

Prototipo de aplicación en la nube para gestionar la tasación y facturación de consumo telefónico de Operadores de Telecomunicaciones

Proyecto Fin de Carrera

15 de junio de 2019

Autor: Jonatan Rodríguez Suárez

Tutores: Francisca Quintana, Carmelo Cuenca



UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática



Fdo.: Jonatan Rodríguez Suárez

Fdo.: Francisca Quintana Domínguez

Fdo.: Carmelo Cuenca Hernández

CONTENIDO

1	Introducción	4
2	Objetivos.....	5
3	Mockups y casos de uso	6
3.1	Casos de uso	6
3.1.1	Gestión de proveedores de telefonía	6
3.1.2	Contratación.....	7
3.1.3	Telefonía	7
3.1.4	Gestión de usuarios	7
3.2	Mockups	9
3.2.1	Plantilla	9
3.2.2	Administración.....	10
3.2.3	Cliente.....	14
3.2.4	Comercial.....	16
4	Entorno de desarrollo y despliegue	20
4.1	Requisitos para la realización del proyecto	20
4.1.1	Recursos hardware	20
4.1.2	Recursos software	20
4.2	Creación de una aplicación básica	23
4.2.1	Instalación del entorno de trabajo.....	23
4.2.2	Creación de una aplicación web con .Net Core	25
4.3	Despliegue de una aplicación básica con Azure App Service	26
4.3.1	Configuración del entorno de despliegue en Microsoft Azure	26
4.4	Despliegue de una aplicación con persistencia de datos.....	28
4.4.1	Configuración de Entity Framework	28
4.4.2	Modelo de datos de ejemplo.....	29
4.4.3	Controlador de MVC.....	30
4.4.4	Despliegue en el entorno de producción.....	31
5	Subida de ficheros CSV de MásMóvil.....	33
5.1	Gestor documental.....	33
5.2	MVC para el fichero CSV de llamadas de MásMóvil	34
5.2.1	Inyección de dependencia del gestor documental	34
5.2.2	Vista de índice de ficheros CSV de MásMóvil	35
5.2.3	Subida de fichero CSV de MásMóvil	36
5.2.4	Visualización del fichero CSV de MásMóvil.....	38



6	Importar fichero CSV de MásMóvil a la base de datos	40
6.1	Estado de importación de los ficheros.....	40
6.2	Importación de los eventos telefónicos desde el fichero CSV de MásMóvil.....	42
6.3	Visualización de eventos telefónicos importados en base de datos.....	46
7	Tarifas	49
7.1	Definición del modelo de datos	49
7.2	MVC de tarifas	50
8	Abonado de telefonía	53
8.1	Definición del modelo de datos	53
8.2	MVC de abonado de telefonía	56
8.3	Informe de eventos telefónico sin abonado vigente	59
9	Cálculo de facturas	61
9.1	Definición del modelo de datos	61
9.2	MVC para facturación	62
9.2.1	Modificación del controlador de abonados para generar facturas por abonado	62
9.2.2	MVC de facturas	63
9.3	Informe de abonados telefónicos con facturas pendientes de emisión	67
10	Autenticación y Seguridad	68
10.1	Middleware de autenticación y servicios.....	68
10.1.1	Configuración del middleware.....	68
10.1.2	Configuración de los servicios.....	69
10.1.3	Inicialización de roles y usuario administrador	71
10.2	Administración de usuarios	74
10.2.1	CRUD para la administración de usuarios.....	74
10.2.2	Asociación de comerciales a usuarios.....	75
10.3	Seguridad de acceso a abonados de telefonía.....	77
10.4	Acceso a comisiones	80
10.5	Menú de usuario.....	82
11	Referencias	85
12	Ilustraciones	86
13	Fragmentos de código	88
14	Anexos	90
14.1	Anexo I: Descripción del fichero de llamadas del proveedor MásMóvil	90

1 INTRODUCCIÓN

Actualmente existen muchas empresas que comienzan a introducirse en el mercado de la telefonía, no como operadores reales, sino como integradores de servicios, ofreciendo productos de telefonía de diferentes operadores telefónicos a sus clientes, bajo la marca o imagen de la propia empresa, que combinado con otros productos o servicios como video-vigilancia, control de flotas y muchos otros, puede resultar más atractivo para los clientes que contratar servicios con las operadoras de telefonía tradicionales.

Las operadoras que venden a otras empresas sus servicios de telefonía, tienen diferentes precios por minuto, precios de establecimiento de llamadas, cuotas de alta, bonos, etc... que pueden variar mucho entre las diferentes compañías. Esta casuística hace que a la empresa que revende telefonía le resulte complicado tener unos precios fijos para sus clientes, ya que juega con distintos costes de compra, y además, necesita de alguna plataforma que le permita retasar los consumos de las distintas operadoras para calcular el precio de venta y poder facturar a sus clientes.

Por lo general, las diferentes operadoras ofrecen ficheros CDR (Call Data Records) o API (Application Programming Interface) para obtener los consumos telefónicos, que habitualmente nos da información simplemente del número de teléfono de origen, destino, fecha y hora de la llamada y su duración. Esto ocasiona uno de los principales problemas con los que se encuentran este tipo de empresas, la facturación a sus clientes. Por lo tanto, las empresas que trabajan como integradores de servicios de telefonía, tienen la necesidad de identificar estas llamadas para poder tasarlas en función de: destino, duración y horario de la llamada, además de identificar a qué cliente pertenece ese destino para poder facturarle la llamada.

2 OBJETIVOS

Este proyecto surge como idea al encontrarme empleado en una empresa que opera como una OMV (Operadora Móvil Virtual) y que se enfrenta a los problemas descritos en el apartado anterior al trabajar como operadora virtual del proveedor de telefonía MasMóvil.

Por lo tanto, el objetivo de este proyecto es realizar un prototipo de aplicación web que permita a las empresas que están en el mercado de la reventa de telefonía, o que pretenden introducirse en él, disponer de un entorno que le permita trabajar con diferentes operadoras de telefonía y calcular la facturación de sus clientes.

Los principales objetivos del proyecto son:

- Proporcionar un entorno a las empresas donde poder crear y gestionar la base de datos de sus clientes, las líneas de teléfono que tienen activas y las tarifas de venta.
- Que el sistema sea capaz de importar consumos telefónicos de diferentes operadoras, retasar las llamadas en función de las tarifas definidas por la empresa revendedora y calcular la facturación telefónica de sus clientes.
- Permitir a los clientes acceder a la plataforma con acceso limitado a su información, así como al detalle de consumo de sus líneas.

Estos objetivos, serán llevados a cabo centrándonos en el proveedor de telefonía MásMóvil, implementando la importación de ficheros de llamadas de dicho proveedor y cuya especificación se detalla en el *Anexo I: Descripción del fichero de llamadas del proveedor MásMóvil*.

3 MOCKUPS Y CASOS DE USO

3.1 CASOS DE USO

En este proyecto se pretende realizar un prototipo de aplicación que será una plataforma única donde podrán acceder tanto el personal de administración de la empresa, los comerciales que trabajen para ella y los clientes que tengan contratadas líneas móviles con la dicha empresa.

Por lo tanto, se definirán tres roles de aplicación que serán:

- Gestor: Es el personal de administración de la empresa
- Comercial: Trabaja para la empresa captando nuevos clientes
- Cliente: Tiene contratada alguna línea de telefonía con la empresa

3.1.1 Gestión de proveedores de telefonía

A continuación, en la Figura 1, se muestran los casos de uso que un usuario *Gestor* podrá realizar para manejar los ficheros de llamadas de MásMóvil.

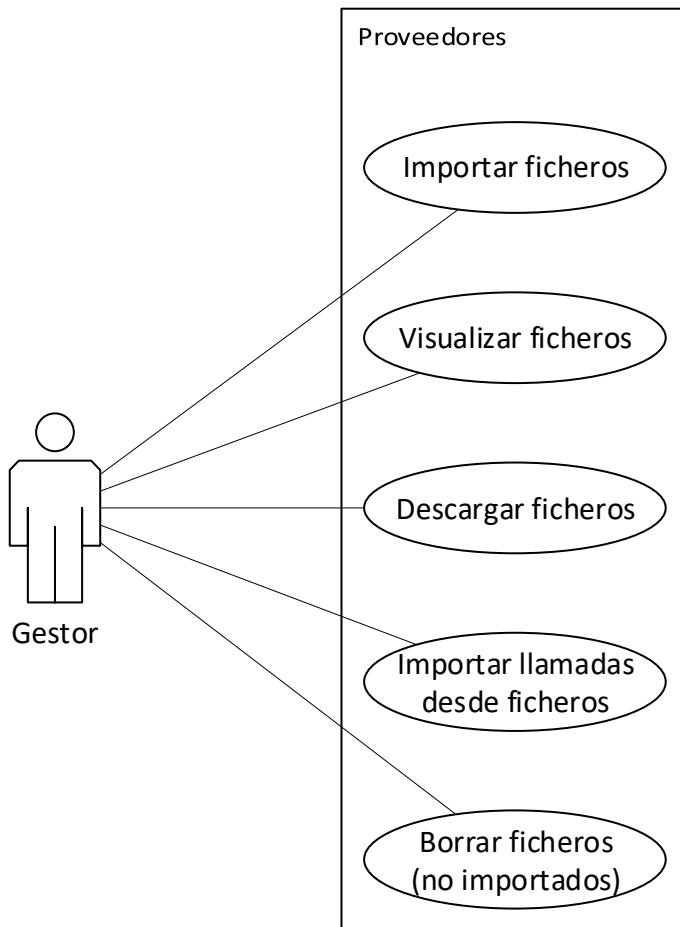


Figura 1: Casos de uso del gestor en el subsistema de proveedores

3.1.2 Contratación

El usuario gestor dispondrá de funcionalidades que le permitirán definir las tarifas, así como las líneas de telefonía y la tarifa asociada, que llamaremos abonado y que será la tarifa vigente para una línea determinada.

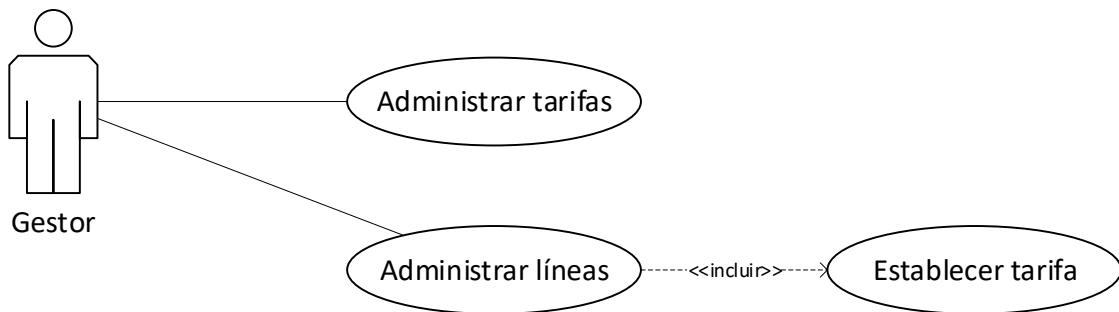


Figura 2: Casos de uso del gestor en el subsistema de Contratación

3.1.3 Telefonía

Los clientes podrán acceder a la aplicación y visualizar sus líneas, así como el registro de llamadas de éstas. También podrán visualizar la facturación generada a partir de sus líneas, así como descargarse una copia de estas.

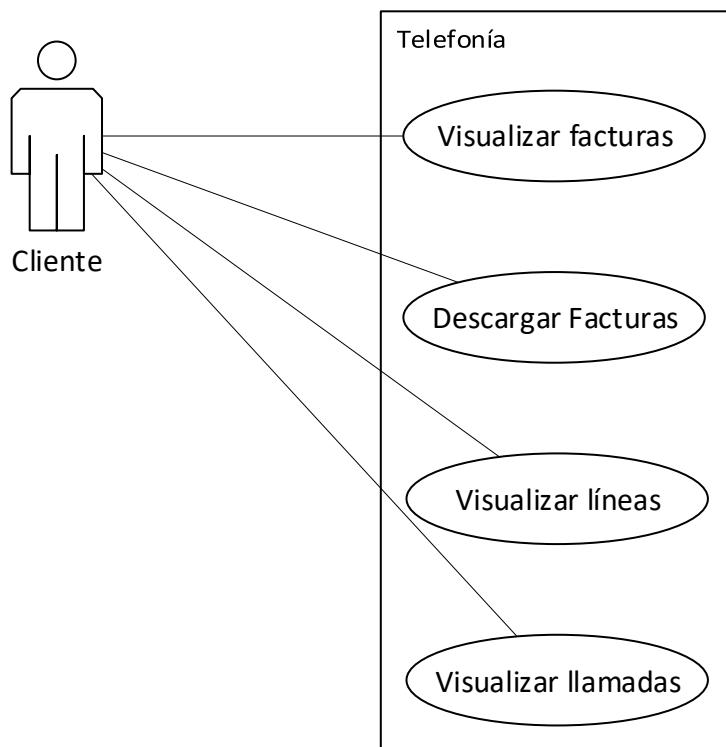


Figura 3: Casos de uso del cliente en el subsistema de telefonía

3.1.4 Gestión de usuarios

Todo usuario podrá editar los datos de su perfil los cuales podrían ser sus datos personales, datos de contacto, etc. Solo un usuario con el rol de administración (gestor) podrá dar de alta nuevos usuarios en la aplicación, asignar roles y establecer los comerciales asociados a cada cliente tal y como se observa en la Figura 4.

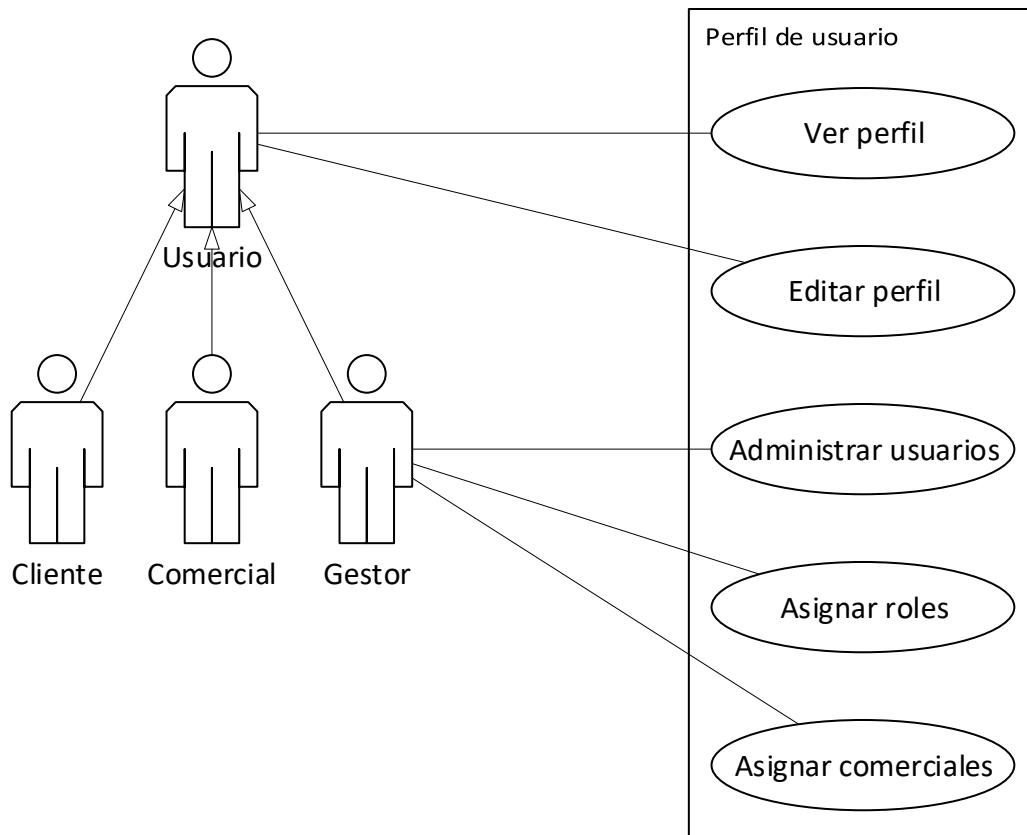


Figura 4: Casos de uso en el subsistema de usuarios

3.2 MOCKUPS

3.2.1 Plantilla

Al tratarse de una aplicación de gestión, se optará por un diseño con barra superior donde se mostrará el usuario ha iniciado sesión, además de un menú lateral donde aparecerán los enlaces a los que el usuario tiene acceso según su rol en la aplicación. En la Figura 5 que se muestra a continuación podemos observar un boceto de cómo sería la plantilla.

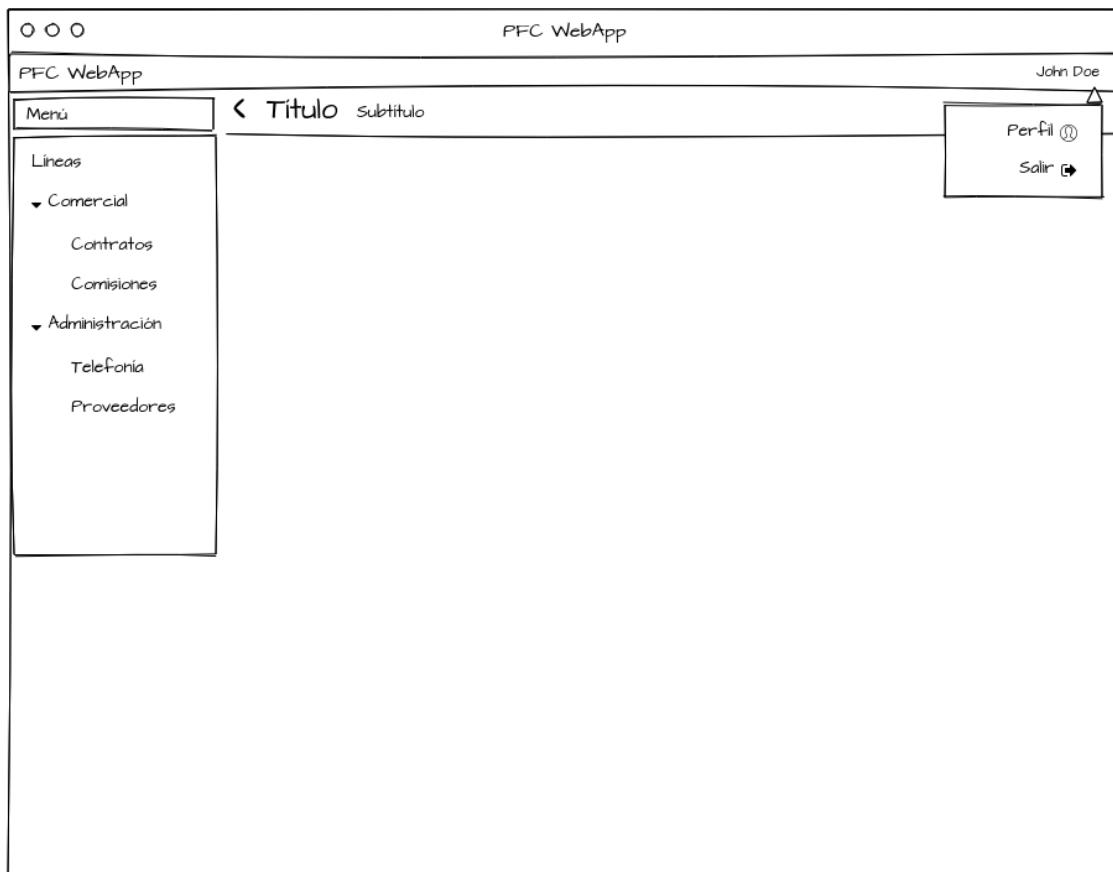


Figura 5: Mockup de la plantilla de la aplicación

Al ser una aplicación con control de acceso por usuario, también se dispondrá de una pantalla para realizar el inicio de sesión que podría ser similar al boceto que se muestra a continuación en la Figura 6.

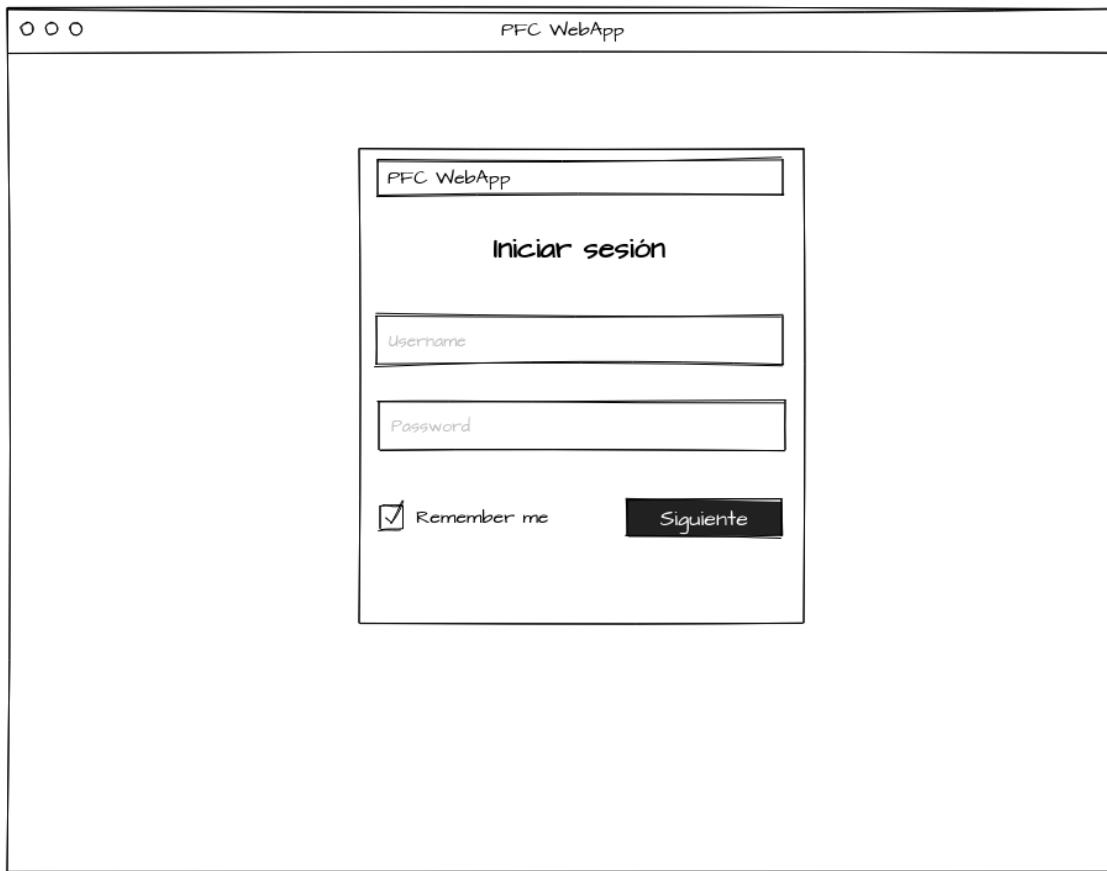


Figura 6: Mockup de la pantalla de inicio de sesión

3.2.2 Administración

3.2.2.1 Gestión de proveedores

Una de las tareas a realizar por los usuarios con rol de administración será cargar los ficheros de llamadas del proveedor MásMóvil, que es el proveedor de ejemplo que se va a usar para la elaboración de este proyecto.

Para ello, se dispondrá de una pantalla similar a la que se muestra en la Figura 7 donde podrá realizar las tareas definidas anteriormente en el apartado 3.1.1 donde se especifican los casos de uso para la Gestión de proveedores.



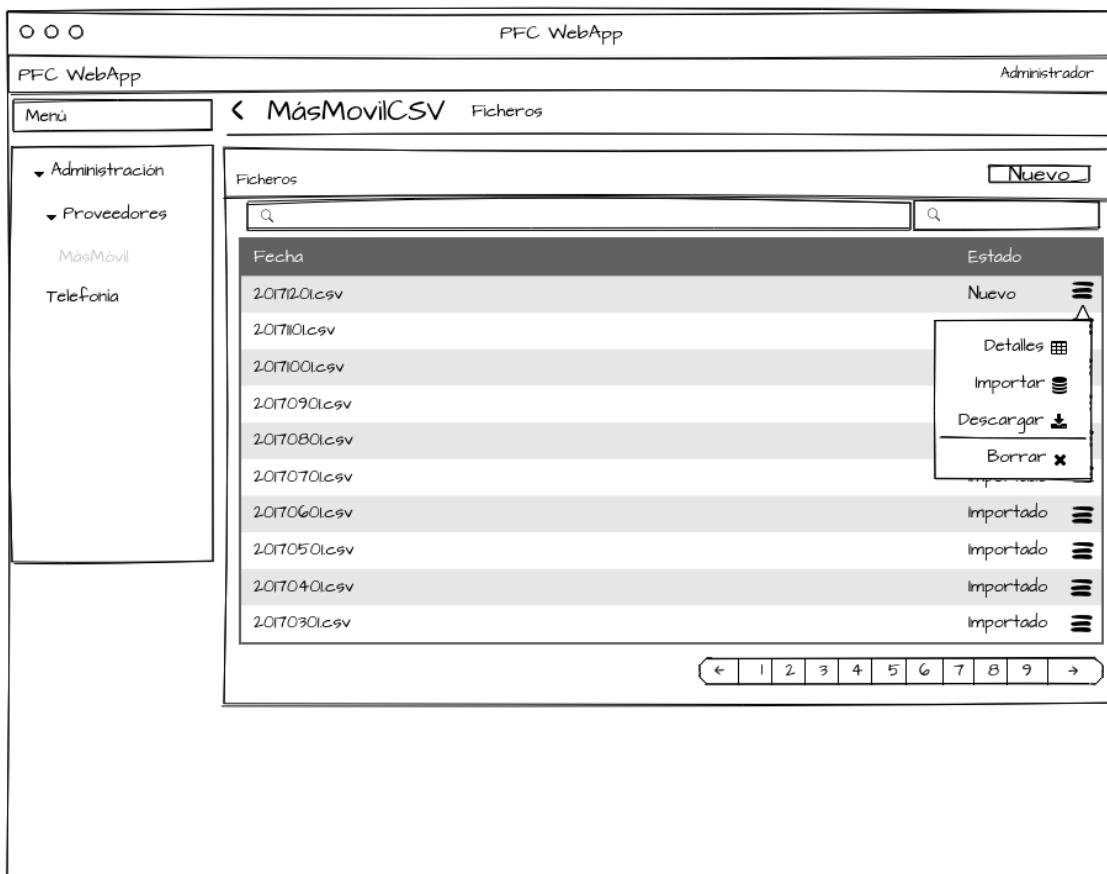


Figura 7: Mockup de la pantalla de importación de ficheros CSV

Desde esta pantalla se podrá realizar tareas comunes sobre los ficheros CSV como son: subir ficheros, visualizarlos, descargarlos e importarlos a la base de datos. También se permitirá borrar aquellos ficheros que no hayan sido importados aún a la base de datos.

Al visualizar los detalles de un fichero se mostrará una pantalla como la que se observa en la Figura 8, donde se listarán los registros del fichero paginados y se podrá establecer filtros de búsqueda para cada uno de los campos.

PFC WebApp																																																																																																																																																									
PFC WebApp																																																																																																																																																									
Administrador																																																																																																																																																									
Menú										< MásMovilCSV Detalles																																																																																																																																															
<ul style="list-style-type: none"> ▼ Administración ▼ Proveedores MásMóvil Telefonía 																																																																																																																																																									
<p>20170201CSV</p> <table border="1"> <thead> <tr> <th>Q</th><th>Q</th><th>Q</th><th>Q</th><th>Q</th><th>Q</th><th>Q</th><th>Q</th><th>Q</th><th>Q</th><th>Q</th></tr> <tr> <th>Dealer</th><th>Fecha.</th><th>Fecha</th><th>Factur.</th><th>MSISD.</th><th>Tipo_D.</th><th>Destino</th><th>Minuto..</th><th>Valor</th><th>Impues..</th><th></th></tr> </thead> <tbody> <tr><td>C0300..</td><td>2017-01..</td><td>2017-01..</td><td>20170101</td><td>6911829..</td><td>VOZ N..</td><td>691343..</td><td>2512</td><td>0.08</td><td>0</td><td></td></tr> </tbody> </table>											Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Dealer	Fecha.	Fecha	Factur.	MSISD.	Tipo_D.	Destino	Minuto..	Valor	Impues..		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0		C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0	
Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q																																																																																																																																															
Dealer	Fecha.	Fecha	Factur.	MSISD.	Tipo_D.	Destino	Minuto..	Valor	Impues..																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
C0300..	2017-01..	2017-01..	20170101	6911829..	VOZ N..	691343..	2512	0.08	0																																																																																																																																																
← 1 2 3 4 5 6 7 8 9 →																																																																																																																																																									

Figura 8: Mockup de la pantalla de visualización de ficheros CSV de MásMóvil

3.2.2.2 Tarifas

Para poder facturar a los clientes cada línea deberá incluir una tarifa vigente, por tanto, existirá un panel de administración donde se mostrará un listado con las tarifas disponibles en la aplicación. En este panel se visualizará el listado de tarifas existentes, pudiéndose crear nuevas tarifas, así como ver los detalles, modificar o eliminar las tarifas ya creadas.

Una tarifa representará la facturación que se aplicará a un abonado, conteniendo el coste de la tarifa, así como los descuentos aplicados a cada uno de los tipos de eventos telefónicos que se pueden dar, que son: llamadas, tráfico de datos y mensajes de texto.

A continuación, en la Figura 9, se muestra un prototipo de cómo sería dicho panel de tarifas.

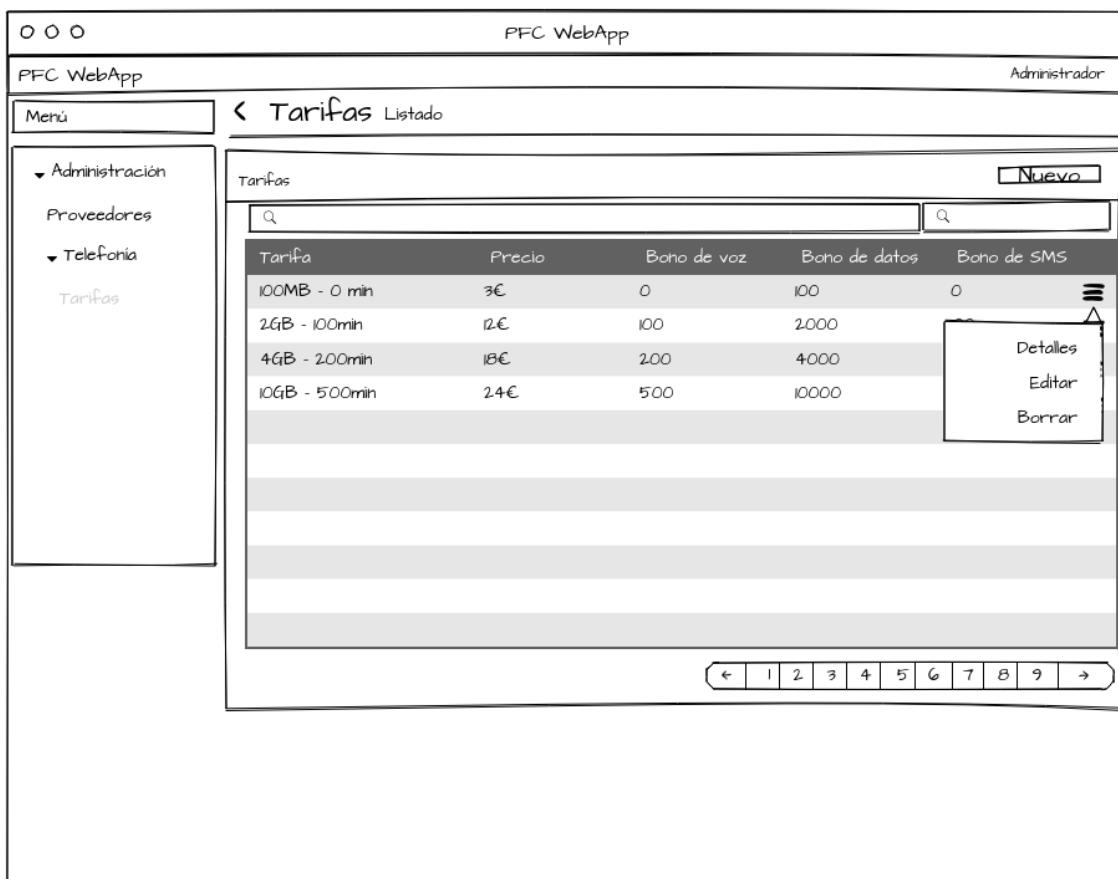


Figura 9: Mockup de la pantalla de administración de tarifas de telefonía

3.2.2.3 Abonado de línea telefónica

Para poder facturar es necesario dar de alta en la aplicación a cada uno de los abonados. El concepto de abonado consiste en la asociación de un identificador de línea telefónica con una tarifa. Además, en el abonado se establecerá un periodo de vigencia que será el período comprendido entre la fecha de alta, es decir, fecha desde la que está haciendo uso de una tarifa concreta, como la fecha de baja, que será cuando deje de usar una tarifa para realizar un cambio de tarifa, o porque el cliente haya causado baja en el uso de la línea.

En la Figura 10 se muestra un boceto de la pantalla de administración de abonados de telefonía.

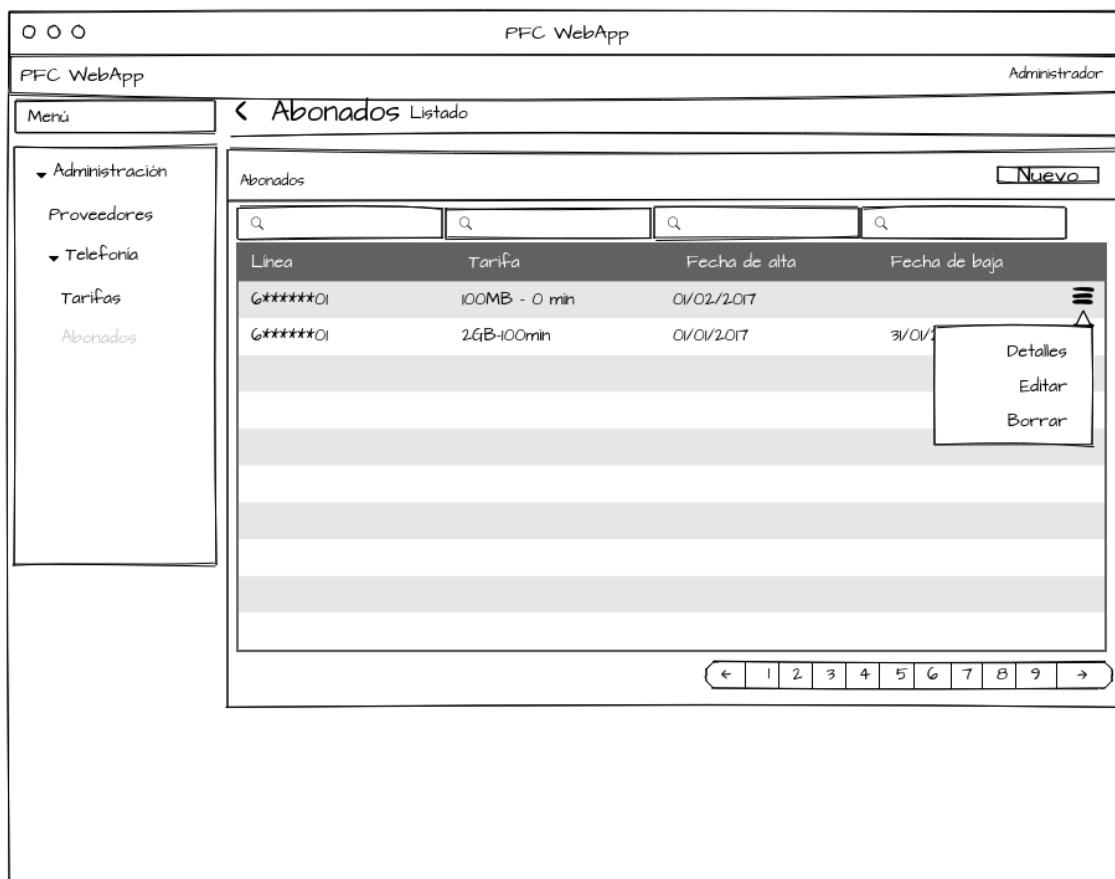


Figura 10: Mockup de la pantalla de administración de abonados de telefonía

3.2.3 Cliente

Un usuario con el rol de cliente tendrá acceso a la aplicación para poder visualizar las líneas que tiene contratadas, así como la facturación que le está generando cada una de sus líneas.

En la Figura 11, se puede observar un boceto de la pantalla donde se muestran las líneas del cliente y la tarifa que tiene contratada en cada una de ellas.

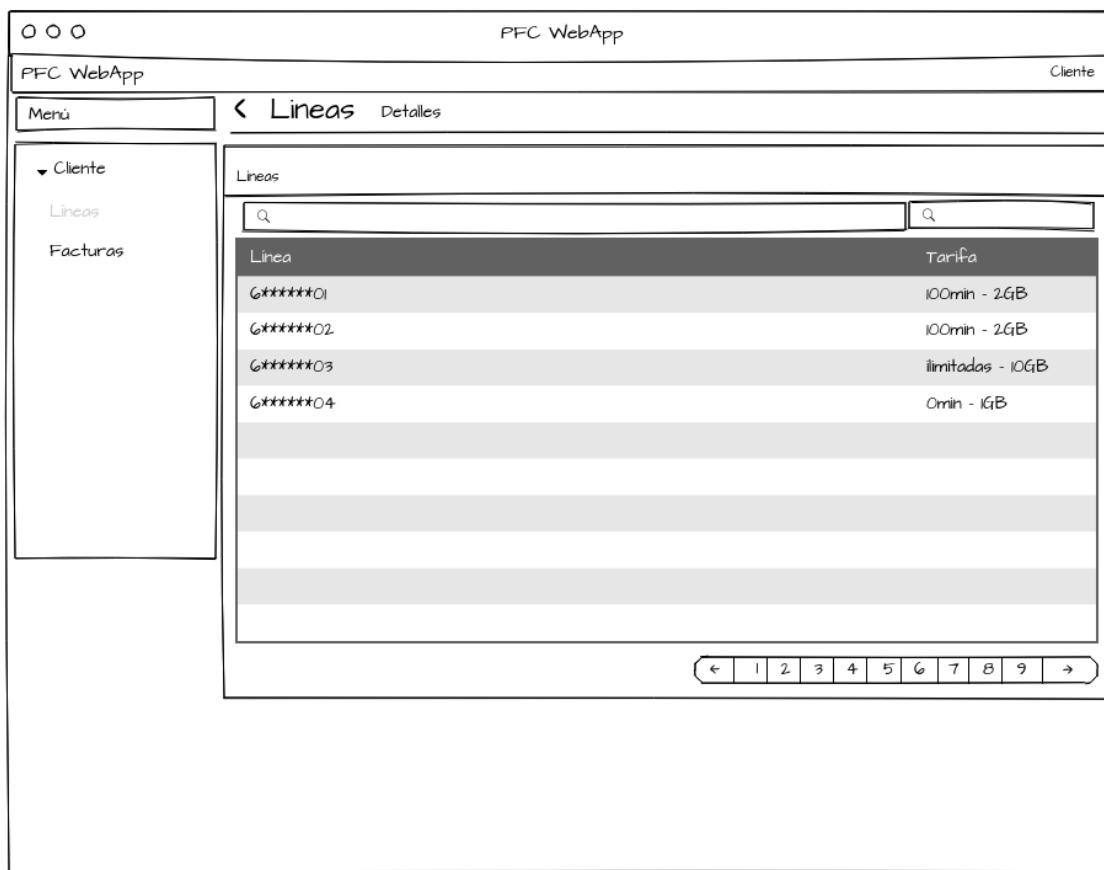


Figura 11: Mockup de las líneas de un cliente

Otra funcionalidad que aporta la aplicación a los clientes es poder visualizar la facturación que se ha generado a partir de sus líneas contratadas. Para ello, dispondrá de otra entrada en el menú de la aplicación donde acceder a sus facturas tal y como se muestra en la Figura 12.

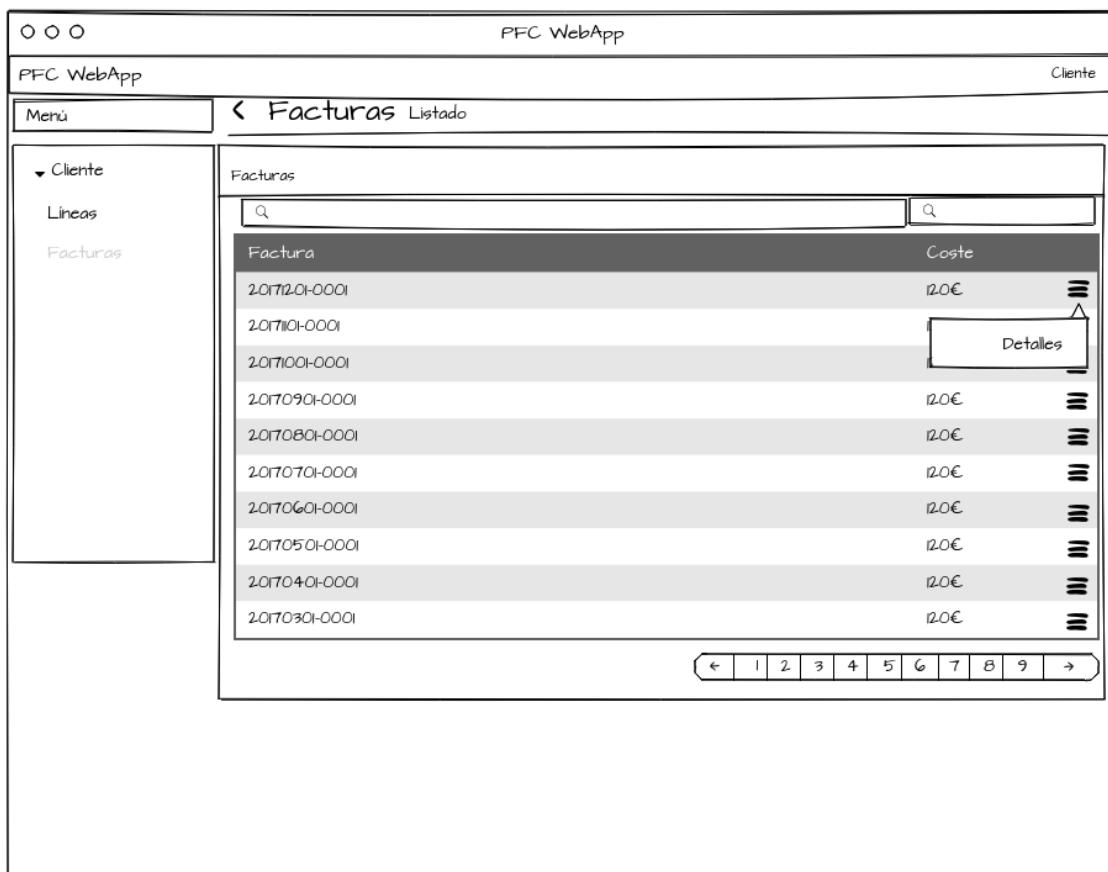


Figura 12: Mockups de la pantalla de facturas del cliente

3.2.4 Comercial

Este rol tiene como cometido conseguir clientes a la empresa, por ello, será asignado como comercial a los clientes que consiga captar y podrá hacer uso de la aplicación para visualizar las comisiones que sus clientes le han generado.

Por tanto, dispondrá de un primer menú donde visualizar el listado de clientes que tiene asignados actualmente tal y como se puede ver en la Figura 13. Desde esta pantalla podrá acceder al detalle de cada cliente para ver las líneas y bonos que tiene contratado, las comisiones que le ha generado, etc.

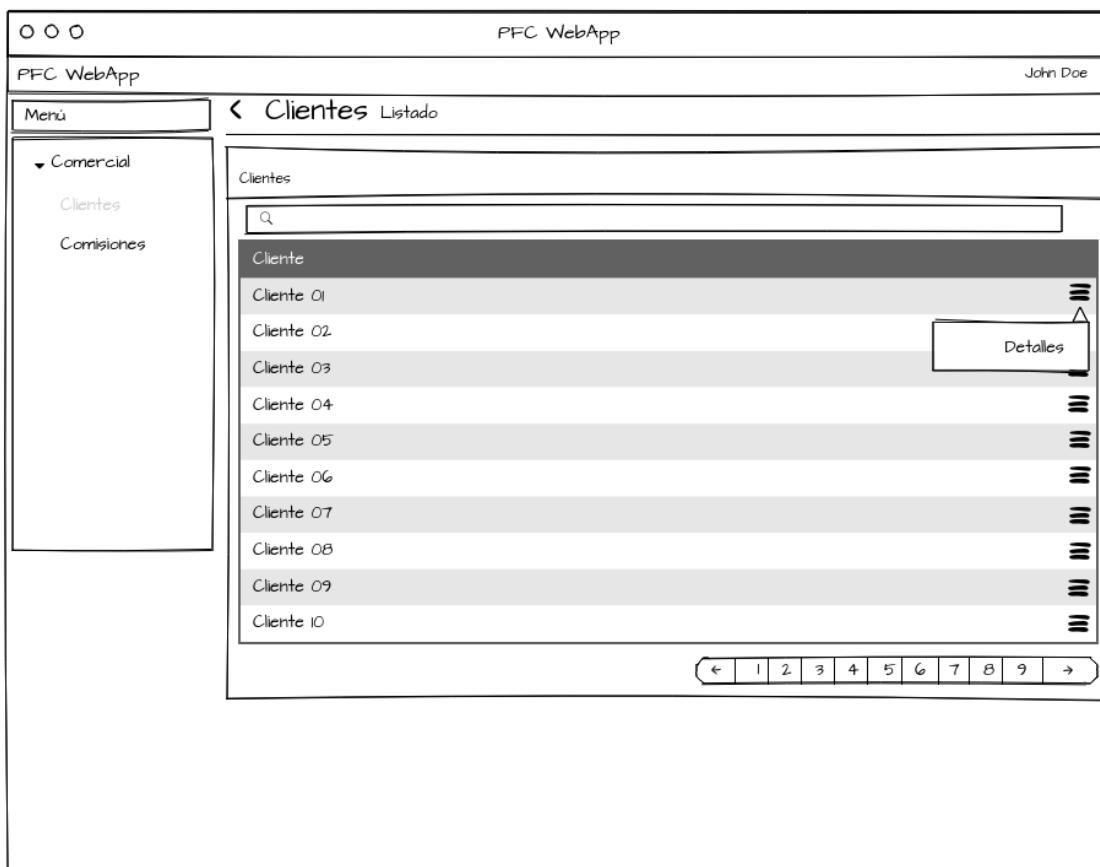


Figura 13: Mockup de la pantalla de clientes asignados a un comercial

Como uno de los usos más significativos para un comercial es el cálculo de su comisión, dispondrá de una pantalla donde podrá visualizarlas tal y como se puede ver en la Figura 14. Estas comisiones estarán calculadas en base a las facturas que se hayan generado para su cartera de clientes en el momento de la facturación.

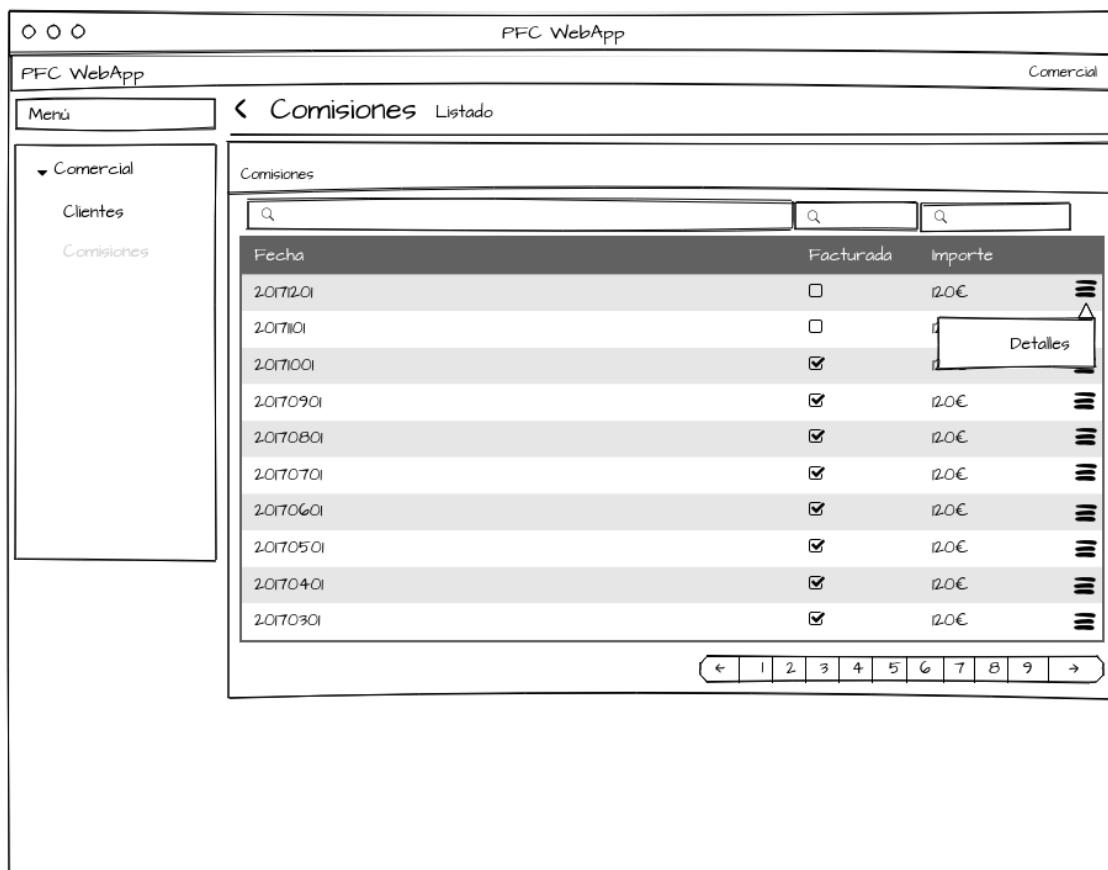


Figura 14: Mockup de la pantalla de comisiones de un comercial

Desde la pantalla de comisiones, el comercial podrá acceder al detalle de cada comisión donde visualizará un desglose de la comisión que le ha generado cada cliente como se muestra en la Figura 15.

The mockup shows a web application interface for managing commissions. At the top right, there are three circular icons (two blue, one orange) and the text "PFC WebApp". On the far right, it says "Comercial". The main header is "PFC WebApp" with a back arrow and "Comisiones Detalles".
On the left, a sidebar menu has "Comercial" expanded, showing "Clientes" and "Comisiones". Under "Comisiones", the ID "20171001" is listed. The main content area has fields for "Fecha" (2017/10/01), "Facturada" (with a checked checkbox), and "Comisión" (120€).
Below this is a table titled "Comisiones" with columns "Cliente" and "Importe". It lists 10 clients with their respective commission amounts:

Cliente	Importe
Cliente 01	12€
Cliente 02	10€
Cliente 03	24€
Cliente 04	5€
Cliente 05	8€
Cliente 06	12€
Cliente 07	18€
Cliente 08	1€
Cliente 09	7€
Cliente 10	33€

At the bottom, there is a navigation bar with page numbers from 1 to 9.

Figura 15: Mockup de la pantalla de detalles de una comisión



4 ENTORNO DE DESARROLLO Y DESPLIEGUE

4.1 REQUISITOS PARA LA REALIZACIÓN DEL PROYECTO

En este apartado, se mostrará el conjunto de herramientas utilizados para la realización de este proyecto, las cuales se agrupan en recursos hardware y recursos software:

4.1.1 Recursos hardware

A continuación, se detallan los recursos hardware que han sido necesarios para la realización de este proyecto.

4.1.1.1 *Estación de trabajo*

Para la elaboración de este proyecto se ha hecho uso de un ordenador personal, cuyas características técnicas, detallamos a continuación:

- Procesador: Procesador Intel® Core™ i7-8700
- Memoria: 16GB DDR4
- Almacenamiento: 250GB SSD

4.1.1.2 *Servidor*

Para el despliegue de la aplicación se usará la plataforma Microsoft Azure. Esta plataforma se encuentra alojada en los Data Centers de Microsoft y ofrece los siguientes servicios de computación en la nube:

- Software como servicio (en inglés software as a service, SaaS)
- Plataforma como servicio (en inglés platform as a service. PaaS)

4.1.2 Recursos software

4.1.2.1 *Sistema Operativo*

El sistema operativo utilizado durante la realización de este proyecto ha sido Microsoft Windows 10 Pro 64Bits, aunque podría usarse Linux y MacOS ya que tanto el framework como el editor de código utilizados para el desarrollo de la aplicación son multiplataforma.

4.1.2.2 *Editores*

Para la elaboración de este proyecto se ha hecho uso de varios editores: editores de código, editores de diagramas, procesadores de textos, etc.

4.1.2.2.1 Visual Studio Code

Para el desarrollo de este proyecto se ha utilizado como editor de código fuente el editor Visual Studio Code (1), desarrollado por Microsoft para Windows, Linux y macOS.

Este editor incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto,¹² aunque la descarga oficial está bajo software propietario.

4.1.2.2.2 Microsoft® Office Word

Para la elaboración de la memoria se ha utilizado el procesador de textos Microsoft® Word, integrado en el paquete de ofimática Microsoft® Office (2).

4.1.2.2.3 Microsoft® Office PowerPoint

Para la elaboración de la presentación se ha utilizado el editor de presentaciones Microsoft® PowerPoint, integrado en el paquete de ofimática Microsoft® Office (2).

4.1.2.2.4 Microsoft® Office Visio

Para la elaboración de los gráficos y diagramas incluidos en esta memoria y la presentación de este proyecto se ha utilizado el editor de gráficos vectoriales Microsoft® Office Visio® que también forma parte de la familia de productos Microsoft® Office (2).

4.1.2.3 Herramientas y entornos de trabajo

A continuación, se describen el conjunto de herramientas y entornos de trabajo utilizados en la elaboración de este proyecto.

4.1.2.3.1 Git

Git (3) es una herramienta de software libre para el control de versiones. Surgió en la comunidad del desarrollo de Linux e impulsada por esta, en particular por Linus Torvalds, y cuyos objetivos para esta nueva herramienta eran los siguientes:

- Velocidad
- Diseño sencillo
- Gran soporte para desarrollo no lineal
- Completamente distribuido
- Capaz de manejar grandes proyectos eficientemente

4.1.2.3.2 .Net Core

.NET Core es una plataforma de desarrollo de uso general de cuyo mantenimiento se encargan Microsoft y la comunidad .NET en GitHub. Es multiplataforma u de código abierto, admite Windows, macOS y Linux y puede usarse en escenarios de dispositivo, nube, IoT e incrustados.

Los lenguajes C#, Visual Basic y F # pueden usarse para escribir aplicaciones y bibliotecas para .NET Core. Los compiladores se ejecutan en .NET Core, lo que permite desarrollar para .NET Core en cualquier lugar donde se ejecute.

4.1.2.3.3 ASP.NET Core y ASP.NET Core MVC

ASP.NET Core (4) es un Framework multiplataforma, de código abierto y de alto rendimiento, que tiene como finalidad compilar modernas aplicaciones conectadas a Internet y basadas en la nube.

Con ASP.NET Core se puede:

- Compilar servicios y aplicaciones web, aplicaciones de IoT y back-ends móviles.
- Usar sus herramientas de desarrollo favoritas en Windows, macOS y Linux.
- Efectuar implementaciones locales y en la nube.

ASP.NET Core MVC (5) es un framework que permite compilar aplicaciones web y API mediante el patrón de diseño Modelo-Vista-Controlador sobre ASP.NET Core.

4.1.2.3.4 Entity Framework Core

Entity Framework Core (6) es una versión ligera, extensible y multiplataforma de la popular tecnología de acceso a datos Entity Framework.

EF Core puede servir como asignador relacional de objetos (O/RM), lo que permite a los desarrolladores de .NET trabajar con una base de datos mediante objetos .NET y eliminar la mayoría del código de acceso a los datos que normalmente deben escribir.

EF Core es compatible con muchos motores de base de datos.

4.1.2.3.5 LINQ

Language-Integrated Query (LINQ) (7) es el nombre de un conjunto de tecnologías basadas en la integración de capacidades de consulta directamente en el lenguaje C#.

Tradicionalmente, las consultas con datos se expresaban como cadenas simples y, además, se ha de aprender un lenguaje de consultas diferente para cada tipo de origen de datos: bases de datos SQL, documentos XML y varios servicios web, entre otros.

Con LINQ, una consulta es una construcción de lenguaje de primera clase, como clases, métodos y eventos. Las expresiones de consulta se escriben con una sintaxis de consulta declarativa, pudiendo realizar operaciones de filtrado, ordenación y agrupamiento en orígenes de datos con el mínimo código, para consultar y transformar datos de bases de datos SQL, conjuntos de datos de ADO .NET, secuencias y documentos XML y colecciones .NET.

4.1.2.3.6 JQuery

JQuery (8) es una biblioteca multiplataforma de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

4.1.2.4 *Tecnologías*

4.1.2.4.1 HTML

HTML (9), sigla en inglés de HyperText Markup Language, hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros.

4.1.2.4.2 CSS

CSS (10), siglas en inglés de Cascading StyleSheets, es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de los documentos web, e interfaces de usuario escritas en HTML.

4.1.2.4.3 JavaScript

JavaScript es un lenguaje de programación interpretado que se utiliza principalmente del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

4.1.2.4.4 JSON

JSON, acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript, aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

4.1.2.4.5 C#

C# (11) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. C# tiene sus raíces en la familia de lenguajes C, y a los programadores de C, C++, Java y JavaScript les resultará familiar inmediatamente.

4.2 CREACIÓN DE UNA APLICACIÓN BÁSICA

4.2.1 Instalación del entorno de trabajo

Para poder comenzar a trabajar y crear una aplicación ASP.NET Core MVC habrá que instalar las herramientas necesarias previamente. A continuación, se detalla el proceso de instalación de las herramientas necesarias y su configuración.

4.2.1.1 SDK de .NET Core 2.0

El SDK se encuentra disponibles para su descarga y su posterior instalación en la página de Microsoft, desde la cual se puede descargar y en la cual se pueden encontrar binarios disponibles para su descarga e instalación en las plataformas Windows, Linux y MacOS.

4.2.1.2 Visual Studio Code

El editor de código podemos obtenerlo en su página web (<https://code.visualstudio.com/>) disponible también la descarga de binarios para su instalación en las diferentes plataformas que hemos mencionado antes.

En la Figura 16 se puede observar una vista previa del editor instalado y funcionando en la plataforma Microsoft Windows 10 Pro.

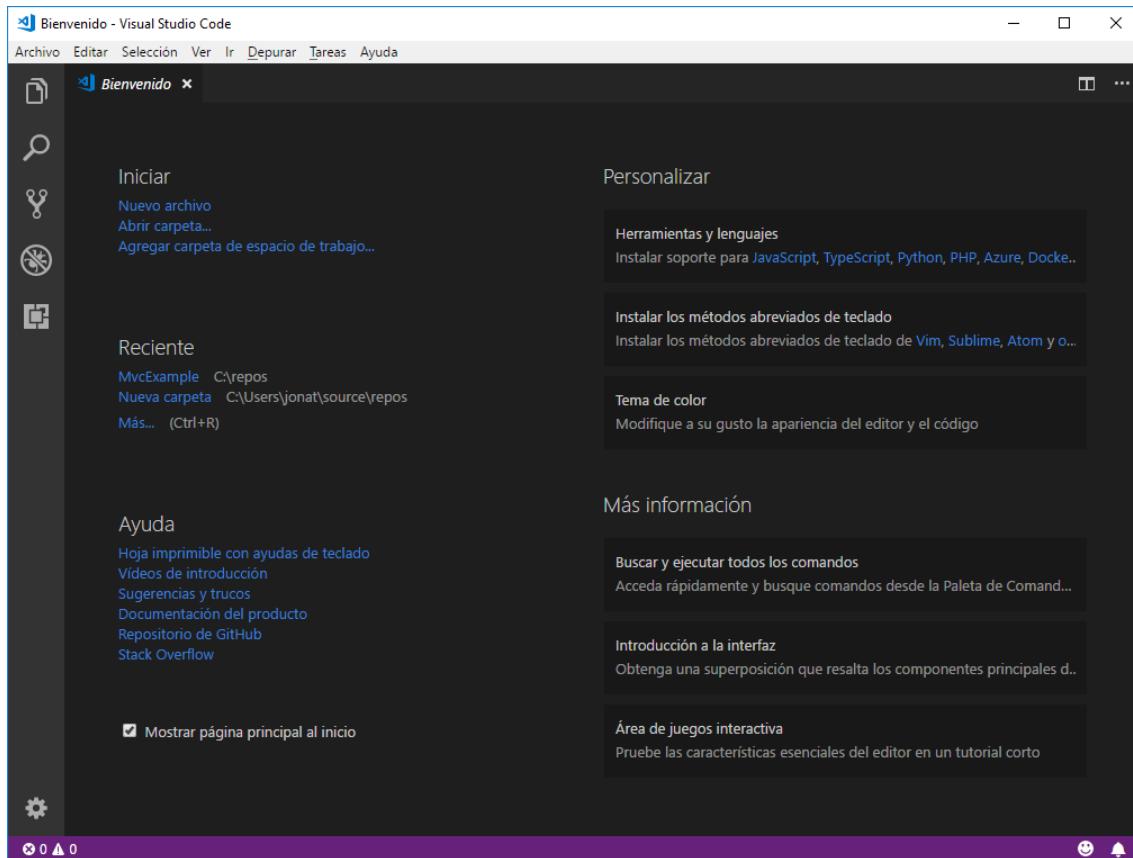


Figura 16: Vista previa de Visual Studio Code

4.2.1.3 Extensión de C# para Visual Studio Code:

Necesitaremos instalar una extensión en Visual Studio Code para añadir soporte con el lenguaje C# al editor. La extensión de C# que usaremos es ms-vscode.csharp (12) la cual proporciona las siguientes características dentro de Visual Studio Code:

- Soporte para la edición con el lenguaje C#, incluyendo resaltado de sintaxis, autocompletado (IntelliSense), navegación hacia definiciones, búsqueda de referencias, etc.
- Utilidades ligeras para el desarrollo con .Net Core.
- Soporte de depuración para .Net Core.
- Soporte para proyectos project.json y .csproj en Windows, macOS y Linux.

Para instalar esta extensión podemos hacerlo de diferentes formas que detallaremos en las secciones que describiremos a continuación.

4.2.1.3.1 VS Code Extension MarketPlace

Esta es una característica que viene integrada en Visual Studio Code por defecto y permite buscar e instalar extensiones para VS Code.

Para abrir el Marketplace podemos hacer clic en el ícono de Extensiones que aparece en la barra lateral del Visual Studio o introduciendo el atajo de teclado (Ctrl + Shift + X)

Dispondremos de un buscador que nos permitirá explorar las extensiones disponibles e instalarla una vez seleccionada, tal y como se muestra en la Figura 17.

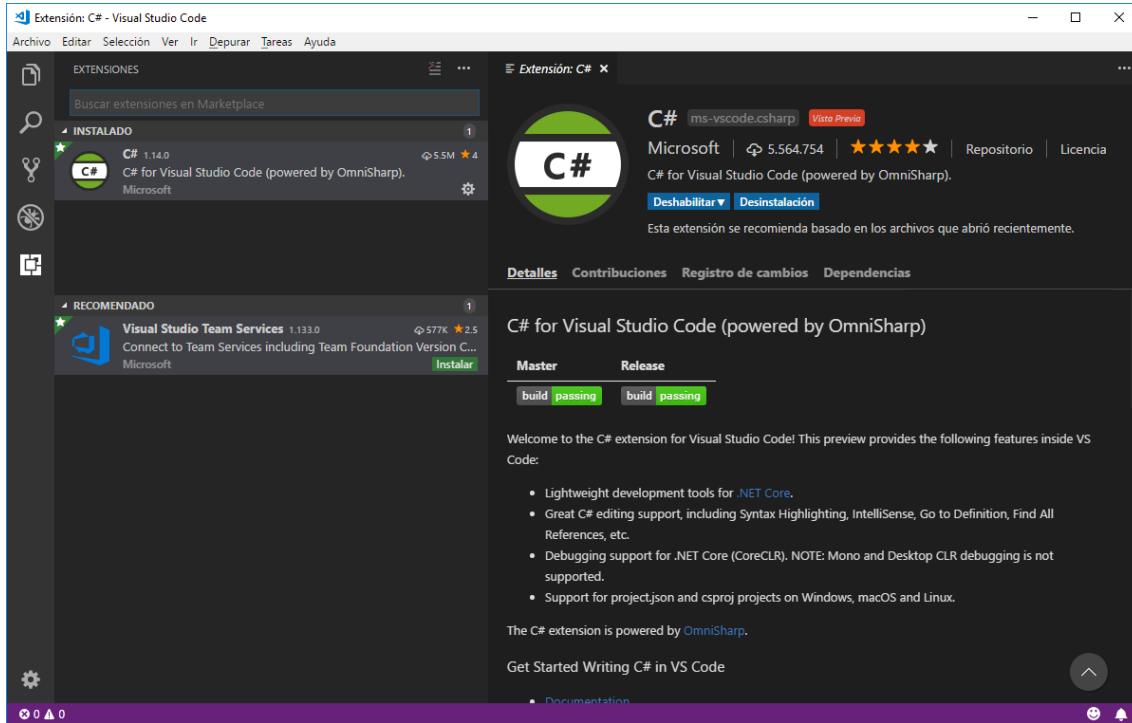


Figura 17: Instalación de extensiones desde el MarketPlace de Visual Studio Code

4.2.1.3.2 Gestión de extensiones desde la línea de comandos

Al instalar nuestro editor, tenemos disponible una variedad de comandos que podremos ejecutar desde una terminal de nuestro sistema operativo o usando la terminal empotrada de la que disponemos en nuestro editor.

Para instalar la extensión de C# podemos ejecutar la orden que podemos ver en Código 1 desde una terminal.

```
Terminal
code --install-extension ms-vscode.csharp
```

Código 1: Instalación de la extensión de C# para Visual Studio Code

4.2.2 Creación de una aplicación web con .Net Core

Desde una terminal ejecutamos los comandos que mostramos en Código 2 con los cuales crearemos una aplicación sencilla haciendo uso del framework ASP.NET Core MVC.

```
Terminal
mkdir MvcExample
cd MvcExample
dotnet new mvc
```

Código 2: Creación de una aplicación web con .NET Core

Abrimos la carpeta MvcExample en Visual Studio Code.

El editor mostrará algún mensaje de advertencia, que aceptaremos, para incluir las configuraciones necesarias para poder compilar y depurar nuestra aplicación.

Por último, para compilar y ejecutar nuestra aplicación presionaremos en **Depurar** (F5). Visual Studio Code lanzará un servidor web Kestrel y ejecutará la aplicación. Además, abrirá en un navegador la URL <http://localhost:5000> que es la usada por defecto.

La plantilla predeterminada proporciona los vínculos Inicio, Acerca de y Contacto, totalmente funcionales como se muestra en la Figura 18.

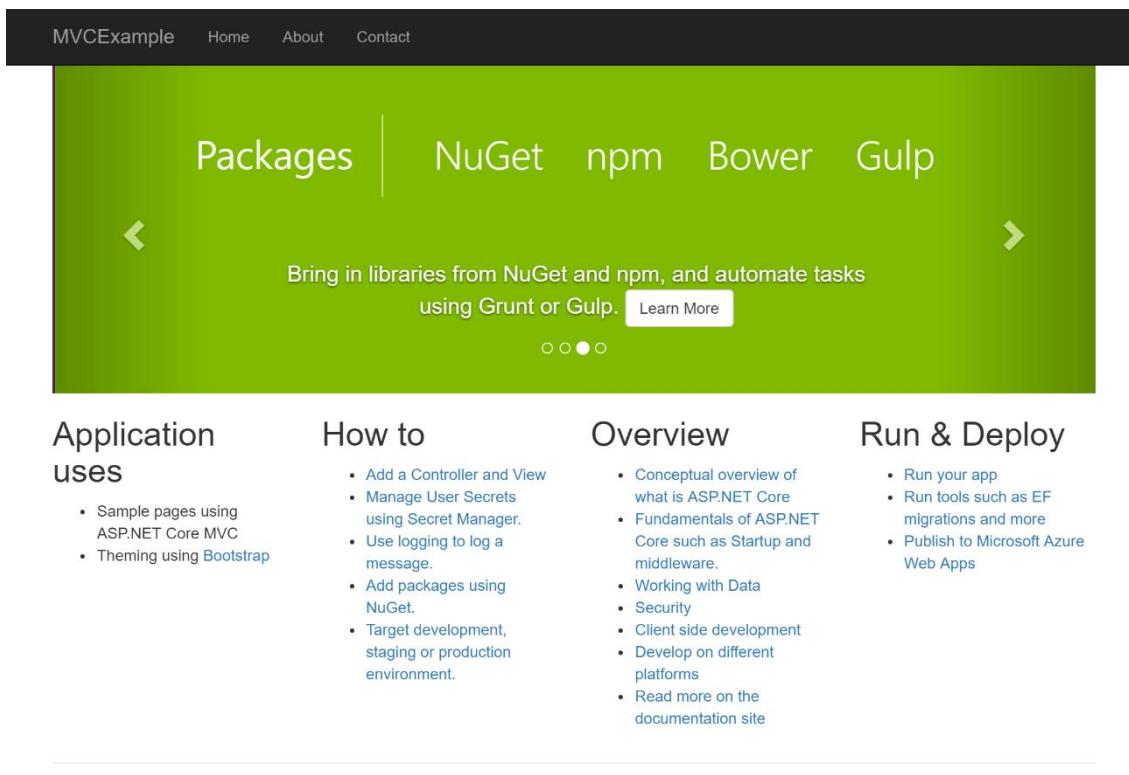


Figura 18: Vista previa de la aplicación básica en ejecución

4.3 DESPLIEGUE DE UNA APLICACIÓN BÁSICA CON AZURE APP SERVICE

Para la elaboración de este proyecto, se hará uso de los servicios SaaS proporcionados por la plataforma de Microsoft Azure. Para ello, se necesitará la herramienta de control de versiones Git y la interfaz de línea de comandos (CLI) de Azure.

4.3.1 Configuración del entorno de despliegue en Microsoft Azure

Para poder hacer uso de Microsoft Azure deberemos estar registrados en la plataforma y disponer de una suscripción.

En este caso hemos creado un perfil en Microsoft Imagine haciendo uso de la cuenta de correo de estudiante de la Universidad, la cual nos permite acceder a herramientas de desarrollo, recursos, etc., además de darnos la posibilidad de activar una cuenta gratuita de Microsoft Azure.

Para configurar el entorno de despliegue podemos hacer uso del panel web de Microsoft Azure, el cual nos ofrece un entorno gráfico donde poder gestionar nuestros recursos de Azure, o bien hace uso de las herramientas de interfaz de línea de comandos que podremos ejecutar desde una consola integrada en el propio portal de Azure o instalar en nuestro equipo los binarios para tener los comandos disponibles en la terminal de nuestro equipo.

4.3.1.1 Creación de la instancia de Azure App Service con herramientas CLI

Las herramientas de CLI para Azure podemos instalarlas en nuestro equipo descargando los binarios desde la web de Microsoft, donde encontraremos binarios disponibles para las plataformas Windows, Linux y MacOS.

Una vez instalados los binarios dispondremos de los comandos necesarios para ejecutar la CLI de Azure en el símbolo del sistema de Windows o PowerShell.

Para iniciar sesión ejecutaremos el comando que se muestra en Código 3.

```
Terminal
```

```
az login
```

Código 3: Inicio de sesión en Azure mediante CLI

Al ejecutar el comando para inicio de sesión se nos mostrará un mensaje indicando que accedamos a <https://microsoft.com/devicelogin> e insertemos el código que nos muestra el mensaje.

Accedemos al enlace donde se nos muestra un formulario como la que se muestra en la Figura 19.





Figura 19: Panel de inicio de sesión del dispositivo

A continuación, introduciremos el código generado por el comando de inicio sesión vinculando así nuestro dispositivo a nuestra cuenta de Microsoft.

Para configurar nuestro servicio de despliegue deberemos ejecutar los comandos que se muestran en Código 4. Debemos crear un grupo de recursos, un plan de servicio de aplicación y por último una aplicación que hace uso del plan y el grupo de recursos configurados anteriormente. También se establecerá una variable de entorno para indicar que la aplicación se está ejecutando en un entorno de producción, que será útil para discriminar acciones en la aplicación desde el propio código fuente.

Terminal

```
az group create --name PFCResourceGroup --location westeurope
az appservice plan create --name PFCAppServicePlan --resource-group PFCResourceGroup --location
westerurope --sku FREE
az webapp create --name PFCWebApp --resource-group PFCResourceGroup --plan PFCAppServicePlan
az webapp config appsettings set --name PFCWebApp --resource-group PFCResourceGroup --settings
ASPNETCORE_ENVIRONMENT="Production"
```

Código 4: Comandos para crear un servicio de aplicación web para el despliegue en Azure

A continuación, se definirán las credenciales de acceso y recuperaremos la Url de acceso para desplegar la aplicación con Git ejecutando los comandos que se muestran en Código 5.

Anotaremos la URL que nos devuelva el comando ya que la necesitaremos luego para configurar nuestro repositorio.

Terminal

```
az webapp deployment user set --user-name jonatanrs --password ****
az webapp deployment source config-local-git -n PFCWebApp -g PFCResourceGroup --query [url] -o tsv
```

Código 5: Definición de credenciales y recuperación de URL para Git

Por último, solo queda configurar el repositorio Git en el directorio donde tengamos el código fuente de nuestra aplicación. Para ello, desde una terminal, nos situamos en el directorio raíz de nuestra aplicación y ejecutamos los comandos que se muestran en Código 6.

Terminal

```
git init
git add --all
git commit -a -m "Primer commit"
git remote add azure https://jonatanrs@pfcwebapp.scm.azurewebsites.net/PFCWebApp.git
git push azure master
```

Código 6: Configuración del repositorio local de Git

Una vez y se haya terminado de subir el código podemos acceder a nuestra aplicación, que se está ejecutando en el servicio de Azure, accediendo desde un navegador a la dirección <http://pfcwebapp.azurewebsites.net/> la cual podemos obtener mediante la CLI de Azure ejecutando el comando que vemos en Código 7.

Terminal

```
az webapp show -n PFCWebApp -g PFCResourceGroup --query defaultHostName -o tsv
```

Código 7: Comando para obtener la Url de la aplicación desplegada

4.4 DESPLIEGUE DE UNA APLICACIÓN CON PERSISTENCIA DE DATOS

En este apartado mostraremos como configurar y desplegar una aplicación básica usando una base de datos para persistir la información de la aplicación. También mostraremos como crear un controlador con acciones para crear, obtener, actualizar, borrar y listar entidades almacenadas en nuestra base de datos.

Para acceder a la base de datos usaremos Entity Framework Core, que no es más que un ORM para la plataforma de Microsoft .NET, que permite trabajar con una base de datos mediante objetos .NET y eliminar la mayoría del código implementado habitualmente para el acceso a la base de datos.

4.4.1 Configuración de Entity Framework

Al crear nuestra aplicación usando la plantilla ya tenemos configurado el paquete Microsoft.AspNetCore.All que nos proporciona un conjunto de paquetes de uso frecuente en el desarrollos de aplicaciones web para la plataforma .NET Core.

Entre los paquetes proporcionados por este conjunto de paquetes, se incluyen los necesarios para usar Entity Framework, además de diferentes proveedores desarrollados por Microsoft que implementan el acceso a base de datos SQL Server y Sqlite, y un proveedor que persiste en memoria.

Entity Framework incluye un conjunto de comandos adicionales para la CLI que empiezan por dotnet ef. Para usar los comandos de CLI, el archivo .csproj de la aplicación debe contener la entrada que se muestra en Código 8 .



Código

```
<ItemGroup>
  <DotNetCliToolReference
    Version="2.0.3" />
</ItemGroup>
```

Código 8: Configuración de las utilidades de consola para Entity Framework

4.4.2 Modelo de datos de ejemplo

Para tener organizado el código crearemos una carpeta Data donde crearemos todos nuestros modelos de datos que usaremos con Entity Framework además de crear la clase con la que se establecerá una sesión para acceder a la base de datos.

En el fragmento Código 9 vemos la implementación de nuestra clase de ejemplo cuyos datos vamos a guardar en base de datos:

Código

```
namespace MVCExample.Data
{
    /// <summary>
    /// Modelo de datos de ejemplo
    /// </summary>
    public class Model
    {
        /// <summary>
        /// Identificador.
        /// </summary>
        public virtual int ID { get; set; }

        /// <summary>
        /// Texto.
        /// </summary>
        /// <value>
        public virtual string Text { get; set; }
    }
}
```

Código 9: Modelo de datos básico para el ejemplo de configuración de Entity Framework

Para poder establecer una sesión con base de datos que permita consultar y modificar los modelos de la aplicación que van a tener persistencia de datos, se definirá una clase que implemente la clase abstracta DbContext proporcionada por Entity Framework.

En esta clase se definen las diferentes propiedades del tipo DbSet<TEntity> que sirven como punto de acceso a las tablas de la base de datos donde se almacenan los registros para cada uno de los modelos que tendrá persistencia en la aplicación.

A continuación, en el fragmento de Código 10 se muestra la implementación de la clase DbContext con una propiedad que da acceso al repositorio de la clase BasicModel.

Código

```
using Microsoft.EntityFrameworkCore;

namespace MVCExample.Data
{
    public class ApplicationDbContext : DbContext
    {
        public virtual DbSet<BasicModel> Modelos { get; set; }
    }
}
```

Código 10: Implementación de DbContext

En el fichero Startup.cs que se encuentra en la raíz del proyecto, se debe configurar el servicio de Entity Framework. Para ello se modificará el método ConfigureServices insertando el código que se muestra en el fragmento de Código 11. También se ha configurado el servicio DbContext para



que ejecute las migraciones al ejecutar la aplicación, que será útil para que se actualice la base de datos en el entorno de producción automáticamente.

Código

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    // Configuración de entity framework.
    services.AddDbContext<Data.ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DbConnection")));
    ...
    // Ejecución automática de migraciones al inicio de la aplicación
    services.BuildServiceProvider().GetService<ApplicationDbContext>().Database.Migrate();
}
}
```

Código 11: Configuración del servicio de Entity Framework en el punto de inicio de la aplicación

Por último, para terminar de configurar Entity Framework, se establecerá una cadena de conexión en el fichero appsettings.json que se encuentra en la raíz del proyecto añadiendo la propiedad ConnectionStrings al objeto JSON tal y como se muestra en Código 12.

Código

```
{
    ...
    "ConnectionStrings": {
        "DbConnection": "Server=(localdb)\\mssqllocaldb;Database=PFCWebApp;Trusted_Connection=True;",
    },
    ...
}
```

Código 12: Configuración de la cadena de conexión con una base de datos SQL Server local

Una vez establecida la cadena de conexión se actualizará la base de datos ejecutando los comandos que se muestran en Código 13.

Terminal

```
dotnet ef migrations add InitialCreate -o Data/Migrations
dotnet ef database update
```

Código 13: Comandos para configurar el sistema de migraciones de Entity Framework.

Con el primer comando se configurará quedará inicializado el sistema de migraciones de Entity Framework y se creará un fichero de migración para el modelo de datos de ejemplo que se ha definido en el DbContext de la aplicación.

Con el siguiente comando se actualizará de forma manual (aunque anteriormente se haya configurado para que se apliquen automáticamente al ejecutar la aplicación) la base de datos aplicando las migraciones pendientes.

4.4.3 Controlador de MVC

En esta sección se mostrará cómo crear un controlador de MVC y sus vistas para acceder al modelo de datos mediante Entity Framework definido en la sección anterior.

Para el desarrollo rápido de este ejemplo se aprovecharán las herramientas de scaffolding proporcionadas por el paquete Microsoft.VisualStudio.Web.CodeGeneration.Design. Para ello se instalará el paquete y se generará el scaffold del controlador y las vistas ejecutando los comandos que se muestran en el cuadro Código 13.

Terminal



```
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design --version 2.0.4
dotnet restore
dotnet aspnet-codegenerator controller -name ModelsController -m BasicModel -dc ApplicationDbContext --relativeFolderPath Controllers --useDefaultLayout --
referenceScriptLibraries
```

Código 14: Scaffold de un controlador básico con acciones CRUD

Al ejecutar la aplicación se podrá acceder al nuevo controlador según la ruta configuradas en la aplicación. Estas rutas están definidas en la clase Startup, tal y como se muestra en Código 15 y que son de la forma /Controlador/Acción/[ID].

Código

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
    ...
}
```

Código 15: Rutas de MVC configuradas por defecto

Por tanto, al arrancar la aplicación y accediendo a la dirección <http://localhost:5000/models> en el navegador veremos una tabla que muestra un enlace para crear nuevas entradas del modelo de datos en la base de datos además de poder editar o borrar los existentes tal y como se observa en la Figura 20.

Figura 20: Vista de índice del scaffold de ejemplo

4.4.4 Despliegue en el entorno de producción

Para realizar el despliegue en producción, se creará una instancia de SQL Database en Azure.

Con los comandos que se muestran en Código 16 se creará una instancia de SQL Server (reemplazando <username> y <password> por el usuario y contraseña que se vaya a utilizar para conectar con la instancia del servidor) además de la base de datos a usar.

Terminal

```
az sql server create --name PFCSqlDatabase --resource-group PFCResourceGroup --location "West Europe" --admin-user <username> --admin-password <password>
az sql db create --resource-group PFCResourceGroup --server pfcsqldatabase --name PFC
```

Código 16: Comando para crear un servidor de SQL Server en Azure



Además de crear la base de datos, se ha de establecer una regla en el cortafuegos del servidor dejando este abierto solamente para otros recursos de Azure ejecutando la orden que se muestra en Código 17.

Terminal

```
az sql server firewall-rule create --resource-group PFCResourceGroup --server pfcsqldatabase -  
-name AllowYourIp --start-ip-address 0.0.0.0 --end-ip-address 0.0.0.0
```

Código 17: Configuración de una regla del cortafuegos del servidor

Por último, se establecerá la cadena de conexión de la base de datos en la configuración de la aplicación web usando el comando que vemos en Código 18.

Terminal

```
az webapp config connection-string set --resource-group PFCResourceGroup --name PFCWebApp --  
settings MyDbConnection='Server=tcp:pfcsqldatabase.database.windows.net,1433;Database=PFC;User  
ID=<username>;Password=<password>;Encrypt=true;Connection Timeout=30;' --connection-string-  
type SQLServer
```

Código 18: Configuración de la cadena de conexión con la base de datos

Una vez realizada la configuración, desplegamos en Azure como se explicó en la sección anterior usando los comandos de git, y se podrá comprobar que se ha desplegado accediendo a la dirección <http://pfcwebapp.azurewebsites.net/models> donde está desplegada la aplicación.



5 SUBIDA DE FICHEROS CSV DE MÁSMÓVIL

Para simplificar el proyecto se hará uso únicamente del formato de ficheros de llamadas definido por el proveedor de telefonías MásMóvil, cuyo formato se puede observar en el documento adjunto como anexo I: Descripción del fichero de llamadas del proveedor MásMóvil.

Para ello se han definido e implementado algunas herramientas que serán útiles para el desarrollo de este apartado y que se reutilizará para los siguientes.

5.1 GESTOR DOCUMENTAL

Para el almacenamiento de los ficheros se ha definido una interfaz que define un gestor documental muy simple.

En este gestor documental se dispondrá de, valga la redundancia, un Gestor documental que se encargará de crear repositorios de documentación. Estos repositorios representarán un almacén de documentación, a modo de carpeta, que permitirá realizar las funciones básicas que cualquier almacén de documentos podría realizar como son listar, crear, actualizar, leer o borrar.

En la Figura 21 se muestra un diagrama de clases UML con la definición del gestor documental que se implementará en la aplicación.

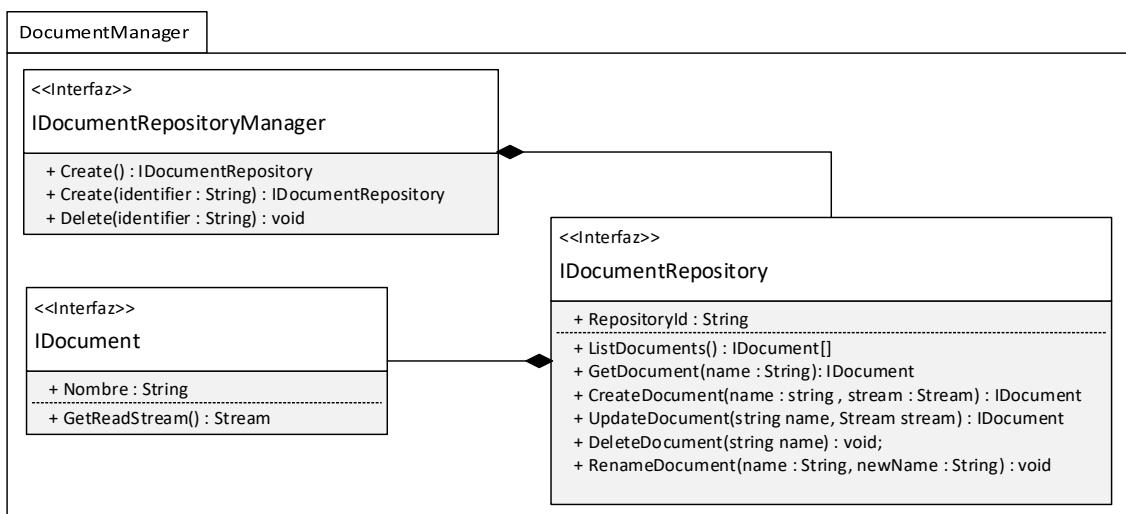


Figura 21: Diagrama de clases UML del Gestor Documental

Para el proyecto se ha realizado una implementación del gestor documental definido anteriormente que hace uso de almacenamiento en disco.

Al definir el gestor documental mediante una interfaz, se podrá implementar diversos gestores documentales haciendo uso de la persistencia que mejor convenga e inyectándose en el arranque de la aplicación cumpliendo así con los principios SOLID.

Para instanciar el Gestor documental implementado, se deberá especificar una ruta de disco al instanciar la clase que implementa la interfaz del gestor documental.

Para poder configurarlo fácilmente se añadirá en el fichero `appsettings.json` una nueva variable de configuración (tal y como se explicó para la cadena de conexión con la base de datos en apartados anteriores) que usaremos para configurar el gestor documental al inyectarlo en el

contenedor de dependencias de ASP.NET Core, quedando el fichero tal y como se muestra en Código 19.

Código
<pre>{ ... "AppSettings": { "DocumentManagerDiskProviderRepositoryPath": "Repositorios" } ... }</pre>

Código 19: Configuración de la ruta de la carpeta usada por el Gestor Documental

Para inyectar la implementación del gestor documental en el contenedor de dependencias de ASP.NET Core se añadirán las líneas que se muestran en Código 20 en el método ConfigureServices de la clase Startup.cs

Código
<pre>public void ConfigureServices(IServiceCollection services) { ... var path = Configuration.GetValue<string>("AppSettings:DocumentManagerDiskProviderRepositoryPath"); services.AddTransient<IDocumentRepositoryManager>(x => new DocumentRepositoryManager(path)); ... }</pre>

Código 20: Inyección del Gestor Documental en el contenedor de dependencias de ASP.NET Core

5.2 MVC PARA EL FICHERO CSV DE LLAMADAS DE MÁSMÓVIL

Para subir los ficheros a la aplicación implementaremos un controlador de MVC con métodos para listar, crear, descargar o borrar ficheros.

Para implementar el controlador se pueden usar las herramientas de scaffolding explicadas en secciones anteriores.

5.2.1 Inyección de dependencia del gestor documental

ASP.NET Core está diseñado para admitir y aprovecha la inyección de dependencias, resolviendo por constructor las dependencias de los controladores MVC de la aplicación.

Por lo tanto, para recibir la dependencia del gestor documental se definirá el constructor que acepte dicha dependencia por parámetro y que se almacenará en la instancia del controlador para que esté disponible en los métodos que se definirán en este. En el fragmento de Código 21 se puede observar cómo se establece la dependencia en el controlador.

Código
<pre>public class MasMovilCSVController : Controller { private readonly IDocumentRepository documentRepository; private readonly IContentTypeProvider _contentTypeProvider; public MasMovilCSVController(IDocumentRepositoryManager documentRepositoryManager, IContentTypeProvider contentTypeProvider) { documentRepository = documentRepositoryManager.Create("MasMovilCSV"); _contentTypeProvider = contentTypeProvider; } ... }</pre>

Código 21: Resolución de dependencia del gestor documental para el controlador de ficheros CSV de MásMóvil



5.2.2 Vista de índice de ficheros CSV de MásMóvil

Haciendo uso de la clase *DataTableDefinition* implementada para este proyecto, se define un método *Index* en el controlador *MasMovilCSVController* donde se configurará la tabla que va a mostrar la lista de ficheros que incluye el repositorio de documentación MasMovil.

Mediante la clase *DataTableDefinition* definimos la tabla especificando la acción del controlador que devolverá los elementos a mostrar en la tabla, las columnas que se mostrarán, así como una serie de acciones globales como subir un nuevo fichero, y acciones relativas a cada fichero como son ver los detalles del fichero, descargar el fichero o eliminar el fichero.

Código

```
public ActionResult Index()
{
    var dataTableDefinition = new DataTableDefinition(documentsQueryable)
        .WithTitle("Ficheros")
        .WithListAction(Url.Action("Index"))
        .MapColumn("Name", "Nombre", 100)
        .AddGlobalActions("Nuevo", Url.Action("Create"))
        .AddElementAction("Detalles", new DataTableDefinition.UrlActionDTO()
    {
        Action = "Details",
        Values = new Dictionary<string, string> { { "id", "Name" } }
    })
    .AddElementAction("Descargar", new DataTableDefinition.UrlActionDTO()
    {
        Action = "Download",
        Values = new Dictionary<string, string> { { "id", "Name" } }
    })
    .AddElementAction("Borrar", new DataTableDefinition.UrlActionDTO()
    {
        Action = "Delete",
        Values = new Dictionary<string, string> { { "id", "Name" } }
    });
}

return View(dataTableDefinition);
}

[HttpPost]
public JsonResult Index(int index = 0, int length = 10, string orderProperty = null, bool
inverse = false, [FromBody]IEnumerable<QueryFilter> query = null)
{
    DataTableDefinition.DataQuery data =
        DataTableDefinition.DataQueryBuilder(index, length, orderProperty, inverse, query,
documentsQueryable);
    return base.Json(data);
}

private IQueryable<object> documentsQueryable => documentRepository
    .ListDocuments()
    .Select(x => new { x.Name })
    .AsQueryable();
```

Código 22: Acción para listar los ficheros CSV de MásMovil

Ejecutando la aplicación y accediendo a la Url <http://localhost/masmovilcsv> se podrá ver el listado de documentos que hay actualmente en el repositorio tal y como se muestra en la ___ a continuación.



The screenshot shows a web application interface for managing files. At the top, there's a dark header with the text 'PFC.WebApp' and links for 'Home', 'About', and 'Contact'. On the right side of the header are 'Register' and 'Log in' buttons. Below the header, the page title is 'MasMóvil Ficheros CSV'. The main content area is titled 'Ficheros' and contains a table of files. The table has columns for 'Nombre' (Name), 'Tamaño' (Size), and 'Última modificación' (Last modified). The 'Nombre' column lists files named from '20170101.txt' to '20171001.txt'. To the right of each file name is a small icon with three dots, and next to it is a context menu with options 'Detalles', 'Descargar', and 'Borrar'. At the bottom of the table, there are navigation buttons for 'Anterior', 'Siguiente', '1', '2', and 'Siguiente'. Above the table, there's a dropdown for 'Elemento por página:' set to '10'. A 'Nuevo' button is located above the table. At the very bottom of the page, there's a copyright notice: '© 2018 - PFC.WebApp'.

Figura 22: Vista de índice con los ficheros CSV de MasMóvil

5.2.3 Subida de fichero CSV de MásMóvil

Para subir un fichero CSV a la aplicación, en el controlador `MasMovilCSVController`, se ha creado un método de acción `Create` que renderiza una vista en el cliente con un formulario para subir el fichero, además de la acción `Create` con el verbo http POST, donde el servidor recibirá el fichero subido.

En este paso se controla que el fichero tenga un formato válido, no permitiendo almacenar ficheros en el servidor de los cuales no se haya podido extraer correctamente todas sus filas.

La aplicación validará el fichero completo mostrando un resumen de las líneas que no ha sido capaz de convertir al tipo de datos esperado para que el usuario pueda corregir el fichero o contactar con su proveedor de telefonía y solucione el problema o informe de posibles cambios en el formato de los registros para poder adaptar la aplicación.

En la Figura 23 se puede apreciar un pantallazo que muestra los errores de validación tras intentar subir un fichero que no tiene un formato válido.

PFC.WebApp Home About Contact Register Log in

MásMóvil CSV Subir

Errores de validación

- Ocurrió un error al parsear la línea 1 del fichero CSV. El número de columnas no coincide. Línea : C03004-0000|03/08/2015 10:26:11|01/07/2015 13:26:48|20150801|607561835|SMS NACIONAL|601219484|0,000000|0,080000|0,000000|asd

Fichero

Ningún archivo seleccionado

© 2018 - PFC.WebApp

Figura 23: Vista con el formulario de subida de fichero CSV mostrando errores de validación tras subir un fichero incorrecto.

Para todo ello se ha implementado una clase que representa el modelo de datos para un registro del formato de ficheros de MásMóvil. Esta clase, cuya parte del código fuente se puede observar en el fragmento de Código 23, contiene un conjunto de propiedades tal y como han sido especificadas por el proveedor, además de métodos para poder instanciar objetos de esta clase a partir de los datos leídos del fichero CSV.

Código

```
public class MasMovilEDR
{
    public string Dealer { get; set; }
    public DateTime FechaExtraccion { get; set; }
    public DateTime Fecha { get; set; }
    public string Factura { get; set; }
    public string MSISDN { get; set; }
    public string Tipo_Destino { get; set; }
    public string Destino { get; set; }
    public decimal Minutos_Bytes { get; set; }
    public decimal Valor { get; set; }
    public decimal Impuestos { get; set; }

    public static MasMovilEDR Parse(string[] values)
    {
        return new MasMovilEDR()
        {
            Dealer = values[0],
            FechaExtraccion = DateTime.Parse(values[1], new CultureInfo("es")),
            Fecha = DateTime.Parse(values[2], new CultureInfo("es")),
            Factura = values[3],
            MSISDN = values[4],
            Tipo_Destino = values[5],
            Destino = values[6],
            Minutos_Bytes = decimal.Parse(values[7], new CultureInfo("es")),
            Valor = decimal.Parse(values[8], new CultureInfo("es")),
            Impuestos = decimal.Parse(values[9], new CultureInfo("es"))
        };
    }
    ...
}
```

Código 23: Clase que representa el modelo de datos de los registros de llamadas del proveedor MásMóvil



También se ha implementado una clase que lee ficheros CSV y los devuelve como una enumeración de filas, donde a su vez, cada fila es una matriz unidimensional de cadenas de texto, correspondiéndose cada índice de la matriz con la columna del fichero CSV que se encuentra en la misma posición que dicho índice.

A continuación, se muestra un fragmento de código de la implementación de la clase *CSVReader*. Como curiosidad, se puede observar la implementación de la interfaz *IEnumerable* y el uso del operador *yield*. Este operador permite implementar de una forma sencilla el patrón *Iterador*, permitiendo recorrer el fichero CSV sin necesidad de cargarlo completamente en memoria, sino que va leyendo solamente una línea del fichero y devolviendo está a quien enumere los elementos de la clase por ejemplo haciendo uso de un bucle *foreach* o usando el iterador de la clase *IEnumerable*.

Código

```
public class CSVReader : IEnumerable<string[]>, IDisposable
{
    ...
    public string[] Headers { get; private set; }

    public CSVReader(Stream stream,
                     char separator = ';',
                     Encoding encoding = null,
                     bool stringDelimiter = false,
                     bool firstLineAsHeader = false)
    {
        ...
        if (firstLineAsHeader)
        {
            _stream.Position = 0;
            using (var streamReader = new StreamReader(_stream, _encoding, true, 4096, true))
            {
                Headers = lineParse(streamReader);
            }
        }
        ...
    }

    public IEnumerator<string[]> GetEnumerator()
    {
        using (var streamReader = new StreamReader(_stream, _encoding, true, 1024, true))
        {
            _stream.Position = 0;

            if (_firstLineAsHeader)
                lineParse(streamReader);

            while (!streamReader.EndOfStream)
                yield return lineParse(streamReader);
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}
```

Código 24: Clase que enumera los registros de un fichero en formato CSV

5.2.4 Visualización del fichero CSV de MásMóvil

Aprovechando la clase *DataTableDefinition*, se definirá la acción *Details* en el controlador *MasMovilCSVController* donde se definirá la tabla a mostrar a partir del modelo de datos que representa los registros de llamadas de MásMóvil.

En la Figura 24 se muestra una captura de pantalla con una tabla donde se muestran los registros de un fichero CSV. Al usar el DataTableDefinition implementado para este proyecto, la tabla está paginada, permite filtrado y ordenación por columnas, y todo de forma asíncrona, descargando desde el servidor solamente los datos de la página a mostrar.

PFC.WebApp Home About Contact Register Log in

MasMóvil Fichero CSV

20170101.txt										Elemento por página:
Dealer	FechaExtraccion	Fecha	Factura	MSISDN	Tipo_Destino	Destino	Minutos_Bytes	Valor	Impuestos	
C03004-0000	2017-01-01T00:00:00	2017-01-23T09:04:16	20170101	633468741	VOZ NACIONAL	658262278	1908	0.08	0	
C03004-0000	2017-01-01T00:00:00	2017-01-04T15:59:36	20170101	633468741	VOZ NACIONAL	602458416	2032	0.08	0	
C03004-0000	2017-01-01T00:00:00	2017-01-21T22:50:25	20170101	633468741	VOZ NACIONAL	661358212	2418	0.08	0	
C03004-0000	2017-01-01T00:00:00	2017-01-13T12:31:57	20170101	633468741	VOZ NACIONAL	686456014	631	0.08	0	
C03004-0000	2017-01-01T00:00:00	2017-01-02T09:51:34	20170101	633468741	VOZ NACIONAL	613819228	458	0.08	0	
C03004-0000	2017-01-01T00:00:00	2017-01-03T02:55:27	20170101	633468741	VOZ NACIONAL	690728561	3130	0.08	0	
C03004-0000	2017-01-01T00:00:00	2017-01-27T02:08:40	20170101	633468741	VOZ NACIONAL	684701866	768	0.08	0	
C03004-0000	2017-01-01T00:00:00	2017-01-01T05:18:50	20170101	633468741	VOZ NACIONAL	632311726	2437	0.08	0	
C03004-0000	2017-01-01T00:00:00	2017-01-03T07:18:36	20170101	633468741	VOZ NACIONAL	663453356	2058	0.08	0	
C03004-0000	2017-01-01T00:00:00	2017-01-13T05:31:03	20170101	633468741	VOZ NACIONAL	696393033	715	0.08	0	

« ‹ 6 7 8 9 10 › »

[Back to List](#)

© 2018 - PFC.WebApp

Figura 24: Vista de un fichero CSV

6 IMPORTAR FICHERO CSV DE MÁSMÓVIL A LA BASE DE DATOS

6.1 ESTADO DE IMPORTACIÓN DE LOS FICHEROS

Para importar los ficheros de llamadas de MásMóvil a la base de datos se han definido los siguientes modelos de datos varios modelos que se detallan a continuación.

Para poder determinar los ficheros que han sido importados a la base de datos se ha definido el modelo que se muestra en la Figura 25.

Como se puede observar el modelo de datos contiene el nombre del fichero, un estado que indica el estado de importación en el que se encuentra, y almacenaremos el número de registros que contiene el fichero para evitar tener que calcularlo en el momento de visualizar el fichero.

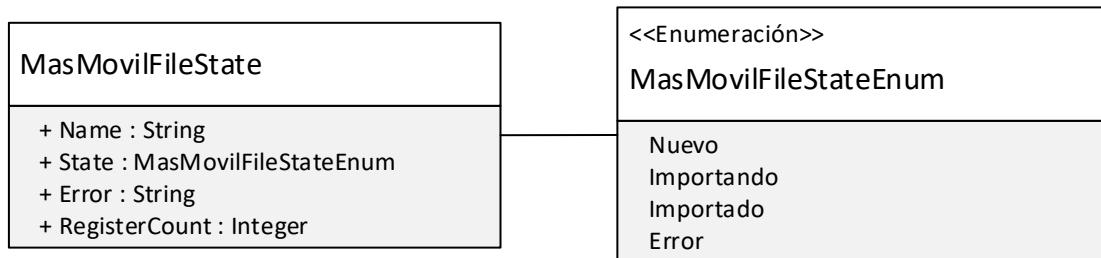


Figura 25: Diagrama de clases UML que representa el modelo de datos para la persistencia de los ficheros importados

El código resultante para este modelo de datos se puede ver en el fragmento de Código 25. En el código se observan algunos atributos que se han añadido al modelo como es el atributo **[Key]**, que lo que permite es indicar al ORM que se trata de la clave primaria para este modelo de datos.

También se observa el atributo **[NotMapped]** con el que se indica al ORM que la propiedad no va a tener persistencia, ya que no es más que una propiedad cuyo valor se calcula automáticamente y que permite obtener una representación en formato de cadena para el enumerado indicando el estado de importación.

Código	<pre> public enum MasMovilFileStateEnum { Nuevo, Importando, Importado, Error } public class MasMovilFileState { [Key] public virtual string Name { get; set; } public virtual MasMovilFileStateEnum State { get; set; } [NotMapped] public virtual MasMovilFileStateEnum StateString => return State.ToString(); public virtual string Error { get; set; } public virtual int RegistersCount { get; set; } } </pre>
--------	--

Código 25: Modelo de datos para la persistencia del estado de importación de los ficheros CDV de MásMóvil

Para indicar al ORM que hay un nuevo modelo que va a tener persistencia, se añade una propiedad en la clase que implementa `DbContext`, que nos dará acceso al contexto de persistencia de datos para nuestro nuevo modelo, tal y como se muestra en el fragmento de Código 26.

Código

```
/// <summary>
/// Contine la entrada a los repositorios de persistencia de datos de la aplicación
/// </summary>
public partial class ApplicationDbContext
{
    /// <summary>
    /// Acceso al contexto de datos de persistencia del estado de los ficheros CSV.
    /// </summary>
    public DbSet<MasMovilFileState> MasMovilFileStates { get; set; }
}
```

Código 26: Entrada en el `DbContext` del modelo de datos para el estado de importación de los ficheros CSV de MásMóvil

Por último y como ya se ha explicado en apartados anteriores, se ejecutan los comandos de EntityFramework para generar una nueva migración y actualizar la base de datos.

Una vez creada la persistencia se han modificado el controlador para contemplar este nuevo modelo de datos. Primero inyectamos la dependencia del contexto de persistencia de datos a través del constructor de la clase del controlador como vemos en el fragmento de Código 27, ya que se ha explicado anteriormente, el Framework de ASP.NET Core resuelve las dependencias de los controladores a través de sus constructores inyectando los servicios que hayamos configurado en el método `ConfigureServices` de la clase `Startup`.

Código

```
private readonly ApplicationDbContext _applicationDbContext;
public MasMovilCSVController(..., ApplicationDbContext applicationDbContext)
{
    ...
    _applicationDbContext = applicationDbContext;
}
```

Código 27: Inyección de dependencia del contexto de datos a través del constructor

El método que crea el fichero quedará entonces como se muestra en el fragmento de Código 28, detectando que no exista ya un fichero subido con el mismo nombre y estableciendo el estado como nuevo una vez subido al gestor documental.

Código

```
public async Task<IActionResult> Create(IFormFile File)
{
    ...
    if(_applicationDbContext.MasMovilFileStates.Any(x => x.Name == File.FileName))
    {
        ModelState.AddModelError("", "$Ya existe un fichero con el mismo nombre");
        return View();
    }
    ...
    _applicationDbContext.MasMovilFileStates.Add(new MasMovilFileState()
    {
        Name = File.FileName,
        RegistersCount = line,
        State = MasMovilFileStateEnum.Nuevo
    });
    await _applicationDbContext.SaveChangesAsync();
    ...
}
```

Código 28: Modificación del método para subir ficheros CSV de MásMóvil para establecer el estado de importación



También, el método de borrado evitar se modificará para que borre el estado de importación una vez borrado el fichero y evite borrar ficheros que ya han sido importados a la base de datos tal y como se muestra en el fragmento de Código 29.

Código

```
public ActionResult Delete(string id, IFormCollection collection)
{
    var fileState = _applicationDbContext.MasMovilFileStates
        .SingleOrDefault(x => x.Name == id);

    if (fileState == null)
        return NotFound();

    if (fileState.State != MasMovilFileStateEnum.Nuevo &&
        fileState.State != MasMovilFileStateEnum.Error)
    {
        ModelState.AddModelError("", "El fichero no se puede borrar porque ha sido importado a
la base de datos");
        return View();
    }

    documentRepository.DeleteDocument(id);
    _applicationDbContext.MasMovilFileStates.Remove(fileState);
    _applicationDbContext.SaveChanges();

    ...
}
```

Código 29: Modificación del método de eliminar para contemplar el estado de importación

6.2 IMPORTACIÓN DE LOS EVENTOS TELEFÓNICOS DESDE EL FICHERO CSV DE MÁSMÓVIL

Por último, definiremos un modelo de datos para la persistencia de los eventos telefónicos.

Este modelo de datos debe ser genérico, y almacenará la información de interés para aplicación independientemente del proveedor de telefonía desde el que provenga, así, existirá un único modelo de datos para los eventos telefónicos que será el modelo de datos del que dependerá el resto de la aplicación.

No obstante, se añadirá información adicional al modelo de datos del evento para determinar el proveedor al que pertenece un evento telefónico concreto, así como un localizador que podamos usar para que el proveedor identifique tal evento.

De este modo, el modelo de datos que se ha definido es tal como se muestra en el diagrama de clases de la Figura 26.

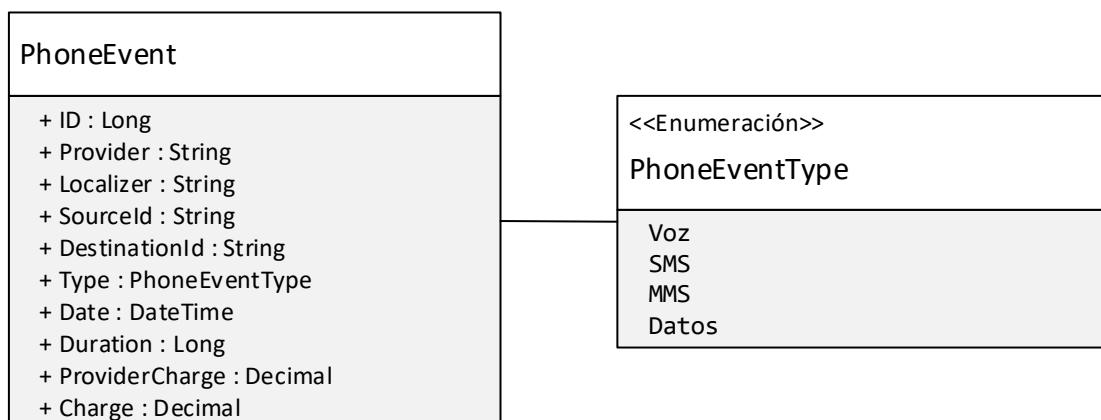


Figura 26: Diagrama de clases UML con el modelo de datos que representa un evento telefónico

A continuación, se muestra un fragmento de código con la implementación del modelo de datos para dar persistencia a los eventos telefónicos.

Un detalle que se puede observar es que no se ha establecido el atributo *[Key]* como sí se hizo para el modelo de datos anterior. Esto no es necesario ya que *EntityFramework* escogerá por defecto la clave primaria si existe una propiedad que se llame *ID* en el modelo de datos definido.

Código

```
public class PhoneEvent
{
    public virtual long ID { get; set; }

    public virtual string Localizer { get; set; }

    public virtual string Provider { get; set; }

    public virtual string SourceId { get; set; }

    public virtual string DestinationId { get; set; }

    public virtual PhoneEventType Type { get; set; }

    public virtual string TypeString => Type.ToString();

    public virtual DateTime Date { get; set; }

    public virtual long Duration { get; set; }

    public decimal ProviderCharge { get; set; }

    public decimal Charge { get; internal set; }
}
```

Código 30: Modelo de datos para la persistencia de los eventos telefónicos

Al igual que en se explicó en el apartado anterior, se añadirá una entrada en la clase *DbContext* para el nuevo modelo de datos y se ejecutarán los comandos de consola para generar una nueva migración y actualizar la base de datos.

Una vez hecho esto, se añadirá un método en el controlador *MasMovilCSVController* para importar un fichero a base de datos tal y como se muestra en el fragmento de Código 31.

Código

```
ConcurrentDictionary<string, Task> importaciones = new ConcurrentDictionary<string, Task>();

public async Task<IActionResult> Import(string id)
{
    var fileState = await _applicationDbContext.MasMovilFileStates
        .SingleOrDefaultAsync(x => x.Name == id);

    if (fileState.State != MasMovilFileStateEnum.Nuevo)
    {
        ModelState.AddModelError("", "El fichero ya ha sido importado");
        return RedirectToAction(nameof(Index));
    }

    var task = GetTask(id);

    if (importaciones.TryAdd(id, task))
    {
        fileState.State = MasMovilFileStateEnum.Importando;
        _applicationDbContext.SaveChanges();

        task.Start();
    }

    return RedirectToAction(nameof(Index));
}
```

Código 31: Método de importación de ficheros CSV de MásMóvil a la base de datos



La importación se realizará mediante la ejecución de una tarea asíncrona tal y como se observa en el fragmento de Código 32 gestionando los posibles errores que se puedan producir durante el proceso y estableciendo el estado de importación del fichero según haya ido el proceso.

Para ello se ha hecho uso de un contenedor concurrente ya que varios usuarios podrían acceder a importar un fichero al mismo tiempo, y de esta forma, el acceso concurrente queda gestionado internamente por el propio contenedor.

Código

```

private Task GetTask(string id)
{
    return new Task(() =>
    {
        try
        {
            using (var events =
                new CSVReader(documentRepository.GetDocument(id).GetReadStream(), '|'))
            {
                IEnumerable<PhoneEvent> entities = events
                    .Select(x => MasMovilEDR.Parse(x))
                    .Select((x, i) => new PhoneEvent()
                {
                    Provider = "MasMovilProvider",
                    Localizer = $"{id}:{i}",
                    Date = x.Fecha,
                    SourceId = x.MSISDN,
                    DestinationId = x.Destino,
                    Duration = x.Minutos_Bytes,
                    Type = PhoneEventType.Voz,
                    ProviderCharge = x.Valor,
                    Charge = x.Valor
                });
                using (IServiceScope serviceScope = _serviceProvider.CreateScope())
                {
                    using (var context =
                        serviceScope.ServiceProvider.GetService<ApplicationDbContext>())
                    {
                        context.PhoneEvents.AddRange(entities);
                        var fileState = context.MasMovilFileStates
                            .SingleOrDefault(x => x.Name == id);
                        fileState.State = MasMovilFileStateEnum.Importado;
                        context.SaveChanges();
                    }
                }
            }
        catch (Exception ex)
        {
            using (IServiceScope serviceScope = _serviceProvider.CreateScope())
            {
                using (var context =
                    serviceScope.ServiceProvider.GetService<ApplicationDbContext>())
                {
                    var fileState = context.MasMovilFileStates
                        .SingleOrDefault(x => x.Name == id);
                    fileState.State = MasMovilFileStateEnum.Error;
                    fileState.Error = ex.Message;
                    context.SaveChanges();
                }
            }
        }
        finally
        {
            importaciones.TryRemove(id, out _);
        }
    });
}

```

Código 32: Tarea asíncrona de importación de eventos telefónicos desde un fichero CSV de MásMóvil



Un detalle a tener en cuenta durante la importación de llamadas para este proveedor es que el importe está establecido en el fichero de llamadas, por lo tanto, asumiremos dicho importe como el coste del evento.

En caso de implementar otros proveedores en los que el importe de eventos no viniese establecido, habría que implementar un proceso de tasación que permita calcular los importes de cada evento durante la fase de importación.

Para poder importar un fichero, se añadirá una nueva acción a la configuración del objeto DataTableDefinition que define la tabla de visualización de ficheros CSV, que contenga un enlace al nuevo método de importación creado en el controlador, así como una nueva columna que muestre el estado de importación, quedando como se muestra en Código 33.

```
Código
var dataTableDefinition = new DataTableDefinition(_applicationDbContext.MasMovilFileStates)

...
    .MapColumn("StateString", "Estado", 20)

...
    .AddElementAction("Importar", new DataTableDefinition.UrlActionDTO()
{
    Action = "Import",
    Values = new Dictionary<string, string> { { "id", "Name" } }
})

...
Código 33: Agregación del enlace de importación a la definición de la tabla de visualización de ficheros CSV de MásMóvil
```

Accediendo a la Url <http://localhost:5000/MasMovilCSV> se podrá ver la nueva configuración de la tabla, así como el enlace para importar disponible en el menú de acciones sobre cada fichero.

MásMóvil CSV Ficheros

© 2018 - PFC.WebApp

Una vez y los ficheros vayan siendo importados veremos cómo su estado cambiará indicando que ya ha sido importado en la tabla de ficheros CSV.

En este apartado se podría mejorar la experiencia del usuario mostrando una barra de progreso para aquellos ficheros que estén en proceso de importación y que se actualizase asíncronamente sin necesidad de que el usuario tenga que refrescar la página para ello.

6.3 VISUALIZACIÓN DE EVENTOS TELEFÓNICOS IMPORTADOS EN BASE DE DATOS

Para la visualización de los eventos en la base de datos se ha definido un nuevo controlador, *PhoneEventsController* que haciendo uso de la clase *DataTableDefinition* permitirá visualizar estos eventos de una forma similar a la que visualizamos los registros del fichero CSV.

Una vez más, y como ya hemos explicado antes, se definirá un constructor que reciba la dependencia con el contexto de datos de *EntityFramework*, y se definirá el método *Index* tal y como se puede ver en el fragmento de Código 34 además de un método para visualizar un registro concreto en más detalle y otro que a través del identificador del proveedor enrutará al usuario hacia el origen del evento telefónico, en este caso, al fichero de origen del proveedor de MásMóvil.

Código

```

public PhoneEventsController(ApplicationDbContext context)
{
    _context = context;
}

public ActionResult Index()
{
    var dataTableDefinition = new DataTableDefinition(_context.PhoneEvents)
        .WithTitle("Eventos telefónicos")
        .WithListAction(Url.Action("Index"))
        .MapColumn("Date", "Fecha", 20, "complex")
        .MapColumn("SourceId", "Origen", 20)
        .MapColumn("DestinationId", "Destino", 20)
        .MapColumn("Duration", "Duración", 20)
        .MapColumn("TypeString", "Tipo", 20)
        .AddElementAction("Detalles", new DataTableDefinition.UrlActionDTO()
    {
        Action = "Details",
        Values = new Dictionary<string, string> { { "id", "ID" } }
    }).AddElementAction("Origen", new DataTableDefinition.UrlActionDTO()
    {
        Action = "Origin",
        Values = new Dictionary<string, string> { { "id", "ID" } }
    });
    return View(dataTableDefinition);
}

[HttpPost]
public JsonResult Index(int index = 0, int length = 10, string orderProperty = null, bool
inverse = false, [FromBody]IEnumerable<QueryFilter> query = null)
{
    DataTableDefinition.DataQuery data =
        DataTableDefinition.DataQueryBuilder(index, length, orderProperty, inverse, query,
_context.PhoneEvents);
    return base.Json(data);
}

public async Task<IActionResult> Details(long id)
{
    var phoneEvent = await _context.PhoneEvents
        .SingleOrDefaultAsync(m => m.ID == id);

    if (phoneEvent == null)
        return NotFound();
    return View(phoneEvent);
}

public async Task<IActionResult> Origin(long id)
{
    var phoneEvent = await _context.PhoneEvents
        .SingleOrDefaultAsync(m => m.ID == id);

    if (phoneEvent == null)
        return NotFound();
}

```



```

switch (phoneEvent.Provider)
{
    case "MasMovilProvider":
        return RedirectToAction(nameof(MasMovilCSVController.Details),
            nameof(MasMovilCSVController).Replace("Controller", ""),
            new { id = phoneEvent.Localizer.Split(":")[0] });
    default:
        return NotFound();
}

```

Código 34: Controlador de visualización de eventos telefónicos

De esta forma, si accedemos a la Url <http://localhost:5000/PhonEvents> se podrá visualizar la tabla con todos los eventos telefónicos que son obtenidos directamente desde la base de datos. En el ejemplo de la Figura 27 vemos como se ha filtrado por un número de origen y un rango de fechas.

Origen	Destino	Tipo	Fecha	Duración	Coste
657530972	656679169	Voz	2017-01-03T01:30:37	3287	0.08
657530972	643493180	Voz	2017-01-03T02:43:53	2027	0.08
657530972	682392800	Voz	2017-01-03T02:53:54	455	0.08
657530972	652010810	Voz	2017-01-03T06:10:30	1658	0.08
657530972	638722244	Voz	2017-01-03T08:40:33	1584	0.08
657530972	660704805	Voz	2017-01-03T13:35:09	2305	0.08
657530972	629467956	Voz	2017-01-03T22:25:45	2414	0.08
657530972	634154971	Voz	2017-01-03T22:33:33	3187	0.08
657530972	621849726	Voz	2017-01-04T03:50:56	1768	0.08
657530972	694582884	Voz	2017-01-04T05:20:13	2184	0.08

© 2018 - PFC.WebApp

Figura 27: Captura de pantalla de la visualización de eventos telefónicos desde la base de datos

Haciendo clic en la opción detalles del menú de acciones de cada evento telefónico se podrá acceder a una vista detallada del evento telefónico.

PFC.WebApp Home About Contact Register Log in

← Eventos telefónicos Detalles

Evento

Localizer	20170101.txt:28387
Provider	MasMovilProvider
Sourceld	657530972
DestinationId	656679169
Type	Voz
Date	03/01/2017 1:30:37
Duration	3287
ProviderCharge	0,08
Charge	0,08

© 2018 - PFC.WebApp

Figura 28: Vista detalle de un evento telefónico

7 TARIFAS

En este apartado se definirán las tarifas que se aplicarán a las líneas telefónicas.

Para ello se ha definido un modelo de datos que representará la información de una tarifa de telefonía, como son el coste y los diferentes bonos aplicados a ella.

7.1 DEFINICIÓN DEL MODELO DE DATOS

Para simplificar, se ha optado por definir tres tipos de bonos aplicables para cada tarifa, y que aplicarán descuentos en la facturación de las líneas asociadas a dicha tarifa. Los bonos definidos son los siguiente:

- Bono de voz: minutos en llamadas sin coste
- Bono de datos: Tráfico de datos sin coste de navegación
- Bono de SMS: Mensajes de texto sin coste

A continuación, se muestra un fragmento de código con la definición del modelo de datos que se usará para dar persistencia en la base de datos a las tarifas.

Código

```
/// <summary>
/// Modelo de datos que represente una tarifa móvil
/// </summary>
public class Plan
{
    /// <summary>
    /// Identificador.
    /// </summary>
    public virtual int ID { get; set; }

    /// <summary>
    /// Nombre del plan.
    /// </summary>
    public virtual string Name { get; set; }

    /// <summary>
    /// Especifica el coste del bono.
    /// </summary>
    public virtual decimal Charge { get; set; }

    /// <summary>
    /// Minutos en llamadas sin coste incluidos en línea.
    /// </summary>
    public virtual int VoicePlan { get; set; }

    /// <summary>
    /// Tráfico de datos sin coste de navegación incluidos en línea.
    /// </summary>
    public virtual int DataPlan { get; set; }

    /// <summary>
    /// SMS sin coste incluidos en línea.
    /// </summary>
    public virtual int SMSPlan { get; set; }
}
```

Código 35: Modelo de datos para la persistencia de las tarifas

Una vez más, se agrega una entrada en la clase DbContext para dar acceso al repositorio de datos para la nueva entidad y, ejecutamos las órdenes de creación y actualización de migraciones de la base de datos.

7.2 MVC DE TARIFAS

Se define un controlador con acciones de CRUD para las tarifas similar a como se han definido otros controladores en la aplicación, obteniendo como resultado una pantalla donde poder visualizar las tarifas configuradas en la aplicación y desde la que se podrá realizar las diferentes acciones sobre éstas.

En la Figura 29 se puede observar esta nueva pantalla donde se visualizan las tarifas existentes, así como las posibles acciones a realizar sobre cada una de ellas. También se observan cuatro tarifas sencillas que se han definido para la realización de este proyecto con diferentes bonos de descuento para las llamadas de voz, consumo de datos y envío de mensajes de texto.

Plan	Coste	Bono de voz	Bono de datos	Bono de sms
100MB - 0min	7.5	0	100	0
1GB - 100min	12	100	1000	50
4GB - 200min	15	200	4000	150
10GB - 500min	25	500	10000	600

© 2018 - PFC.WebApp

Figura 29: Pantalla de visualización de tarifas

A continuación, en la Figura 30 se observa la pantalla de creación de una nueva tarifa y los diferentes campos que componen este modelo.

PFC.WebApp Home About Contact Register Log in

Tarifas Crear

Name	<input type="text"/>
*VoicePlan	<input type="text"/>
*DataPlan	<input type="text"/>
*SMSPlan	<input type="text"/>
*Charge	<input type="text"/>

[Guardar](#) [Volver](#)

© 2018 - PFC.WebApp

Figura 30: Pantalla de creación de tarifa nueva

En la Figura 31, una pantalla que muestra una vista detallada para una tarifa de telefonía ya existente que incluye un botón para navegar hacia la pantalla de edición.

PFC.WebApp Home About Contact Register Log in

Tarifas Detalles

Name	100MB - 0min
Charge	7,50
VoicePlan	0
DataPlan	100
SMSPlan	0

[Editar](#) [Volver](#)

© 2018 - PFC.WebApp

Figura 31: Pantalla con la vista detallada de una tarifa

Una pantalla para poder editar una tarifa ya existente como muestra la Figura 32.

PFC.WebApp Home About Contact Register Log in

Tarifas Editar

Name

*VoicePlan

*DataPlan

*SMSPlan

*Charge

[Guardar](#) [Volver](#)

© 2018 - PFC.WebApp

Figura 32: Pantalla con la vista de edición de una tarifa

Y por último, como se observa en la Figura 33, una pantalla con la confirmación de borrado de una tarifa ya existente.

PFC.WebApp Home About Contact Register Log in

Tarifas Borrar

¿Está seguro de querer borrar esta tarifa?

Name	100MB - 0min
Charge	7,50
VoicePlan	0
DataPlan	100
SMSPlan	0

[Borrar](#) [Volver](#)

© 2018 - PFC.WebApp

Figura 33: Pantalla de borrado de una tarifa

8 ABONADO DE TELEFONÍA

En telefonía existe el concepto de abonado, que no es más que una línea de teléfono con una tarifa vigente sobre la cual posteriormente se calculará la facturación.

Por tanto, para este apartado se definirá un modelo de datos que represente este concepto, en el que se establecerá el número de teléfono sobre el cual se ha creado el abonado, así como la tarifa que tiene vigente.

8.1 DEFINICIÓN DEL MODELO DE DATOS

Para determinar la vigencia de dicha tarifa, se definirán un modelo de datos tal y como se muestra en el diagrama de clases UML de la Figura 34. Se han definido dos fechas, fecha de alta y fecha de baja del abonado, una relación con la tarifa asociada.

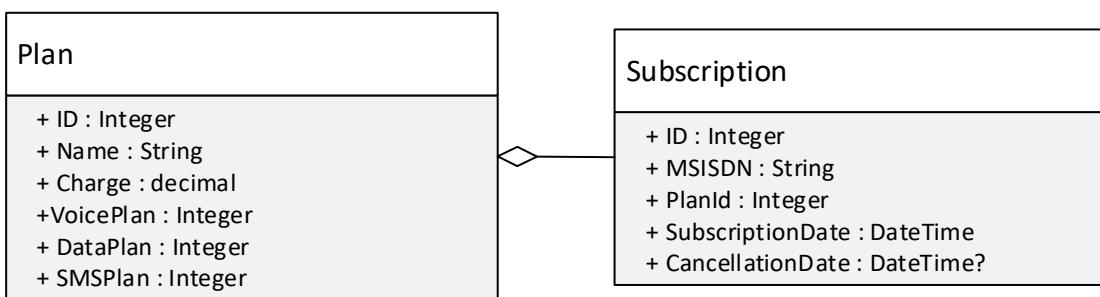


Figura 34: Diagrama de clases UML que muestra la relación de agregación entre una suscripción y la tarifa asociada

En el fragmento de Código 35 se muestra la implementación del modelo de datos para la entidad de abonado de telefonía que se ha definido.

Código	<pre> /// <summary> /// Entidad que representa el modelo de datos de un abono telefónico /// </summary> public class Subscription { /// <summary> /// Identificador. /// </summary> public virtual int ID { get; set; } /// <summary> /// Número de teléfono (Mobile Station Integrated Services Digital Network.) /// </summary> [Required, MaxLength(15), Phone] public virtual string MSISDN { get; set; } /// <summary> /// Identificador de la tarifa asociada al abonado. /// </summary> public virtual int PlanId { get; set; } /// <summary> /// Propiedad de navegación hacia la tarifa asociada al abonado. /// </summary> [ForeignKey(nameof(PlanId))] public virtual Plan Plan { get; set; } /// <summary> /// Obtiene o establece la fecha de alta del abonado. /// </summary> [DataType(DataType.Date)] public virtual DateTime SubscriptionDate { get; set; } </pre>
--------	---



```

/// <summary>
/// Obtiene o establece la fecha de baja.
/// </summary>
[DataType(DataType.Date)]
public virtual DateTime? CancellationDate { get; set; }
}

```

Figura 35: Modelo de datos para la persistencia de los abonados de telefonía

En este fragmento de código, se pueden apreciar nuevos atributos de decoración que son utilizados por el Framework MVC para realizar validaciones además de ser utilizados por el ORM EntityFramework para definir las columnas de la tabla en la base de datos:

- Required: Indica que el campo es requerido, provocando errores de validación cuando está vacío. EntityFramework lo utiliza para definir la columna como no nula en la base de datos.
- MaxLength: Valida la longitud máxima del campo de texto. EntityFramework establece la longitud para la columna en la base de datos.
- Phone: Valida que tenga formato de número de teléfono.
- ForeignKey: Define una clave ajena que relaciona esta entidad con la tarifa en base de datos (usado por EntityFramework).
- DataType: Define el formato con el que se mostrarán las fechas en los formularios.

En el siguiente cuadro se muestra el código generado por EntityFramework donde se puede observar cómo afectan dichos atributos a la generación de las tablas en la base de datos.

Código

```

protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "Subscriptions",
        columns: table => new
        {
            ID = table.Column<int>(nullable: false)
                .Annotation("SqlServer:ValueGenerationStrategy", "IdentityColumn"),
            CancellationDate = table.Column<DateTime>(nullable: true),
            MSISDN = table.Column<string>(maxLength: 15, nullable: false),
            PlanId = table.Column<int>(nullable: false),
            SubscriptionDate = table.Column<DateTime>(nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Subscriptions", x => x.ID);
            table.ForeignKey(
                name: "FK_Subscriptions_Plan_PlanId",
                column: x => x.PlanId,
                principalTable: "Plan",
                principalColumn: "ID",
                onDelete: ReferentialAction.NoAction);
        });
}

migrationBuilder.CreateIndex(
    name: "IX_Subscriptions_PlanId",
    table: "Subscriptions",
    column: "PlanId");
}

protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Subscriptions");
}

```

Figura 36: Código generado por el sistema de migraciones de Entity Framework para el modelo de datos del Abonado

En este punto hay que prestar bastante atención ya que este será el modelo de datos que va a configurar la tabla en la base de datos. En este caso se ha editado para cambiar el comportamiento de eliminación en cascada de la base de datos, lo que provocará que se dispare una excepción desde EntityFramework al intentar eliminar una tarifa que está asociada con algún abonado.

Por tanto, se modificará la acción de borrado en el controlador de tarifas para capturar el posible fallo y mostrar las notificaciones al usuario, tal y como se observa en la Figura 37.

The screenshot shows a web application interface. At the top, there is a dark header bar with the text "PFC.WebApp" and navigation links for "Home", "About", and "Contact" on the left, and "Register" and "Log in" on the right. Below the header, the main content area has a title "Tarifas Borrar" with a back arrow icon. A red error box is displayed, containing the heading "Errores" and a single bullet point: "• No se ha podido borrar el plan, compruebe que no esté asignado a ningún abonado". Below the error box, a confirmation message asks "¿Está seguro de querer borrar esta tarifa?". Underneath this message, a table lists the tariff's details: Name (4GB - 200min), Charge (15,00), VoicePlan (200), DataPlan (4000), and SMSPlan (150). At the bottom of the form are two buttons: a blue "Borrar" button and a white "Volver" button.

© 2018 - PFC.WebApp

Figura 37: Mensaje de validación en la pantalla de borrado de una tarifa que está siendo usada por algún abonado

8.2 MVC DE ABONADO DE TELEFONÍA

Como en apartados anteriores, generamos el controlador y las vistas para el modelo de Abonados obteniendo una vista de índice tal y como se muestra en la Figura 38.

Número de teléfono	Tarifa	Fecha de alta	Fecha de baja
612345789	100MB - 0 min	2018-05-01T00:00:00	2018-05-01T00:00:00
612345789	4GB - 200min	2018-05-02T00:00:00	

© 2018 - PFC.WebApp

Figura 38: Pantalla de visualización de abonados

Hay que tener en cuenta, que cuando un cliente quiera modificar una tarifa de datos habrá de establecerse la fecha de finalización del abonado actual y crear un nuevo abonado sobre la misma línea de teléfono con una fecha de alta posterior a la fecha de baja del último abonado existente para dicho número de teléfono.

Por lo tanto, habrá que validar esta restricción a la hora de crear un nuevo abonado, así como bloquear la edición y eliminación de los abonados que ya estén dados de baja y para los que ya exista un abonado posterior con el mismo número de teléfono.

A continuación, en la Figura 39 se observa la pantalla de edición bloqueada por existir un abonado con fecha de alta posterior al abonado que se intenta editar.

PFC.WebApp Home About Contact Register Log in

Abonados Editar

El formulario ha sido inhabilitado por la siguiente razón: Existe algún abonado con fecha de alta posterior al actual

***MSISDN**

612345789

***PlanId**

6

***SubscriptionDate**

01/05/2018 0:00:00

CancellationDate

01/05/2018 0:00:00

[Guardar](#) [Volver](#)

© 2018 - PFC.WebApp

Figura 39: Pantalla de edición bloqueada por existencia de un abonado con fecha de alta posterior.

Y en la Figura 40 se puede ver el mismo bloqueo esta vez para la pantalla de borrado.

PFC.WebApp Home About Contact Register Log in

Abonados Borrar

El formulario ha sido inhabilitado por la siguiente razón: Existe algún abonado con fecha de alta posterior al actual

¿Está seguro de querer
borrar este abonado?

MSISDN	612345789
SubscriptionDate	01/05/2018
CancellationDate	01/05/2018
Plan	100MB - 0 min

[Borrar](#) [Volver](#)

© 2018 - PFC.WebApp

Figura 40: Pantalla de borrado de abonado bloqueada por existencia de un abonado con fecha de alta posterior

Al crear o editar un abonado, además de las validaciones que se validarán automáticamente por el Framework MVC según los atributos explicados al comienzo de este apartado, se realizarán de forma manual las validaciones siguientes:

- Que las fechas de alta y baja no estén cruzadas
- Que la fecha de alta sea mayor que la fecha de baja de abonados anteriores para la misma línea
- Que exista algún abonado anterior que no tenga establecida la fecha de baja

A continuación, en la Figura 41, se muestra la pantalla de edición de un abonado mostrando alguno de los errores de validación mencionados anteriormente.

PFC.WebApp Home About Contact Register Log in

Abonados Editar

The MSISDN field is not a valid phone number.
La fecha de baja no puede ser menor a la fecha de alta

***MSISDN**
ABCDEFGHI
The MSISDN field is not a valid phone number.

***PlanId**
10GB - 500min

***SubscriptionDate**
01/05/2018

CancellationDate
30/04/2018
La fecha de baja no puede ser menor a la fecha de alta

Guardar **Volver**

© 2018 - PFC.WebApp

Figura 41: Pantalla de edición de abonado mostrando errores de validación

8.3 INFORME DE EVENTOS TELEFÓNICO SIN ABONADO VIGENTE

Una vez llegados a este punto, y con los datos de los que ya se dispone, es posible generar una isla de datos mediante una consulta a la base de datos que nos pueda dar un informe detallado de los eventos telefónicos importados para los cuales no existe un abonado con suscripción vigente a una tarifa de datos.

Para ello se definirá un controlador de generación de informes que podrá ser utilizado para la generación de otros informes que se puedan sacar a partir de la información disponible y que sean de utilidad para usuarios de esta plataforma.

Este controlador se conformará de una vista de índice donde se mostrará un listado de enlaces con los tipos de informe disponibles, como se puede observar en la Figura 42.

The screenshot shows a dark-themed web application header with navigation links: 'PFC.WebApp', 'Home', 'About', 'Contact', 'Register', and 'Log in'. Below the header, the main content area has a title 'Informes' with a back arrow icon, followed by 'Listado'. A single item in a list is shown: 'Eventos telefónicos sin abonado vigente'. At the bottom of the page, there is a copyright notice: '© 2018 - PFC.WebApp'.

Figura 42: Vista de índice de los informes disponibles

Y como resultado del informe, se visualizará una tabla en la que se mostrará una fila por cada número de teléfono detectado que no disponga de abonado vigente, indicando el número de eventos detectados, así como la fecha del primer evento y el último sobre dicha línea.

Esto dará una información valiosa al usuario que realiza las tareas de gestión dentro de la aplicación que le permitirá detectar y dar de alta los abonados faltantes.

En la Figura 43: Informe de eventos telefónicos sin abonado vigente podemos ver el resultado de un informe a partir de los datos de prueba que se han ido cargando en apartados anteriores.

PFC.WebApp Home About Contact Register Log in

Informes Eventos telefónicos sin abonado vigente

MSISDN	EventsCount	First	Last
600363867	4	28/02/2017 1:57:48	28/02/2017 18:00:43
603452136	82	01/02/2017 6:05:40	28/02/2017 20:22:48
603905860	99	01/02/2017 2:19:58	28/02/2017 10:55:43
603957878	85	01/01/2017 2:52:56	28/01/2017 0:23:39
606096393	93	01/01/2017 4:41:42	28/01/2017 20:48:12
607133782	97	01/01/2017 0:22:38	28/01/2017 20:52:21
608381706	109	01/01/2017 18:36:44	28/01/2017 20:21:12
610139474	102	01/01/2017 11:06:36	28/01/2017 22:32:18
611399987	91	01/02/2017 8:52:20	28/02/2017 9:52:51
615843286	94	01/01/2017 2:57:41	28/01/2017 18:05:06
617768281	91	01/01/2017 0:46:54	28/01/2017 18:56:13
617878785	91	01/02/2017 0:24:21	28/02/2017 12:54:48
618042529	111	01/01/2017 23:45:40	28/01/2017 22:39:30
618052477	115	01/02/2017 5:36:39	28/02/2017 23:51:29
618814549	102	01/01/2017 1:46:38	28/01/2017 21:55:29

Figura 43: Informe de eventos telefónicos sin abonado vigente

Y en la tabla de Código 36 podemos ver el fragmento de código con mayor relevancia para la generación de este informe.

Código

```
public async Task<IActionResult> PhoneEventsWithoutSubscription(int id)
{
    var result = await _context.PhoneEvents
        .GroupJoin(_context.Subscriptions,
            pe => pe.SourceId,
            s => s.MSISDN,
            (pe, s) => new
            {
                PhoneEvent = pe,
                Subscription = s.Where(x => pe.Date >= x.SubscriptionDate &&
(!x.CancellationDate.HasValue || pe.Date < x.CancellationDate.Value.AddDays(1)))
            })
        .Where(x => !x.Subscription.Any())
        .Select(x => x.PhoneEvent)
        .GroupBy(x => x.SourceId)
        .Select(x => new PhoneEventsWithoutSubscriptionDTO
        {
            MSISDN = x.Key,
            EventsCount = x.Count(),
            First = x.Select(pe => pe.Date).Min(),
            Last = x.Select(pe => pe.Date).Max()
        })
        .Distinct()
        .ToListAsync();

    return View(result);
}
```

Código 36: Fragmento de código con la generación de los datos del informe de eventos telefónicos sin abonado vigente

9 CÁLCULO DE FACTURAS

Uno de los principales objetivos de este proyecto ha sido llegar a disponer de toda la información necesaria en la aplicación para poder generar la factura mensual de consumo de cada uno de los abonados que tenemos en la aplicación y que disponen de una tarifa vigente.

9.1 DEFINICIÓN DEL MODELO DE DATOS

Para ello se ha definido un modelo de datos para persistir la información de cada factura como son: fechas que definen el período de facturación, coste de la tarifa asociada, precio y consumo para cada tipo de evento telefónico, subtotal, impuestos aplicados y coste total de la factura.

Este modelo de datos contendrá el identificador del abonado para el cual ha sido generada dicha factura y, además, durante el período de facturación se establecerá a cada evento telefónico que se haya producido en el período de facturación, el identificador de la factura, para así poder obtener posteriormente el listado de llamadas de la factura.

A continuación, en la Figura 44, se muestra un diagrama de clases UML que representa la factura según lo comentado en los párrafos anteriores.

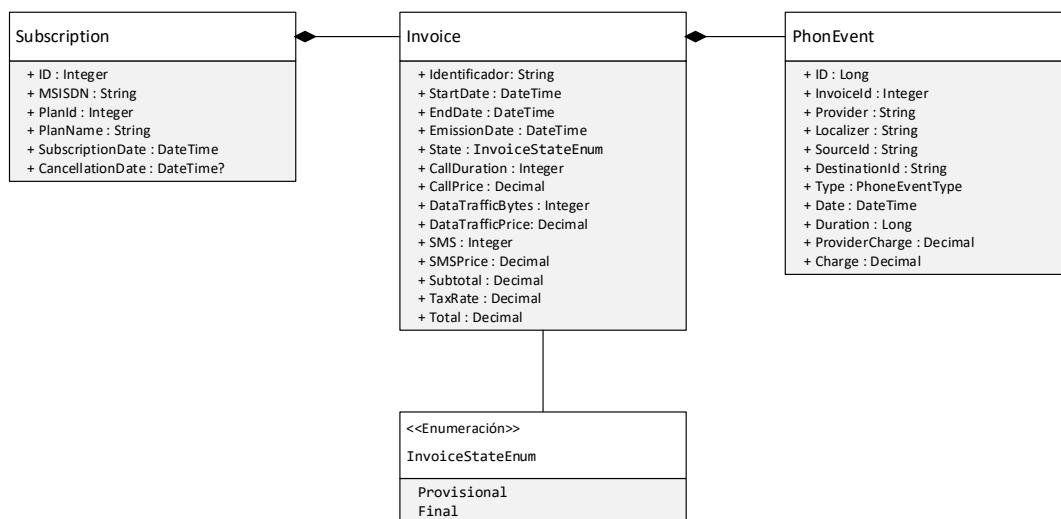


Figura 44: Diagrama de clases UML que representa una factura telefónica

Algo a destacar en la definición del modelo de datos es que no se ha establecido una relación de clave ajena entre el modelo de datos de factura y el modelo de datos de evento telefónico, ya que esto obligaría a mantener un índice en base de datos sobre la tabla de eventos telefónicos la cual contendrá una cantidad muy grande de registros y tener un índice en una tabla de estas características penalizaría la inserción de registros en dicha tabla.

Otro detalle en el diseño del modelo de datos de facturas es que se ha definido una propiedad que determina el estado en el que se encuentra dicha factura que, en principio, podrá tener un valor de entre los posibles definidos por la enumeración **InvoiceStateEnum**: Provisional o Final.

De esta forma, al generar las facturas estas se generarán en un estado provisional para permitir a los usuarios con el rol de gestor poder revisar dichas facturas antes de emitirlas definitivamente para que el cliente pueda acceder a ellas.

9.2 MVC PARA FACTURACIÓN

9.2.1 Modificación del controlador de abonados para generar facturas por abonado

Para simplificar la implementación de este objetivo, se añadirá a cada abonado de telefonía una acción que permita generar una factura estableciendo el período para el cual se desea generar, período que será habitualmente un mes completo, excepto en aquellos casos en los que el abonado se haya dado de alta o baja en medio del ciclo de facturación, en cuyo caso el ciclo estará delimitado teniendo en cuenta las fechas de alta y/o de baja del abonado.

Así, tendremos una tabla dinámica con filtros de búsqueda y ordenación disponible en la vista detalle de cada abonado, desde la que podremos visualizar las facturas emitidas. Esta tabla mostrará el identificador de la factura, así como información del período de facturación coste y estado de la factura.

En la siguiente imagen, Figura 45, se muestra una captura de la pantalla de la vista de detalle de un abonado en la que se ha añadido la tabla de facturas para dicho abonado.

PFC.WebApp Home About Contact Register Log in

Abonados Detalles

MSISDN	600363867
SubscriptionDate	01/02/2017
CancellationDate	27/02/2017
Plan	100MB - 0min

Facturas

Factura	Mes	Año	Estado	Total
F0000000013	febrero	2017	Provisional	17.44

Nuevo

Elemento por página: 10

© 2018 - PFC.WebApp

Figura 45: Pantalla de detalle de un abonado de telefonía con sus facturas

Además, se han añadido acciones al controlador de abonados para generar las facturas. Este controlador muestra una pantalla donde se seleccionaría el período para el cual se desea generar la factura, controlando que no permita generar facturas para períodos ya existentes o que no haya facturas pendientes de emisión para ningún período.

En la Figura 46, se observa una captura de la pantalla de generación de facturas para un abonado. En dicha pantalla se dispone de un selector para elegir el período para el cual se va a generar la factura.

PFC.WebApp Home About Contact Register Log in

Abonados Generar factura

*Period

[Volver](#) [Guardar](#)

© 2018 - PFC.WebApp

Figura 46: Pantalla de generación de facturas para un abonado

En caso de no haber facturas pendientes de emisión para el abonado seleccionado se mostrará el formulario bloqueado mostrando un mensaje de información indicando que no hay facturas pendientes, tal y como se puede observar en la Figura 47.

PFC.WebApp Home About Contact Register Log in

Abonados Generar factura

El formulario ha sido inhabilitado por la siguiente razón: No hay facturas pendientes de emisión para el abonado actual

*Period

[Volver](#) [Guardar](#)

© 2018 - PFC.WebApp

Figura 47: Pantalla de generación de facturas para un abonado sin facturas pendientes de emisión

9.2.2 MVC de facturas

Como se ha explicado en apartados anteriores, durante la generación de facturas, estas se marcarán como facturas en estado provisional, permitiendo así a un usuario gestor, generar todas las facturas y poder revisarlas antes de emitirlas definitivamente a los clientes.

Para poder visualizar todas las facturas, se ha creado un controlador donde poder ver todas las facturas generadas en la aplicación, de esta forma un usuario gestor no tendrá que ir accediendo a cada abonado para visualizar las facturas y encontrar las que están pendientes. En esta pantalla podrá visualizar las facturas y filtrarlas por período, estado de facturación y dispondrá de opciones para borrar o emitir definitivamente dicha factura.

En la siguiente imagen, Figura 48, se puede observar una captura con la visualización de facturas y las acciones disponibles para cada una de ellas.

PFC.WebApp Home About Contact Register Log in

Facturas Listado

Facturas						Elemento por página:
Factura	Periodo de facturación	Periodo de facturación	Estado	Total		
F000000004	febrero	2017	Final	13.29		
F000000013	febrero	2017	Provisional	17.44		
						Detalles Emitir Borrar

© 2018 - PFC.WebApp

Figura 48: Pantalla de visualización de facturas

A continuación, en la Figura 49, se muestra una imagen con el detalle de una factura, en la que se puede visualizar el desglose por conceptos de los diferentes cargos que componen la factura, así como los impuestos aplicados y el total. También se muestra la línea a la que pertenece la factura, el período de facturación y una tabla con los eventos telefónicos asociados a esta factura.

PFC.WebApp Home About Contact Hola A00000000 Salir

Facturas Detalles

Identificador	F000000001	StartDate	01/01/2017	EndDate	31/01/2017	EmissionDate	01/01/0001	State	Provisional		
CallDuration	4.11:55:26	CallPrice	16,64	DataTrafficBytes	0B	DataTrafficPrice	0,00	SMS	0		
SMSPrice	0,00	Subtotal	22,24	TaxRate	0,07	Total	23,80	Subscription	600132481		
Eventos telefónicos											
Origen	600132481	Destino	679849501	Tipo	Voz	Fecha	2017-01-27T13:33:58	Duración	00:18:17	Coste	0.08
	600132481		627123133		Voz		2017-01-20T01:49:10		00:55:00		0.08

Figura 49: Captura de pantalla de la vista detallada de una factura

Las opciones de emitir y borrar tendrán un control para no permitir el borrado ni la emisión de facturas que ya han sido emitidas.

En la siguiente imagen, Figura 50, se muestra una captura de pantalla con el formulario de emisión de factura bloqueado por la razón comentada anteriormente.

The screenshot shows a web application interface for 'PFC.WebApp'. At the top, there is a navigation bar with links for 'Home', 'About', 'Contact', 'Register', and 'Log in'. Below the navigation bar, the main content area has a title 'Facturas Emitir' with a back arrow icon. A yellow warning box displays the message: 'El formulario ha sido deshabilitado por la siguiente razón: No se puede emitir la factura porque ya ha sido emitida.' (The form has been disabled for the following reason: The invoice cannot be issued because it has already been issued.) Below the warning, there is a table listing various parameters:

Identificador	F0000000004
StartDate	28/02/2017
EndDate	28/02/2017
EmissionDate	07/06/2018
State	Final
CallDuration	02:03:50
CallPrice	0,32
DataTrafficBytes	0B
DataTrafficPrice	0,00
SMS	0
SMSPrice	0,00
Subtotal	7,82
TaxRate	0,70
Total	13,29
Subscription	600363867

At the bottom of the form, there are two buttons: 'Emitir' (in a blue box) and 'Volver' (in a white box).

© 2018 - PFC.WebApp

Figura 50: Captura de pantalla de la vista de emisión de factura bloqueada por estar ya emitida

De forma homóloga, en la Figura 51, se puede observar la pantalla de borrado de una factura cuyo formulario se encuentra bloqueado porque dicha factura ya ha sido emitida.

◀ Facturas Borrar

El formulario ha sido deshabilitado por la siguiente razón: La factura no se puede borrar porque ya ha sido emitida de forma definitiva

¿Está seguro de querer
borrar esta factura?

Identificador	F00000000004
StartDate	28/02/2017
EndDate	28/02/2017
EmissionDate	07/06/2018
State	Final
CallDuration	02:03:50
CallPrice	0,32
DataTrafficBytes	0B
DataTrafficPrice	0,00
SMS	0
SMSPrice	0,00
Subtotal	7,82
TaxRate	0,70
Total	13,29
Subscription	600363867

[Borrar](#)[Volver](#)

© 2018 - PFC.WebApp

Figura 51: Captura de pantalla del formulario de borrado de una factura bloqueado por estar emitida dicha factura



9.3 INFORME DE ABONADOS TELEFÓNICOS CON FACTURAS PENDIENTES DE EMISIÓN

Para facilitar la tarea de generación de facturas se ha añadido un nuevo enlace a la vista de informes donde se podrá obtener un listado de abonados de telefonía cuyo período de vigencia contenga meses para los que no se haya generado factura alguna.

En la siguiente captura se muestra el nuevo enlace que se ha añadido al listado de informes

PFC.WebApp Home About Contact Register Log in

Informes Listado

- Eventos telefónicos sin abonado vigente
- Abonados telefónicos con facturas pendientes de emisión

© 2018 - PFC.WebApp

Y en la Figura 52, una captura con el resultado del informe donde se muestra un resumen de los abonados para los cuales se ha detectado posibles facturas pendientes de emisión, así como el período de vigencia de dicho abonado y las facturas que se han emitido para este.

También se proporciona un enlace para poder acceder a la vista detallada del abonado donde podremos generar las facturas pendientes para este.

PFC.WebApp Home About Contact Register Log in

◀ Informes Abonados telefónicos con facturas pendientes de emisión

MSISDN	SubscriptionDate	CancellationDate	Facturas emitidas	
600363867	28/02/2017		1	🔗
603452136	01/02/2017		0	🔗
603905860	01/02/2017		0	🔗
603957878	01/01/2017		0	🔗
606096393	01/01/2017		0	🔗
607133782	01/01/2017		0	🔗
608381706	01/01/2017		0	🔗
610139474	01/01/2017		0	🔗
611399987	01/01/2017		0	🔗
615843286	01/01/2017		0	🔗

© 2018 - PFC.WebApp

Figura 52: Captura de pantalla del informe de abonados telefónicos con facturas pendientes de emisión

De esta forma, y junto al informe generado en la sección anterior, se proporciona al usuario gestor de dos informes con información muy útil a la hora de trabajar en la aplicación.

10 AUTENTICACIÓN Y SEGURIDAD

Para la autenticación de usuarios, nos basaremos en la plantilla que proporciona MVC y que detallaremos a continuación.

10.1 MIDDLEWARE DE AUTENTICACIÓN Y SERVICIOS

En los proyectos de ASP.Net Core se puede configurar la autenticación a través la configuración de middlewares y servicios.

10.1.1 Configuración del middleware

La arquitectura de ASP.Net Core soporta la inserción de middlewares en la aplicación para controlar las solicitudes y respuestas tal y como se muestra en la Figura 53.

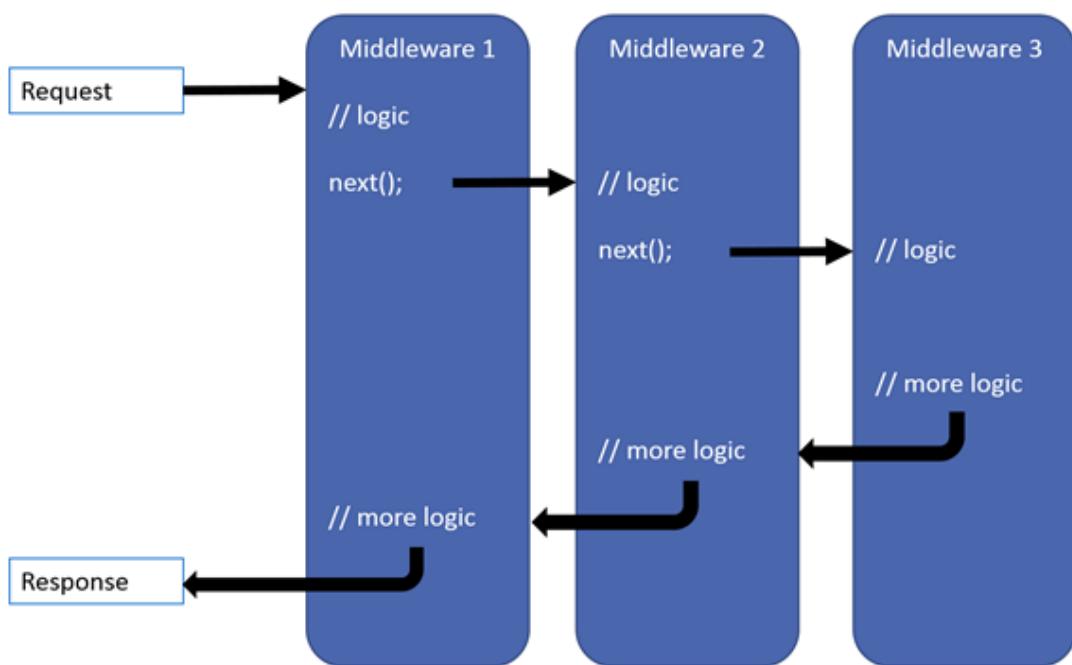


Figura 53: Canalización de solicitudes mediante middleware en ASP.Net Core

Además, ASP.Net Core proporciona un middleware que es responsable de la autenticación automática y del control de solicitudes de autenticación remota.

Para agregar este middleware en el flujo de las solicitudes, en el método Configure de la clase Startup.cs se debe añadir la línea que se muestra en el Código 37 con lo que quedará registrado el Middleware de autenticación.

Código

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...
    app.UseAuthentication();
    ...
}
  
```

Código 37: Fragmento de código con la configuración del Middleware de autenticación

10.1.2 Configuración de los servicios

Para la autenticación se hará uso del servicio Identity ofrecido por ASP.Net Core que proporciona servicios de autenticación y autorización basado en cookie.

Para ello, se modificará la clase ApplicationDbContext que da acceso a la base de datos mediante el ORM EntityFramework tal y como se muestra en el fragmento de Código 39, haciendo que esta implemente IdentityDbContext, proporcionada por el propio EntityFramework y que da soporte de persistencia de datos necesarios por el servicio Identity de ASP.Net Core.

Código

```
public partial class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    ...
}
```

Código 38: Modificación del contexto de datos de EntityFramework para dar soporte al servicio Identity

Para la configuración del contexto de datos con soporte para Identity, se ha implementado una clase que hereda de IdentityUser y que será el objeto que almacenará los datos de usuario necesarios para la autenticación de la aplicación y que más adelante se podrá modificar para añadir información de los usuarios. A continuación, en el Código 39 se muestra la implementación de dicha clase.

Código

```
public class ApplicationUser : IdentityUser { }
```

Código 39: Implementación de IdentityUser para el almacenamiento de los datos de usuario y el soporte de Identity

Una vez modificado el contexto de datos se ejecutarán los comandos para generar la migración que ajuste la base de datos para los nuevos modelos de datos que dan soporte a Identity.

Por último, se configurará el servicio de Identity tal y como se observa en el fragmento de Código 40, donde se ha agregado el servicio indicando la clase que representará los usuarios, el contexto de datos donde persistirá la información de los usuarios, así como el proveedor de tokens por defecto que será usado para recuperar la contraseña o verificar el correo electrónico.

Código

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationContext>()
        .AddDefaultTokenProviders();

    if (HostingEnvironment.IsDevelopment())
        services.AddTransient<IEmailSender>(x => new ConsoleEmailSender());
    else
        services.AddTransient<IEmailSender>(x => new EmailSender(...));

    services.AddMvc()
        .AddMvcOptions(options =>
    {
        AuthorizationPolicy filterPolicy = new AuthorizationPolicyBuilder()
            .RequireAuthenticatedUser()
            .Build();

        AuthorizeFilter filter = new AuthorizeFilter(filterPolicy);
        options.Filters.Add(filter);
    });
    ...
}
```

Código 40: Configuración del servicio de Identity



Además, se ha añadido también un servicio para el envío de correos electrónicos para la regeneración de la contraseña, así como el registro de usuarios y la confirmación del correo electrónico, etc. Se ha creado un servicio para el entorno de desarrollo que muestra el mensaje por la consola de depuración, así como una implementación que usa el protocolo SMTP y que se podrá configurar con una cuenta de correo electrónico real para mandar correos desde el entorno de despliegue.

Otro detalle que se puede observar en el fragmento de código anterior es la configuración que se ha definido para el servicio de MVC, donde se ha añadido un filtro de autorización el cual requiere que haya un usuario autenticado para poder acceder a cualquier controlador.

Este cambio, provocará errores de autorización al intentar acceder a cualquier controlador a menos que se especifique que se permite el acceso de forma anónima. En el siguiente fragmento de Código 41 se muestra el controlador *Home*, el cual se ha modificado agregando el decorador *AllowAnonymous* al nivel de la clase para permitir el acceso anónimo a las pantallas de *Inicio*, *About* y *Contact*.

Código

```
[AllowAnonymous]
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }

    public IActionResult About()
    {
        ViewData["Message"] = "Your application description page.";
        return View();
    }

    public IActionResult Contact()
    {
        ViewData["Message"] = "Your contact page.";
        return View();
    }

    public IActionResult Error()
    {
        return View(new ErrorViewModel
        {
            RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier
        });
    }
}
```

Código 41: Atributo de decoración *AllowAnonymous* para permitir el acceso sin autentificación al controlador *Home*



10.1.3 Inicialización de roles y usuario administrador

Una vez agregado los filtros de autenticación al Framework, se necesitará de al menos un usuario para poder acceder a la aplicación, ya que no se dispondrá de un registro de usuarios, sino que será un usuario con rol de administración o gestor el que registre el resto de los usuarios.

Por tanto, se añadirá un código de inicialización durante el método Configure que se ejecuta al inicio de la aplicación, tal y como se hizo para que se ejecutaran las migraciones en el arranque de la aplicación. De esta forma, queda implementada la inicialización de usuarios tal y como se muestra en el fragmento de Código 1. Se ha agrupado la ejecución de las migraciones junto con la inicialización de roles y usuario en un mismo método ya que juntos conforman la inicialización de la base de datos.

Código

```
using (var scope = app.ApplicationServices
        .GetService<IServiceScopeFactory>().CreateScope())
{
    // Ejecución de las migraciones
    using (var context = scope.ServiceProvider
        .GetRequiredService<ApplicationDbContext>())
    {
        await context.Database.MigrateAsync();
    }

    // Creamos los roles
    using (var roleManager = scope.ServiceProvider
        .GetRequiredService<RoleManager<IdentityRole>>())
    {
        if (!roleManager.Roles.Any())
        {
            await roleManager.CreateAsync(new IdentityRole()
            {
                Name = "SuperAdmin",
                NormalizedName = "Super Administrador"
            });
            await roleManager.CreateAsync(new IdentityRole()
            {
                Name = "Gestor", NormalizedName = "Administración" });
            await roleManager.CreateAsync(new IdentityRole()
            {
                Name = "Comercial", NormalizedName = "Comercial" });
        }
    }

    using (var userManager = scope.ServiceProvider
        .GetRequiredService<UserManager< ApplicationUser >>())
    {
        // Creamos un usuario con derechos de supervisión
        if (!userManager.Users.Any())
        {
            ApplicationUser superAdminUser = new ApplicationUser()
            {
                UserName = "A00000000",
                Email = "pfcwebapp@outlook.com",
                EmailConfirmed = true,
            };

            await userManager.CreateAsync(superAdminUser, "S@metsis2018");
            await userManager.AddToRoleAsync(superAdminUser, "SuperAdmin");
        }
    }
}
```

Código 42: Inicialización de roles y usuario con permisos de administrador



Una vez configurada la autenticación se puede comprobar cómo, accediendo desde el navegador a la Url de cualquier controlador de los creados en el proyecto, el Framework redirigirá la conexión automáticamente a la pantalla de *Inicio de sesión*. En la Figura 54 se muestra la pantalla de *Inicio de sesión* que se ha ajustado para mostrar un estilo similar al que fue definido en los mockups.

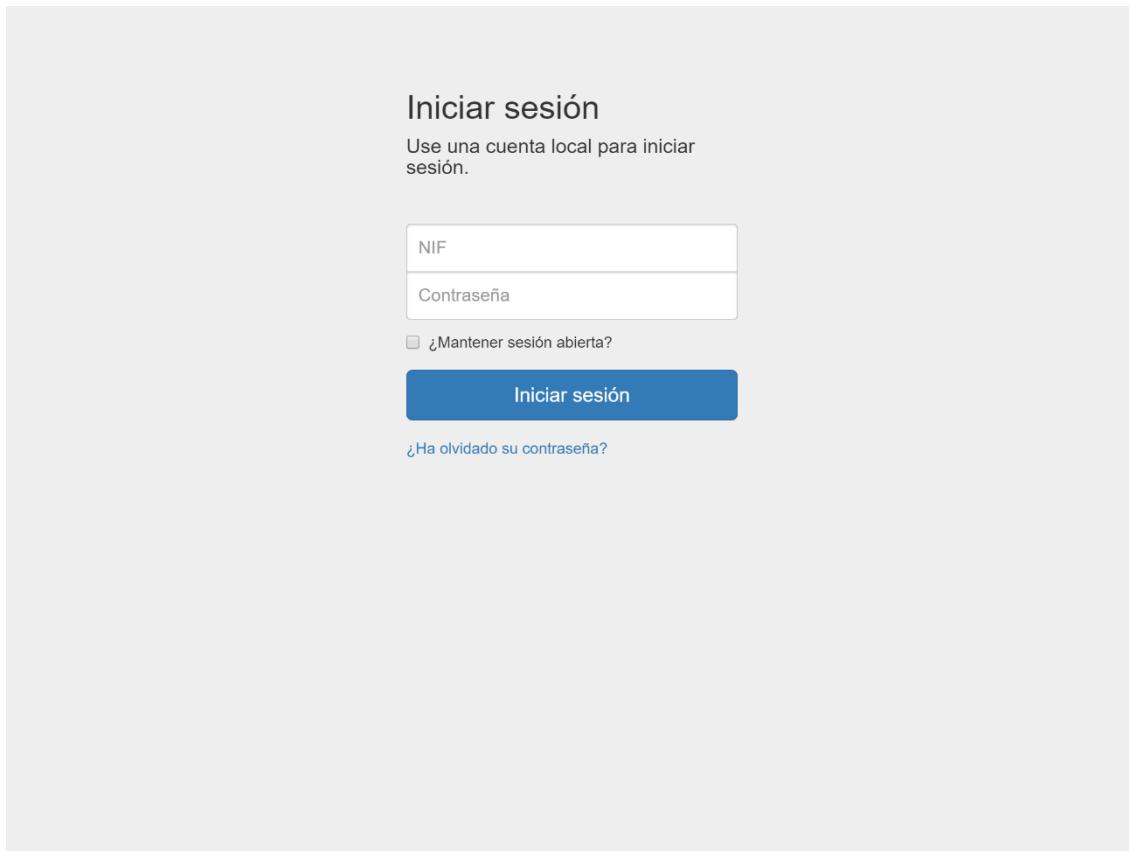


Figura 54: Pantalla de Inicio de sesión

La plantilla de MVC sobre ASP.Net Core proporciona un controlador por defecto para las acciones relacionadas con el perfil de usuario, como llenar los datos de usuario, cambiar o recuperar la contraseña, configurar el inicio de sesión mediante doble factor, etc.

A continuación, en la Figura 55, se muestra la pantalla de configuración del perfil de usuario, donde el propio usuario podrá modificar sus datos de perfil. En esta pantalla se añadirían el resto de información que pudiera ser de interés para la aplicación, como documentación, datos personales, etc.

PFC.WebApp Home About Contact Hola A00000000 Salir

Profile

Manage your account

Change your account settings

Profile	Profile
Password	Username
Two-factor authentication	A00000000
Email	pfwebapp@outlook.com ✓
Phone number	<input type="text"/>
Save	

© 2018 - PFC.WebApp

Figura 55: Pantalla de edición de los datos del perfil de usuario

En la siguiente imagen, Figura 56, se muestra la pantalla de cambio de contraseña.

PFC.WebApp Home About Contact Hola A00000000 Salir

Change password

Manage your account

Change your account settings

Profile	Change password
Password	Current password
Two-factor authentication	<input type="text"/>
New password	<input type="text"/>
Confirm new password	<input type="text"/>
Update password	

© 2018 - PFC.WebApp

Figura 56: Pantalla de cambio de contraseña

10.2 ADMINISTRACIÓN DE USUARIOS

Como ha sido definido en el análisis previo del proyecto, la administración de usuarios la realizará un usuario de la aplicación que tenga asignado el rol de Gestor o Administrador de la aplicación. Por lo tanto, se ha implementado un controlador para gestionar los usuarios de la aplicación, así como las acciones para asignar roles de administración o comercial a estos e incluso, bloquear el acceso de estos.

10.2.1 CRUD para la administración de usuarios

En la ilustración Figura 57, se muestra una captura de pantalla con la tabla de usuarios y las acciones disponibles para cada uno de ellos. Se ha implementado un CRUD similar al resto de pantallas de la aplicación, con las acciones básicas de creación, edición y borrado, así como acciones extra para asignar o desasignar roles o bloquear el acceso a un usuario determinado.

PFC.WebApp Home About Contact Hola A00000000 Salir

Administración Listado

Usuarios			Elemento por página:
			<input type="button" value="10"/>
<input type="button" value="Nuevo"/>			
<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>	<input type="button" value="X"/>
44730589M		false	<input type="button" value="..."/>
<input type="button" value="<"/> <input type="button" value="<"/> <input type="button" value="1"/> <input type="button" value=">"/> <input type="button" value=">"/>			<input type="button" value="(Des) Asignar Gestor"/> <input type="button" value="(Des) Asignar Comercial"/> <input type="button" value="(Des) Bloquear"/>

© 2018 - PFC.WebApp

Figura 57: Pantalla de administración de usuarios

A continuación, en el fragmento de Código 43, se muestra como se ha configurado la seguridad de acceso a este controlador, teniendo ya en cuenta el rol que debe tener el usuario para poder acceder a esta pantalla, que en este caso es el rol de Gestor o administrador.

Código

```
[Authorize(Roles = "SuperAdmin, Gestor")]
public class AdminUsersController : Controller
{
    ...
}
```

Código 43: Filtro de autorización para el acceso al controlador de administración de usuarios

De esta forma, si cualquier usuario al que no se le haya otorgado el rol de Gestor o que no sea el administrador de la aplicación intentase acceder a esta pantalla, la aplicación respondería con un mensaje de error indicando que no dispone de permisos para el acceso a esta pantalla. Al establecerse el filtro a nivel de Controlador queda controlado el acceso a cualquier acción de este. De este mismo modo se controlará el acceso a los controladores que permiten realizar tareas de administración como son: importación de llamadas, gestión de tarifas, edición de abonados de telefonía, facturación, etc.



10.2.2 Asociación de comerciales a usuarios

Para asociar comerciales a usuarios se modificará la administración de usuarios permitiendo asignar a un usuario, otro usuario que como comercial de este. Para ello se agregará una propiedad al modelo del usuario de la aplicación que establezca una relación de comercial entre un usuario y otro con rol de comercial. Esta relación no será obligatoria ya que se podrá dar de alta clientes que no tengan asignado un comercial, así como modificarlos en un futuro para cambiar de comercial o desasignarle el comercial en cuestión.

De esta forma, el modelo de datos de usuario quedará modificado tal y como se muestra en el fragmento de Código 44. Se generará una migración con las utilidades de interfaz de comandos de Entity Framework para ajustar la base de datos con este nuevo cambio en el modelo de datos.

Código

```
public class ApplicationUser : IdentityUser
{
    /// <summary>
    /// Comercial asignado al usuario.
    /// </summary>
    [ForeignKey(nameof(ComercialId))]
    public virtual ApplicationUser Comercial { get; set; }

    /// <summary>
    /// Identificador del comercial asignado al usuario.
    /// </summary>
    public virtual string ComercialId { get; set; }
}
```

Código 44: Modificación del modelo de datos de usuario para asociarlo a un comercial

Una vez hecho esto, se modificará las acciones de administración de usuarios para las acciones de creación y edición de usuarios, permitiendo elegir el comercial asignado a cada usuario.

Se ha modificado la lista de usuarios para mostrar una columna indicando el comercial asociado en caso de tener un comercial asignado. De esta forma la pantalla de administración quedaría como se muestra en la Figura 58.

The screenshot shows a user management interface. At the top, there's a navigation bar with links for PFC.WebApp, Home, About, Contact, Hola A00000000, and Salir. Below the navigation is a title 'Administración de usuarios' followed by 'Listado'. On the left, there's a sidebar with a 'Nuevo' button. The main area has a table with columns: Nombre, Roles, Comercial, and Bloqueado. The 'Comercial' column contains dropdown menus. The first row shows '44730590Y' in the Nombre column, '44730589M' in the Comercial column, and 'false' in the Bloqueado column. The second row shows '44730589M' in the Nombre column, 'Gestor | Comercial' in the Roles column, and 'false' in the Bloqueado column. At the bottom of the table, there are navigation buttons (back, forward, etc.) and a dropdown for 'Elemento por página' set to 10. A context menu is open over the second row, listing options: Editar, (Des) Asignar Gestor, (Des) Asignar Comercial, (Des) Bloquear, and Borrar. The footer of the page includes the text '© 2018 - PFC.WebApp'.

Figura 58: Pantalla de administración de usuarios con Comercial asignado

A continuación, en la Figura 59, se puede observar como quedan las pantallas de edición y creación tras la asociación de usuario con comercial. En ella se ha añadido un selector para asociar el comercial asignado al usuario mediante su NIF.

PFC.WebApp Home About Contact Hola A00000000 Salir

Administración de usuarios Crear

NIF
AXXXXXXX

*Email
xxx@email.com

CommercialId
44730589M ▾

Guardar Volver

© 2018 - PFC.WebApp

Figura 59: Pantalla de creación de usuarios con asociación de comercial

10.3 SEGURIDAD DE ACCESO A ABONADOS DE TELEFONÍA

En esta sección se detallarán los cambios realizados para filtrar el acceso a los datos de la aplicación, permitiendo a los clientes acceder solamente a las líneas telefónicas que sean de su propiedad, así como las facturas generadas sobre dichos abonados telefónicos.

Para ello, se modificará la clase que da soporte como modelo de datos para la persistencia de los abonados de telefonía, añadiéndose una propiedad que relacione dicho modelo con un usuario de la aplicación. De esta forma el modelo de datos quedará modificado conteniendo las líneas de código que se muestran en el fragmento de Código 45.

Código

```
public class Subscription
{
    [Required]
    public virtual string UserId { get; set; }

    /// <summary>
    /// Usuario al que pertenece el Abonado.
    /// </summary>
    public virtual ApplicationUser User { get; set; }
}
```

Código 45: Modificación del modelo de datos de Abonado de línea telefónica para asociarlo a un usuario como cliente

Como se ha comentado en otros casos, tras este cambio se ejecutarán el comando de Entity Framework que generará un código migración para mantener la base de datos actualizada acorde al modelo de datos tras la modificación.

Una vez modificado el modelo de datos se ajustará el CRUD de abonados de telefonía para que muestre el usuario al que pertenece el abonado, así como permita asignar el propietario al dar de alta nuevos abonados.

En la siguiente Figura 60, se muestra el formulario de creación de abonados con el nuevo selector para asignar el cliente al que quedará asociado dicho abonado de telefonía.

PFC.WebApp Home About Contact Hola A00000000 Salir

Abonados Crear

*UserId
44730590Y

*MSISDN

*PlanId
100MB - 0 min

*SubscriptionDate
dd/mm/aaaa

CancellationDate
dd/mm/aaaa

Guardar Volver

© 2018 - PFC.WebApp

Figura 60: Pantalla de creación de abonados con el selector del identificador del cliente

Ahora que ya se pueden crear abonados de telefonía asociados a un usuario como cliente asignado, se modificará el controlador para filtrar el listado de abonados mostrando solo aquellos que están asociados al usuario que esté autenticado en la aplicación, y controlar el acceso a las acciones que este podrá realizar sobre dichos abonados en función del rol.

Para ellos se ha modificado el helper de Datatable permitiendo asignar un role requerido para determinadas acciones, encargándose éste de ignorar aquellos enlaces en la renderización de las vistas para los cuales el usuario no tendría acceso, además de agregarle los filtros de autorización pertinentes a cada una de las acciones para evitar que un usuario pudiera forjar una solicitud para intentar acceder a estas acciones que se han filtrado visualmente.

De esta forma el código del controlador quedará modificado tal y como se muestra en el fragmento de Código 46. En este fragmento se muestran solo aquellas líneas que intervienen en los filtros de información y el control de la seguridad.

Código

```
public async Task<IActionResult> Index()
{
    var dataTableDefinition = DataTableDefinition.Create<Subscription>()
    ...
    .AddGlobalActions("Nuevo", Url.Action(nameof(Create)), "Gestor,SuperAdmin")
    ...
    .AddElementAction("Editar", new DataTableDefinition.UrlActionDTO()
    {
        Action = "Edit",
        Values = new Dictionary<string, string> { { "id", "ID" } }
    }, "Gestor,SuperAdmin")
    .AddElementAction("Borrar", new DataTableDefinition.UrlActionDTO()
    {
        Action = "Delete",
        Values = new Dictionary<string, string> { { "id", "ID" } }
    }, "Gestor,SuperAdmin")
    ...
}

[HttpPost]
public async Task<JsonResult> Index(...)
{
    var user = await _userManager.FindByNameAsync(User.Identity.Name);

    DataTableDefinition.DataQuery data = DataTableDefinition
        .DataQueryBuilder(index, length, orderProperty, inverse, query, _context.Subscriptions
        .Include(x => x.User)
        .Include(x => x.Plan)
        .Where(x => User.IsInRole("SuperAdmin")
            || User.IsInRole("Gestor")
            || x.UserId == user.Id));
    ...
}

[Authorize(Roles = "Gestor,SuperAdmin")]
public IActionResult Create()
{
    ...
}

[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Gestor,SuperAdmin")]
public async Task<IActionResult> Create(...)
{
    ...
}

public async Task<IActionResult> Details(int id)
{
    var subscription = await _context.Subscriptions
        .Include(s => s.Plan)
```



```

    .Include(s => s.User)
    .SingleOrDefaultAsync(s => s.ID == id && s.User.UserName == User.Identity.Name);

}

[Authorize(Roles = "Gestor,SuperAdmin")]
public async Task<IActionResult> Edit(int id)
{
    ...
}

[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Gestor,SuperAdmin")]
public async Task<IActionResult> Edit(...)
{
    ...
}

[Authorize(Roles = "Gestor,SuperAdmin")]
public async Task<IActionResult> Delete(int id)
{
    ...
}

// POST: Subscriptions/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Gestor,SuperAdmin")]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    ...
}

```

Código 46: Modificaciones en el controlador de abonados para filtrar y controlar la seguridad relativa al usuario

A continuación, se muestra una captura de pantalla de la lista de abonados para un cliente, observándose como solamente tiene acceso a sus abonados de telefonía, así como acceso exclusivamente a la vista detallada de cada una de ellas y no a las acciones de uso reservadas para el personal con rol de administración o gestión.

Usuario	Número de teléfono	Tarifa	Fecha de alta	Fecha de baja
44730589M	636581559	100MB - 0 min	2018-07-07T00:00:00	

© 2018 - PFC.WebApp

Figura 61: Pantalla de abonados de telefonía con los resultados para un cliente sin permisos de administración

10.4 ACCESO A COMISIONES

Se ha implementado un controlador de uso exclusivo para los usuarios con el rol de comercial desde el cual podrán visualizar las facturas que han generado los usuarios que tiene asignados como comercial, de esta forma, dispone de una pantalla donde poder filtrar por los diferentes campos de las facturas y calcular sus comisiones.

Para llevar a cabo esta utilidad se ha modificado la acción que emite las facturas, asignando en ese momento el comercial asociado al cliente del abonado para el cual se está emitiendo la factura, en caso de que el cliente tenga un comercial asignado. De esta forma, la pantalla de comisiones solo mostrará al comercial aquellas facturas que tenga asignadas.

En el siguiente fragmento de Código 47 se muestra la modificación llevada a cabo en la acción de emisión de facturas donde se asigna el comercial que recibirá comisiones para dicha factura.

Código

```
public async Task<IActionResult> EmitConfirmed(int id)
{
    var invoice = await _context.Invoice
        .Include(x => x.Subscription.User)
        .SingleOrDefaultAsync(m => m.ID == id);
    if (invoice.State != InvoiceStateEnum.Provisional)
    {
        ModelState.AddModelError("", "La factura ya ha sido emitida, si desea volver a generar la factura bórrela y vuelva a generarla");
        return View(invoice);
    }

    invoice.State = InvoiceStateEnum.Final;
    invoice.EmissionDate = DateTime.Now;
    invoice.ComercialId = invoice.Subscription.User.ComercialId;

    await _context.SaveChangesAsync();

    return RedirectToAction(nameof(Index));
}
```

Código 47: Modificación en la de emisión de facturas para asignar el comercial que comisiona sobre dicha factura

A continuación, en el fragmento de Código 48, se muestra la implementación del controlador que permite al usuario con rol de *Comercial* visualizar un listado de facturas sobre las que comisionará.

Este controlador se ha implementado usando el Helper de DataTableDefinition como en casos anteriores y en su implementación se puede observar cómo se ha aplicado el filtro de autorización a nivel de controlador para permitir el acceso exclusivamente a usuarios con rol de Comercial.

Código

```
[Authorize(Roles = "Comercial")]
public class CommissionController : Controller
{
    private readonly ApplicationDbContext _context;

    public CommissionController(ApplicationDbContext context)
    {
        _context = context;
    }

    public override void OnActionExecuting(ActionExecutingContext context)
    {
        base.OnActionExecuting(context);
        ViewBag.Title = "Comisiones";
    }

    public IActionResult Index()
    {
```



```

ViewBag.SubTitle = "Listado";

    var dataTableDefinition = DataTableDefinition.Create<Invoice>()
        .WithTitle("Comisiones")
        .MapColumn(nameof(Invoice.Identificador), "Factura", 100)
        .MapColumn(nameof(Invoice.YearPeriod), "Período de facturación", 100)
        .MapColumn(nameof(Invoice.MonthPeriod), "Período de facturación", 100)
        .MapColumn(nameof(Invoice.Total), "Total", 100);

    return View(dataTableDefinition);
}

[HttpPost]
public JsonResult Index(int index = 0, int length = 10, string orderProperty = null, bool
inverse = false, [FromBody]IEnumerable<QueryFilter> query = null)
{
    IQueryable<Invoice> invoice = _context.Invoice
        .Where(x => x.Comercial.UserName == User.Identity.Name && x.State >
InvoiceStateEnum.Provisional);

    DataTableDefinition.DataQuery data =
        DataTableDefinition.DataQueryBuilder(index, length, orderProperty, inverse, query,
invoice);

    return Json(data);
}
}

```

Código 48: Implementación del controlador de comisiones para comerciales

La vista resultante se puede observar en la Figura 62, donde se muestra una captura de pantalla con las facturas sobre las que comisiona un usuario con rol de Comercial.

Factura	Periodo de facturación	Periodo de facturación	Estado	Total
F000000002	junio	2018	Final	3.21

© 2018 - PFC.WebApp

Figura 62: Captura de pantalla de la pantalla de comisiones

10.5 MENÚ DE USUARIO

Para un acceso a las diferentes pantallas se ha diseñado un menú lateral con los enlaces de acceso a los controladores, filtrando aquellos a los que se tiene permiso.

Para ello se ha hecho uso de reflexión, que es una característica de .NET que proporciona objetos con la descripción de los tipos. Se ha implementado una clase que mediante reflexión explora los controladores de la aplicación, capturando aquellos que tienen filtro de autorización con roles definidos. De esta forma podremos generar un menú que muestre solamente los enlaces a los que tiene acceso el usuario que tenga iniciada la sesión.

En el siguiente fragmento de Código 49 se muestra como se ha cargado dicha información analizando los controladores con reflexión:

Código

```
public static class ControllerAuthorization
{
    static IDictionary<string, string[]> ControllerRoles;

    static ControllerAuthorization()
    {
        ControllerRoles = Assembly.GetExecutingAssembly().GetTypes()
            .Where(t => typeof(Controller).IsAssignableFrom(t))
            .Select(t => new {
                t.Name,
                Roles = t.GetCustomAttribute<AuthorizeAttribute>()?.Roles?.Split(",") ?? new string[0]
            })
            .ToDictionary(x => x.Name, x => x.Roles);
    }

    public static string[] ControllerAuthorizedRoles(string controllerName)
    {
        controllerName = controllerName.EndsWith("Controller")
            ? controllerName
            : controllerName + "Controller";
        return ControllerRoles[controllerName];
    }
}
```

Código 49: Carga dinámica de los controladores con filtro de autorización por Roles

Además, se ha modificado el layout base de la aplicación para dibujar el menú como panel lateral en el lado izquierdo.

Con la información obtenida anteriormente de los controladores, el código que genera el menú se simplifica bastante, evitando tener que actualizar los menús si se cambian los roles de acceso de algún controlador.

En el fragmento de Código 50 siguiente se puede observar la generación del menú y lo simple que ha quedado:

Código

```
@functions {
    Microsoft.AspNetCore.Html.IHtmlContent RenderIfPermitted(string controller, string text)
    {
        var roles = ControllerAuthorization.ControllerAuthorizedRoles(controller);
        return (!roles.Any() || roles.Any(x => User.IsInRole(x)))
            ? Html.Partial("_MenuEntry", Tuple.Create(controller, text))
            : new Microsoft.AspNetCore.Html.HtmlContentBuilder();
    }
}
```



```

<ul class="nav nav-sidebar">
    @RenderIfPermitted("AdminUsers", "Usuarios")
    @RenderIfPermitted("Commission", "Comisiones")
    @RenderIfPermitted("Invoices", "Facturas")
    @RenderIfPermitted("MasMovilCSV", "MasMóvil")
    @RenderIfPermitted("PhoneEvents", "Eventos")
    @RenderIfPermitted("Plans", "Tarifas")
    @RenderIfPermitted("Report", "Informes")
    @RenderIfPermitted("Subscriptions", "Abonados")
</ul>

```

Código 50: Plantilla con la generación de los elementos del menú

A continuación, se muestran varias capturas de pantalla donde se puede observar el resultado final del menú. Cada una de las capturas muestra un menú diferente renderizado para cada uno de los roles definidos en la aplicación.

Nombre	Roles	Comercial	Bloqueado
00000001R		00000000T	false
00000000T	Comercial		false

Figura 63: Menú de un usuario con el rol gestor

Facturas Listado

Factura	Periodo de facturación	Periodo de facturación	Estado	Total
F0000000007	enero	2017	Final	5.99
F0000000010	enero	2017	Final	22.43

© 2018 - PFC.WebApp

Figura 64: Menú de un usuario con el rol cliente

Comisiones Listado

Factura	Periodo de facturación	Periodo de facturación	Total
F0000000007	enero	2017	5.99
F0000000010	enero	2017	22.43

© 2018 - PFC.WebApp

Figura 65: Menú de un usuario con el rol comercial

11 REFERENCIAS

1. Visual Studio Code. [En línea] <https://code.visualstudio.com/>.
2. Microsoft® Office. [En línea] <https://products.office.com/es-es/home>.
3. Git. [En línea] <https://git-scm.com/>.
4. Introducción a ASP.NET Core. [En línea] <https://docs.microsoft.com/en-us/aspnet/core/>.
5. Información general de ASP.NET Core MVC. [En línea] <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview>.
6. Descripción general de Entity Framework Core. [En línea] <https://docs.microsoft.com/ef/core/>.
7. (LINQ), Language-Integrated Query. [En línea] [https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq).
8. jQuery. [En línea] <https://jquery.com/>.
9. W3C HTML. [En línea] <https://www.w3.org/html/>.
10. Cascading Style Sheets. [En línea] <https://www.w3.org/Style/CSS/>.
11. Guía de C#. [En línea] <https://docs.microsoft.com/es-es/dotnet/csharp/index>.
12. C# for Visual Studio Code. [En línea] <https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp>.



12 ILUSTRACIONES

Figura 1: Casos de uso del gestor en el subsistema de proveedores.....	6
Figura 2: Casos de uso del gestor en el subsistema de Contratación	7
Figura 3: Casos de uso del cliente en el subsistema de telefonía	7
Figura 4: Casos de uso en el subsistema de usuarios	8
Figura 5: Mockup de la plantilla de la aplicación	9
Figura 6: Mockup de la pantalla de inicio de sesión	10
Figura 7: Mockup de la pantalla de importación de ficheros CSV.....	11
Figura 8: Mockup de la pantalla de visualización de ficheros CSV de MásMóvil	12
Figura 9: Mockup de la pantalla de administración de tarifas de telefonía	13
Figura 10: Mockup de la pantalla de administración de abonados de telefonía	14
Figura 11: Mockup de las líneas de un cliente	15
Figura 12: Mockups de la pantalla de facturas del cliente	16
Figura 13: Mockup de la pantalla de clientes asignados a un comercial	17
Figura 14: Mockup de la pantalla de comisiones de un comercial	18
Figura 15: Mockup de la pantalla de detalles de una comisión	19
Figura 16: Vista previa de Visual Studio Code.....	23
Figura 17: Instalación de extensiones desde el MarketPlace de Visual Studio Code	24
Figura 18: Vista previa de la aplicación básica en ejecución.....	25
Figura 19: Panel de inicio de sesión del dispositivo	27
Figura 20: Vista de índice del scaffold de ejemplo.....	31
Figura 21: Diagrama de clases UML del Gestor Documental	33
Figura 22: Vista de índice con los ficheros CSV de MasMóvil	36
Figura 23: Vista con el formulario de subida de fichero CSV mostrando errores de validación tras subir un fichero incorrecto.	37
Figura 24: Vista de un fichero CSV	39
Figura 25: Diagrama de clases UML que representa el modelo de datos para la persistencia de los ficheros importados.....	40
Figura 26: Diagrama de clases UML con el modelo de datos que representa un evento telefónico	42
Figura 27: Captura de pantalla de la visualización de eventos telefónicos desde la base de datos	47
Figura 28: Vista detalle de un evento telefónico	48
Figura 29: Pantalla de visualización de tarifas	50
Figura 30: Pantalla de creación de tarifa nueva.....	51
Figura 31: Pantalla con la vista detallada de una tarifa	51
Figura 32: Pantalla con la vista de edición de una tarifa.....	52
Figura 33: Pantalla de borrado de una tarifa	52
Figura 34: Diagrama de clases UML que muestra la relación de agregación entre una suscripción y la tarifa asociada	53
Figura 35: Modelo de datos para la persistencia de los abonados de telefonía	54
Figura 36: Código generado por el sistema de migraciones de Entity Framework para el modelo de datos del Abonado.....	54
Figura 37: Mensaje de validación en la pantalla de borrado de una tarifa que está siendo usada por algún abonado	55
Figura 38: Pantalla de visualización de abonados.....	56

Figura 39: Pantalla de edición bloqueada por existencia de un abonado con fecha de alta posterior.....	57
Figura 40: Pantalla de borrado de abonado bloqueada por existencia de un abonado con fecha de alta posterior	57
Figura 41: Pantalla de edición de abonado mostrando errores de validación.....	58
Figura 42: Vista de índice de los informes disponibles	59
Figura 43: Informe de eventos telefónicos sin abonado vigente.....	60
Figura 44: Diagrama de clases UML que representa una factura telefónica	61
Figura 45: Pantalla de detalle de un abonado de telefonía con sus facturas.....	62
Figura 46: Pantalla de generación de facturas para un abonado	63
Figura 47: Pantalla de generación de facturas para un abonado sin facturas pendientes de emisión	63
Figura 48: Pantalla de visualización de facturas	64
Figura 49: Captura de pantalla de la vista detallada de una factura.....	64
Figura 50: Captura de pantalla de la vista de emisión de factura bloqueada por estar ya emitida	65
Figura 51: Captura de pantalla del formulario de borrado de una factura bloqueado por estar emitida dicha factura.....	66
Figura 52: Captura de pantalla del informe de abonados telefónicos con facturas pendientes de emisión	67
Figura 53: Canalización de solicitudes mediante middleware en ASP.Net Core	68
Figura 54: Pantalla de Inicio de sesión.....	72
Figura 55: Pantalla de edición de los datos del perfil de usuario.....	73
Figura 56: Pantalla de cambio de contraseña	73
Figura 57: Pantalla de administración de usuarios	74
Figura 58: Pantalla de administración de usuarios con Comercial asignado	75
Figura 59: Pantalla de creación de usuarios con asociación de comercial.....	76
Figura 60: Pantalla de creación de abonados con el selector del identificador del cliente	77
Figura 61: Pantalla de abonados de telefonía con los resultados para un cliente sin permisos de administración.....	79
Figura 62: Captura de pantalla de la pantalla de comisiones	81
Figura 63: Menú de un usuario con el rol gestor	83
Figura 64: Menú de un usuario con el rol cliente	84
Figura 65: Menú de un usuario con el rol comercial	84

13 FRAGMENTOS DE CÓDIGO

Código 1: Instalación de la extensión de C# para Visual Studio Code	25
Código 2: Creación de una aplicación web con .NET Core	25
Código 3: Inicio de sesión en Azure mediante CLI	26
Código 4: Comandos para crear un servicio de aplicación web para el despliegue en Azure.....	27
Código 5: Definición de credenciales y recuperación de URL para Git	28
Código 6: Configuración del repositorio local de Git	28
Código 7: Comando para obtener la Url de la aplicación desplegada	28
Código 8: Configuración de las utilidades de consola para Entity Framework	29
Código 9: Modelo de datos básico para el ejemplo de configuración de Entity Framework.....	29
Código 10: Implementación de DbContext.....	29
Código 11: Configuración del servicio de Entity Framework en el punto de inicio de la aplicación	30
Código 12: Configuración de la cadena de conexión con una base de datos SQL Server local....	30
Código 13: Comandos para configurar el sistema de migraciones de Entity Framework.....	30
Código 14: Scaffold de un controlador básico con acciones CRUD.....	31
Código 15: Rutas de MVC configuradas por defecto	31
Código 16: Comando para crear un servidor de SQL Server en Azure.....	31
Código 17: Configuración de una regla del cortafuegos del servidor	32
Código 18: Configuración de la cadena de conexión con la base de datos.....	32
Código 19: Configuración de la ruta de la carpeta usada por el Gestor Documental	34
Código 20: Inyección del Gestor Documental en el contendor de dependencias de ASP:.NET Core	34
Código 21: Resolución de dependencia del gestor documental para el controlador de ficheros CSV de MásMóvil	34
Código 22: Acción para listar los ficheros CSV de MásMóvil	35
Código 23: Clase que representa el modelo de datos de los registros de llamadas del proveedor MásMóvil	37
Código 24: Clase que enumera los registros de un fichero en formato CSV.....	38
Código 25: Modelo de datos para la persistencia del estado de importación de los ficheros CDV de MásMóvil	40
Código 26: Entrada en el DbContext del modelo de datos para el estado de importación de los ficheros CSV de MásMóvil	41
Código 27: Inyección de dependencia del contexto de datos a través del constructor	41
Código 28: Modificación del método para subir ficheros CSV de MásMóvil para establecer el estado de importación.....	41
Código 29: Modificación del método de eliminar para contemplar el estado de importación ...	42
Código 30: Modelo de datos para la persistencia de los eventos telefónicos	43
Código 31: Método de importación de ficheros CSV de MásMóvil a la base de datos.....	43
Código 32: Tarea asíncrona de importación de eventos telefónicos desde un fichero CSV de MásMóvil	44
Código 33: Agregación del enlace de impotación a la definición de la tabla de visualización de ficheros CSV de MásMóvil	45
Código 34: Controlador de visualización de eventos telefónicos	47
Código 35: Modelo de datos para la persistencia de las tarifas.....	49
Código 36: Fragmento de código con la generación de los datos del informe de eventos telefónicos sin abonado vigente	60



Código 37: Fragmento de código con la configuración del Middleware de autenticación	68
Código 38: Modificación del contexto de datos de EntityFramework para dar soporte al servicio Identity	69
Código 39: Implementación de IdentityUser para el almacenamiento de los datos de usuario y el soporte de Identity	69
Código 40: Configuración del servicio de Identity	69
Código 41: Atributo de decoración AllowAnonymous para permitir el acceso sin autentificación al controlador Home	70
Código 42: Inicialización de roles y usuario con permisos de administrador.....	71
Código 43: Filtro de autorización para el acceso al controlador de administración de usuarios.	74
Código 44: Modificación del modelo de datos de usuario para asociarlo a un comercial	75
Código 45: Modificación del modelo de datos de Abonado de línea telefónica para asociarlo a un usuario como cliente	77
Código 46: Modificaciones en el controlador de abonados para filtrar y controlar la seguridad relativa al usuario	79
Código 47: Modificación en la de emisión de facturas para asignar el comercial que comisiona sobre dicha factura.....	80
Código 48: Implementación del controlador de comisiones para comerciales	81
Código 49: Carga dinámica de los controladores con filtro de autorización por Roles.....	82
Código 50: Plantilla con la generación de los elementos del menú.....	83

14 ANEXOS

14.1 ANEXO I: DESCRIPCIÓN DEL FICHERO DE LLAMADAS DEL PROVEEDOR MÁSMÓVIL

MM! DESCRIPCIÓN EDR
SEPTIEMBRE'13 · V2.0



MÁSMÓVIL EDR EVENT DATA RECORD

CAMPO	TIPO	DESCRIPCIÓN														
DEALER	VARCHAR2	Código de dealer-tienda														
FECHA_EXTRACCION	DATE (DD/MM/YYYY HH:MM:SS)	Fecha de exportación del fichero														
FECHA	DATE (DD/MM/YYYY HH:MM:SS)	Fecha del evento (llamada, sms, mms, datos)														
FACTURA	STRING (8) YYYYMMDD	Fecha de cierre de facturación. Por ejemplo, tráfico del 15/05/2012 tendrá una fecha de factura de 20120601.														
MSISDN	STRING (9)	Número de teléfono origen (MÁSMÓVIL)														
TIPO_DESTINO	STRING (256)	Tipología de tráfico														
		<table border="1"> <thead> <tr> <th>VALOR</th><th>DESCRIPCIÓN</th></tr> </thead> <tbody> <tr> <td>VOZ NACIONAL</td><td>Destino Voz nacional</td></tr> <tr> <td>SMS NACIONAL</td><td>Destino SMS nacional</td></tr> <tr> <td>MMS NACIONAL</td><td>Destino MMS nacional</td></tr> <tr> <td>NUMERO ESPECIAL</td><td>Destino nacional-especial</td></tr> <tr> <td>DATOS NACIONAL</td><td>GPRS nacional</td></tr> <tr> <td>COUNTRY</td><td>Destino voz internacional - País destino</td></tr> <tr> <td>SMS INTERNACIONAL</td><td>Destino SMS internacional</td></tr> </tbody> </table>	VALOR	DESCRIPCIÓN	VOZ NACIONAL	Destino Voz nacional	SMS NACIONAL	Destino SMS nacional	MMS NACIONAL	Destino MMS nacional	NUMERO ESPECIAL	Destino nacional-especial	DATOS NACIONAL	GPRS nacional	COUNTRY	Destino voz internacional - País destino
VALOR	DESCRIPCIÓN															
VOZ NACIONAL	Destino Voz nacional															
SMS NACIONAL	Destino SMS nacional															
MMS NACIONAL	Destino MMS nacional															
NUMERO ESPECIAL	Destino nacional-especial															
DATOS NACIONAL	GPRS nacional															
COUNTRY	Destino voz internacional - País destino															
SMS INTERNACIONAL	Destino SMS internacional															
Destino, dos posibles valores:																
<ul style="list-style-type: none"> ▪ Número de teléfono destino en caso de tráfico de VOZ/SMS/MMS ▪ "DATA" en caso de tráfico de datos (GPRS) 																
Volumen de tráfico, dependiendo del tipo de tráfico:																
<ul style="list-style-type: none"> ▪ Número de segundos, en caso de VOZ. ▪ 0 en caso de SMS/MMS ▪ Kilobytes (KB) con dos decimales. Por ejemplo el consumo de 512,00KB sera mostrado como 51200 																
VALOR	NUMBER	Coste del evento. Impuestos indirectos no incluidos.														
IMPUESTOS	NUMBER	Impuestos indirectos asociados a valor (IVA, IPSI, IGIC)														

- MÁSMÓVIL depositará esta información en un fichero con formato de texto plano (.TXT) cuyo formato será *Dealer-tienda_AñoMesDiaHoraMinutosSegundos.txt*
- Dentro del fichero, se usará el carácter *pipe* (|) como tabulador entre los diferentes campos.
- Los ficheros serán depositados en un servidor de MÁSMÓVIL, el cliente podrá descargarlos usando el protocolo SFTP.
- MÁSMÓVIL depositará los ficheros de manera diaria, a día vencido. El tráfico del día N será depositado en el servidor el día N+1 y al final el mes se realizará un fichero especial con el tráfico mensual.
- Detalles de conexión: Servidor SFTP: 212.166.71.71
Usuario: Código de Dealer
Contraseña: Sera facilitada por mail siguiendo el procedimiento indicado.