UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso de Profesionales en Actividad

TÍTULO: Implementación de subsistema de consultoría de programas extrapresupuestarios de la Dirección General de Cultura y Educación de la Provincia de Buenos Aires

AUTOR: Pablo Nahuel Sangiácomo

DIRECTOR ACADÉMICO: Licenciado Federico Cristina

DIRECTOR PROFESIONAL: Ingeniero Leandro van der Wildt

CARRERA: Licenciatura en Sistemas

Resumen

Actualmente me desempeño en el Área de Sistemas de Dirección Provincial de Infraestructura Escolar (DPIE), en donde brindamos soporte informático a empleados de DPIE, tanto del Área de liquidaciones extrapresupuestaria de educación de la Dirección General de Cultura y Educación (DGCyE), como de otras reparticiones. El objetivo de esta tesina es detallar el análisis, diseño, desarrollo e implementación de una solución informática que permita llevar a cabo el circuito de liquidación de honorarios de los consultores de programas extrapresupuestarios que facturan a la DGCyE de la Provincia de Buenos Aires, entendiendo por extrapresupuestarios aquellos programas que poseen financiamiento Nacional. Particularmente la tesina se centrará en las actividades llevadas a cabo a lo largo de la creación de dicho sistema.

Palabras Clave

HTML. CSS. Javascript. jQuery. Bootstrap. PHP. Symfony. Web service. Interpretación de documentos. Metodologías ágiles. SCRUM. XP. AFIP. Honorarios. Retenciones. Autogestión

Trabajos Realizados

Se analizaron las tecnologías utilizadas en el desarrollo de aplicaciones WEB y la aplicación de metodologías ágiles para el desarrollo del mismo.
Se analizaron los requerimientos y se diseñó, desarrolló e implementó una solución informática para cubrir los requerimientos solicitados.
Se analizaron los resultados obtenidos de una encuesta realizada a los usuarios finales

Conclusiones

Se implementó una solución informática al proceso de facturación y liquidación de los contratos extrapresupuestarios de la DGCyE. Se pudo comprobar que la utilización de metodologías ágiles, como XP resulta muy útil al desarrollar sistemas con requerimientos cambiantes. Se estudiaron y analizaron aproximaciones para la administración y gestión de documentos digitales. Se ha logrado enriquecer la interacción de los webservices gracias a una expansión en la lógica relacionada con bitácoras y manejo de excepciones. La implementación del sistema fue una experiencia positiva, como se refleja en la encuesta realizada a usuarios finales en donde hubo comentarios favorables sobre la mejora en la metodología de trabajo.

Trabajos Futuros

- Integración con los servicios de Gestión Documental Electrónica Buenos Aires (GDEBA) para llevar el seguimiento de los documentos electrónicos.
- Integración con el programa de Interoperabilidad de la Subsecretaría para la Modernización del Estado.
- Perfeccionar el circuito ante fallos ocasionados al momento en el que el consultor intenta adjuntar un comprobante.
- Mejorar la interfaz para dispositivos móviles para facilitar la carga de facturas a los consultores.

Fecha de la presentación: Diciembre 2019

ÍNDICE

1. Introducción

En esta sección se presentará la motivación y objetivos de la tesina conforme a los conceptos que se van a tratar en la tesina y la manera en que está organizada la misma.
En la primera parte se va a presentar el dominio de la tesina desarrollada y en la segunda se va a hacer una breve descripción del resto de los capítulos que la componen.

1.1. Motivación

Actualmente trabajo en el Área de Sistemas de Dirección Provincial de Infraestructura Escolar (DPIE), en donde brindamos soporte informático a empleados de DPIE, tanto del Área de liquidaciones extrapresupuestaria de educación de la Dirección General de Cultura y Educación, como de otras reparticiones. Desde nuestra área desarrollamos e implementamos un sistema de gestión denominado CASIA. Es un software que se encuentra en continuo crecimiento, y sobre el cual nuevas funcionalidades son incorporadas al mismo constantemente.
Actualmente es utilizado en la carga y gestión de expedientes a través de las distintas áreas, internas y externas; administración de disposiciones y resoluciones; gestión de las obras en sus distintas etapas entre las que se destacan anteproyecto, proyecto, licitación, contratación, facturación, liquidación y pago; gestión de los traslados y viáticos desde su carga en el sistema hasta la liquidación y pago, entre otras funcionalidades.
Uno de los subsistemas que desarrollé para el mismo es el de consultoría y autogestión. La tesina va a centrarse en este subsistema. Se realizará un recorrido por el análisis de los requerimientos, el diseño de las entidades y el proceso de desarrollo realizado para poder satisfacer las funcionalidades solicitadas. El mismo fue realizado para el Área de Consultoría, la cual se encarga de recibir y liquidar los comprobantes de los consultores que facturan a los programas extrapresupuestarios de la Dirección General de Cultura y Educación de la provincia de Buenos Aires. Mensualmente reciben de los consultores más de 4000 comprobantes por email como documentos adjuntos. Luego controlan uno a uno si cumple con los requerimientos de la contratación y proceden a liquidarlos y realizar los pagos. Dada la cantidad de comprobantes que se deben manejar mensualmente y el volumen de controles que el personal tiene que realizar, resultaba imprescindible la automatización de las mismas. Analizado el problema planteado, se propuso que la carga de los comprobantes en el sistema sea autogestionada por cada consultor. De esta manera,

el personal puede enfocar su esfuerzo en la carga de la información de los consultores, los contratos y en realizar controles vía sistema en forma semiautomática. Indudablemente esta automatización no puede reemplazar la intervención de los liquidadores en los controles, pero sí puede optimizar y mejorar los controles manuales debido a que se disminuyen los errores de carga humana.

1.2. Objetivos

Analizar, diseñar, desarrollar e implementar una solución informática que permita llevar a cabo el circuito de liquidación de honorarios de los consultores de programas extrapresupuestarios que facturan a la Dirección General de Cultura y Educación (DGCyE) de la Provincia de Buenos Aires, entendiendo por extrapresupuestarios aquellos programas que poseen financiamiento Nacional. El sistema a desarrollar respetará los requerimientos planteados por el Área de Consultoría de dicha dirección.
Para llevar a cabo el desarrollo propuesto, se incorporará un nuevo subsistema al actual sistema, el cual fue previamente desarrollado por el área de Sistemas de la Dirección Provincial de Infraestructura Escolar. Particularmente la tesina se va a centrar en el análisis, diseño y desarrollo de dicho subsistema. En cuanto al relevamiento de requerimientos, el mismo será llevado a cabo por los analistas del Área de Sistemas, al igual que los testeos de funcionalidad y las capacitaciones a los correspondientes usuarios.

1.3. Estructura

La tesis está conformada de la siguiente manera:

Capítulo 2: Aplicaciones WEB. Se describen las características generales de las aplicaciones WEB, así como también se detalla el modelo cliente/servidor y se presenta un background teórico de las tecnologías utilizadas en el desarrollo del sistema tanto en el lado del servidor, como en el lado del cliente.

Capítulo 3: Diseño de la solución. Esta sección estará abocada a presentar cómo se diseñó el sistema implementado, presentando las etapas del desarrollo del mismo. Posteriormente se detallan los requerimientos del sistema desarrollado acompañando de documentos de análisis y mockups de diseño.

Capítulo 4: Metodologías de trabajo utilizadas. Se abordarán las metodologías de trabajo utilizadas en el desarrollo del sistema elaborado, centrándose en las metodologías ágiles.

Capítulo 5: Desarrollo del sistema. Se describe el proceso de desarrollo del sistema, haciendo hincapié en incumbencias técnicas encontradas durante el proceso.

Capítulo 6: Implementación y análisis de resultados obtenidos. Se presentarán los resultados obtenidos luego de la implementación del sistema y se contrastará con los requerimientos iniciales.

Capítulo 7: Conclusiones y Trabajos Futuros. Por último, se realizará un estudio del proceso de análisis, diseño, desarrollo e implementación utilizado en la realización del sistema. Adicionalmente se repasan algunas modificaciones o funcionalidades para implementar a futuro.

2. Aplicaciones WEB

Como se mencionó en el capítulo anterior, el nuevo sistema se incorporó a otro sistema existente. El sistema desarrollado es una aplicación WEB. Una aplicación web (web-based application) es un tipo especial de aplicación cliente/servidor, donde tanto el cliente (el navegador, explorador o visualizador) como el servidor (el servidor web) y el protocolo mediante el que se comunican (HTTP[1]) están estandarizados y no han de ser creados por el programador de aplicaciones. El protocolo HTTP forma parte de la familia de protocolos de comunicaciones TCP/IP[2], que son los empleados en Internet. Estos protocolos permiten la conexión de sistemas heterogéneos, lo que facilita el intercambio de información entre distintas computadoras. En las aplicaciones web suelen distinguirse tres niveles: el nivel superior que interacciona con el usuario (el cliente web, normalmente un navegador), el nivel inferior que proporciona los datos (la base de datos) y el nivel intermedio que procesa los datos (el servidor web).

2.1. Arquitectura Cliente/Servidor

Previamente se mencionaron los términos cliente y servidor. Como es una aplicación WEB, el sistema se desarrolló basándose en la arquitectura Cliente/Servidor.

Cliente/servidor es una arquitectura de red en la que cada ordenador o proceso en la red es cliente o servidor. Normalmente, los servidores son ordenadores potentes dedicados a gestionar unidades de disco (servidor de archivos), impresoras (servidor de impresoras), tráfico de red (servidor de red), datos (servidor de bases de datos) o incluso aplicaciones (servidor de aplicaciones), mientras que los clientes son máquinas menos potentes y usan los recursos que ofrecen los servidores. Dentro de los clientes se suelen distinguir dos clases: los clientes inteligentes (rich client) y los clientes tontos (thin client). Los primeros son ordenadores completos, con todo el hardware y software necesarios para poder funcionar de forma independiente. Los segundos son terminales que no pueden funcionar de forma independiente, ya que necesitan de un servidor para ser operativos. En nuestra aplicación, los clientes son inteligentes debido a que usan un navegador web para acceder y utilizar el mismo.

[1] Protocolo de transferencia de hipertexto (en inglés: Hypertext Transfer Protocol)

[2] Protocolo de control de transmisión (en inglés Transmission Control Protocol) y Protocolo de Internet (en inglés Internet protocol)

En los primeros tiempos de la computación cliente-servidor, cada aplicación tenía su propio programa cliente que servía como interfaz de usuario que tenía que ser instalado por separado en cada computadora personal de cada usuario. El cliente realizaba peticiones a otro programa —el servidor— que le daba respuesta. Una mejora en el servidor, como parte de la aplicación, requería normalmente una mejora de los clientes instalados en cada computadora personal, añadiendo un coste de soporte técnico y disminuyendo la productividad.

A diferencia de lo anterior, las aplicaciones web generan dinámicamente una serie de páginas en un formato estándar, como HTML o XHTML, soportados por los navegadores web comunes. Se utilizan lenguajes interpretados en el lado del cliente, directamente o a través de plugins tales como JavaScript, Java, etc., para añadir elementos dinámicos a la interfaz de usuario. Generalmente cada página web en particular se envía al cliente como un documento estático, pero la secuencia de páginas ofrece al usuario una experiencia interactiva. Durante la sesión, el navegador web interpreta y muestra en pantalla las páginas, actuando como cliente para cualquier aplicación web.

Otro tema a considerar es que cuando se indica el término cliente, no se refiere únicamente a las computadoras, sino también a todo dispositivo que tenga conexión a internet y que soporte la utilización de un navegador. Hoy en día tenemos, además de computadoras de escritorio, notebooks, laptops, netbooks, smartphones, tablets y en los últimos años se han agregado dispositivos como heladeras gracias al crecimiento de internet de las cosas[3]. En la Figura 2.1 se puede apreciar una variedad de dispositivos clientes que pueden interactuar con los servidores.

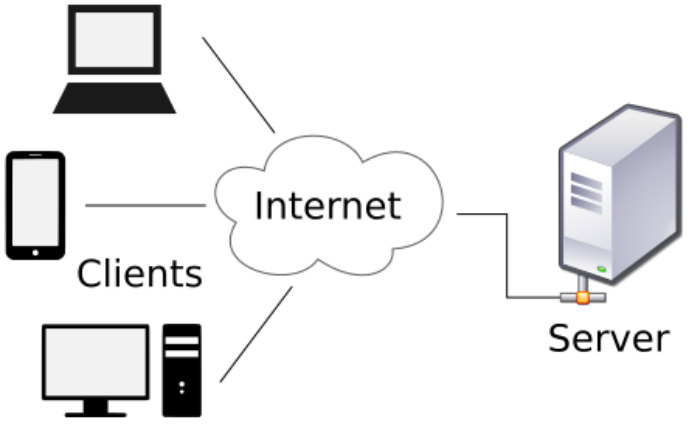

Figura 2.1: Ejemplos de diversos clientes

[3] El internet de las cosas (en inglés, Internet of Things, abreviado IoT; IdC, por sus siglas en español) se refiere a la interconexión en red de objetos cotidianos, que a menudo están equipados con inteligencia ubicua [28]

2.1.1. Cliente

El cliente web es un programa con el que interacciona el usuario para solicitar a un servidor web el envío de los recursos que desea obtener mediante HTTP. En la Figura 2.2 se muestra como es la interacción entre los clientes y el servidor.

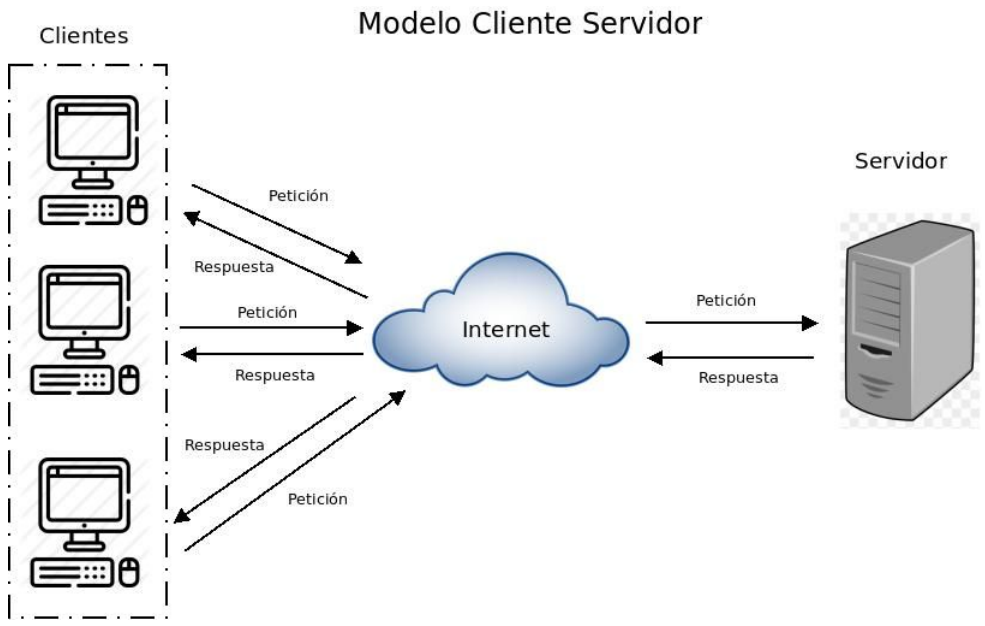

Figura 2.2: Modelo cliente/servidor

La parte cliente de las aplicaciones web suele estar formada por el código HTML que forma la página web más algo de código ejecutable realizado en lenguaje de script del navegador (JavaScript o VBScript) o mediante pequeños programas (applets) realizados en Java. Por tanto, la misión del cliente web es interpretar las páginas HTML y los diferentes recursos que contienen (imágenes, sonidos, etc.). Las tecnologías que se suelen emplear para programar el cliente web son: HTML, CSS, JavaScript, las cuales se describirán en secciones posteriores.

2.1.2. Servidor

Un cliente es un solicitante de servicios y un servidor es un proveedor de servicios. Los servidores típicos son servidores de archivos, servidores de bases de datos y servidores de impresoras de línea. Las arquitecturas cliente / servidor se basan en patrones arquitectónicos cliente/servicio, el más simple de los cuales consiste en un servicio y múltiples clientes. Este patrón tiene varias variaciones. Además, se deben considerar ciertas decisiones sobre el diseño de arquitecturas cliente / servidor, como por ejemplo si el servidor debe diseñarse como un subsistema secuencial o concurrente, qué patrones de estructura arquitectónica usar para el diseño de la arquitectura cliente/servidor, y qué patrones de comunicación arquitectónica usar para la interacción entre los clientes y los servicios.

Un servidor es un sistema de hardware/software que proporciona uno o más servicios para múltiples clientes. Un servicio en un sistema cliente / servidor es un componente de software de aplicación que satisface las necesidades de múltiples clientes. A veces, un servidor admitirá sólo un servicio o quizás más de uno; por otro lado, un servicio grande puede abarcar más de un nodo de servidor. En los sistemas cliente/servidor, el servicio se ejecuta en un nodo de servidor fijo y el cliente tiene una conexión fija al servidor.

A continuación, se describe una variedad de patrones de estructura arquitectónica de software de cliente/servicio que van desde múltiples clientes con un solo servicio hasta múltiples clientes con múltiples servicios y arquitecturas de cliente/servidor de múltiples niveles.

Patrón arquitectónico de cliente múltiple/servicio único

El patrón arquitectónico Cliente múltiple/Servicio único consta de varios clientes que solicitan un servicio y un servicio que cumple con las solicitudes de los clientes. La arquitectura cliente/servidor más simple y común tiene un servicio y muchos clientes, y por esta razón, el patrón arquitectónico de Cliente múltiple/Servicio único también se conoce como el patrón Cliente/Servidor o Cliente/Servicio.

Un ejemplo de este patrón proviene del sistema bancario. Este sistema contiene múltiples cajeros automáticos y un servicio bancario. Para cada cajero automático hay un subsistema de cliente ATM, que maneja las solicitudes de los clientes leyendo la tarjeta ATM y

solicitando detalles de la transacción en el teclado / pantalla. Para una solicitud de retiro aprobada, el cajero automático distribuye efectivo, imprime un recibo y expulsa la tarjeta de cajero automático. El servicio bancario mantiene una base de datos de cuentas de clientes y tarjetas de cajero automático de clientes. Valida las transacciones en cajeros automáticos y aprueba o rechaza las solicitudes de los clientes, según el estado de las cuentas de los clientes.

Patrón arquitectónico de cliente múltiple/servicio múltiple

Los sistemas cliente / servidor más complejos pueden admitir múltiples servicios. En el patrón Cliente múltiple / Servicio múltiple, además de los clientes que solicitan un servicio, un cliente puede comunicarse con varios servicios, y los servicios pueden comunicarse entre sí. En el patrón Cliente múltiple / Servicio múltiple cada servicio reside en un nodo de servidor separado, y el mismo cliente puede invocar ambos servicios. Con este patrón, un cliente podría comunicarse con cada servicio secuencialmente o podría comunicarse con múltiples servicios al mismo tiempo.

Un ejemplo del patrón arquitectónico de Cliente múltiple / Servicio múltiple es un consorcio bancario que consiste en múltiples bancos interconectados. Continuando con el ejemplo de ATM, además de que varios clientes de ATM acceden al mismo servicio bancario, es posible que un cliente de ATM acceda a múltiples servicios bancarios. Esta función permite a los clientes acceder a su propio servicio bancario desde un cliente de cajero automático de un banco diferente.

Patrón arquitectónico de cliente/servicio multinivel

El patrón Cliente / Servicio de varios niveles tiene un nivel intermedio (es decir, una capa) que proporciona un rol de cliente y de servicio. Un nivel intermedio es un cliente de su nivel de servicio y también proporciona un servicio para sus clientes. Es posible tener más de un nivel intermedio. Cuando se ve como una arquitectura en capas, se considera que el cliente está en una capa más alta que el servicio porque el cliente depende y utiliza el servicio.

2.1.3. Ventajas y desventajas del modelo Cliente/Servidor

Se enumeran a continuación las ventajas y desventajas de la arquitectura cliente/servidor

- Ventajas

- ✓ Centralización del control de los recursos, datos y accesos.
- ✓ Facilita la integración entre diferentes sistemas y comparte información permitiendo por ejemplo que las máquinas ya existentes puedan ser utilizadas mediante una interfaz más amigable para el usuario. De esta manera podemos integrar varias PCs con sistemas medianos y grandes sin necesidad de que todos tengan que utilizar el mismo sistema operativo.
- ✓ El modelo cliente servidor permite además proporcionar a las diferentes áreas de una empresa generar un orden de trabajo en donde cada sector puede trabajar en su área, pero accediendo al mismo servidor e información que los demás sin generar conflictos. Esto es de gran utilidad ya que si ponemos como ejemplo una empresa con varios empleados al momento de trabajar es importante que todos puedan hacerlo en simultáneo.
- ✓ Toda la información es almacenada en el lado del servidor, que suele tener mayor seguridad que los clientes
- ✓ Facilidad de mantenimiento y actualización del lado del servidor: Esto es porque el lado del servidor se puede mantener o actualizar fácilmente. Por ejemplo, una actualización se aplica a un único servidor, pero los beneficios los obtienen múltiples clientes generalmente sin necesidad de que éstos actualicen nada.

- Desventajas
  - x Si ocurren fallas del lado del servidor, el servicio queda suspendido para todos los clientes y puede ocasionar pérdida de datos cargados por el cliente como los formularios.
  - x En general los componentes de un servidor son costosos. Por lo que, si algún componente hardware se avería, ponerlo en funcionamiento nuevamente puede ocasionar costos elevados
  - x Si hay muchos clientes conectados al mismo tiempo, el servidor puede saturarse.

2.2. Tecnologías del Servidor

En la sección anterior se describieron los distintos patrones relacionados con la arquitectura Cliente/Servidor. Para nuestro caso práctico, se trata de una arquitectura Cliente múltiple/Servicio único.

El servidor donde está alojado el sistema se compone de un Apache (versión 2.4.18) y corre php 7.0.33 sobre el sistema operativo Ubuntu (versión 16.04.6). El motor de base de datos es MariaDB (versión 10.1.38).

El sistema fue desarrollado utilizando el framework Symfony en su versión 2.8. Recientemente se actualizó a la versión 3.4 de debido a que es una versión LTS[4] y la versión 2.8 tenía sólo soporte para actualizaciones de seguridad.

Un framework es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. Es una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio, y provee una estructura y una especial metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio.

Se decidió utilizar un framework debido a sus grandes ventajas con respecto a realizar un desarrollo desde cero. Entre las mismas, se pueden destacar:

- Utilización de patrones de diseño: los patrones de diseño nos ayudan a encontrar soluciones efectivas a problemas comunes que se nos presentan en el proceso de desarrollo de nuestras aplicaciones web, estos patrones nos presentan esquemas para definir estructuras de diseño para la construcción de aplicaciones, los frameworks dependiendo sus características pueden poseer varios patrones de diseño, el más común es el patrón modelo vista controlador (MVC[13]) que nos permite estructurar nuestra aplicación separando por una capa de datos (modelo), capa de la lógica del negocio (controlador) y capa de presentación (vista).
- Funcionalidades pre-programadas: al construir una aplicación desde cero nos vamos a encontrar con funcionalidades comunes o a veces vamos a requerir de librerías de terceros para implementarlo dentro de nuestra aplicación o finalmente tendremos que desarrollarlo nosotros módulo por módulo. Esto a veces llega a ser un tiempo mal invertido ya que podemos estar reinventando funcionalidades que un framework ya las puede facilitar y de manera mucho más consistente. Esto permite utilizar la

[4] El soporte a largo plazo (en inglés, Long Term Support, abreviadamente, LTS) es un término informático usado para nombrar versiones o ediciones especiales de software diseñadas para tener soportes durante un período más largo que el normal

fuerza de trabajo en el desarrollo de funcionalidades específicas y particulares de nuestra aplicación.

- Estructura de directorios y archivos: utilizando un framework podemos adquirir disciplina siguiendo una línea de convenciones en los directorios y archivos que define el framework. Esto nos ayuda a ser más organizados de tal forma que cuando nuestra aplicación crezca ya sabremos donde estarán situados cada uno de los componentes del mismo. Los directorios y archivos deben seguir un patrón de nombres según las convenciones que nos exija el framework a utilizar.
- Seguridad: un framework nos puede brindar herramientas integradas como ser validaciones de datos, prevenciones a inyecciones SQL, ataques XSS, CSRF, modificación de cookies, entre otros.
- Código limpio y ordenado: al seguir las convenciones recomendadas por el framework a la hora de programar y organizar los archivos, se obtendrá un código más limpio y ordenado.
- Código mantenible: al tener un código más limpio y ordenado obtendremos un código mantenible. Al momento que nuestra aplicación crezca o queramos hacer algún cambio significativo o simplemente para un futuro cuando otros desarrolladores o nosotros mismos volvamos a ver nuestro código sepamos donde se encuentra ubicado cada módulo, función o componente y si anteriormente hemos seguido las reglas de tener un código bien estructurado, limpio y ordenado como resultado tendremos un código mantenible.
- Actualizaciones y mejoras: cuanto más popular es un framework, más desarrolladores trabajarán en nuevas funcionalidades y mejoras de performance y seguridad. Esto nos permite ir mejorando la aplicación a medida que vayan saliendo nuevas versiones del framework.
- Plugins e integración con otras herramientas: los frameworks tienen una serie de plugins desarrollados bajo las mismas convenciones del mismo que nos brindan funcionalidades específicas. También está la integración con otras herramientas lo cual se convierte en otra facilidad más para los desarrolladores.

Como se describe en [9], Symfony es un completo framework diseñado para optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web.

Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El

resultado de todas estas ventajas es que no se debe “reinventar la rueda” cada vez que se crea una nueva aplicación web.
Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows. Symfony es un framework libre.

Previamente se mencionó el patrón Modelo Vista Controlador (MVC). El mismo es un patrón de diseño de software comúnmente utilizado para desarrollar interfaces de usuario que divide la lógica del programa relacionada en tres elementos interconectados. Esto se hace para separar las representaciones internas de la información de las formas en que la información es presentada y aceptada por el usuario. Seguir el patrón arquitectónico MVC desacopla estos componentes principales permitiendo la reutilización de código y el desarrollo paralelo.
Sus tres componentes son:

- Modelo: El componente central del patrón. Es la estructura de datos dinámicos de la aplicación, independiente de la interfaz de usuario. Gestiona directamente los datos, la lógica y las reglas de la aplicación.
- Vista: cualquier representación de información como un gráfico, diagrama o tabla. Son posibles múltiples vistas de la misma información, como un gráfico de barras para la administración y una vista tabular para los contadores.
- Controlador: acepta la entrada y la convierte en comandos para el modelo o la vista.

Además de dividir la aplicación en estos componentes, el diseño modelo-vista-controlador define las interacciones entre ellos. El modelo es responsable de administrar los datos de la aplicación. Recibe la entrada del usuario desde el controlador. La vista significa la presentación del modelo en un formato particular. El controlador responde a las interacciones de entrada de usuario y realiza interacciones sobre los objetos del modelo de datos. El controlador recibe la entrada, opcionalmente la valida y luego pasa la entrada al modelo. En la Figura 2.3 se puede ver el modelo MVC implementado en symfony2.

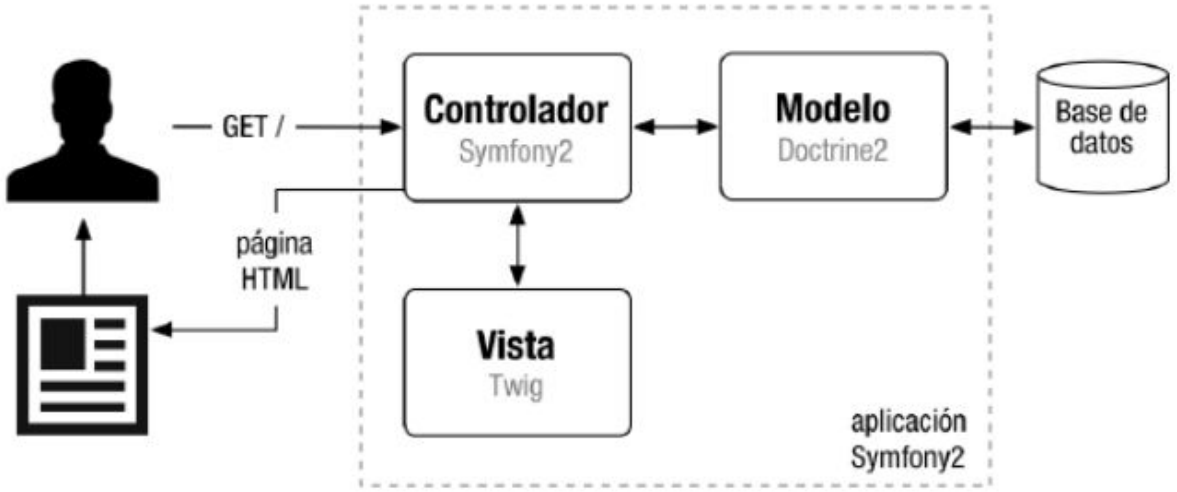

Figura 2.3: MVC en symfony2

2.3. Tecnologías del Cliente

A continuación, vamos a enumerar las tecnologías utilizadas del "lado cliente" para desarrollar el sistema. Cuando se habla de un sitio web, lo primero que viene a la mente es la palabra HTML. La versión más reciente de HTML es la 5.

HTML5 provee básicamente tres características: estructura, estilo y funcionalidad. Nunca fue declarado oficialmente, pero, incluso cuando algunas APIs (Interfaz de Programación de Aplicaciones) y la especificación de CSS3 por completo no son parte del mismo, HTML5 es considerado el producto de la combinación de HTML, CSS y Javascript. Estas tecnologías son altamente dependientes y actúan como una sola unidad organizada bajo la especificación de HTML5. HTML está a cargo de la estructura, CSS presenta esa estructura y su contenido en la pantalla y Javascript hace el resto.

HTML usa un lenguaje de etiquetas para construir páginas web.

CSS es un lenguaje de hojas de estilo que trabaja junto con HTML para proveer estilos visuales a los elementos del documento, como tamaño, color, fondo, bordes, etc.

Javascript es un lenguaje interpretado usado para múltiples propósitos. Los motores más actuales de Javascript, transforman el código Javascript en código máquina con la finalidad de acercarse a velocidades de ejecución similares a aquellas encontradas en aplicaciones de escritorio. Esta mejorada capacidad permitió superar viejas limitaciones de rendimiento y confirmar el lenguaje Javascript como la mejor opción para la web.

Al programar aplicaciones WEB, podemos reutilizar código utilizando librerías o bibliotecas JavaScript. Éstas son archivos de JavaScript que contienen funciones o utilidades que podemos utilizar para solucionar determinados problemas.

Existen ciertas librerías javascript que contienen una serie de herramientas y engloban varias librerías, las cuales pueden ser consideradas frameworks. De igual manera como se mencionó al repasar las tecnologías del servidor, se mencionó que utilizar un framework es una buena práctica. Esto también aplica para el "lado cliente". Sin embargo, es más sencillo incorporar un framework desde un principio de un proyecto. Para el caso de nuestro sistema, se utilizó jQuery y otras librerías compatibles con la misma desde un principio, y actualmente sería muy costoso incorporar frameworks del lado cliente. Algunos frameworks populares y ampliamente utilizados en las aplicaciones web de hoy en día son Angular, React, Vue entre otros.

La librería principal que se utilizó para programar la parte cliente de la aplicación fue jQuery. La misma facilita muchas tareas, su API es simplista y permite un cambio radical en la forma de escribir JavaScript. jQuery se destaca en las siguientes áreas:

- Hace que iterar y atravesar el *DOM*[5] sea mucho más fácil a través de sus diversos métodos integrados.
- Facilita la selección de elementos del DOM para usar selectores, tal como lo haría en CSS.
- Facilita agregar sus propios métodos personalizados a través de su arquitectura de plugins.
- Ayuda a reducir la redundancia en la navegación y la funcionalidad de la interfaz de usuario, como pestañas, CSS y cuadros de diálogo emergentes, animaciones y transiciones, entre otros.

Para acelerar el proceso de desarrollo, se decidió utilizar el framework css Bootstrap. Este framework funciona sobre jQuery y provee un conjunto de componentes con interfaz amigable para el usuario y diseñados para visualizarse correctamente en dispositivos como computadoras, tablets y smartphones. Como se puede apreciar en la Figura 2.4 ofrece varios estilos para mismos componentes con sólo cambiar o agregar clases a tags html. Uno de los componentes principales del mismo es el sistema de grillas. Es clave para

[5] El Modelo de Objetos del Documento (en inglés, Document Object Model) es una interfaz multiplataforma e independiente del lenguaje que trata un documento XML o HTML como una estructura de árbol en la que cada nodo es un objeto que representa una parte del documento

generar interfaces *responsive*[6] ya que divide el ancho de una página en doce columnas y define el porcentaje correspondiente de cada una dependiendo de la resolución de la misma. En la Figura 2.5 se puede observar las distintas clases que se pueden utilizar dependiendo de cuantas columnas queremos que ocupe un elemento. Se puede ver que las clases contienen xs. Éstas determinan cómo se ven los elementos en dispositivos extra pequeños. También provee las mismas clases para dispositivos medianos, grandes y extra grandes con las clases sm, md y lg respectivamente.

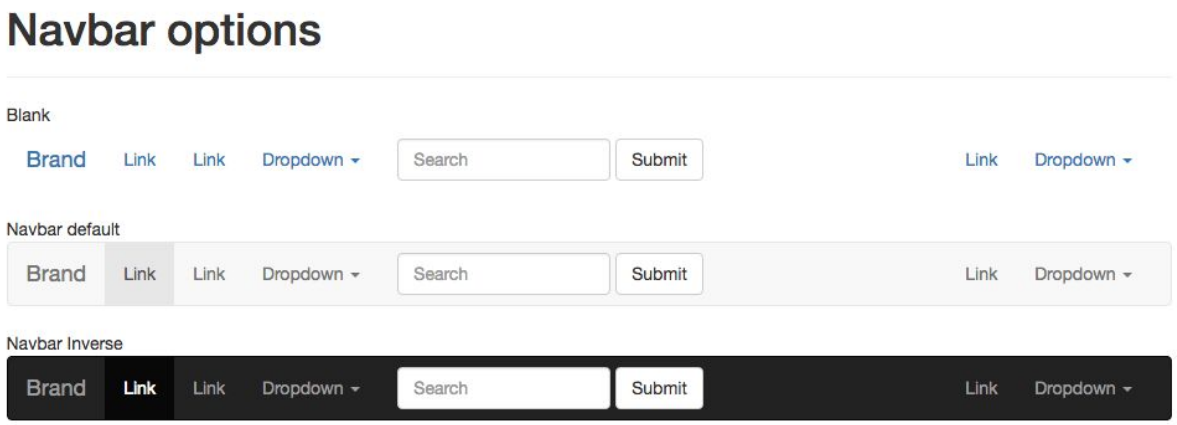

Figura 2.4: Ejemplo de barra de navegación de Bootstrap

Es práctica común diseñar las interfaces del sistema basándose en componentes de Bootstrap, lo cual permite acelerar el desarrollo debido a que se pueden tomar como base dichos componentes

[6] El diseño web adaptable (también diseño web adaptativo o responsivo), es una filosofía de diseño y desarrollo cuyo objetivo es adaptar la apariencia de las páginas web al dispositivo que se esté utilizando para visitarlas

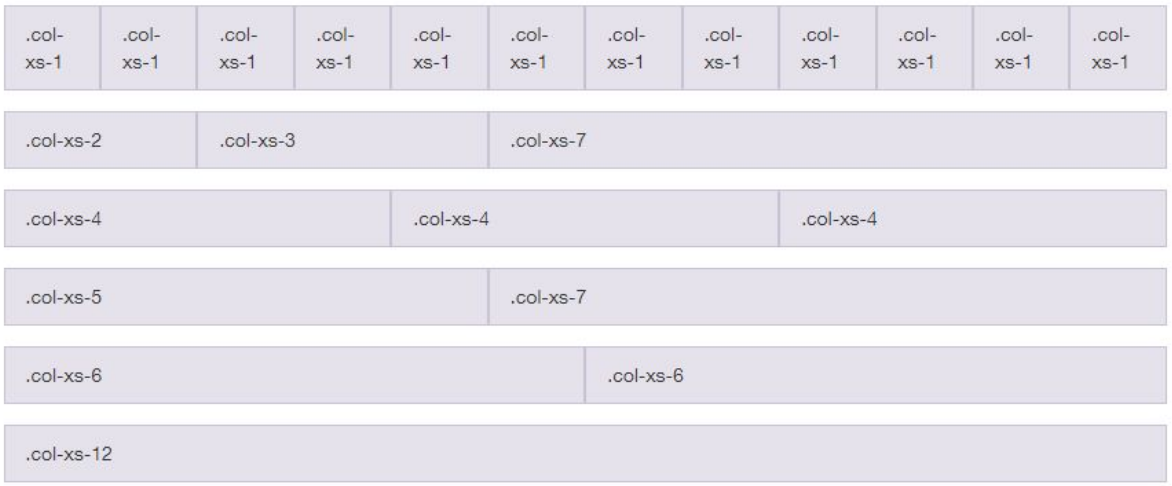

Figura 2.5: Sistema de grilla de Bootstrap

3. Diseño de la solución

Para poder realizar una solución informática que cumpla con las necesidades del cliente, primero se deben relevar y analizar los requerimientos. En esta sección vamos a enumerar los requerimientos del sistema desarrollado y se va a presentar cómo debería ser la interfaz del sistema desarrollado para cumplir con los requerimientos del cliente.
En esta sección nos vamos a centrar en los requisitos funcionales. Un requisito funcional define una función del sistema de software o sus componentes. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Los requisitos funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que se supone, un sistema debe cumplir. Los requisitos de comportamiento para cada requisito funcional se muestran en los casos de uso. Son complementados por requisitos no funcionales, que se enfocan en cambio en el diseño o la implementación. De los requerimientos no funcionales ya se expuso en el capítulo 2 cuando se habló de las características del servidor.
En [3] se describen las características que se deben tener en cuenta a la hora de obtener los requerimientos:

- Especificado por escrito: Como todo contrato o acuerdo entre dos partes.
- Posible de probar o verificar. Si un requerimiento no se puede comprobar, entonces ¿cómo se sabe si se cumplió con él o no?
- Conciso: Un requerimiento es conciso si es fácil de leer y entender. Su redacción debe ser simple y clara para aquellos que vayan a consultarlo en un futuro.
- Completo: Un requerimiento está completo si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.
- Consistente: Un requerimiento es consistente si no es contradictorio con otro requerimiento.
- No ambiguo: Un requerimiento no es ambiguo cuando tiene una sola interpretación. El lenguaje usado en su definición, no debe causar confusiones al lector.”

Como se puede observar en lo enumerado anteriormente, es evidente la importancia de obtener y analizar los requerimientos para poder enfocar los esfuerzos en proporcionar una solución que deje al cliente satisfecho

3.1. Requerimientos del sistema

Si bien existen diversas técnicas para realizar ingeniería de requerimientos, como las entrevistas abiertas, entrevistas cerradas, o los casos de uso, entre otros; éstos son tópicos que exceden la temática central de este trabajo. Por otro lado, se van a indicar ejemplos de casos de uso y diagramas de flujo generados por los analistas funcionales de DPIE, los cuales fueron utilizados para análisis posteriores.

De acuerdo con lo relevado por los analistas funcionales de DPIE se describirán los requerimientos que se deben cumplir para satisfacer las demandas del sistema a desarrollar.

En la Figura 3.1 se muestra el diagrama de casos de uso generado por los analistas. Como se puede observar en el mismo, se presentan tres actores, el Usuario consultor, el Liquidador, y el mismo Sistema. Se incluye a éste último como actor debido a que realiza tareas importantes durante la intervención de los otros dos actores al utilizar el mismo.

Por un lado tenemos los usuarios del *frontend*[7]. Para estos usuarios se debe realizar una interfaz más amigable y sencilla, debido a que, en general, no son usuarios tan avanzados en el uso de sistemas. Estos usuarios serían los consultores que facturan para los programas extrapresupuestarios de la Dirección General de Cultura y Educación de la Provincia de Buenos Aires. De acuerdo a los requerimientos, deben poder realizar las siguientes tareas:

- Ingresar al sistema.
- Ver información de los contratos vigentes y concluidos.
- Cargar mensualmente comprobantes para cada contrato.
- Ver seguimiento de las distintas etapas por las que transitan los comprobantes.
- Descargar comprobantes de retención de Ingresos Brutos y Ganancias de cada comprobante presentado.

[7] Front-end y back-end son términos que se refieren a la separación de intereses entre una capa de presentación y una capa de acceso a datos, respectivamente. Pueden traducirse al español el primero como interfaz, frontal final o frontal y el segundo como motor, dorsal final o zaga.

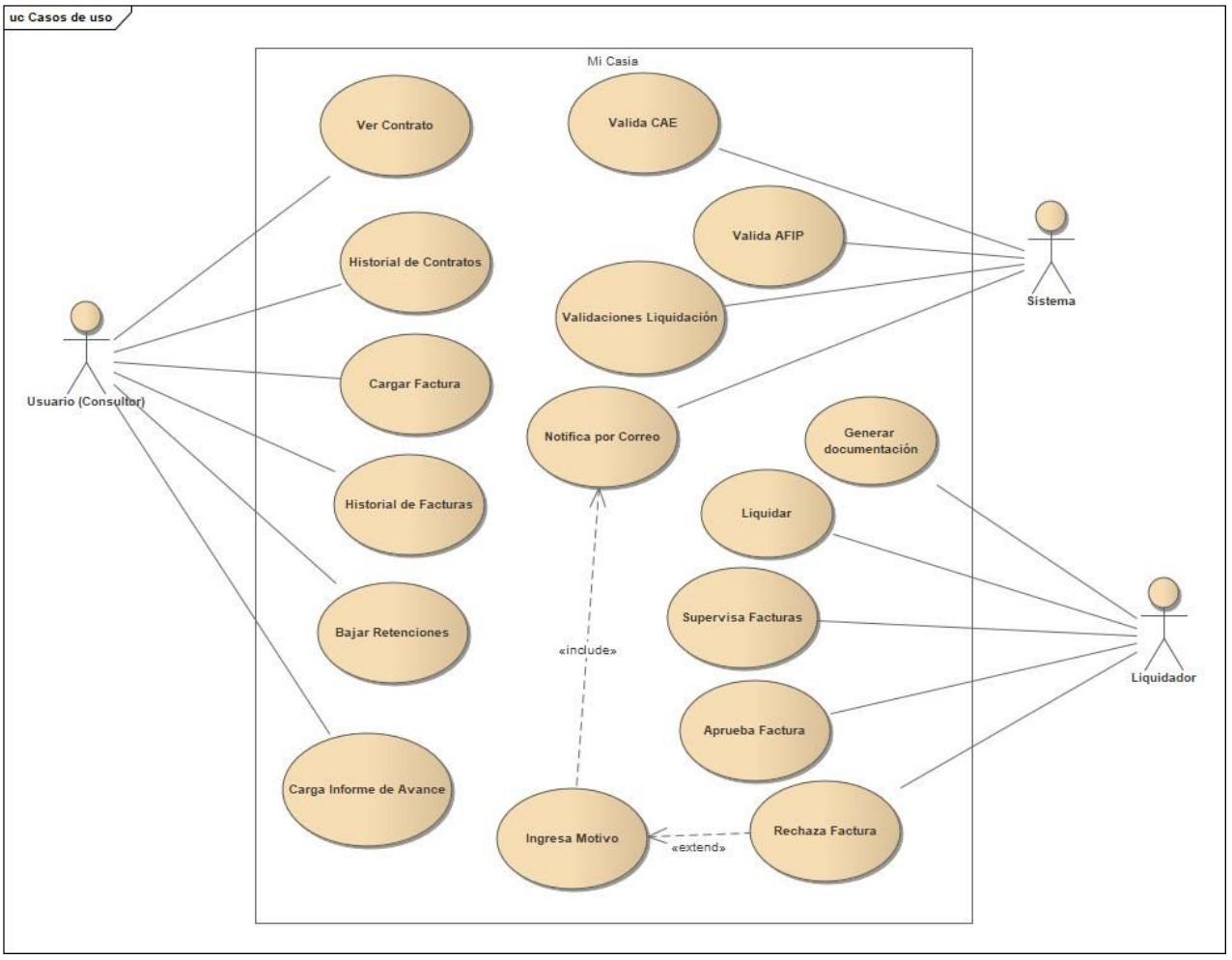

Figura 3.1: Casos de uso

Por otro lado, tenemos los usuarios del *backend*[7], los cuales abarca al personal del Área de Consultoría de la Dirección General de Cultura y Educación de la Provincia de Buenos Aires. Son usuarios más avanzados y la interfaz que se les provee pone mayor énfasis en la funcionalidad y operatividad. A estos usuarios se les debe permitir efectuar las funcionalidades descritas a continuación:

- Carga de consultores con sus datos de contacto y datos impositivos.
- Dar de alta un contrato a un consultor especificando los períodos y los montos de los informes de avance que debe presentar.
- Administrar los comprobantes cargados por los consultores permitiendo aprobar y rechazar.
- Preliquidar los comprobantes de varios consultores al mismo tiempo, y realizar los cálculos de las retenciones de Ingresos Brutos y Ganancias dependiendo de los datos impositivos previamente cargados en el sistema.

- Liquidar las preliquidaciones asignándoles fecha y número.
- Generar la documentación requerida para avanzar con el proceso de liquidación en las áreas consecutivas.
- Generar reportes gerenciales con estadísticas de los consultores que adeudan comprobantes, la correspondencia entre lo facturado y lo liquidado, entre otros.

Como último actor tenemos al Sistema. El mismo se debe comunicar varias veces con AFIP a través de *web services*. Realiza las siguientes tareas automáticas, las cuales son originadas por acciones de los otros actores:

- Cuando el actor Usuario liquidador da de alta un contrato, el sistema debe enviar un email notificando el mismo. Si no tiene usuario, se debe crear uno y comunicar las credenciales de acceso en el mismo email.
- Cuando el actor Usuario liquidador rechaza un comprobante, el sistema debe enviar un email notificando la situación, acompañado de aclaraciones del liquidador para corregir el comprobante.
- Cuando el Usuario consultor carga un comprobante, el sistema debe descargar la constancia de CUIT a la fecha, obtener los datos del mismo y validar el CAE (Código de Autorización Electrónico) con AFIP.

Consecutivamente en la Figura 3.2 se muestra un diagrama de flujo extendido. En el mismo se puede apreciar la mecánica a seguir al momento que el usuario carga el comprobante en el sistema. Los hitos más importantes de este proceso son cuatro:

1. Primero se deben obtener los datos del comprobante.
2. Luego se debe validar el CAE mediante webservices de AFIP enviando los datos obtenidos del comprobante.
3. Posteriormente se utiliza otro webservices de AFIP para validar el CUIT y la fecha de inicio de actividades del consultor.
4. Por último, se realizan validaciones adicionales:
    a. La fecha de emisión del comprobante debe ser igual o posterior al último día del mes facturado.
    b. La fecha desde debe ser el primer día del mes facturado.
    c. La fecha hasta debe ser el último día del mes facturado.
    d. La fecha de vencimiento para pago debe ser al menos 60 días posteriores a la fecha de la factura.

e. El monto del comprobante debe coincidir con el informe actual.
f. La fecha de inicio de actividades debe coincidir con la indicada en AFIP

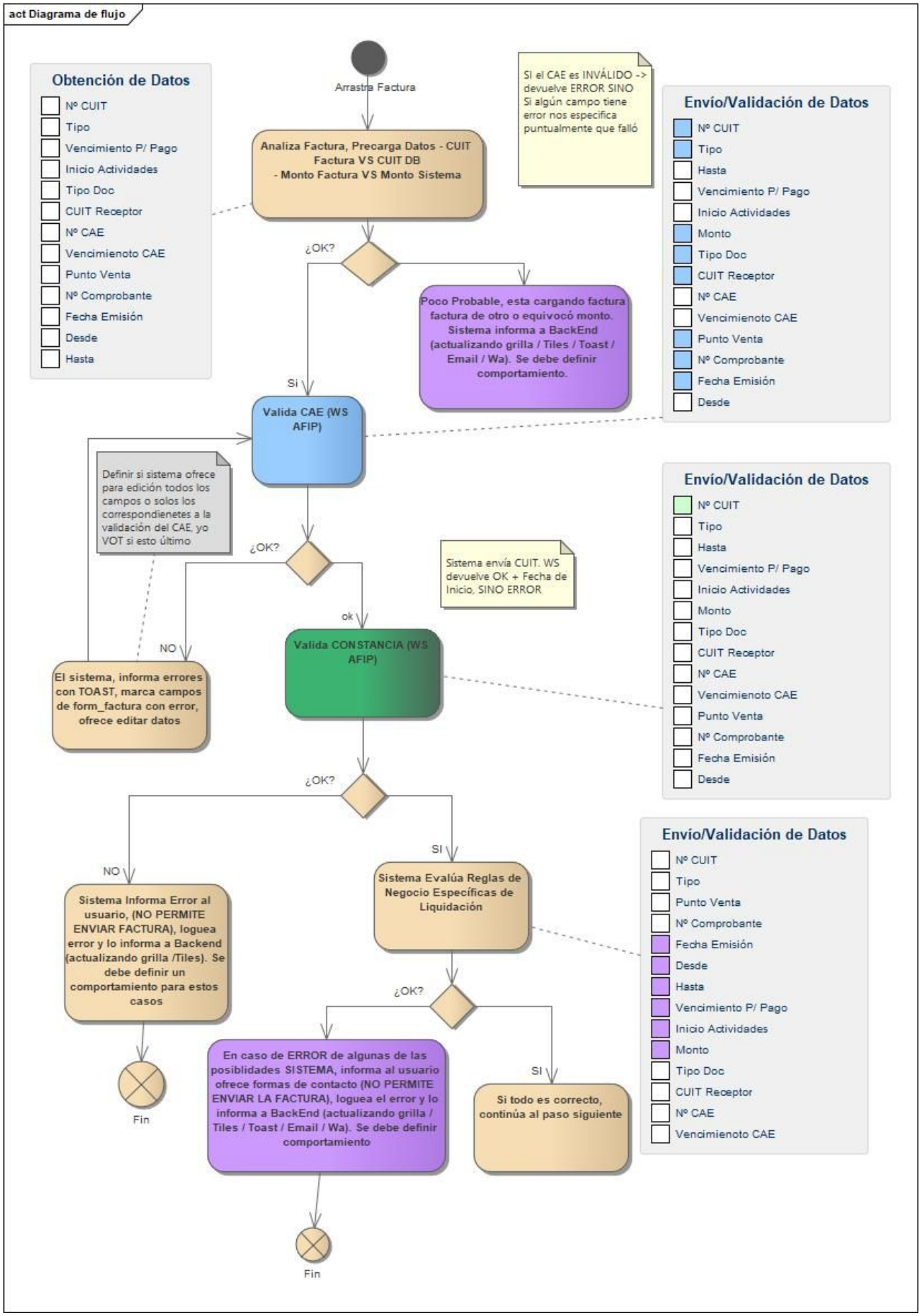

Figura 3.2: Diagrama de flujo extendido

Otros puntos importantes que debe soportar el sistema son la preliquidación y la liquidación. Básicamente la preliquidación consiste en agrupar los comprobantes seleccionados en un conjunto denominado autorización contable. Luego ésta es liquidada, es decir se aprueba esta autorización contable asignándole una determinada fecha y número. Esto convierte a la preliquidación en liquidación y a una autorización contable en orden de pago. En adelante utilizaremos AC para referirnos a una autorización contable, y OP para abreviar orden de pago. Existen dos formas de preliquidar. Una es generando una AC por cada comprobante seleccionado; y, la otra forma, es agrupar los comprobantes por CUIT y generar una AC por cada CUIT. Esta última se utiliza cuando se quieren preliquidar varios comprobantes de una persona en una misma oportunidad.

Como se mencionó previamente, al momento de preliquidar se realiza el cálculo de las retenciones, tanto de ingresos brutos como de ganancias, respetando las fórmulas e importes indicados en la Resolución General 4525 de AFIP que modifica la Resolución General 830 de AFIP con vigencia a partir del 01/08/2019.

3.2. Mockups

Un *mockup* o maqueta es un diseño digital de una web y / o aplicación. Las maquetas se utilizan en la fase de diseño inicial para visualizar ideas y conceptos en el contexto del diseño web e incluyen la estructura de navegación, el sitio y los elementos de diseño en detalle. Los mockups pueden ser plantillas producidas con programas de edición de imágenes sin funcionalidad o diseños que se crean con herramientas especiales de maquetas y donde los elementos de control ya están vinculados con funciones simples.

Los términos mockup, *wireframe* y prototipo a menudo se consideran lo mismo en la práctica, sin embargo, en realidad son tres tipos diferentes de representaciones de diseño como parte de la creación rápida de prototipos y se utilizan en diferentes etapas del diseño antes de la programación real.

Un wireframe es la forma más simple de plantilla y no incluye ningún color, tipografía, imágenes o gráficos, a diferencia de una maqueta. Este primer diseño de boceto puede ser creado a mano o en la computadora. La funcionalidad está completamente ausente en esta pantalla. Por lo tanto, los wireframes se consideran diseños de baja fidelidad.

Las maquetas pueden construirse sobre cualquier estructura existente y seguir desarrollándola. Integrando color, tipografía, imágenes y gráficos, se acercan mucho al diseño final y ya lo representan. Por lo tanto, son preferibles a efectos de presentación.

Un *mockup* clásico es estático. Sin embargo, las modernas herramientas de maquetación permiten la integración de funciones simples como los enlaces, de modo que, dependiendo de la complejidad, pueden clasificarse como diseños de mediana a alta fidelidad y bordear los prototipos.

Se pueden utilizar maquetas complejas de varias partes como base para prototipos que, dependiendo de su estilo, también se pueden considerar diseños de mediana a alta fidelidad. A diferencia de la clásica maqueta estática, los prototipos son siempre interactivos y contienen la mayoría de las funciones de la página web o aplicación planificada.

Las maquetas se utilizan en la fase inicial de desarrollo de webs y aplicaciones para la presentación y el control de calidad. Sirven para coordinar con el cliente las ideas y los requisitos de la interfaz de usuario con respecto a las funciones básicas, la navegación, la arquitectura de contenidos y el diseño.

En la Figura 3.3 se puede apreciar la diferencia entre wireframe, mockup y prototipo.

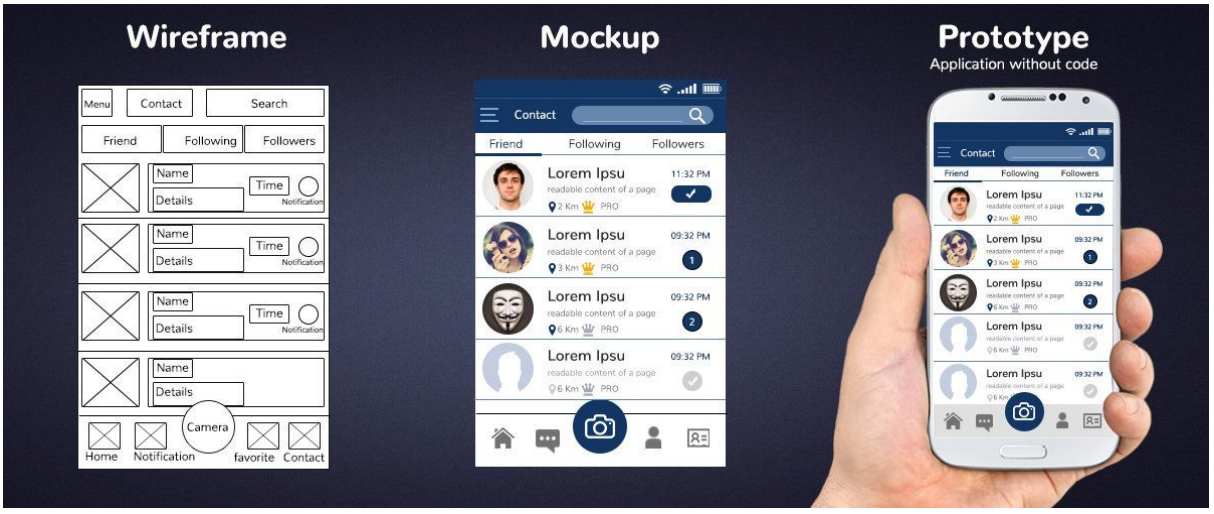

Figura 3.3: Wireframe/Mockup/Prototipo

Las maquetas se utilizan para pruebas de usabilidad sin un gran esfuerzo de programación previo. De este modo, cualquier problema se detecta antes de crear el prototipo y se reduce el riesgo de que un concepto tenga que ser completamente revisado a mitad de la fase de desarrollo. En general, los mockups pueden contribuir a ahorrar tiempo y dinero en un proyecto online.

En nuestro equipo de desarrollo, los analistas suelen hacer bosquejos wireframe con los clientes en las primeras entrevistas de obtención de requerimientos y luego analizar suelen diseñar mockups y prototipos con funcionalidades de navegación o ventanas emergentes

para obtener validaciones con los clientes sobre la interfaz con herramientas como Balsamiq o Axure.

A continuación, se incluyen algunos mockups y e imágenes de prototipos diseñados por los analistas de cómo debería ser la interfaz del sistema para los Usuarios consultor.

Figura 3.4: Mockup contratos Frontend

En la Figura 3.4 se muestra como debería ser la interfaz que ve un Usuario consultor cuando entra al sistema. Como se puede observar, debe aparecer un listado con los contratos del mismo, indicando ciertos datos como el programa, el período, el importe total, la cantidad de informes, y la cantidad de informes pagados.

Cuando se hace click en el detalle del contrato (botón “Ver detalle” en la figura 3.4) se debe ir a otra pantalla. En la Figura 3.5 se puede ver el detalle del mismo. Se muestra para cada informe de avance el mes, el tipo de informe, si tiene o no comprobante, el estado del mismo (si fue aprobado, rechazado, pagado, etc.) y la opción de cargar la factura o de imprimir las retenciones.

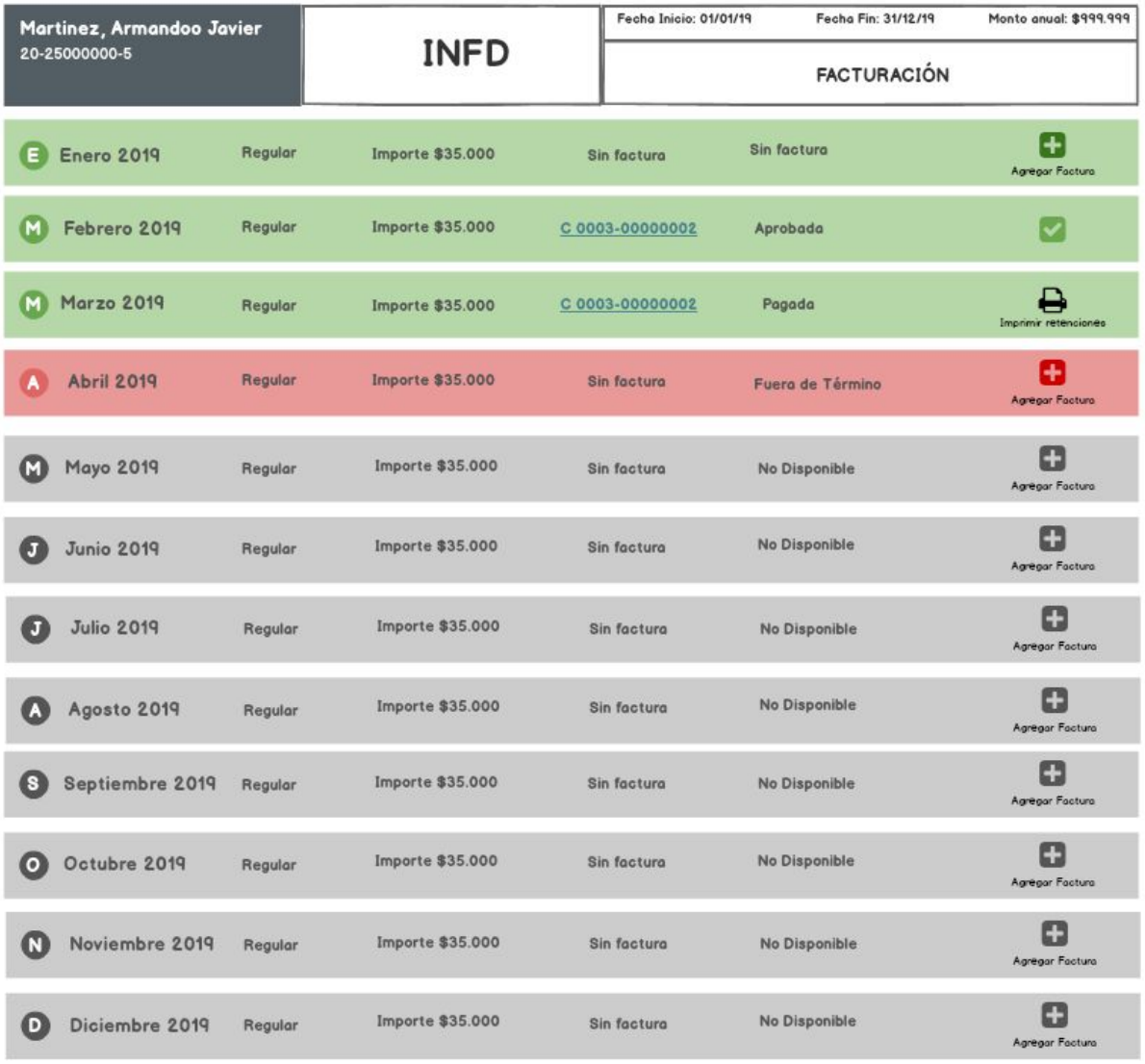

Figura 3.5: Mockup detalle contratos Frontend

Cuando el Usuario consultor hace click en “Agregar factura”, se despliega un *wizard*[8] para cargar la misma. Se muestran tres pasos. En el primer paso se solicita que se adjunte la factura, como se puede ver en la Figura 3.6. Una vez adjuntada se envía al servidor, se parsea y se obtienen los datos de la misma. También se debe validar en AFIP su condición frente a los impuestos y el CAE, y realizar las validaciones de DGCyE mencionadas en la sección 3.1. En la Figura 3.7 se puede observar el detalle del paso 1 luego de hacer las validaciones. También se puede ver que se informa de los campos que tuvieron error en la validación por medio de una burbuja o tooltip.

[8] Es un tipo de interfaz de usuario que presenta al usuario una secuencia de cuadros de diálogo que lo guían a través de una serie de pasos bien definidos. Las tareas que son complejas, poco frecuentes o poco familiares pueden ser más fáciles de realizar con un asistente.

Figura 3.6: Mockup Wizard carga de factura. Primer paso

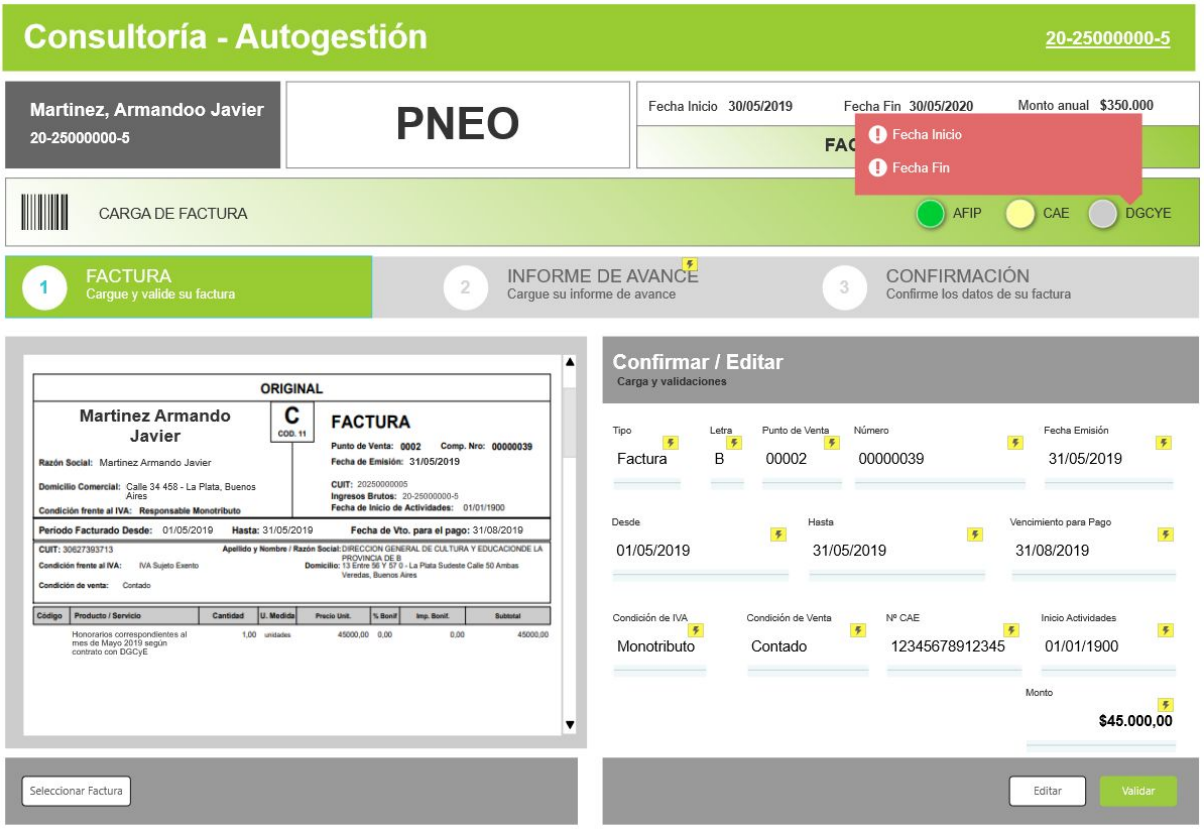

Figura 3.7: Mockup Wizard carga de factura. Primer paso con factura

Figura 3.8: Mockup Wizard carga de factura. Primer paso

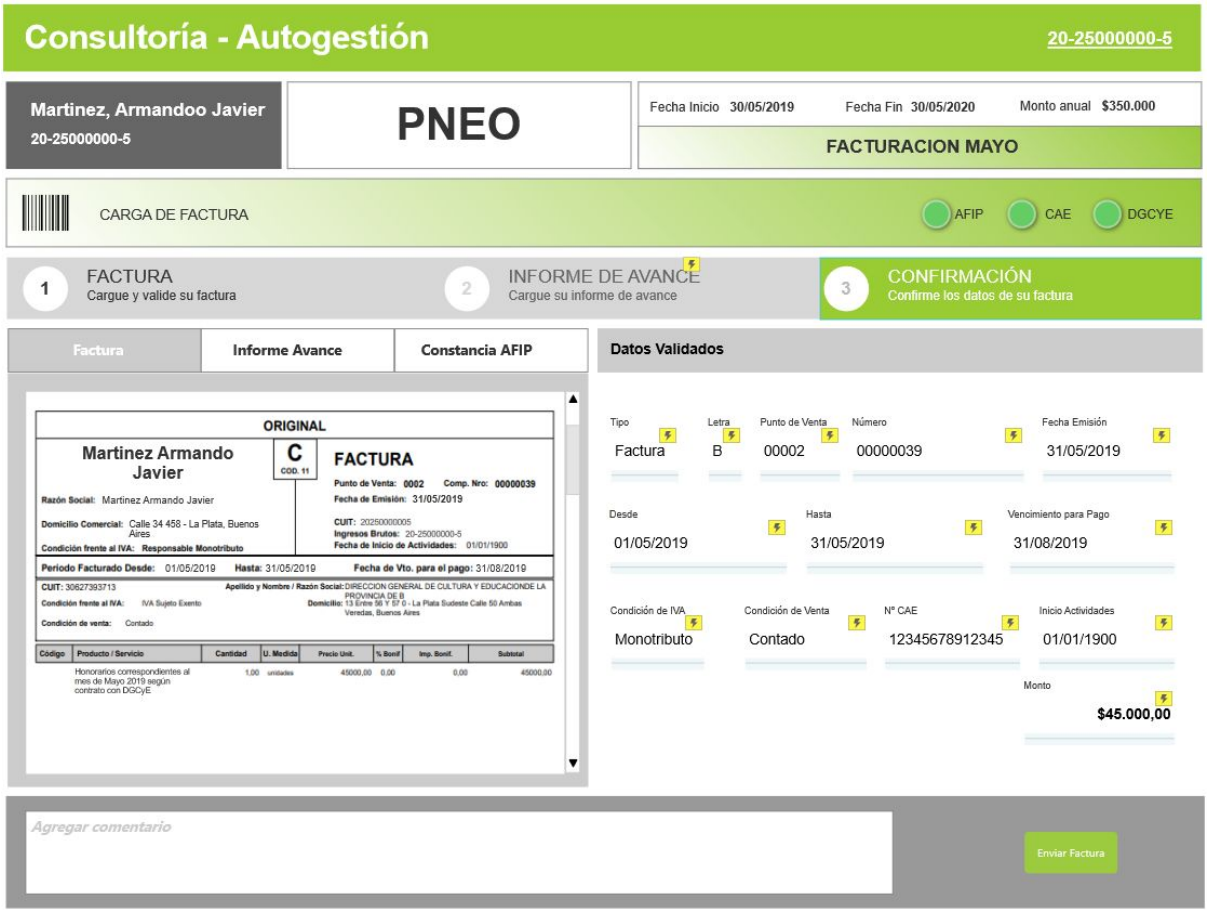

Figura 3.9: Mockup Wizard carga de factura. Primer paso

Luego de que el Usuario consultor completó satisfactoriamente el paso 1, pasa al paso 2 en donde debe cargar un detalle de las tareas realizadas en el mes facturado. En la Figura 3.8 se muestra un mockup con el paso 2. Por último, cuando completa el paso 2, llega al paso final. En este paso se muestra un resumen de los datos de la factura parseada, la constancia de CUIT obtenida de AFIP y el detalle del informe de avance con las tareas realizadas. Además, se permite agregar alguna observación adicional, como se puede ver en la Figura 3.9

Con respecto a la interfaz de backend, es decir la interfaz de los liquidadores, se obviaron los mockups debido a que son muy similares al resto del sistema backend. En éste predominan tablas con listados de ítems y formularios tabulares debido a que está más orientado a la productividad de los usuarios. Además, se trata de usuarios más avanzados en el uso de sistemas, por lo que se enfocan los esfuerzos en nuevas funcionalidades por sobre la estética y en muchas ocasiones no se confeccionan mockups, debido a que la interfaz es similar. Igualmente se incluye un mockup de cómo debería ser la interfaz de aprobación y rechazo de comprobantes que utilizan los Usuarios liquidador. Como se puede ver en la Figura 3.9, los liquidadores pueden ver cada comprobante, con su constancia y el detalle del informe. Para ello se proporcionan pestañas que se muestran a la izquierda con cada ítem. Los mismos muestran el contenido correspondiente al ir haciendo click en cada uno de los comprobantes del listado.

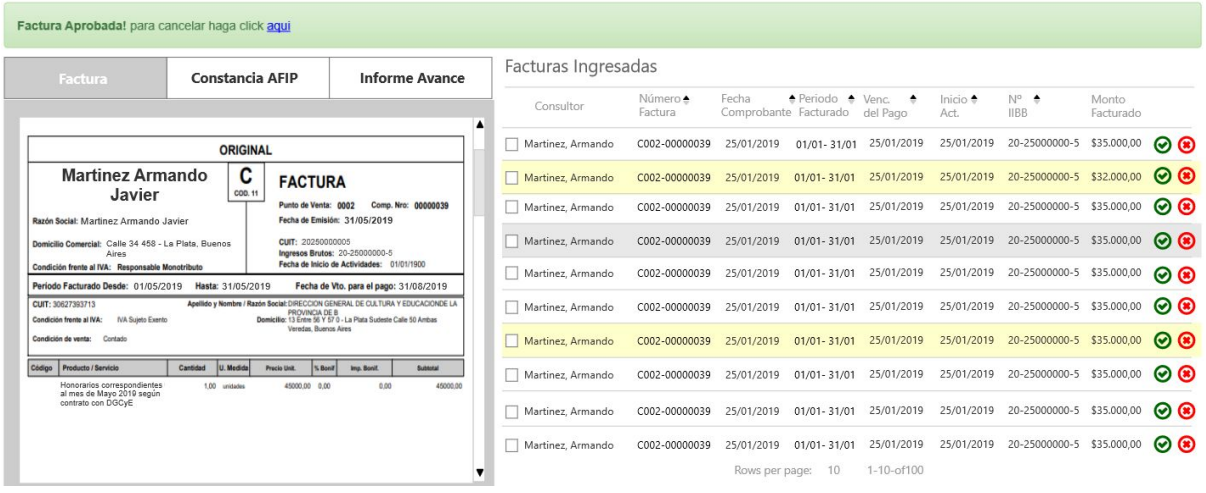

Figura 3.9: Mockup aprobación/rechazo de factura Backend

4. Metodologías de trabajo utilizadas

Cuando se piensa en cómo construir un software, el modelo en cascada describe un método que es lineal, cerrado y secuencial. Se toma el proyecto de desarrollo y se divide en etapas o fases de desarrollo. Cada una de estas fases tiene un objetivo. Se avanza a la siguiente fase una vez que se alcanzó el objetivo. En la Figura 4.1 se puede observar cada una de las etapas secuenciales del modelo de cascada:

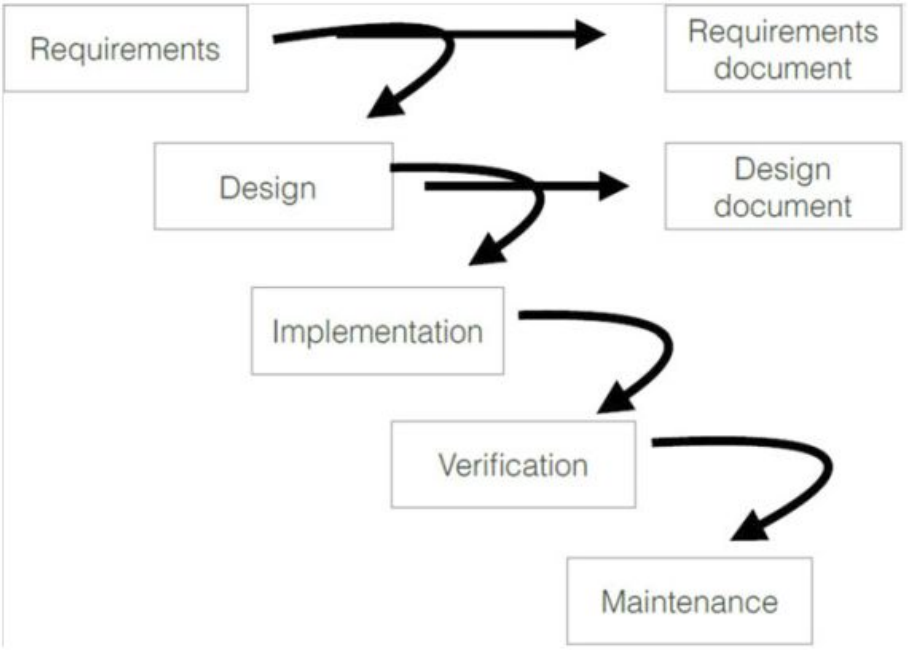

Figura 4.1: Modelo cascada

Requerimientos: Un equipo comenzará con la fase de requisitos donde los analistas de negocios o marketing de productos definen los requisitos. El objetivo de esta etapa es generar un documento de requerimientos.

Diseño: El trabajo de diseño consiste en cosas como decidir cómo el equipo escribirá el código. Ellos averiguarán los lenguajes y las técnicas a utilizar, para ofrecer la mejor funcionalidad y valor, e incluso quién trabajaría en qué.

Implementación: es donde los desarrolladores escriben el código y transforman las ideas en un producto funcional. Los desarrolladores toman los requisitos tal como se definieron en

la fase de Requerimientos y crean valor para los clientes. Al final de la fase de Implementación, se produce software, y ese software se asemeja a un producto viable.

Verificación: El software necesita estar en buen estado de funcionamiento. En la fase de Verificación, los ingenieros de control de calidad prueban el software. Trabajan para asegúrese de que el equipo entregue un producto de alta calidad a los clientes. Los ingenieros de control de calidad toman el documento de diseño producido durante la fase de diseño y crean planes de testing a partir de él. Ejecutan los planes de prueba contra el software producido en la fase de Implementación. El personal de desarrollo aborda cualquier defecto encontrado. Los ingenieros de garantía de calidad trabajan como un equipo separado. Ellos no se consideran parte del desarrollo.

Mantenimiento: una vez que todos acordaron que el producto está listo, se lanza o se considera Generalmente disponible (del inglés: *generally available*). Una vez que el producto está lanzado, se inicia la fase de Mantenimiento. Ingenieros de mantenimiento o los desarrolladores abordan los defectos descubiertos por los clientes y los empresarios deciden si se necesita otra versión del producto.

Como sostiene McKenna en [4], este enfoque tiene sentido. A primera vista, el modelo en cascada parece una buena forma de producir software de calidad. Sin embargo, hay algunos defectos importantes con el Modelo en cascada que son muy problemáticos en la economía basada en aplicaciones del siglo XXI. Si los requisitos cambian en un proyecto de tipo *Waterfall* después de la fase de requerimientos, todo tiene que parar y los nuevos requerimientos deben ser evaluados. Esto provoca un efecto dominó donde todas las otras fases se "expulsan". Esto agrega costos al proyecto y hace que la fecha de GA sea mucho más difícil de alcanzar. Actualmente vivimos y trabajamos en un mundo donde los requisitos cambian rápidamente, por lo que hay muchas oportunidades para que los requisitos cambien bastante durante un lanzamiento de doce o dieciocho meses.

4.1. Metodologías ágiles

¿Por qué el modelo en cascada tarda tanto? Doce a dieciocho meses puede ser considerado mucho tiempo. McKenna opina que los clientes estaban acostumbrados a esa cadencia.

Eugenia Bahit en [5] comenta sobre este tema que, en el año 2009, el Standish Group (www.standishgroup.com) elaboró un informe llamado Chaos Report, el cual arrojó como resultado que solo el 32% de los proyectos de desarrollo de Software han sido exitosos. El mismo informe, indica que más del 45% de las funcionalidades del Software entregadas al usuario, jamás se utilizan.

El desarrollo ágil de software (define la misma autora), no es más que una metodología de gestión de proyectos adaptativa, que permite llevar a cabo, proyectos de desarrollo de software, adaptándose a los cambios y evolucionando en forma conjunta con el software.

Las metodologías tradicionales (o predictivas), encaran las fases que componen el ciclo de vida del desarrollo de Software de manera sucesiva. Es decir, que una fase sucede a otra, y cada fase, ocupa un espacio lineal en él.

En cambio, las metodologías ágiles, solapan estas etapas, permitiendo ahorrar tiempo, evitando la dependencia (cada etapa es independiente de la otra) y haciendo del ciclo de vida, un proceso iterativo (se inicia con el relevamiento, se finaliza con la implementación y se vuelve a comenzar para abordar nuevas funcionalidades).

4.2. Manifiesto del agilísmo

El 12 de febrero de 2001 diecisiete críticos [31] de los modelos de mejora del desarrollo de software basados en procesos, convocados por Kent Beck, quien había publicado un par de años antes Extreme Programming Explained, libro en el que exponía una nueva metodología denominada Extreme Programming, se reunieron en Snowbird, Utah para tratar sobre técnicas y procesos para desarrollar software. En la reunión se acuñó el término “Métodos Ágiles” para definir a los métodos que estaban surgiendo como alternativa a las metodologías formales (CMMI, SPICE) a las que consideraban excesivamente “pesadas” y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas previas al desarrollo.

Los integrantes de la reunión resumieron los principios sobre los que se basan los métodos alternativos en cuatro postulados, lo que ha quedado denominado como Manifiesto Ágil.

En el Manifiesto Ágil, podemos leer los cuatro valores que impulsa el agilísmo:

Individuos e interacciones sobre procesos y herramientas

Mientras que las metodologías tradicionales, centran su esfuerzo en definir procesos y herramientas que regulen la gestión de un proyecto, las metodologías ágiles, prefieren valorar la idoneidad de cada individuo, depositando en ésta, la confianza necesaria para lograr una buena comunicación, fluida e interactiva entre todos los participantes del proyecto.

Software funcionando sobre documentación extensiva

Mientras que las metodologías predictivas (o tradicionales), invirtien una gran cantidad de tiempo y dedicación al desarrollo de documentos que describan detalladamente el alcance de un sistema así como el diseño de prototipos o bocetos que intenten proveer una imagen visual del futuro Software, las metodologías ágiles proponen destinar ese tiempo al desarrollo de la aplicación, centrando el mismo, en pequeñas partes 100% operativas y funcionales, que por un lado, otorguen mayor valor al negocio y por otro, permitan probar Software real (no prototipos ni bocetos).

Colaboración con el cliente sobre negociación contractual

Las metodologías predictivas, plantean un sistema de gestión de riesgos, mediante el cual, cada cambio sugerido por el cliente, debe superar una serie de procesos administrativos, tales como la ampliación/modificación del alcance, la reelaboración de presupuestos y una nueva estipulación de tiempo de entrega lo que conlleva a la inevitable modificación de un cronograma de entregables. A diferencia de esto, las metodologías ágiles, prefieren integrar al cliente al proyecto, facilitando la comunicación directa de éste con el equipo de desarrollo, a fin de generar un ambiente de colaboración mutua, donde los “cambios” sugeridos por el cliente, no signifiquen un riesgo, sino un “aporte” al valor del Software.

Respuesta ante el cambio sobre seguir un plan

En la gestión tradicional, antes de comenzar el desarrollo del Software (llamado “proceso de ejecución”), se elabora un detallado plan donde se describe un alcance funcional minucioso y preciso, así como también, se pautan fechas de entregas que se pretende, no sufran variaciones.

Cuando en pleno proceso de ejecución, el cliente solicita algún cambio, al existir un plan previsto que no puede desviar su rumbo, surgen una serie de conflictos derivados del

cambio, los cuales conllevan a reelaborar el plan inicial y, en consecuencia, extender o prorrogar los plazos de entrega.
Por el contrario, las metodologías ágiles proponen adaptarse a los cambios sugeridos por el cliente, lo que significa, que, a la modificación propuesta, no se sumarán escollos administrativos, sino por el contrario, se acelerarán los procesos, actuando en consecuencia de lo que ha sido pedido.
Vale aclarar, que si bien las metodologías ágiles, valoran los elementos de la derecha (derivados de las propuestas tradicionalistas), otorgan un mayor valor a los de la izquierda. Y ésto, tiene su fundamento en los doce principios, que también se exponen en el Manifiesto Ágil.

Los doce principios del agilísimo

En el Manifiesto Ágil, podemos leer los doce principios en los cuáles se argumentan los valores del Manifiesto. Estos son:

- Principio #1: Satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Principio #2: Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Principio #3: Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Principio #4: Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- Principio #5: Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- Principio #6: Conversación cara a cara.
- Principio #7: El software funcionando es la medida principal de progreso.
- Principio #8: Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- Principio #9: La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- Principio #10: La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

- Principio #11: Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- Principio #12: A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

A continuación, vamos a hacer un breve repaso por dos metodologías ágiles muy utilizadas: SCRUM y XP.

4.3. SCRUM

Scrum es un marco de trabajo que nos permite encontrar prácticas emergentes en dominios complejos, como la gestión de proyectos de innovación. No es un proceso completo, y mucho menos, una metodología. En lugar de proporcionar una descripción completa y detallada de cómo deben realizarse las tareas de un proyecto, genera un contexto relacional e iterativo, de inspección y adaptación constante para que los involucrados vayan creando su propio proceso. Esto ocurre debido a que no existen ni mejores ni buenas prácticas en un contexto complejo. Es el equipo de involucrados quien encontrará la mejor manera de resolver sus problemáticas. Este tipo de soluciones serán emergentes.

El equipo de desarrollo se encuentra apoyado en dos roles: el *Scrum Master* y el *Product Owner*. El Scrum Master es quien vela por la utilización de Scrum, la remoción de impedimentos y asiste al equipo a que logre su mayor nivel de performance posible. Puede ser considerado un *coach* o facilitador encargado de acompañar al equipo de desarrollo. El Product Owner es quien representa al negocio, *stakeholders*, cliente y usuarios finales. Tiene la responsabilidad de conducir al equipo de desarrollo hacia el producto adecuado.

El progreso de los proyectos que utilizan Scrum se realiza y verifica en una serie de iteraciones llamadas *Sprints*. Estos Sprints tienen una duración fija, preestablecida de no más de un mes. Al comienzo de cada Sprint el equipo de desarrollo realiza un compromiso de entrega de una serie de funcionalidades o características del producto en cuestión.

Al finalizar el Sprint se espera que estas características comprometidas estén terminadas, lo que implica su análisis, diseño, desarrollo, prueba e integración al producto. En este momento es cuando se realiza una reunión de revisión del producto construido durante el Sprint, donde el equipo de desarrollo muestra lo construido al Product Owner y a cualquier stakeholder interesado en participar. El feedback obtenido en esta reunión puede ser incluido entre las funcionalidades a construir en futuros Sprints.

4.4. XP

eXtreme Programming (programación extrema) también llamado XP, es posiblemente el proceso ágil más popular y de alto perfil, el cual fue desarrollado por Kent Beck, Ward Cunningham y Ron Jeffries [30]. XP evolucionó como resultado de desarrolladores y testers que buscaban nuevas formas de entregar software confiable de alta calidad, evitando procedimientos excesivamente formales.

A diferencia de Scrum, XP propone solo un conjunto de prácticas técnicas, que, aplicadas de manera simultánea, pretenden enfatizar los efectos positivos de en un proyecto de desarrollo de Software.

eXtreme Programming se apoya en cinco valores, los cuales enfatizan la esencia colaborativa del equipo. Estos valores son:

Comunicación

En XP, todo es trabajado en equipo: desde el relevamiento y análisis hasta el código fuente desarrollado. Todo se conversa cara a cara, procurando hallar soluciones en conjunto a los problemas que puedan surgir.

Simplicidad

Se pretende desarrollar solo lo necesario y no perder tiempo en detalles que no sean requeridos en el momento. En este aspecto, se asemeja a otra metodología ágil, denominada Kanban, en la cual, un proceso “anterior” sólo produce lo que el proceso posterior demanda.

Retroalimentación

El objetivo de eXtreme Programming es entregar lo necesario al cliente, en el menor tiempo posible. A cambio, demanda al cliente, un feedback continuo -retroalimentación-, a fin de conocer sus requerimientos e implementar los cambios tan pronto como sea posible.

Respeto

El equipo respeta la idoneidad del cliente como tal (sólo éste, es quien conoce el valor para el negocio) y el cliente, a la vez, respeta la idoneidad del equipo (confiando en ellos profesionalmente para definir y decidir el “cómo” se llevará a cabo el desarrollo de lo requerido).

Coraje

Se dice que en XP un equipo debe tener el valor para decir la verdad sobre el avance del proyecto y las estimaciones del mismo, planificando el éxito en vez de buscar excusas sobre los errores.

4.5. Encontrando la metodología adecuada

El sistema desarrollado se encuentra en constante crecimiento debido a que se van recibiendo nuevos requerimientos funcionales a diario. Nuestro equipo de desarrollo encontró en las metodologías ágiles una herramienta eficaz para manejar las funcionalidades solicitadas con el menor impacto y resolverlas lo más pronto posible.

No podemos utilizar las metodologías tradicionales porque no se cuenta con plazos de varios meses o incluso años. Hay situaciones en las que hay que resolver problemas en pocas semanas o a veces en pocos días. Por consiguiente, utilizamos las técnicas SCRUM y XP para abordar las nuevas funcionalidades, e incluso a veces combinación de ambas dependiendo de la necesidad y la urgencia de implementar las mismas.

Al utilizar ambas técnicas se presentan ventajas y desventajas asociadas, las cuales se detallarán posteriormente en el presente trabajo.

5. Desarrollo del sistema

En este capítulo se realizará un recorrido a lo largo del proceso de diseño y desarrollo del sistema.

Durante el análisis orientado a objetos, se hace hincapié en encontrar y describir los objetos (o conceptos) en el dominio del problema. Por ejemplo, en nuestro dominio, algunos de los conceptos incluyen el Contrato y el Comprobante.

Durante el diseño orientado a objetos, se enfatiza en definir los objetos de software y cómo colaboran para cumplir con los requisitos. Por ejemplo, en nuestro sistema, un objeto de software de Contrato puede tener un atributo de fecha de inicio y un método getFechaInicio.

Finalmente, durante la implementación o la programación orientada a objetos, se implementan objetos de diseño, como una clase Contrato o Comprobante en el lenguaje php que es el que se utilizó en el desarrollo del sistema.

Un diagrama de clase de diseño (DCD) ilustra las especificaciones para las clases e interfaces de software en una aplicación. La información típica incluye:

- Clases, asociaciones y atributos
- Interfaces, con sus operaciones y constantes
- Métodos
- Información del tipo de atributo
- Navegabilidad
- Dependencias

5.1. Diseño arquitectural

Subsiguientemente se presentará el diagrama de clases modelado para el sistema. El mismo fue realizado en el lenguaje UML. El lenguaje de modelado unificado (UML) es un lenguaje para especificar, visualizar, construir y documentar los artefactos de los sistemas de software, así como para el modelado de negocios y otros sistemas que no son de software.

Se dividirá el diagrama de clases en secciones para facilitar su interpretación, y se detallarán los criterios de diseño llevado a cabo.

En una primera instancia existen consultores que tienen contratos. El sistema ya cuenta con otros tipos de beneficiarios, como los proveedores de obras. Existen datos que son comunes a los beneficiarios consultores y a los beneficiarios proveedores como por ejemplo

el CUIT. Para evitar duplicación de los datos, se definió una entidad **Beneficiario** que es una entidad abstracta y es padre de **Consultor**. Los datos comunes de un beneficiario se modelaron en una entidad denominada **DatosBeneficiario**. Ésta última entidad, tiene los datos comunes a todos los beneficiarios. De esta manera se puede tener una persona tanto como consultor o como proveedor sin tener repetidos en la base datos como el CUIT, la dirección o incluso las cuentas bancarias a donde se realizan los pagos. En la Figura 5.1 se puede ver el diagrama de clases conteniendo las entidades antes mencionadas.

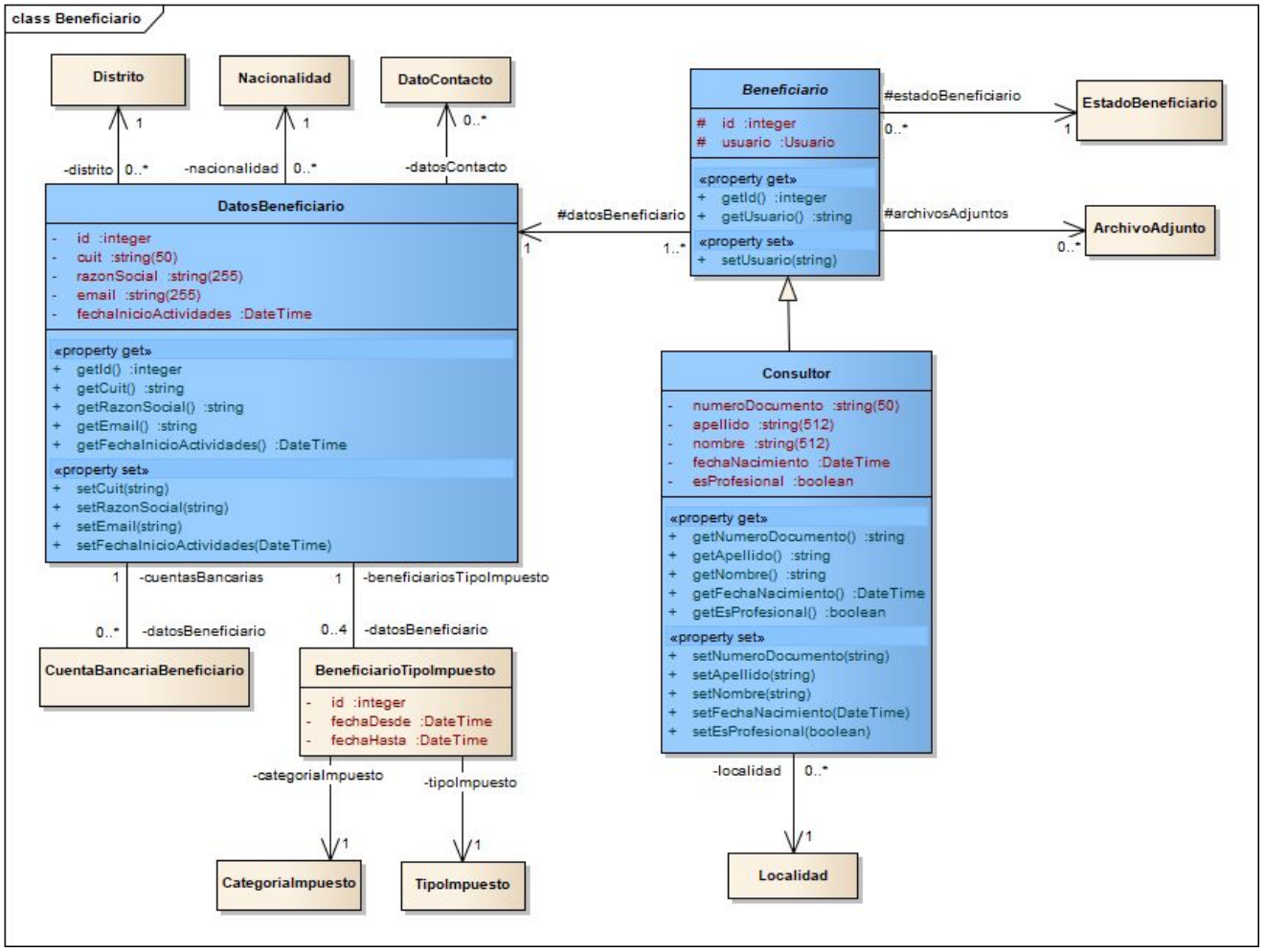

Figura 5.1: Diagrama de clases de Consultores

Cabe destacar la relación de **Beneficiario** con **BeneficiarioTipoImpuesto**. Esta entidad contiene la categoría (si está Inscripto o Exento, por ejemplo) para cada impuesto (IVA, IIBB, SUSS y Ganancias) y el rango de fechas en las que tiene esa categoría.

Una vez definidas las entidades para soportar la carga de los consultores, el paso siguiente es analizar los contratos. Los contratos extrapresupuestarios tienen una fecha de inicio y una de fin y se define una cierta cantidad de informes de avance con un monto estipulado de antemano. A cada informe de avance se le asocia un comprobante (puede ser una factura o un recibo). Del mismo modo que pasó con los beneficiarios, los contratos se modelaron de tal manera de soportar varios módulos como por ejemplo el de obras. Por eso tenemos una entidad padre abstracta **Contrato**, en la cual tenemos la información común de los contratos como el importe total y la fecha de inicio y fin; y, una entidad hija **ContratoConsultoria** donde tenemos datos específicos de los contratos de consultoría como el conjunto de informes de avance. También se modelaron los informes de avance bajo la entidad **InformeAvanceConsultoria** a causa de que cada informe de avance puede tener un importe diferente y puede ser de un tipo distinto (regular, aguinaldo, actualización, adicional) y corresponde a cierto mes y año (puede haber incluso varios informes para un mes dado). Por último, tenemos al comprobante, que siguiendo el mismo razonamiento se modeló una entidad padre **Comprobante** y una hija **ComprobanteConsultoria**. Para el caso particular de esta entidad, los comprobantes deben ser enviados a distintas áreas durante el proceso de carga, liquidación, pago, entre otros, así como también son adjuntados a un expediente. Es por eso que en la Figura 5.2 se puede observar que Comprobante extiende primero de **DocumentoAdjuntable** lo que le da las características para ser adjuntado a un expediente; y, por último, extiende de **DocumentoEnviable** para poder lograr un seguimiento de las áreas por donde transitó el mismo.

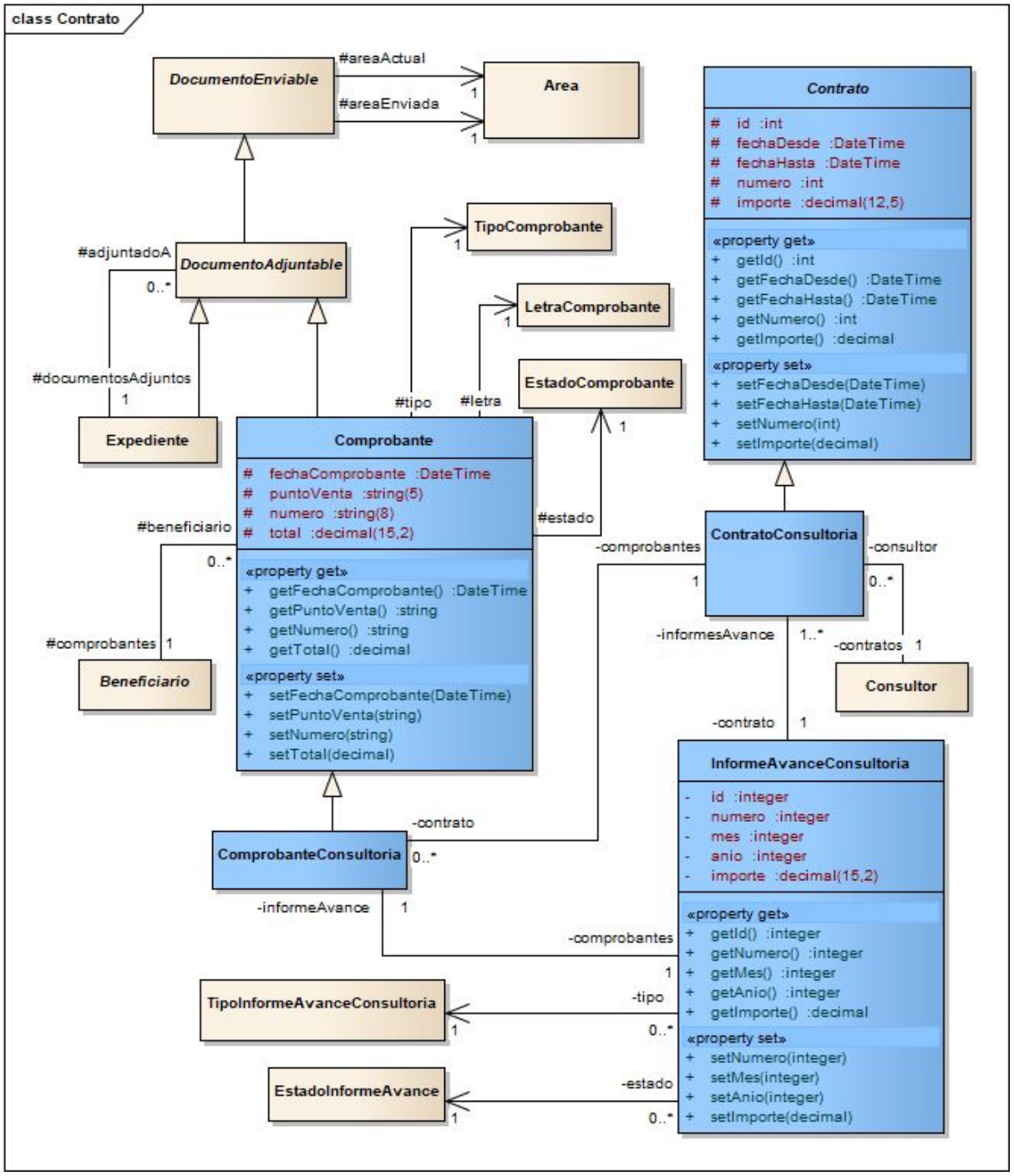

Figura 5.2: Diagrama de clases de Contratos

Como se mencionó previamente, los comprobantes cargados por los consultores deben ser aprobados por los liquidadores. De este modo, se decidió no utilizar la entidad **Comprobante** para los comprobantes cargados por los consultores, sino que se optó por modelar otra entidad que tenga los datos del comprobante, la copia digital del comprobante cargado por el consultor, la constancia de CUIT del momento de carga y que tenga una asociación con el informe de avance de un contrato de consultoría. Como se puede ver en

la Figura 5.3 la entidad **ComprobanteConsultoriaAutogestion** es la utilizada para almacenar los datos resultantes de la carga de comprobante de un consultor. La misma es utilizada por los liquidadores tanto para rechazar - lo que genera envío de mail indicando los motivos - como para aprobar y generar un Comprobante en el sistema. De esta manera sólo van a aparecer los comprobantes en el sistema una vez que fueron aprobados por los liquidadores.

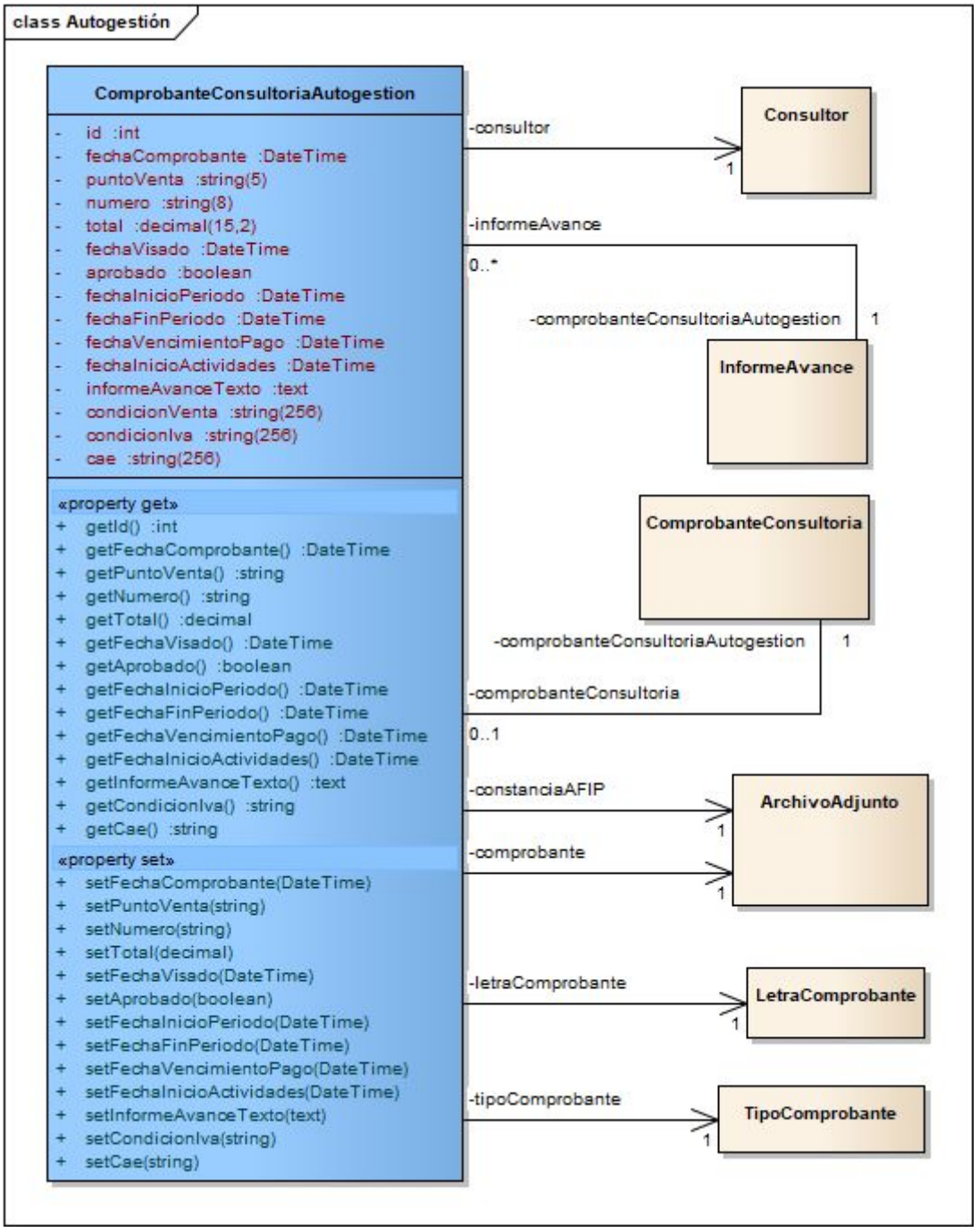

Figura 5.3: Diagrama de clases de Autogestión

Por último, tenemos el proceso de liquidación. Teniendo en cuenta que se pueden liquidar varios comprobantes a la vez, se optó por modelar una entidad **Liquidacion** que tiene una o varias órdenes de pago. Cada orden de pago contiene uno o más comprobantes. Esto último se debe a que, al momento de liquidar, si se liquidan más de un comprobante de un consultor, se puede optar por "agrupar por CUIT", lo que generaría una orden de pago con los comprobantes de ese consultor. La liquidación cuenta de dos etapas, una llamada la preliquidación, en la cual se generan autorizaciones contables (AC); y la etapa de liquidación donde se generan las órdenes de pago (OP). A nivel de información, la diferencia entre las AC y las OP es que a las órdenes de pago se les asigna una fecha y un número. Debido a esto se optó por modelar todo en una sola entidad denominada **OrdenPago**. La misma tiene relación con los comprobantes y tiene una serie de retenciones (entidad **Retencion** en la Figura 5.4) por cada impuesto. En este caso se calculan solamente IIBB y Ganancias. Si repasamos los requerimientos del consultor, ésta última entidad nos permite resolver la descarga de las retenciones de los comprobantes cargados por los mismo. En la Figura 5.4 también se puede observar otra relación ManyToMany para tener un histórico de las liquidaciones.

Cabe una mención adicional sobre cómo se programaron los usuarios y sus permisos. Para modelar la entidad Usuario se utilizó el bundle FOSUser debido a que permite utilizar usuarios con grupos de permisos o roles. Esto facilita generar grupos con roles específicos de acuerdo al área en que se desempeña cada usuario. Por ejemplo, tenemos usuarios que se encargan de validar las facturas o recibos y otros se encargan de liquidarlas. Adicionalmente se agregaron áreas a las cuáles los usuarios hacen los envíos de los documentos y programas con los que pueden operar.
Con esto hemos concluido el repaso por las entidades más relevantes del sistema.

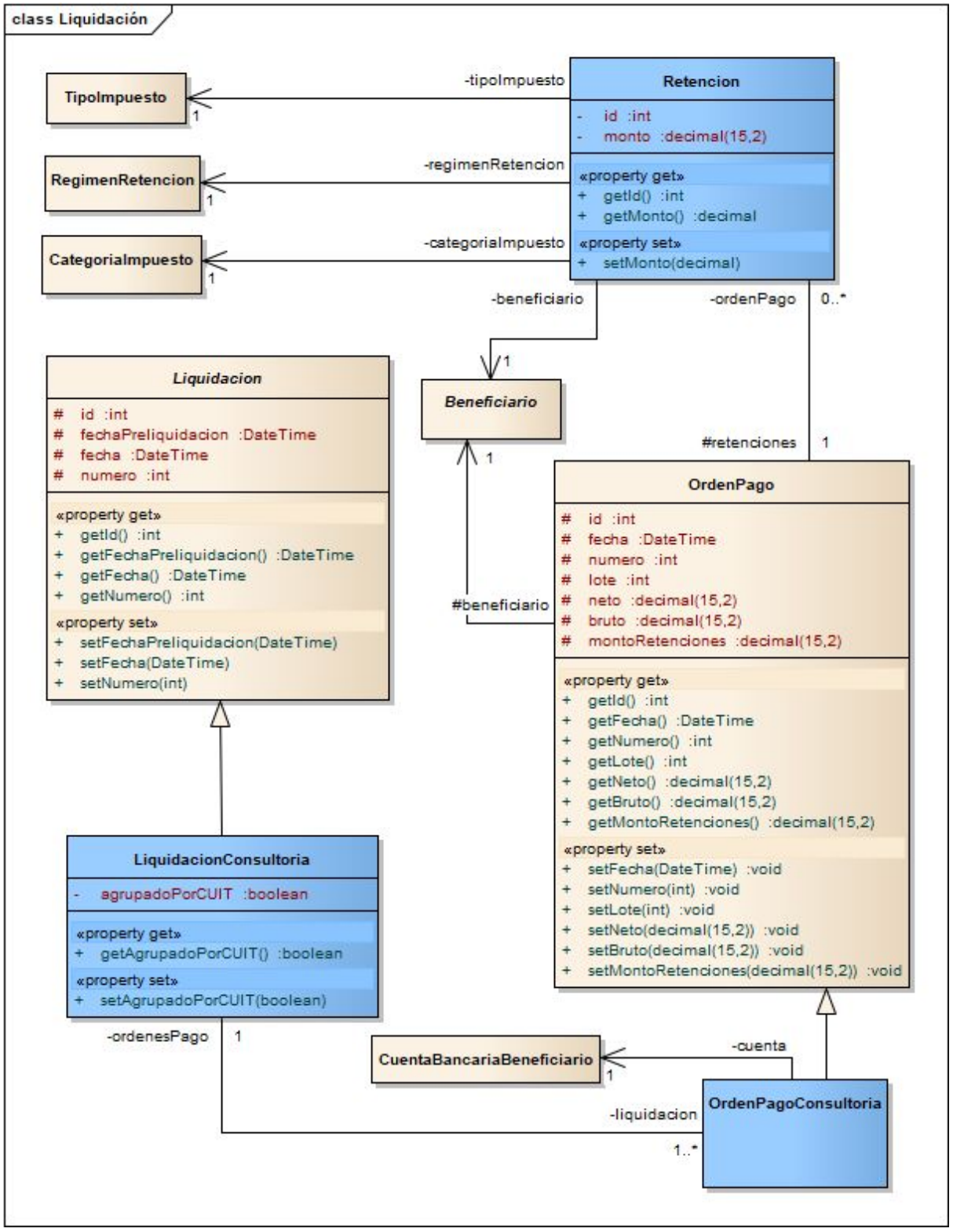

Figura 5.4: Diagrama de clases de Liquidación

5.2. Detalles tecnológicos

A continuación, se describirá un conjunto de cuestiones técnicas que se presentaron a la hora de desarrollar el sistema.

Los usuarios que acceden al frontend son consultores que poseen cuenta en AFIP. En un principio se había considerado la posibilidad de hacer un login con las credenciales de acceso a AFIP. Esta idea fue descartada debido a que muchos de los usuarios de frontend ya tenían cuenta en el sistema CASIA, y se optó por que los usuarios tengan una sola cuenta, dejando sólo la del sistema y no la posibilidad de AFIP.

Para poder realizar validaciones y obtener ciertos datos de cada consultor de AFIP se utilizó una serie de web services. Cabe recordar que un servicio web es un sistema de software diseñado para soportar la interacción máquina-a-máquina, a través de una red, de forma interoperable. Cuenta con una interfaz descrita en un formato procesable por un equipo informático (específicamente en WSDL), a través de la que es posible interactuar con el mismo mediante - por ejemplo - el intercambio de mensajes SOAP, típicamente transmitidos usando serialización XML sobre HTTP conjuntamente con otros estándares web.

Particularmente se utilizaron los webservices de AFIP para realizar tareas como:

- verificar que el consultor esté inscripto en Monotributo o que sea Responsable Inscripto
- obtener la fecha de inscripción en el impuesto para hacer controles adicionales
- obtener la Constancia de Inscripción - AFIP del consultor
- validar la factura mediante CAE (Código de Autorización Electrónico)

En la Figura 5.5 se presenta un ejemplo de cómo se solicita la validación de CAE, mediante el método constatar del WebService WSCDC de AFIP. En la misma se pueden apreciar los parámetros enviados para solicitar dicha constatación. También se puede apreciar que se envía un objeto Auth que contiene los datos de autenticación CUIT, un Token y Sign, los cuales se obtienen al generar una autenticación exitosa con otro WebService de AFIP denominado WSAA (Webservice de Autenticación y Autorización).

En la Figura 5.6 se puede apreciar una respuesta exitosa en la validación de CAE de la solicitud consultada en la Figura 5.5

```xml
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsf="http://servicios1.afip.gob.ar/wscdc/">
 <soapenv:Header/>
 <soapenv:Body>
 <ComprobanteConstatar>
 <Auth>
 <Token>111</Token>
 <Sign>11111111</Sign>
 <Cuit>300000000007</Cuit>
 </Auth>
 <CmpReq>
 <CbteModo>CAE</CbteModo>
 <CuitEmisor>20000000001</CuitEmisor>
 <PtoVta>1</PtoVta>
 <CbteTipo>1</CbteTipo>
 <CbteNro>2</CbteNro>
 <CbteFch>20101014</CbteFch>
 <ImpTotal>300.8</ImpTotal>
 <CodAutorizacion>60428000005029</CodAutorizacion>
 <DocTipoReceptor>80</DocTipoReceptor>
 <DocNroReceptor>300000000007</DocNroReceptor>
 </CmpReq>
 </ComprobanteConstatar>
 </soapenv:Body>
</soapenv:Envelope>
```

Figura 5.5: Ejemplo de solicitud de validación de CAE

```xml
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <soap:Body>
 <ComprobanteConstatarResponse>
 <ComprobanteConstatarResult>
 <CmpResp>
 <CbteModo>CAE</CbteModo>
 <CuitEmisor>20000000001</CuitEmisor>
 <PtoVta>1</PtoVta>
 <CbteTipo>1</CbteTipo>
 <CbteNro>2</CbteNro>
 <CbteFch>20101014</CbteFch>
 <ImpTotal>300.8</ImpTotal>
 <CodAutorizacion>60428000005029</CodAutorizacion>
 <DocTipoReceptor>80</DocTipoReceptor>
 <DocNroReceptor>30000000007</DocNroReceptor>
 </CmpResp>
 <Resultado>A</Resultado>
 <FchProceso>20130729204436</FchProceso>
 </ComprobanteConstatarResult>
 </ComprobanteConstatarResponse>
 </soap:Body>
</soap:Envelope>
```

Figura 5.6: Ejemplo de respuesta exitosa de validación de CAE

Cuando el consultor carga la factura en el sistema, ésta se analiza y procesa para obtener los datos de la misma. Para satisfacer los requerimientos, es necesario obtener del comprobante los siguientes datos: fecha, punto de venta, número, importe total, tipo de comprobante (Factura/Recibo), letra del comprobante (B/C/E), período facturado, fecha de vencimiento para el pago, número de CAE, cuit al que se factura, condición frente al IVA del cuit facturado, condición de venta y fecha de inicio de actividades.
Una vez que la factura llega al servidor, se realiza la verificación de que el archivo adjunto esté en formato PDF. Los comprobantes descargados directamente de AFIP contienen seguridad mediante contraseña que restringe el uso de los mismos. Sólo permite hacer impresiones y se restringe todo tipo de modificación. Utiliza un nivel de codificación RC4 de 128 bits. RC4 es un esquema de cifrado de flujo simétrico - que como se demuestra por ejemplo en [29] no es completamente seguro - pero se sigue utilizando debido a que es un cifrado rápido y es soportado en una gran variedad de hardware. Por este motivo se utilizó el aplicativo qpdf[9] que permitió desbloquear los documentos. Adicionalmente fue necesario convertir los archivos desbloqueados a la versión 1.4 de pdf debido a que la librería php que se utilizó para parsear los documentos soporta hasta esta versión sin inconvenientes. Asimismo, 1.4 es la versión que utiliza AFIP para generar la mayoría de los pdf. Si bien es un formato del año 2001, no es de extrañar que AFIP lo continúe utilizando para proporcionar mayor compatibilidad.

A fin de desbloquear los archivos PDF, en una primera instancia se procedió a una búsqueda de determinados términos dentro del documento que permitieron la obtención de datos. Si bien esta “técnica” funcionó correctamente en un muestreo de casi 1000 facturas que nos habían proporcionado, se tuvo que optar por otra opción, dado que el código dependía mucho de que se respetara cierta estructura en los datos, y como se explicó previamente, no era una solución completa debido a que algunos comprobantes fallaban. Si bien era una cantidad mínima, debido a que esta funcionalidad iba a ser usada por miles de usuarios se optó por considerar otra opción.

El problema radicaba en que no todos los consultores generaban el comprobante de la misma manera o utilizaban algún software generador de pdf alternativo. La mayoría utilizaba el sitio de AFIP para generar el mismo, mientras que algunos pocos la generaban mediante web services de AFIP. Otro inconveniente fue que AFIP no siempre genera el comprobante de la misma manera, es decir que no siempre están los datos en la misma ubicación.

[9] http://qpdf.sourceforge.net/

Para resolver esto se optó por la generación de patrones para identificar cada uno de los datos del comprobante. Cada patrón contiene el índice (atributo “index”) donde se encuentra la palabra clave, y el índice del valor (atributo “value_index”) donde se debe ir a buscar el dato. Se generó un archivo JSON que contiene un arreglo con todos los patrones. También se programó un generador de patrones para resolver los casos nuevos que se fueron generando. Actualmente existen 33 patrones para los comprobantes de tipo factura y 10 para los comprobantes de tipo recibo. En la Figura 5.7 mostrada a continuación se puede ver el detalle de la estructura del archivo de patrones para parsear los comprobantes:

```
[
  {"TemplateName": "Factura1",
    "content": {
    "Fecha Emision": {"index": 1,"value_index": 11},
    "Tipo Factura": {"index": 2,"value_index": 25},
    "Periodo Facturado": {"index": 5,"value_index": 10},
    "Condicion Venta": {"index": 6,"value_index": 14},
    "Condicion IVA": {"index": 7,"value_index": 27},
    "Apellido y Nombre": {"index": 8},
    "Domicilio": {"index": 9},
    "DGCyE": {"index": 13,"value_index": 13},
    "CUIT Emisor": {"index": 15,"value_index": 12},
    "IIBB": {"index": 16},
    "Inicio Actividades": {"index": 17,"value_index": 29},
    "Punto Venta": {"index": 18,"value_index": 18},
    "Domicilio Comercial": {"index": 19,"value_index": 4},
    "CUIT DGCyE": {"index": 30,"value_index": 30},
    "Subtotal": {"index": 37},
    "Importe otros tributos": {"index": 38},
    "Importe Total": {"index": 39,"value_index": 36},
    "CAE": {"index": 43,"value_index": 47},
    "Fecha Vencimiento CAE": {"index": 44},
    "Esta Administracion": {"index": 45},
    "Pagina": {"index": 50}
    }
  },
  {"TemplateName": "Factura2",
    ...
  },
  ...
]
```

Figura 5.7: Template de patrones de comprobantes

En la Figura 5.8 se muestra el código resumido utilizado para extraer los datos de los comprobantes. Primeramente, se identifica la posición de ciertas palabras clave dentro del documento y se genera un arreglo con la posición de cada elemento presente en el template antes mencionado. Luego se compara el template para ver si existe alguna

coincidencia con uno de los patrones. Si se encuentra una coincidencia, se extraen los datos de cada elemento en donde indica cada patrón. Finalmente se realizan validaciones sobre los datos obtenidos para determinar si los mismos son correctos, como por ejemplo que el CUIT obtenido de la factura sea un número y tenga los once dígitos que componen dicho elemento. Este método también se utiliza para agregar nuevos patrones.

```
/**
* Recibe el nombre de un archivo en formato pdf desbloqueado,
* parsea el mismo, y extrae los datos. Se retornan en un array
*
* @param type $file - Nombre del archivo con path completo
* @param type $tpl - Template
* @return array
*/
public function extractData($file, $tpl = NULL) {
  $pdf = $this->parser->parseFile($file . '.pdf');
  $details = $pdf->getDetails();
  $pages = $pdf->getPages();
  $datos = explode("\n", $pages[0]->getText());
  //Archivo de templates, tengo que ir a buscarlos a disco
  $tmp_saved = file_get_contents(dirname($this->container->getParameter('kernel.root_dir')) .
'/src/AppBundle/Service/templateFacturas.json');
  $tmp_saved = json_decode($tmp_saved, TRUE);
  //Leo el patron de datos del pdf
  $datos_leidos['Fecha Emision'] = array('index' => key(preg_grep('/Fecha de Emisión\b/', $datos)));
  $datos_leidos['Apellido y Nombre'] = array('index' => key(preg_grep('/Apellido y Nombre\b/', $datos)));
  //... se obvian el resto de los campos
  //Busco el patron de datos en el archivo de patrones, si existe devuelvo el patron, si no false
  if ($tpl !== null) {
    $template = $tpl;
  } else {
    $template = $this->compareTemplate($datos_leidos, $tmp_saved);
  }
  $template = $template['content'];
  $cuitComprobante = trim($datos[$template['CUIT Emisor']['value_index']]);
  $fechaComprobante = trim($datos[$template['Fecha Emision']['value_index']]);
  //...
  //Validaciones sobre cada uno de los campos que se extraen
  $status = array('codComprobante' => 'OK',
      'caeComprobante' => 'OK',
          //...
  );
  //Cuit del comprobante
  if (strlen($cuitComprobante) != 11 || !is_numeric($cuitComprobante)) {
    $status['cuitComprobante'] = array('El número de CUIT es incorrecto ', $cuitComprobante);
  }
  //Fecha del comprobante
  $regexFecha = '/^([0-2][0-9]|3[0-1])(\/|-)(0[1-9]|1[0-2])\2(\d{4})$/';
  if (!preg_match($regexFecha, $fechaComprobante)) {
    $status['fechaComprobante'] = array('La fecha del comprobante es incorrecta ', $fechaComprobante);
  }
  //...
  $fact = array('codComprobante' => $codComprobante,
      'caeComprobante' => $caeComprobante,
      //...
      'status' => $status
  );
  return $fact;
}
```

Figura 5.8: Método para extraer datos de los comprobantes

Tanto la comunicación con los web services de AFIP como el desbloqueador y parser de los comprobantes, fueron programados en un *service* de symfony. Según la documentación oficial del libro de symfony:

“...un servicio es cualquier objeto PHP que realiza algún tipo de tarea global. La palabra servicio es una palabra deliberadamente genérica que se utiliza en el ámbito de la informática para describir un objeto creado para un propósito específico (por ejemplo, el envío de emails). Cada servicio se puede utilizar en cualquier parte de la aplicación…”
“...La ventaja de pensar en servicios es que obliga a separar cada funcionalidad de la aplicación en una serie de servicios. Como cada servicio se limita a una tarea específica, es posible acceder fácilmente a cada servicio y usar su funcionalidad siempre que la necesites...”.

Armand define claramente con un ejemplo en [6] que un servicio es solo una instancia específica de una clase dada. Por ejemplo, cada vez que se accede a doctrine (mapeador objeto-relacional (ORM) escrito en PHP que proporciona una capa de persistencia para objetos PHP) con $this->get('doctrine') en un controlador, implica que se está accediendo a un servicio. Este servicio es una instancia de la clase Doctrine EntityManager, pero ésta instancia nunca debe ser creada por uno mismo. El código necesario para crear este administrador de entidades en realidad no es tan simple ya que requiere una conexión a la base de datos, algunas otras configuraciones, y así sucesivamente. Sin este servicio ya definido, se tendría que crear esta instancia en el código propio. Posiblemente sea necesario repetir esta inicialización en cada controlador, lo que hace que la aplicación sea más desordenada y difícil de mantener.
Algunos de los servicios predeterminados presentes en Symfony2 son los siguientes:

- El lector de anotaciones
- Assetics: la biblioteca de gestión de activos
- El despachador de eventos.
- Los widgets de formulario y la fábrica de formularios
- El kernel de Symfony2 y HttpKernel
- Monolog: la biblioteca de registro
- El enrutador
- Twig: el motor de plantillas

Si tenemos un controlador que ha comenzado a ser bastante complicado con código extenso, una buena manera de refactorizarlo y simplificarlo será mover parte del código a los servicios. Se han descripto todos estos servicios comenzando con "el" (o “la”) y un

sustantivo singular. Esto es porque la mayoría de las veces, los servicios serán objetos *singleton*[10] donde se necesita una sola instancia.

Como pudimos observar anteriormente los servicios son una buena práctica para mantener el código ordenado y que sea más fácil de mantener. Por ese motivo se tomó la decisión de hacer servicios tanto para la comunicación con los webservices de AFIP como el desbloqueador y parser de los comprobantes.

[10] Según fue definida en [13]. Un singleton se asegura que la clase tenga una única instancia, y proporcione un punto de acceso global a la misma.

6. Implementación y análisis de resultados obtenidos

El sistema se terminó de desarrollar y las funcionalidades desarrolladas fueron sometidas a pruebas o *testing* por parte de los analistas funcionales del equipo y también por los usuarios finales liquidadores. Al programar el sistema se realizaron pruebas básicas para corroborar el correcto funcionamiento, pero siempre es necesario realizar pruebas de *testing* utilizando técnicas como caja negra y blanca, entre otras, para detectar errores y corroborar el cumplimiento de los requerimientos. Una vez que dichas pruebas concluyeron y se solucionaron los errores encontrados, el sistema se implementó, esto es: se puso en producción. A continuación, se detallarán los resultados de la interfaz tanto para el *frontend* como para *backend* y se contrastarán con los requerimientos y los *mockups* iniciales.

6.1. Interfaces *Frontend*

Estas interfaces se diseñaron pensadas para usuarios finales menos experimentados con software, es por eso que la interfaz es más sencilla de entender y utilizar.
Una vez que el consultor ingresa al sistema, puede visualizar el listado de sus contratos. Como se puede ver en la Figura 6.1 se generó el listado de contratos con un resumen de los datos más relevantes del mismo con un botón para ver el detalle. También se puede observar que se agregó un botón de Ayuda. Si el usuario hace clic en el mismo, se ejecuta un asistente que indica ciertos aspectos de la interfaz y las acciones que puede realizar. Para darle al usuario una experiencia más “amigable”, el asistente se ejecuta la primera vez que ingresa en cada pantalla. Para facilitar la comprensión de la interfaz, se oscurece toda la interfaz a excepción del elemento del sistema que se está explicando. En la Figura 6.2 se puede ver un ejemplo del asistente en funcionamiento.

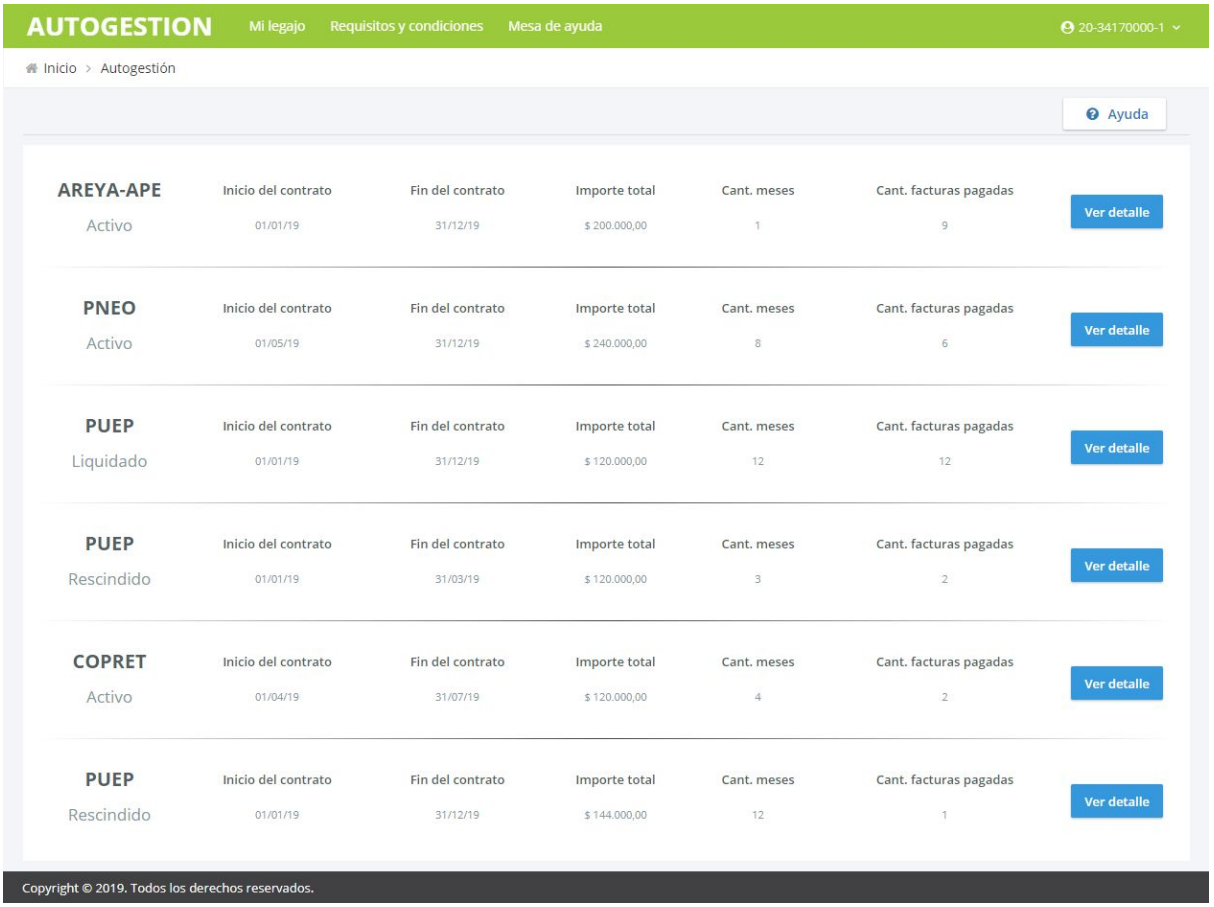

Figura 6.1: Listado de contratos

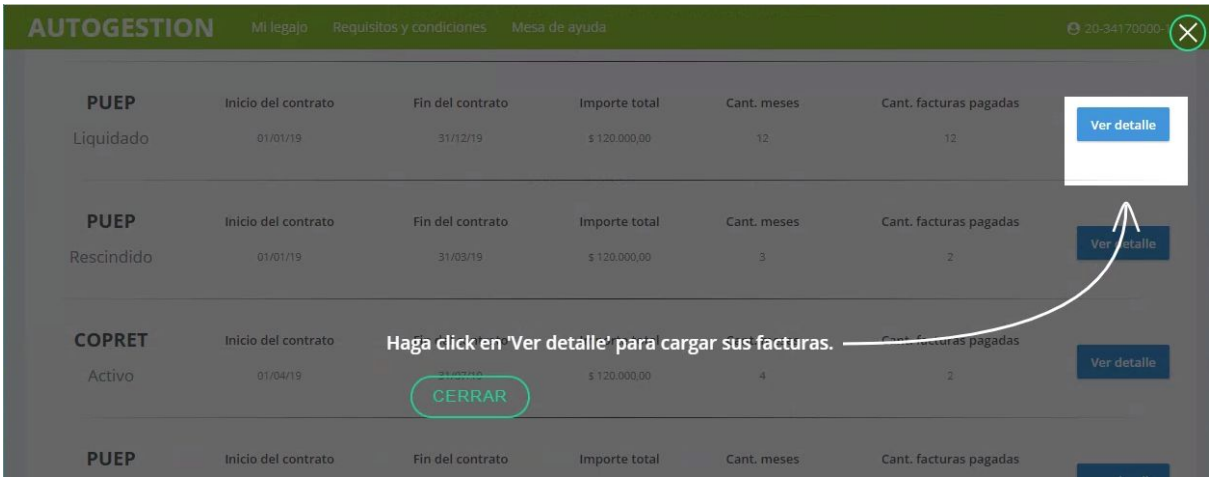

Figura 6.2: Asistente de interfaz

Cuando el usuario ingresa al detalle de un contrato se muestra una cabecera con el resumen de los datos del contrato como el programa, el período y el importe total del mismo, y un listado con todos los informes de los que debe presentar un comprobante para cobrar los honorarios. En la Figura 6.3 se puede observar dicha interfaz, la cual sufrió

ciertas modificaciones con respecto al diseño inicial. Anteriormente en la sección 3 se mostró un *mockup* que tenía un solo botón para cargar la factura o para descargar un pdf con las retenciones del informe de avance. Como se puede observar en la Figura 6.4 se agregó un botón “Ver detalle” que despliega un detalle con el comprobante adjuntado, el texto que indicó en el informe de avance, el pdf con la constancia de CUIT obtenida de AFIP al momento de adjuntar el comprobante y un fragmento con los datos obtenidos al procesar el comprobante. Otro detalle que se alteró fue el botón “Agregar comprobante”. Solicitaron que el mismo aparezca sólo a partir de una fecha que configuran desde el *backend* para cada mes de facturación de cada año. También que el mismo se muestre de color verde mientras se encuentre entre los días 20 y 25 del mes a facturar, y de color naranja cuando la fecha sea posterior al 25 del mes a facturar y no se haya presentado el comprobante.

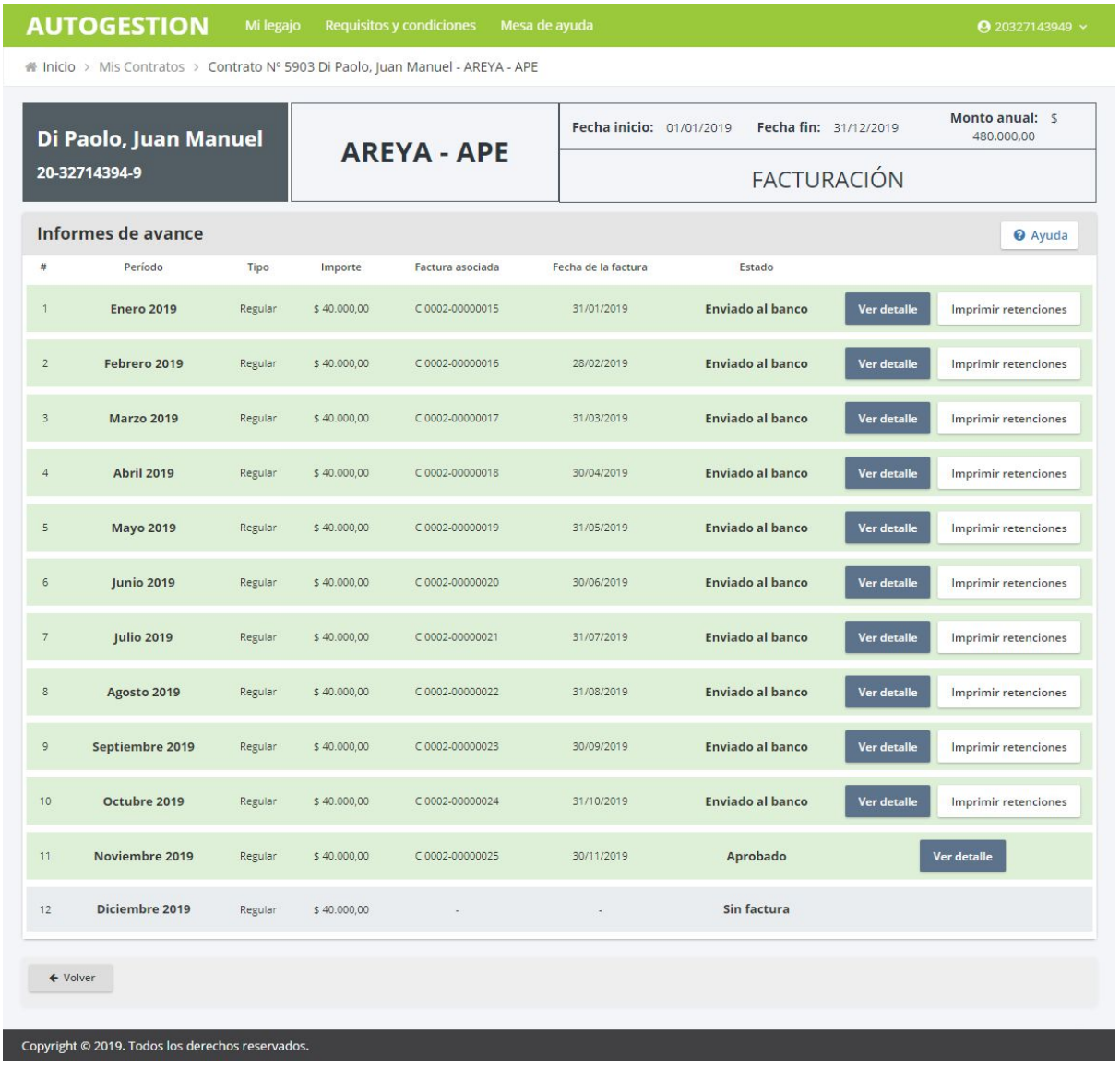

Figura 6.3: Listado de informes de avance de un contrato

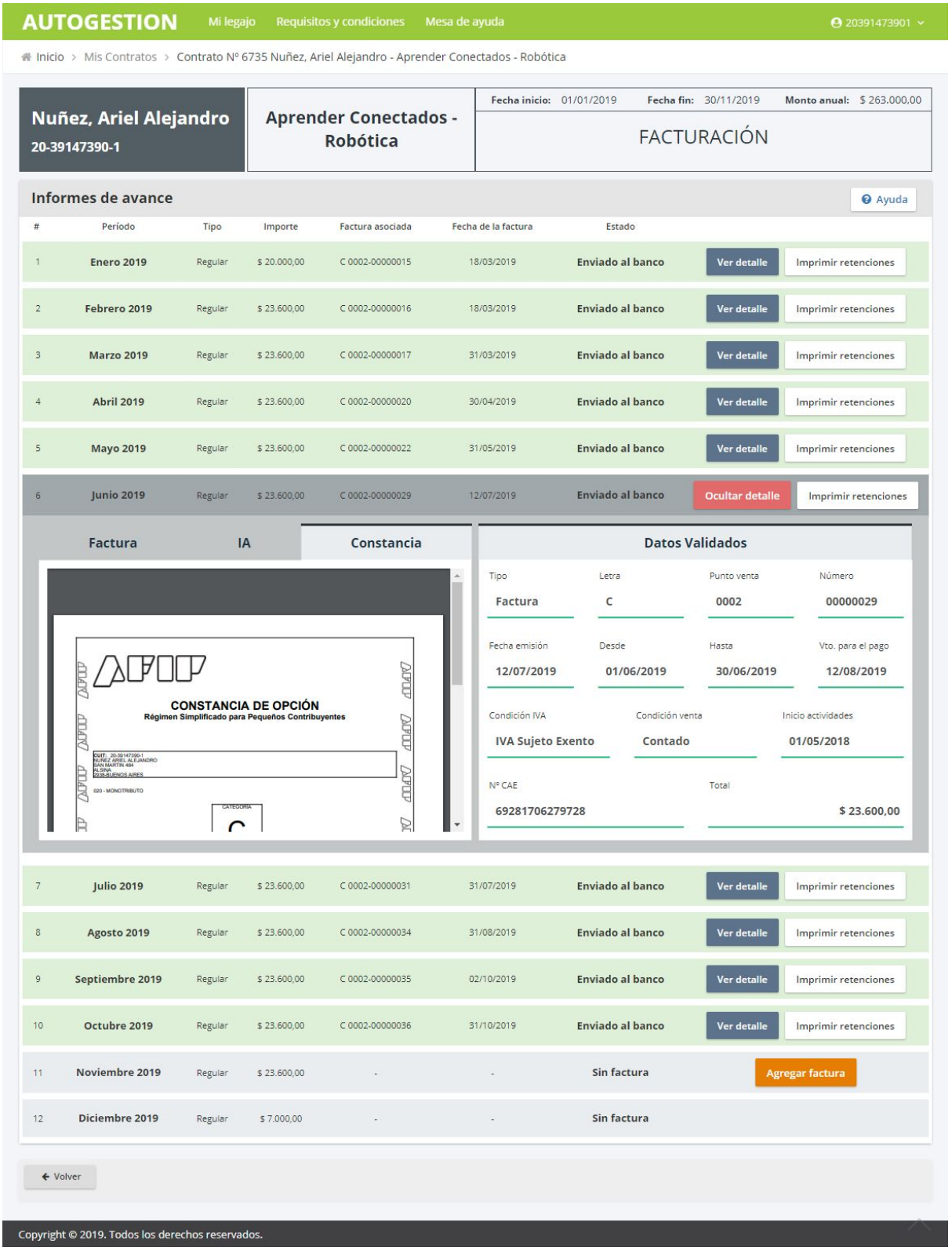

Figura 6.4: Listado de informes de avance de un contrato. Detalle expandido

En la Figura 6.5 se presenta cómo quedó la interfaz de carga de un comprobante. Debido a que es un formulario que cuenta con varios pasos, se agregó un asistente de ayuda que

guía al usuario paso a paso para la carga mismo. Se hicieron algunos cambios estéticos como la modificación de algunos colores para ofrecer un mayor contraste y facilitar la comprensión.

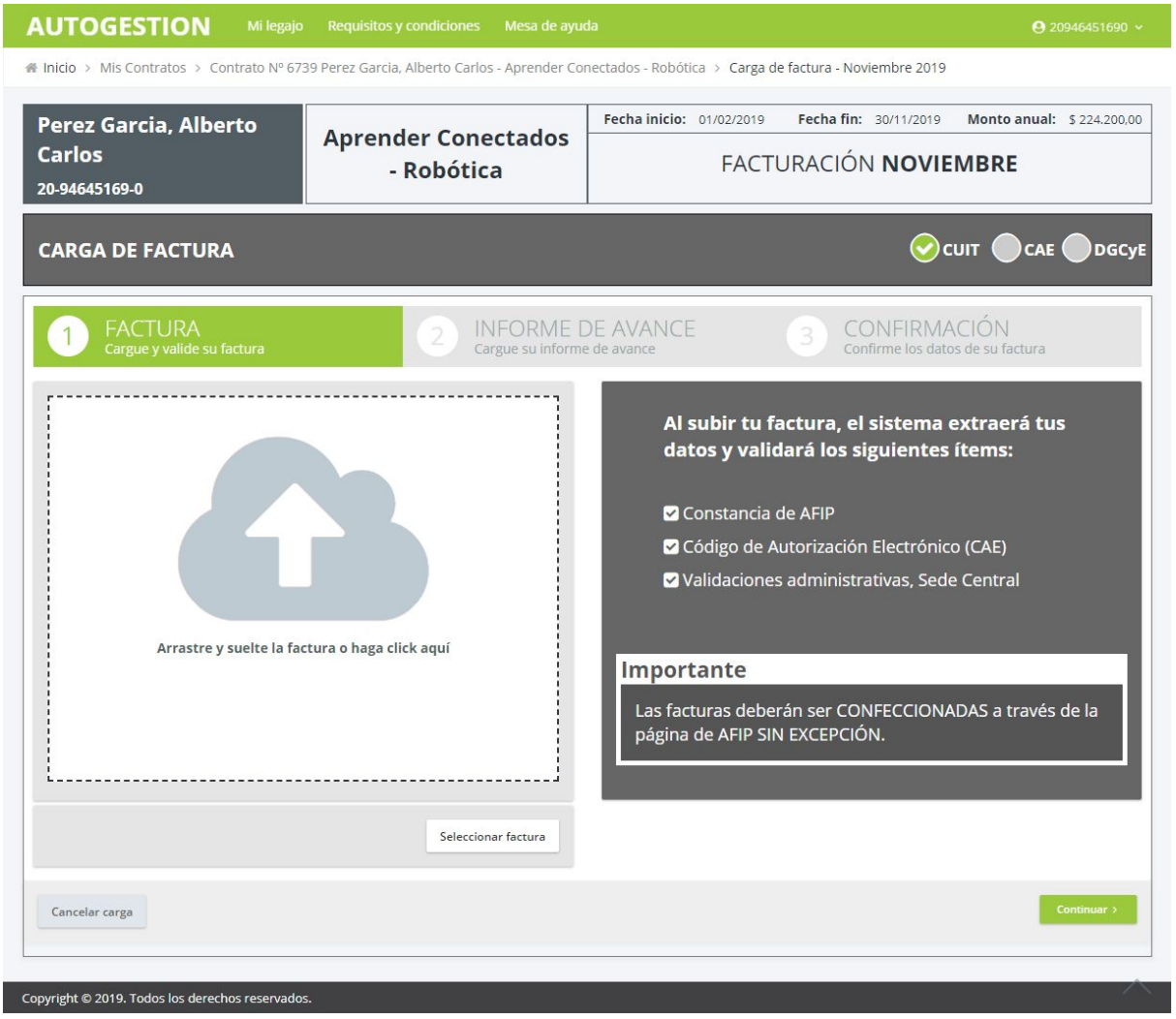

Figura 6.5: Wizard de carga de comprobante

También se mejoró la comunicación de errores al realizar las validaciones al obtener los datos del comprobante. En un principio se había pensado mostrar los mismos como *tooltips* [11], pero se optó por mostrar los mismos en mensajes emergentes más claros como se ve en la figura 6.6. En la misma se puede ver que los errores del formulario se indican como un listado con el motivo del error en un recuadro rojo oscuro. También se indican los errores en cada campo del formulario con color rojo para facilitar la detección de los mismos.

En las Figuras 6.7, 6.8 y 6.9 se muestran los pasos 1, 2 y 3 del *wizard* de carga de facturas.

[11] Globo con texto emergente. Es una ayuda visual para aclarar o indicar un concepto

Figura 6.6: Notificaciones de errores

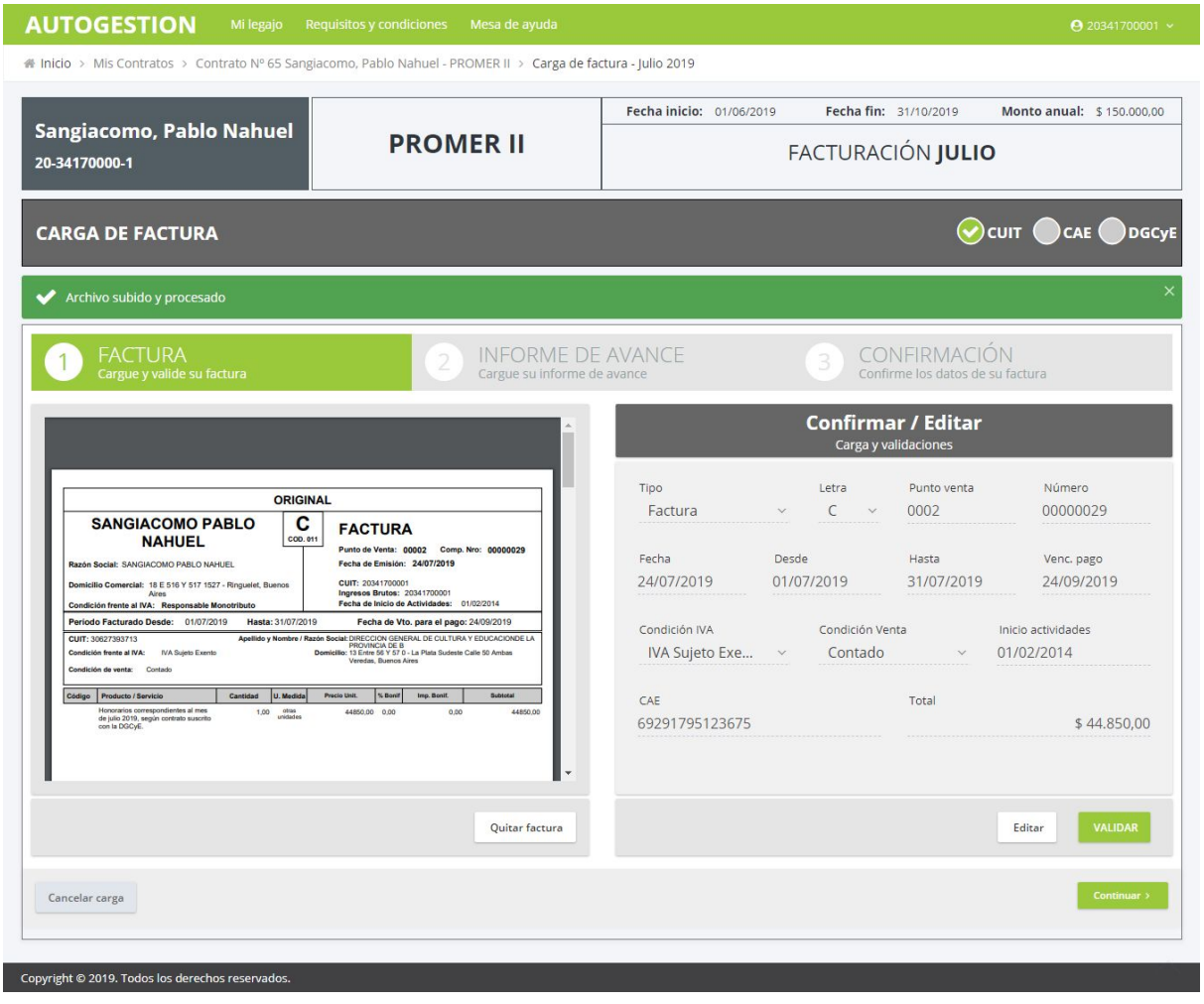

Figura 6.7: Wizard de carga de comprobante. Paso 1

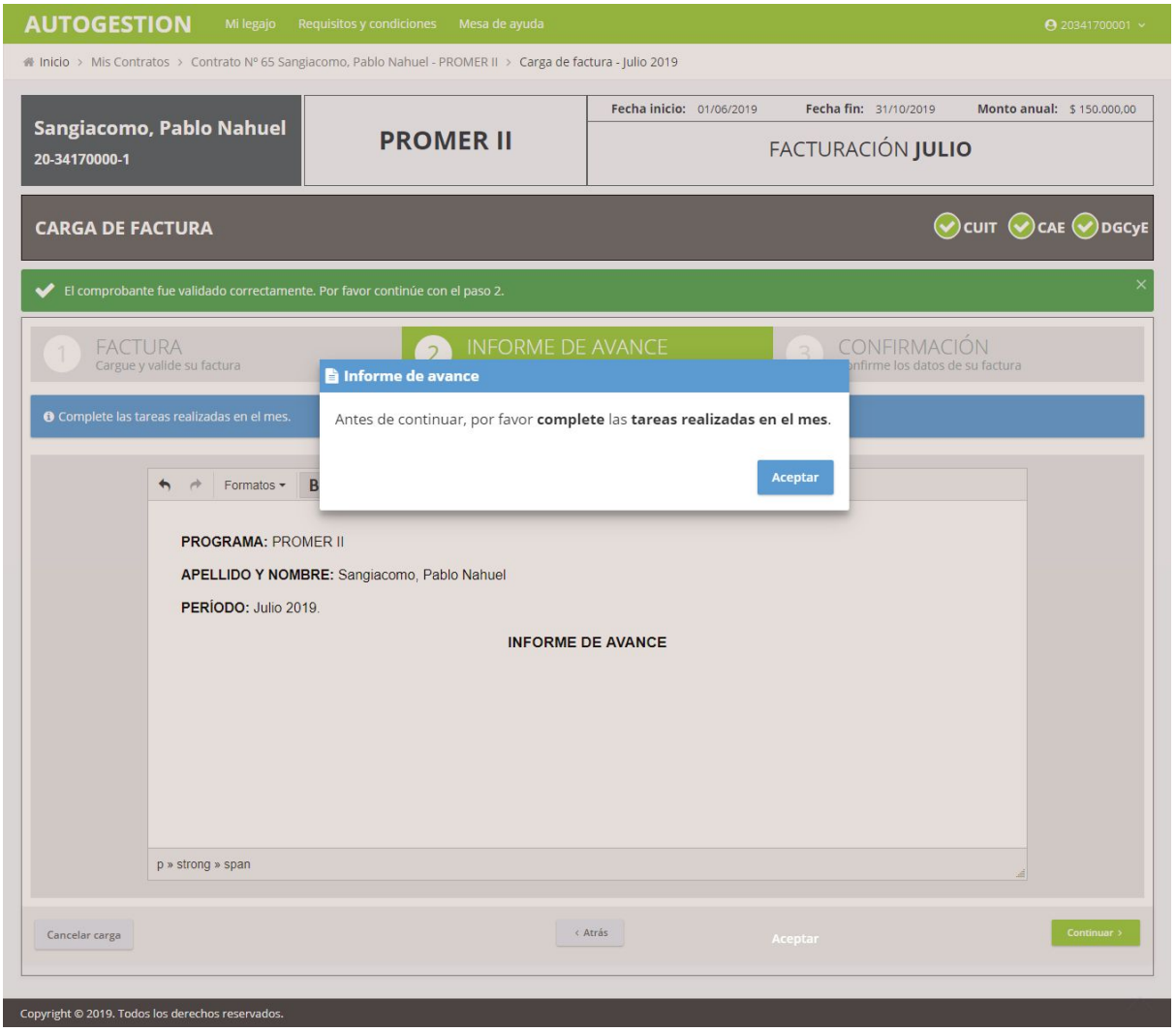

Figura 6.8: Wizard de carga de comprobante. Paso 2

En la figura 6.8 se puede apreciar una notificación para que el consultor preste atención en completar las tareas realizadas durante el mes facturado. Esta notificación fue solicitada debido a que algunos consultores la pasaban por alto y los liquidadores terminaban rechazándoles el comprobante. Para facilitarle esta tarea, al llegar al paso 2, se precargan automáticamente en el campo de texto algunos datos como el programa del contrato, el apellido y nombre del consultor y el período facturado. También se agregó una herramienta *WYSIWYG*[12] para que completen el informe de avance.

[12] WYSIWYG, acrónimo de What You See Is What You Get (en español, "lo que ves es lo que obtienes")

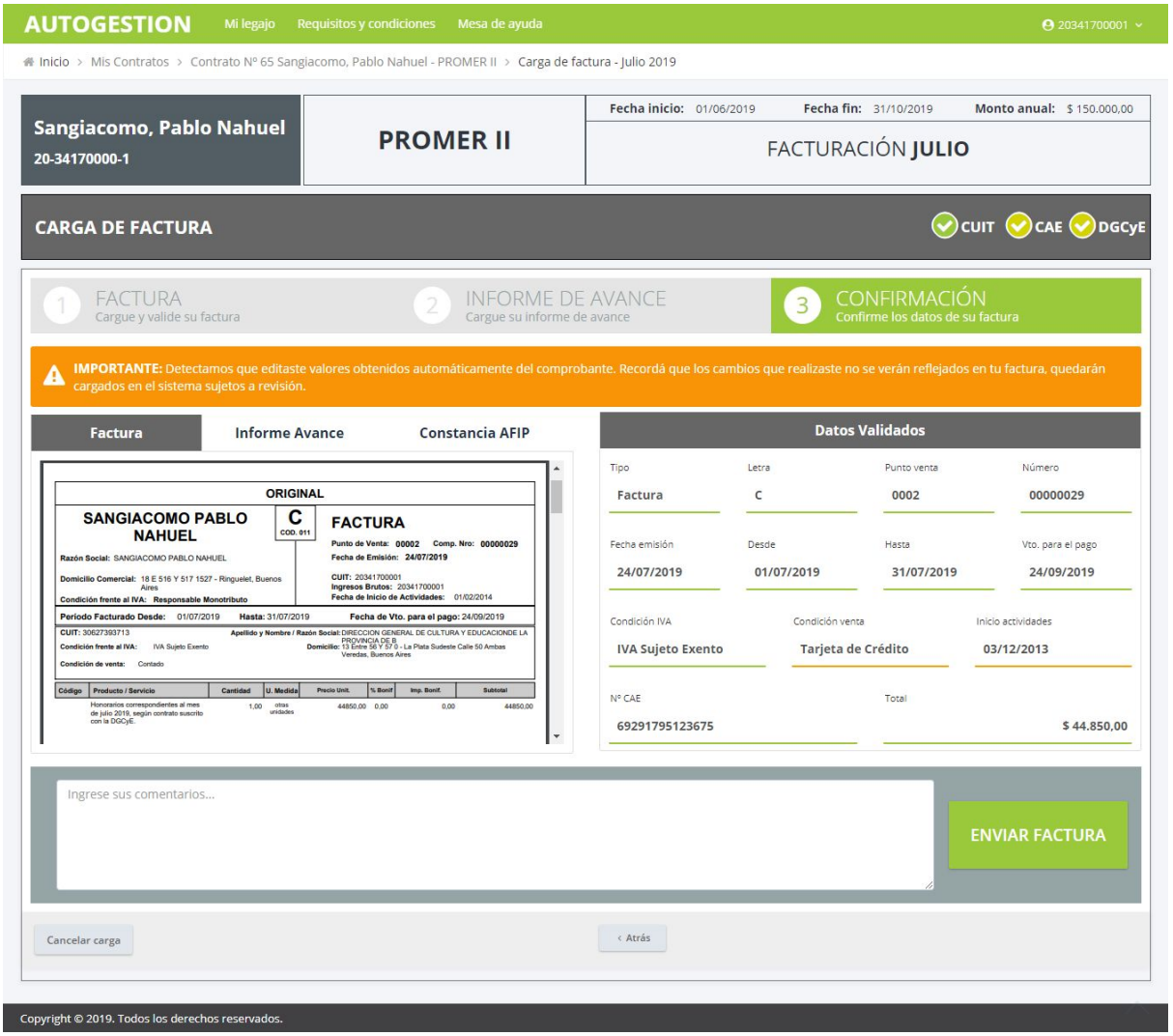

Figura 6.9: Wizard de carga de comprobante. Paso 3

En la figura 6.9 se puede ver qué sucede si el consultor modifica algún dato del formulario de la factura luego de que el sistema haya interpretado los datos de la misma. Se indica con un recuadro naranja que la misma será sujeta a revisión por parte de los liquidadores y también se marcan con naranja los campos que modificó. Recordemos que se dejan modificar los datos interpretados por el sistema debido a que pueden ocurrir errores en el proceso de parseo y se solicitó darle la oportunidad al consultor de corregir los mismos manualmente para luego los liquidadores los revisen.

6.2. Interfaces *Backend*

Estas interfaces se diseñaron pensadas para usuarios con experiencia en el uso de sistemas, por lo que en las mismas se muestran gran cantidad de datos en forma tabular y varias acciones.

Cuando un liquidador da de alta un contrato, el sistema verifica que el consultor tenga usuario. En caso de no tenerlo se genera uno para el mismo. Luego se envía un mail notificando al consultor del nuevo contrato y sus datos de acceso, si se tratase de un usuario nuevo. En la Figura 6.10 se puede observar la interfaz de carga de un contrato. Se puede notar que el formulario de carga es más complejo y la interfaz posee una mayor densidad de componentes visuales con respecto a las mostradas antes del *frontend*.

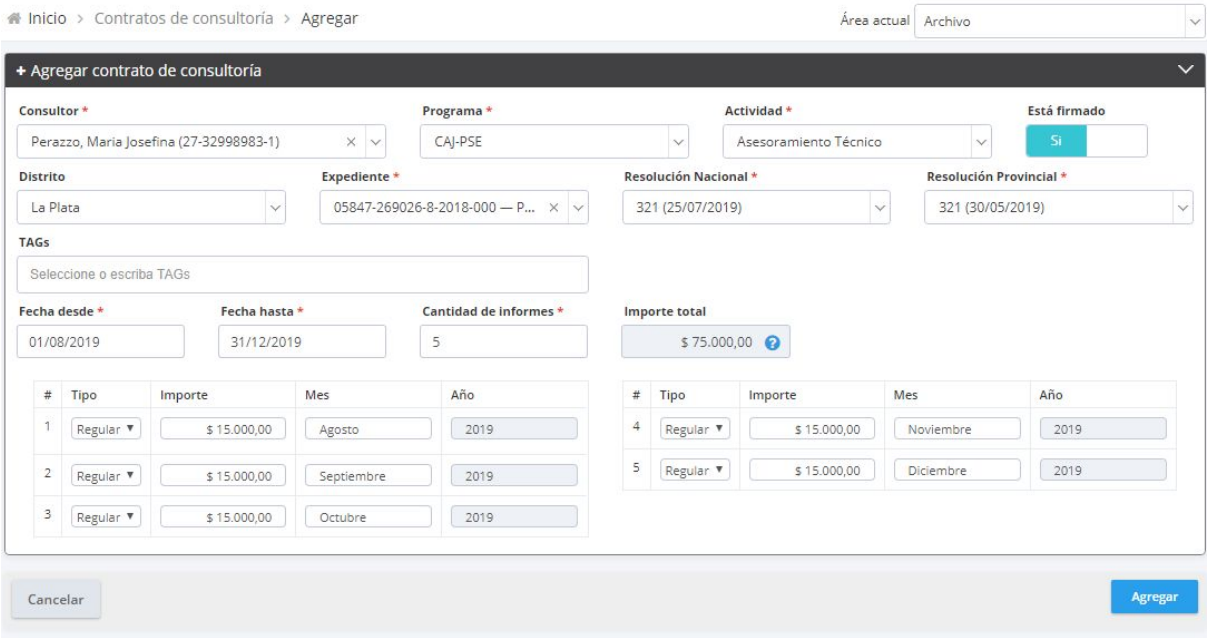

Figura 6.10: Ejemplo de carga de contrato

En la Figura 6.11 se puede ver la interfaz de aprobación/rechazo de los comprobantes de autogestión de consultoría. La línea vertical de color violeta que se observa permite cambiar el ancho de las previsualizaciones y la tabla. De esta manera el liquidador puede hacer las comparaciones de manera más fácil.

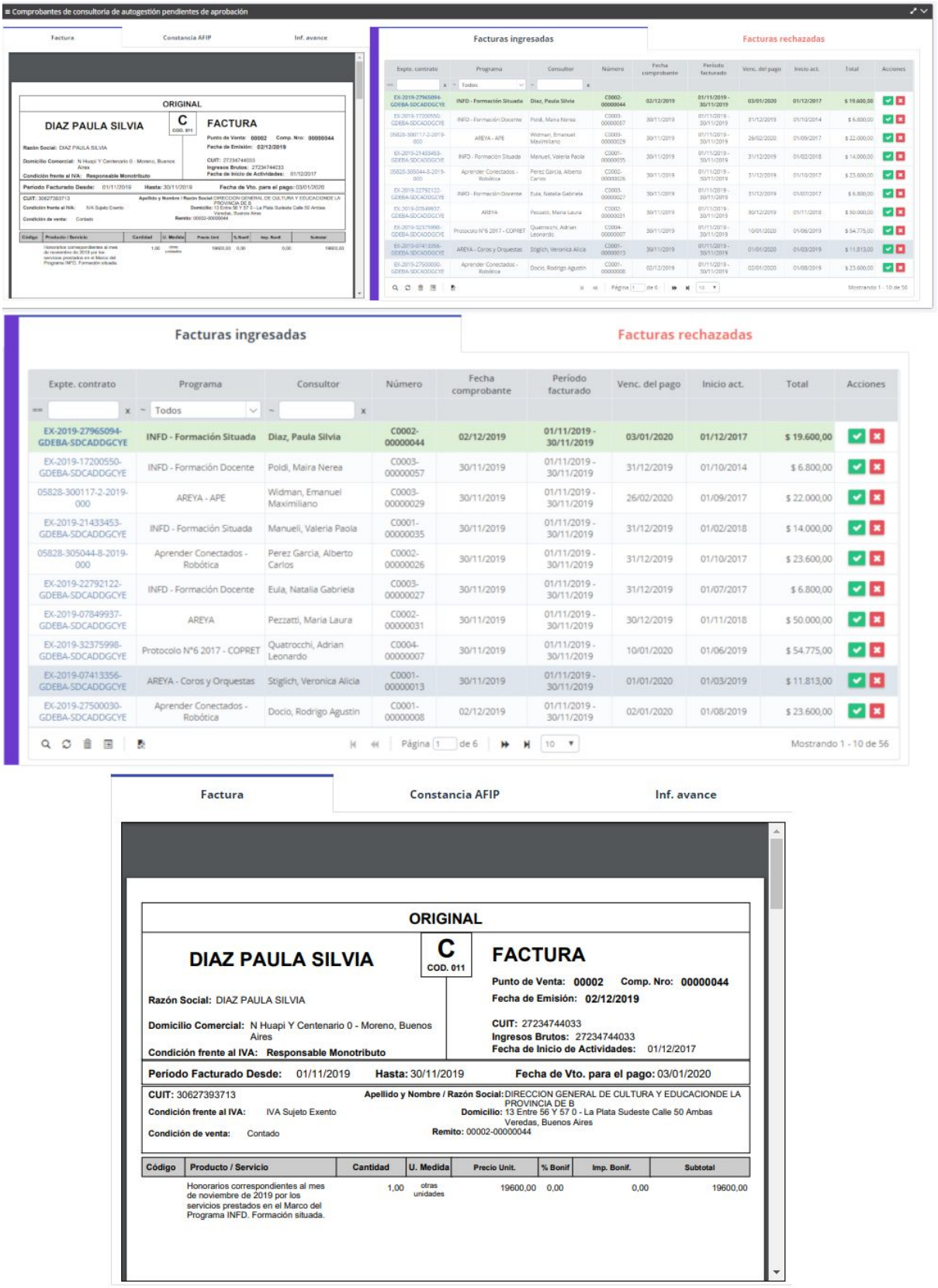

Figura 6.11: Interfaz de aprobación/rechazo de comprobantes

Se agregaron ciertos datos en la tabla para facilitar la tarea de los liquidadores como el expediente. Cuando el liquidador selecciona un comprobante, aparecen tres pestañas con el detalle del adjunto, el informe de avance y la constancia. Si el consultor realizó alguna modificación manual en los datos de la factura, se agrega una pestaña con dichas modificaciones para que el liquidador las pueda evaluar. Cuando el liquidador aprueba un comprobante, el mismo aparece visible en el listado de comprobantes de consultoría para seguir con los posteriores procesos de preliquidación, liquidación y pago. Si el liquidador rechaza un comprobante, se muestra una interfaz para indicar el motivo del rechazo y se envía un mail notificando de dicha situación al consultor. Para facilitarle la tarea al liquidador y generar una notificación homogénea a todos los consultores, se definieron ítems de errores que el liquidador puede seleccionar, y al hacerlo, se va agregando una leyenda predeterminada al contenido del mail a enviar. Esto se puede observar en la Figura 6.12

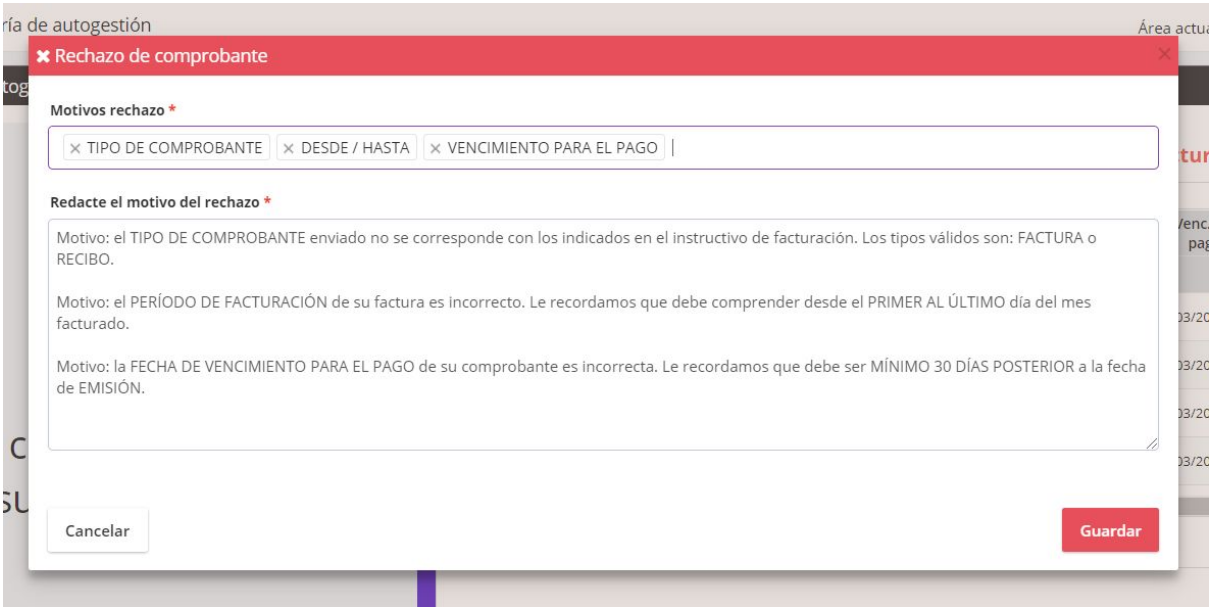

Figura 6.12: Rechazo de comprobante

A continuación, en la figura 6.13 se puede observar una vista previa de una preliquidación indicando para cada comprobante el importe del mismo y las retenciones de IIBB y Ganancias si las hubiera. También se puede ver que está el impuesto IVA, esto es debido a que recientemente se adicionó el cálculo de este impuesto debido a que por una nueva disposición la entidad se convirtió en agente de retención de IVA. Al confirmar la preliquidación se generan planillas de Excel para facilitar los controles de los liquidadores.

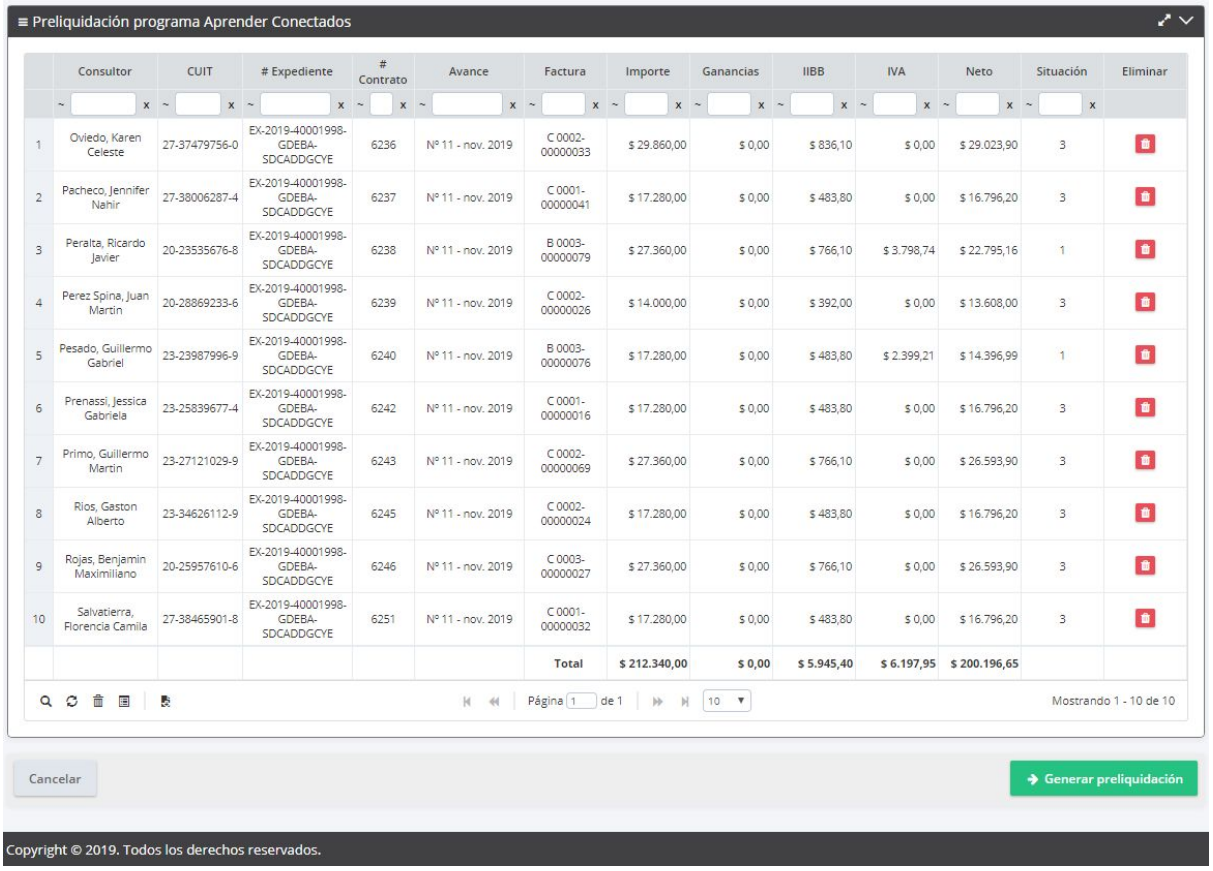

Figura 6.13: Ejemplo de Preliquidación

La carga del adjunto de los comprobantes era muy importante debido a que uno de los objetivos era generar la documentación necesaria para avanzar hasta el proceso de pago de honorarios. Para ello se agregó una funcionalidad para imprimir - de un conjunto de comprobantes liquidados - la primera hoja de la factura adjuntada, el detalle del informe de avance ingresado por los consultores, y la constancia de CUIT al momento de subir el comprobante. Esta funcionalidad aceleró considerablemente la generación de esta documentación debido a que antes debían imprimir las facturas de cada mail que les enviaban los consultores.

Luego de implementar el sistema se solicitó un panel para mensurar la productividad diaria de los liquidadores. Como se puede ver en la Figura 6.14, se incluyó una tabla totalizando para cada programa las facturas cargadas, aprobadas, rechazadas, liquidadas, entre otros datos estadísticos. También en la parte inferior de la Figura 6.14 se agregó un listado con cuántos consultores adeudan la carga de facturas a la fecha del día y un gráfico que contiene la relación de facturas cargadas sobre el total para un mes determinado, el cual puede elegir el usuario que está viendo el reporte.

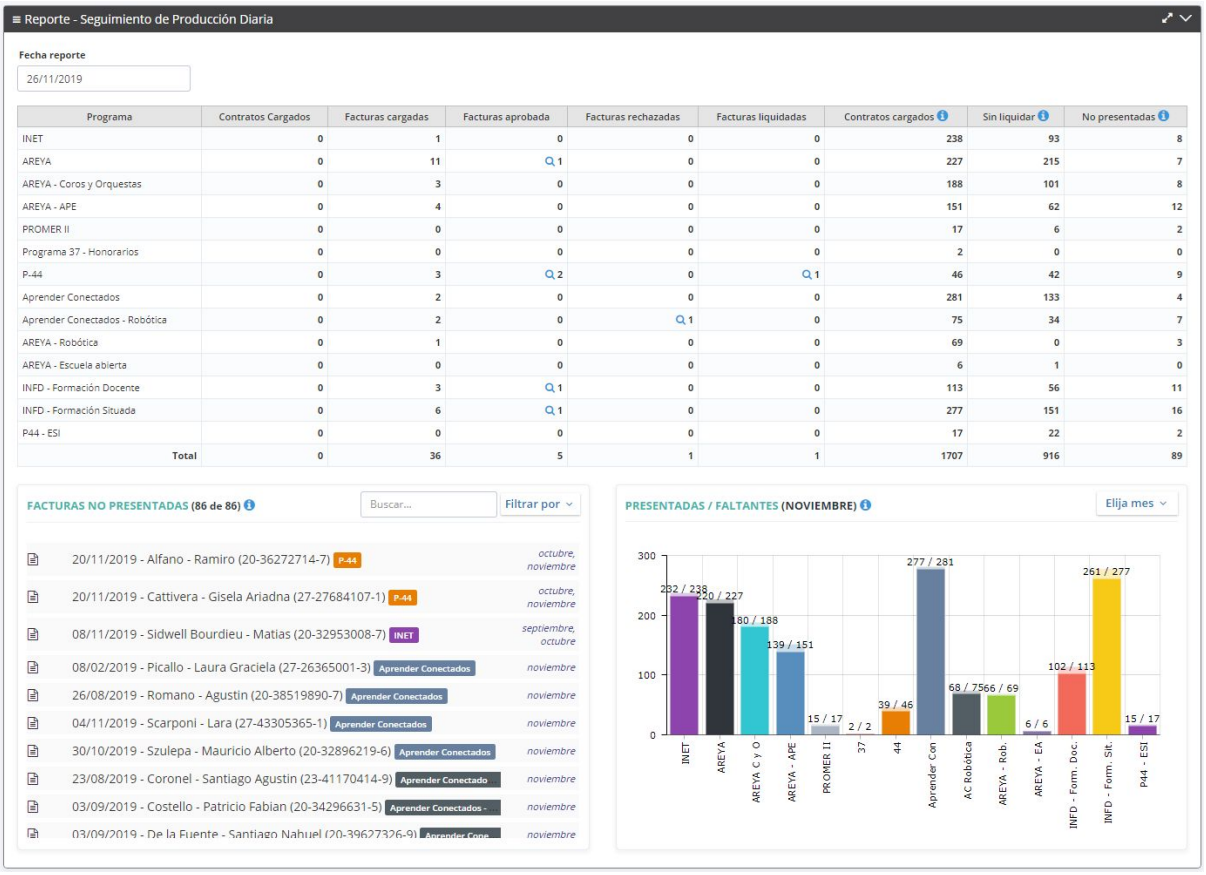

Figura 6.14: Panel gerencial

En un principio se habían planteado que el sistema iba a ser utilizado por más de 4.000 consultores, pero terminaron siendo un poco más de 2.000. Igualmente es un número considerable de usuarios por lo que el sistema y las características del servidor se pensaron para soportar miles de usuarios en simultáneo.

6.3. Encuesta a usuarios finales

Al poco tiempo de implementar el sistema, solicitaron incorporar una encuesta a los consultores. La misma, la deben completar los usuarios al ingresar al sistema por primera vez como se puede observar en la Figura 6.15. Me pareció apropiado incluir la encuesta en el documento de la tesina debido a que la primera pregunta es sobre qué les pareció el sistema, la tercera sobre si la implementación del sistema agilizó el tiempo de cobro y la última sobre qué opina sobre metodología nueva en comparación con la anterior. Contiene cuatro preguntas con valores controlados - donde eligen una opción -, y una última pregunta

de texto libre en la que pueden agregar comentarios adicionales, los cuales sirvieron para ajustar algunos detalles de la interfaz.

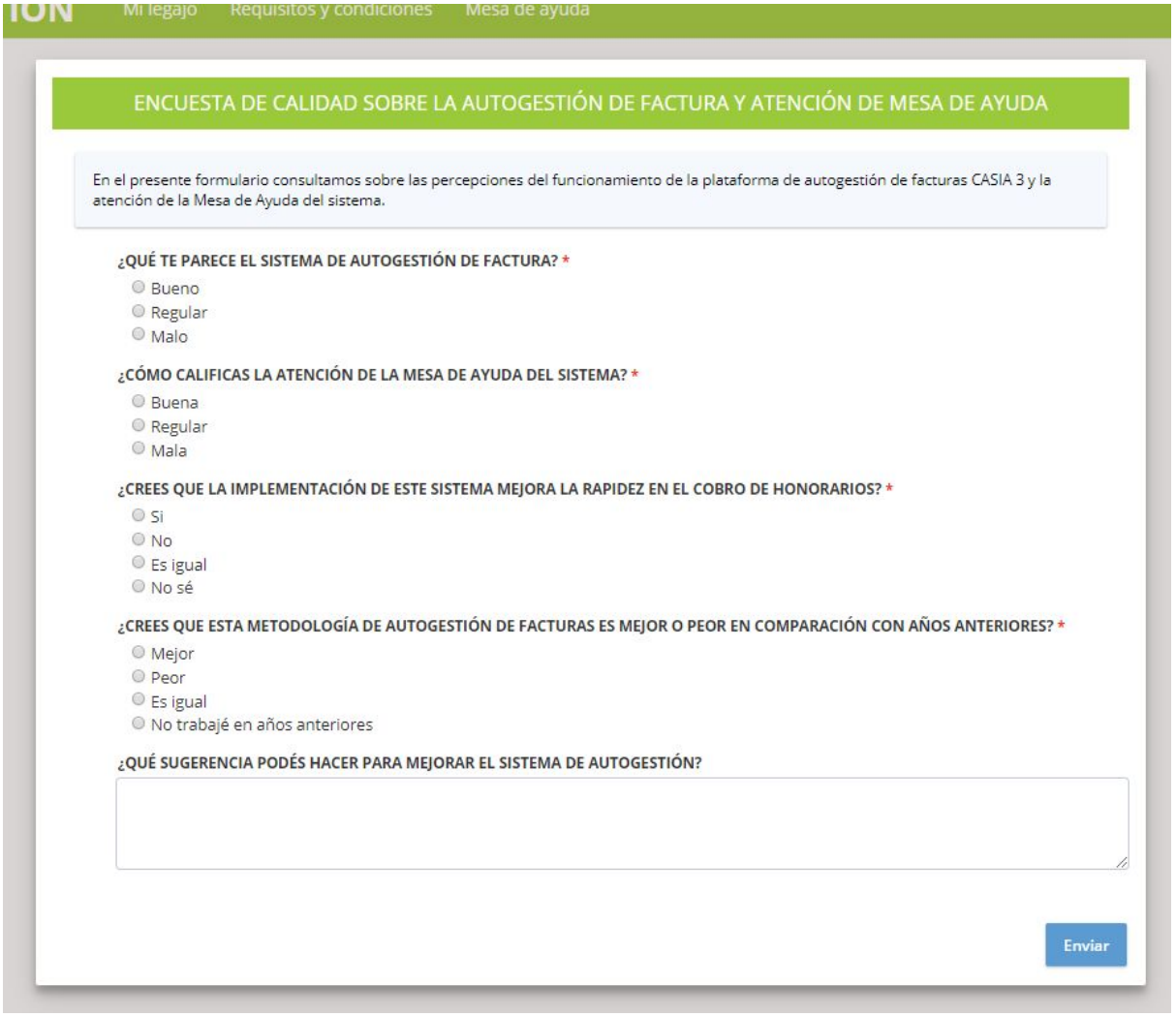

Figura 6.15: Encuesta realizada a los consultores

Posteriormente solicitaron realizar un panel para ver los resultados de la encuesta gráficamente y en forma tabular. El mismo se puede ver en la figura 6.16 en donde hay un gráfico que muestra la relación de la cantidad de consultores que contestaron por sobre la cantidad total; un listado de actividad reciente donde se detalla hace cuánto contestó cada consultor el cuestionario; un gráfico con cada pregunta de valor controlado; y, una tabla que contiene todas las respuestas de cada consultor incluidas las sugerencias y que se puede exportar a excel para futuros análisis.

Figura 6.16: Panel de encuesta

Los resultados de la encuesta fueron los siguientes:

1- ¿Qué te parece el sistema de autogestión de factura?		
Buena	1796	92,10%
Regular	139	7,13%
Malo	15	0,77%

2- ¿Cómo calificas la atención de la mesa de ayuda del sistema?		
Buena	1686	86,46%
Regular	243	12,46%
Mala	21	1,08%

3- ¿Crees que la implementación de este sistema mejora la rapidez en el cobro de honorarios?		
Sí	1394	71,49%
No sé	367	18,82%
Es igual	127	6,51%
No	62	3,18%

4- ¿Crees que esta metodología de autogestión de facturas es mejor o peor en comparación con años anteriores?		
Mejor	1369	70,21%
Es igual	132	6,77%
No trabajé en años anteriores	432	22,15%
Peor	17	0,87%

De los comentarios proporcionados por los consultores se ajustaron algunos detalles como por ejemplo que el asistente se ejecute automáticamente la primera vez que cargan la factura debido a algunos usuarios les parecía un poco molesto la ayuda en cargas posteriores. Uno de los motivos de comentarios negativos en la pregunta 1 fue que, al agregar la encuesta, la misma aparecía en el primer ingreso del usuario. No habíamos tenido en cuenta los usuarios que iniciaban un contrato al momento de agregar la encuesta, por lo que les aparecía la encuesta y contestaron mensajes negativos debido a que nunca habían llegado a utilizar el sistema. Esto se arregló al día siguiente luego de analizar los comentarios de los consultores.

Por las respuestas obtenidas de la encuesta se puede ver que el sistema tuvo muy buena aceptación. En la pregunta número 3 está la opción “No sé” debido a que hay consultores que no habían facturado en años anteriores. Si se excluye esa opción y se recalculan los totales, un 88,06% opina que se redujo el tiempo de cobro de honorarios. De la misma manera, en la pregunta 4 está la opción “No trabajé en años anteriores”. Si excluimos esa

opción y recalculamos los totales, un 90,18% opina que la utilización del sistema implementado es mejor que la metodología anterior.

7. Conclusiones y Trabajos futuros

7.1. Conclusiones

Durante el transcurso de la tesina, hemos repasado y analizado cada una de las etapas implicadas en el análisis, diseño, desarrollo e implementación del subsistema para dar una solución informática al proceso de facturación y liquidación de los contratos extrapresupuestarios de la Dirección General de Cultura y Educación de la provincia de Buenos Aires.

Se pudo comprobar que la utilización de metodologías ágiles, como XP resulta muy útil al desarrollar sistemas con requerimientos cambiantes. También al utilizar estas metodologías, se hace participar más a los usuarios finales lo que genera un ambiente más colaborativo entre el cliente y el equipo de sistema, y los usuarios finales están más predispuestos a dar devoluciones debido a que se sienten escuchados.

De acuerdo con lo expuesto, podría considerarse que se optó a una buena técnica para resolver el problema de parseo de los comprobantes adjuntados, debido a que permite adaptarse a cambios de estructura de los documentos con una mínima intervención manual.

Con la comunicación con web services también se obtuvieron resultados positivos agregando logs de las llamadas y respuestas a los mismos y teniendo en cuenta las posibles excepciones que se podrían originar al comunicarse con sistemas externos como la falta de conexión a internet.

Se utilizaron librerías como jQuery y Bootstrap que están mantenidas por una comunidad grande de programadores y existen muchos complementos que ayudan a agilizar los tiempos de desarrollo.

Con respecto al sistema implementado, se obtuvieron resultados satisfactorios debido a que el mismo es utilizado a diario por los consultores y los liquidadores, y permitió agilizar el proceso de cobro de honorarios como se vio reflejado en los resultados de la encuesta mostrados en la sección 6. Cabe destacar que hasta el momento desde la implementación se han aprobado más de 18.000 comprobantes cargados por los consultores.

A lo largo del desarrollo del sistema pude aplicar conocimientos adquiridos durante mi carrera académica y la experiencia conseguida en mi carrera profesional. También obtuve conocimientos nuevos al lidiar con comunicación con sistemas externos e interpretar los datos de los comprobantes.

7.2. Trabajos futuros

- Integración con los servicios de Gestión Documental Electrónica Buenos Aires (GDEBA) para llevar el seguimiento de los documentos electrónicos. Actualmente se hace un seguimiento de los documentos digitales en las áreas internas de DPIE, DGCyE, y otras reparticiones, pero se está proyectando hacer el seguimiento de los mismos en GDEBA.
- Integración con el programa de Interoperabilidad [35] de la Subsecretaría para la Modernización del Estado, con el fin de posibilitar el intercambio de información y conocimiento entre los sistemas de la provincia, a fin de satisfacer principios de simplificación registral y procedimental, y facilitar la automatización y aplicación de inteligencia artificial. Es un proyecto interesante y plantea un desafío debido a que estos servicios utilizan Bus de Servicio Empresarial (ESB por sus siglas en inglés) en lugar de utilizar Arquitectura Orientada a Servicios (SOA por sus siglas en inglés) con la cual estoy más familiarizado.
- Perfeccionar el circuito ante fallos ocasionados al momento en el que el consultor intenta adjuntar un comprobante. Actualmente se guarda en un log cuando no se puede parsear el pdf para luego generar un nuevo template que soporte documentos con el nuevo patrón. Tal vez se pueda generar una notificación por mail o notificar en el mismo sistema.
- Migrar el sistema a la versión 4.4 de symfony. Si bien el sistema se encuentra en la versión 3.4 de symfony - la cual es LTS -, se podría contemplar en hacer una migración a la versión 4.4 lo que daría soporte hasta fines de 2023 en vez de hasta fines del 2020 como en la versión actual.
- Mejorar la interfaz para dispositivos móviles para facilitar la carga de facturas a los consultores. Aunque el diseño implementado es responsive, podría acondicionarse para que dé la sensación de estar utilizando una aplicación

Bibliografía y referencias

[1] Sergio Luján Mora (2002). Programación de aplicaciones web: historia, principios básicos y clientes web

[2] Hassan Gomaa (2010). Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures.

[3] Arias Chaves Michael (2005). La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software. InterSedes: Revista de las Sedes Regionales. Disponible en: https://www.redalyc.org/articulo.oa?id=66612870011

[4] McKenna, Dave (2016). The Art of Scrum How Scrum Masters Bind Dev Teams and Unleash

[5] Eugenia Bahit (2012). Scrum y eXtreme Programming para Programadores

[6] Armand, Sébastien (2014). Extending symfony2 web application framework

[7] Craig Larman (2001). An Introduction to Object-Oriented Analysis and Design and the Unified Process

[8] Alaimo Diego Martí (2013). Proyectos Ágiles con Scrum: Flexibilidad, aprendizaje, innovación y colaboración en contextos complejos

[9] F. Sierra, J. Acosta, J. Ariza y M. Salas (2013). Estudio y análisis de los framework en php basados en el modelo vista controlador para el desarrollo de software orientado a la web

[10] Juan Diego Gauchat (2012). El gran libro de HTML5, CSS3 y Javascript

[11] York, Richard (2015). Web Development with jQuery

[12] Jiménez, Iván (2014). Aplicación de Metodologías Ágiles al Diseño de la UX.

[13] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994). Design Patterns: Elements of Reusable Object-Oriented Software

[14] Apache - Sitio oficial https://httpd.apache.org/

[15] PHP - Manual oficial https://www.php.net/manual/en/index.php

[16] Ubuntu - Sitio oficial https://www.ubuntu.com/

[17] MariaDB - Documentación oficial https://mariadb.com/kb/en/library/documentation/

[18] Symfony - Sitio oficial https://symfony.com/

[19] Symfony - Libro oficial de la versión 2.8 https://symfony.com/pdf/Symfony_book_2.8.pdf

[20] HTML - Sitio oficial https://www.w3.org/html/

[21] Jquery - Sitio oficial https://jquery.com/

[22] JqueryUI - Sitio oficial https://jqueryui.com/

[23] Webservices AFIP https://www.afip.gob.ar/ws/

[24] https://github.com/FriendsOfSymfony/FOSUserBundle

[25] https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice

[26] https://tools.ietf.org/html/rfc7231 (Junio 2014)

[27] Walter Goralski (2017). The Illustrated Network: How TCP/IP Works in a Modern Network (Segunda edición)

[28] Ashton Kevin. (2009). That 'internet of things' thing. RFID journal, 22(7), 97-114.

[29] Stosic, Lazar. (2012). RC4 stream cipher and possible attacks on WEP. International Journal of Advanced Computer Science and Applications (IJACSA). 3. 110-114.

[30] Vijay Sikka (2004). Maximizing ROI on Software Development

[31] https://agilemanifesto.org/

[32] https://es.wikipedia.org/wiki/Soporte_de_largo_plazo

[33] https://en.wikipedia.org/wiki/Wizard_(software)

[34] https://es.wikipedia.org/wiki/Dise%C3%B1o_web_adaptable

[35] https://www.gba.gob.ar/modernizacion/interoperabilidad.html