Universidad Zaragoza
1542

Trabajo Fin de Grado

Desarrollo de un módulo de gestión de viajes y gastos

ANEXOS

Autor

Nicolás Bailo Ortiz

Director y ponente

Fernando Cortés Franco

Director/CEO (Endalia)

Santiago Velilla Marco

Departamento de Informática e Ingeniería de sistemas (Universidad de Zaragoza)

Escuela de Ingeniería y Arquitectura

2016

Tomo 2/2

ÍNDICE DE CONTENIDOS

endalia®
HR Consulting & Software

ESTÁNDAR DE DOCUMENTACIÓN

DESARROLLO DE UN MÓDULO DE GESTIÓN DE VIAJES Y GASTOS

VERSIÓN 2.0
PUBLICADO EL 29/08/2016

HISTÓRICO DE REVISIONES

Fecha	Versión	Descripción	Autor
03/09/2007	1.0	Redacción del Estándar de Documentación de Endalia	Endalia
23/08/2016	2.0	Modificaciones del documento para adaptarlo al TFG	Nicolás Bailo Ortiz

ÍNDICE

1. INTRODUCCIÓN

1.1 Propósito del documento

En el presente documento se define el estándar de documentación del trabajo de desarrollo del módulo de gestión de viajes y gastos. Se definen los formatos, diseños, tipología de fuentes y plantillas a utilizar en la elaboración de esta documentación.

1.2 Alcance del documento

Las especificaciones de este documento alcanzan a toda la documentación generada durante el proyecto.

1.3 Definiciones

Fuente: un miembro de una familia de tipo de letra.

1.4 Referencias

En este documento no se realizan referencias a otros documentos del trabajo.

1.5 Resumen

El presente documento es el estándar de documentación de Endalia. Se compone de cuatro apartados:

1. Se muestra el propósito del documento y se define su alcance. Se proporciona una lista de acrónimos y definiciones útiles para la comprensión del documento, así como una lista de los documentos del proyecto referenciados y el presente resumen.
2. Se especifica el formato de documentación del proyecto.
3. Plantillas de documentación.
4. Bibliografía y referencias Web utilizadas en la confección de este documento.

2. FORMATO DE DOCUMENTACIÓN

2.1 Fuentes y estilos

A continuación se definen los tipos de fuente utilizados para los diferentes formatos de texto del documento:

- Título 1: Helvética Neue 14, Negrita.
- Título 2: Helvética Neue 12, Negrita.
- Titulo 3: Helvética Neue 10.
- Texto normal: Helvética Neue 9.
- Texto en pie de imágenes: Helvética Neue 9, Cursiva.

Texto de Código fuente o acciones de línea de comandos: Courier New 9.

2.2 Interlineado y formato de página

El interlineado utilizado en la documentación será sencillo. El texto se justificará por ambos márgenes.

Se empezará página nueva entre apartados de primer nivel (Título 1).

Se evitará dejar títulos de segundo y tercer nivel como última línea de una página, siendo ubicados en la página siguiente.

Entre todos los títulos independientemente del nivel, habrá dos saltos de línea de separación.

Las relaciones de elementos se separarán mediante un salto de línea y se indicarán con un punto al principio de la línea, sin tabulado. En el caso de subrelaciones, se indicarán mediante tabulados sin guion de la siguiente manera:

- Elemento 1
    - Elemento de Nivel 2
        - Elemento de Nivel 3
        - Elemento de Nivel 3
        - Elemento de Nivel 3
    - Elemento de Nivel 2
    - Elemento de Nivel 2
    - Elemento de Nivel 2
    - Elemento de Nivel 2
- Elemento 2

2.3 Imágenes y diagramas

Las imágenes y diagramas se colocarán centrados y ajustando en lo posible su tamaño a los márgenes habituales de la página, excepto en el caso de que su tamaño sea excesivo para visualizarlos de manera óptima dentro de esos márgenes, en cuyo caso se colocarán en posición apaisada en una página nueva. Todas las imágenes estarán numeradas y se les referenciará en el texto al que acompañan y mediante una definición centrada debajo de las mismas, de la siguiente manera (Figura 1):

Figura 1. Ejemplo de formato de imagen

3. PLANTILLAS DE DOCUMENTACIÓN

Las plantillas explicadas en este apartado se encuentran guardadas en formato electrónico. Por razones de espacio no se pueden mostrar aquí a tamaño real.

El objetivo de este apartado es especificar el formato de los documentos para cualquiera que desarrolle algún texto para el proyecto. Por ello se muestra una captura a tamaño reducido y una explicación de las partes que las componen y la manera de utilizarlas. Como muestra del aspecto final sirve el presente documento.

3.1 Plantilla de documento

3.1.1 Hoja 1. Portada

El formato de la portada de todos los documentos es el siguiente (Figura 2).

Figura 2. Hoja 1 – Portada

En él se indica el título del documento con fuente Helvética Neue 60 y se incluye información de copyright y de control de distribución y autorización.

Todas las demás páginas del documento poseen un pie de página común, en el que aparece el logotipo de Endalia e incluye el título del documento, así como información del número de página.

3.1.2 Hoja 2. Información de copyright

El formato de la segunda hoja de todos los documentos es el siguiente (Figura 3).

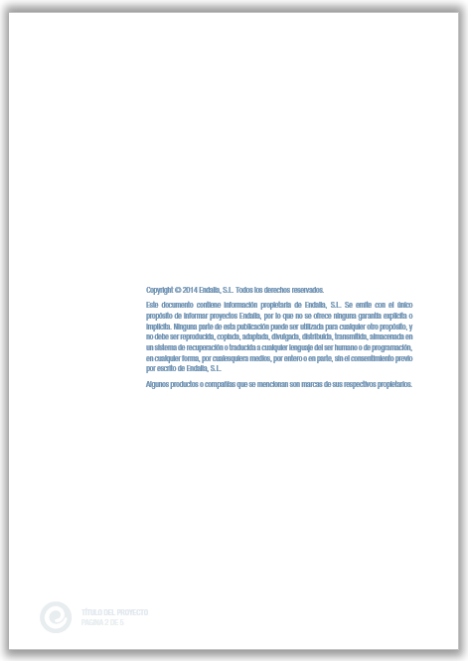


Figura 3. Hoja 2 - Información de copyright

En esta hoja, aparte del pie de página, aparece información relativa al copyright del documento.

3.1.3 Hoja 3. Histórico de revisiones

El formato de la tercera hoja de todos los documentos es el siguiente (Figura 4).

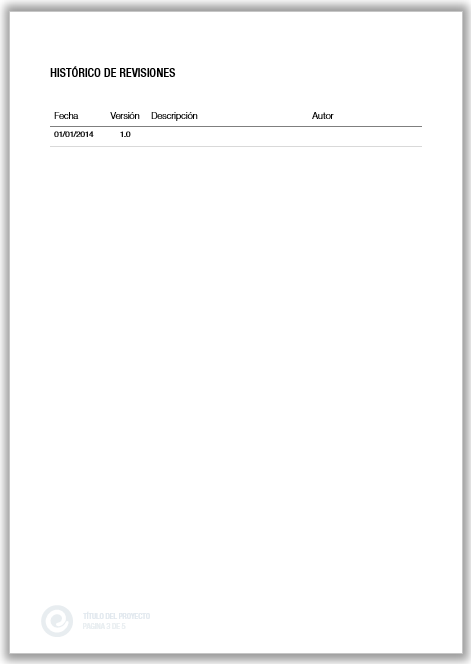


Figura 4. Hoja 3 - Histórico de revisiones

En esta hoja, aparte del pie de página, aparece un cuadro con información acerca de las sucesivas revisiones realizadas sobre el documento.

3.1.4 Hoja 4 y siguientes. Índice

El formato del índice se indica en la siguiente imagen (Figura 5). Se genera automáticamente con Word, estableciendo el tipo de letra de los apartados de nivel 1 a Helvética Neue 9.

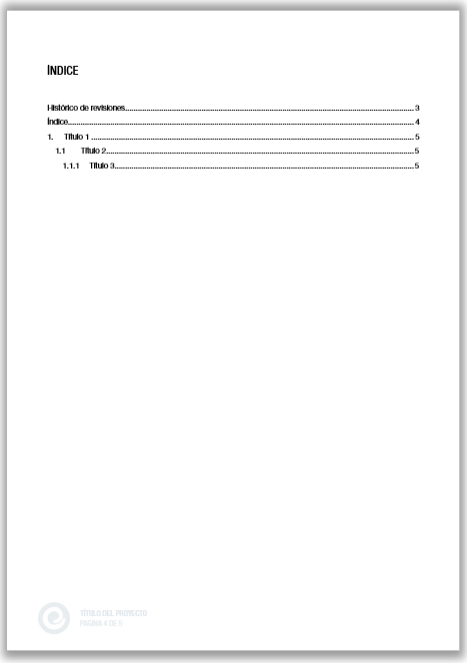
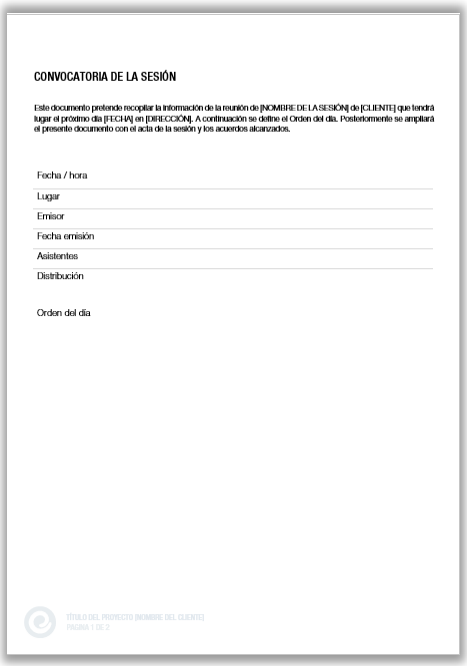


Figura 5. Índice

3.2 Plantilla de acta de reunión

El formato de plantilla de acta de reunión es el siguiente (Figura 6 y Figura 7). En la primera hoja se incluye la fecha, hora y lugar de la reunión, la persona emisora del documento, la fecha de emisión del mismo, los asistentes, la distribución y el orden del día. En la segunda hoja se incluye el acta de la reunión, donde aparecen, divididos por secciones, los distintos acuerdos alcanzados, la persona responsable de los mismos, la fecha de compromiso y el estado, expresado en cifra porcentual.



CONVOCATORIA DE LA SESIÓN

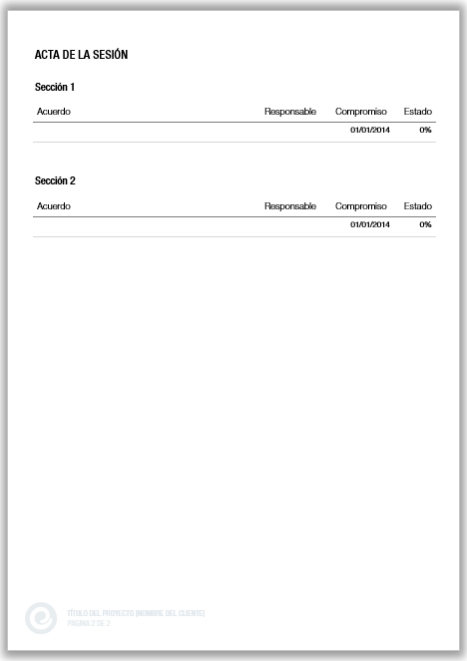
Este documento pretende recopilar la información de la reunión de [NOMBRE DE LA SESIÓN] de [CLIENTE] que tendrá lugar el próximo día [FECHA] en [DIRECCIÓN]. A continuación se detine el Orden del día. Posteriormente se ampliará el presente documento con el acta de la sesión y los acuerdos alcanzados.

Fecha / hora	
Lugar	
Emisor	
Fecha emisión	
Asistentes	
Distribución	

Orden del día

TÍTULO DEL PROYECTO [NOMBRE DEL CLIENTE]
PÁGINA 1 DE 2

Figura 6. Acta de reunión, hoja 1



ACTA DE LA SESIÓN

Sección 1

Acuerdo	Responsable	Compromiso	Estado
		01/01/2014	0%

Sección 2

Acuerdo	Responsable	Compromiso	Estado
		01/01/2014	0%

TÍTULO DEL PROYECTO [NOMBRE DEL CLIENTE]
PÁGINA 2 DE 2

Figura 7. Acta de reunión, hoja 2

4. BIBLIOGRAFÍA

4.1 Referencias

[R1] Edward J Huth *Scientific Style and Format: The CBE Manual for Authors, Editors, and Publishers* Cambridge University Press 1994

[R2] Estándar de documentación de Endalia S.L.

4.2 Referencias web

[W1] http://www.wikipedia.org

[W2] http://www.monografias.com/trabajos6/dosi/dosi.shtml

[W3] http://www.apa.org/journals/webref.html

endalia®
HR Consulting & Software

ESTÁNDAR DE CODIFICACIÓN

DESARROLLO DE UN MÓDULO DE GESTIÓN DE VIAJES Y GASTOS

VERSIÓN 2.0
PUBLICADO EL 29/08/2016

HISTÓRICO DE REVISIONES

Fecha	Versión	Descripción	Autor
03/09/2007	1.0	Redacción del Estándar de Documentación de Endalia	Endalia
23/08/2016	2.0	Modificaciones del documento para adaptarlo al TFG	Nicolás Bailo Ortiz

ÍNDICE

1. INTRODUCCIÓN

1.1 Propósito del documento

El presente documento describe las normas que deben seguirse en el desarrollo de cualquier tipo de elemento de codificación realizado en este trabajo. El objetivo primordial de la confección de un estándar de codificación se basa en homogeneizar el proceso de implementación y codificación, para lograr obtener beneficios en la comprensión del código, en la verificación y validación del mismo, y en posibles modificaciones posteriores.

Existen un gran número de razones por las que definir las convenciones de código es importante para los programadores:

- El 80% del coste del código de un programa se invierte en su mantenimiento.
- Casi ningún software lo mantiene toda su vida el autor original.
- Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo.
- La distribución de código fuente como producto exige su presentación de manera adecuada.

De este modo, el presente documento pretende ser una colección de reglas que deben aplicarse a todo el código generado, con el propósito de que sea homogéneo. Esta homogeneidad permitirá una comprensión más efectiva del código tanto para su autor como para otros programadores, facilitando su distribución y mantenimiento.

1.2 Alcance del documento

Este documento se ubica dentro de la fase inicial de desarrollo del módulo de gestión de viajes y gastos, como elemento necesario previo a la realización de cualquier tipo de código fuente del proyecto, y será utilizado como guía durante toda la fase de implementación.

1.3 Acrónimos

- ANSI: American National Standards Institute.
- HTML: Hypertext Markup Language.
- ID: Identificador.
- URL: Uniform Resource Locator.

1.4 Definiciones

- Pascal-Casing: Notación en la que un identificador está compuesto por múltiples palabras juntas comenzando cada una de ellas por una letra mayúscula.
- Camel-Casing: Notación similar a Pascal-Casing con la excepción de que la letra inicial del identificador debe ser minúscula.

1.5 Referencias

En este documento no se realizan referencias a otros documentos del trabajo.

1.6 Resumen

El presente documento describe las normas que deben seguirse en el desarrollo de cualquier tipo de elemento de codificación realizada en este proyecto. Se compone de 5 apartados:

1. Se muestra el propósito del documento y se define su alcance. Se proporciona una lista de acrónimos y definiciones útiles para la comprensión del documento, así como una lista de los documentos del proyecto referenciados y el presente resumen.
2. Se muestra la estructura de codificación de los diferentes tipos de archivos de código fuente desarrollados en el trabajo.
3. Se muestran reglas, recomendaciones y buenas prácticas para el desarrollo del código fuente del trabajo.
4. Se muestra el estándar de nombrado de los elementos de base de datos.
5. Bibliografía y referencias Web utilizadas en la confección de este documento.

2. ESTRUCTURA DE LOS FICHEROS

2.1 Introducción

En este apartado se define la estructura y organización de los archivos de código fuente del proyecto. Se especifica la codificación tanto de los archivos de clases, acceso a datos y servicios web (.cs) como de los archivos de interfaz de usuario (.aspx) y los archivos de código subyacente (.aspx.cs).

2.2 Codificación de archivos de interfaz de usuario .aspx

A continuación se muestra la estructura de codificación de los archivos de interfaz de usuario. Para una comprensión más sencilla se muestra mediante una tabla con dos columnas. En la columna de la izquierda aparece un índice para facilitar la posterior descripción de la región de la estructura definida en la columna de la derecha.

<1>Declaración de página	`<%@ Page Language="C#"`
	`AutoEventWireup="true"`
	`Inherits="myNamespace"`
	`MasterPageFile="myMaster"`
	`CodeBehind="myClass" %>`

<2>Importación *de controles*	`<%@ Register TagPrefix="myControlPrefix"`
	`TagName="myControl"`
	`Src="myControlSrc" %>`

<3>Definición de elementos de la página	`<Código de definición de elementos HTML>`

2.2.1 Declaración de página

En esta región se define la página o en su caso el control que representa el archivo. Es en esta región donde se enlaza la página con el código subyacente y con un fichero Master (en caso de definir una página y no un control).

2.2.2 Importación de controles

En esta región se definen los controles que se utilizarán a lo largo de la página. Es necesario referenciarlos mediante un prefijo y un nombre, así como indicar en qué ruta se encuentra su código fuente.

2.2.3 Definición de elementos de la página

En esta región se define la estructura de la página o del control que representa el archivo, mediante el uso de elementos tanto HTML como del framework de ASP.NET.

2.3 Codificación de archivos de código subyacente .aspx.cs

A continuación se muestra la estructura de codificación de los archivos de código subyacente que será comentada posteriormente. De la misma manera que en el apartado anterior, se muestra mediante una tabla con dos columnas. En la columna de la izquierda aparece un índice para facilitar la posterior descripción de la región de la estructura definida en la columna de la derecha.

<1>Directivas *using*	`using System;` `<directivas using>`

<2>Declaración *namespace* y *class*	`namespace MyNamespace1`
	`{`
	`public class MyClass`
	`{`

<3> Región variables	`#region variables`
	`#region Variables I18N`
	`<declaraciones de variables de internacionalización>`
	`#endregion Variables I18N`
	`#region Variables globales`
	`<declaraciones de variables globales>`
	`#endregion Variables globales`
	`#endregion variables`

<4> Región *Page_Load*	`#region Page_Load`
	`<Código Page_Load>`
	`#endregion Page_load`

<5> Región *I18N*	`#region I18N`
	`<Código LoadI18N>`
	`<Código class_Init>`
	`#endregion I18N`

<6> Región *DataBind*	`#region DataBind`
	`<Código DataBind>`
	`#endregion DataBind`

<7> Región Eventos	`#region Eventos`
	`<Código Eventos provenientes de controles>`
	`#endregion Eventos`

<8> Región Eventos de botones	`#region Eventos de botones`
	`<Código Eventos clic botones>`
	`#endregion Eventos de botones`

<9> Región Métodos de ordenación	#region Métodos de ordenación <Código Métodos de ordenación> #endregion Métodos de ordenación
<10> Región Código generado *Web Forms*	#region Código generado por el diseñador de Web Forms #endregion Código generado por el diseñador de Web Forms
	} }

2.3.1 Consideraciones generales sobre las regiones

Como se ha visto en el apartado 2.3 y para facilitar la organización y estructuración del código fuente se utilizan las directivas #region y #endregion. Las regiones no aportan funcionalidad como tal, se utilizan para marcar y agrupar una sección del código. Las regiones que se especifican en el apartado 2.3 son las obligatorias en el caso de que aparezcan los elementos para los que han sido definidas. En caso de que aparezcan elementos no especificados en las regiones del apartado anterior podrán definirse nuevas regiones para especificar la sección de código referida a los eventos y métodos del elemento o control. En cualquier caso, no se permitirán eventos o métodos que no estén incluidos dentro de alguna región.

2.3.2 Región directivas *using*

En esta región se colocarán, por orden alfabético creciente, las directivas que especifiquen las clases utilizadas en el código fuente definido en la clase actual.

2.3.3 Región declaración *namespace* y *class*

En esta región se colocarán las cabeceras que especifican el espacio de nombres en los que se integra el código y el nombre de la clase. Esta última irá precedida por una cabecera en la que se especificarán los siguientes datos:

```
/// <summary>
/// Nombre del fichero : Nombre del fichero
/// Autor              : Nombre del autor
/// Descripción        : Descripción de la funcionalidad y objetivo del fichero
/// </summary>
```

2.3.4 Región declaración de variables

Esta región estará integrada por tres subregiones que se especifican a continuación:

- Región variables *I18N*: en esta región aparecen las declaraciones de cadenas de internacionalización que son obtenidas del archivo de recursos y que se utilizan para definir todos los textos que son presentados al usuario.
- Región variables globales: en esta región aparecen las declaraciones de variables globales utilizadas acompañadas de una descripción de su funcionalidad.

2.3.5 Región *Page Load*

En esta región se coloca el código del evento *Page_Load* que se lanza cada vez que la página es lanzada o recargada.

2.3.6 Región *I18N*

Esta región está integrada por dos métodos:

- Método *Init()*: realiza la lectura del fichero de recursos en el que se almacenan las cadenas de internacionalización, cargando estas en variables de cadena.
- Método *LoadI18N()*: carga en los campos de texto del aspx las variables de cadenas obtenidas en el método *Init*.

2.3.7 Región *DataBind*

En esta región se colocan los métodos que enlazan orígenes de datos a controles de servidor como *DataGrids* o árboles.

2.3.8 Región Eventos

Se utiliza está región para definir para cada uno de los métodos que capturan los diversos eventos de diversos controles y que se especifican en la tabla del apartado 2.2.

2.3.9 Región Eventos de botones

En esta región se colocan los métodos que capturan los eventos clic de los distintos botones ubicados en el archivo aspx.

2.3.10 Región Métodos de ordenación

Esta región engloba los métodos que ordenan los diferentes orígenes de datos que son utilizados en el código.

2.3.11 Región Código generado por el diseñador de *Web Forms*

Esta región se genera automáticamente gracias a la herramienta de desarrollo, y consta de dos métodos:

- *InitializeComponent()*: contiene las declaraciones de los métodos del código que capturan los diferentes eventos producidos por los controles ubicados en el código aspx.
- *OnInit()*: este método se lanza al cargar la página y realiza la llamada al método *InitializeComponent* para comenzar la ejecución del código del servidor.

2.4 Codificación de archivos de clase, acceso a datos y servicios web .cs

A continuación se muestra la estructura de codificación de los archivos de definición de clases, acceso a datos y servicios web. De la misma manera que en los apartados anteriores, se muestra mediante una tabla con dos columnas. En la columna de la izquierda aparece un índice para facilitar la posterior descripción de la región de la estructura definida en la columna de la derecha.

<1>Directivas *using*	`using System;` `<directivas using>`

<2>Declaración *namespace* y *class*	`namespace MyNamespace1`
	`{`
	`public class MyClass`
	`{`

<3> Región atributos	`#region atributos`
	`<Declaración atributos>`
	`#endregion atributos`

<4> Región propiedades	`#region propiedades`
	`<declaración propiedades>`
	`#endregion propiedades`

<5> Región constructores	`#region constructores`
	`<Métodos constructores clase>`
	`#endregion constructores`

<6> Región Métodos	`#region métodos`
	`<Código Métodos>`
	`#endregion métodos`

	`}` `}`

2.4.1 Región directivas *using*

En esta región se colocarán, por orden alfabético creciente, las directivas que especifiquen las clases utilizadas en el código fuente definido en la clase actual.

2.4.2 Región declaración *namespace* y *class*

En esta región se colocarán las cabeceras que especifican el espacio de nombres en los que se integra el código y el nombre de la clase. Esta última irá precedida por una cabecera en la que se especificarán los siguientes datos:

```
/// <summary>
/// Nombre del fichero : Nombre del fichero
/// Autor              : Nombre del autor
/// Descripción        : Descripción de la funcionalidad y objetivo del fichero
/// </summary>
```

2.4.3 Región atributos

En esta región se colocará la declaración de los atributos de una clase. Los atributos se nombrarán mediante Pascal-Casing exceptuando la primera letra, que será en minúscula y precedida por un guion bajo de este modo:

```
private int _requestID;
```

Asimismo en esta región se declararán las constantes de la clase, que se nombrarán con mayúsculas.

2.4.4 Región propiedades

En esta región se coloca la declaración de las propiedades públicas de la clase. Las propiedades se nombran con Pascal-Casing y, en el caso de que representen el acceso al valor de un atributo de la clase, su nombre es el mismo del atributo sin el guión bajo y con la primera letra en mayúscula, especificando el acceso a los métodos *get* y *set* de este modo:

```
public int RequestID
{
       get{ return _requestID; }
       set{ _requestID = value; }
}
```

2.4.5 Región constructores

En esta región se colocará la declaración de los métodos constructores de la clase.

2.4.6 Región métodos

En esta región se colocará la declaración de los métodos de la clase.

2.5 Codificación de archivos de recursos de internacionalización

Los archivos .txt a partir de los cuales se generan los archivos de recursos de internacionalización se construirán del siguiente modo:

Para cada una de las secciones del programa que tengan una entidad lo suficientemente importante como para ser diferenciada, se colocará un comentario y a continuación la relación de las etiquetas. El nombrado de las etiquetas se hace del siguiente modo:

- La parte inicial del nombre de la etiqueta será la misma que el nombre del archivo .aspx en el que se utilizará la etiqueta. A continuación se colocará un guion bajo seguido de un prefijo que indicará la utilización de la etiqueta seguida de un guion bajo, siguiendo la siguiente convención:
    - etiqueta o campo de texto: `_lbl_`
    - etiqueta de *hyperlink*: `_lnk_`
    - texto de botón: `_btn_`
- En el caso de que la etiqueta sea un *tooltip,* se colocará a continuación el sufijo `_ToolTip`
- A continuación se colocará un nombre descriptivo de la función de la etiqueta que utilizará Pascal-Casing. No se especifica una norma rígida para este nombrado pero a continuación se muestran unos ejemplos que muestran buenas prácticas del mismo.

```
MyTrips_lbl_lblCost
MyTrips_btn_btnSend
MyTrips_btn_btnSend_ToolTip
```

3. REGLAS DE CODIFICACIÓN

3.1 Identación

Dada la actual uniformidad y estandarización de los editores utilizados para el desarrollo de código C# en .NET, se utilizará el tabulador como unidad de identación estándar. Los comentarios se identarán al mismo nivel de identación que el código que se esté documentando.

3.1.1 Longitud de línea

Se recomienda no escribir líneas con más de 80 caracteres, ya que no son bien manejadas por muchos terminales y herramientas.

3.1.2 Ruptura de líneas

Cuando una expresión no entre en una sola línea, se debe romper de acuerdo a estos principios generales:

- Romper después de una coma.
- Romper antes de un operador.
- Preferir las rupturas de alto nivel a las de bajo nivel.
- Alinear la nueva línea con el principio de la expresión al mismo nivel de la línea anterior.

3.2 Comentarios

En C# hay tres formas de escribir comentarios:

- La primera consiste en encerrar todo el texto que se desee comentar entre caracteres /* y */ siguiendo la siguiente sintaxis:

```
/*<texto>*/
```

    - Estos comentarios pueden abarcar tantas líneas como sea necesario. No es posible anidar comentarios de este tipo.

- En la segunda se considera como indicador del comienzo del comentario la pareja de caracteres // y como indicador de su final el fin de línea. Por tanto, la sintaxis que siguen estos comentarios es:

```
// <texto>
```

- La tercera manera es utilizando el trío de caracteres ///. Este tipo de comentario tiene la particularidad de ser reconocido y utilizado por las herramientas de generación automática de documentación y será el utilizado para la descripción de métodos y clases, ya que en el caso de los primeros genera la estructura de etiquetas o *tags* de documentación de nombres, parámetros y valores de retorno utilizados para la documentación automatizada.

```
/// <texto>
```

3.2.1 Aplicación de los comentarios

Como se ha comentado en el punto anterior, el tercer tipo de comentario (el que va precedido de los caracteres ///) se utiliza para crear de manera automática las etiquetas que permiten la generación automática de documentación.

Aparte de este punto, los comentarios deberían usarse para una introducción del código y proporcionar información adicional que no está disponible en el propio código. Los comentarios sólo deberían tener información que sea relevante para leer y entender el programa. Por ejemplo, información sobre cómo está construida la clase correspondiente o en qué directorio reside no debería ser incluida como comentarios.

Las discusiones no triviales o decisiones de diseño no obvias son apropiadas, pero debemos evitar la duplicidad de información que esté presente en el código. Es demasiado fácil que los comentarios redundantes se queden anticuados. En general, debemos evitar cualquier comentario que se pueda quedar anticuado cuando el código evolucione.

La frecuencia en los comentarios algunas veces refleja una pobre calidad de código. Cuando nos sintamos obligados a llenarlo de comentarios, debemos considerar la reescritura del código para hacerlo más claro. Los comentarios no deben encerrarse en grandes cajas dibujadas con asteriscos u otros caracteres. Los comentarios nunca deberían incluir caracteres especiales como saltos de página, etc.

Los siguientes puntos son técnicas de comentarios recomendadas.

- Cuando se modifica el código, se mantienen siempre actualizados los comentarios circundantes.
- Evitar los comentarios recargados, como las líneas enteras de asteriscos. En su lugar se utilizan espacios para separar los comentarios y el código.
- Evitar rodear un bloque de comentarios con un marco tipográfico. Puede resultar agradable, pero es difícil de mantener.
- Antes de la implementación, quitar todos los comentarios temporales o innecesarios, para evitar cualquier confusión en la futura fase de mantenimiento.
- Si se necesita realizar comentarios para explicar una sección de código compleja, examinar el código para decidir si se debería volver a escribir. Siempre que sea posible, no documentar un código malo, sino volver a escribirlo. Aunque, por regla general, no debe sacrificarse el rendimiento para hacer un código más simple para el usuario, es indispensable un equilibrio entre rendimiento y mantenibilidad.
- Usar frases completas al escribir comentarios. Los comentarios deben aclarar el código, no añadirle ambigüedad.
- Ir comentando al mismo tiempo que se programa, porque probablemente no habrá tiempo de hacerlo más tarde. Por otro lado, aunque se tuviera oportunidad de revisar el código que se ha escrito, lo que parece obvio hoy es posible que seis semanas después no lo sea.
- Evitar comentarios superfluos o inapropiados, como comentarios divertidos al margen.
- Usar los comentarios para explicar el propósito del código como si fueran traducciones interlineales.
- Comentar cualquier cosa que no sea legible de forma obvia en el código.
- Para evitar problemas recurrentes, hacer siempre comentarios al depurar errores y solucionar problemas de codificación, especialmente cuando se trabaje en equipo.
- Hacer comentarios en el código que esté formado por bucles o bifurcaciones lógicas. Se trata en estos casos de áreas clave que ayudarán a los lectores del código fuente.
- Realizar los comentarios en un estilo uniforme, respetando una puntuación y estructura coherentes a lo largo de toda la aplicación.
- Separar los comentarios de sus delimitadores mediante espacios. Si se respeta esta norma, los comentarios serán más claros y fáciles de localizar si trabaja sin indicaciones de color.

3.2.2 Formatos de implementación de comentarios

Los programas pueden tener cuatro estilos de implementación de comentarios:

- Bloque de comentarios: Los bloques de comentarios se usan para proporcionar descripciones de ficheros, métodos, estructuras de datos y algoritmos. Los bloques de comentarios podrían usarse al principio de cada fichero y antes de cada método. También pueden usarse en otros lugares, como dentro de los métodos. Para este tipo de comentario se preferirá la estructura /* - */. Un bloque de comentario debería ir precedido por una línea en blanco para configurar un apartado del resto del código:

```
/*
 * Esto es un bloque de comentarios.
 */
```

- Comentarios de una línea: Los comentarios cortos pueden aparecer como una sola línea identada al nivel del código que la sigue. Si un comentario no se puede escribir en una sola línea, debería seguir el formato de los bloques de comentario. Un comentario de una sola línea debería ir precedido de una sola línea en blanco. Para este tipo de comentario se preferirá utilizar los caracteres '//'. A continuación se muestra un ejemplo:

```
if (condition)
{
        // Código de la condición.
        ...
}
```

- Comentarios finales: Los comentarios muy cortos pueden aparecer en la misma línea que el código que describen, pero deberían separarse lo suficiente de las sentencias. Si aparece más de un comentario en el mismo trozo de código, deberían estar identados a la misma altura. Para este tipo de comentario se preferirá utilizar los caracteres '//'. Aquí tenemos un ejemplo utilizando estos caracteres y la estructura /* - */:

```
if (a == 2)
{
        return TRUE;                    // caso especial
}
else
{
         return isPrime(a);             /* otro comentario */
}
```

3.3 Declaraciones

3.3.1 Número de declaraciones por línea

Se recomienda una declaración por línea ya que mejora los comentarios. En otras palabras:

```
int level;       // nivel de identación
int size;        // tamaño
```

se prefiere sobre:

```
int level, size;
```

No debemos poner diferentes tipos en la misma línea. Por ejemplo:

```
int foo, fooarray[]; //Evitar
```

3.3.2 Inicialización

Debemos intentar inicializar las variables locales donde son declaradas. La única razón para no inicializar una variable donde es declarada es si el valor inicial depende de algún cálculo que tiene que ocurrir antes.

3.3.3 Situación

Ponemos las declaraciones sólo al principio de los bloques. No debemos esperar a declarar variables hasta que son usadas por primera vez; puede confundir al programador y estorbar la portabilidad del código dentro del ámbito.

```
void myMethod()
{
  int int1 = 0;                    // comienzo de bloque

  if (condition)
  {
        int int2 = 0;              // comienzo de bloque if
        ...
  }
}
```

La única excepción a esta regla son los indexados para los bucles, que en C# pueden ser declarados en la sentencia for:

```
for (int i = 0; i < maxLoops; i++) { ... }
```

Debemos evitar las declaraciones locales que oculten las declaraciones de nivel superior. Por ejemplo, no debemos declarar el mismo nombre de variable en un bloque interno:

```
int count;

myMethod() {

  if (condition) {

    int count; // Evitar

  }

}
```

3.4 Sentencias

3.4.1 Sentencias simples

Cada línea debe contener, como máximo, una sentencia. Por ejemplo:

```
argv++;          // Correcto
argc++;          // Correcto
argv++; argc--;  // Evitar
```

3.4.2 Sentencias compuestas

Las sentencias compuestas son sentencias que contienen listas de sentencias encerradas entre llaves (“{sentencias}”). Se ilustrarán ejemplos en las siguientes secciones.

- Las sentencias encerradas deben identarse uno o más niveles que la sentencia compuesta.
- La llave (‘{‘) de apertura debe empezar una nueva línea a continuación de la que empieza la sentencia compuesta; la llave de cierre (‘}’) debe empezar una nueva línea y estar identada con el principio de la sentencia compuesta.
- Las llaves se usan alrededor de todas las sentencias, incluso para sentencias simples, cuando éstas forman parte de una estructura de control como una sentencia if-else o for. Esto hace más fácil la adición de sentencias sin introducir errores debido al olvido de las llaves.

Las únicas excepciones a esta regla serán para los métodos get y set de las clases de acceso a datos descritos en el apartado 2.4.4.

3.4.3 Sentencias de retorno

Una sentencia de retorno no deberá usar paréntesis a menos que el valor de retorno sea más obvio de esta forma. Por ejemplo:

```
return;
return myDisk.size();
return (size ? size : defaultSize);
```

3.4.4 Sentencias *if, if-else, if else- if else*

Las sentencias de tipo *if-else* deberán tener la siguiente forma:

```
if (condition)
{
  statements;
}

if (condition)
{
  statements;
}
else
{
  statements;
}


if (condition)
{
  statements;
}
else if (condition)
{
  statements;
}
else
{
  statements;
}
```

Las sentencias *if* siempre usan llaves. Debemos evitar el siguiente caso:

```
if (condition) //Evitar, se han omitido las llaves {}!
  statement;
```

Aunque es perfectamente válido a nivel de código, puede llevar a confusión y a la introducción de errores.

3.4.5 Sentencias *for*

Una sentencia *for* deberá tener la siguiente forma:

```
for (initialization; condition; update)
{
  statements;
}
```

Una sentencia *for* vacía, en la cual todo el trabajo se hace en las cláusulas de inicialización, condición y actualización, deberá tener la siguiente forma:

```
for (initialization; condition; update);
```

Cuando usamos el operador como en las cláusulas de inicialización o actualización de una sentencia *for*, debemos evitar la complejidad de usar más de tres variables. Si es necesario, debemos usar sentencias separadas antes del bucle *for*, para la cláusula de inicialización, o al final del bucle, para la cláusula de actualización.

3.4.6 Sentencias *while*

Una sentencia *while* deberá tener la siguiente forma:

```
while (condition)
{
    statements;
}
```

Una sentencia *while* vacía deberá tener la siguiente forma:

```
while (condition);
```

3.4.7 Sentencias *do-while*

Una sentencia *do-while* deberá tener la siguiente forma:

```
do
{
  statements;
}
while (condition);
```

3.4.8 Sentencias *switch*

Una sentencia *switch* deberá tener la siguiente forma:

```
switch (expression)
{
      case constant-expression:
            statement
            break;

      case constant-expression:
           statement
           /* continúa sin salto */

            [default:
                  statement
                   jump-statement]
}
```

Cada vez que un *case* no incluye una sentencia *break*, debemos añadir un comentario donde normalmente iría la sentencia *break*. Esto se ve en el ejemplo de código anterior con el comentario "/* continúa sin salto */". En cualquier caso se deberá evitar este tipo de construcción.

Toda sentencia *switch* deberá incluir un valor *default*. El *break* en el *case* por defecto es redundante, pero evita un error de caída si añadimos después otro *case*.

3.4.9 Sentencias *try-catch*

Una sentencia *try-catch* deberá tener la siguiente forma:

```
try
{
        statements;
}
catch (Exception e)
{
        statements;
}
```

Una sentencia *try-catch* también puede ir seguida de un bloque *finally*, que se ejecuta sin importar si se ha completado con éxito o no el bloque try:

```
try
{
        statements;
}
catch (Exception e)
{
        statements;
}
finally
{
        statements;
}
```

3.5 Espacios en blanco

3.5.1 Líneas en blanco

Las líneas en blanco mejoran la lectura separando secciones de código que están relacionadas lógicamente.

Siempre se deberán usar dos líneas en blanco en las siguientes circunstancias:

- Entre secciones de un fichero fuente.
- Entre definiciones de clases e interfaces.

Siempre se deberá usar una línea en blanco en las siguientes circunstancias:

- Entre métodos.
- Entre las variables locales de un método y su primera sentencia.
- Antes de un bloque de comentarios o un comentario simple.
- Entre secciones lógicas dentro de un método para mejorar su lectura.

3.5.2 Espacios en blanco

Los espacios en blanco deberán usarse en las siguientes circunstancias:

- Una palabra clave seguida por un paréntesis deberían estar separados por un espacio en blanco:

```
while (true)
{
        ...
}
```

- No se deberá usar un espacio en blanco entre un nombre de método y su paréntesis de apertura. Esto ayuda a distinguir las palabras clave de las llamadas a métodos.
- Después de las comas en una lista de argumentos debe aparecer un espacio en blanco.
- Todos los operadores binarios excepto "." Deberían estar separados de sus operandos por espacios. Los espacios en blanco nunca deben separar los operadores unarios como incremento ("++"), y decremento ("—") de sus operadores. Por ejemplo:

```
a += c + d;
a = (a + b) / (c * d);
while (d++ = s++)
{
   n++;
}
```

- Las expresiones de una sentencia deberán estar separadas por espacio. Ejemplo:

```
for (expr1; expr2; expr3)
```

3.6 Convenciones de nombres

3.6.1 Clases

Se deberá usar Pascal-Casing para el nombrado de clases. Debemos intentar mantener los nombres de clases simples y descriptivos. Debemos usar palabras completas y evitar acrónimos y abreviaturas (a menos que la abreviatura se use muy ampliamente como URL o HTML). Ejemplo:

```
public class HelloWorld
{
      ...
}
```

3.6.2 Métodos

Los métodos deberán ser verbos. Se deberá usar Pascal-Casing para su nombrado. Ejemplo:

```
public class HelloWorld
{
      void SayHello(string name)
      {
            ...
      }
}
```

3.6.3 Variables y parámetros

Se usará Camel-Casing para el nombrado de variables y parámetros. Los nombres de variables deberán ser cortos y llenos de significado. La elección de una variable debería ser mnemónica, es decir, diseñada para indicar al observador casual su utilización. Se deben evitar los nombres de variable de un sólo carácter, excepto para variables temporales. Algunos nombres comunes de este tipo de variables son: i, j, k, m, y n para enteros. Ejemplo:

```
public class HelloWorld
{
      int totalCount = 0;
      void SayHello(string name)
```

```
        {
                string fullMessage = "Hello " + name;
        }
}
```

3.6.4 Constantes

Los nombres de variables constantes de clases y las constantes ANSI deberán escribirse todo en mayúsculas con las palabras separadas por subrayados (“_”). Se deberían evitar las constantes ANSI para facilitar la depuración. Ejemplo:

```
public static int MIN_WIDTH = 4;
public static int MAX_WIDTH = 999;
public static int GET_THE_CPU = 1;
```

3.7 Hábitos de programación

No debemos hacer pública ninguna variable de instancia o de clase sin una buena razón. A menudo las variables de instancia no necesitan ser asignadas/consultadas explícitamente. Normalmente, esto sucede como efecto lateral de llamadas a métodos. Un ejemplo apropiado de una variable de instancia pública es el caso en que la clase es esencialmente una estructura de datos, sin comportamiento.

3.7.1 Referencias a variables y métodos de clase

Evitar usar un objeto para acceder a una variable o método de clase (*static*). Usar el nombre de la clase en su lugar. Por ejemplo:

```
classMethod();            //correcto
AClass.classMethod();     //correcto
anObject.classMethod();   //Evitar
```

3.7.2 Constantes

Las constantes numéricas (literales) no se deben codificar directamente, excepto -1, 0 y 1, que pueden aparecer en un bucle *for* como contadores.

3.7.3 Asignaciones de variables

Evitar asignar el mismo valor a varias variables en la misma sentencia. Es difícil de leer. Ejemplo:

```
fooBar.fChar = barFoo.lchar = 'c'; // ¡EVITAR!
```

No usar el operador de asignación en un lugar donde se pueda confundir con el de igualdad. Ejemplo:

```
if (c++ = d++) // ¡EVITAR!
{
  ...
}
```

se debe escribir:

```
if ((c++ = d++) != 0)
{
  ...
```

```
}
```

No debemos usar asignaciones embebidas como un intento de mejorar el rendimiento en tiempo de ejecución. Ese es el trabajo del compilador. Ejemplo:

```
d = (a = b + c) + r; // ¡EVITAR!
```

Debería escribirse como:

```
a = b + c;
d = a + r;
```

3.7.4 Paréntesis

En general es una buena idea usar paréntesis en expresiones que implican distintos operadores para evitar problemas con el orden de precedencia de los operadores. Incluso si parece claro el orden de precedencia de los operadores, podría no ser así para otros. No se debe asumir que otros programadores conozcan el orden de precedencia.

```
if (a == b && c == d)      // Evitar
if ((a == b) && (c == d)) // Correcto
```

3.7.5 Variables de retorno

Debemos intentar hacer que la estructura de nuestro programa se corresponda con nuestra intención. Por ejemplo:

```
if ( booleanExpression)
{
      return true;
}
else
{
      return false;
}
```

debería escribirse:

```
return booleanExpression;
```

De forma similar,

```
if (condition)
{
      return x;
}
return y;
```

debería escribirse como:

```
  return (condition ? x : y);
```

3.7.6 Expresiones antes de '?' en el operador condicional

Si una expresión contiene un operador binario antes de ? en el operador ternario ?:, se debe colocar entre paréntesis. Ejemplo:

```
(x >= 0) ? x : -x;
```

3.7.7 Comentarios especiales

Debemos usar XXX, en un comentario para indicar que algo tiene algún error pero funciona. Usar la etiqueta *FIXME* para marcar algo que tiene algún error y no funciona.

4. ESTÁNDAR DE NOMBRADO DE BASE DE DATOS

4.1 Idioma a utilizar

Para el nombrado de todos los elementos de la base de datos, el idioma a utilizar será el inglés**,** salvo que se especifique de manera explícita lo contrario**.**

4.2 Convenciones de nombrado de tablas

4.2.1 Nombrado de tablas de entidad

El nombrado de las tablas de entidad dentro de la base de datos seguirá la siguiente codificación:

- Todos los nombres comenzarán por tres letras descriptivas del módulo o ámbito de aplicación de la tabla. La primera de estas tres letras será mayúscula y las otras dos restantes serán minúsculas. Los prefijos generales predefinidos con una descripción de los mismos se encuentran en el apartado 4.2.3 de este documento. En el caso de que el ámbito de aplicación de la tabla a nombrar difiera de los que aparecen en este apartado se definirá un nuevo prefijo y se añadirá a los existentes.
- En el caso de que la tabla forme parte del desarrollo de una sección en concreto dentro de un módulo, o forme parte de un conjunto de tablas relacionadas entre sí dentro de un ámbito de aplicación de la base de datos, se colocarán después del prefijo anterior y separadas por un guion bajo '_' tres letras descriptivas de la sección. La primera de estas letras será mayúscula y las otras dos restantes serán minúsculas. Si la tabla no tuviese relación con otras dentro del ámbito definido por el prefijo inicial estas tres letras no son necesarias.
- A continuación y separado por un guion bajo '_' se colocará un nombre descriptivo en plural del contenido de la tabla. Se deberá usar Pascal-Casing para este el nombrado. Se deberá intentar mantener los nombres simples y descriptivos. Se deberán usar palabras completas y evitar acrónimos y abreviaturas (a menos que la abreviatura se use muy ampliamente como ID, URL o HTML, en cuyo caso puede escribirse en mayúsculas).

  Ejemplos:

```
Gen_AdministrationItems
Orh_Emp_Employees
Orh_Trip_TripExpenses
```

4.2.2 Nombrado de tablas de relación

Las tablas de relación comenzarán con un prefijo 'R_'.

A continuación, y en singular, se colocarán utilizando Pascal-Casing el nombre descriptivo de las tablas de entidad que relaciona, pudiendo reducirse si es muy largo alguno de ellos.

De este modo, para relacionar las tablas 'Orh_Trip_TripRequests' y 'Orh_Emp_Employees' se creará la tabla 'R_TripRequest_Employee'.

4.2.3 Nombres de tablas predefinidos

- R: Relación. Tablas de relación.
- Gen: Generic. Tablas relativas a la organización general de la base de datos.

- Orh: Organización y recursos humanos.
- Trip: Trips. Tablas relativas a viajes.

4.3 Convenciones de nombrado de campos

Una norma establecida es que no puede haber dos campos dentro de una misma base de datos con el mismo nombre. Asimismo, el orden de los campos de una tabla debe seguir un orden lógico, (no tiene sentido colocar, <nombre, teléfono, apellidos, fax...>, sino <nombre, apellidos, teléfono, fax…>) Los nombres de los campos, dependiendo de si pertenecen a una tabla de entidad o de relación, se construyen siguiendo las normas de los siguientes apartados:

4.3.1 Nombrado de Campos de Tablas de Entidad

- Nombrado de claves primarias: Las claves primarias comenzaran con el prefijo 'pk_'. Como finalización del nombre tendrán la terminación 'ID'.
- Nombrado de claves ajenas: Las claves ajenas comenzaran con el prefijo 'fk_'. Si la clave ajena apunta a una clave primaria de otra tabla como finalización del nombre tendrá la terminación 'ID'.
- Nombrado de campos: Todos los campos de una misma tabla tendrán un mismo prefijo que resumirá el nombre descriptivo de la tabla. La primera de estas letras será mayúscula y las restantes serán minúsculas. Por ejemplo, si la tabla es 'Orh_Trip_TripRequest' todos los campos de la tabla en este punto empezarían por 'TripRequest'. Otro ejemplo: Si la tabla es 'Orh_Emp_Employees' los campos de la tabla empezarían por 'Employee' o 'Emp'.

  A continuación se colocará el nombre del campo. Para nombrar los campos de la tabla se utilizarán nombres descriptivos de su significado. Se deberá usar Pascal-Casing para este nombrado. Se deberá intentar mantener los nombres simples y descriptivos. Se deberán usar palabras completas y evitar acrónimos y abreviaturas (a menos que la abreviatura se use muy ampliamente como ID, URL o HTML, en cuyo caso puede escribirse en mayúsculas).

  Ejemplos:

```
fk_EmpCountry
pk_EmpID
EmpName
EmpGender
```

4.3.2 Nombrado de campos de tablas de relación

- Nombrado de claves ajenas: Las claves ajenas comenzarán por 'fk_', a continuación aparecerán, utilizando Pascal-Casing, los nombres descriptivos de la tablas de entidad que se relacionan (pudiendo resumirse si son muy largos), y se finalizarán con el nombre descriptivo de la tabla a la que apuntan seguido de ID.

  Ejemplo:

  En la tabla 'R_Employee_Language' que relaciona las tablas 'Orh_Emp_Employees' y 'Orh_Emp_Languages' se nombrarán de esta manera las claves ajenas:

```
fk_EmpLanEmployeeID
fk_EmpLanLanguageID
```

- Nombrado de campos: Los campos en las tablas de relación se nombrarán mediante Pascal-Casing. Comenzaran con los nombres descriptivos de la tablas de entidad que se relacionan (pudiendo resumirse si son muy largos). A continuación se colocará el nombre del campo. Para nombrar los campos de la tabla se utilizarán nombres

descriptivos de su significado. Se deberá intentar mantener los nombres simples y descriptivos. Se deberán usar palabras completas y evitar acrónimos y abreviaturas (a menos que la abreviatura se use muy ampliamente como ID, URL o HTML, en cuyo caso puede escribirse en mayúsculas).

Ejemplos:

En la tabla 'R_Employee_Language' que relaciona las tablas 'Orh_Emp_Employees' y 'Orh_Emp_Languages' los nombres de los campos podrían ser:

```
EmpLanQualification
EmpLanIsDeleted
EmpLanLastUpdate
```

5. BIBLIOGRAFÍA

5.1 Referencias

[R1] Juval Lowy 2003. *C# Coding Standard*. Idesign Inc 2004.

5.2 Referencias web

[W1] http://www.dotnetspider.com /tutorials/bestpractices.aspx

[W2] http://msdn.microsoft.com/

[W3] http://vyaskn.tripod.com/object_naming.htm

endalia®
HR Consulting & Software

PLAN DE GESTIÓN DE CONFIGURACIONES

DESARROLLO DE UN MÓDULO DE GESTIÓN DE VIAJES Y GASTOS

VERSIÓN 2.0
PUBLICADO EL 29/08/2016

HISTÓRICO DE REVISIONES

Fecha	Versión	Descripción	Autor
04/09/2007	1.0	Redacción del Estándar de Documentación de Endalia	Endalia
23/08/2016	2.0	Modificaciones del documento para adaptarlo al TFG	Nicolás Bailo Ortiz

ÍNDICE

1. INTRODUCCIÓN

1.1 Propósito del documento

En el presente documento se definen las políticas, estándares, procedimientos y actividades asociadas a la gestión de configuraciones de este proyecto.

En el desarrollo de software son habituales los cambios en todos los elementos generados, generalmente debidos a modificaciones de especificación de requisitos y fallos de análisis o diseño. Es preciso llevar un control y registro de los cambios con el fin de reducir errores, aumentar la calidad y la productividad y evitar los problemas que puede acarrear una incorrecta sincronización en dichos cambios, al afectar a otros elementos del sistema o a las tareas realizadas por otros miembros del equipo de trabajo (si los hubiera).

En términos generales, la Gestión de Configuración del Software (GCS) se puede definir como una disciplina cuya misión es controlar la evolución de un sistema software.

Según Wayna A. Babich, una de las personas que más han publicado sobre este tema: “El arte de coordinar el desarrollo de software para minimizar la confusión, se denomina Gestión de Configuración Software. La GCS es el arte de identificar, organizar y controlar las modificaciones que sufre el software que construye un equipo de programación. El objetivo es maximizar la productividad minimizando los errores”.

El objetivo de la GCS es mantener la integridad de los productos que se obtienen a lo largo del desarrollo de los sistemas de información, garantizando que no se realizan cambios incontrolados y que todos los participantes en el desarrollo del sistema disponen de la versión adecuada de los productos que manejan.

La GCS se realiza durante todas las actividades asociadas al desarrollo del sistema, y continúa registrando los cambios hasta que éste deja de utilizarse.

De la misma forma, la GCS facilita el mantenimiento del sistema, aportando información precisa para valorar el impacto de los cambios solicitados y reduciendo el tiempo de implementación de un cambio, tanto evolutivo como correctivo. Asimismo, permite controlar el sistema como producto global a lo largo de su desarrollo, obtener informes sobre el estado de desarrollo en que se encuentra y reducir el número de errores de adaptación y/o integración del sistema, lo que se traduce en un aumento de calidad del producto, de la satisfacción del cliente y, en consecuencia, de la mejora de la organización.

Los productos registrados en el sistema de gestión de la configuración se encuentran identificados y localizados unívocamente, de manera que la información relativa a los productos es de fácil acceso.

Como resumen, se puede afirmar que la GCS es una disciplina de control, dentro del proceso de desarrollo del trabajo.

1.2 Alcance del documento

Las actividades de GCS se prolongan a lo largo del ciclo de vida del producto software.

1.3 Acrónimos

- ECS: Elemento de Configuración de Software.
- GC: Gestor de Configuraciones.
- GCS: Gestión de Configuraciones de Software.
- IEEE: Institute of Electrical and Electronic Engineers (Instituto de Ingenieros Eléctricos y Electrónicos).

- PGCS: Plan de Gestión de Configuraciones de Software.
- RUP: Rational Unified Process.
- TFS: Team Foundation Server.

1.4 Definiciones

- Artefacto: cualquier tipo de información creada, producida, cambiada o utilizada por el equipo de desarrollo del proyecto.
- Pascal-Casing: notación en la que un identificador está compuesto por múltiples palabras juntas comenzando cada una de ellas por una letra mayúscula.
- Camel-Casing: notación similar a Pascal-Casing con la excepción de que la letra inicial del identificador debe ser minúscula.
- Elemento de Configuración de Software: cualquier artefacto sujeto a todas las especificaciones estipuladas en el plan de gestión de configuraciones de software.
- Línea base: Punto de referencia en el proceso de desarrollo que queda marcado por la aprobación de uno o varios elementos de Configuración de Software, mediante una revisión técnica formal.
- Revisión: Instancia de un ECS, en un momento dado del proceso de desarrollo, que es almacenada en un repositorio, y que puede ser recuperada en cualquier momento para su uso o modificación.

1.5 Referencias

En este documento no se realizan referencias a otros documentos del proyecto.

1.6 Resumen

Este documento describe el plan de gestión de configuraciones de la AWSC. Se compone de cinco apartados:

1. Introducción del documento, definición del propósito y alcance del mismo.
2. Descripción de la organización de la gestión de configuraciones.
3. Especificación de las tareas a realizar para la gestión de configuraciones.
4. Descripción de las herramientas utilizadas para la gestión de configuraciones.

Bibliografía y referencias Web utilizadas para la realización de este documento.

2. ESPECIFICACIONES DE GESTIÓN

La gestión de configuraciones de software es el proceso de identificar y definir los elementos en el sistema, controlando el cambio de estos elementos a lo largo de su ciclo de vida, registrando e informando del estado de los elementos y las solicitudes de cambio, y verificando que los elementos estén completos y que sean los correctos.

El propósito de la Gestión de Configuración del Software es establecer y mantener la integridad de los productos de software durante su ciclo de vida, para lo que, siguiendo el estándar del IEEE Std 828-1990, se realizan las siguientes actividades:

- Identificación de la Configuración: consiste en identificar la estructura del producto, sus componentes y la naturaleza de éstos, y en hacerlos únicos y accesibles de alguna forma.
- Control de Cambios en la Configuración: trata de controlar las versiones y entregas de un producto y el control de cambios que se producen en él a lo largo de su ciclo de vida.
- Generación de Informes de Estado: para informar acerca del estado de los componentes de un producto y de las solicitudes de cambio, recogiendo estadísticas acerca de la evolución del producto.
- Auditoría de la Configuración: su finalidad es validar la completitud de un producto y la consistencia entre sus componentes, asegurando que el producto es lo que el usuario quiere.

No obstante, debido a las características de este trabajo en particular, se considera que la realización de las dos últimas actividades no aporta beneficios suficientes como para implantar su procedimiento, por lo que se desestima su empleo; ya que los dos primeros puntos ya aportan una buena calidad en la gestión de configuraciones.

3. ACTIVIDADES DE GESTIÓN DE CONFIGURACIONES

En esta sección se van a describir los artefactos considerados para almacenar en las bibliotecas junto con sus convenciones de nombrado, así como el modo de gestionar el control de cambios de dichas configuraciones.

3.1 Identificación de la Configuración

Para la correcta clasificación de los artefactos, se van a agrupar en distintas líneas base, asignando identificadores apropiados a cada uno de ellos. La definición de las líneas base parte de los flujos de trabajo definidos en la metodología RUP utilizada, lo que lleva a la siguiente taxonomía:

- Planificación del proyecto.
- Gestión de configuraciones.
- Requisitos.
- Análisis.
- Diseño.
- Implementación.
- Pruebas.
- Entrega del producto.

Para el nombrado de los diferentes ECS que constituye cada artefacto que sigue las normas definidas en este PGCS, se debe seguir la convención de un nombre en mayúsculas, que resuma lo más claramente posible el contenido del artefacto, seguido de una descripción breve del tipo de elemento que constituye, si se estima necesario, excepto en el caso de artefactos correspondientes a código fuente, que se nombrarán siguiendo la notación Pascal-Casing, o archivos binarios y scripts que se nombrarán siguiendo Camel-Casing. En cualquier caso, el nombre del artefacto deberá estar concluido por un punto seguido de una extensión representativa del contenido del mismo siguiendo los estándares de nombrado de archivos.

3.2 Gestión de la Configuración

Para la gestión de configuraciones del presente trabajo, se definen las siguientes bibliotecas:

- Biblioteca de trabajo. Se establece al inicio del trabajo y gestiona el área de trabajo de los ECS que se encuentran activos, esto es, sobre los que se están realizando modificaciones.
- Biblioteca maestra. Contiene elementos de configuración finalizados respecto a una iteración concreta del proceso de desarrollo. Sus elementos solamente pueden ser accedidos en modo de lectura una vez han sido establecidos.
- Biblioteca de copias de seguridad. Contiene copias incrementales de las bibliotecas de trabajo y maestra generadas periódicamente.

Para la gestión de las bibliotecas se emplea la herramienta Microsoft Team Foundation Server la cual será explicada en detalle en el apartado 4 de este mismo documento.

3.3 Control de cambios de la Configuración

Para la gestión de cambios de la configuración, se van a tener en cuenta dos tipos de modificaciones:

- Control de cambios informal. Cambios realizados en ECS fuera de la política de control de cambios, siendo éstos responsabilidad del desarrollador. Esta ausencia de control está motivada por la necesidad de realizar modificaciones de un modo más dinámico al comienzo del desarrollo de un ECS cuando los cambios son continuos y las notificaciones y controles de corrección podrían saturar el proceso.
- Control de cambios formal. Cambios realizados cuando un ECS es transferido a la biblioteca de trabajo o a la biblioteca maestra, para ello, el responsable de la modificación debe realizar una solicitud de cambio indicando si éste está motivado por la detección y resolución de un defecto, por una variación en los requisitos o por la realización de una mejora, la prioridad de la modificación según la relevancia de ésta y la línea base a la que afecta.

4. HERRAMIENTAS UTILIZADAS

Como se ha explicado en los anteriores apartados, la herramienta utilizada para la gestión de cambios de las configuraciones de software es Microsoft Team Foundation Server (TFS), aplicación que se integra con el entorno de desarrollo Visual Studio (también de Microsoft), permitiendo realizar gestión de código, recopilación de datos, generación de informes y trazado de proyectos en un entorno colaborativo de desarrollo de software.

El TFS proporciona una interfaz completamente integrada con el entorno de desarrollo utilizado para el desarrollo del presente proyecto, lo que facilita las labores de petición, comprobación y gestión de modificaciones de ECS. Asimismo, proporciona herramientas para gestionar posibles conflictos en ECS modificados por distintos desarrolladores y el acceso que estos deben tener a cada uno de los recursos.

También permite la gestión de diferentes bibliotecas, aunque, por motivos de seguridad, la biblioteca de copias de seguridad debe ser gestionada de un modo alternativo, mediante copias incrementales de ficheros.

A continuación, se describen las principales funcionalidades del TFS empleadas para la gestión de configuraciones:

- Bloquear para modificación. Esta funcionalidad permite el bloqueo tanto exclusivo como no exclusivo de un ECS, indicando la intención por parte del desarrollador de modificarlo. Si lo bloquea en modo exclusivo no permite que otros desarrolladores puedan tener acceso al mismo excepto que sea en modo de sólo lectura.

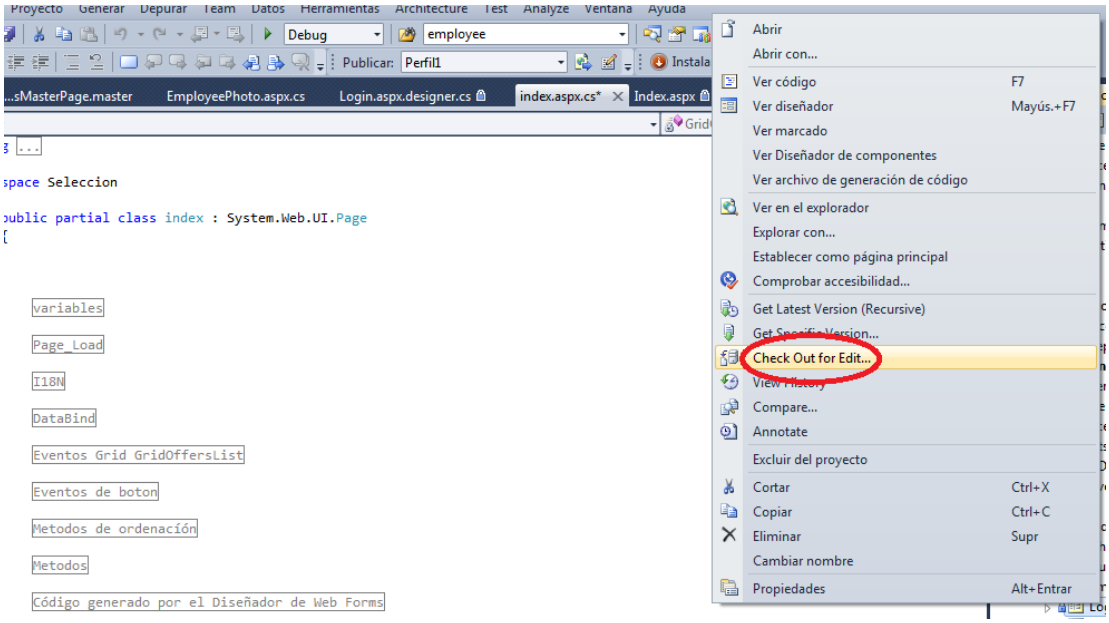


Figura 1. Bloqueo de ECS

- Descargar versión de la biblioteca. Esta funcionalidad permite la actualización de ECS en la biblioteca local de cada desarrollador para obtener la última versión con los últimos elementos modificados por el resto del equipo.

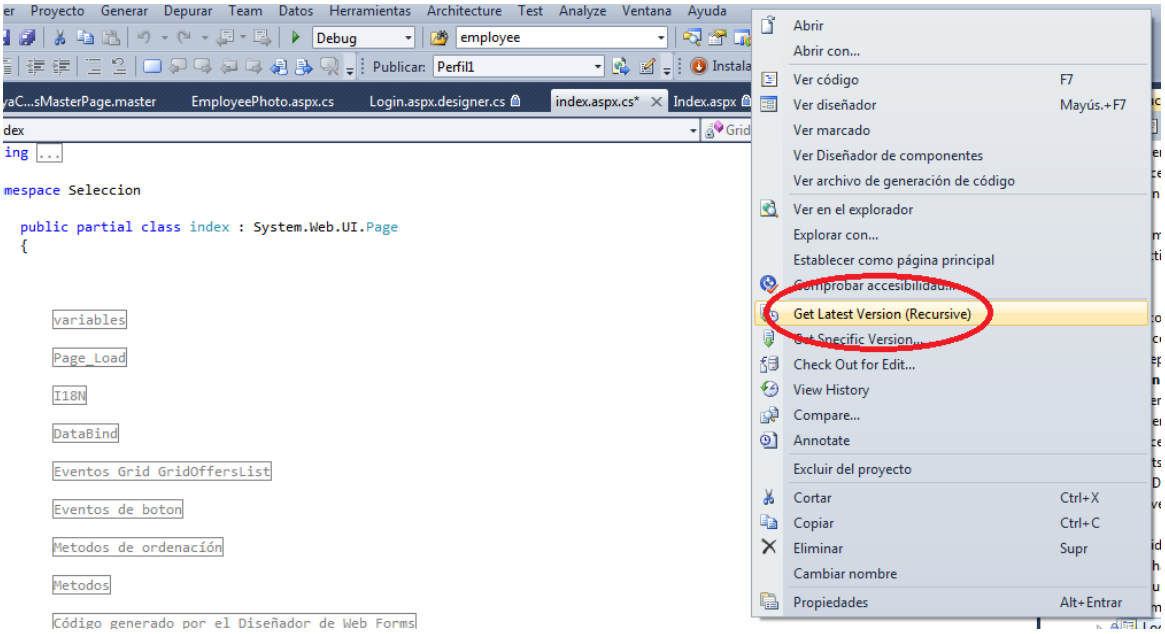


Figura 2. Descarga de versión

- Modificación de ECS. Esta funcionalidad permite la actualización de uno o varios ECS por parte de un desarrollador que los tenía previamente bloqueados, debiendo éste indicar la petición en el formato previamente establecido.

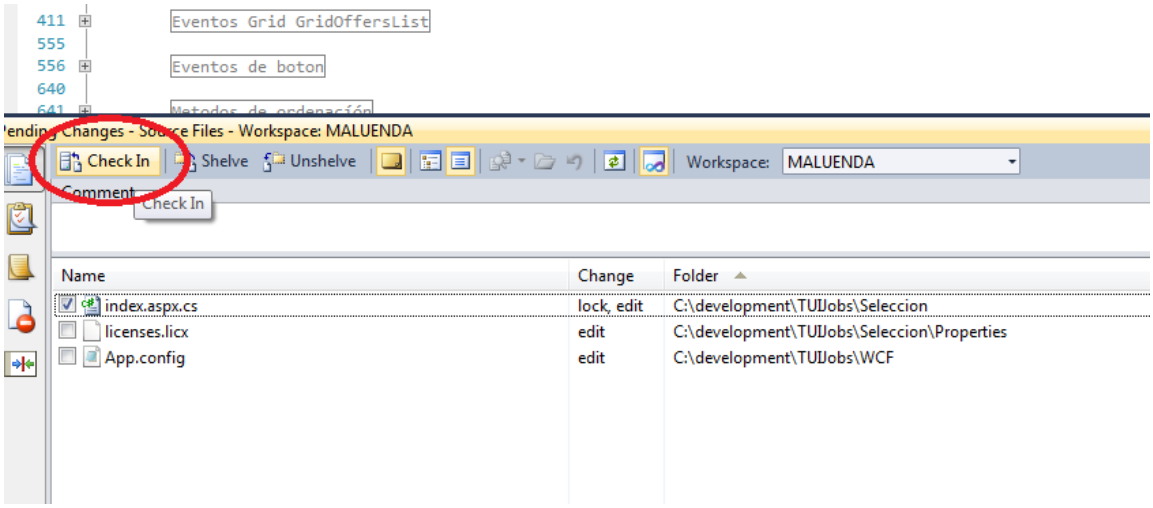


Figura 3. Modificación de ECS

- Resolución de conflicto en ECS modificado por diferentes desarrolladores. Esta funcionalidad permite mezclar el contenido de un ECS modificado simultáneamente por diferentes desarrolladores. Para que surja la necesidad de emplear esta funcionalidad es necesario que los desarrolladores bloqueen dicho artefacto en modo no exclusivo.

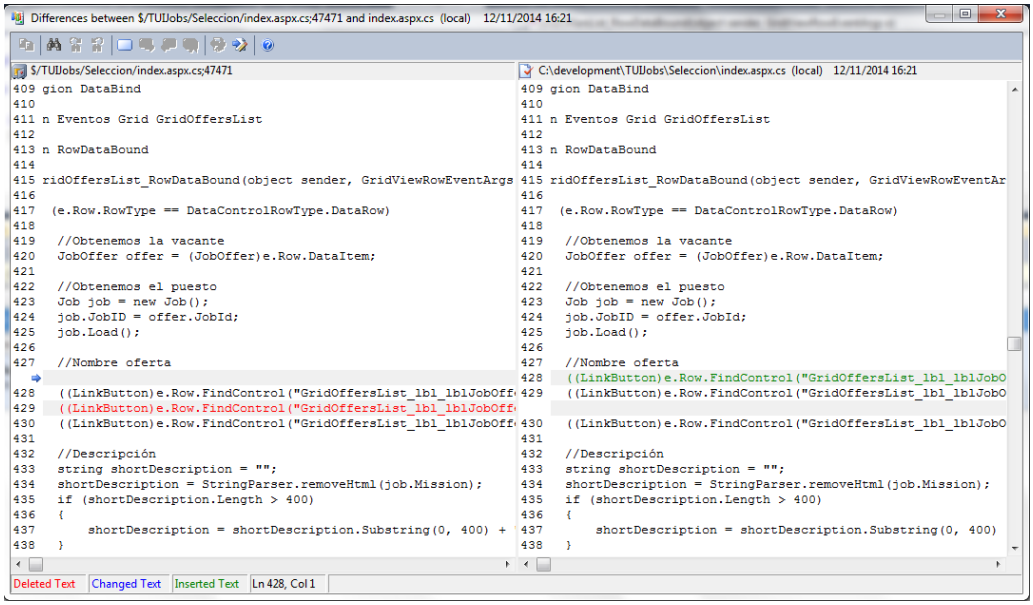


Figura 4. Resolución de conflictos

- Deshacer las últimas modificaciones realizadas, y volver a la versión previa de ECS. Esta funcionalidad permite revertir los últimos cambios realizados por el desarrollador, volviendo así a la versión anterior que fue registrada.

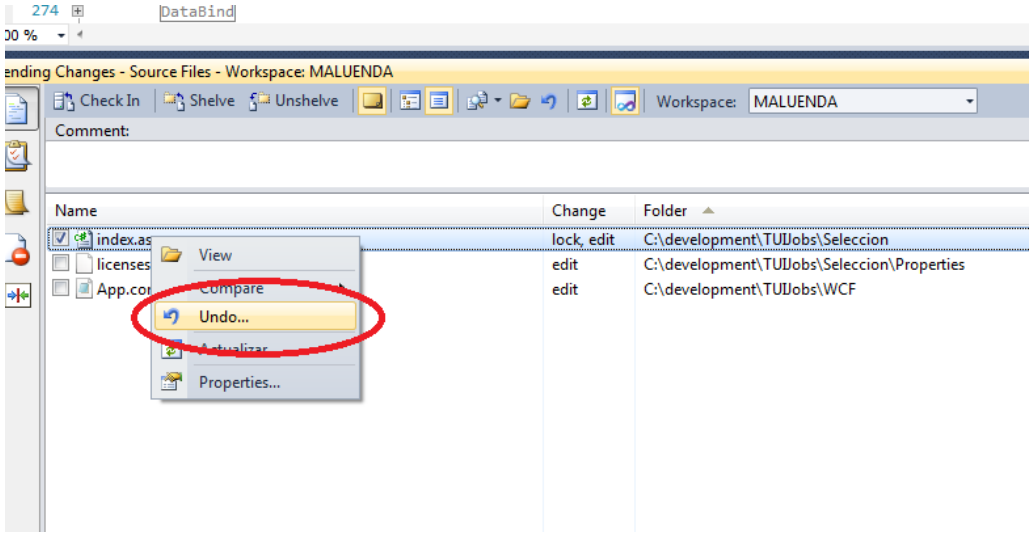


Figura 5. Descartar modificaciones

5. BIBLIOGRAFÍA

5.1 Referencias

[R1] I. Jacobson, G. Booch, J. Rumbaugh. 2000. *El Proceso Unificado de Desarrollo de Software*. Pearson Education

5.2 Referencias web

[W1] http://www.wikipedia.org

[W2] http://www.ieee.org

[W3] http://www.histaintl.com/soluciones/configuracion/configuracion.php

endalia®
HR Consulting & Software

ESPECIFICACIÓN DE REQUISITOS

DESARROLLO DE UN MÓDULO DE GESTIÓN DE VIAJES Y GASTOS

VERSIÓN 1.0
PUBLICADO EL 29/08/2016

HISTÓRICO DE REVISIONES

Fecha	Versión	Descripción	Autor
23/08/2016	1.0	Redacción inicial del documento	Nicolás Bailo Ortiz

ÍNDICE

1. INTRODUCCIÓN

1.1 Propósito del documento

El presente documento pretende presentar y describir los requisitos del sistema a desarrollar. Estos requisitos se dividen en requisitos funcionales, que pretenden identificar la funcionalidad objetivo del sistema, y en requisitos no funcionales, que son relativos a la seguridad, interfaz, rendimiento o escalabilidad del sistema a desarrollar. Además, es esta especificación lo que guiará el resto del desarrollo software del proyecto.

Una buena especificación de requisitos debe ser [R1]:

- Correcta
- Inequívoca
- Completa
- Consistente
- Categorizada según su importancia
- Comprobable
- Modificable
- Identificable

1.2 Alcance del documento

Este documento describe el resultado de la fase de estudio de los requerimientos del cliente. A lo largo del mismo se describirán los distintos requisitos funcionales y no funcionales que el sistema desarrollado debe cumplir.

1.3 Acrónimos

- RF: Requisito funcional
- RNF: Requisito no funcional
- PDF: Portable Document Format
- GUI: Graphical User Interface

1.4 Definiciones

- Graphical User Interface (GUI): es un tipo de interfaz de usuario que permite a los usuarios interactuar con un sistema mediante iconos visuales e indicadores gráficos en lugar de usar una interfaz basada en texto o comandos por teclado. [W1]

1.5 Referencias

Este documento no contiene referencias a otros documentos del trabajo.

1.6 Resumen

El objetivo de este documento es el de presentar los resultados obtenidos del estudio de las necesidades del cliente, identificando los diferentes requisitos que el sistema diseñado debe cumplir. Se compone de los siguientes apartados:

1. Una introducción en la que se presenta el documento, su propósito y su alcance, así como una lista de acrónimos, definiciones y referencias que puedan ser de utilidad.
2. La descripción general, en la que se presenta la funcionalidad del sistema a desarrollar, así como el tipo de usuarios que lo utilizará
3. Un listado detallado con los requisitos funcionales del sistema, su nivel de criticidad y sus dependencias.
4. Un listado detallado con los requisitos no funcionales del sistema, su nivel de criticidad y sus dependencias.
5. La bibliografía utilizada en esta fase del proyecto.

2. DESCRIPCIÓN GENERAL

2.1 Funciones generales del sistema

Este módulo de gestión de viajes y gastos es un sistema que permite gestionar solicitudes asociadas a viajes o gastos dentro de una empresa. Los empleados podrán solicitar la aprobación de un viajo o gasto, así como realizar la liquidación del mismo una vez se haya aprobado y efectuado. Para ello, se permite la creación, modificación y envío de solicitudes por parte de los empleados y la aprobación, rechazo o devolución de las mismas por parte de los responsables.

2.2 Características de los usuarios

Los usuarios que interactuarán con el módulo son:

- Por un lado, los usuarios que no tienen personas a su cargo en el seno de la empresa podrán únicamente crear, modificar y enviar solicitudes. No tendrán acceso a la aprobación de solicitudes puesto que ésta se hace en línea descendente en la estructura organizativa de la empresa.
- Por otro lado, los usuarios que sean responsables de otros empleados podrán realizar aprobaciones, teniendo por lo general acceso a la totalidad del módulo.

Dependiendo de su puesto en la estructura organizativa de la empresa, un usuario puede pertenecer al primer grupo o a ambos.

Por último, se habrá de considerar que los usuarios no poseen una formación específica, por lo que es posible que no estén familiarizados con el uso de aplicaciones web.

3. REQUISITOS FUNCIONALES DEL SISTEMA

Nombre	**RF – 01: Visualización de solicitudes realizadas por el usuario**
Descripción	El módulo contará con una pantalla donde se muestre un histórico de las solicitudes creadas por el usuario. Además, ésta deberá permitir algún tipo de interacción que permita alterar las solicitudes mostradas (filtrado, ordenación).
Nivel de criticidad	Alto
Dependencias	-

Nombre	**RF – 02: Visualización de solicitudes de colaboradores**
Descripción	El módulo contará con una sección en la que se muestren los listados de solicitudes de los empleados que estén por debajo del usuario en la estructura organizativa de la empresa (los llamados *colaboradores*. Además, se deberá mostrar un resumen por puesto, con las solicitudes de los empleados por debajo de ese puesto.
Nivel de criticidad	Alto
Dependencias	-

Nombre	**RF – 03: Visualización de solicitudes pendientes de aprobar por el usuario**
Descripción	El módulo contará con una sección en la que se listen las solicitudes que el usuario tenga que aprobar, de forma que le facilite la tarea de aprobar solicitudes pendientes.
Nivel de criticidad	Alto
Dependencias	-

Nombre	**RF – 04: Visualización de las diferentes secciones del módulo en función de distintos permisos de usuario**
Descripción	Cada sección de este módulo deberá tener un permiso de usuario asociado, de forma que el equipo de RRHH de la empresa cliente pueda gestionar el grado de acceso al módulo de cada usuario de forma independiente.
Nivel de criticidad	Alto
Dependencias	Visualización de solicitudes realizadas por el usuario, visualización de solicitudes de colaboradores y visualización de solicitudes pendientes de aprobar por el usuario.

Nombre	**RF – 05: Creación de una solicitud de viaje o gasto**
Descripción	Los usuarios podrán crean diferentes solicitudes de viajes o gastos en función de los tipos establecidos por RRHH. El proceso de creación de solicitudes deberá ser intuitivo y lo más breve posible.
Nivel de criticidad	Alto
Dependencias	-

Nombre	**RF – 06: Visualización de una solicitud de viaje o gasto**
Descripción	El módulo contará con una sección en la que se listen las solicitudes que el usuario tenga que aprobar, de forma que le facilite la tarea de aprobar solicitudes pendientes.
Nivel de criticidad	Alto
Dependencias	-

Nombre	**RF – 07: Modificación de una solicitud de viaje o gasto**
Descripción	Los usuarios podrán modificar las solicitudes de viajes o gastos siempre y cuando éstas no hayan sido enviadas para ser aprobadas.
Nivel de criticidad	Alto
Dependencias	Creación de una solicitud de viaje o gasto, visualización de una solicitud de viaje o gasto.

Nombre	RF – 08: Envío de una solicitud de viaje o gasto
Descripción	Los usuarios podrán enviar las solicitudes de viajes o gastos desde la sección de detalles la solicitud, para que sus respectivos responsables la aprueben.
Nivel de criticidad	Alto
Dependencias	Visualización de una solicitud de viaje o gasto.

Nombre	RF – 09: Gestión de documentos y facturas
Descripción	El módulo debe ofrecer un sistema de gestión de documentos asociados a solicitudes y de facturas asociadas a los distintos gastos de una solicitud.
Nivel de criticidad	Alta
Dependencias	-

Nombre	RF – 10: Generación de un informe de detalles de una liquidación
Descripción	Los usuarios podrán descargar un informe en formato PDF con información relativa a la liquidación de la oferta seleccionada. La generación de dicho informe será posible únicamente cuando la liquidación haya sido completada y enviada para su aprobación.
Nivel de criticidad	Medio
Dependencias	-

Nombre	RF – 11: Notificación vía email de las actualizaciones de estado de una solicitud
Descripción	Los usuarios implicados en la siguiente fase de una solicitud serán notificados vía email cuando dicha solicitud se encuentre en una fase en la que ellos tienen que realizar alguna acción. El email contendrá la información básica de la solicitud y un enlace a la solicitud para facilitar el acceso a ésta.
Nivel de criticidad	Medio
Dependencias	-

Nombre	**RF – 12: Acceso al módulo desde la pantalla principal de la aplicación**
Descripción	Los usuarios podrán acceder al listado de sus solicitudes a través de un acceso directo situado en la página principal de la aplicación. Además, se mostrará un indicador que avise del número de solicitudes pendientes a aprobar, que también hará las funciones de acceso directo a la sección de aprobaciones.
Nivel de criticidad	Medio
Dependencias	-

4. REQUISITOS NO FUNCIONALES DEL SISTEMA

Nombre	**RNF – 01: Eficiencia del sistema**
Descripción	Toda funcionalidad del sistema y navegación entre las diferentes secciones del módulo debe responder al usuario en menos de 3 segundos.
Nivel de criticidad	Alto
Dependencias	-

Nombre	**RNF – 02: Integración del módulo en la aplicación de Endalia**
Descripción	El sistema deberá integrarse en el conjunto del Sistema de Gestión y Organización de Recursos Humanos de la entidad, conviviendo con el resto de módulo como uno más del conjunto global. Esto implica adecuarse al estilo, diseño y funcionalidad del resto de la aplicación.
Nivel de criticidad	Alto
Dependencias	-

Nombre	**RNF – 03: Aplicación parametrizable**
Descripción	El módulo será fácilmente parametrizable en función de las necesidades del cliente. En esto se incluyen los distintos tipos de solicitudes que los usuarios pueden crear, los distintos tipos de gastos que pueden ser asociados a solicitudes y las distintas fases de aprobación por parte de los responsables de la empresa.
Nivel de criticidad	Medio
Dependencias	-

Nombre	RNF – 03: Aplicación parametrizable
Descripción	El módulo será fácilmente parametrizable en función de las necesidades del cliente. En esto se incluyen los distintos tipos de solicitudes que los usuarios pueden crear, los distintos tipos de gastos que pueden ser asociados a solicitudes y las distintas fases de aprobación por parte de los responsables de la empresa.
Nivel de criticidad	Alto
Dependencias	-

Nombre	RNF – 04: Interacción con el usuario mediante GUI consistente
Descripción	La interacción con el usuario deberá realizarse a través de una GUI consistente con la interfaz ya desarrollada en el resto de la aplicación. Es decir, deberá respetar el diseño de elementos como tablas, botones o colores, así como la distribución habitual de las pantallas.
Nivel de criticidad	Alto
Dependencias	Integración del módulo en la aplicación de Endalia

Nombre	RNF – 05: Aplicación intuitiva
Descripción	El módulo deberá ser intuitiva y fácil de usar, incluso para usuarios poco familiarizados con este tipo de aplicaciones o tecnologías.
Nivel de criticidad	Medio
Dependencias	-

Nombre	RNF – 06: Internacionalización
Descripción	El módulo deberá poder adaptarse fácilmente a otras culturas, tanto en el idioma de los textos como en el formato de fechas o decimales. Además, deberá tener en cuenta la moneda utilizada en la gestión de gastos.
Nivel de criticidad	Medio
Dependencias	-

Nombre	RNF – 07: Tolerancia a fallos
Descripción	El módulo deberá ser tolerante a fallos, por lo que deberá poder recuperarse de un error no controlado en el sistema y ofrecer información relativa a dicho error.
Nivel de criticidad	Alto
Dependencias	-

Nombre	RNF – 08: Disponibilidad del sistema
Descripción	El módulo deberá ser accesible las 24 horas del día y los 7 días de la semana, adecuándose a la disponibilidad del sistema global de la entidad. Además, deberá permitir el acceso concurrente a múltiples usuarios sin que esto conlleve una pérdida de rendimiento o inconsistencias en los datos.
Nivel de criticidad	Alto
Dependencias	-

Nombre	RNF – 09: Resolución
Descripción	El módulo deberá ser correctamente visualizado en pantallas con resoluciones iguales o superiores a 1024x768, adecuándose a los estándares del resto del sistema de la organización.
Nivel de criticidad	Alto
Dependencias	-

Nombre	RNF – 10: Seguridad e integridad de los datos
Descripción	El módulo contiene información potencialmente confidencial por lo que no se deberá permitir el acceso a un usuario no registrado en la aplicación. Por otro lado, se deberá asegurar la integridad de los datos en las transacciones con la base de datos.
Nivel de criticidad	Alto
Dependencias	-

Nombre	**RNF – 11: Compatibilidad con distintos navegadores**
Descripción	El módulo deberá funcionar como mínimo en los siguientes navegadores: Internet Explorer (versiones superiores a la 8.0), Mozilla Firefox y Google Chrome, adecuándose a los estándares de la aplicación de Endalia.
Nivel de criticidad	Alto
Dependencias	-

5. BIBLIOGRAFÍA

5.1 Referencias

[R1] IEEE Recommended Practice for Software Requirements Specifications, Std 830-1998.

5.2 Referencias web

[W1] http://www.wikipedia.org

endalia®
HR Consulting & Software

ANÁLISIS

DESARROLLO DE UN MÓDULO DE GESTIÓN DE VIAJES Y GASTOS

VERSIÓN 1.0
PUBLICADO EL 29/08/2016

HISTÓRICO DE REVISIONES

Fecha	Versión	Descripción	Autor
23/08/2016	1.0	Redacción inicial del documento	Nicolás Bailo Ortiz

ÍNDICE

1. INTRODUCCIÓN

1.1 Propósito del documento

En este documento se presenta la fase de análisis de este proyecto software. Partiendo de los requisitos descritos en el documento de especificación de requisitos se va a realizar un análisis de la arquitectura del sistema a desarrollar, identificando casos de uso y la estructura de paquetes. También se realizará un estudio de los requisitos especiales del sistema. Los resultados se este análisis marcarán las pautas a seguir en las siguientes fases del desarrollo.

1.2 Alcance del documento

Este documento muestra los resultados de la fase de análisis del proyecto. En él, se describirá el proceso de análisis para a continuación realizar un estudio de los casos de uso y sus diferentes actores, así como un análisis de paquetes.

1.3 Acrónimos

- GUI: Graphic User Interface.
- UML: Unified Modeling Language.

1.4 Definiciones

- Caso de uso: especificación de las secuencias de acciones que pueden ser efectuadas por un sistema, subsistema o clase por interacción con actores externos.
- Colaborador: un empleado es colaborador de otro cuando el primero está por debajo del segundo en la estructura organizativa de la empresa. Además, se le denomina “colaborador directo” cuando el primer empleado está justo un puesto por debajo del segundo en dicha estructura.
- Diagrama de actividades: notación gráfica que representa los flujos de trabajo de los componentes en un sistema, paso a paso. Muestra el flujo de control general del sistema.
- Diagrama de casos de uso: notación gráfica que describe un caso de uso.
- Diagrama de secuencia: notación gráfica que muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo.
- UML: lenguaje de modelado de sistemas de software desarrollado por *Rational* y de uso extendido.

1.5 Referencias

En este documento se referencian los siguientes documentos del proyecto:

- Especificación de requisitos: documento en el que se especifican los requisitos del sistema.

1.6 Resumen

En este documento se presenta la fase de análisis del trabajo. Se compone de los siguientes apartados:

1. Introducción, propósito y alcance del documento.
2. Descripción del proceso de análisis utilizado en este proyecto.
3. Identificación y descripción de los casos de uso.
4. Realización de un análisis de paquetes.
5. Identificación de los requisitos especiales del sistema.
6. Presentación de la bibliografía utilizada para la realización de este documento.

2. DESCRIPCIÓN DEL PROCESO

El flujo de trabajo del análisis se compone de distintas actividades, como son: análisis de los casos de uso, análisis de paquetes y análisis de requisitos especiales comunes.

El análisis de paquetes pretende identificar los componentes esenciales del sistema, garantizando que mediante los mismos se cumplen los objetivos definidos en el documento de requisitos, así como en la descripción de los casos de uso, y describir las dependencias entre paquetes, de forma que pueda estimarse el efecto de cambios futuros.

El análisis de los requisitos especiales comunes tiene el objetivo de identificar las peculiaridades del sistema que determinarán restricciones y especificaciones para fases futuras del proyecto.

3. ANÁLISIS DE LOS CASOS DE USO

3.1 Introducción

En este apartado se va a realizar el análisis de los casos de uso del sistema, identificando todos los procesos que tienen lugar en el sistema.

El análisis de un caso de uso se realiza con el objetivo de profundizar en la funcionalidad desarrollada por el mismo. Para formalizar el detalle de los casos de uso, se van a utilizar las siguientes técnicas:

- Diagramas de caso de uso: describen la relación entre los usuarios del sistema y los casos de uso.
- Diagramas de secuencia: muestran la vista dinámica del caso de uso. Es decir, describen las interacciones existentes a lo largo del tiempo.
- Diagramas de actividades: describen el flujo de trabajo del caso de uso.

3.2 Actores de los casos de uso

En el ámbito de este proyecto, se denominan actores a los distintos usuarios que interaccionan con la aplicación, los cuales han sido caracterizados en el documento de especificación de requisitos. Para realizar el presente análisis, se distinguirán dos tipos de usuarios diferenciados:

- Solicitantes:
    - Son usuarios que acceden al módulo con el objetivo de manipular sus propias solicitudes.
    - Pueden crear una nueva solicitud, editar una ya existente, enviarla para aprobación, o simplemente visualizar sus solicitudes a modo de consulta.
- Aprobadores:
    - Son usuarios de tipo solicitante, pero con privilegios añadidos.
    - Además de realizar las acciones que propias de los usuarios de tipo solicitante, pueden visualizar las solicitudes de los empleados situados por debajo en la estructura organizativa y aprobar las solicitudes que estén en una fase que dependan de ellos mismos.

3.3 Casos de uso

En esta sección se especifican diversos casos de uso del módulo de Gestión de Viajes y Gastos. Cada caso de uso de presentará mediante una breve descripción y un diagrama explicativo. Para comenzar, se mostrará un esquema general de los casos de uso del sistema, lo que ofrecerá una visión global del funcionamiento del sistema desde el punto de vista del usuario. A continuación se detallarán los casos de uso de los distintos actores.

3.3.1 Esquema general de los casos de uso

El esquema general de los casos de uso de la aplicación se muestra en la Figura 1. En él se presenta la base sobre la que se van a plantear los casos de uso explicados más adelante. Además, se puede apreciar que los dos tipos de actores se deben comunicar de forma idéntica con el sistema.

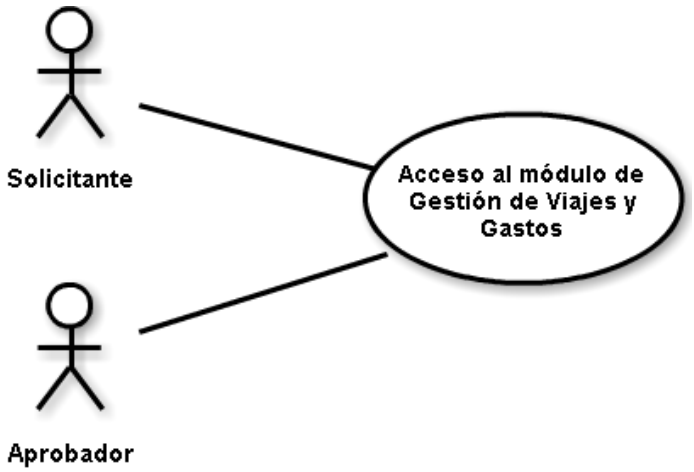


Figura 1. Esquema general de casos de uso

3.3.2 Solicitantes

A continuación se van a describir los casos de uso relacionados con los usuarios de tipo solicitante. En total se dispone de cinco casos de uso diferenciados, tal y como muestra la Figura 2. Como se ha explicado anteriormente, todos los casos de uso detallados a continuación requieren un acceso previo al módulo de Gestión de Viajes y Gastos, mediante autentificación previa introduciendo un nombre de usuario y una contraseña. Cabe destacar que en ningún caso se puede considerar el registro y autentificación del usuario como parte del módulo desarrollado y por lo tanto de este proyecto, ya que éste se engloba en una aplicación en la que ya existe esta funcionalidad, por lo que no se presenta el caso de uso relativo a esta funcionalidad.

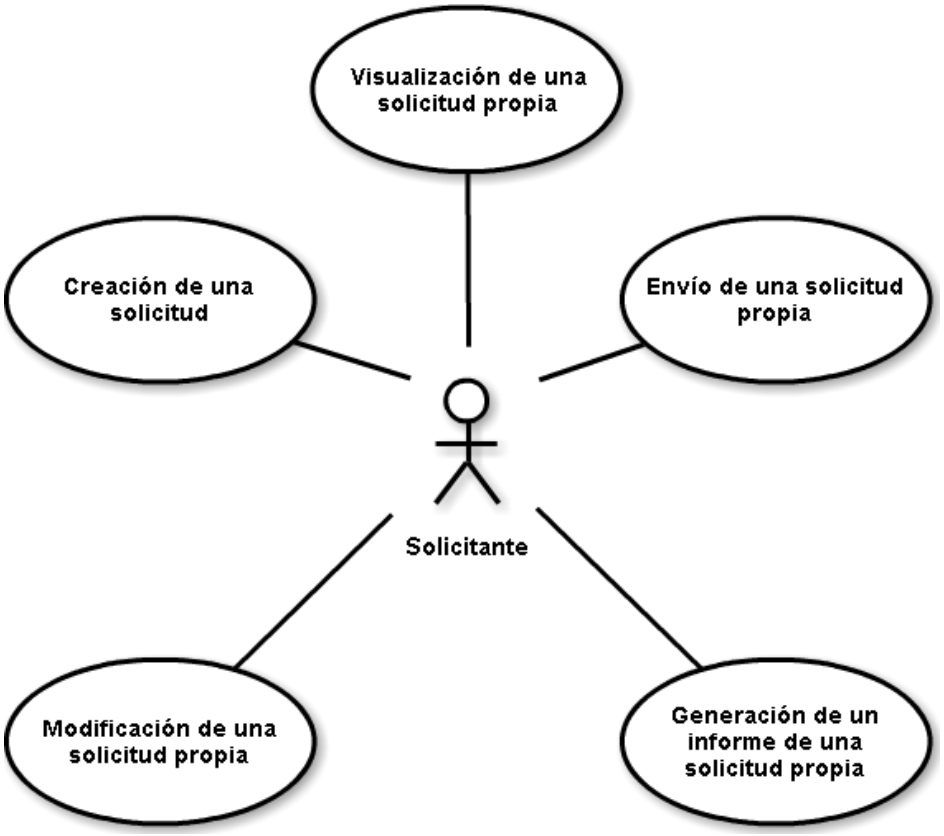


Figura 2. Casos de uso de un usuario de tipo solicitante

Así, los casos de uso mostrados en la figura son los siguientes:

- Creación de una nueva solicitud.
- Visualización de una solicitud creada por el usuario.
- Edición de una solicitud creada por el usuario.
- Envío de una solicitud creada por el usuario.
- Generación de un informe relativo a una solicitud creada por el usuario.

A continuación se procede a detallar los diferentes casos de uso en orden, mostrando un diagrama de secuencia así como un diagrama de actividades para cada uno:

- Creación de una nueva solicitud. El usuario interactúa con la GUI del sistema y mediante un asistente crea una solicitud con la información básica. Esta solicitud se guarda en formato borrador y permite que el usuario siga introduciendo información sobre la misma interactuando con la GUI.
    - Diagrama de secuencia (Figura 3):

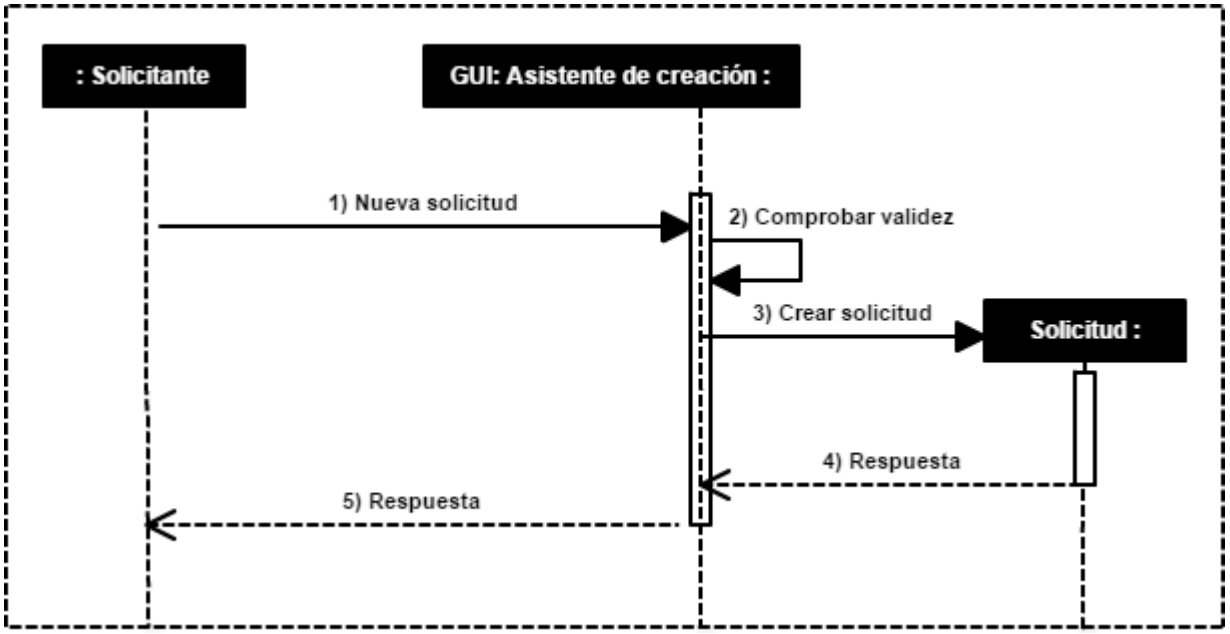


Figura 3. Diagrama de secuencia del caso de uso "Creación de una nueva solicitud"

- Diagrama de actividades (Figura 4):

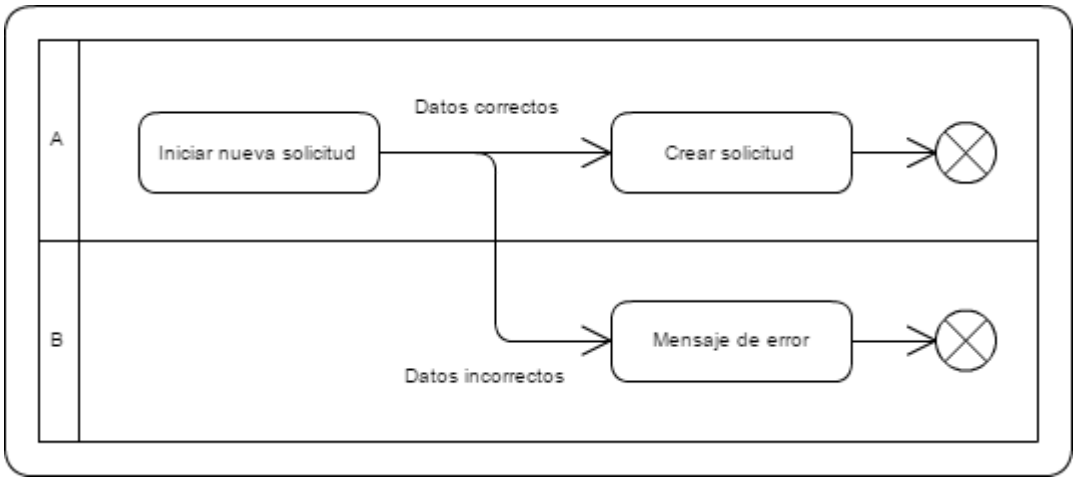


Figura 4. Diagrama de actividades del caso de uso "Creación de una nueva solicitud"

- Visualización de una solicitud creada por el usuario. El usuario interactúa con la GUI seleccionando una solicitud que haya sido creada por él mismo. El sistema muestra a continuación la información correspondiente a dicha solicitud.
    - Diagrama de secuencia (Figura 5):

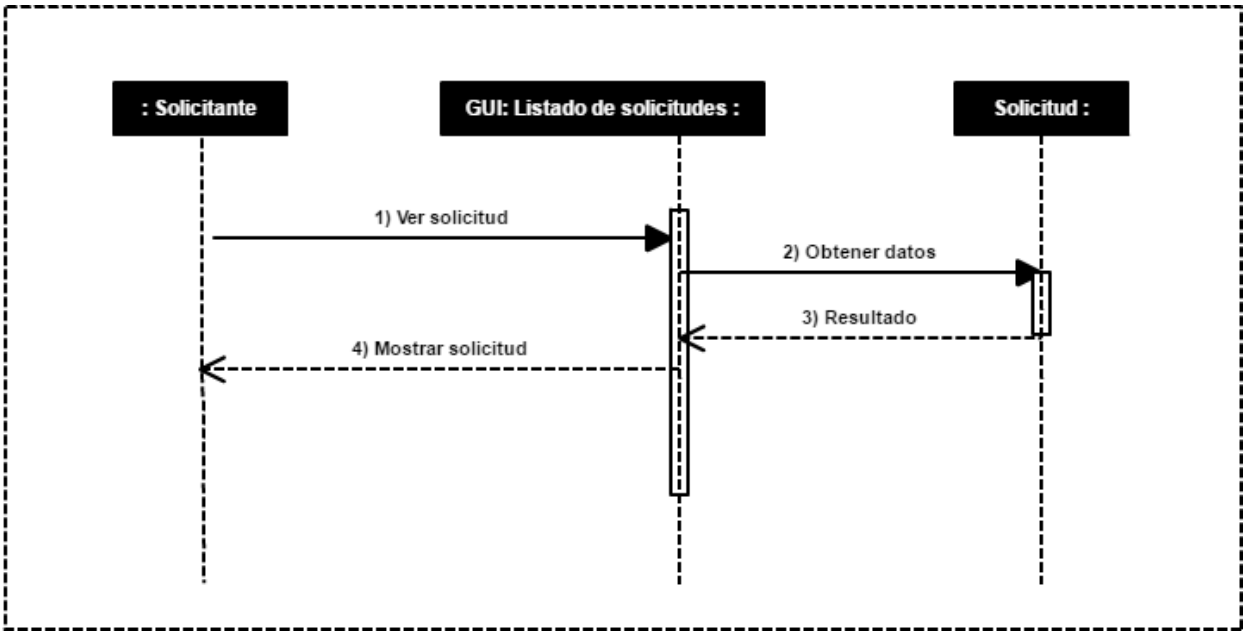


Figura 5. Diagrama de secuencia del caso de uso "Visualización de una nueva solicitud"

- Diagrama de actividades (Figura 6):

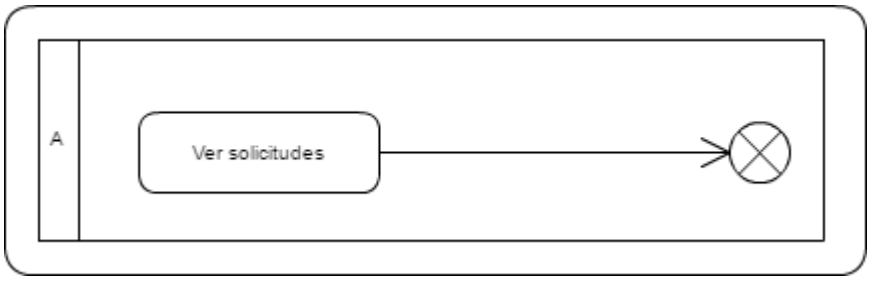


Figura 6. Diagrama de actividades del caso de uso "Visualización de una nueva solicitud"

- Edición de una solicitud creada por el usuario. El usuario deberá seleccionar una solicitud que él haya creado de las lista de solicitudes para a continuación modificar cualquiera de sus campos y validar los cambios. El sistema notificará del resultado de la actualización.
    - Diagrama de secuencia (Figura 7):

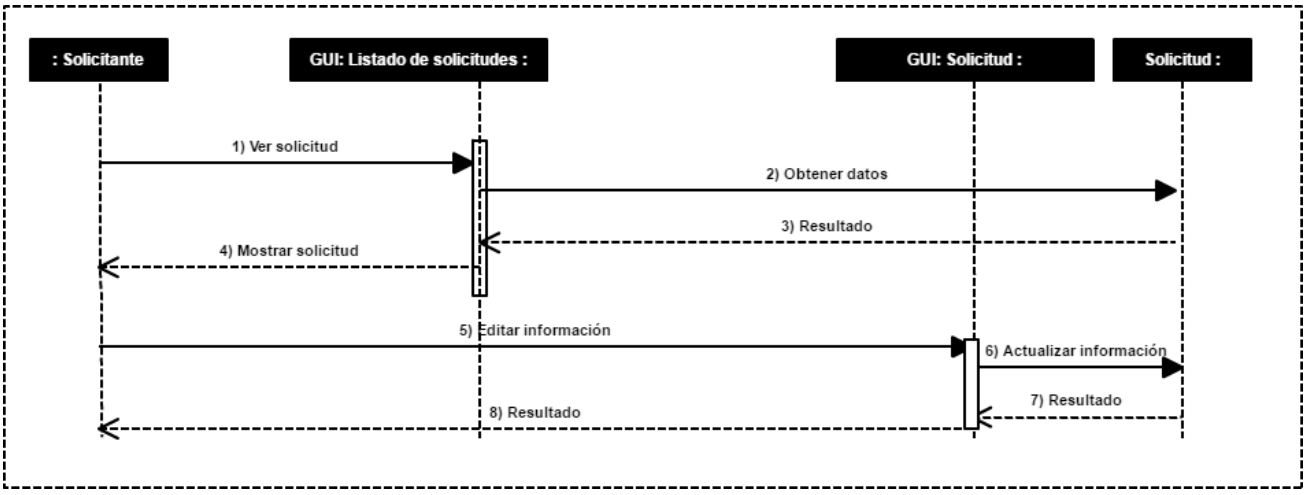


Figura 7. Diagrama de secuencia del caso de uso "Edición de una solicitud creada por el usuario"

- Diagrama de actividades (Figura 8):

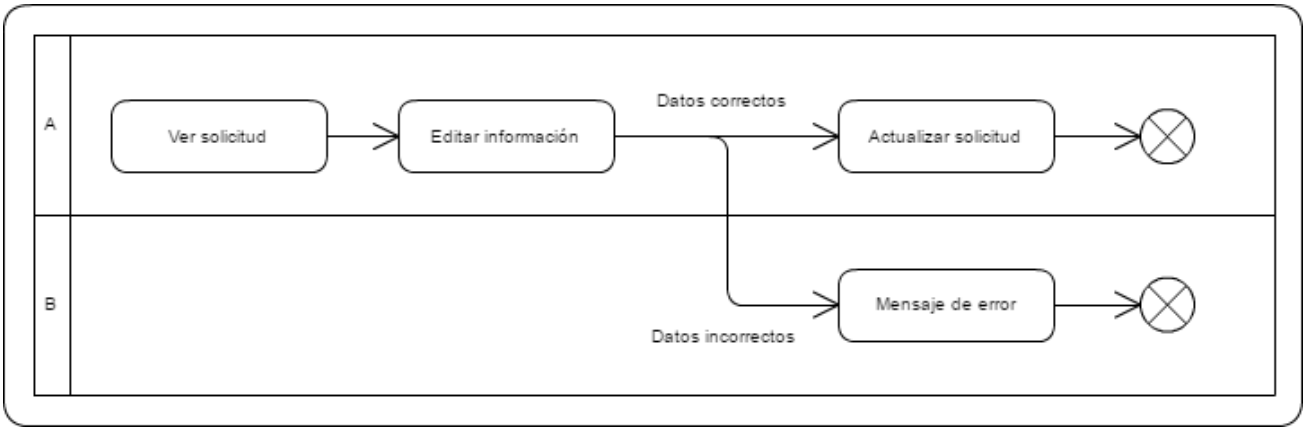


Figura 8. Diagrama de actividades del caso de uso "Edición de una solicitud creada por el usuario"

- Envío de una solicitud creada por el usuario. El usuario deberá seleccionar una de las solicitudes que él haya creado desde el listado de solicitudes para a continuación realizar el envío para aprobación de dicha solicitud. El sistema actualizará el estado de dicha solicitud y enviará el email correspondiente.
    - Diagrama de secuencia (Figura 9):

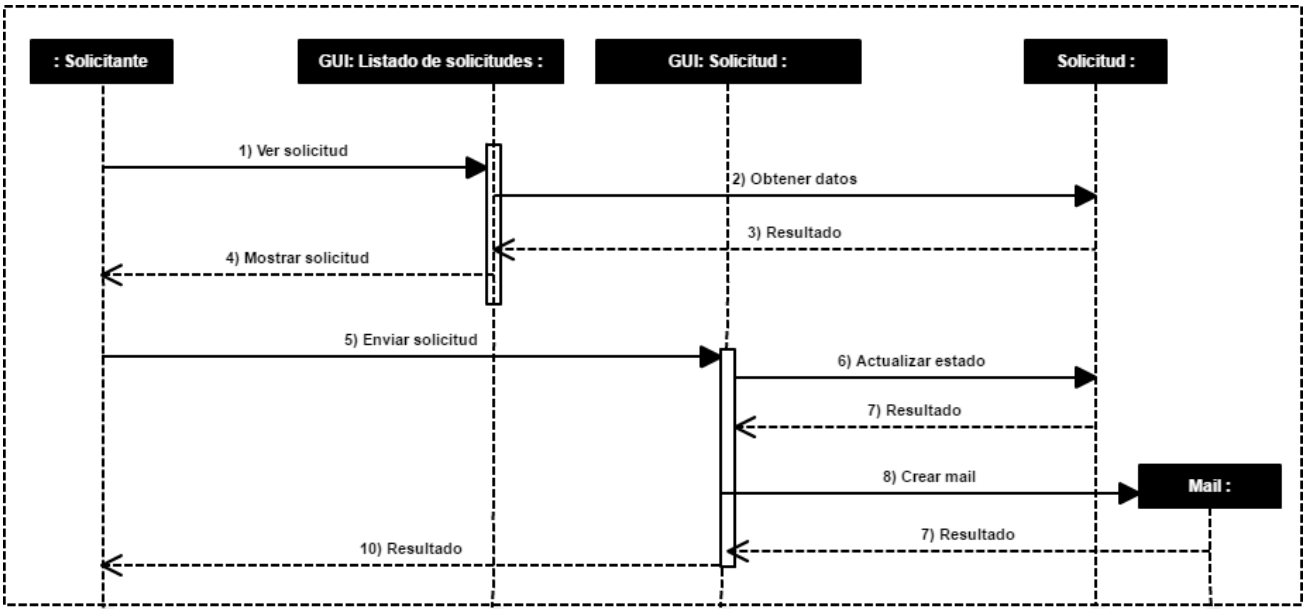


Figura 9. Diagrama de secuencia del caso de uso "Envío de una solicitud creada por el usuario"

    - Diagrama de actividades (Figura 10):

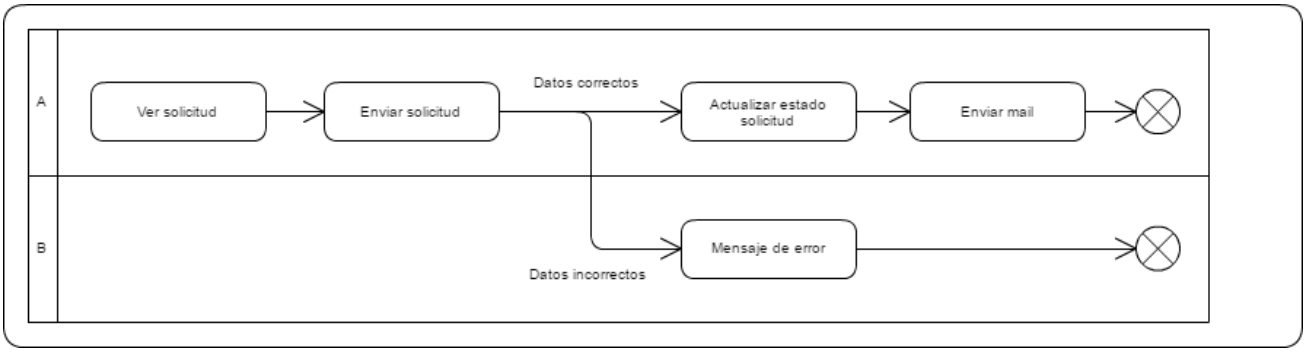


Figura 10. Diagrama de actividades del caso de uso "Envío de una solicitud creada por el usuario"

- Generación de un informe relativo a una solicitud creada por el usuario. El usuario deberá seleccionar una de las solicitudes que él haya creado desde el listado de solicitudes y a continuación realizar la descarga del informe relativo a dicha solicitud.

- Diagrama de secuencia (Figura 11):

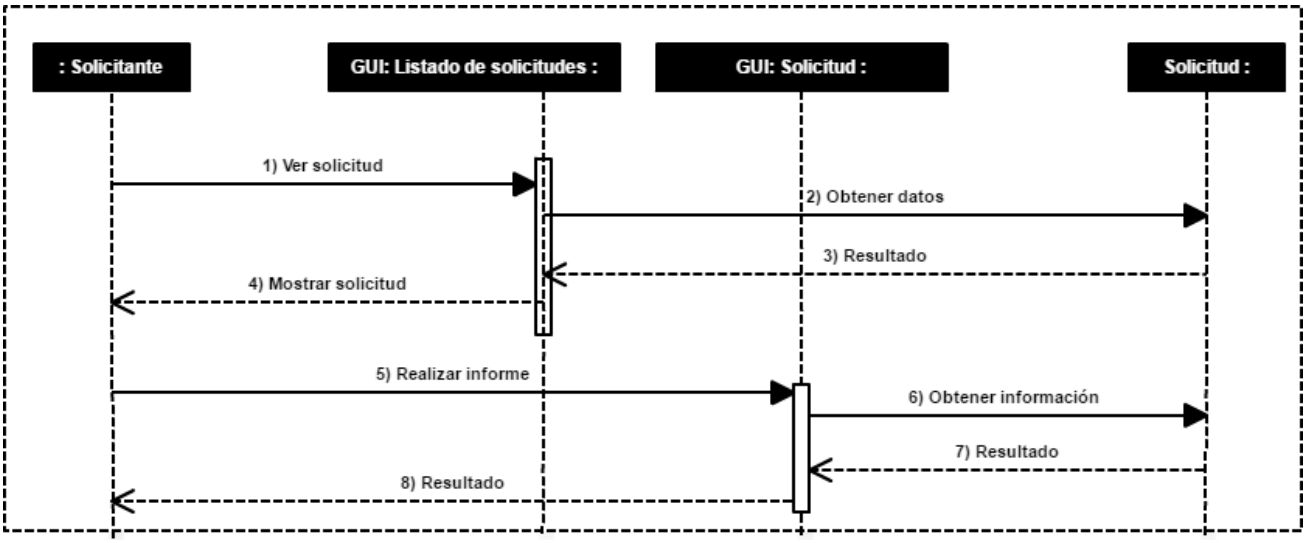


Figura 11. Diagrama de secuencia del caso de uso "Generación de un informe relativo a una solicitud creada por el usuario"

- Diagrama de actividades (Figura 12):

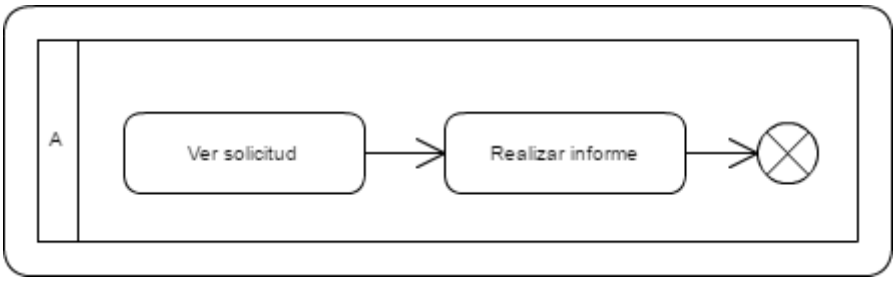


Figura 12. Diagrama de actividades del caso de uso "Generación de un informe relativo a una solicitud creada por el usuario"

3.3.3 Aprobadores

A continuación se van a describir los casos de uso relacionados con los usuarios de tipo aprobador. En total se dispone de seis casos de uso diferenciados, tal y como muestra la Figura 13. Como ocurría con los casos de uso de los usuarios de tipo solicitante, todos los casos de uso detallados a continuación requieren un acceso previo al módulo de Gestión de Viajes y Gastos, mediante autentificación previa introduciendo un nombre de usuario y una contraseña.

Como se puede observar en la Figura 13, cinco de los seis casos de uso del usuario de tipo aprobador son idénticos a los casos de uso del usuario de tipo solicitante, por lo que en este apartado sólo se explicará el sexto caso de uso: “Aprobación, rechazo o devolución de una solicitud”. Los diagramas realizados en el apartado anterior son perfectamente válidos para los casos de uso análogos del usuario de tipo aprobador.

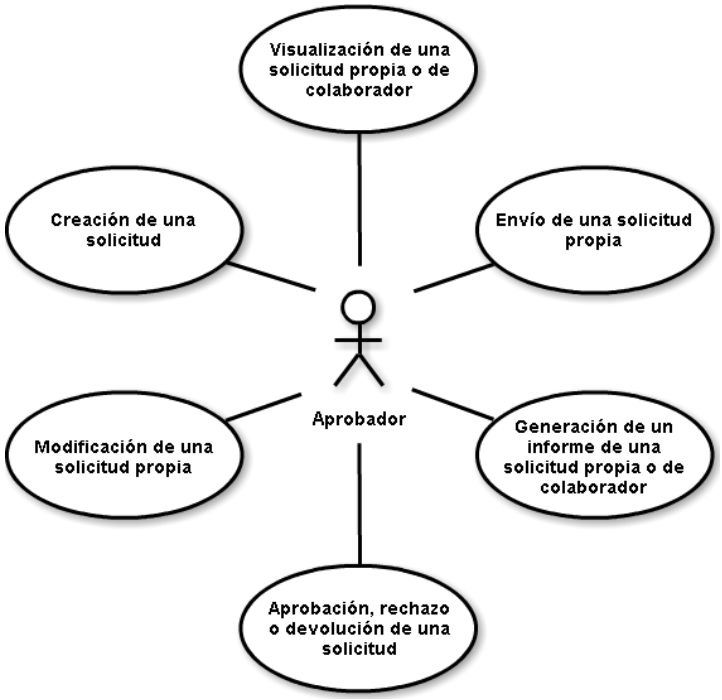


Figura 13. Casos de uso de un usuario de tipo aprobador

- Aprobación, rechazo o devolución de una solicitud. El usuario deberá acceder a la solicitud pendiente de aprobar y deberá realizar su aprobación, rechazo o devolución. El usuario tendrá la posibilidad de introducir comentarios al respecto de la aprobación antes de confirmar su decisión.
    - Diagrama de secuencia (Figura 14):

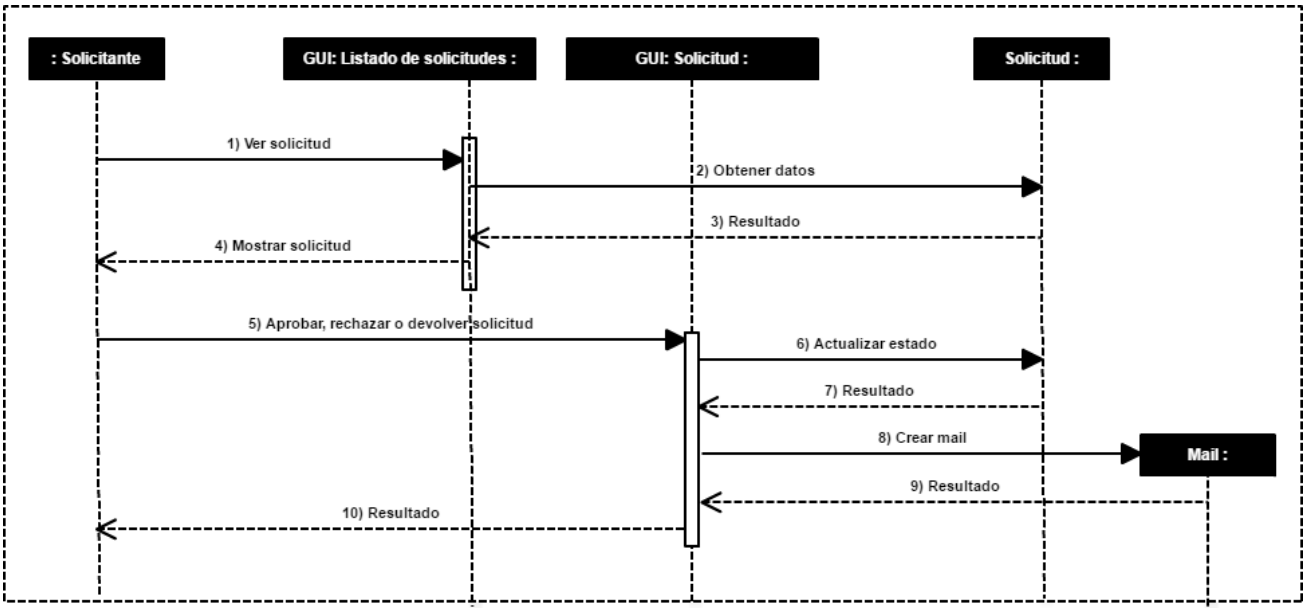


Figura 14. Diagrama de secuencia del caso de uso "Aprobación, rechazo o devolución de una solicitud"

- Diagrama de actividades (Figura 15):

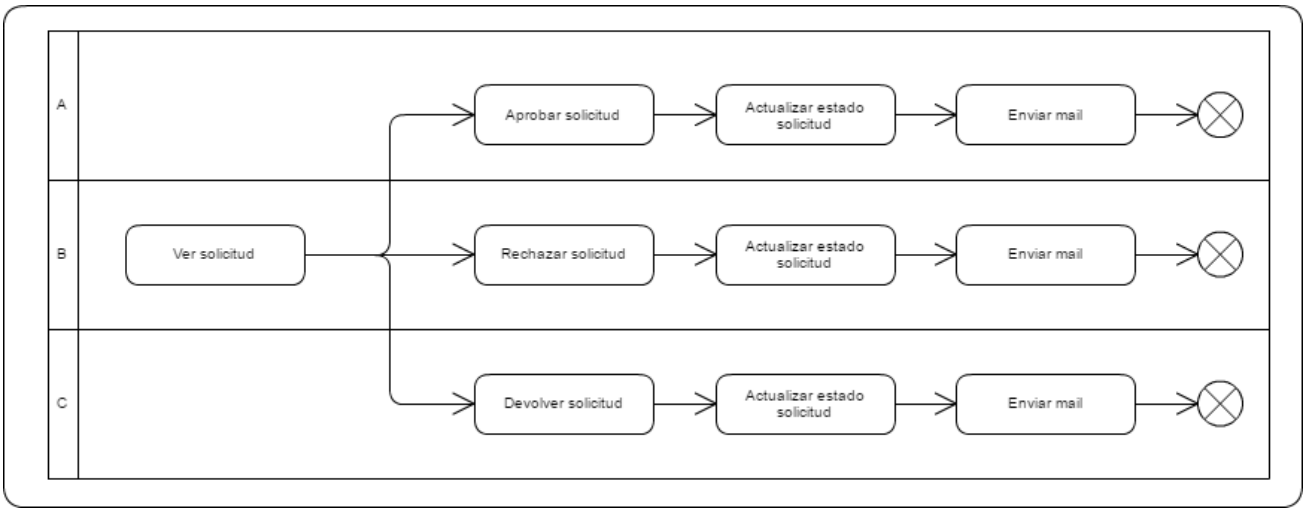


Figura 15. Diagrama de actividades del caso de uso "Aprobación, rechazo o devolución de una solicitud"

4. ANÁLISIS DE PAQUETES

4.1 Introducción

Una vez descritos los casos de uso, se procede a identificar las entidades relevantes y sus relaciones. Se van a definir paquetes de análisis para organizar las diferentes entidades y funcionalidades del sistema en partes más manejables.

4.2 Identificación de paquetes de análisis

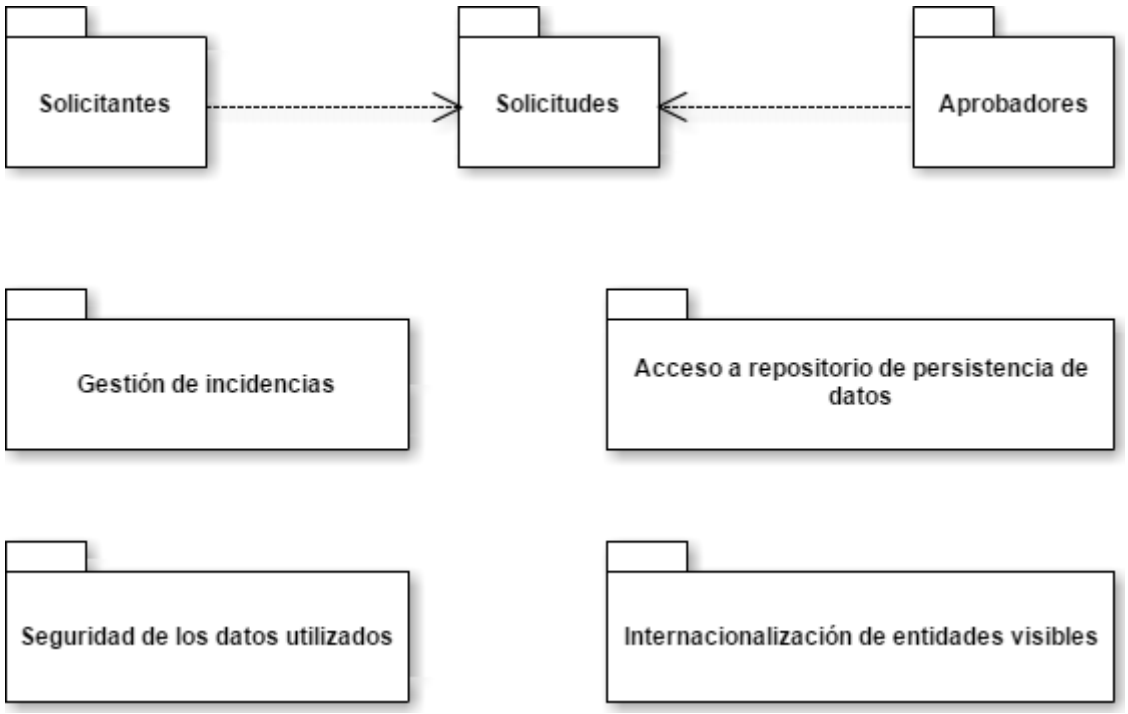


Figura 16. Paquetes de análisis del módulo de gestión de viajes y gastos

4.2.1 Solicitudes

El paquete de análisis “Solicitudes” engloba todo lo relacionado con las solicitudes creadas en el sistema. El paquete engloba los casos de uso “Generación de un informe de una solicitud propia” y “Generación de un informe de una solicitud propia o de colaborador”.

Figura 17. Paquete de análisis "Solicitudes"

4.2.2 Solicitantes

Debido a las diferencias entre solicitantes y aprobadores se ha decidido separar estas dos entidades en dos paquetes diferencias. Se agrupan en este paquete los casos de uso “Visualización de una solicitud propia”, “Creación de una solicitud”, “Envío de una solicitud propia” y “Modificación de una solicitud propia”.

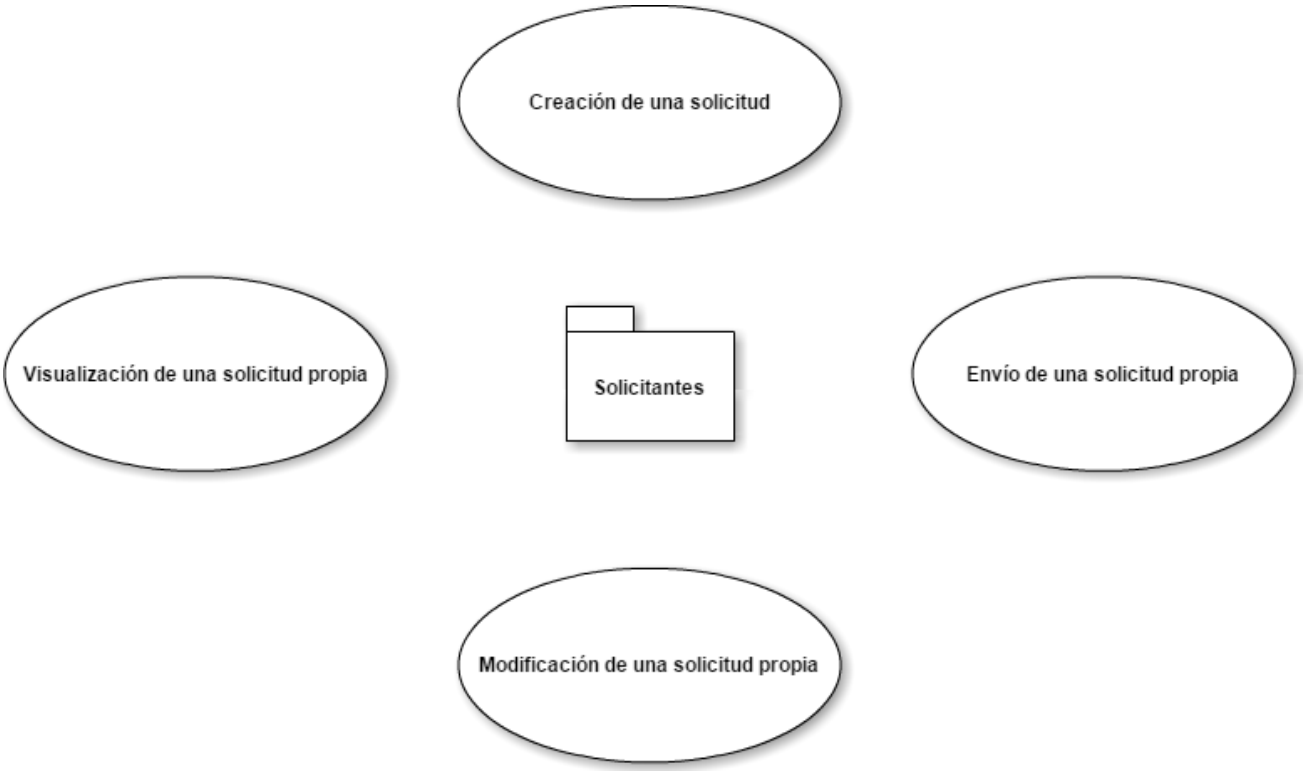


Figura 18. Paquete de análisis "Solicitantes"

4.2.3 Aprobadores

Los aprobadores, tal y como se ha explicado anteriormente, son usuarios de tipo solicitante que poseen permisos adicionales, lo que les permite más interacción con el sistema, y por lo tanto más casos de uso asociados. En él se agrupan los siguientes casos de uso: “Visualización de una solicitud propia o de colaborador”, “Envío de una solicitud propia”, “Creación de una solicitud”, “Modificación de una solicitud propia” y “Aprobación, rechazo o devolución de una solicitud”.

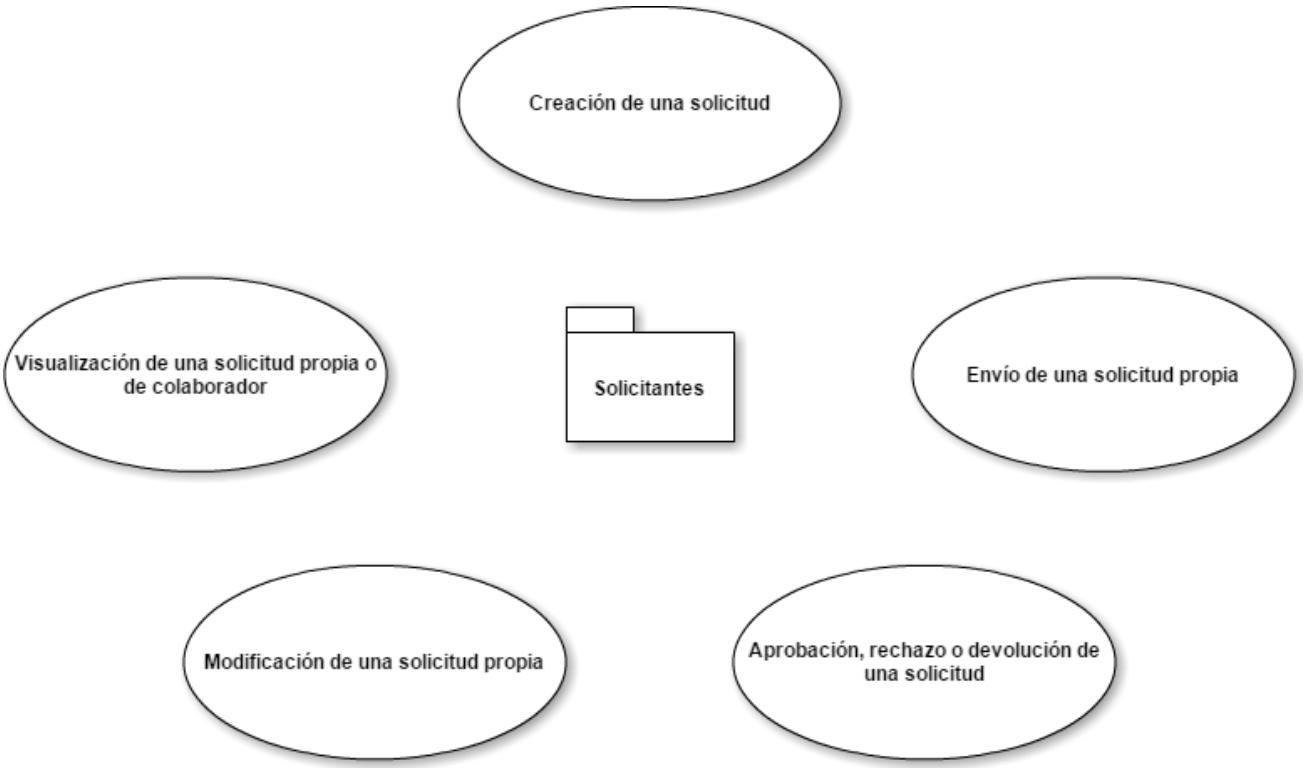


Figura 19. Paquete de análisis "Aprobadores"

4.2.4 Gestión de incidencias

Este paquete de análisis engloba la funcionalidad necesaria para dejar constancia de las incidencias que hubiere en el sistema para su correcta identificación y posterior resolución.

4.2.5 Acceso a repositorio de persistencia de datos

El paquete de análisis “Acceso a repositorio de persistencia de datos” engloba la funcionalidad necesaria para asegurar la persistencia de las entidades del sistema que lo precisen. Este paquete de análisis no surge a partir de la fase de captura de requisitos, sino que proviene de las necesidades identificadas en el propio análisis del sistema, por lo que no contiene ningún caso de uso identificado.

4.2.6 Internacionalización de entidades visibles

Este paquete de análisis engloba la funcionalidad necesaria para que los textos, formato de fecha y resto de elementos que dependan del idioma o de la cultura, puedan ser definidos en función de estos.

4.2.7 Seguridad de los datos utilizados

Este paquete de análisis engloba la funcionalidad necesaria para asegurar la seguridad e integridad de los datos.

5. REQUERIMIENTOS ESPECIALES

En esta sección se describen requerimientos especiales identificados durante la fase de análisis y que son importantes para el sistema. Normalmente, no están referidos a funcionalidad final de cara al usuario, sino que son restricciones o necesidades propias del sistema.

5.1 Persistencia

El sistema a desarrollar debe garantizar la persistencia de algunos objetos de los identificados en la fase de análisis. La definición y especificación de las clases que necesitarán de esta propiedad de persistencia se hará en fases posteriores al análisis. Asimismo, el medio sobre el cual se hará efectiva la capacidad persistente del sistema será concretado y definido en la fase del diseño del sistema.

En esta fase de análisis, se ha definido el paquete 'Acceso a repositorio de persistencia de datos', el cual contendrá la funcionalidad necesaria para garantizar dicha perspectiva.

5.2 Tolerancia a fallos

El sistema debe ser capaz de recuperarse de una acción no permitida, y volver a un estado estable y válido. El paquete de análisis "Gestión de incidencias" será el que contenga la funcionalidad necesaria para que el sistema sea tolerante a fallos.

5.3 Internacionalización

El sistema debe permitir la internacionalización del mismo, es decir, debe ser posible modificar fácilmente todos los textos mostrados por el sistema en función del idioma y/o cultura del usuario. El paquete de análisis "Internacionalización de entidades visibles" será el que contenga la funcionalidad correspondiente a este punto.

5.4 Seguridad

En cualquier proyecto software es un requisito fundamental que el sistema garantice la seguridad de las comunicaciones entre la GUI y el repositorio de almacenamiento, así como la privacidad de los datos. En el caso de este trabajo, este requerimiento se va a llevar a cabo con el uso de un Framework de mapeo de objetos relacionales que garantiza la seguridad, integridad y privacidad de los datos, lo que está incluido en el paquete de análisis "Seguridad de los datos utilizados".

5.5 Disponibilidad del sistema

El sistema debe ser multiusuario, y además, debe acceder a un repositorio de información, por lo que se debe controlar el acceso de cada usuario a la información almacenada, evitando escrituras simultáneas en el repositorio de datos. También se debe garantizar la concurrencia en las interacciones de los usuarios con la aplicación, sin que empeore el rendimiento.

6. BIBLIOGRAFÍA

6.1 Referencias

[R1] I. Jacobson, G. Booch, J. Rambaugh. 2000. “El Proceso Unificado de Desarrollo de Software”. Pearson Education.

[R2] I. Jacobson, G. Booch, J. Rambaugh. 1999. “El lenguaje unificado de modelado. Manual de referencia”. Ed. Addison Wesley.

[R3] Martin Fowler. 1999. “UML Destilled”. Addison-Wesley 1999 (2nd Ed.).

6.2 Referencias web

[W1] http://www.uml.org

[W2] http://www.wikipedia.org

[W3] http://www.rational.com

[W4] http://www.cs.ualberta.ca/~pfiguero/soo/uml/

[W5] http://www.endalia.com

endalia®
HR Consulting & Software

DISEÑO

DESARROLLO DE UN MÓDULO DE GESTIÓN DE VIAJES Y GASTOS

VERSIÓN 1.0
PUBLICADO EL 29/08/2016

HISTÓRICO DE REVISIONES

Fecha	Versión	Descripción	Autor
23/08/2016	1.0	Redacción inicial del documento	Nicolás Bailo Ortiz

ÍNDICE

1. INTRODUCCIÓN

1.1 Propósito del documento

El objetivo del diseño es obtener, a partir del análisis, un punto de partida para actividades de implementación, capturando los requisitos, interfaces y clases a partir de las especificaciones de requisitos y análisis previos.

1.2 Alcance del documento

El alcance del documento comprende toda la fase de diseño del módulo de gestión de viajes y gastos.

1.3 Acrónimos

- ACID: Atomicity-Consistency-Isolation-Durability.
- API: Application Programming Interface.
- ASP: Active Server Pages.
- BCL: Base Class Library.
- CLI: Common Language Infrastructure.
- CLR: Common Language Runtime.
- CLS: Common Language Specification.
- CTS: Common Type System.
- DDL: Data Definition Language.
- DML: Data Manipulation Language.
- ECMA: European Computer Manufacturer Association.
- FTP: File Transfer Protocol.
- FTPS: File Transfer Protocol Segure.
- GDI: Graphics Device Interface.
- GUI: Graphics User Interface.
- HQL: Hibernate Query Language.
- HTTP: HyperText Transfer Protocol.
- HTTPS: HyperText Transfer Protocol Secure.
- IEC: International Electrotechnical Commission.
- IIS: Internet Information Services.
- ISO: International Organization for Standardization.
- JIT: Just-in-Time.
- MSIL: Microsoft Intermediate Language.

- NNTP: Network News Transfer Protocol.
- OBRM/ORM: Object-Relational Mapping.
- SGBD: Sistema Gestor de Bases de Datos.
- SMTP: Simple Mail Transfer Protocol.
- TCP/IP: Transmission Control Protocol / Internet Protocol.
- T-SQL: Transact – Structured Query Language.
- WSDL: Web Services Descriptor Language.
- XML: eXtensive Markup Language.

1.4 Definiciones

- Principio ACID: es el conjunto de propiedades de una base de datos que aseguran la realización de transacciones seguras. En concreto, ACID es un acrónimo de *Atomicity, Consistency, Isolation and Durability* (Indivisibilidad, Consistencia, Aislamiento y Durabilidad en castellano).
    - Indivisibilidad: es la propiedad que asegura que todas las tareas incluidas en la transacción han sido realizadas, o bien que ninguna de ellas lo ha sido.
    - Consistencia: es la propiedad que asegura que la base de datos está en un estado coherente antes del comienzo de la transacción, y que queda en otro estado coherente (sea el mismo u otro) después de la finalización de la transacción.
    - Aislamiento: es la propiedad que asegura que una operación externa a la transacción no puede acceder a un estado intermedio de los producidos durante la misma.
    - Durabilidad: es la propiedad que asegura que una vez realizada la transacción con éxito, ésta persistirá y no se podrá deshacer aunque falle el sistema.
- *Framework*: es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado.
- Mapeo objeto-relacional (ORM): es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos.
- *Tooltip*: es una herramienta de ayuda que funciona al situar o pulsar con el ratón sobre algún elemento gráfico, mostrando una ayuda adicional para informar al usuario de la finalidad del elemento sobre el que se encuentra. Los *tooltip* son una variación de los globos de ayuda y es un complemento muy usado en programación, dado que proporcionan información adicional sin necesidad de que el usuario la solicite.

1.5 Referencias

Este documento contiene referencias a los siguientes documentos del trabajo, contenidos en la sección “Anexos”:

- Especificación de requisitos: documento que describe la especificación de requisitos realizada a partir de las necesidades del cliente.

- Análisis del sistema: documento que describe la arquitectura del sistema desarrollado, junto con un listado de casos de uso y su estructura de paquetes.

1.6 Resumen

En este documento se describe el proceso de diseño del módulo de gestión de viajes y gastos. Se compone de ocho apartados:

1. Introducción del documento, definición del propósito y alcance del mismo.
2. Se describe el proceso de diseño seguido para la confección de este documento.
3. Se describen las decisiones y restricciones iniciales del diseño, y se comparan y describen diferentes alternativas.
4. Descripción de la arquitectura del sistema tanto a nivel físico como a nivel de organización del mismo.
5. Detalle de las clases identificadas.
6. Detalle del diseño de la base de datos.
7. Prototipado de la interfaz de las diferentes pantallas con la que ha de interactuar el usuario del sistema.
8. Bibliografía y referencias Web utilizadas durante esta fase del proyecto.

2. DESCRIPCIÓN DEL PROCESO

Partiendo de las entidades, casos de uso y paquetes identificados en la fase de análisis y de los requisitos del sistema identificados en la especificación de requisitos, se procede a la descripción de la arquitectura del sistema, para su posterior implementación. A partir de los casos de uso, se identifican las necesidades de interacción entre el usuario y el sistema, y se define la estructura de la interfaz, así como los prototipos del mismo. A partir de las entidades se describen las tablas necesarias en base de datos, así como sus clases de acceso a datos.

3. CONSIDERACIONES INICIALES

3.1 Introducción

En esta sección se describen las primeras decisiones y especificaciones de diseño del módulo de gestión de viajes y gastos, que sirven como base para el diseño del resto del sistema. Asimismo, se describen las principales características de las distintas tecnologías utilizadas.

3.2 Especificaciones tecnológicas

El trabajo que nos ocupa se lleva a cabo en el marco de la empresa Endalia. Este hecho condiciona la tecnología a utilizar que, evidentemente, debe ser la usada en el resto de aplicaciones llevadas a cabo en dicha empresa, con el objeto de hacerlas compatibles y fácilmente integrables.

Es por esto que el módulo de gestión de viajes y gastos se va a desarrollar en la plataforma .NET de Microsoft, con ASP en las páginas Web, usando el lenguaje de programación C# y el gestor de base de datos Microsoft SQL Server.

3.3 Especificaciones de diseño

A partir de los requisitos identificados en el documento de análisis de requisitos, podemos definir las siguientes características necesarias del sistema:

- Acceso a base de datos de forma transaccional, cumpliendo el principio ACID.
- Escalabilidad: el sistema debe soportar más carga de trabajo sin necesidad de modificar el software.
- Extensibilidad: el sistema debe soportar la adición de nuevos componentes y funcionalidades sin que ello afecte al resto de componentes.
- Usabilidad: el sistema debe poder ser manejado de forma intuitiva.
- Seguridad: el sistema debe ser fiable, tanto a nivel de autenticación, como de autorización, y debe mantener la privacidad de la información confidencial.
- Rendimiento: el sistema debe soportar un incremento en la carga de trabajo sin que ello repercuta notablemente en el usuario.

3.4 Plataforma .NET

Microsoft .NET es, de acuerdo con la definición de Microsoft, una plataforma que comprende servidores, clientes y servicios. Consiste en un conjunto de aplicaciones como Visual Studio .NET, los servicios .NET, etc. Esta plataforma es una implementación basada en estándares abiertos como SOAP, WSDL o C#. Desde el punto de vista del programador, el entorno .NET ofrece un solo entorno de desarrollo para todos los lenguajes que soporta (por ejemplo, Visual Basic, C#, C++, Visual J#, Fortran, Cobol…).

3.4.1 .NET Framework

El "*framework*" o marco de trabajo constituye la base de la plataforma .NET (Figura 1), y denota la infraestructura sobre el cual se reúnen un conjunto de lenguajes, herramientas y servicios que simplifican el desarrollo de aplicaciones en un entorno de ejecución distribuido.

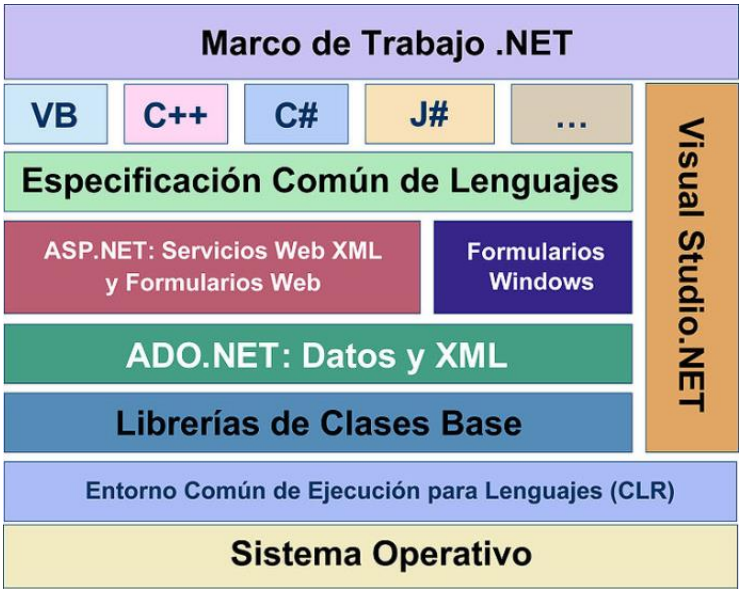


Figura 1. .NET Framework

Bajo el nombre .NET Framework, o Marco de Trabajo .NET, se encuentran reunidas una serie de normas, entre las cuales se encuentran:

- La norma que define las reglas que debe seguir un lenguaje de programación para ser considerado compatible con el marco de trabajo .NET (ECMA-335, ISO/IEC 23271). Por medio de esta norma se garantiza que todos los lenguajes desarrollados para la plataforma ofrezcan al programador un conjunto mínimo de funcionalidad y compatibilidad con todos los demás lenguajes de la plataforma.
- La norma que define el lenguaje C# (ECMA-334, ISO/IEC 23270). Éste es el lenguaje insignia del marco de trabajo .NET y pretende reunir las ventajas de lenguajes como C/C++ y Visual Basic en un solo lenguaje.
- La norma que define el conjunto de funciones que debe implementar la librería de clases base (BCL, siglas en inglés) (incluido en ECMA-335, ISO/IEC 23271). Tal vez el más importante de los componentes de la plataforma, esta norma define un conjunto funcional mínimo que debe implementarse para que el marco de trabajo sea soportado por un sistema operativo. Aunque Microsoft implementó esta norma para su sistema operativo Windows, la publicación de la norma abre la posibilidad de que sea implementada para cualquier otro sistema operativo existente o futuro, permitiendo que las aplicaciones corran sobre la plataforma, independientemente del sistema operativo para el cual hayan sido implementadas.

Los principales componentes del marco de trabajo son:

- El conjunto de lenguajes de programación.
- La Biblioteca de Clases Base o BCL.
- En entorno Común de Ejecución para Lenguajes o CLR.

Debido a la publicación de la norma para la infraestructura común de lenguajes (CLI por sus siglas en inglés), el desarrollo de lenguajes se facilita, por lo que el marco de trabajo .NET soporta ya más de 20 lenguajes de programación (C#, Visual

Basic, C++, Perl, Python, Fortran…) y es posible desarrollar cualquiera de los tipos de aplicaciones soportados en la plataforma con cualquiera de ellos.

3.4.2 Common Language Runtime

El CLR (Figura 2) es el verdadero núcleo del Framework de .NET, entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes, ampliando el conjunto de servicios del sistema operativo.

Figura 2. Common Language Runtime (CLR)

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .NET en un código intermedio (MSIL, *Microsoft Intermediate Language*), similar al *BYTECODE* de Java. Para generar dicho código, el compilador se basa en el *Common Language Specification* (CLS), que determina las reglas necesarias para crear ese código MSIL compatible con el CLR.

Para ejecutarse se necesita un segundo paso: un compilador JIT (*Just-in-Time*), que es el que genera el código máquina real que se ejecuta en la plataforma del cliente. De esta forma se consigue con .NET independencia de la plataforma hardware. La compilación JIT la realiza el CLR a medida que el programa invoca métodos. El código ejecutable obtenido se almacena en la memoria caché del ordenador, siendo recompilado de nuevo solo en el caso de producirse algún cambio en el código fuente.

3.4.3 Biblioteca de Clases Base (BCL)

La Biblioteca de Clases Base (*Base Class Library*, BCL) proporciona las clases básicas predefinidas que manejan la mayoría de las operaciones que se encuentran involucradas en el desarrollo de aplicaciones, incluyendo entre otras:

- Interacción con los dispositivos periféricos.
- Manejo de datos (ADO.NET).
- Administración de memoria.
- Cifrado de datos.

- Transmisión y recepción de datos por distintos medios (XML, TCP/IP).
- Administración de componentes Web que corren tanto en el servidor como en el cliente (ASP.NET).
- Manejo y administración de excepciones.
- Manejo del sistema de ventanas.
- Herramientas de despliegue de gráficos GDI.
- Herramientas de seguridad e integración con la seguridad del sistema operativo.
- Manejo de tipos de datos unificado.
- Interacción con otras aplicaciones.
- Manejo de cadenas de caracteres y expresiones regulares.
- Operaciones aritméticas.
- Manipulación de archivos de imágenes.
- Aleatoriedad.
- Generación de código.
- Manejo de idiomas.
- Auto descripción de código.
- Interacción con el API Win32 o Windows API.
- Compilación de código.

Esta funcionalidad se encuentra organizada por medio de espacios de nombre jerárquicos en lo que se denomina *Namespace*, como se puede apreciar en la siguiente figura (Figura 3):

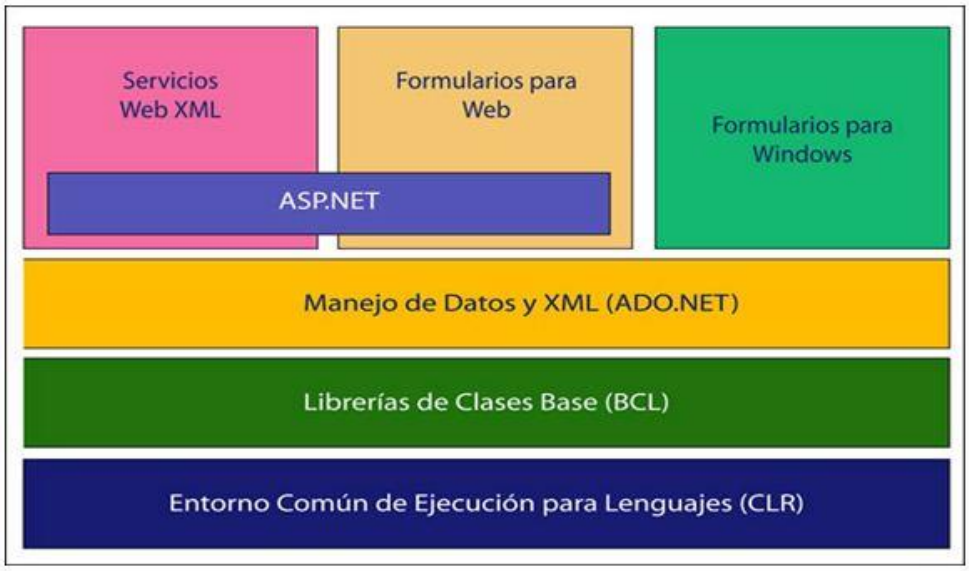


Figura 3. Esquema de distribución jerárquica de Namespaces

3.5 SQL Server

Es un SGBD para bases de datos relacionales desarrollado por Microsoft. Sus principales características son:

- Soporte para transacciones (bajo el principio ACID).
- Escalabilidad.
- Estabilidad.
- Seguridad.
- Soporta procedimientos almacenados.
- Entorno gráfico que permite ejecutar comandos DDL y DML.
- T-SQL como lenguaje de consultas nativo.

Una base de datos de SQL Server es una colección de tablas con columnas de un tipo definido, más otros objetivos como restricciones, vistas, procedimientos almacenados o índices. El espacio de almacenamiento está dividido en páginas de 8KB, que es la unidad básica de entrada-salida para una operación de SQL Server. Las filas de cada tabla se almacenan físicamente en fichero o bien en un montículo (*heap*) o bien en un árbol-B. Los índices (que son estructuras para acelerar el acceso a datos en las consultas) definidos son almacenados siempre en árboles-B. Hay dos tipos de índices en SQL Server:

- Índices agregados, en los que se almacenan los datos de la fila indexada en las hojas del árbol.
- Índices no agregados, que en las hojas del árbol-B almacenan una referencia a la hoja del índice agregado correspondiente, o bien una referencia a la página correspondiente.

Sólo puede haber un índice agregado por tabla, y éste habitualmente es la clave primaria de ésta.

3.6 NHibernate

NHibernate es un Framework de Mapeo objeto-relacional (ORM de sus siglas en inglés) para la plataforma .NET y el lenguaje C#. Facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación mediante archivos declarativos (XML). NHibernate es software libre y soporta los motores de base de datos más habituales del mercado: MySQL, PostgreSQL, Oracle, MS SQL Server, etc.

NHibernate basa su configuración en un fichero XML en el que se le indica información sobre los recursos a mapear así como diversas opciones de configuración:

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
    <session-factory>
        <property name="dialect">NHibernate.Dialect.MsSql2005Dialect</property>
        <property name="connection.provider">
NHibernate.Connection.DriverConnectionProvider</property>
        <property name="connection.isolation">ReadUncommitted</property>
        <property name="connection.driver_class">
NHibernate.Driver.SqlClientDriver</property>
        <property name="connection.connection_string_name">conn</property>
        <property name="default_schema">dbo</property>
        <property name="adonet.batch_size">10</property>
        <property name="show_sql">false</property>
        <property name="proxyfactory.factory_class">
NHibernate.ByteCode.Castle.ProxyFactoryFactory, NHibernate.ByteCode.Castle</property>
        <mapping assembly="IntegraDBAccess"/>
    </session-factory>
</hibernate-configuration>
```

La utilización de NHibernate nos permite la creación de dos *Namespaces* diferentes para gestionar el acceso a los datos. El primero, llamado *Domain*, contendrá los ficheros de mapeo de NHibernate así como los ficheros de representación de las diferentes entidades mapeadas en forma de objetos. El segundo *Namespace*, llamado *DataAccess*, contiene las clases donde se encuentran los métodos para acceder a los datos.

El mapeo de una entidad guardada en la base de datos se realiza mediante un fichero de mapeo (en formato XML), dónde se definen los atributos y las relaciones de dicha entidad, tal y como se muestra en el siguiente ejemplo:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="IntegraDBAccess"
namespace="integradbaccess.Domain">
  <class name="Orh_Trip_TripRequest" table="Orh_Trip_TripRequests" dynamic-
update="true" >
    <id name="ID" type="int" column="pk_TripRequestId">
      <generator class="identity"/>
    </id>
    <property name="Name" column="TripRequestName" not-null="false" />
    <property name="Code" column="TripRequestCode" not-null="true" />
    <property name="Description" column="TripRequestDescription" not-null="false" />
    <property name="Status" column="TripRequestStatus" not-null="false" />
    <property name="BeginDate" column="TripRequestBeginDate" not-null="true" />
    <property name="EndDate" column="TripRequestEndDate" not-null="true" />
    <many-to-one name="Employee" class="Orh_Emp_Employee" >
      <column name="fk_TripRequestEmployeeId" />
    </many-to-one>
    <many-to-one name="Project" class="Prj_ActionPlanItem" >
      <column name="fk_TripRequestProjectId" />
    </many-to-one>
  </class>
</hibernate-mapping>
```

Por otro lado, NHibernate ofrece su propio lenguaje de consultas, el *Hibernate Query Language* (HQL). Permite al programador escribir consultas similares a SQL directamente contra los objetos de datos mapeados. Permite modificar los objetos y añadir restricciones sobre éstos. Además, se caracterizan por ser consultas independientes de la base de datos utilizada.

3.7 IIS

Internet Information Services, desarrollado por Microsoft, es un conjunto de servicios sobre TCP/IP que permiten a la máquina donde está alojado proporcionar funcionalidad de servidor de aplicaciones. Actualmente soporta los siguientes protocolos: HTTP, HTTPS, FTP, FTPS, SMTP y NNTP.

Como servidor web, permite definir una serie de directorios virtuales, donde se encuentra cada una de las aplicaciones indexadas por él, resolviendo las peticiones de páginas estáticas, y gestionando las necesidades de código dinámico de las mismas. Asimismo, proporciona mecanismos de seguridad y autenticación de usuarios, donde permite una total integración con el *Active Directory* de Microsoft Windows.

4. DISEÑO DE LA ARQUITECTURA

4.1 Introducción

En esta sección se describe la estructura del sistema, tanto desde el punto de visto físico como lógico, detallando las decisiones que se han tomado en dichos ámbitos, así como las restricciones que estaban impuestas por la organización en la que se desarrollaba el módulo de gestión de viajes y gastos.

4.2 Estructura general del sistema

La estructura general del sistema está definida por la plataforma de otros servicios web de Endalia. Por lo tanto, la estructura del proyecto debe adaptarse a la de estos servicios. A continuación se detallan las restricciones que deben mantenerse, referente a la estructura:

- Servidor IIS: servidor de aplicaciones donde se aloja la plataforma de servicios web. Resuelve peticiones (http en este caso, pero también de otros tipos) y las asigna a su aplicación correspondiente, en función de los permisos previamente establecidos.
- Base de datos: repositorio de datos de la aplicación, implementado desde el sistema gestor de bases de datos SQL Server 2012.

4.3 Estructura de capas del sistema

Una posible forma de organizar un sistema de un tamaño considerable es agrupando funcionalidades que comparten la misma naturaleza, funcionalidad y estructura en capas. De esta manera, se consigue que cambios en algún componente (por ejemplo, presentación) no afecten al resto de elementos del sistema; lo que proporciona una óptima escalabilidad y ayuda a mejorar el rendimiento.

El sistema está basado en la siguiente arquitectura multicapa típica:

- Capa de cliente: formada por los componentes que se ejecutan en el cliente. En nuestro caso es el navegador web desde el que se accede a la plataforma y el código JavaScript que se ejecuta en algunas páginas de la aplicación.
- Capa de presentación. Es la capa que se encarga de crear el interfaz gráfico que renderiza la capa de cliente y de gestionar las interacciones de éste con el sistema. Esto se corresponde en el sistema con las páginas aspx y su código subyacente.
- Capa de lógica de negocio. Contiene los objetos que representan los datos almacenados en el repositorio de datos, así como la lógica necesaria para procesarlos. En nuestro sistema se corresponde con la capa de acceso a datos.
- Capa de integración. Contiene objetos que automatizan el acceso a datos. Esto se corresponde con los procedimientos almacenados en la base de datos.
- Capa de datos. Contiene los sistemas de información de la aplicación, habitualmente una base de datos. En nuestro sistema se corresponde con la base de datos y con los archivos de recursos.

4.4 Estructura de subsistemas

Los subsistemas son un medio para organizar el modelo en partes más pequeñas y manejables. Una de las opciones para realizar esta actividad se basa en la identificación de subsistemas de diseño a partir de los paquetes definidos en la fase de análisis. La correspondencia no siempre debe ser uno a uno, ya que intervienen ciertos condicionantes que la limitan, pero sí constituye un punto de partida para iniciar la identificación.

En los siguientes apartados se enumeran y explican los subsistemas de diseño que forman la aplicación.

4.4.1 Subsistema de solicitudes

En este subsistema se realizan las funcionalidades propias de las solicitudes de viajes y gastos, como la realización de informes relativos a los gastos de una solicitud.

4.4.2 Subsistema de solicitantes

En este subsistema se realizan las funcionalidades propias de los solicitantes de viajes y gastos, como la creación de solicitudes, su modificación, visualización y envío.

4.4.3 Subsistema de aprobadores

En este subsistema se realizan las funcionalidades propias de los aprobadores de viajes y gastos. Estas son, además de las funcionalidades propias de los solicitantes, las de aprobar, rechazar o devolver solicitudes.

4.4.4 Subsistema de gestión de incidencias

Subsistema en el cual se desarrolla la funcionalidad identificada en el paquete “Gestión de incidencias”, la relacionada con el control de los eventos y errores producidos por el sistema.

4.4.5 Subsistema de acceso a base de datos

En este subsistema se desarrollan las funcionalidades necesarias para gestionar el acceso a base de datos de manera transparente y eficiente.

4.4.6 Subsistema de internacionalización de entidades visibles

En este subsistema se desarrollan las funcionalidades necesarias para que la aplicación pueda adaptar sus contenidos de manera automática a la cultura en la que se ejecute.

4.4.7 Subsistema de seguridad de datos utilizados

En este subsistema se desarrollan las funcionalidades necesarias para que la aplicación sea segura y consiga una integridad completa de los datos.

5. CLASES DEL SISTEMA

5.1 Introducción

En este apartado se detallan las clases del módulo de gestión de viajes y gastos. Estas clases se dividen en cuatro grupos:

- Clases de interfaz: son las encargadas de crear la GUI y gestionar las interacciones del usuario con el sistema.
- Clases de acceso a datos: son las encargadas de gestionar la persistencia de los datos del sistema y la interacción con la base de datos.
- Clases de mapeo objeto-relacional de datos: son las encargadas de crear una representación de los datos persistentes en la base de datos en forma de objetos utilizables desde el código de la aplicación.
- Clases auxiliares.

5.2 Clases de interfaz

En esta sección se van a analizar las clases de interfaz de la aplicación. Como ya se ha comentado, el proyecto se desarrolla en .NET y el interfaz en ASP. La estructura básica de la página web que contiene cabecera y menús está contenida en la página "*master*". Esto permite la creación del resto de las páginas del sistema como un contenido de los master, y por lo tanto la reutilización de la estructura base, permitiendo la realización de modificaciones en la estructura básica de forma rápida y sencilla.

También facilita la adaptación de la aplicación para otra organización distinta a la que se ha destinado en este trabajo.

A continuación se comentan las distintas clases de interfaz agrupándolas por subsistemas.

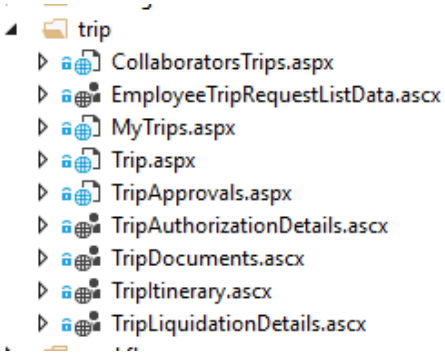


Figura 4. Clases de interfaz del módulo de gestión de viajes y gastos

5.2.1 Clases del subsistema de solicitudes

- Trip.aspx: formulario que contiene información sobre una solicitud de viaje o gasto.
- TripAuthorizationDetails.ascx: formulario que contiene información sobre la fase de autorización previa de una solicitud de viaje o gasto.
- TripLiquidationDetails.ascx: formulario que contiene información sobre la fase de liquidación de una solicitud de viaje o gasto.
- TripItinerary.ascx: formulario que contiene información sobre los itinerarios asociados a una solicitud de viaje.
- TripDocuments.ascx: formulario que contiene información sobre los documentos adjuntos a una solicitud de viaje o gasto.

5.2.2 Clases del subsistema de solicitantes

- MyTrips.aspx: formulario que contiene información sobre las solicitudes realizadas por el usuario registrado.
- EmployeeTripRequestListData.ascx: formulario que presenta un listado de solicitudes en función de ciertos parámetros de entrada.

5.2.3 Clases del subsistema de aprobadores

- CollaboratorsTrips.aspx: formulario que contiene información sobre las solicitudes realizadas por los usuarios colaboradores del usuario registrado.
- TripApprovals.aspx: formulario que contiene información sobre las solicitudes pendientes de aprobar del usuario registrado.

5.3 Clases de acceso a datos

En esta sección se van a analizar las clases de acceso a datos del módulo de gestión de viajes y gastos. En la siguiente figura (Figura 5), se puede observar el listado completo, aunque únicamente se detallarán las más destacadas:

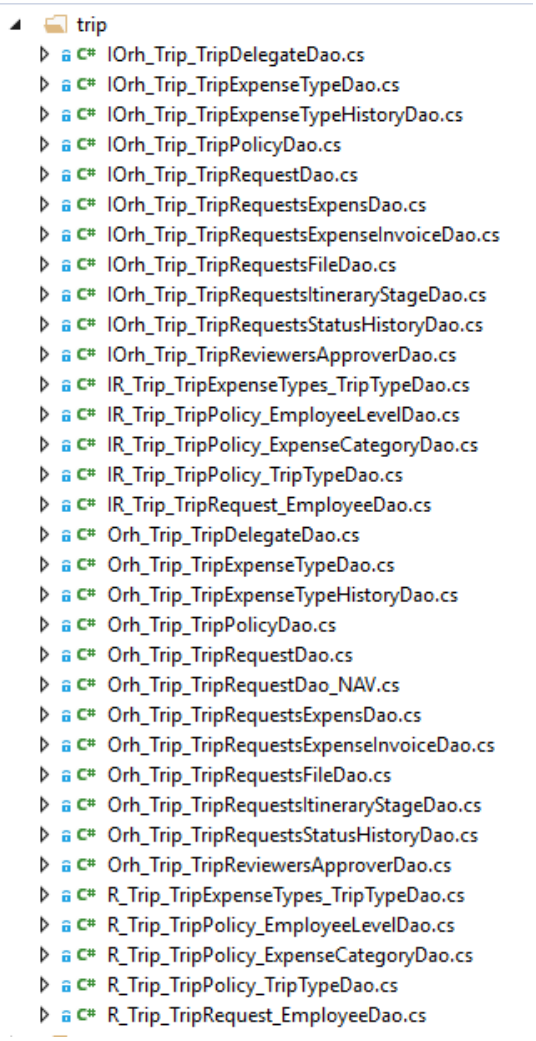


Figura 5. Clases de acceso a datos del módulo de gestión de viajes y gastos.

- Orh_Trip_TripRequestDao.cs: es la clase de acceso a las solicitudes de viajes o gastos.
- Orh_Trip_TripRequestsExpenseDao.cs: es la clase de acceso a los gastos internos asociados a las solicitudes de viajes o gastos.
- Orh_Trip_TripRequestsFileDao.cs: es la clase de acceso a los documentos asociados a las solicitudes de viajes o gastos.
- Orh_Trip_TripRequestsItineraryStageDao.cs: es la clase de acceso a los itinerarios asociados a las solicitudes de viajes o gastos.
- Orh_Trip_TripExpenseTypeDao.cs: es la clase de acceso a los diferentes tipos de gastos disponibles.
- Orh_Trip_TripRequestsExpenseInvoiceDao.cs: es la clase de acceso a las facturas asociadas a los diferentes gastos dentro de las solicitudes de viajes o gastos.

5.4 Clases de mapeo objeto-relacional de datos

En esta sección se van a analizar las clases de mapeo objeto-relacional del módulo de gestión de viajes y gastos. En la siguiente figura (Figura 6), se puede observar el listado completo, aunque únicamente se detallarán las más destacadas:

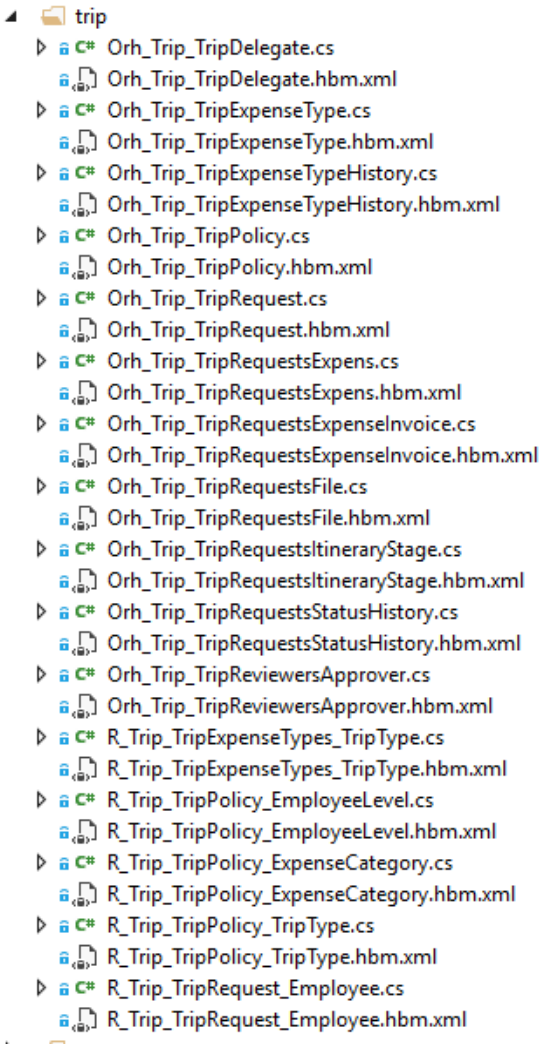


Figura 6. Clases de mapeo objeto-relacional de datos del módulo de gestión de viajes y gastos

- Orh_Trip_TripRequest.cs y Orh_Trip_TripRequest.hbm.xml: son las clases que representan las solicitudes de viajes y gastos.
- Orh_Trip_TripRequestsExpens.cs y Orh_Trip_TripRequestsExpens.hbm.xml: son las clases que representan los diferentes gastos internos asociados a las solicitudes de viajes y gastos.
- Orh_Trip_TripRequestsFile.cs y Orh_Trip_TripRequestsFile.hbm.xml: son las clases que representan los documentos adjuntos a las solicitudes de viajes y gastos.
- Orh_Trip_TripRequestsItineraryStage.cs y Orh_Trip_TripRequestsItineraryStage.hbm.xml: son las clases que representan los itinerarios asociadas a las solicitudes de viajes.
- Orh_Trip_TripExpenseType.cs y Orh_Trip_ TripExpenseType.hbm.xml: son las clases que representan los diferentes tipos de gastos disponibles.
- Orh_Trip_TripRequestsExpenseInvoice.cs y Orh_Trip_TripRequestsExpenseInvoice.hbm.xml: son las clases que representan las facturas asociadas a los diferentes gastos dentro de las solicitudes de viajes y gastos.

5.5 Clases auxiliares

Las clases auxiliares son un tipo de clases con diferentes utilidades sobre los datos del sistema. En esta tipología se engloban desde las clases genéricas de los *Frameworks* utilizados hasta clases para encriptación de datos que, por motivos de seguridad y confidencialidad, no se detallarán.

6. DISEÑO DE LA BASE DE DATOS

6.1 Introducción

En este apartado se presenta una vista global del diseño de la base de datos, resultante de las entidades identificadas en el análisis, adecuando al repositorio de datos que utiliza la aplicación. Como ya se ha comentado anteriormente, el repositorio de datos a utilizar es una base de datos relacional gestionada desde el SGBD MSSQL Server 2012.

También se describen las tablas de la base de datos junto con sus atributos y las relaciones existentes entre ellas.

6.2 Diseño general de la base de datos

Debido al tamaño de la base de datos, se va a separar en subconjuntos de menor tamaño, ya que no sería posible representarlo de otro modo en este formato de documento. Ya que el elemento principal de este módulo son las solicitudes de viajes y gastos, se dividirá en sus respectivas fases. Así, el primer subconjunto se corresponde a la autorización previa de una solicitud, y el segundo con la liquidación de la misma.

6.2.1 Autorización previa

En primer lugar cabe destacar que todas las solicitudes están relacionadas con un empleado de la tabla *Orh_Emp_Employees*, lo que indica quien ha sido su creador.

En la fase de autorización previa, se podrán adjuntar documentos relacionados con el viaje o gasto (*Orh_Trip_TripRequestsFiles*) o diferentes itinerarios, si la solicitud se trata de un viaje (*Orh_Trip_TripRequestsItineraryStages*).

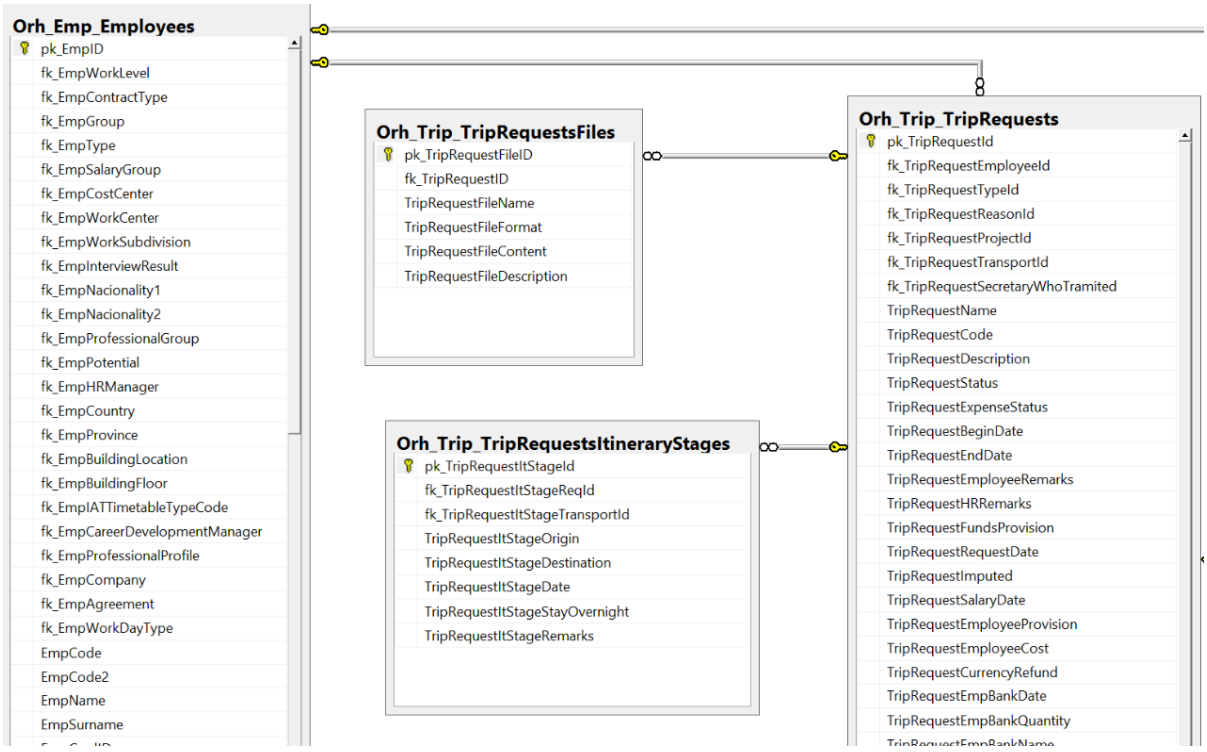


Figura 7. Esquema general de los datos implicados en la autorización previa de una solicitud.

6.2.2 Liquidación

En la fase de liquidación una solicitud se relaciona con un listado de gastos (*Orh_Trip_TripRequestExpenses*) de diferentes tipos (*Orh_Trip_TripExpenseTypes*). Además, cada gasto asociado a una solicitud puede contener una factura adjunta (*Orh_Trip_TripRequestsExpenseInvoices*). Por último, el sistema guarda un histórico de las diferentes tipologías de gastos que se han creado (*Orh_Trip_TripExpenseTypeHistory*).

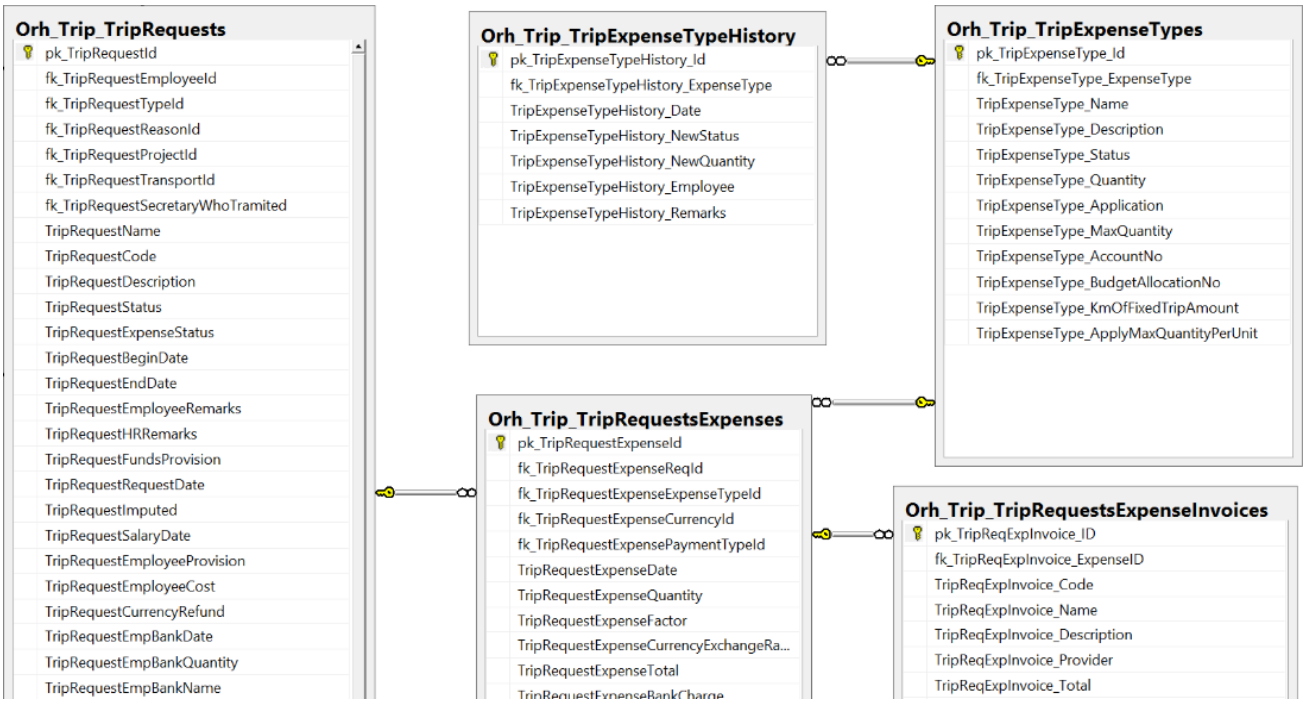


Figura 8. Esquema general de los datos implicados en la liquidación de una solicitud.

7. PROTOTIPADO DE LA INTERFAZ

7.1 Introducción

En este apartado se muestran los prototipos de interfaz de pantalla diseñados para el sistema, teniendo en cuenta el interfaz de la web corporativa de la organización a la que va destinado este proyecto realizado dentro de Endalia.

Los objetivos y decisiones tomadas en este punto son:

- La resolución mínima para la que se diseñará la aplicación es 1024x768.
- Primar la claridad y la usabilidad por encima de otros aspectos, proporcionando un entorno claro e intuitivo para los usuarios.
- Evitar la aparición de barras de desplazamiento horizontal, y limitar dentro de lo posible el uso de la barra de desplazamiento vertical, ocupando a poder ser únicamente la pantalla visible.

7.2 Interfaz básico

A continuación se muestra la base del diseño del interfaz gráfico, enmarcando las diferentes secciones que lo componen (Figura 9).

Los aspectos más significativos son los siguientes:

- Cabecera: debe ocupar la parte superior de la pantalla. El diseño de la misma es el utilizado por la organización en su web corporativa.
- Menú: se trata de un menú desplegable, situado en el lateral izquierdo.
- Contenido: la parte central de la página.

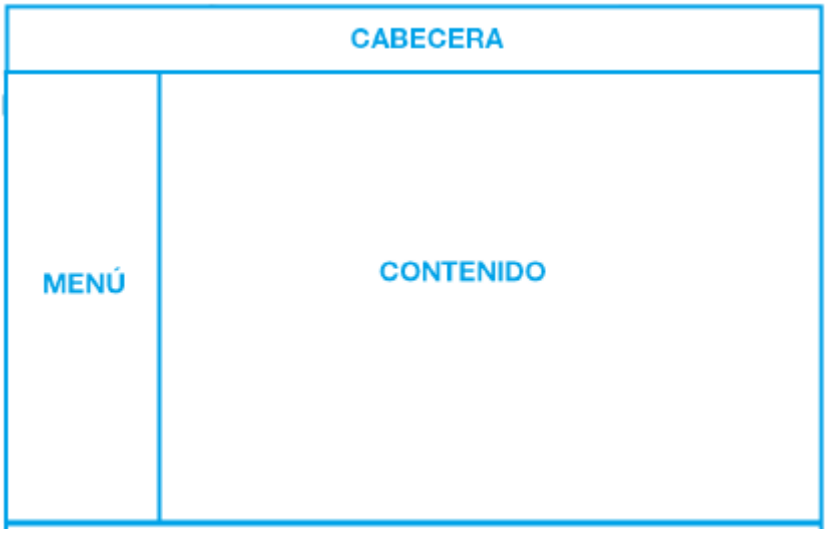


Figura 9. Prototipo de interfaz básico

7.3 Interfaz del contenido de la página de Mis Viajes y Gastos

A continuación se muestra el prototipo de la página de Mis Viajes y Gastos, que contiene un listado de las solicitudes creadas por el usuario (Figura 10):

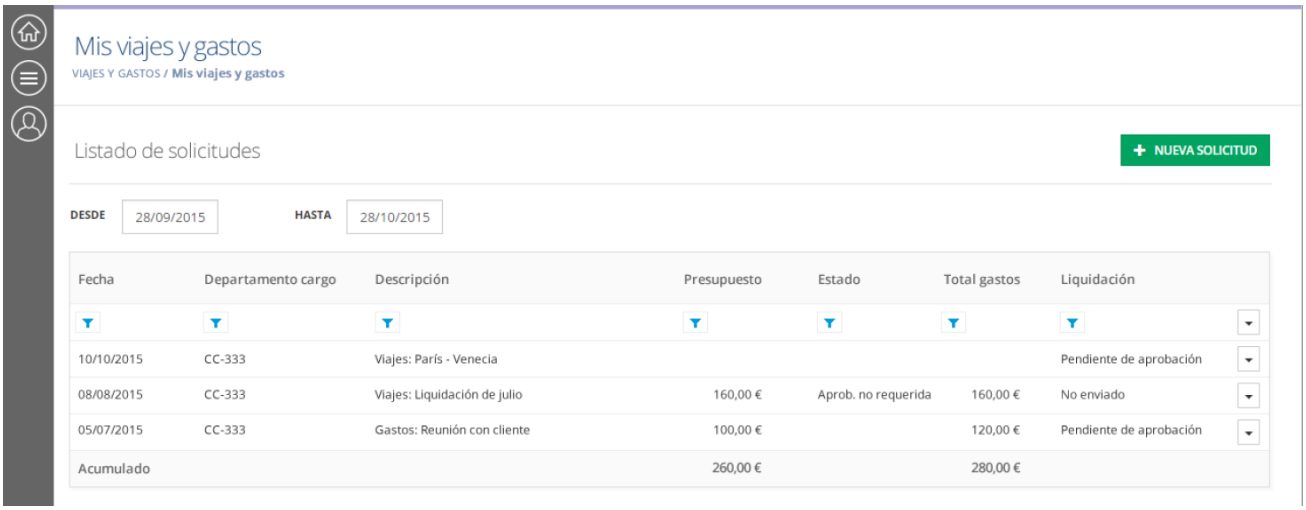


Figura 10. Prototipo de la página de Mis Viajes y Gastos.

7.4 Interfaz del contenido de la página de Viajes y Gastos de Colaboradores

A continuación se muestra el prototipo de la página de Viajes y Gastos de Colaboradores, que contiene un listado de empleados que son colaboradores del usuario (Figura 11), y que al acceder a uno de esos empleados muestra un listado de las solicitudes creadas por dicho empleado (Figura 12):

Figura 11. Prototipo de la página de Viajes y Gastos de colaboradores: listado de colaboradores.

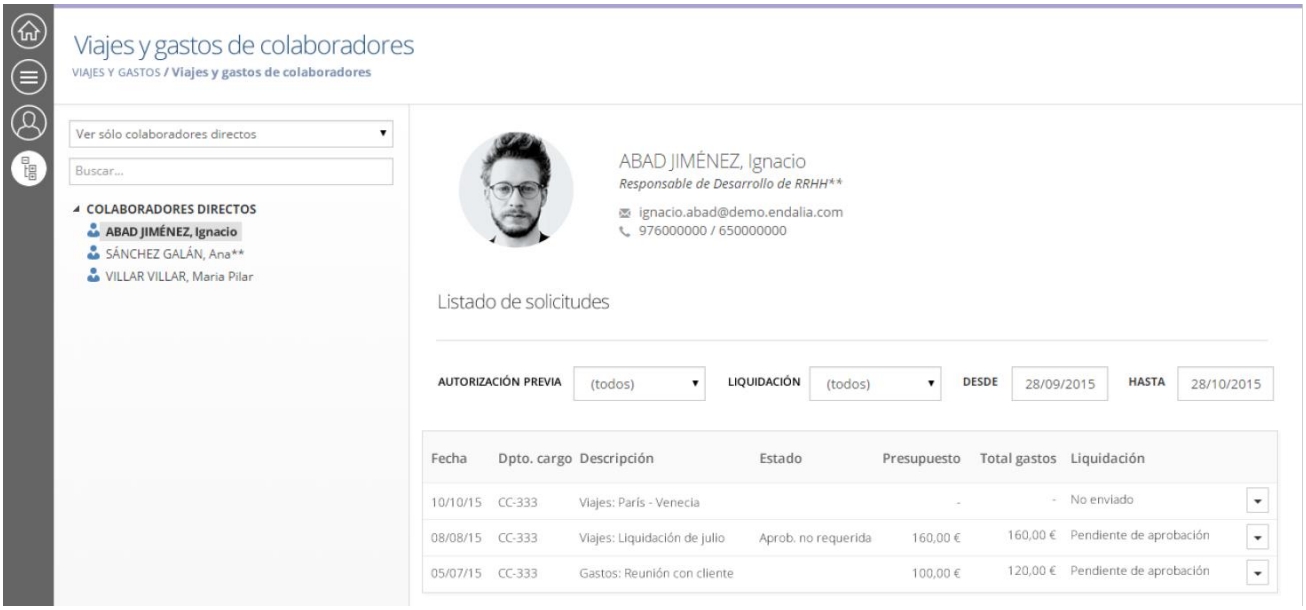


Figura 12. Prototipo de la página de Viajes y Gastos de colaboradores: solicitudes de colaborador.

7.5 Interfaz del contenido de la página de Solicitudes Pendientes de Aprobación

A continuación se muestra el prototipo de la página de Solicitudes Pendientes de Aprobación, que contiene un listado de las solicitudes que el usuario registrado tiene pendientes de aprobar (Figura 13):

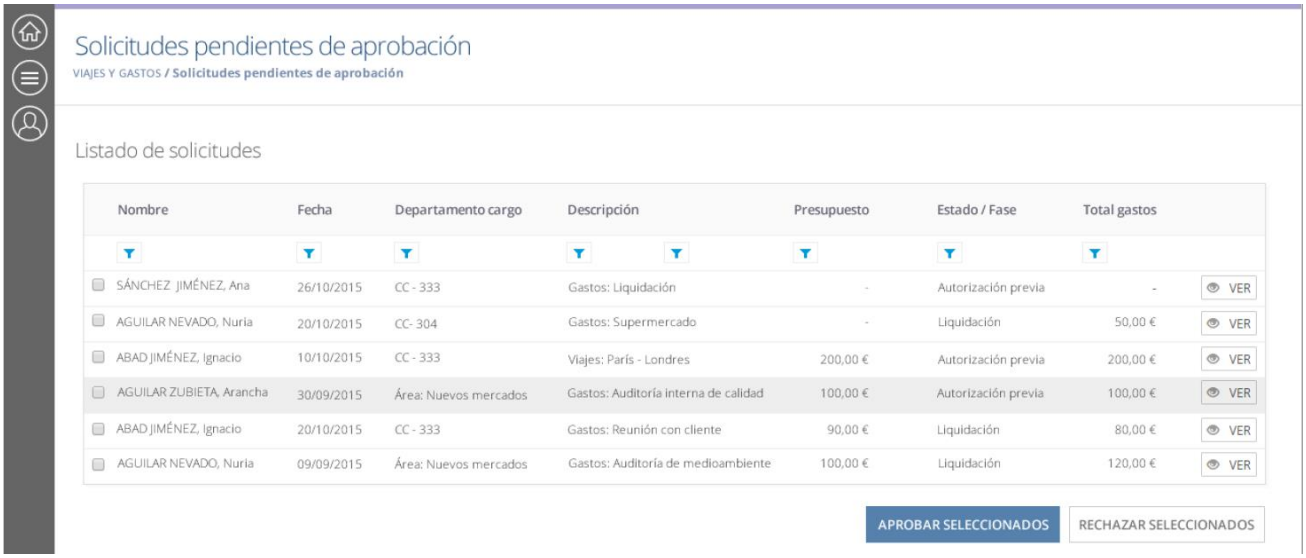


Figura 13. Prototipo de la página de Solicitudes Pendientes de Aprobación.

7.6 Interfaz del contenido de la página de autorización previa de una solicitud

A continuación se muestra el prototipo de la página de autorización previa de una solicitud, que contiene un formulario con la información relativa a dicha fase de una solicitud (Figura 14):

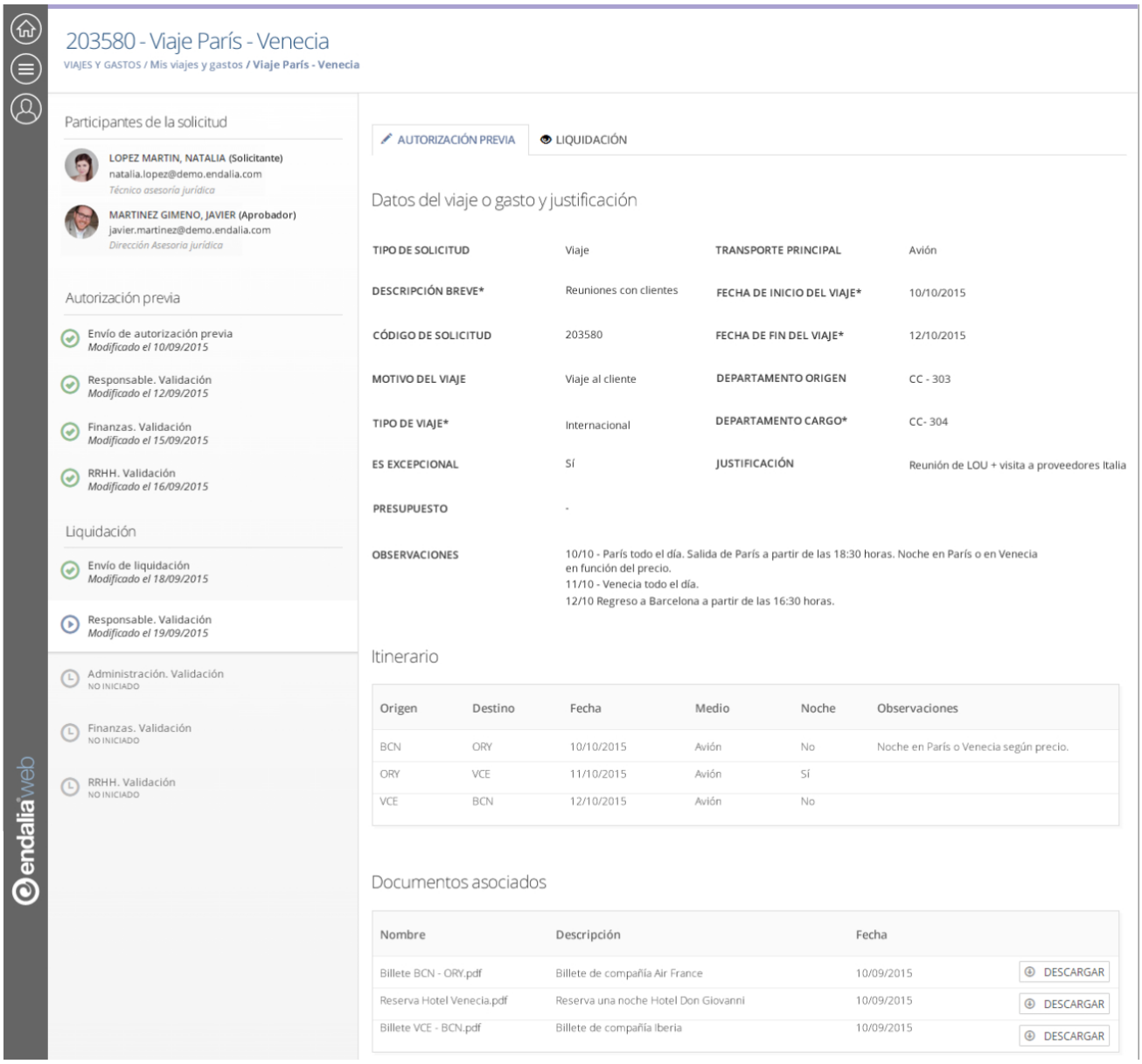


Figura 14. Prototipo de la página de autorización previa de una solicitud.

7.7 Interfaz del contenido de la página de liquidación de una solicitud

A continuación se muestra el prototipo de la página de autorización previa de una solicitud, que contiene un desglose con los diferentes gastos asociados a la solicitud (Figura 15):

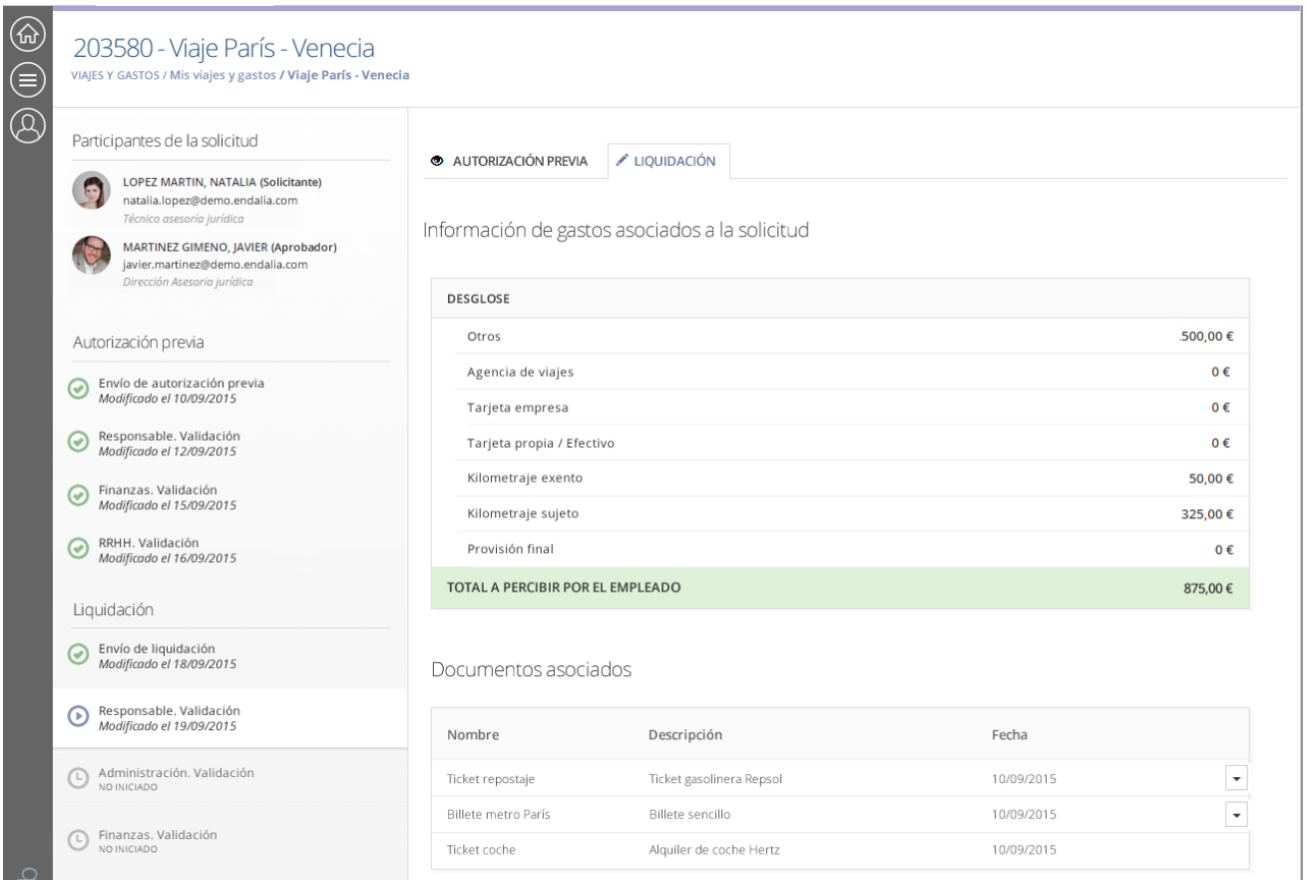


Figura 15. Prototipo de la página de liquidación de una solicitud.

7.8 Interfaz del asistente de creación de un nuevo gasto para la liquidación de una solicitud

A continuación se muestra el prototipo del asistente de creación de un nuevo gasto para la liquidación de una solicitud, que contiene un formulario con la información básica de dicho gasto (Figura 16):

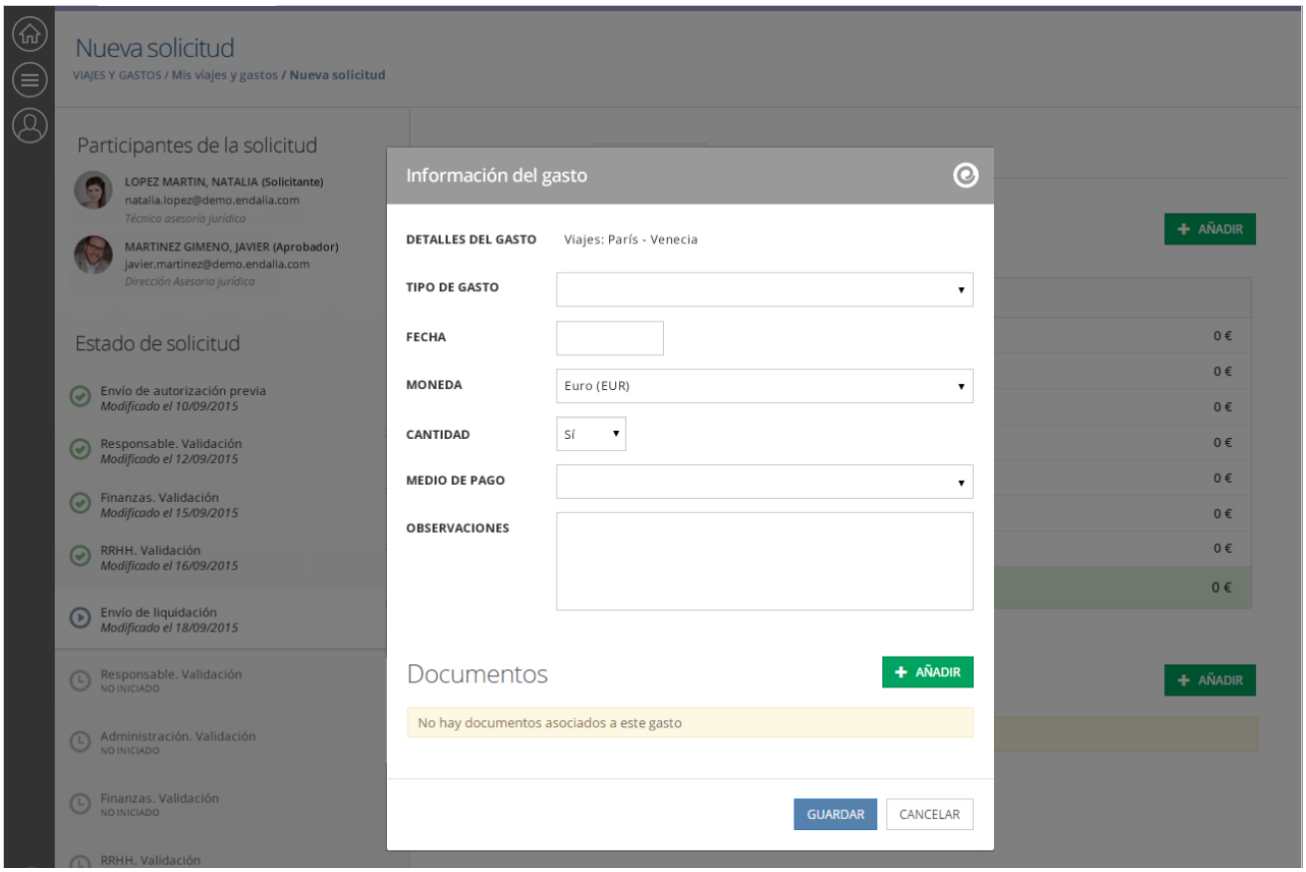


Figura 16. Prototipo del asistente de creación de un nuevo gasto para la liquidación de una solicitud.

8. BIBLIOGRAFÍA

8.1 Referencias

[R1] I. Jacobson, G. Booch, J. Rumbaugh. 2000. “El Proceso Unificado de Desarrollo de Software”. Pearson Education.

[R2] I. Jacobson, G. Booch, J. Rumbaugh. 1999. “El lenguaje unificado de modelado. Manual de referencia”. Ed. Addison Wesley.

[R3] J. Rumbaugh 1991. “Modelado y Diseño Orientado a Objetos”. Ed. Prentice Hall, 1991.

[R4] Hoang Lam, Thuan L. Thai. “.NET Framework Essentials, 3rd Edition”. O’Reilly 2003.

[R5] Art Gittleman. “Computing With C# and the .Net Framework”. Jones and Bartlett Publisheres 2003.

8.2 Referencias web

[W1] http://www.microsoft.com/net

[W2] http://www.wikipedia.org

[W3] http://www.uml.org

[W4] http://webdocs.cs.ualberta.ca/~pfiguero/soo/uml

[W5] http://www.rational.com

[W6] http://www.webstyleguide.com/index.html

[W7] http://www.endalia.com

endalia®
HR Consulting & Software

IMPLEMENTACIÓN

DESARROLLO DE UN MÓDULO DE GESTIÓN DE VIAJES Y GASTOS

VERSIÓN 1.0
PUBLICADO EL 29/08/2016

HISTÓRICO DE REVISIONES

Fecha	Versión	Descripción	Autor
23/08/2016	1.0	Redacción inicial del documento	Nicolás Bailo Ortiz

ÍNDICE

1. INTRODUCCIÓN

1.1 Propósito del documento

Este documento presenta la fase de implementación del trabajo de desarrollo del sistema de gestión de viajes y gastos. Partiendo del trabajo realizado en las fases de análisis y diseño se desarrolla una versión funcional del módulo de Gestión de Viajes y Gastos integrada en el sistema de Gestión y Organización de Recursos Humanos de la organización Endalia.

1.2 Alcance del documento

Este documento describe la fase de implementación del módulo de gestión de viajes y gastos, que corresponde a una de las últimas fases del desarrollo del trabajo.

1.3 Acrónimos

- ASP: Active Server Pages.
- DDL: Data Definition Language.
- DML: Data Manipulation Language.
- FTP: File Transfer Protocol.
- GUI: Graphical User Interface.
- HQL: Hibernate Query Language.
- HTML: HyperText Markup Language.
- HTTP: HyperText Transfer Protocol.
- HTTPS: HyperText Transfer Protocol Secure.
- IIS: Internet Information Services.
- SDK: Software Development Kit.
- SGBD: Sistema Gestor de Base de Datos.
- SMTP: Simple Mail Transfer Protocol.
- SSRS: SQL Server Reporting Services.
- SQL: Structured Query Language.
- T-SQL: Transact-Structured Query Language.
- XML: eXtensible Markup Language.

1.4 Definiciones

- Archivo de recursos: archivo en el que se almacenan datos que se corresponden con cierta información que maneja el sistema, pero que no dependen específicamente de las clases que contienen lógica.
- Control de versiones: gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo.
- Hoja de estilo: lenguaje formal usado para definir la presentación de un documento descrito en HTML o XML.
- Transacción: una transacción en un Sistema de Gestión de Bases de Datos es un conjunto de órdenes que se ejecutan formando una unidad de trabajo, es decir, en forma indivisible o atómica.
- SGBD transaccional: SGBD capaz de mantener la integridad de los datos, haciendo que las transacciones no puedan finalizar en un estado intermedio. Cuando por alguna causa el sistema debe cancelar la transacción, deshace las órdenes ejecutadas hasta dejar la base de datos en su estado inicial (llamado punto de integridad), como si la orden de la transacción nunca se hubiese realizado.

1.5 Referencias

Este documento contiene referencias a los siguientes documentos del trabajo, contenidos en la sección “Anexos”:

- Estándar de codificación: documento que establece las normas que deben seguirse en un desarrollo en cuanto a la codificación del mismo.
- Especificación de requisitos: documento que describe la especificación de requisitos realizada a partir de las necesidades del cliente.
- Análisis del sistema: documento que describe la arquitectura del sistema desarrollado, junto con un listado de casos de uso y su estructura de paquetes.
- Diseño del sistema: documento que describe el modelo por el cual se va a realizar la implementación del sistema.

1.6 Resumen

Este documento describe el proceso de implementación del módulo de gestión de viajes y gastos. Se compone de ocho apartados:

1. Introducción del documento, definición del propósito y alcance del mismo.
2. Se describe el proceso de implementación seguido.
3. Se describen las tecnologías, herramientas y lenguajes empleados durante la implementación del sistema.
4. Se describe el proceso seguido para la implementación de la internacionalización de la aplicación.
5. Se describe el proceso seguido para la gestión de incidencias ocurridas en el sistema.
6. Detalle de las consideraciones necesarias para la implementación del acceso a datos.
7. Se describe el proceso seguido para la implementación del interfaz de usuario.
8. Se describe el proceso seguido para la implementación de las notificaciones por correo electrónico.
9. Se describe el proceso seguido para la implementación de la generación de informes.
10. Bibliografía y referencias web utilizadas para la realización de este documento.

2. DESCRIPCIÓN DEL PROCESO

A partir del trabajo realizado en la fase de diseño, se procede a realizar el proceso de implementación del sistema. Dicho proceso permitirá obtener un módulo totalmente funcional y perfectamente integrado en el sistema de Endalia, que resuelva las necesidades del cliente establecidas mediante los requisitos funcionales del sistema así como siguiendo las pautas establecidas en la fase de análisis del trabajo. Durante la realización de esta fase se realizarán pequeñas pruebas para comprobar el correcto avance del desarrollo, acompañadas de un conjunto de pruebas finales realizadas cuando la implementación se complete. Estas últimas pruebas son las documentadas en el anexo correspondiente.

Dentro del proceso de implementación propiamente dicho podemos identificar dos fases:

- Implementación de la base de datos: creación de las tablas y las relaciones necesarias para el nuevo sistema en la base de datos, el conjunto de clases de acceso a datos correspondientes y sus respectivos mapeos a objetos.
- Implementación de los subsistemas identificados en el diseño, adecuándolos a la interfaz definida previamente.

En este trabajo se ha realizado únicamente la segunda de estas fases, ya que el modelo relacional de tablas ya había sido creado previamente por la entidad para un proyecto similar. El trabajo realizado en la primera de las fases ha consistido únicamente en la validación de dicho modelo relacional para asegurar su validez en el módulo de gestión de viajes y gastos.

3. TECNOLOGÍAS, HERRAMIENTAS Y LENGUAJES

Para el desarrollo del presente proyecto se han utilizado las siguientes tecnologías, lenguajes y herramientas:

- Microsoft .NET Framework. Plataforma de desarrollo descrito en el apartado 3.4 del Documento de Diseño.
- Microsoft SQL Server 2012. SGBD utilizado para la gestión de la base de datos del sistema y descrito en el apartado 3.5 del Documento de Diseño.
- C#. Lenguaje de programación de propósito general nativo de la plataforma .NET.
- ASP .NET. Plataforma de desarrollo de aplicaciones web, que se basa en ASP para la capa de presentación.
- JavaScript. Lenguaje interpretado normalmente embebido en páginas HTML y que puede ser entendido y ejecutado por la práctica totalidad de los navegadores Web modernos.
- Microsoft Visual Studio. Entorno de programación y depuración de código de la plataforma .NET.
- Microsoft SQL Management Studio 2012. Herramienta gráfica que permite realizar tareas de mantenimiento de la base de datos sobre SQL Server 2012.
- Microsoft Team Foundation Serve. Herramienta que permite, entre otras funcionalidades, la gestión del control de versiones de un proyecto sobre Visual Studio.
- IIS. Servidor de aplicaciones de Microsoft que permite gestionar servidores HTTP, HTTPS, FTP o SMTP, entre otros. Se describe en el Documento de Diseño, en el apartado 3.7.
- T-SQL. Lenguaje de acceso a datos basado en SQL. Está descrito en el apartado 6.2.1 del presente documento.
- NHibernate. Framework de mapeo objeto-relacional que facilita la representación de entidades relacionales de base de datos en objetos accesibles desde la aplicación desarrollada.
- Infragistics NetAdvantage 8. Librería para .NET que contiene controles para diversos entornos de desarrollo dentro de dicha plataforma, como ASP.NET o Winforms.
- Log4Net. Herramienta para ayudar en la generación de ficheros de registro.
- Navegadores: Microsoft Internet Explorer (versiones superiores a la 8.0), Mozilla Firefox y Google Chrome.

4. IMPLEMENTACIÓN DE LA INTERNACIONALIZACIÓN

4.1 Definición

Uno de los requisitos iniciales planeados para el módulo de gestión de viajes y gastos es la internacionalización del sistema. La internacionalización se define como el "proceso de diseñar una aplicación que pueda ser adaptada a distintos idiomas y culturas sin necesidad de efectuar cambios estructurales en la misma". En este apartado se describe la solución adoptada en el sistema para realizar la internacionalización del mismo.

4.2 Objetivos de la internacionalización

Un sistema internacionalizado persigue las siguientes características:

- El sistema debe poder ser ejecutado en cualquier lugar del mundo.
- El texto mostrado por el sistema debe estar en el idioma del usuario final.
- El texto mostrado por el sistema no debe estar codificado dentro de la aplicación, sino que debe ser almacenado de forma externa y ser recuperable de forma dinámica en tiempo de ejecución.
- Otros aspectos culturales como números, fechas u horas deben aparecer en el formato e idioma del usuario.

4.3 Elementos a internacionalizar

Los elementos susceptibles de ser internacionalizados en el módulo de gestión de viajes y gastos son:

- Textos.
- Números: pueden variar por el carácter delimitador de decimales, o el separador de miles, por ejemplo.
- Fechas y horas: pueden variar de muchas formas, por ejemplo, el orden en que se indican los días y los meses de una fecha.
- Monedas: pueden varias los símbolos monetarios que son utilizados (€, $....).

4.4 Reglas clave de desarrollo para la internacionalización

A continuación se presentan un conjunto de reglas que es necesario cumplir para desarrollar de manera óptima el proceso de internacionalización de una aplicación .NET.

- Identificar los elementos dependientes de la cultura. Los mensajes de texto son los componentes que de manera más obvia varían con la cultura, ya que deben ser traducidos al idioma del usuario final. Sin embargo, hay otros tipos de datos que podrían variar con la región o el idioma, como ya se ha comentado en el apartado anterior.
- Aislar el texto traducible. El texto traducido debe agruparse y aislarse en archivos de recursos. El texto traducido incluye mensajes de estado, mensajes de error, etiquetas de componentes GUI, etc. La codificación siguiente, en la que los mensajes se asignan de manera implícita en el código fuente, debe evitarse totalmente:

```
lblName.Text = "Nombre";
```

- Formatear números y monedas. Si la aplicación muestra números y monedas, será necesario darles el formato adecuado. Evitar:

```
Double amount = 25.0;

TextboxAmount.Text = amount.ToString + "€";
```

- Formatear fechas y horas. El formato de las fechas y horas varía según la cultura. Evitar:

```
DateTime date = "10/05/2005";

TextboxData = date.ToString();
```

- Reservar espacio suficiente en el GUI. El texto de las etiquetas varía de longitud según el idioma en que esté codificado, siendo por ejemplo los mensajes en inglés generalmente más cortos que en otros idiomas.

4.5 Proceso de internacionalización

El proceso de internacionalización de aplicaciones .NET es el siguiente:

- Almacenar las etiquetas en un archivo de texto que llamaremos archivo de origen de datos o de recursos de cadena (este punto puede variar, pudiéndose guardar también en otros formatos como archivos .resx). El formato de estos archivos en el siguiente:
    - Una declaración de cadena por línea en la que se declara una etiqueta de internacionalización y su valor correspondiente dentro de la referencia cultural que define al archivo separados por un signo '='. Por ejemplo:

      ```
      Index_lbl_lblUserName=Usuario
      ```

    - Comentarios precedidos por un punto y coma ';' al principio de la línea

      ```
      ;esto es un comentario dentro del archive de datos
      ```

- Convertir el archivo de origen de datos en un formato compatible con los ensamblados satélites (.resources). Para realizar este paso se utiliza la herramienta *Resgen*. Esta herramienta convierte archivos .txt y archivos .resx (formato de recursos basado en XML) en archivos .resources binarios de *Common Language Runtime*, que se pueden incrustar en un archivo ejecutable binario de motor de tiempo de ejecución o compilar en ensamblados satélite.
- Compilar el ensamblado satélite con los archivos de recursos. A continuación, para generar el ensamblado satélite es necesario utilizar otra herramienta del SDK denominada al.exe (*Assembly Linker*) la cual crea dicho ensamblado y a su vez embebe el recurso correspondiente.
- Declarar en el código fuente las cadenas de internacionalización. En el código fuente de la aplicación en que vayamos a utilizar los recursos hay que crear un campo de tipo cadena de texto con un modificador de acceso privado para cada uno de los datos que serán cargados desde el ensamblado satélite. La codificación de estas variables se describe en el documento 'Estándar de codificación'.

```
private string Index_lbl_lblUserName
```

- Cargar en el código fuente los valores correspondientes del fichero de recursos compilado:

```
Assembly a = Assembly.Load("labels");

rm = new ResourceManager("labels", a);
```

```
if (rm != null)
{
        Index_lbl_lblUserName = rm.GetString("Index_lbl_lblUserName");
}
```

Para poder trabajar con la clase *ResourceManager* es necesario declarar el espacio de nombres *System.Resources*.

Una vez inicializado el objeto 'rm', se cargan los valores del archivo de recursos a los campos. Para ello se utiliza el método *GetString* del objeto 'rm', el cual recibe como parámetros el nombre del campo que queremos invocar del archivo de recursos que nos provee el ensamblado satélite.

El método LoadI18N carga en los campos de texto del formulario las variables de cadenas obtenidas de la siguiente forma:

```
btnSave.Text = Index_btn_btnSave;
```

5. IMPLEMENTACIÓN DE LA GESTIÓN DE INCIDENCIAS

El sistema de gestión de incidencias, o Log de la aplicación, se encarga de registrar secuencialmente los eventos del sistema en un fichero, guardando información acerca del tipo de evento, de cuándo ocurrió y qué lo causó. Dicho fichero puede posteriormente ser consultado, auditado y analizado para conocer el uso, funcionamiento y posibles problemas del sistema.

La siguiente información es registrada siempre, independientemente del resto del contenido del registro de log:

- Fecha y hora de registro del evento.
- Sistema gestor del log (clase que provoca la llamada).
- Etiqueta o tag con el tipo de mensaje según la siguiente clasificación:
    - DEBUG: mensaje informativo para depuración de la ejecución del sistema.
    - INFO: mensaje informativo sobre algún suceso acaecido en el sistema.
    - WARM: mensaje de aviso sobre intento de alguna ejecución peligrosa para el sistema.
    - ERROR: mensaje acerca de una ejecución fallida pero controlada del sistema.
    - FATAL: mensaje de interrupción total de la ejecución del sistema.

La siguiente información variará dependiendo del tipo de registro:

- Mensaje de error generado en caso de error de ejecución (por ejemplo en caso de fallo de ejecución de una instrucción SQL escribiría el mensaje devuelto por SQL Server).
- Cualquier mensaje informativo que necesite ser registrado por el sistema en un momento dado.

Para registrar en el fichero de Log las interacciones del usuario con la aplicación, se emplea la librería *log4net*, pero es necesario que en cada evento de la aplicación se escriba explícitamente en él, por ejemplo:

```
Log.Write("DEBUG: Index.GridRequestList_RowCommand command=" + e.CommandName);

Log.Write("DEBUG: Employee.Remove EmployeeID=" + EmpID.ToString());

Log.Write("ERROR: Employee.Remove EmployeeID=" + EmpID.ToString() + " " + e.ToString());
```

6. IMPLEMENTACIÓN DEL ACCESO A BASE DE DATOS

6.1 Introducción

Como se ha especificado en el documento de diseño, el Sistema Gestor de Base de Datos utilizado ha sido Microsoft SQL Server 2012, al que se accede a través del *Framework* NHibernate.

En este apartado se van a describir las principales características e implicaciones del uso conjunto de ambos sistemas en la implementación propiamente dicha del módulo de gestión de viajes y gastos.

6.2 SQL Server 2012

Es un sistema de gestión de bases de datos relacionales (SGBD) basado en lenguaje SQL, con las características principales:

- Soporte de transacciones.
- Gran estabilidad.
- Seguridad.
- Escalabilidad.
- Soporta procedimientos almacenados.
- Potente entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.
- Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y las terminales o clientes de la red sólo acceden a la información.
- Permite administrar información de otros servidores de datos.
- No es multiplataforma, ya que sólo está disponible en sistemas operativos de Microsoft.

6.2.1 T-SQL

Transact-SQL o T-SQL es una extensión a SQL con las siguientes características:

- Lenguaje de control de flujo. Instrucciones para permitir la ejecución de bloques condicionados e iterativos-

  ```
  IF-THEN-ELSE, CASE-WHEN, WHILE, ...
  ```

- Variables locales.

  ```
  DECLARE @EmployeeID INT
  ```

- Funciones nativas para procesado de cadenas, gestión de fechas, funciones matemáticas, etcétera.

  ```
  REPLACE(@foo, '-', '')
  ```

- Mejoras en sentencias *DELETE* y *UPDATE*, para permitir uniones en la cláusula *FROM*.

6.2.2 Transacciones

Una transacción es un conjunto de operaciones que debe ejecutarse como una sola unidad, según el principio ACID expuesto en el documento de Diseño.

En SQL Server, las sentencias SQL son tratadas como transacciones, y si ocurre cualquier tipo de incidencia durante su ejecución, el sistema es capaz de volver al estado anterior a su ejecución.

En ocasiones se requiere ejecutar un conjunto de sentencias SQL de forma transaccional, para lo que el lenguaje proporciona las sentencias de gestión de transacciones:

- BEGIN TRAN. Indica el comienzo de una transacción.
- ROLLBACK TRAN. Deshace todos los cambios realizados por la transacción activa.
- COMMIT TRAN. Una vez que las operaciones de la transacción se han completado con éxito, se indica a la base de datos que vuelva a un estado consistente.

```
DECLARE @Error INT
BEGIN TRANSACTION
INSERT INTO
NombreTabla
(
        Campo1,
        Campo2
)
VALUES
(
        @Valor1,
        @Valor2
)
SET @Error = @@ERROR
IF @Error != 0 GOTO ERROR_HANDLER

SELECT SCOPE_IDENTITY() AS 'Identity'
SET @Error = @@ERROR
IF @Error != 0 GOTO ERROR_HANDLER

COMMIT TRANSACTION
RETURN SCOPE_IDENTITY()
ERROR_HANDLER:
IF @@TRANCOUNT != 0 ROLLBACK TRANSACTION
RETURN @Error
```

6.2.3 Procedimientos almacenados

Un procedimiento almacenado (*Stored Procedure*) de Microsoft SQL Server es un conjunto de instrucciones T-SQL que residen físicamente en la base de datos. Las principales características de los procedimientos almacenados son las siguientes:

- Ejecución directamente en el motor de base de datos que suele estar en un servidor separado, evitando tráfico entre el SGBD y la aplicación cliente.
- Posibilidad de almacenamiento del plan de ejecución por parte del SGBD, lo que repercute positivamente en la eficiencia de la consulta.
- Encapsula la lógica de negocio, de tal modo que permiten que varios programas cliente puedan usarla, reduciendo el coste de mantenimiento de los mismos.
- Versatilidad. Al estar desarrollados en T-SQL, permiten realizar funciones del mismo nivel de complejidad que las que se pueden realizar en el código de la aplicación.

```
CREATE PROCEDURE (NombreProcedimiento)
(
	@parametro1 INT,
	@parametro2 INT
)
AS SELECT
	SUM(EnvPollLevelValue) AS IndicatorSUM
FROM
	NombreTabla1
WHERE
	<condición booleana>
```

6.2.4 Desencadenadores

Los desencadenadores (o *triggers*) son, al igual que los procedimientos almacenados, un conjunto de sentencias T-SQL almacenadas en base de datos, pero con la particularidad de ser ejecutadas automáticamente cuando un usuario realiza una acción en la tabla de la base de datos que lleva asociado el desencadenador.

Su principal utilidad es el control de actualizaciones y borrados en cascada, que no pueden ser controlados por el SGBD debido a ciclos múltiples.

Se pueden crear desencadenadores para los siguientes tipos de instrucciones:

- *INSERT*: Inserción de información en la base de datos.
- *UPDATE*: Actualización de información de la base de datos.
- *DELETE*: Eliminación de información de la base de datos.

A continuación se muestra un ejemplo de un *trigger* que controla que al eliminar una entrada de la tabla 1 se elimine una entrada asociada a esa tabla, en la tabla 2:

```
CREATE TRIGGER [NombreTrigger] ON [NombreTabla1] FOR DELETE AS DELETE FROM NombreTabla2
WHERE <condición booleana>
```

6.3 Acceso a datos

Las clases de acceso a datos están asociadas a cada una de las tablas de la base de datos, y contienen diversos métodos que pueden ser de actualización, consulta, eliminación o inserción de datos.

En este trabajo se han reutilizado algunas clases de acceso a datos ya implementadas anteriormente en el software de Endalia, pero realizando modificaciones en los métodos utilizados para el módulo de gestión de viajes y gastos para adaptarlos a las páginas implementadas.

6.3.1 Mapeo objeto-relacional de las entidades en base de datos

Gracias al uso del *Framework* NHibernate, se ha realizado un mapeo de las entidades presentes en la base de datos en forma de objetos accesibles desde el código de la aplicación. Así, cada entidad tiene su fichero de configuración (en formato XML) en el que se detallan sus diferentes atributos y relaciones, así como un fichero (en formato .cs) que constituye la representación del objeto en código, y que permite al desarrollador acceder a la entidad desde cualquier parte del proyecto.

6.3.2 Clases de acceso a datos

Además de las clases de representación de objetos, se han creado unas clases de acceso a datos con métodos comunes que permiten traer una gran cantidad de información de la base de datos. Los métodos incluidos en estas clases permiten obtener datos a medida, según las necesidades de las páginas desarrolladas.

Por otro lado, estas clases ofrecen una capa extra en la arquitectura software de la aplicación, permitiendo separar la aplicación de la propia gestión de la información en la base de datos. Esto se consigue en parte con el lenguaje de consultas de NHibernate, el *Hibernate Query Language* (HQL).

7. IMPLEMENTACIÓN DE LA INTERFAZ DE USUARIO

7.1 Introducción

El diseño definitivo de la interfaz viene definido por los esquemas de diseño de pantalla especificados en el documento de diseño.

La interfaz de usuario tendrá en cuenta el diseño ya existente en la web corporativa de la organización, que por motivos de confidencialidad no se especificará.

A continuación, se especifica la manera en que se han utilizado los diferentes elementos que componen la interfaz finalmente diseñada, y se muestran capturas de pantalla del mismo.

7.2 Elementos de interfaz

7.2.1 Iconos e imágenes

La utilización de imágenes e iconos ha sido destinada siempre a facilitar al usuario la utilidad y funcionamiento de los diferentes elementos y partes del sistema, acompañándolos siempre preferiblemente de una etiqueta textual que especifica la funcionalidad o el destino del elemento representado por la imagen, llamada *tooltip*.

Los iconos de acceso adecuadamente escogidos facilitan a los usuarios encontrar las diferentes secciones del sistema con mayor rapidez.

7.2.2 Tablas

Se han utilizado tablas siempre que ha sido necesario para mostrar listados de datos. El texto de la cabecera destaca sobre el resto haciéndolo en negrita, y las filas alternas deben tener colores de fondo alternos para facilitar la legibilidad de las tablas.

7.2.3 Editores de fechas

Se han utilizado, siempre que ha sido posible, editores de fechas que permiten una navegación gráfica a través de diferentes fechas mediante la visualización de un calendario.

7.2.4 Colores y fuentes

Los colores a utilizar vienen determinados por los colores corporativos de la organización a la que va destinado el proyecto del módulo de gestión de viajes y gastos; del mismo modo que ocurre con las fuentes y estilos.

7.2.5 Estilos

Todos los estilos CSS del sistema deben estar registrados en archivos CSS u hojas de estilo, facilitando al máximo las futuras modificaciones de estilo y permitiendo su adaptación a otras organizaciones. En esta hoja de estilos también deben estar incluidos los colores y fuentes comentados en el punto anterior.

7.3 Pantallas del sistema

A continuación se presenta el diseño definitivo de las principales interfaces del sistema; ya que por el tamaño del mismo, entrar en detalle en cada una de las pantallas existentes sería excesivo para este documento.

7.3.1 Página “Mis Viajes y Gastos”

En la figura a continuación (Figura 1) se muestra una captura de pantalla de la sección “Mis Viajes y Gastos”, donde el usuario puede ver un listado con sus solicitudes.

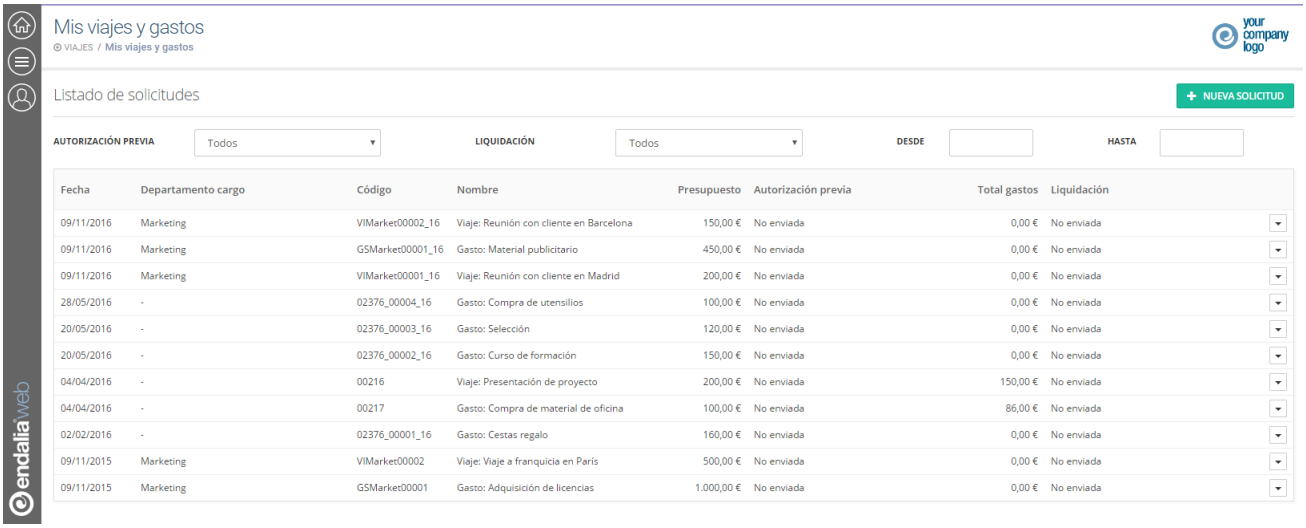


Figura 1. Página de Mis Viajes y Gastos.

La página ofrece un pequeño número de filtros que el usuario podrá utilizar para seleccionar el subconjunto de solicitudes que quiera ver. Estas solicitudes se pueden ver o editar gracias al menú desplegable situado en la parte derecha de cada fila de la tabla.

Además, el botón situado a la derecha del subtítulo “Listado de solicitudes” permite abrir un asistente para la creación de una solicitud, tal y como se muestra en la captura a continuación:

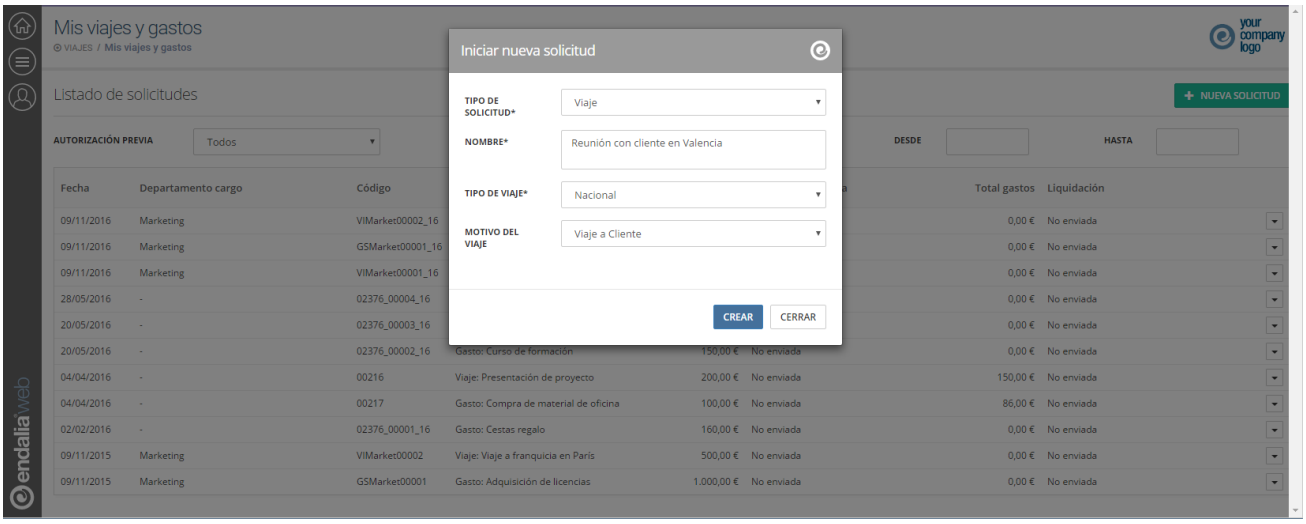


Figura 2. Asistente de creación de una solicitud.

7.3.2 Página “Viajes y Gastos de colaboradores”

En la figura a continuación (Figura 3) se muestra una captura de pantalla de la sección “Viajes y Gastos de colaboradores”. En esta sección, se muestra en la parte izquierda un árbol con la estructura organizativa de la empresa (siempre por debajo del usuario), mientras que en la parte derecha hay un resumen de las solicitudes de los empleados que ocupan esos puestos.

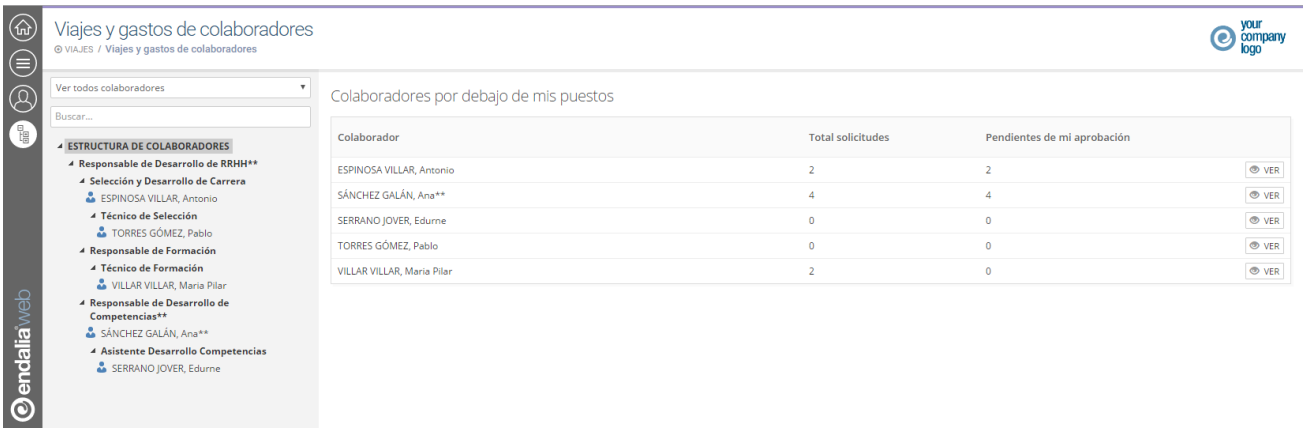


Figura 3. Página de Viajes y Gastos de colaboradores, resumen por empleado.

Al seleccionar un empleado, la aplicación pasa a mostrar el listado de solicitudes de dicho empleado. Con los botones situados a la derecha de cada fila de la tabla se puede tanto acceder a los detalles de la solicitud como aprobarla si se tienen los permisos necesarios, tal y como se muestra en la figura a continuación (Figura 4):

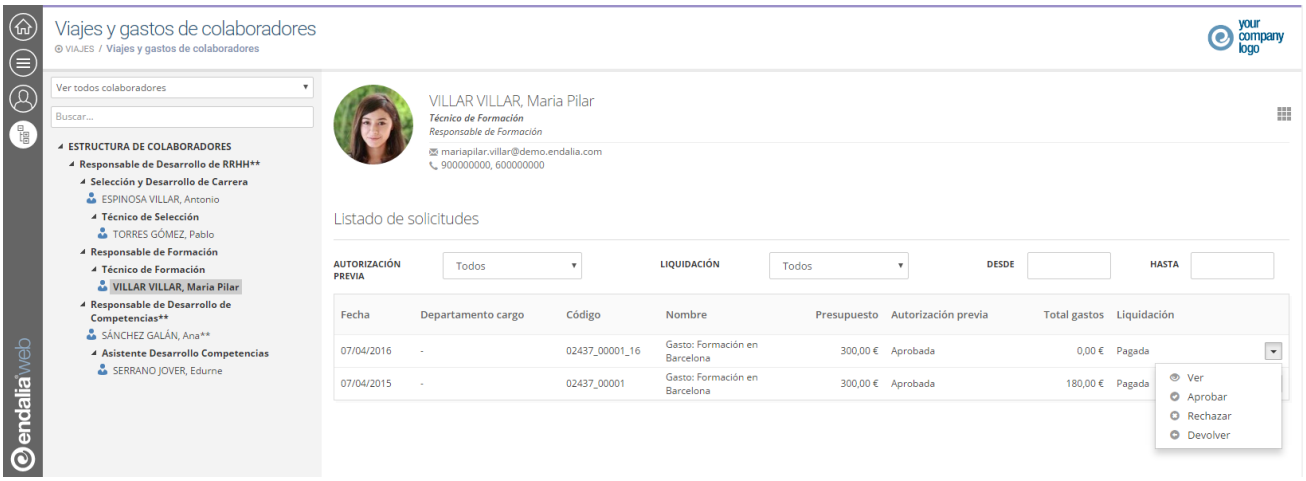


Figura 4. Página de Viajes y Gastos de colaboradores, solicitudes de un empleado.

7.3.3 Página “Aprobación de viajes y gastos”

En la figura a continuación (Figura 5) se muestra una captura de pantalla de la sección “Aprobación de viajes y gastos”, dónde el usuario puede ver las solicitudes que tiene pendientes, acceder a sus detalles y aprobar una o varias a la vez.

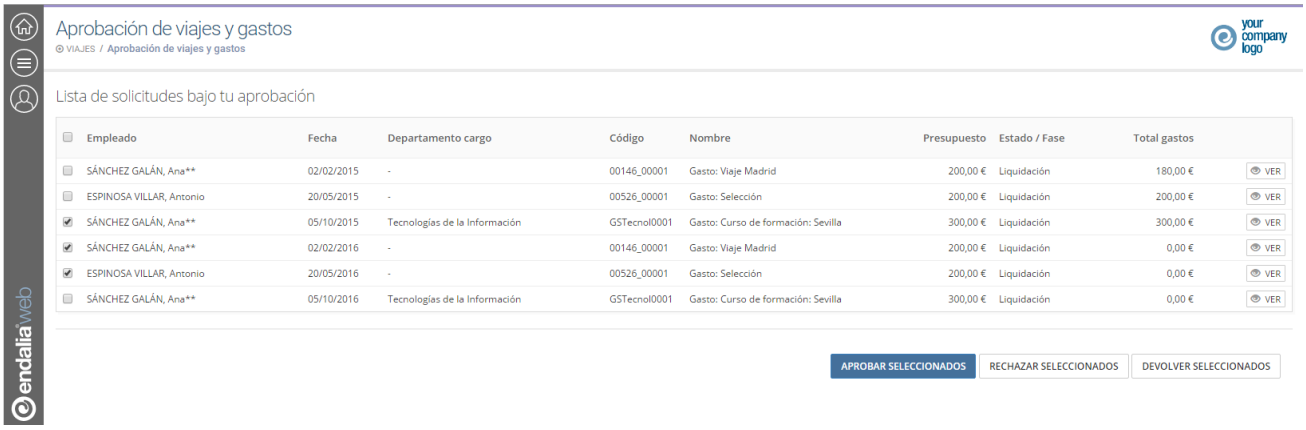


Figura 5. Página de Aprobación de Viajes y Gastos.

Al seleccionar una opción de aprobación (aprobar, rechazar o devolver) se abre un asistente que invita al usuario a escribir observaciones acerca de su elección. Estas observaciones las recibirá el empleado solicitante en su notificación vía email.

7.3.4 Página de detalles de una solicitud

En este apartado se mostrarán las páginas que componen los detalles de una solicitud, tanto en su modo de visualización (donde la solicitud no es editable) como en su modo de edición. Como se ha explicado anteriormente, cada solicitud está dividida en dos fases diferenciadas: Autorización Previa y Liquidación.

7.3.5 En la fase de Autorización Previa

Como se ve en la siguiente figura (Figura 6), en esta primera fase el solicitante deberá rellenar un formulario con la información básica de su solicitud. También tiene la posibilidad de introducir un itinerario de viaje, o de añadir documentos asociados, a través de un asistente similar al visto en el apartado 7.3.1 para crear una solicitud nueva.

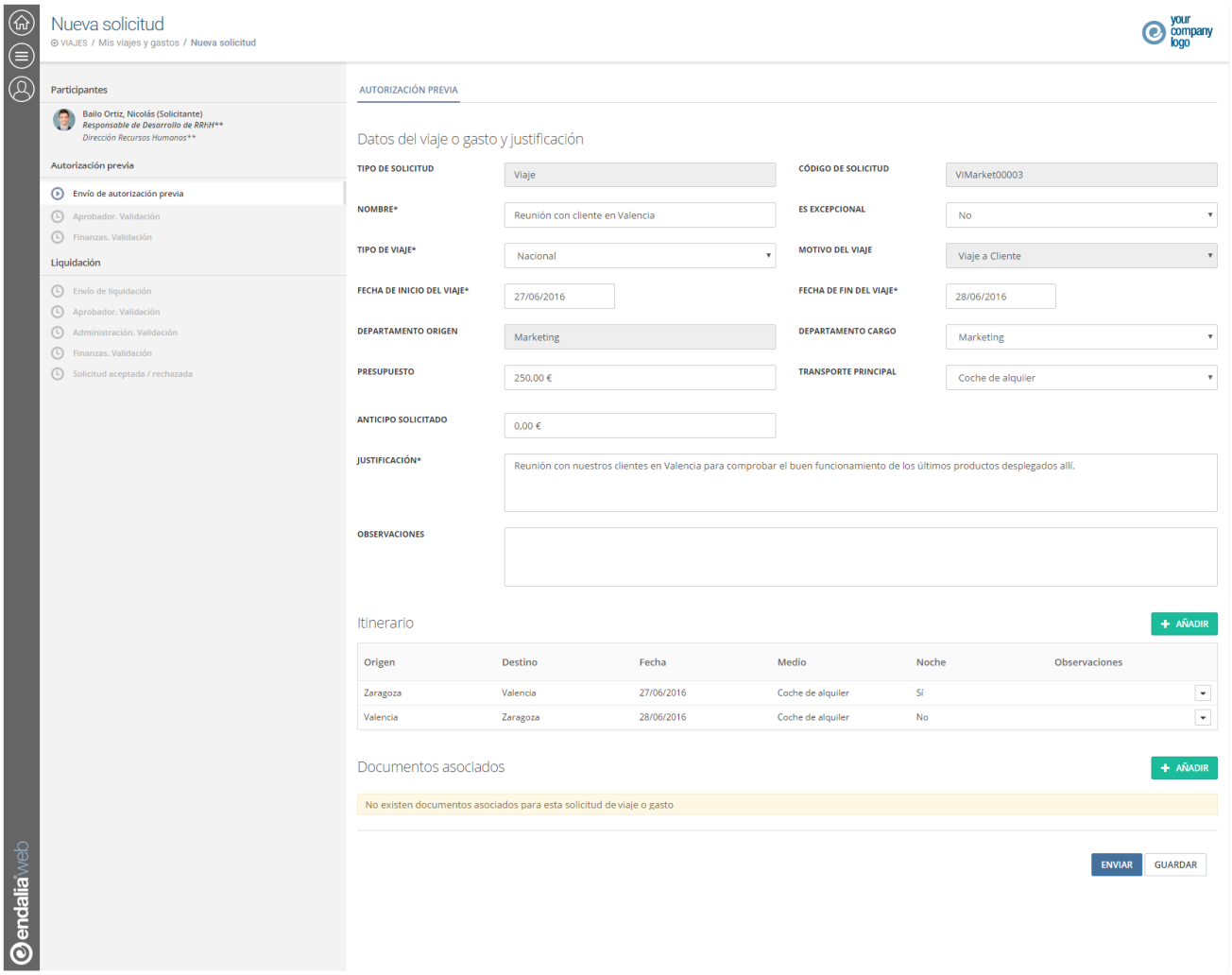


Figura 6. Página de edición de la Autorización Previa de una solicitud.

Por otro lado, cuando se accede a esta pantalla una vez que la solicitud ha sido enviada ésta se muestra en modo de no edición. Tal y como se ve en la siguiente figura (Figura 7), si se accede siendo un aprobador aparecerán también los botones correspondiente a la aprobación, rechazo o devolución de la solicitud.

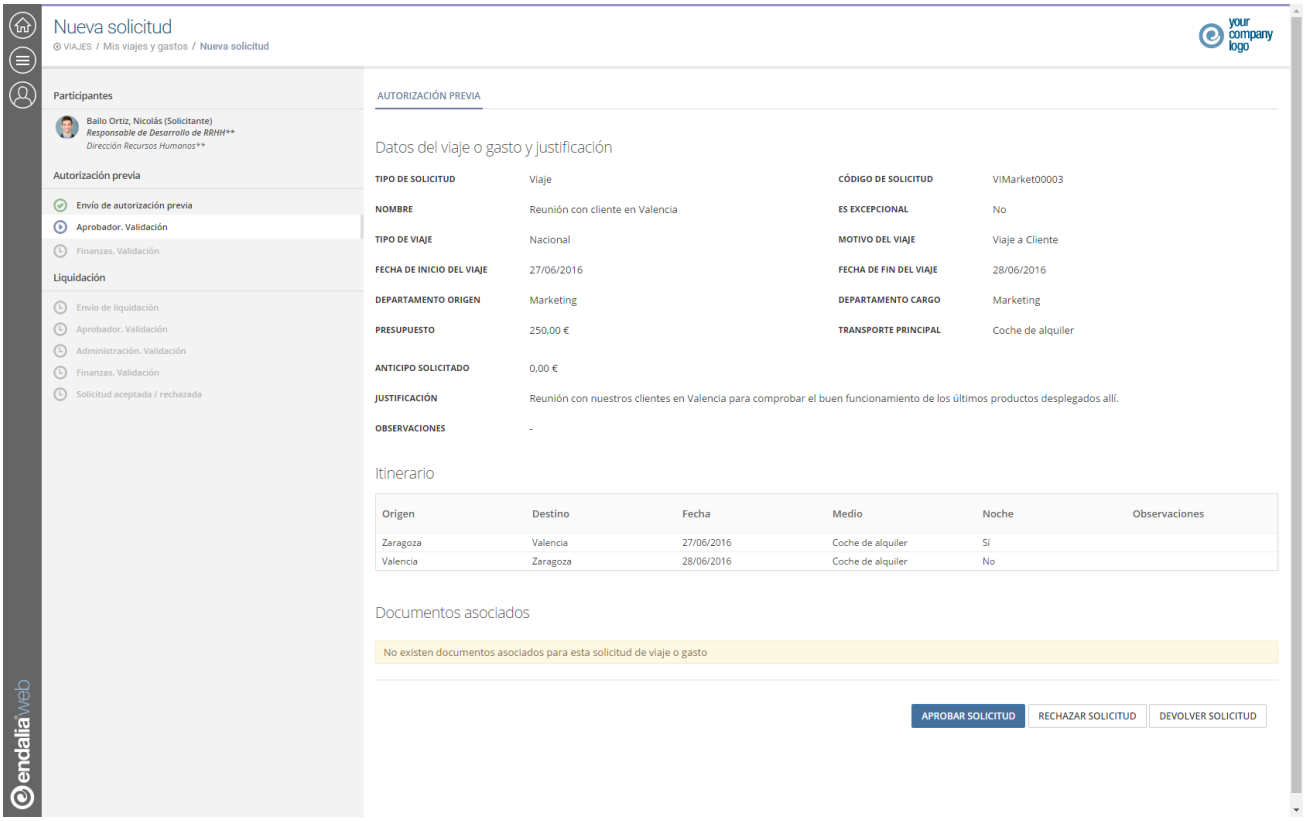


Figura 7. Página de visualización de la Autorización Previa de una solicitud.

7.3.6 En la fase de Liquidación

Como se ve en la siguiente figura (Figura 8), el objetivo de esta fase es introducir una serie de gastos que conformarán el gasto total o el viaje reflejado en la solicitud. La pantalla ofrece un listado de los gastos introducidos así como un desglose por forma de pago con el total a percibir por el empleado. Además, también permite la edición de algunos de los campos de información básica de la solicitud.

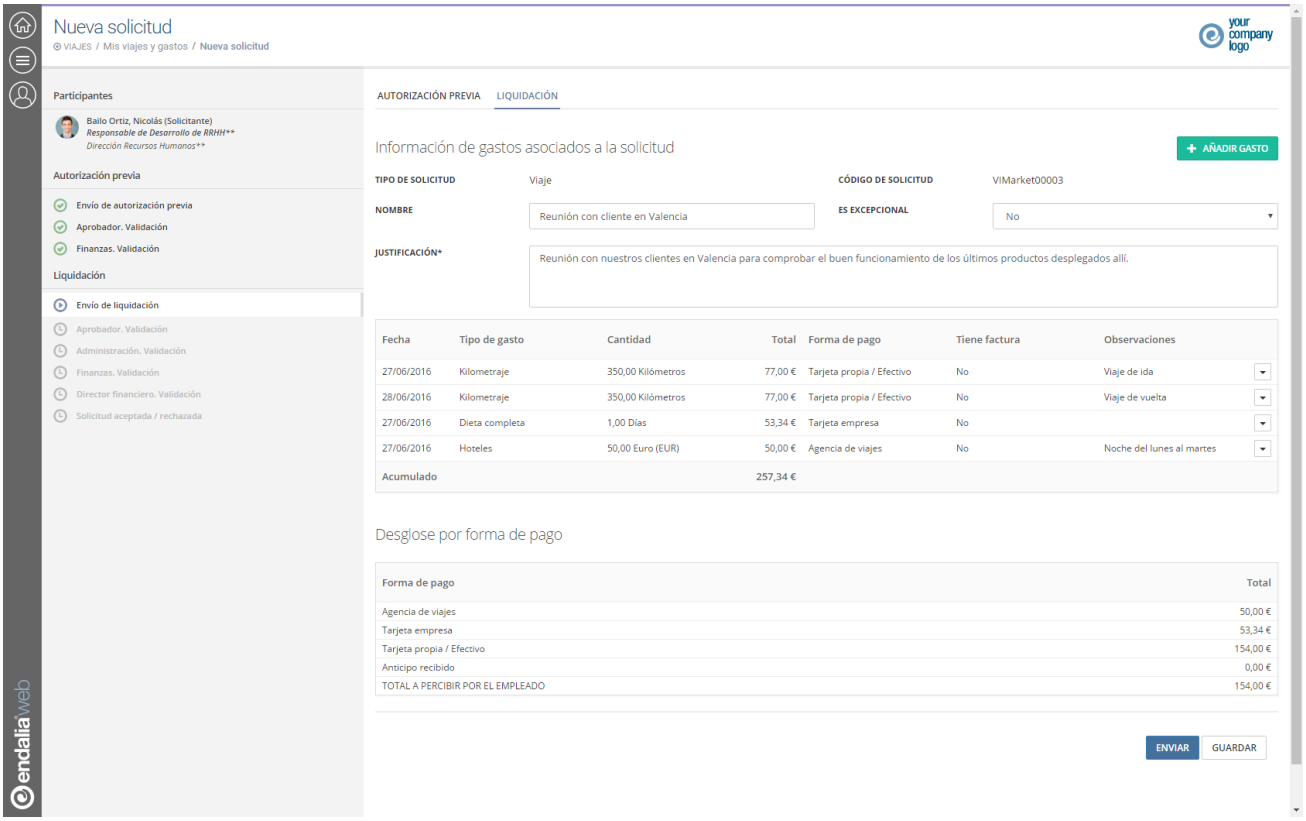


Figura 8. Página de edición de la Liquidación de una solicitud.

Para introducir un nuevo gasto, se abre el asistente que se muestra en la siguiente figura (Figura 9), donde el solicitante deberá introducir el tipo, la fecha, la forma de pago y la cuantía, que depende del tipo de gasto introducido (puede ser kilometraje, un número de días o una cantidad de dinero). El sistema hará la conversión a euros y la mostrará.

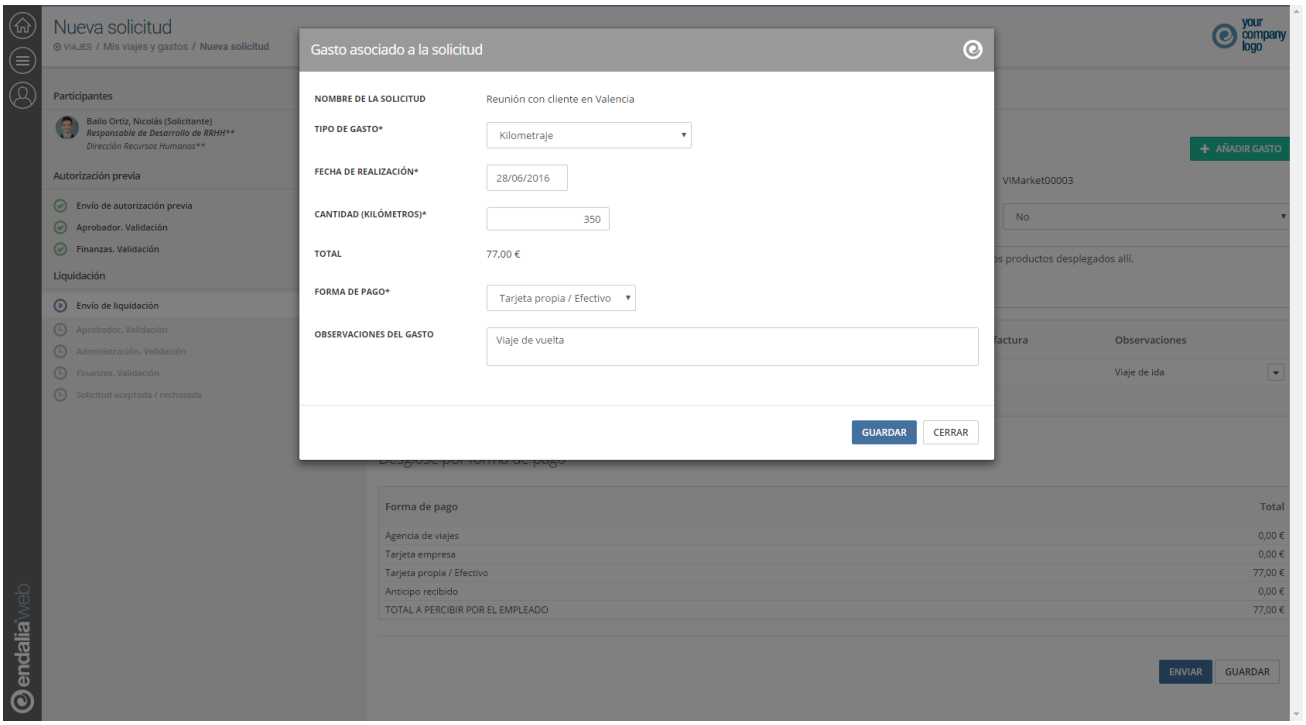


Figura 9. Asistente de creación de un gasto asociado a una solicitud.

Además, el sistema permite seleccionar cualquier gasto introducido y asociarle una o varias facturas, también mediante un asistente, como se puede ver en la siguiente figura (Figura 10).

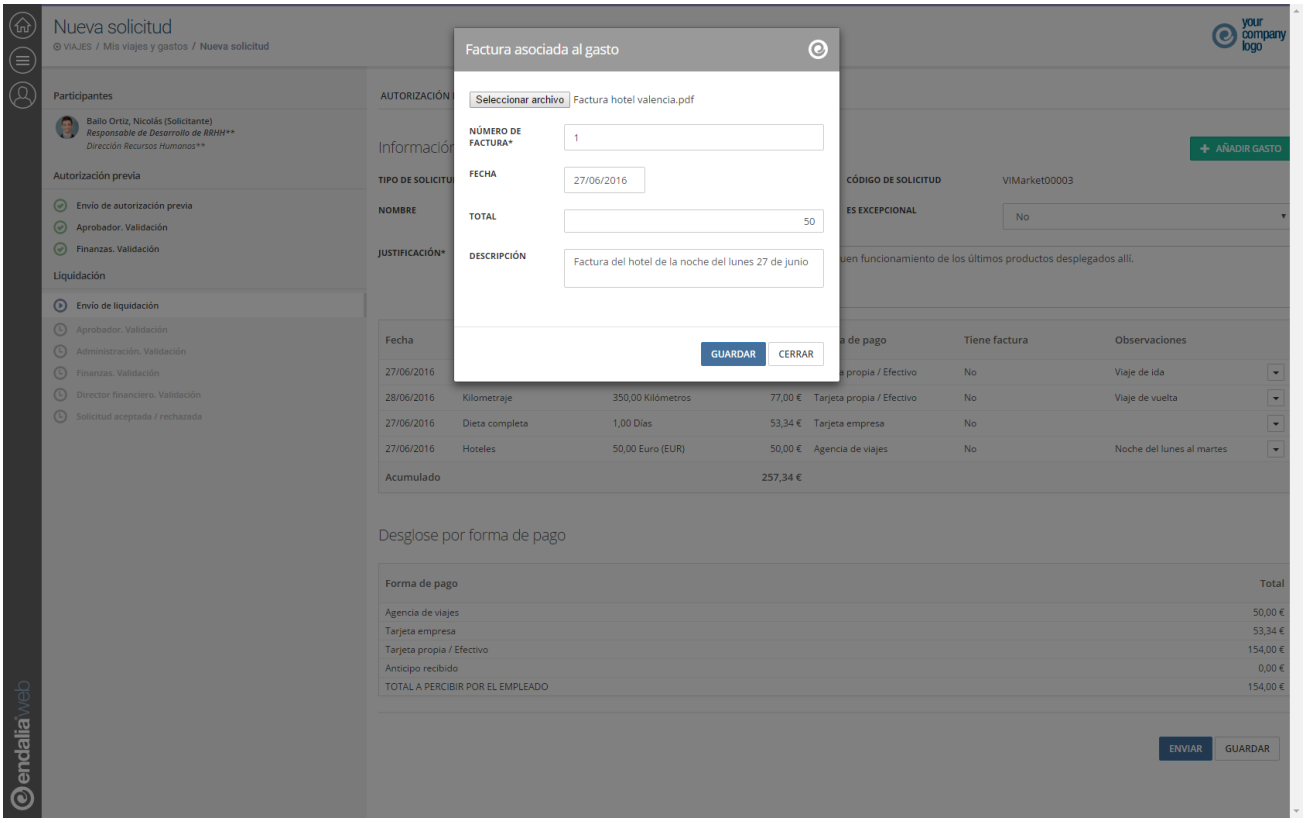


Figura 10. Asistente de inserción de factura.

Por último, esta pantalla también puede ser vista sin permisos de edición, ya sea porque el solicitante ha enviado la solicitud o porque es el responsable el que está visualizando la solicitud para aprobarla. En la siguiente figura (Figura 11) se muestra la pantalla en este último caso.

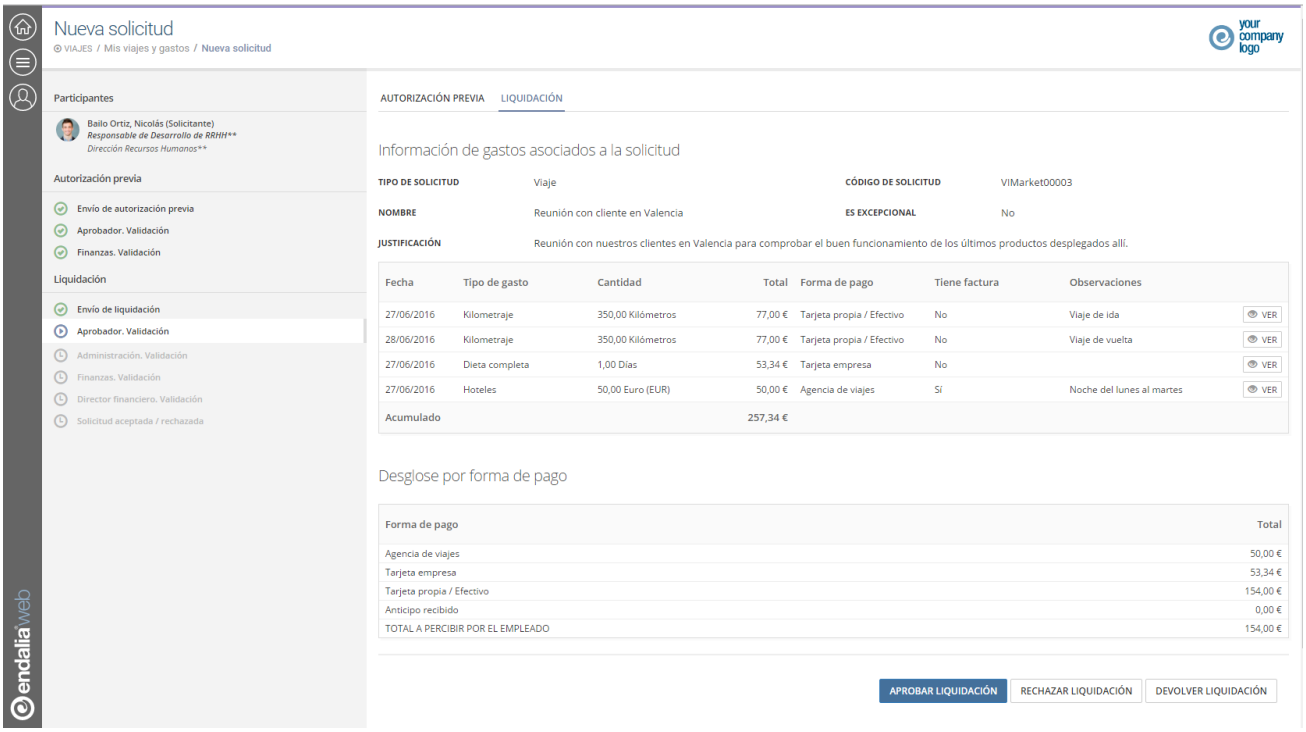


Figura 11. Página de visualización de la Liquidación de una solicitud.

8. IMPLEMENTACIÓN DE LAS NOTIFICACIONES VÍA EMAIL

Se ha desarrollado un sistema de notificaciones vía email cuyo objetivo es el de acelerar el proceso de aprobación de una solicitud. Así, un usuario no deberá entrar en la aplicación para saber si tiene solicitudes pendientes de aprobar, sino que el sistema le notificará la existencia de una solicitud pendiente de aprobar en el momento de su existencia. Estas notificaciones contienen también un enlace, en forma de botón, a los detalles de la solicitud en la aplicación web.

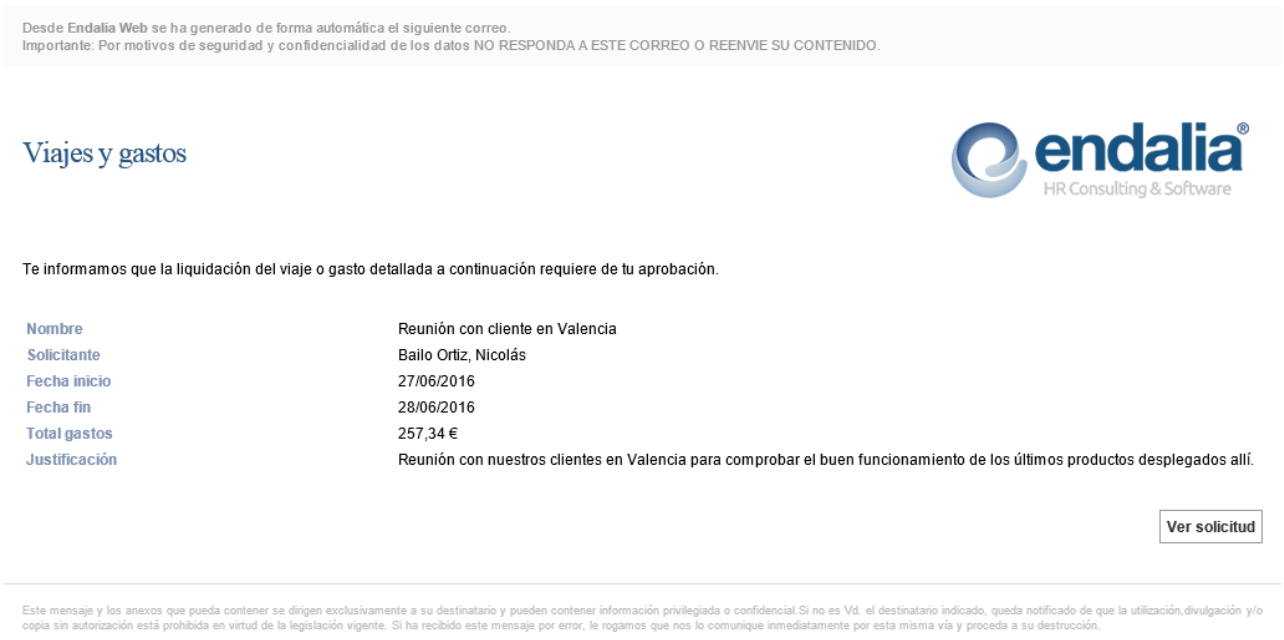
Para realizar el envío de estas notificaciones se ha utilizado la librería *System.Net.Mail* del *Framework* .NET. Esta librería permite enviar mensajes de correo electrónico mediante el protocolo SMTP.

Se han creado un total de ocho notificaciones, cuatro para cada fase de una solicitud:

- Cuando una solicitud requiere de la aprobación de un responsable (es decir, cuando el empleado la envía o cuando un aprobador anterior la aprueba en su fase) se envía una notificación a dicho responsable.
- Cuando una solicitud es totalmente aprobada dentro de su fase, se notifica el empleado solicitante. Si la fase aprobada es la Autorización Previa, se le invita a continuar la solicitud rellenando la Liquidación.
- Cuando una solicitud es rechazada en cualquiera de sus fases, se informa al empleado solicitante de que la solicitud ha sido rechazada por un aprobador, y que no se podrán realizar más cambios sobre la misma.
- Cuando una solicitud es devuelta en cualquiera de sus fases, se informa al empleado

Además, los responsables pueden dejar observaciones referentes a sus aprobaciones, rechazos o devoluciones, que el empleado solicitante podrá ver en el email de notificación.

A continuación se muestra un ejemplo de notificación enviada por el sistema, en este caso un email informando a un responsable de que tiene una solicitud pendiente de aprobar en la fase de Liquidación.



Desde **Endalia Web** se ha generado de forma automática el siguiente correo.
Importante: Por motivos de seguridad y confidencialidad de los datos NO RESPONDA A ESTE CORREO O REENVIE SU CONTENIDO.

Viajes y gastos

endalia HR Consulting & Software

Te informamos que la liquidación del viaje o gasto detallada a continuación requiere de tu aprobación.

Nombre	Reunión con cliente en Valencia
Solicitante	Bailo Ortiz, Nicolás
Fecha inicio	27/06/2016
Fecha fin	28/06/2016
Total gastos	257,34 €
Justificación	Reunión con nuestros clientes en Valencia para comprobar el buen funcionamiento de los últimos productos desplegados allí.

Ver solicitud

Este mensaje y los anexos que pueda contener se dirigen exclusivamente a su destinatario y pueden contener información privilegiada o confidencial.Si no es Vd. el destinatario indicado, queda notificado de que la utilización,divulgación y/o copia sin autorización está prohibida en virtud de la legislación vigente. Si ha recibido este mensaje por error, le rogamos que nos lo comunique inmediatamente por esta misma vía y proceda a su destrucción.

Ilustración 1. Notificación vía email enviada a un aprobador.

9. IMPLEMENTACIÓN DE LA GENERACIÓN DE INFORMES

El sistema ofrece la posibilidad de exportar un informe en formato PDF que contenga información relacionada con una solicitud. Además de la información general del viaje o gasto, el informe ofrece un listado de los gastos incluidos en dicha solicitud, con un acumulado total al final del documento.

Para realizar la creación y exportación de informes se ha utilizado la herramienta SQL Server Reporting Services (SSRS). Se trata de un software de generación de informes de Microsoft integrado en los paquetes de Microsoft SQL Server, que permite generar informes dinámicamente, de forma rápida y sencilla.

A continuación se muestra un ejemplo de informe asociado a una solicitud, en este caso una solicitud de un viaje.

endalia
HR Consulting & Software

Informe de solicitud de gastos

Código	00001
Nombre	Reunión en Madrid
Empleado	Bailo Ortiz, Nicolás
Código de empleado	00001
Fecha de solicitud	23/05/2014
Fecha de inicio	02/06/2014
Fecha de fin	06/06/2014
Tipo de solicitud	Nacional
Justificación	Reunión con nuevos clientes en Madrid

Desglose de gastos asociados

Fecha	Tipo	Cantidad	Total
02/06/2014	Dieta completa	4,00	176,00 €
06/06/2014	Media dieta	1,00	26,00 €
02/06/2014	Taxi	-	42,85 €
02/06/2014	Taxi	-	30,00 €
03/06/2014	Taxi	-	6,30 €
04/06/2014	Taxi	-	6,20 €
05/06/2014	Taxi	-	28,00 €
06/06/2014	Taxi	-	7,35 €
06/06/2014	Taxi	-	30,00 €
06/06/2014	Taxi	-	33,60 €
		IMPORTE A PERCIBIR	**386,30 €**

Figura 12. Ejemplo de informe generado para una solicitud de viaje.

10. BIBLIOGRAFÍA

10.1 Referencias

[R1] I. Jacobson, G. Booch, J. Rumbaugh. 2000. “El Proceso Unificado de Desarrollo de Software”. Pearson Education.

10.2 Referencias web

[W1] http://www.microsoft.com/net

[W2] http://www.wikipedia.org

[W3] http://www.uml.org

[W4] http://www.endalia.com

endalia®
HR Consulting & Software

PRUEBAS

DESARROLLO DE UN MÓDULO DE GESTIÓN DE VIAJES Y GASTOS

VERSIÓN 1.0
PUBLICADO EL 29/08/2016

HISTÓRICO DE REVISIONES

Fecha	Versión	Descripción	Autor
23/08/2016	1.0	Redacción inicial del documento	Nicolás Bailo Ortiz

ÍNDICE

1. INTRODUCCIÓN

1.1 Propósito del documento

El presente documento describe la fase de pruebas del módulo de viajes y gastos desarrollado. El objetivo de esta fase es el de asegurar la calidad del software y el correcto funcionamiento de la implementación realizada en la fase anterior, así como garantizar que se cumplen los requisitos planteados al comienzo de la realización de este trabajo.

1.2 Alcance del documento

Este documento contiene los resultados obtenidos en la última de las fases del trabajo: la realización de pruebas del módulo de viajes y gastos.

1.3 Acrónimos

- PU: Prueba Unitaria
- PS: Prueba de Sistema

1.4 Definiciones

- Caso de prueba: conjunto de condiciones o variables bajo las cuáles un analista determinará si una aplicación, un sistema software, o una característica de éstos es parcial o completamente satisfactoria.
- Prueba de sistema: conjunto de verificaciones de la interconexión de varios subsistemas dentro de una aplicación o sistema software.
- Prueba unitaria: conjunto de verificaciones de un subsistema de una aplicación o sistema software.

1.5 Referencias

Este documento contiene referencias a los siguientes documentos del trabajo, contenidos en la sección “Anexos”:

- Especificación de requisitos: documento que describe la especificación de requisitos realizada a partir de las necesidades del cliente.
- Análisis del sistema: documento que describe la arquitectura del sistema desarrollado, junto con un listado de casos de uso y su estructura de paquetes.

1.6 Resumen

En este documento se presentan el proceso y los resultados de la fase de pruebas del Módulo de Gestión de Viajes y Gastos. Se compone de los cinco apartados siguientes:

1. Introducción, propósito y alcance del documento.
2. Descripción del proceso de pruebas realizado en este proyecto.
3. Descripción y resultados de las pruebas unitarias.

4. Descripción y resultados de las pruebas de sistema.
5. Presentación de la bibliografía utilizada para la realización de este documento.

2. DESCRIPCIÓN DEL PROCESO

Se han realizado una serie de pruebas sobre el módulo desarrollado con el objetivo de asegurar tanto el correcto funcionamiento como la calidad del software realizado. Se puede introducir esta fase en cualquier punto del desarrollo de un software, dependiendo del tipo de pruebas que se vayan a realizar. En este caso, y dado que el objetivo final del trabajo es la entrega de un software funcional a un cliente, se ha decidido colocar estas pruebas como la última de las fases del desarrollo software, para obtener la máxima información posible sobre la calidad del sistema entregado al cliente. Por otro lado, cabe destacar que el hecho de que esta fase sea la última no significa que el desarrollo termine necesariamente aquí. Si el módulo no superara alguna de las pruebas realizadas se deberá volver a una fase anterior para realizar los cambios necesarios en el software. Se trata por lo tanto de un proceso circular.

Para este desarrollo en concreto se han planteado dos tipos de pruebas bien diferenciadas:

1. Pruebas unitarias.
2. Pruebas de sistema.

Las pruebas unitarias responden a la necesidad de testear pequeños componentes del módulo que por sus implicaciones en el conjunto del sistema se consideran más críticas o importantes. Se trata de casos fácilmente aislables y, por lo tanto, sencillos de comprobar de forma semiautomática.

Por otro lado, las pruebas de sistema permiten comprobar la calidad del software desarrollado mediante la validación de la comunicación entre los distintos subsistemas que lo componen. Se tratan en este de casos de pruebas más amplios, difícilmente automatizables, pero que aseguran el correcto funcionamiento de la totalidad del módulo en su conjunto. Idealmente, y en el seno de la organización en la que se realiza este trabajo, estas pruebas siempre se realizan por un equipo independiente al que ha desarrollado el sistema. No obstante, dada la naturaleza académica de este proyecto, dichas pruebas fueron realizadas con anterioridad por el responsable del trabajo, para luego ser nuevamente validadas por el equipo independiente.

Por último, cabe destacar que el conjunto de pruebas seleccionado responde al último de los propósitos de este trabajo: realizar un software que responda a las necesidades del cliente, las cuales han sido definidas en las fases de requisitos y análisis del proyecto. Por lo tanto, un sistema que supere con éxito la colección de pruebas detalladas a continuación será un sistema que solvente correctamente las problemáticas planteadas por el cliente al comienzo de este trabajo.

3. PRUEBAS UNITARIAS

Una prueba unitaria tiene como objetivo el comprobar el correcto funcionamiento de una parte de código, pudiendo ser un método o un conjunto de éstos. Para realizar correctamente las pruebas unitarias, se han de crear casos de prueba que sean independientes entre ellos y del resto de la aplicación. Así, se podrá comprobar el correcto funcionamiento de dicha parte de código.

Una prueba unitaria debe ser, idealmente:

- Automatizable: debe ejecutarse en su totalidad sin intervención manual.
- Completa: debe abarcar la mayor cantidad de código posible.
- Repetible: debe poder ser ejecutada en múltiples ocasiones.
- Independiente: la ejecución de una prueba no puede afectar a la ejecución de otra.
- Profesional: debe estar documentada y formar parte de la documentación del proyecto.

Dada la gran cantidad de casos unitarios que existen en un proyecto de estas características se presentan únicamente pruebas de algunas partes del código, consideradas críticas o complejas.

Identificador	**PU - 01**
Nombre	Envío de una solicitud.
Descripción	Creación, inserción de datos obligatorios y envío de una solicitud para su posterior aprobación.
Resultados esperados	La solicitud pasa al estado “Pendiente de aprobador” y muestra en su detalle toda la información introducida en la misma.
¿Prueba superada?	Sí.

Identificador	**PU – 02**
Nombre	Cálculo del siguiente aprobador.
Descripción	El cálculo del siguiente aprobador tiene que tener en cuenta varios factores como por ejemplo la estructura organizativa de la empresa, el sistema de privilegios de la aplicación en la que se integra este módulo o algunos campos de la propia solicitud. En esta prueba se envía una solicitud y se comprueba que el aprobador calculado es el correcto.
Resultados esperados	El aprobador correspondiente recibe un correo electrónico notificando que debe aprobar una solicitud en la aplicación.
¿Prueba superada?	Sí.

Identificador	**PU – 03**
Nombre	Generación de un informe.
Descripción	Obtención de los datos relativos a la solicitud de la cual se va a realizar el informe, generación y descarga del mismo.
Resultados esperados	Se descarga un informe en formato PDF con información sobre la solicitud seleccionada, así como un desglose de sus gastos asociados.
¿Prueba superada?	Sí.

4. PRUEBAS DE SISTEMA

Las pruebas de sistema tienen como objetivo validar que se cumplen correctamente los casos de uso definidos anteriormente en el desarrollo del software. Por esto, se han realizado diferentes pruebas de sistema en función de los casos de uso planteados en la fase de análisis.

Estas pruebas permiten, a diferencia de las pruebas unitarias, comprobar el correcto funcionamiento de los diferentes subsistemas que componen este módulo y verificar que éstos se comunican correctamente entre ellos.

Identificador	**PS – 01**
Nombre	Visualización de una solicitud.
Descripción	Se ha de comprobar la correcta visualización de una solicitud.
Resultados esperados	La aplicación debe mostrar la información relativa a la solicitud, con todos los campos que contengan información rellenos.
¿Prueba superada?	Sí.

Identificador	**PS – 02**
Nombre	Creación de una solicitud.
Descripción	Se ha de comprobar el proceso de creación de una solicitud mediante el asistente de creación de solicitudes.
Resultados esperados	La aplicación debe en primer lugar mostrar el asistente de creación de solicitudes. Una vez introducida la información en el asistente debe comprobar que todos los campos obligatorios se han rellenado. Si falta alguna información, se informará al usuario permitiéndole modificar los valores introducidos, sino se creará la solicitud y el sistema dirigirá al usuario a la página de edición de una solicitud.
¿Prueba superada?	Sí.

Identificador	**PS – 03**
Nombre	Modificación de una solicitud.
Descripción	Se ha de comprobar la correcta edición de una solicitud ya creada.
Resultados esperados	La aplicación debe comprobar que todos los campos obligatorios siguen rellenados de forma correcta tras la edición. Si no es así, deberá informar al usuario. Si todo está correcto, deberá actualizar la base de datos con los cambios introducidos por el usuario.
¿Prueba superada?	Sí.

Identificador	**PS – 04**
Nombre	Envío de una solicitud.
Descripción	Se ha de comprobar que el sistema reacciona correctamente ante el envío de una solicitud para su aprobación.
Resultados esperados	Primero, la aplicación deberá realizar un guardado de los datos de la solicitud tal y como se describe en la prueba PS-03. Una vez se ha actualizado la información de la solicitud, el sistema deberá: • Pasar la solicitud a la siguiente fase. • Calcular quien es el siguiente aprobador y notificarle por email que tiene una solicitud pendiente de aprobar.
¿Prueba superada?	Sí.

Identificador	**PS – 05**
Nombre	Generación de un informe relativo a una solicitud.
Descripción	Se ha de comprobar la correcta visualización de la opción de generar informe, ya que no siempre deberá estar disponible, así como la correcta generación del informe con los datos de la solicitud seleccionada.
Resultados esperados	La aplicación deberá mostrar la opción de generar un informe de una solicitud sólo si ésta se encuentra en fase de liquidación, ya ha sido enviada para su aprobación y no ha sido rechazada por un aprobador. Una vez generado el informe, éste debe contener la información básica relativa a dicha solicitud y un listado de sus gastos asociados.
¿Prueba superada?	Sí.

Identificador	**PS – 06**
Nombre	Correcta visualización de las diferentes secciones en función del usuario.
Descripción	Se ha de comprobar la correcta visualización de las secciones del módulo, en función de los privilegios de cada usuario.
Resultados esperados	La aplicación debe mostrar: • La sección “Mis Viajes y Gastos” a todos los usuarios. • La sección “Viajes y Gastos de colaboradores” a todos los usuarios que sean responsables de al menos un empleado. • La sección “Aprobación de Viajes y Gastos” a todos los usuarios que sean responsables de al menos un empleado y a los usuarios que tengan el privilegio de aprobar en cualquiera de las fases de una solicitud.
¿Prueba superada?	Sí.

Identificador	**PS – 07**
Nombre	Aprobación de una solicitud.
Descripción	Se ha de comprobar la correcta aprobación de una solicitud.
Resultados esperados	La aplicación debe calcular cual es la siguiente fase y avanzar la solicitud cuando un responsable pulse el botón correspondiente a aprobar solicitud. Si la nueva fase depende de un aprobador, se deberá notificar a dicho responsable mediante un correo electrónico, sino, deberá notificar al empleado solicitante.
¿Prueba superada?	Sí.

Identificador	**PS – 08**
Nombre	Rechazo de una solicitud.
Descripción	Se ha de comprobar el correcto rechazo de una solicitud.
Resultados esperados	La aplicación debe pasar la solicitud a estado rechazado cuando un responsable pulse el botón correspondiente a rechazar solicitud. Se deberá notificar al empleado solicitante mediante un correo electrónico.
¿Prueba superada?	Sí.

Identificador	**PS – 09**
Nombre	Devolución de una solicitud.
Descripción	Se ha de comprobar la correcta devolución de una solicitud.
Resultados esperados	La aplicación debe pasar la solicitud al estado “No Enviado”, donde el empleado solicitante puede volver a editarla. Se deberá notificar a dicho empleado mediante un correo electrónico.
¿Prueba superada?	Sí.

5. BIBLIOGRAFÍA

5.1 Referencias

Este documento no contiene referencias a textos impresos.

5.2 Referencias web

[W1] http://www.wikipedia.org

[W2] http://www.endalia.com

endalia®
HR Consulting & Software

MANUAL DE USUARIO

DESARROLLO DE UN MÓDULO DE GESTIÓN DE VIAJES Y GASTOS

VERSIÓN 1.0
PUBLICADO EL 29/08/2016

HISTÓRICO DE REVISIONES

Fecha	Versión	Descripción	Autor
23/08/2016	1.0	Redacción inicial del documento	Nicolás Bailo Ortiz

ÍNDICE

1. INTRODUCCIÓN

1.1 Propósito del documento

El presente documento presenta los manuales de usuario del módulo de Gestión de Viajes y Gastos desarrollado. Dicho manual fue entregado a los usuarios del sistema junto con la actualización del sistema que les habilitaba el módulo. Pretende ser una guía que marque el correcto uso de la aplicación, presentando sus diferentes partes y ofreciendo indicaciones de cómo se han de rellenar las solicitudes correctamente.

1.2 Alcance del documento

Este documento contiene los manuales de usuario que se ha entregado a los usuarios del sistema.

1.3 Definiciones

- Guía de uso: documento de comunicación técnica destinado a dar asistencia a las personas que utilizan un sistema en particular.
- Colaborador: un empleado es colaborador de otro cuando el primero está por debajo del segundo en la estructura organizativa de la empresa. Además, se le denomina “colaborador directo” cuando el primer empleado está justo un puesto por debajo del segundo en dicha estructura.

1.4 Referencias

Este documento no contiene referencias a otros documentos del trabajo.

1.5 Resumen

En este documento se presentan los manuales de usuario entregados a los usuarios del Módulo de Gestión de Viajes y Gastos. Al haber dos tipos de usuarios (solicitantes y aprobadores), se han creado dos tipos de manuales diferentes. El documento se compone de los tres apartados siguientes:

1. Introducción, propósito y alcance del documento.
2. Consideraciones de confidencialidad.
3. Manual de usuario solicitante, también llamado *colaborador*.
4. Manual de usuario aprobador.

2. CONSIDERACIONES DE CONFIDENCIALIDAD

Por motivos de confidencialidad se han ocultado o tapado algunas de las imágenes y logos de los manuales mostrados en este documento. Esto se corresponde con los rectángulos de color negro presentes en la mayoría de las páginas de las guías.

3. MANUAL DE USUARIO SOLICITANTE

A continuación se presenta el manual entregado a los usuarios de tipo solicitante (colaborador). Dado el formato origen de esta guía (PowerPoint) se muestra en imágenes, a razón de una por página:

Figura 1. Manual de usuario solicitante, página 1

Figura 2. Manual de usuario solicitante, página 2

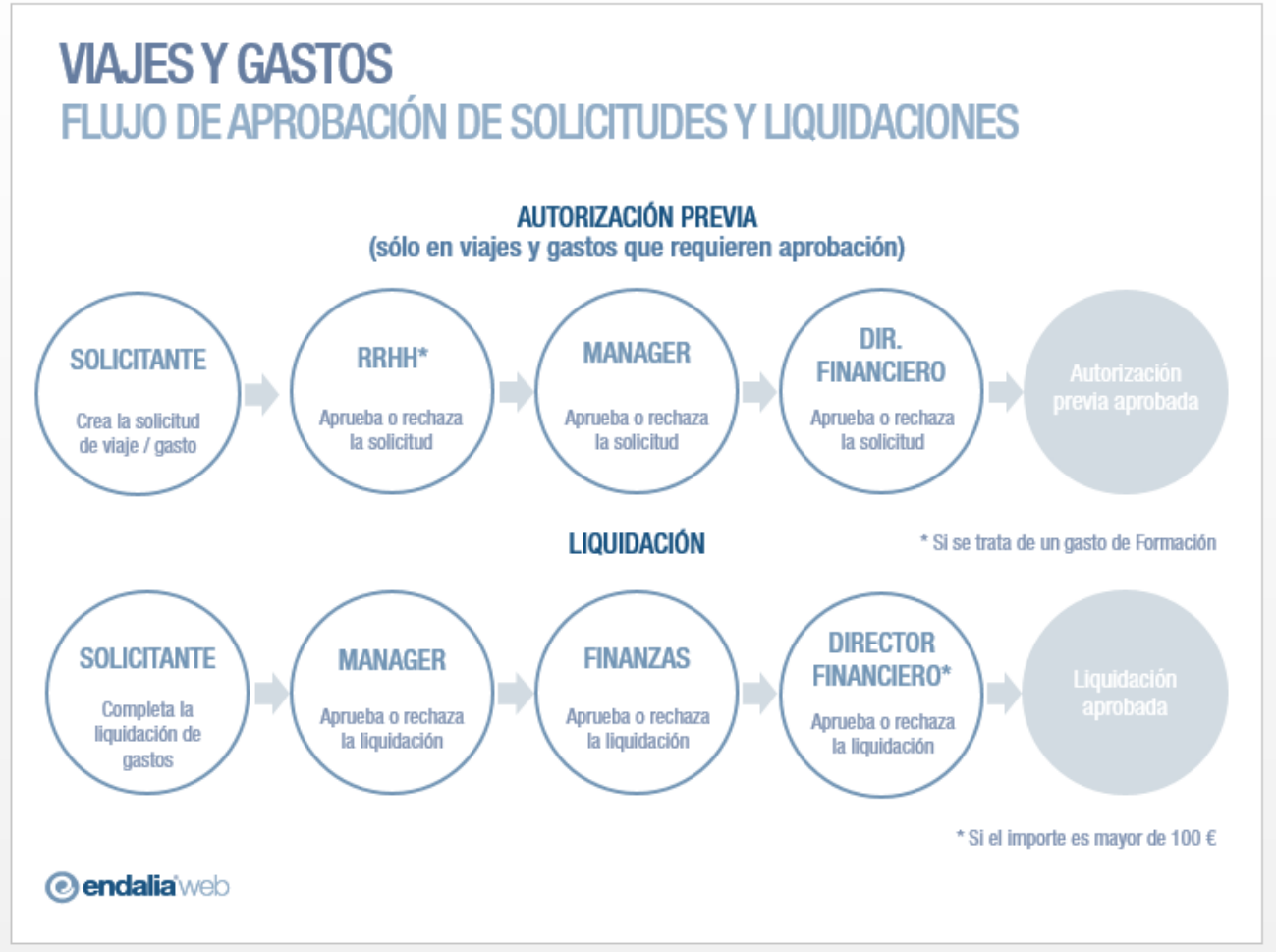


Figura 3. Manual de usuario solicitante, página 3

Figura 4. Manual de usuario solicitante, página 4

Figura 5. Manual de usuario solicitante, página 5

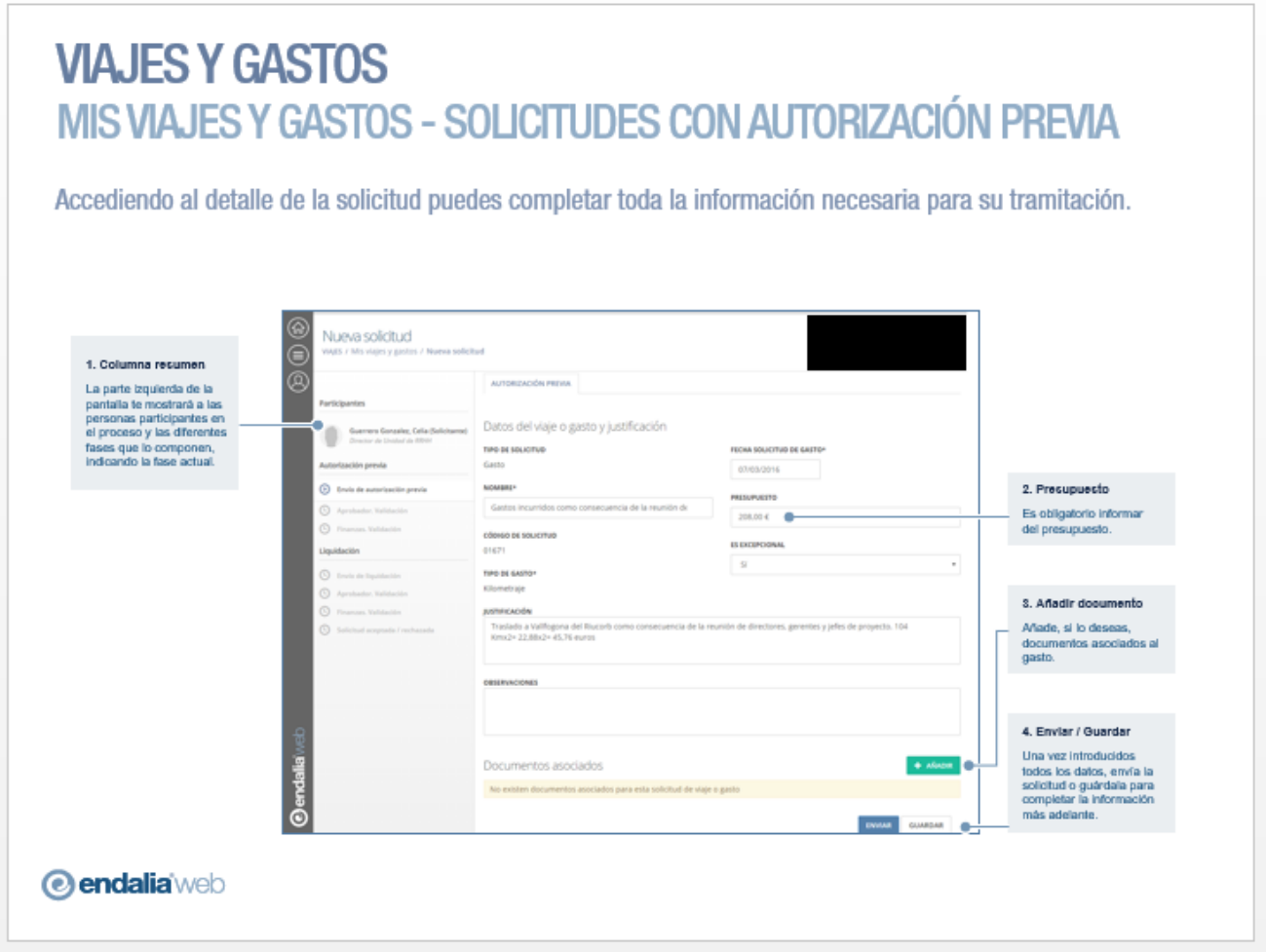


Figura 6. Manual de usuario solicitante, página 6

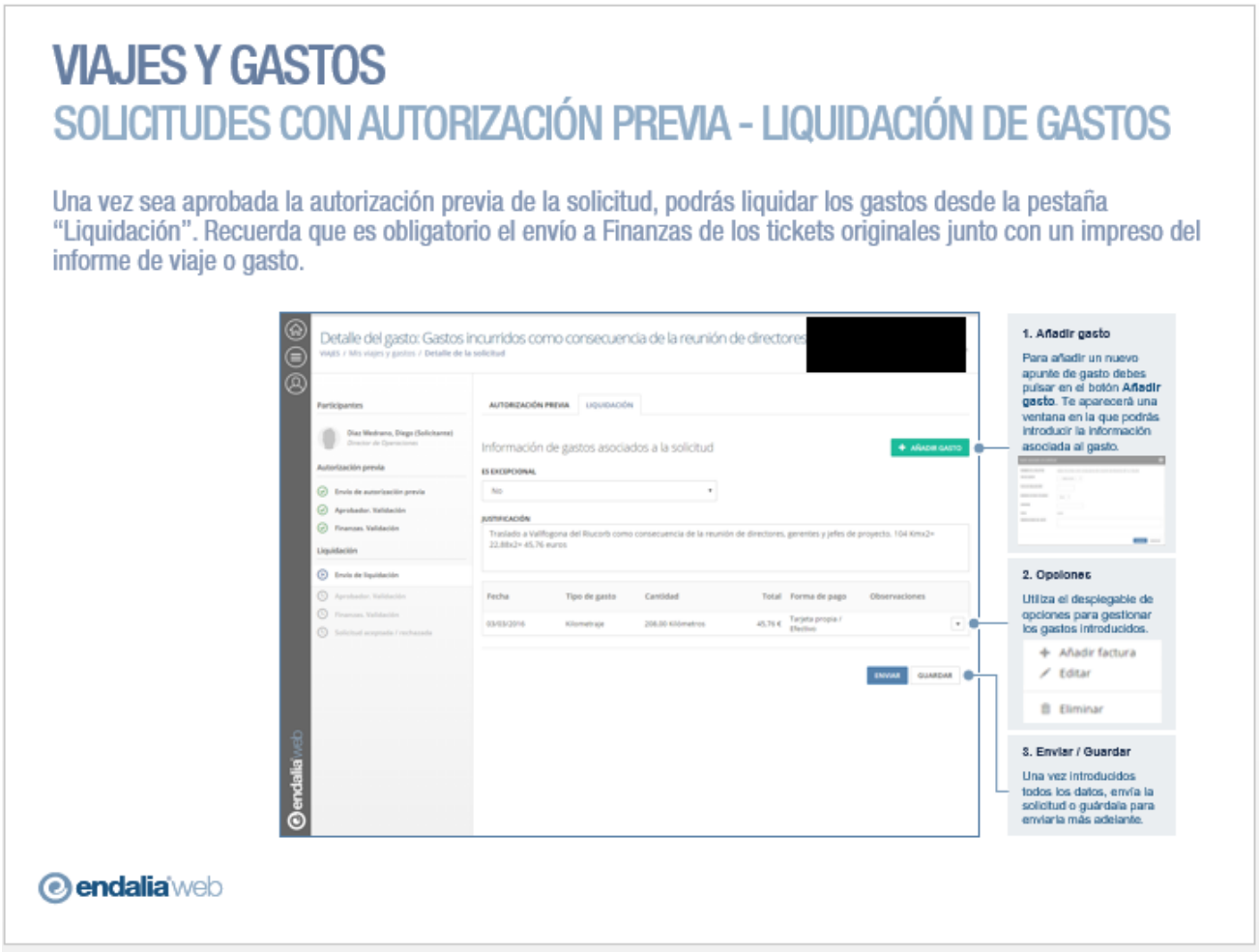
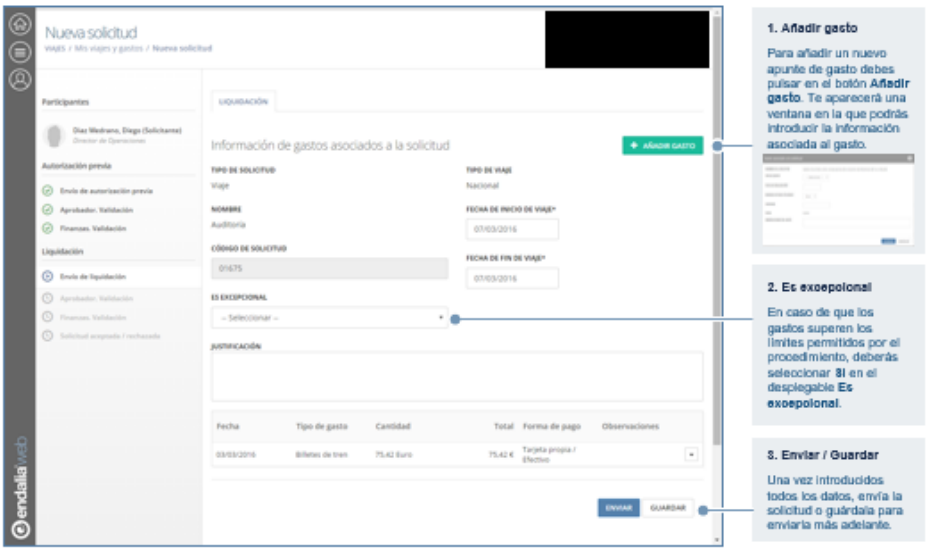


Figura 7. Manual de usuario solicitante, página 7

VIAJES Y GASTOS

MIS VIAJES Y GASTOS - SOLICITUDES SIN AUTORIZACIÓN PREVIA

En los viajes y gastos que no requieren autorización previa, debes introducir los datos básicos del viaje o gasto y los apuntes de gasto antes de enviar la liquidación.



endalia web

Figura 8. Manual de usuario solicitante, página 8

Figura 9. Manual de usuario solicitante, página 9

Figura 10. Manual de usuario solicitante, página 10

4. MANUAL DE USUARIO APROBADOR

A continuación se presenta el manual entregado a los usuarios de tipo aprobador. De nuevo, dado el formato origen de esta guía (PowerPoint) se muestra en imágenes, a razón de una por página:

Figura 11. Manual de usuario aprobador, página 1

Figura 12. Manual de usuario aprobador, página 2

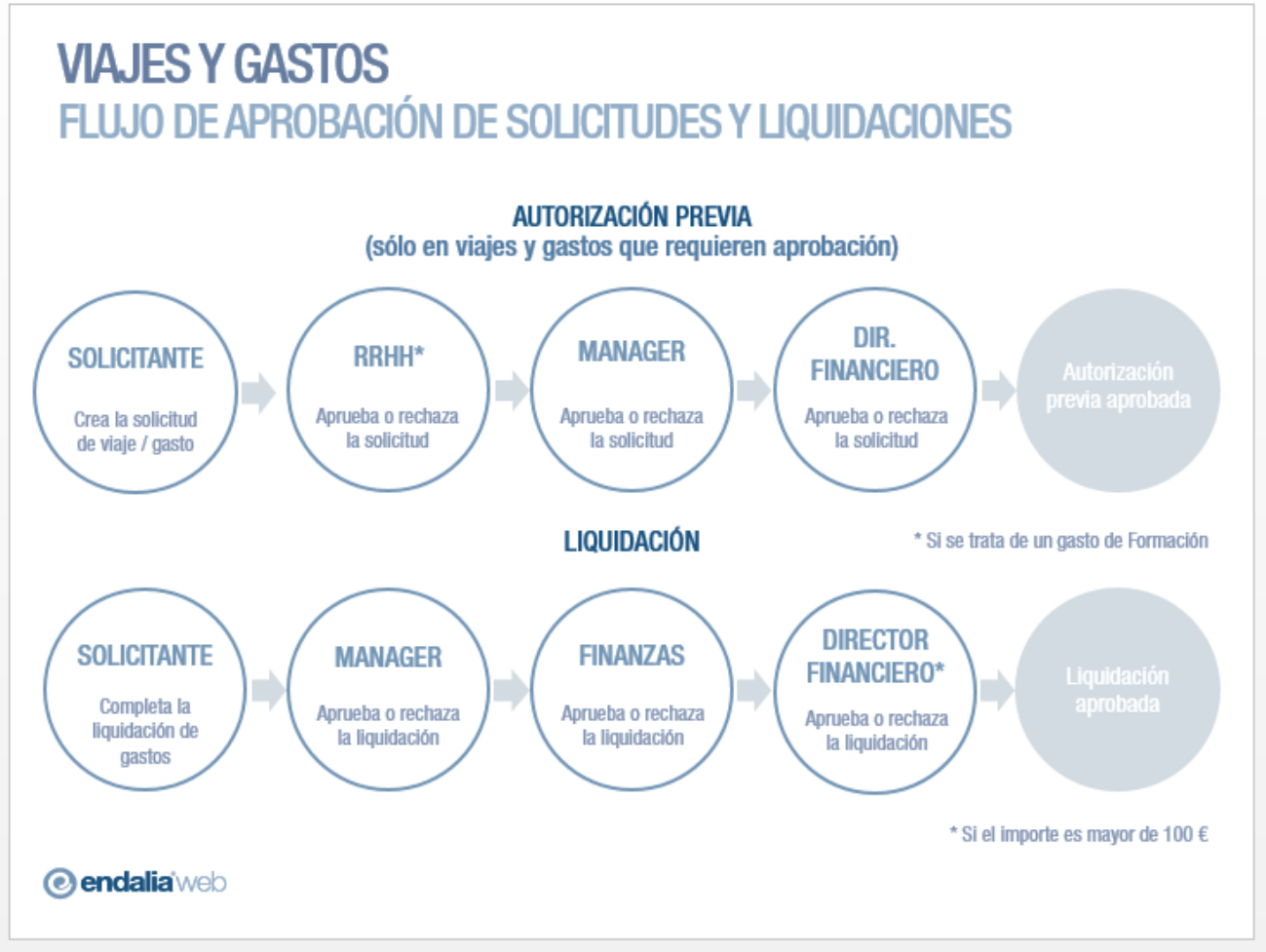


Figura 13. Manual de usuario aprobador, página 3

Figura 14. Manual de usuario aprobador, página 4

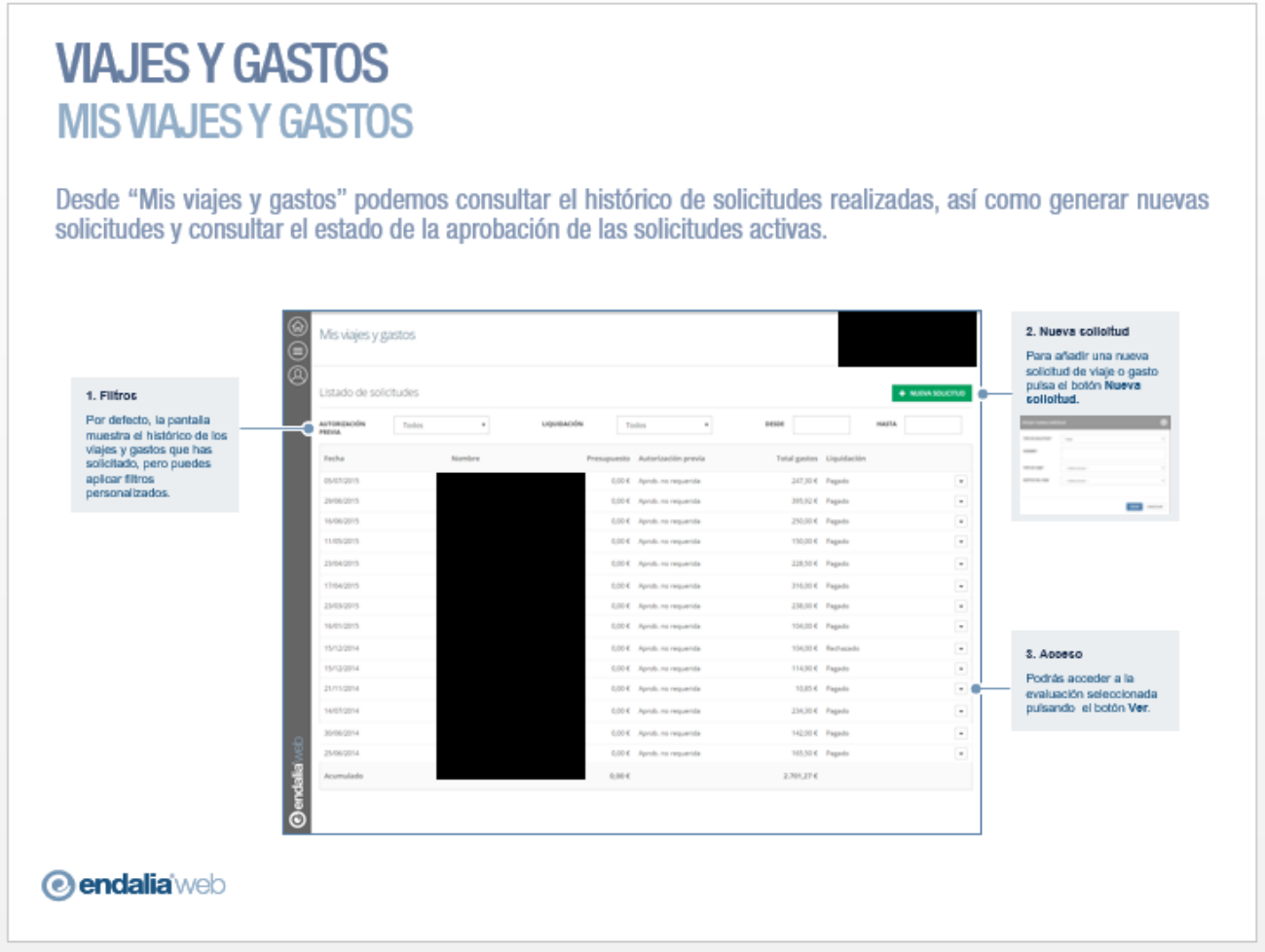


Figura 15. Manual de usuario aprobador, página 5

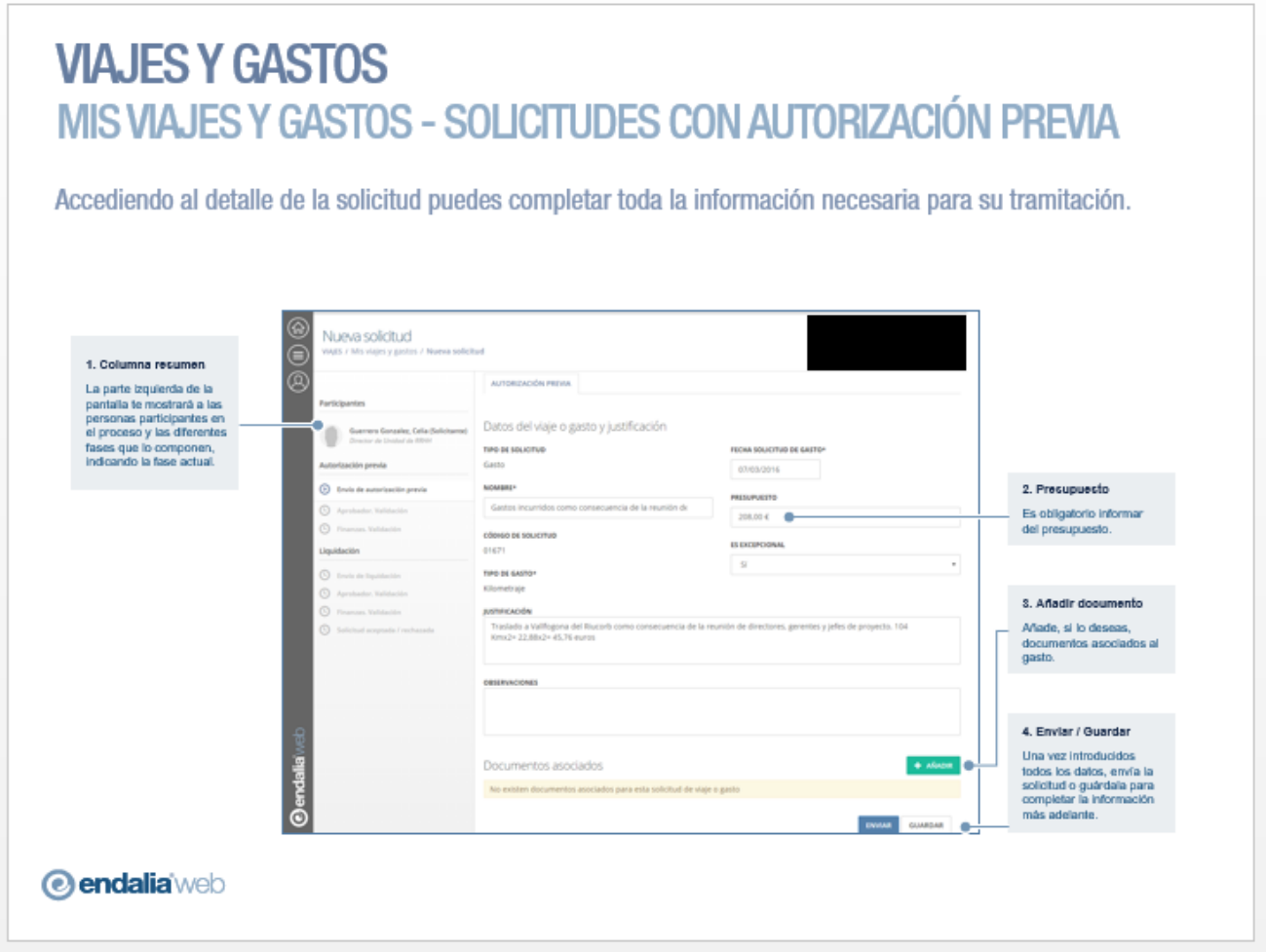


Figura 16. Manual de usuario aprobador, página 6

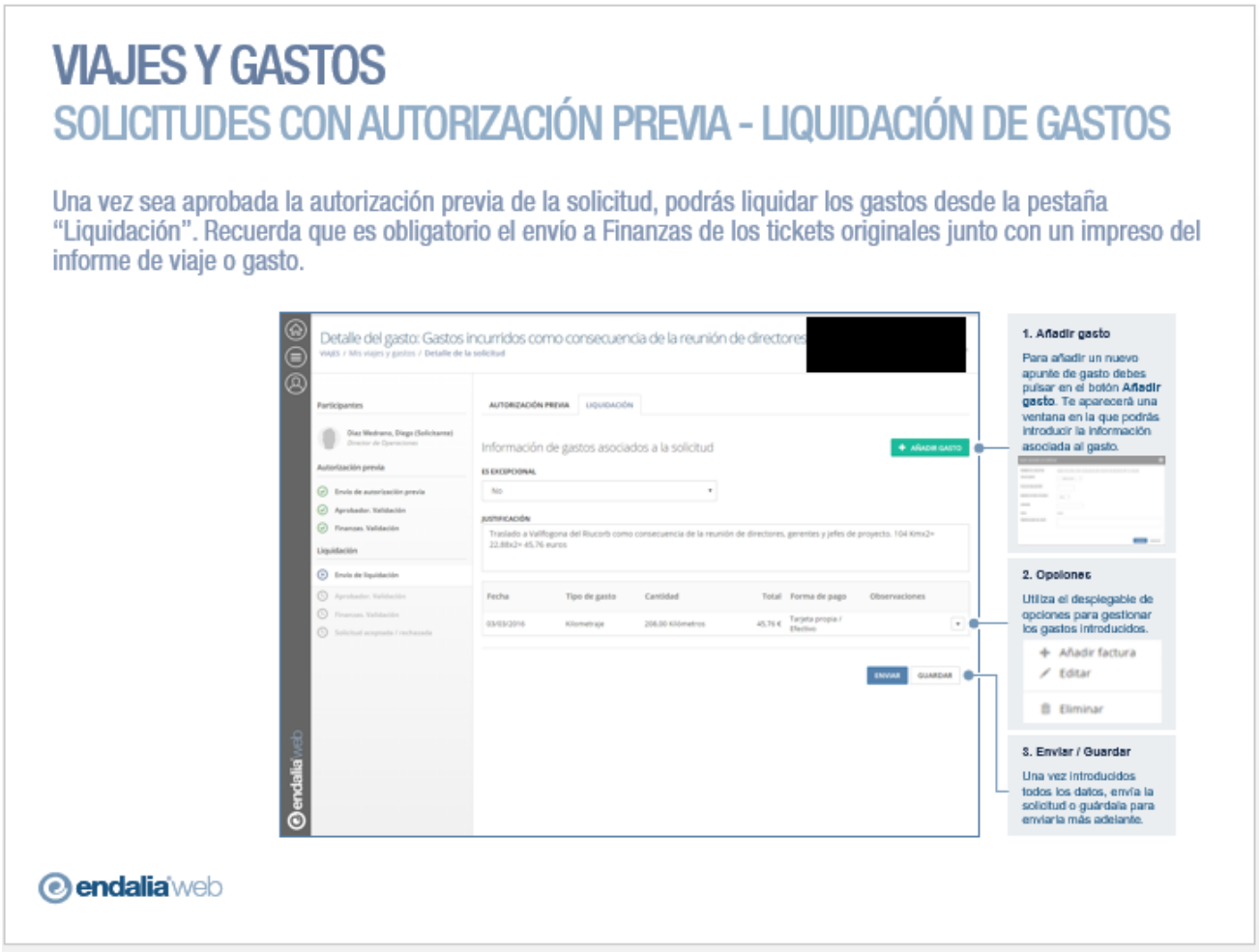


Figura 17. Manual de usuario aprobador, página 7

Figura 18. Manual de usuario aprobador, página 8

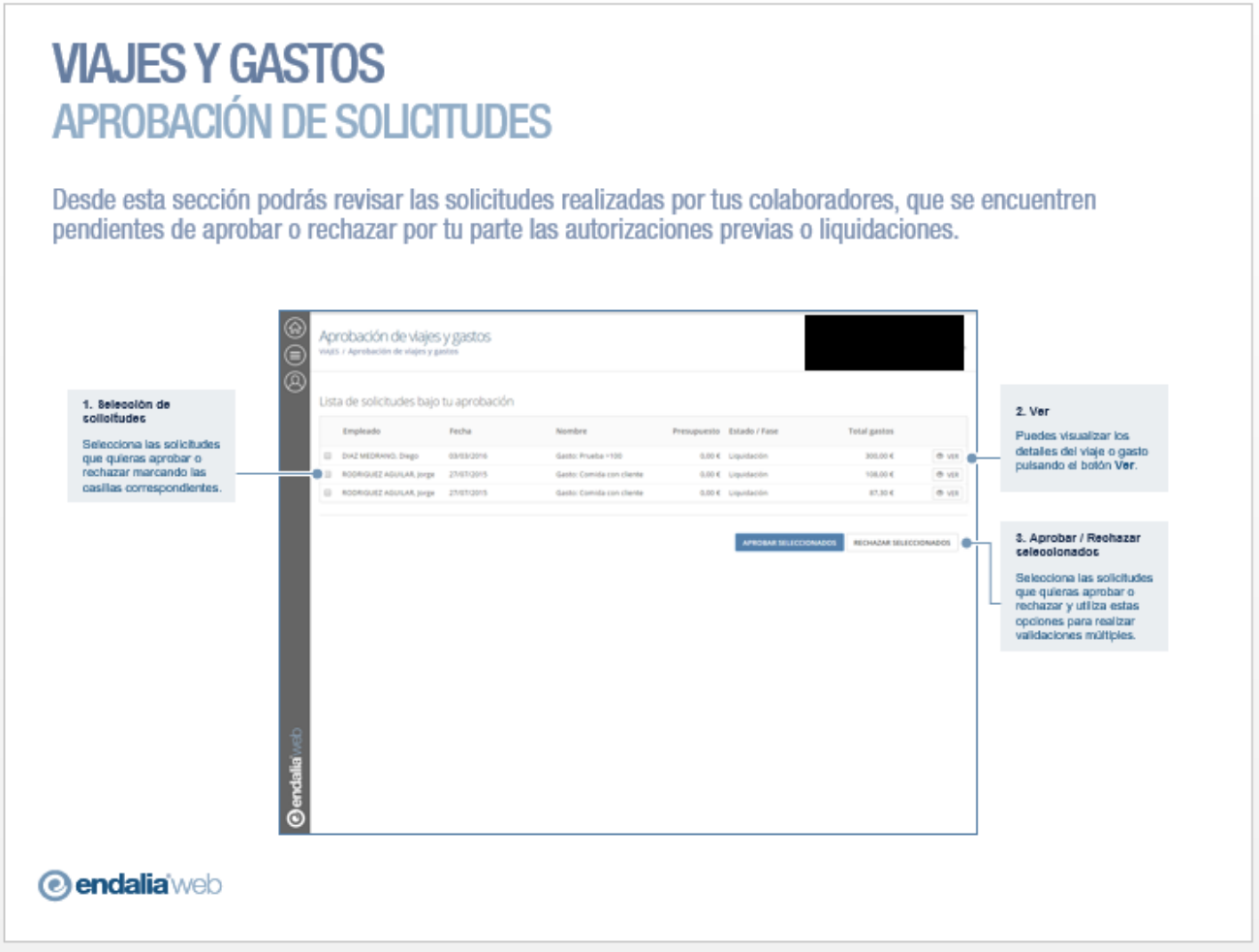


Figura 19. Manual de usuario aprobador, página 9

Figura 20. Manual de usuario aprobador, página10

Figura 21. Manual de usuario aprobador, página11