y en cual sección, título y rol posee. Por último, hay dos métodos que introducen tanto mensajes de éxito como de fracaso.

- **AccountController:** controla la primera ejecución del programa y vistas que lo componen, como el login y el formulario de inicio para guardar el usuario administrador, además

de comprobar los requisitos de contraseña usados para lanzar un mensaje de advertencia donde indica que dicha contraseña no coincide con la segunda para evitar errores al introducirla, a continuación, redireccionará la página al *home* donde aparecerá el mensaje de bienvenida y el menú lateral con las secciones. Como última utilidad responde al cierre de sesión para redireccionar a la página de login donde se podrá volver a iniciar sesión.

- **ClienteController, ProyectosController:** son dos controladores que realizan las mismas funciones que son la de mostrar, crear, borrar y editar los valores de la base de datos, gestionando el control de las vistas que muestran los formularios de estas funciones y realizando dichas funciones, siempre y cuando cumplan los requisitos. La única diferencia entre ambos es, los datos en los que realizan cambios, ya que uno es en la lista de clientes y el otro en la lista de proyectos.

- **UsuarioController:** se basa en las mismas funciones de los anteriores con la diferencia de que sólo el usuario administrador podrá acceder a los datos de todos los usuarios crear, editar y borrarlos, y por otro lado también gestión del perfil, donde se podrá modificar algunos campos de los datos del usuario conectado, todos aquellos de carácter personal, es decir, nombre, apellidos, teléfono, contraseña, etc.

- **ImputacionesController:** es el controlador que gestiona las vistas para imputar los registros del día, así como las estadísticas de dichos registros o eliminando los mismos.

Funcionamiento de un método controlador:

El funcionamiento básico de un método controlador se explica mediante un ejemplo. En particular, el método en concreto que se va a presentar es el que se ocupa de la vista principal de las imputaciones, obtiene los datos a introducir en la vista y la genera cuando se produce un método HTTP GET.

En este método se puede observar una llamada asíncrona que comprobará si la id (identificación) contiene los datos de la sesión de usuario. Si no es así, los obtendrá, comprobará el rol del usuario, para ver si es administrador o no y poder acceder a todas las acciones.

Por último, introducirá dichos datos del usuario, además de las listas de estos si fuera administrador y la lista de imputaciones, en un *viewbag*, que es la forma de compartir información entre el controlador y la vista. También podríamos haber usado el viewdata, pero mientras que

el lo que hemos mostrado se basa en diccionarios, viewbag se usa de forma dinámica. Por último el método devolverá la vista requerida.

Código del método Index, que muestra la página principal de la clase ImputacionesController:

```
// GET: Imputaciones
  public async Task<ActionResult> Index(int? id)
  {
  if (!id.HasValue)
      id = Sesion.GetUsuario();
  else
      ViewBag.ID = id;
  if (Sesion.GetRol())
    ViewBag.lstUsuarios = await db.I_USUARIOS.
        Where(x => x.id_usuario!= id).ToListAsync();

  I_USUARIOS usuario = await db.I_USUARIOS.FindAsync(id);
  ViewBag.Nombre = usuario.nombre + " " + usuario.apellidos;

  System... serializer = new System.Web.Script.Serialization.JavaScriptSerializer();
  ViewBag.lstImputaciones=ser...(await db.I_IMPUTACIONES.
        Where(x => x.id_usuario == id).ToListAsync()));
  return View();
    }
```

A continuación, presentamos otro método de la clase ImputacionesController, en este caso con el método POST. Este procedimiento se llama igual que el anterior, pero se gestiona ante un request POST de la vista index y lo usamos para introducir imputaciones en la base de datos:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Index(I_IMPUTACIONES i_IMPUTACIONES)
```

4.4.7. Seguridad y cookies

Para guardar las contraseñas usamos un sistema de codificación MD5 de 128 bits que es simple de usar (además de gratuito). Lo usaremos con cada contraseña en el momento de *login* y lo compararemos con la contraseña guardada en la base de datos con esta misma codificación para comprobar si el usuario es el correcto.

A continuación, se mostrará cómo se crea dicha codificación, convirtiendo a []bytes la palabra a codificar, generando el hash y convirtiendo de []bytes a hexadecimal con un bucle *for* y se devuelve convirtiéndolo en string:

```
public static string CreateMD5(string input)
{
    // Use input string to calculate MD5 hash
    using (System.Security.Cryptography.MD5 md5 =
     System.Security.Cryptography.MD5.Create())
    {
        byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(input);
        byte[] hashBytes = md5.ComputeHash(inputBytes);

        // Convert the byte array to hexadecimal string
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < hashBytes.Length; i++)
        {
            sb.Append(hashBytes[i].ToString("X2"));
        }
        return sb.ToString();
    }
}
```

Por otro lado, para cifrar y descifrar usamos TripleDESCryptoServiceProvider mediante una palabra clave o keyword que completará el tamaño necesario de 128 a 192 en incremento de 64 bits. Este método se usará para cifrar y descifrar la sesión de usuario con los datos necesarios en la aplicación que se incluirá en las cookies.

Cookies: Las cookies son pequeños trozos de información que se guardan en el navegador (el cliente). En nuestro caso guardaremos en la tabla de usuarios el id único del usuario conectado, su nombre, su apellido y si es administrador o no.

```
string sesionUser=i_usuario.id_usuario+"|"+i_usuario.nombre+
```

```
" "+i_usuario.apellidos+ "|"+i_usuario.admin;
```

Guardaremos la sesión y llamaríamos al método *SetCookie*:

```
Session["Usuario"] = sesionUser;
CookieStore.SetCookie("sesion", sesionUser, TimeSpan.FromDays(1));
```

A continuación, cifraremos la variable value donde esta nuestra información de la sesión y primero comprobamos que la cookie no exista. Si es así, se actualiza el tiempo de expiración y se añadirá el contenido encriptado de los datos. Si no, obtenemos la cookie antigua, actualizamos la expiración e introducimos el contenido de la variable encriptada de nuevo por si ha cambiado y se sustituye.

```
public static void SetCookie(string key, string value, TimeSpan expires){
HttpCookie encodedCookie = new HttpCookie(key, Security.Encrypt(value, "***"))
if (HttpContext.Current.Request.Cookies[key] != null)
            {
                var cookieOld = HttpContext.Current.Request.Cookies[key];
                cookieOld.Expires = DateTime.Now.Add(expires);
                cookieOld.Value = encodedCookie.Value;
                HttpContext.Current.Response.Cookies.Add(cookieOld);
            }
            else
            {
                encodedCookie.Expires = DateTime.Now.Add(expires);
                HttpContext.Current.Response.Cookies.Add(encodedCookie);
            }
```

4.4.8. Vistas

Las vistas contienen el contenido visual partiendo de código HTML y usando las herramientas necesarias para poder incluir los modelos creados y las variables del propio controlador, explicaremos el contenido de cada carpeta que forma el esqueleto de las vistas, dichas carpetas tienen un sentido y un orden ya que cada una coincide con cada uno de los controladores explicados previamente que son los que manejaran las vistas de cada carpeta y pasaremos al final a exponer el funcionamiento básico de una vista representativa de este proyecto.

- **Sección Account:** lo maneja el AccountController y se basa en tres vistas, login y usuario.

  - Login: es la vista inicial que se compone de un cuadro con un formulario de texto en el que introduciremos nuestro usuario y nuestra contraseña y validaremos los datos en la base de datos en el account controller para darnos acceso a la aplicación.

  - Usuario: es el primer formulario de entrada en la aplicación que sólo aparece la primera vez que se ejecuta y que otorga el primer usuario administrador, a partir de este momento solo podrá entrar dicho usuario creado con todos los privilegios y a partir de ahí podrá crear los demás usuarios y otorgarles los permisos deseados según su cargo o función de cara a la aplicación.

- **Sección clientes y Proyectos:** se controlará mediante ClientesController y Proyectos-Controller respectivamente, pero en ambas vistas el contenido y estructura de estas es la misma, tan solo cambia el modelo donde una depende del modelo de clientes y la otra del modelo de proyectos. Ambas se basan en Clientes/ProyectosForm, Create, Edit e Index. En el apartado *create* y *edit* simplemente se usan para separar en el controlador las distintas funciones que representan, pero ambos son direccionados a la vista *Form* de la siguiente manera: Ejemplo para clientes:

```
@model Proyecto_Intranet.Models.I_CLIENTES

@Html.Partial("_ClientesForm", Model)
```

  En la primera parte usamos `razor` para incluir el modelo que deseamos incluir y en la segunda línea volvemos a realizar la funcionalidad de `razor partial` para direccionar la vista, incluyendo en ella el modelo antes mencionado.

  - Index: es la pantalla principal, en la que una tabla muestra los registros de la base de datos dependiendo del modelo al que pertenece de forma sencilla e intuitiva. En esta pagina no se podrán alterar los campos de cada registro, pero sí se podrá ordenar de mayor a menor si es un número o fecha y alfabéticamente si son palabras. Además, en la parte final de la tabla se mostrarán unos iconos de colores para acceder de forma fácil a editar un registro y vincularlo con su vista o para eliminar de forma permanente, en este caso un mensaje de confirmación le informará de lo que se va a realizar para evitar posibles errores. El botón borrar no te direccionará a ninguna

vista concreta, sino que eliminará el registro de la base de datos, volverás a la página index, con los datos actualizados y con un mensaje de confirmación si se ha borrado con éxito el registro o si se ha producido un error de algún tipo también se mostrará. Por otro lado, en la parte inferior de la vista se sitúa un botón para crear un nuevo registro a la base de datos que te direccionará a la vista pertinente.

- Form: es la vista más compleja, ya que depende de si queremos crear o editar un registro, si el modelo contiene valor, el modelo aparecerá en los campos del formulario ya rellenos para modificarlos lo que el usuario desee y después validando dichos campos se podrá modificar los registros de la base de datos. Una vez hecho se redireccionará siempre a la página index e informará si dicha modificación se ha llevado con éxito. Si el modelo es *null* es porque lo que se quiere es crear un nuevo registro con lo que los campos aparecerán vacíos, para que los rellenemos. En ambos casos la introducción de algún campo que sea erróneo devolverá un mensaje en dicho campo explicando el motivo por el cual no ha sido validado y sus requerimientos.

- **Sección Usuarios:** lo maneja el Usuarioscontroller y se basa en tres vistas, UsuariosForm, Create, Edit Index y Perfil. Esta sección, como veremos más adelante, sólo se encuentra visible si tienes el nivel de administrador, se compone por el Create, Edit, UsuariosForm y el index. Que son las mismas vistas que en la sección anterior, pero para el modelo de usuario. Pero en este caso además contiene la vista perfil y esta sí será accesible independientemente de los permisos de usuario activo, pudiendo acceder desde cualquier vista ya que se encuentra en el *layout* que es la vista sobre la que se asientan las demás y explicaremos más adelante. En esta vista aparecerá un formulario con la mayoría de información del usuario activo y disponible para comprobar que todo sea correcto y pueda cambiarlo en cualquier momento. Además, se podrá cambiar la contraseña propia. Se trata de la ventana a la cual se deberá entrar nada más obtener la clave propia de cada usuario para comprobar los datos personales y modificar las contraseñas predeterminadas por el usuario administrador que otorgo los datos.

- **Sección Home:** en esta sección sólo tenemos la vista index y se compone del logo de la empresa, de un mensaje de bienvenida y de un menú lateral en la parte izquierda que

puede ser desplegable si se quisiera, que contiene todas las secciones disponibles para acceder según tu nivel de seguridad. Para limitar el acceso a la sección usuario, desde el servicio Account se obtiene la lista de secciones que generará un procedimiento en RazorUtils para añadir las secciones según la base de datos, ya sea desplegable o no y para administradores o usuarios normales.

- **Sección Imputaciones:** en imputaciones es la sección que funciona de manera totalmente diferente, se compone de Index como página principal, Estadísticas e imputar.

  - Index: en este caso sigue siendo la vista principal, pero se compone por un calendario donde se podrá visualizar los días del mes presente con todas las imputaciones y con un código de colores para saber sobre que se han tratado, además en la parte inferior se puede visualizar el botón imputar que nos redireccionará a la vista de imputaciones y estadísticas que hará lo propio. El tercer botón es otras imputaciones para acceder a las imputaciones del resto de usuarios y poder modificarlas, pero sólo si tiene el rol administrador podrá visualizar y acceder a esta utilidad.

    ```
    @if (Sesion.GetRol())
    {
      <a class="btn btn-primary" to href="#"
      data-target="#OtrosModal" data-toggle="modal">
      <i class="fa fa-eye"></i>@Resources.Messages.other_imp </a>
    }
    ```

    Desde la Sesión en el procedimiento GetRol devuelve *true* siempre y cuando el usuario sea administrador y de esta forma es como limitamos el acceso.

  - Imputaciones: en esta vista aparecerá un formulario con la información necesaria para introducir una imputación en el calendario, por ejemplo, hora de inicio y fin, fecha, de que proyecto se trata etc.

  - Estadísticas: en esta vista se desplegará una gráfica indicando cuantas horas por semanas trabajadas para llevar la contabilidad global de cada semana y obtener el control de los usuarios.

- **Sección Shared:** esta sección se compone de vistas que no se controlan por ningún controlador, sino que son vistas auxiliares del resto de vistas y vistas que hacen de plantilla de fondo para todas las demás vistas, que es lo que llamamos layout y se compone de

Firstlayout, Layout que es el que usamos por defecto, Mensajes, NavBar, Simplelayout, Error, Lockout.

- FirstLayout: es la primera plantilla que se usa para el formulario de entrada que sólo aparece en la primera ejecución y es la más simple. En este caso sólo se usa en esta vista y se vincula con la vista usuario de esta manera:

```
@{
  ViewBag.Title = "Usuario";
  Layout = "~/Views/Shared/_FirstLayout.cshtml";
}
```

  De esta forma iniciamos el layout de todas las vistas.

- Layout: este es el layout por defecto que usa en toda la aplicación y el más completo, ya que contiene el menú superior con el saludo al usuario y el botón de cerrar de la sesión y de acceso a la vista del perfil, el último botón definido por una *i* se encuentra en desarrollo e introducirá una serie de imágenes con la información de ayuda necesaria en cada vista. También contiene el *footer* y la barra lateral con las secciones, en cuadrado que deja libre en el centro es el contenido que va a ir cambiando por las secciones.

- Mensajes: es una plantilla para informar de mensajes de éxito o de error en todas las funciones.

- NavBar: vista que forma parte del layout y que se conforma del menú superior con todos los botones, el saludo al usuario activo.

- SimpleLayout: es la plantilla que compone el login con una imagen central para amenizar la entrada y dar una bienvenida cálida y no contiene ni la sección lateral ni el menú superior, ya que todavía no se ha registrado el usuario.

- Error y Lockout: vista que se despliega cuando se produce un error al manejar alguna vista por el motivo que sea y si se bloquea al usuario se desplegará la vista Lockout.

- **Funcionamiento de una vista como el formulario de proyectos:**

```
@model Proyecto_Intranet.Models.I_PROYECTOS
```

```
@using Proyecto_Intranet.Utils
@using Proyecto_Intranet.Models
```

- Introducimos el modelo de proyectos y la sección *utils*.

```
@using (Html.DefaultForm()){
    <div class="form-horizontal">
        @Html.AntiForgeryToken()
        @if (Model != null && Model.id_proyecto > 0)
        {
            @Html.HiddenFor(x => x.id_proyecto) }
```

- Comprobamos si el modelo de datos está vacío. Si es así, mostramos el formulario vacío, ya que esto nos indica que nos encontraremos en la pestaña *crear*. Si por el contrario contiene datos supondrá que estamos en *editar* y el formulario aparecerá con la información del modelo para que modifiquemos lo que necesitemos.

```
<input type="hidden" value="@ViewBag.oldcontroller" name="oldcontroller"/>
<h4 class="col-md-offset-1">@Tools.GetControllerTitle() </h4>
```

- Primero obtenemos el controlador antiguo, por si queremos volver a la página anterior y el título de la vista.

```
<div class="form-group">
  <div class="col-md-offset-1 col-md-5">
      @Html.LabelFor(model => model.nombre, null,
      htmlAttributes: new { @class = "control-label col-md-4" })
```

- Con `razor` hemos creado la etiqueta para que muestre el nombre la sección de la base de datos e introducimos el atributo clase al título del *div*.

```
    <div class="col-md-8">
        @Html.TextBoxValFor(x => x.nombre)
```

- `razor` nos permite acceder para crear el formulario de texto de entrada.

```
<div class="btn-group-ge">
    @Html.ButtonVolver()
    @Html.ButtonSubmit()
```

- Además, creamos los botones de volver y de *submit* para validar los cambios.

Capítulo 5

Conclusiones

Global Tree es una aplicación para pequeñas empresas creada en C# con grandes posibilidades de futuro y con mejoras que añadirían valor y funcionalidades a la aplicación, otorgando la posibilidad de controlar telemáticamente desde cualquier lugar y en cualquier momento toda la aplicación.

Todos los datos se almacenan en un servidor privado sin la necesidad de depender de terceros, lo que concede flexibilidad, ya que en cada momento se podrá actualizar y mejorar el servicio u ofrecer servicios y funcionalidades nuevas.

Su finalidad es mejorar la eficiencia y el control de la empresa sumada a la capacidad de proporcionar el poder de usar sus cualidades a cualquier empleado gracias a su sistema de roles.

En cuanto a la consecución de objetivos podemos concluir que la aplicación *Global Tree* cumple su objetivo principal de gestionar una empresa basada en teletrabajo. Se le podría añadir múltiples nuevas funcionalidades y facultades, tantas como tipos de empresas hay ya que dependiendo del tipo de necesidad se podrá cubrir con algunas modificaciones.

5.1. Aplicación de lo aprendido

Las asignaturas más importantes para la elaboración de este proyecto han sido bastante diversas, como **Informática I** donde aprendí todos aquellas fundamentos de la programación que me ayudaron a comprender mejor la materia. En **Informática II** aprendí a crear y entender la arquitectura de cliente servidor, en **Construcción de Servicios** y **Aplicaciones Audiovisuales en Internet** es donde di mis primeros pasos en la parte web.

En cuanto al framework, no fue hasta **Gráficos y visualización en 3D** cuando el uso de Eclipse me ayudó a comprender mejor Visual Studio.

Finalmente la asignatura que más tiene en común con mi proyecto y una de las culpables de que me decidiera a crear esta aplicación fue **Laboratorio de tecnologías Audiovisuales en la Web**, en la cual aprendí a mezclar todas las tecnologías web como HTML, CSS, JavaScript, AJAX y en la parte del servidor las prácticas de Django o Node.js me hicieron comprender mejor su funcionalidad.

5.2. Lecciones aprendidas

Voy a enumerar todas aquellas aptitudes y habilidades adquiridas, durante el desarrollo de este proyecto. Son:

1. La capacidad de crear proyectos en solitario.
2. Aprender a usar tecnologías nuevas.
3. Profundizar en la herramienta Visual Studio.
4. Conocer mejor el lenguaje C# y sus procedimientos.
5. La utilización de Entity Framework
6. Creación de bases de datos en SQL mediante la herramienta SQL Management Studio.
7. Cifrado MD5.
8. Aprender a usar Bootstrap.
9. Creación y uso de cookies.
10. Uso de diccionarios dentro de la aplicación.
11. Uso y funcionamiento del modelo MVC.
12. Aprendizaje de sentencias Linq.

5.2.1. Proceso de aprendizaje de nuevas tecnologías

Esta sería la lección más importante que he aprendido y por ello la voy a desarrollar a continuación. Como he comentado al principio, este proceso de aprendizaje a la hora de realizar un proyecto es bastante arduo.

Al no tener un tutor que me guiara o ayudara con todas aquellas dudas surgidas, he descubierto que la mayoría de las dudas relacionadas con mi proyecto, ya las había tenido otra persona. Por ello realizaba búsquedas en muchos foros, casi siempre en inglés ya que obtenía muchas más respuestas, como por ejemplo para instalar una extensión o un paquete, cómo realizar bien un formulario, o cómo es el funcionamiento del framework de Visual Studio. En el caso de Visual Studio, el manual documentado en Internet[1] donde explica todo lo relacionado con el framework tanto en su funcionamiento como en el de los procedimiento y tipos de objetos relacionados con C#, me ayudó mucho ya que no sólo hay una extensa explicación sino que además suele tener múltiples ejemplos que me ayudaron a comprenderlo.

El segundo paso que realizaba era buscar una página que compilara online C#[2] para probar todos esos ejemplos de los que hablaba, para comprender mejor su funcionamiento. Además cuando me surgía un error, Visual Studio es bastante preciso a la hora de especificar el dónde y el por qué. Si no fuera así, en un proyecto tan grande sería imposible hacerlo de manera eficiente. Aún así, los mensajes de error muchas veces eran bastante genéricos y en mi primera búsqueda me daba cuenta de que podrían ser debido a múltiples causas. Por ello me decantaba por aislar el problema siempre que se pudiera en las páginas referenciadas anteriormente donde comprobaba el lenguaje y lo ejecutaba.

Otra forma de ver los problemas era poniendo puntos de interrupción, ya que al depurar en modo debug me resultó muy útil para reconocer el error y por qué ocurría. Si esto no funcionaba pasaba a ejecutar el código, línea a línea viendo exactamente todo lo que sucedía.

Uno de los problemas que más tardé en comprender y que me funcionara fue entender cómo mover información desde un controlador a una vista de distintas formas, por lo que realicé la búsqueda pertinente[3] y después de comprenderlo decidí usar el método que más se adecuara

[1] https://docs.microsoft.com/es-es/visualstudio/windows/?view=vs-2019

[2] https://rextester.com/

[3] https://www.c-sharpcorner.com/UploadFile/abhikumarvatsa/various-ways-to-pass-data-from-controller-to-view-in-mvc/

siguiendo los criterios de funcionalidad y complejidad.

5.3. Trabajos futuros

Global Tree es el esqueleto de una gran aplicación con grandes posibilidades de mejora, ya sea por añadir otro tipo de tablas o incluir nuevas funcionalidades:

- **Información de ayuda e idiomas:** una de las primeras mejoras sería la de añadir un sistema de ayuda mediante imágenes con texto informativo en el que se explique de forma intuitiva cada pantalla con sus datos a mostrar, funciones, botones etc. De esta forma sería más sencillo el aprendizaje de la aplicación.

  Esta función ya se encuentra disponible, tan solo se necesita incluir dichas imágenes y vincularlas a cada una de las pantallas para ayudar al usuario final. En cuanto a introducir nuevos idiomas, sería tan sencillo como añadir diccionarios traduciendo los que ya tenemos.

- **Descarga de informes:** otra de las futuras mejoras sería la de añadir la descarga de documentos informativos como los calendarios, desglosando las imputaciones tanto de un usuario como de todos y la información relativa a sus estadísticas, además de introducir dicha funcionalidad para descargar la lista de usuarios o proyectos.

- **Estadísticas:** en estos momentos se puede comprobar la cantidad de horas semanales trabajadas, por lo que se podría añadir la opción de visualizarlo por horas mensuales para ayudar a conocer las horas imputadas por trabajador y también se podría añadir la cantidad de horas invertidas en cada proyecto vinculado para evaluar si las horas previstas en el mismo se están cumpliendo de forma correcta, y así mejorar la estimación en futuros proyectos.

- **Roles:** la mejora más significativa tiene que ver con el sistema de roles. Ahora el sistema tiene un rol de administrador y otro de usuario. Manteniendo el rol administrador, se podría añadir el rol gestor para que lleve a cabo los pagos teniendo acceso a los datos de los empleados, el rol cliente por si un cliente quisiera acceder a la evolución del proyecto limitando los permisos de acceso a las secciones que no se quieren mostrar por cualquier

motivo, el rol comercial que podría viajar y crear un cliente accediendo a la aplicación desde cualquier lugar (recordemos que esta función actualmente está reservada sólo al administrador), o el rol empleado que sustituiría al rol usuario

- **Seguridad:** en cuanto a seguridad en este momento usamos la MD5, ya que es la criptografía que se ha usado en los últimos años, la cual es relativamente sencilla de implementar y basado en 128 bits lo que conlleva operaciones para romperlo. Pero el algoritmo de cifrado MD5 como vulnerable o *oficialmente roto* data de 2004, cuando Xiaoyun Wang y su equipo anunciaron el descubrimiento de colisiones de hash para MD5 [9].

  Esto conlleva cierto riesgo y por consiguiente una futura mejora sería evolucionar a SHA que se basa en 168 bits, pero se antoja vulnerable con lo que en concreto utilizaremos SHA-256 o SHA-512 [6]. A modo de implementación no supone un gran problema ya que ya existen bibliotecas especializadas como System.Security.Cryptography.SHA512.

- **Mensajes:** por último otra de las posibles mejoras sería añadir sistema interno de comunicación entre los miembros de la empresa mediante mensajes de texto que guardaríamos en la base de datos basado en las alertas, e incluso la opción de añadir un chat entre usuarios.

Bibliografía

[1] Documentación de .net framework. https://docs.microsoft.com/es-es/dotnet/framework/.

[2] Referencia de c#. https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/.

[3] E. de los Trabajadores. Real decreto legislativo 2/2015, de 23 de octubre, por el que se aprueba el texto refundido de la ley del estatuto de los trabajadores. *Boletín Oficial del Estado*, (255), 2015.

[4] L. Esposito. Good practices for asp.net mvc apps.

[5] B.-A. Guérin. *ASP. NET con C# en Visual Studio 2017 Diseño y desarrollo de aplicaciones Web*. Ediciones Eni, 2018.

[6] S. Gupta, N. Goyal, and K. Aggarwal. A review of comparative study of md5 and ssh security algorithm. *International Journal of Computer Applications*, 104(14), 2014.

[7] M. Otto and J. Thornto. Getbootstrap. http://getbootstrap.com, 2016.

[8] J. Thibault Aranda, J. L. Briz Velasco, J. L. Fandos, and J. M. Álvarez López. El teletrabajo. *Acciones e investigaciones sociales*, (8):0203–233, 1998.

[9] X. Wang, D. Feng, X. Lai, and H. Yu. Collisions for hash functions md4, md5, haval-128 and ripemd. *IACR Cryptology ePrint Archive*, 2004:199, 2004.