Trabajo de fin de título

Entorno de ‘reporting’ para una aplicación ‘Fintech’: Finbook

Titulación: Grado de ingeniería informática

Autor: Juan Kevin Trujillo Rodríguez

Tutor: José Juan Hernández Cabrera

Fecha: Julio/2020

ULPGC
Universidad de Las Palmas de Gran Canaria
Escuela de Ingeniería Informática

Ser padres es más que dar la vida,

por eso se lo dedico a Juan Santiago y Carmen Delia, mis padres,

por su apoyo incondicional y por quererme como un hijo en todo momento.

Resumen

Las nuevas tecnologías de Big Data están suponiendo un avance importante en diversos sectores, como ocurre en el financiero, donde el análisis de los datos o facturas es cada vez mayor. Siguiendo esta línea, se desarrolla una aplicación 'fintech' llamada Finbook, para la gestión de los estados financieros, fundamentalmente destinada a las pequeñas empresas. En concreto, el módulo de 'reporting' tiene como objetivo facilitar la visualización del Impuesto sobre el Valor Añadido (IVA), tanto el repercutido como el soportado, para el posterior uso de los datos. Esto supondría una ventaja para las empresas ya que el futuro se enfoca en la digitalización de las facturas, por lo que su análisis y tratamiento en este módulo será más fácil y eficaz.

Abstract

The new Big Data technologies are representing an important advance in diverse sectors, such as in the financial sector, where the analysis of the data or invoices is increasing. According of this, a fintech application called Finbook is being developed for the management of financial statements, mainly concentrates on the small companies. Specifically, the reporting module aims to facilitate the visualization of Value Added Tax (VAT), both input and output, for the subsequent use of the data. This would be an advantage for companies since the future focuses on the digitalization of invoices, so their analysis and processing in this module will be easier and efficient.

Tabla de contenido

Ilustraciones

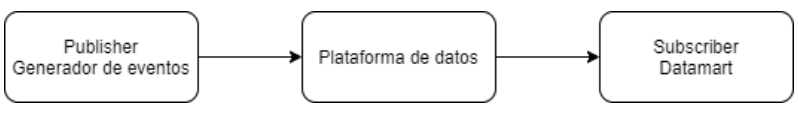
1. Introducción

En los últimos años, han aumentado las tecnologías y los conocimientos con respecto al tratamiento de los datos, que ha servido para resolver problemas concurrentes en diversos sectores, uno de ellos donde se ha hecho más visible es en el financiero en el cual con la aparición de las tecnologías de Big Data, estas permiten el tratamiento de una gran cantidad de datos a tiempo real, lo cual facilita diferentes estudios y visualizar más datos que con el uso de otra tecnología. Estos aspectos concretos son fundamentales para las aplicaciones *Fintech* o *Financial Technology* (en español Tecnología Financiera) que tienen como objetivo principal mejorar y automatizar procesos o servicios relacionados con este dominio.

Las aplicaciones *fintech* están enfocadas hacia los bancos, directores financieros (*Chief Financial Officers*), como también grandes y pequeños inversores, mejorando la gestión de sus operaciones financieras mediante el uso de software y algoritmos especializados. Teniendo en cuenta a quienes va dirigidas, al gobierno de un país también le facilitaría la gestión de los tributos, si las empresas digitalizaran todas sus facturas y las firmara con un certificado único, tal como ha implantado México en busca de evitar el fraude fiscal.

La arquitectura y el desarrollo de este proyecto están planteados como un modelo *Publisher – Subscriber* (Ilustración 1), que son utilizadas en las soluciones de Big Data. Este tipo de arquitecturas surgen en contextos donde las aplicaciones son distribuidas y basadas en la nube, y los componentes/módulos del sistema deben encargarse de proporcionar información a otros componentes a medida que van sucediendo los eventos. De esta forma, permite establecer un servicio de mensajería asíncrono en la que los *Publishers* son los generadores de eventos/mensajes y los *Subscribers* son *Datamarts*. Los *Publishers* se encargan de enviar datos y generar eventos en las plataformas de datos, y los *Subscribers* obtienen esos datos a medida que van llegando a la misma plataforma, para analizarlos y visualizarlos de una manera más amigable para el usuario, a través de gráficas, así como, también la generación de informes.

Ilustración 1: Diagrama arquitectónico 'Publisher - Subscriber'



Fuente: Elaboración propia

2. Hacia la digitalización de las facturas

La factura electrónica podría definirse como la evolución de las facturas tradicionales, convirtiéndolas en más seguras ya que son enviadas a través de medios electrónicos, por lo que quedan almacenadas y validadas y, en caso de no ser necesario, no hay porque imprimirla.

Nos encontramos en una época en la que se intenta automatizar y digitalizar procesos repetitivos o que nos ayudarían a tener una vida más cómoda. Además, con las nuevas tecnologías en ciencia de datos y análisis de grandes volúmenes de datos (Big Data), muchos sectores están explotando estos recursos para ser pioneros en su sector.

Si hablamos de las facturas, hoy en día no se ha llevado a cabo su digitalización completa, solo en algunos lugares se ha hecho, como es el caso de México, ya que su implementación tiene muchos beneficios, como pueden ser la eliminación del uso del papel y, por ende, beneficio para el medioambiente, como también la simplificación del trabajo que conlleva su realización y gestión.

Una empresa que invierta y apueste por la digitalización de las facturas, estará avanzando hacia un sistema más ecológico, lo que llevaría positivamente a la lucha contra el cambio climático, ya que es una cuestión en la que todos debemos participar activamente.

En relación con los gobiernos, la eliminación de la factura tal y como se conoce, y su avance hacia la digitalización ayudaría además de al medioambiente, también al análisis y prevención del blanqueo de capitales o fraude fiscal, y las gestiones tributarias o inspecciones serían más eficientes, es por ello, que los países con un sistema obsoleto para la época actual deberían invertir en el desarrollo de sistemas más automatizados, pero siempre con la premisa de que el software sea de calidad, escalable y seguro.

2.1. La factura electrónica en México

Para una mejor explicación de la factura electrónica nos vamos a centrar en la estructura y sistema implementado por México, ya que son los pioneros en este tema. La factura electrónica en este país surgió como medida para contrarrestar la evasión de impuestos.

Existen dos modelos de comprobante:

- Comprobante fiscal digital (CFD): Es un archivo electrónico en formato y extensión XML que pueden ser generados por un proveedor autorizado de certificación de facturas digitales.

- Comprobante fiscal digital a través de internet (CFDI): Es un archivo electrónico en formato y extensión XML, el cual debe ser enviado previamente a un Proveedor Autorizado de Certificación (PAC) para que valide la factura antes de ser entregada al cliente, enviándole una copia al Servicio de Administración Tributaria (SAT). Algo que hay que tener en cuenta, es que la persona necesita conexión a internet y contratar a un PAC. En este último es en el que nos centraremos a lo largo de nuestro proyecto. (Alves Fernández, M, et al., 2010)

En el caso de las empresas en lo que respecta a las facturas electrónicas han de conservarlas en su formato original durante al menos cinco años para efectos fiscales y durante diez años para efectos comerciales.

2.2. La factura electrónica en Europa

El organismo encargado de llevar a cabo la legislación europea en materia de la factura electrónica es el Consejo de la Comunidad Económica Europea que se encarga de regular el impuesto sobre el valor añadido (IVA). Según la Directiva 2010/45/UE del Consejo de 13 de julio de 2010, las facturas pueden transmitirse electrónicamente siempre y cuando se garantice la autenticidad, la integridad y la legibilidad, mediante la firma digital o a través de un intercambio electrónico de datos.

Se ha implementado este sistema de facturación como una solución óptima para facilitar el mercado común evitando de esta manera obstáculos en las relaciones transfronterizas. Por eso, desde el año 2019 todos los Estados miembros de la Unión Europea están obligados a insertar la factura electrónica entre el sector público y sus proveedores a nivel nacional y, a escala local y regional la obligatoriedad de la factura electrónica comenzó desde abril de 2020.

Siguiendo esta regla, cada país podrá establecer la normativa que considere oportuna en lo que respecta a esta materia. No obstante, la Unión Europea trabaja en la implantación de un modelo estándar que facilite la actividad a las empresas a la hora de realizar a intercambiar las facturas con otros países.

2.3. La factura electrónica en España

La factura electrónica en España comenzó a ser obligatoria desde el año 2.015 para todos los proveedores de la administración pública cuyas facturas superen los 5.000€. Hoy en día, es utilizado por un total de 8.000 administraciones públicas pertenecientes a las diferentes Comunidades Autónomas del país y a la Administración General del Estado (Sánchez, M., 2019).

El formato del fichero de la factura establecido de forma oficial por la Agencia Tributaria es el XML. Sin embargo, se pueden utilizar otros como PDF, HTML, DOC, XLS, JPEG, GIF o TXT.

Debido a que la factura digital es transmitida a través de medios tecnológicos cabe la posibilidad de que se produzca una violación de seguridad, en la que por ejemplo se reciba un ataque que nos "robe" el correo y comience a enviar a nuestros clientes correos adjuntos simulando que son facturas, pero en realidad es un tipo de programa maligno (*malware*) que el cliente ejecutará cuando quiera ver esa supuesta factura. No obstante, son numerosas las ventajas que trae consigo la factura electrónica, por ello, al igual que las administraciones públicas de España, muchas empresas y países en la escala global se van sumando de forma voluntaria a esta nueva era digital, en la cual la transmisión de facturas entre empresas y, entre éstas y personas físicas, tanto a nivel nacional como internacional, es mucho más sencilla, cómoda, económica y de mayor celeridad.

3. Finbook

3.1. ¿Qué es una aplicación Fintech?

Fintech deriva de las dos palabras inglesas: *Finance* y *Technology*. Se trata de una aplicación destinada al estado financiero, por lo que suelen ser utilizada por bancos y por grandes empresas e inversores, cuyo objetivo es la de acercar cada vez más a las personas al mundo de las finanzas a través de medios digitales.

3.2. ¿Qué es Finbook?

Con este trabajo de fin de título se aporta una aplicación *fintech* llamada '**Finbook'**, que está orientada a pequeñas empresas que tienen la obligación de liquidar el Impuesto sobre el Valor Añadido (IVA), ya sea, mensual, trimestral, semestral o anual, permitiéndoles tener una clara visión de su estado financiero, con respecto al IVA devengado, que es el que aplica la empresa en las facturas que emite o el IVA soportado, que es el que paga en las facturas que recibe por compras, servicios, etc.

Atendiendo a la cantidad resultante entre el IVA devengado y soportado, si esta es positiva se deberá abonar el exceso a la Agencia Tributaria, en caso contrario, si el resultado es negativo la persona física o jurídica obtendrá el abono correspondiente.

En base a esto, utilizar una aplicación Fintech, como Finbook en nuestro caso, y facturas electrónicas permite a las empresas y a la Agencia Tributaria un mayor control de las finanzas, por lo que la evasión de impuestos se vería mermada dado que sería más complejo llevar a cabo este tipo de fraude fiscal.

Además, el uso de este tipo de aplicaciones implicaría una reducción de costes en la impresión de las facturas, lo que supondría una reducción en las emisiones de dióxido de carbono (CO_2) y, por ende, se estaría contribuyendo a la lucha contra el cambio climático.

4. Situación de partida y estado actual

4.1. Motivación

Debido al amplio abanico de posibilidades del que disponíamos para escoger tema del trabajo de fin de título, me encontraba indeciso a la hora de elegir un proyecto. Sin embargo, algunos compañeros me comentaron que tenían previsto realizar un trabajo relacionado con el Big Data, lo que me parecía interesante, ya que es algo reciente e innovador a nivel socioeconómico, así que hablé con el tutor y él me invitó a una de las reuniones. En dicha reunión nos explicó el trabajo y en qué consistiría el desarrollo de una arquitectura 'Publisher–Subscriber', las cuales son utilizadas y demandadas en la actualidad sobre todo en el ámbito de Big Data, y como yo estaba buscando algo relacionado como lo que él planteaba, pues tome la decisión de unirme al equipo de trabajo.

4.2. Objetivos principales

El objetivo principal de este trabajo de fin de título es la realización de un entorno de usuario mediante una aplicación web para el proyecto global (Finbook), en el que se visualizará el estado de la liquidación del IVA de la empresa, como también diferentes formas de ver esos datos, lo que en inglés se llama *Data Visualization*.

Siguiendo el diagrama arquitectónico (Ilustración 1), este entorno de usuario actúa como un *Subscriber* (*datamart*), permitiendo a las empresas visualizar de diversas maneras los datos contenidos en la plataforma de datos.

4.3. Estado actual

Si bien es cierto que la factura digital se está implementando en ciertos ámbitos de estado financiero, como la banca a través de las aplicaciones móviles, es un procedimiento que se encuentra todavía en fase de desarrollo ya que es una herramienta que se va involucrando progresivamente en las empresas. Esto, ahorra costes en lo que se refiere a todos los elementos que se necesitan para la impresión de facturas, lo que también supone una reducción en la contaminación del aire, hecho que resulta favorable para la reducción de los gases de efecto invernadero y, de esta manera, contrarrestar el cambio climático.

5. Metodología

5.1. Organización

El proyecto de Finbook es lo bastante grande por lo que para su realización se ha dividido en cuatro módulos diferentes, que serán desarrollados cada uno por un integrante del equipo de trabajo el cual está integrado por:

- **Gerardo Santana Franco**: desarrollará un módulo para conocer los estados financieros de la empresa, permitiéndole además subir nuevas facturas a la plataforma de datos.
- **Juan Alberto Ureña Rodríguez**: se encargará del desarrollo de un simulador de facturas en el que los parámetros son modificables.
- **Raúl Lozano Ponce**: implementará una plataforma de datos, en la que recibirá las facturas del simulador y las almacenará para que los *subscribers* puedan obtener los datos. Además, adicionalmente desarrolló una aplicación para firmar y permitir el inicio de sesión mediante certificado digital.
- **Juan Kevin Trujillo Rodríguez**: me centraré en realizar un módulo de informes en el que los usuarios podrán visualizar el IVA devengado y soportado acorde a un periodo de tiempo y, en el que, a su vez podrán recibir informes en formato PDF por email.

Cabe destacar que los módulos pueden trabajar de manera independientes por lo que no dependen unos de otros, ya que por ejemplo para el módulo de este trabajo se ha creado unos datos ficticios para llenar la base de datos.

5.2. Planificación

El plan de trabajo para que se planteó al inicio del desarrollo de este trabajo, y que se ha seguido para llegar al resultado final obtenido, ha sido el siguiente:

- Estudiar las arquitecturas ‘Big Data’
- Planificar el trabajo con los integrantes del grupo, así como la creación de los repositorios necesarios en GitHub.
- Recolectar y analizar los distintos datos que conforman una factura y nómina. Como también las distintas formas de representar los datos.
- Diseñar la estructura de la aplicación, campos necesarios para la base de datos.
- Desarrollar el ‘Subscriber’ que permita a los usuarios la visualización del estado actual para la liquidación del IVA, con los datos contenidos en la plataforma de datos.
- Implementar distintas formas de representar los datos de la plataforma de datos.

- Desarrollar el sistema de inicio de sesión (*log-in* / *sign-in*) utilizando una aplicación de firma digital.
- Testear y buscar posibles errores.
- Integración final de los distintos módulos del proyecto.
- Realizar la memoria final del trabajo y preparar la presentación del trabajo.

5.3. Herramientas utilizadas

Para el desarrollo del módulo de informes de Finbook y obtener el resultado que se esperaba, se ha utilizado diversas herramientas:

- **AdminLTE**: plantilla para tener un diseño profesional y que fuera adaptable a cualquier dispositivo, para adaptarse al tiempo del que se disponía, y finalmente lograr el resultado esperado. No obstante, si algo no se veía como se esperaba, se modificaba el HTML y CSS.
- **Git**: sistema de control de versiones junto con el diseño de flujo de trabajo llamado *Gitflow*.
- **Java**: lenguaje de programación que suele usarse en la mayoría de las aplicaciones, por su escalabilidad y versatilidad, lo que hace que el lenguaje sea idóneo para un trabajo de fin de título, como también para posibles ampliaciones en el futuro.
- **Apache Maven**: se ha utilizado junto con el lenguaje de programación Java, ya que facilita la gestión e instalación de dependencias que requiera el proyecto, quedando todo centralizado en un archivo llamado “pom.xml”.
- **MongoDB**: usada esta base de datos no relacional (NoSQL) porque en el campo del Big Data y Ciencia de Datos (*Data Science*) predominan estas. Además, era innovador utilizarla en este proyecto porque en nuestro grado de la Universidad de Las Palmas de Gran Canaria no se ha impartido en las asignaturas de base de datos.
- **Sparkjava**: utilizado este micro *framework* por recomendación del tutor, ya que facilita la implementación del *Back-end* con un servidor Jetty y, por otra parte, tiene diferentes motores de plantillas basados en java que se pueden utilizar. En este caso como motor de plantilla se ha usado *Velocity*.

6. Desarrollo

6.1. Diseño estructural de la base de datos

Para la realización de este proyecto se ha decido usar una base de datos NoSQL, MongoDB, dado que su uso está cada vez más extendido y se usa en el sector del Big Data. La estructura seguida tiene especial relación con el formato que implementa el módulo de generador de facturas.

Para MongoDB no es necesario definir una estructura determinada desde un principio, ya que al no ser una base de datos relacional no tenemos que estar uniendo campos entre tablas, así que se puede definir una estructura y cambiarla más adelante sin ser un trabajo tedioso.

Ilustración 2: Estructura de una factura en la base de datos


```
{
  "_id": {
    "$oid": "5ef7a9b3908cf39702eff601"
  },
  "invoiceUUID": "4000001",
  "invoiceDate": {
    "$date": "2020-05-18T00:21:33Z"
  },
  "zipCode": "37394",
  "invoiceType": "income",
  "usoCFDI": "G01",
  "issuerName": "el patio de lola",
  "issuerId": "DEMC",
  "receiverName": "Crystal Rainford",
  "receiverId": "3000046",
  "concept": "Bill of a eating",
  "subtotal": "55.786178826619164",
  "taxRate": "0.15",
  "totalDue": "64.15410565061204",
  "currency": "euro",
  "xmlFile": "PD94bWwgdmVyc2lvbj0iMS4wIiBlbm...",
},
```


Fuente: Elaboración propia

En la estructura de nuestra factura los campos que más destacan por estar basado en el sistema de México son: el *usoCFDI*, *issuerId* y *receiverId*, añadiendo también el *xmlFile* (ilustración 2):

- **UsoCFDI**: código que identifica el tipo de factura, como pueden ser donativos (D04), gastos en general (G03), etc.

- **IssuerId y receiverId**: se guarda el RFC, una clave única que identifica a la persona, tanto emisor como receptor.
- **XmlFile**: trata de un campo de texto que guarda el contenido de la factura en XML convertida a Base64.

Con relación al campo concept, este consiste en definir el producto o servicio para por el cual se genera la factura y se paga. En un principio se desarrolló una factura en la que se disponía de una lista de conceptos, y cada concepto tenía unidades, tasa de descuento, cantidad de impuestos, y precio final por concepto, además de contener también un producto con un código propio, descripción del producto y una tasa de impuesto, ya que no todos los productos se gravan igual, y el precio por unidad, en la imagen siguiente, se puede observar mucho mejor (ilustración 3).

Ilustración 3: Estructura de una lista de conceptos de una factura


```
"concepts": [
  {
    "product": {
      "code": "CB29813",
      "description": "iPad Pro 2020 de 11 pulgadas y 128 GB con Wi-Fi + Cellular - Gris espacial",
      "taxRate": 7,
      "price": 1099
    },
    "units": 1,
    "discountRate": 3,
    "taxAmount": 0,
    "amount": 1066.03
  },
  {
    "product": {
      "code": "CB24941",
      "description": "Sistema de compensación equitativa por copia privada para iPad (Canon Digital)",
      "taxRate": 0,
      "price": 3.15
    },
    "units": 1,
    "discountRate": 0,
    "taxAmount": 0,
    "amount": 3.15
  }
],
```


Fuente: Elaboración propia

No obstante, para seguir una misma estructura en todos los módulos del proyecto Finbook, se optó por simplificar el modelo de la factura al mostrado anteriormente (ilustración 2).

6.2. Desacoplar el sistema de rutas

En relación con el sistema de rutas que implementa *Spark*, se decidió desacoplar dicho sistema de nuestro código, separando las rutas por una parte y el código general del proyecto por otra, con el fin de poder sustituir el *framework* en un futuro si fuera necesario, y evitar tener una dependencia muy fuerte con *Spark*.

Ilustración 4: Clase CustomResponse

```java
package io.finbook.responses;

import io.finbook.sparkcontroller.ResponseCreator;
import io.finbook.sparkcontroller.TemplateEngine;

public class CustomResponse {

    private CustomResponse() {
    }

    public static ResponseCreator ok(ResponseStructure body) {
        return (req, res) -> {
            res.status( statusCode: 200);
            return new TemplateEngine().render(body);
        };
    }

    public static ResponseCreator notFound(ResponseStructure body) {
        return (req, res) -> {
            res.status( statusCode: 404);
            return new TemplateEngine().render(body);
        };
    }
```

Fuente: basado en (Herring, 2015)

Para llevarlo a cabo, se creó una clase llamada *CustomResponse* (ilustración 4) en la que se define métodos por cada tipo de respuesta que se quiera dar, por ejemplo, un 200 – OK, o un 404 – Not Found, entre otros códigos HTTP que se pueden usar. Los métodos devuelven un objeto de la interfaz creada *ResponseCreator* que extiende de la interfaz *Route* de *Spark*, con lo que nos aseguramos de que nuestro código no tendrá que importar las librerías del *framework*.

Ilustración 5: Clase TemplateEngine

```java
package io.finbook.sparkcontroller;

import io.finbook.responses.ResponseStructure;
import spark.ModelAndView;
import spark.template.velocity.VelocityTemplateEngine;

import java.io.File;
import java.util.HashMap;
import java.util.Map;

public class TemplateEngine {

    protected VelocityTemplateEngine instance;
    protected String templatesFolder = "template";
    protected String templatesExtension = ".vm";

    public TemplateEngine() { instance = new VelocityTemplateEngine(); }

    public String getTemplateURL(String template){
        return templatesFolder + File.separatorChar + template + templatesExtension;
    }

    public String render (ResponseStructure response){
        Map model = response.getData() == null ? new HashMap<>() : response.getData();
        return instance.render(new ModelAndView(model, getTemplateURL(response.getView())));
    }

}
```

Fuente: Elaboración propia

Los métodos reciben un objeto de la clase creada *ResponseStructure*, que contiene un mapa, con los datos que se le quiere pasar a la vista, y un campo de texto con la ruta de la vista a ser renderizada. Dicho objeto se le pasa al método render de la clase *TemplateEngine* (Ilustración 5), donde se define el motor de plantilla a utilizar, en este caso *Velocity*, y se forma la ruta completa al fichero de la vista dentro del proyecto para que sea renderizado por el método de *Velocity* llamado render, que se le pasa un objeto *ModelAndView* con los datos y la ruta final.

Finalmente, la clase *Routes* creada tiene toda la responsabilidad de manejar las peticiones y respuestas y así evitar que las librerías de *Spark* estén por todo el proyecto. Para ello hace uso del método *map* que devuelve las repuestas de nuestro proyecto a una ruta de *Spark*, mediante la interfaz creada *Converter*.

Ilustración 6: Ejemplo de la ruta Index-Home

```java
get(Path.HomeRoutes.INDEX, map((req, res) -> HomeCommand.index(Auth.isLogged(req))));
```

Fuente: Elaboración propia

Para definir todas las rutas en un único lugar y que su modificación se más fácil, se ha creado la clase *Path* que a su vez contiene otras clases, en la que se definen constantes que son usadas en el proyecto a la hora de seleccionar la ruta por HTTP o la ruta a una vista de la plantilla.

Ilustración 7: Fragmento de la clase Path

```java
package io.finbook.util;

public class Path {

    public static class HomeRoutes {
        public static final String INDEX = "/";
    }

    // Authentication routes
    public static class AuthRoutes {
        // Section
        public static final String AUTH = "/auth";

        // Routes
        public static final String SIGN_IN = "/sign-in";
        public static final String SIGN_CERTIFICATE = "/sign-certificate";
        public static final String SIGN_OUT = "/sign-out";
    }

    public static class AdminRoutes {
        // Section
        public static final String ADMIN = "/admin";

        // Filters
        public static final String ADMIN_FILTER = "/*";

        // ROUTES
        public static final String DASHBOARD = "/dashboard";
```

Fuente: Elaboración propia

6.3. Funciones

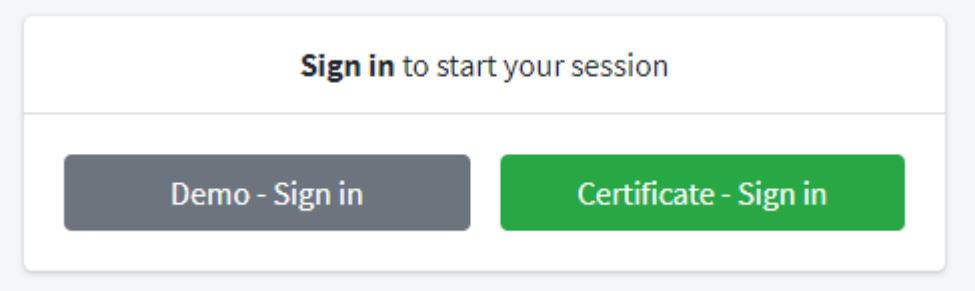
6.3.1. Inicio de sesión con certificado electrónico

Como sistema de identificación de usuario se implementa un sistema de inicio de sesión mediante certificado digital, dado que se ha buscado la similitud con el sistema de autenticación de México, y también por ser un sistema más seguro que un sistema basado en usuario y contraseña.

La autenticación es necesaria para acceder a todas las funciones de este trabajo, aunque se ha implementado un acceso demo, para que usuarios sin certificado digital puedan probar todas las opciones disponibles (ilustración 8).

Para los usuarios invitados (demo) se han cargado en la base de datos unas facturas de ejemplo, para que tengan disponibles todas las funciones, para los que usen su certificado electrónico, estarán disponibles los datos que se obtengan de la plataforma de datos.

Ilustración 8: Selección de sistema de autenticación

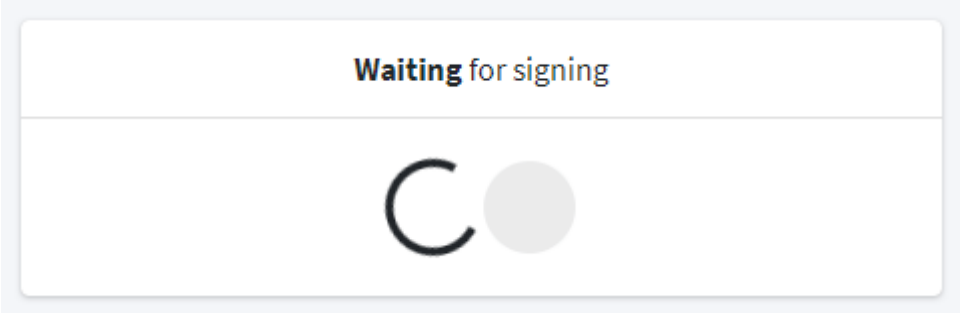


Fuente: Elaboración propia

Si el usuario decide seleccionar el acceso de prueba, automáticamente se le asignará a la sesión el valor de “DEMO”, y accederá a la parte privada donde probar las funciones de la aplicación.

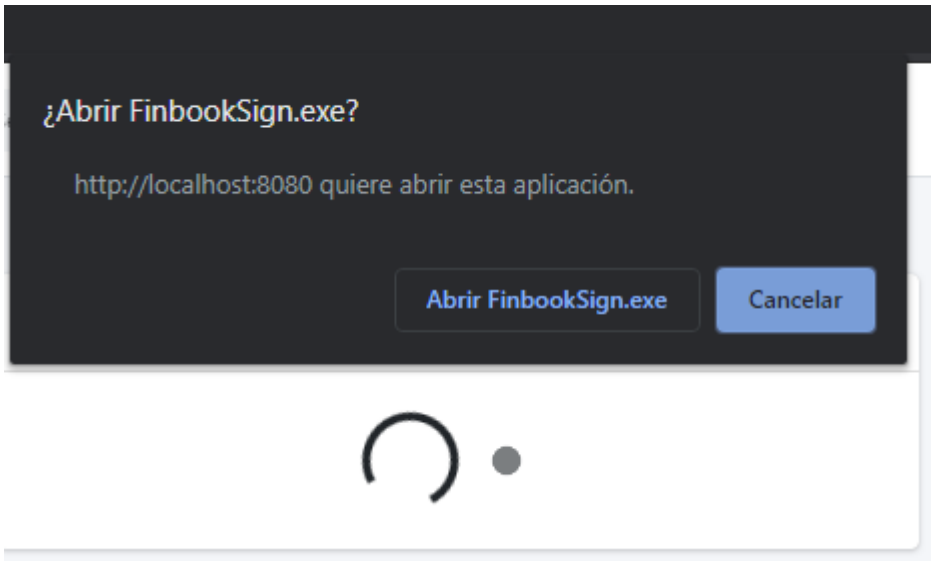
En caso contrario, si selecciona el acceso con certificado digital, se le redireccionará a otra página, en la que el navegador notificará al usuario de que la página está intentando abrir la aplicación de Firma de Finbook desarrollada por uno de mis compañeros (Lozano Ponce, 2020a) e instalada en nuestro ordenador.

Ilustración 9: Vista de espera hasta que usuario firme con su certificado 1



Fuente: Elaboración propia

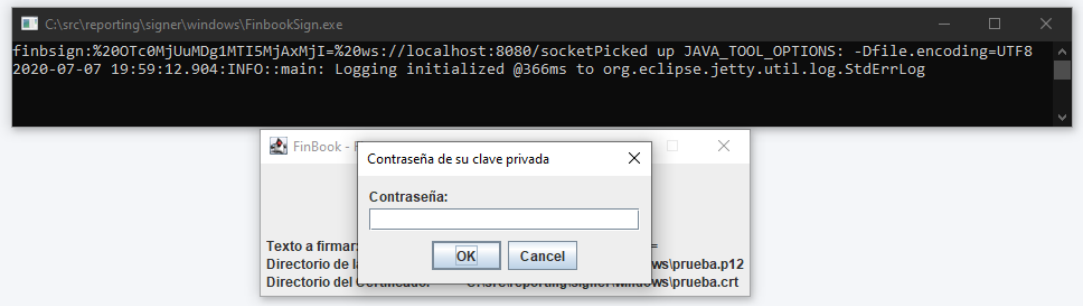
Ilustración 10: Notificación al usuario de la apertura de FinbookSign



Fuente: Elaboración propia

Seguidamente, el usuario cuando haga clic para abrir la aplicación se le mostrará la aplicación de Firma (Ilustración 11). Es importante destacar que la aplicación de Firma está desarrollada con *OpenJDK*, versión libre de Java, en su versión 11, por lo que, si se tiene una versión menor, no funcionará.

Ilustración 11: Apertura de la aplicación de Firma



Fuente: (Lozano Ponce, 2020a)

Este paso, el usuario tendrá que escribir la contraseña de su certificado digital para firmar el texto aleatorio que se ha generado, para luego comprobar que su firma es válida.

Si se escribe correctamente su contraseña y la firma se realiza correctamente, se le mostrará un mensaje al usuario haciéndole saber que el proceso ha sido realizado con éxito, y si luego en el servidor la verificación de la firma es correcta, pues se le permitirá el acceso a la aplicación.

La aplicación Firma nos devuelve un objeto JSON que contiene dos claves con sus respectivos valores:

- **Id**: es el texto aleatorio que se le ha mandado pero firmado, si el que se le manda y el que se recibe son el mismo, entonces la firma es válida.
- **Sign**: es la firma en un array de bytes.

Para la obtención de la respuesta es necesario la creación de un *WebSocket* en el servidor que recibirá la respuesta después de que el usuario haya firmado, y que se verificará mediante *JavaScript* en la página de espera para iniciar sesión.

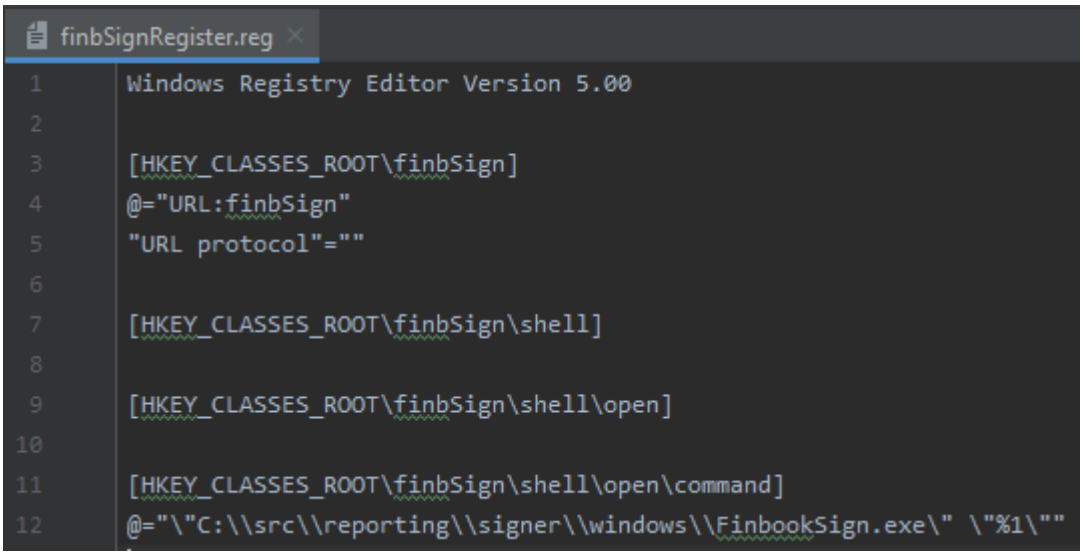
Si la firma es válida, entonces le pasaremos el JSON al método del servidor para que verifique si el campo *sign* es correcto y que contiene el identificado del usuario, que puede ser el DNI en España, o en el caso de México el RFC. Si contiene el identificador, entonces nos lo devolverá, en caso contrario devolverá un "null".

La verificación de la firma en bytes (*sign*) se realiza con el método *validateSign* la clase *Verifier* de la librería *sign-verifier* desarrollada por mi compañero (Lozano Ponce, 2020b).

Para la implementación de este sistema de autenticación mediante certificado digital se ha necesitado crear un protocolo personalizado en el que el navegador reconozca y solicite la apertura de la aplicación de firma instalada en el ordenador. Esto se ha hecho tanto en Windows como en MacOS, y así asegurar la disponibilidad en estos dos sistemas operativos que son los más usados.

Para realizarlo en Windows se necesita crear un fichero con extensión "**.reg**" como se muestra en la siguiente ilustración (Ilustración 12).

Ilustración 12: Fichero para crear un protocolo en Windows



Fuente: (Santana Franco, 2020)

En el fichero de registro se tiene que especificar el nombre del protocolo, que en este caso es "*finbsign*" y la ruta absoluta donde se encuentra el programa a ejecutar, escapando las barras divisorias de la ruta "\".

Una vez creado, se hace doble clic sobre el fichero para que se ejecute y se instale en el ordenador, por lo que después de instalarlo, si se utiliza ese protocolo lo que hará es ejecutar el programa que está definido en la ruta del registro, *FinbookSign*.

Para crear este programa ejecutable, se ha creado un script en el lenguaje de programación C, y luego se ha compilado.

Ilustración 13: Programa en C para Windows

windows\FinbookSign.c | mac\FinbookSign.c

```c
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <locale.h>

int main(int argc, char *argv[])
{
char command[256];
strcpy(command, "java -jar C:/src/reporting/signer/windows/Firma.jar ");
 for (int i = 1; i < argc; ++i){
    strcat(command, argv[i]);
    printf("%s",argv[i]);
 }
system(command);
return 0;
}
```

Fuente: Elaboración propia

En este programa C, establece una posición de memoria (puntero) de 256 bytes que define un campo de caracteres, donde luego se le copia la ruta absoluta al programa de Firma. Luego se crea un bucle el cual va a coger los argumentos que recibe y se concatenarán al campo de texto. Una vez finalizado, se hace una llamada al sistema con el comando para que se ejecute.

En MacOS se usa el mismo programa C, pero hay que definir la ruta correspondiente donde tengamos la aplicación Firma y los certificados que se usaran para firmar.

Para compilar dicho programa en C, simplemente debemos tener instalado el compilar de C/C++ en nuestro ordenador y ejecutar el siguiente comando:

```
gcc FinbookSign.c -o FinbookSign
```

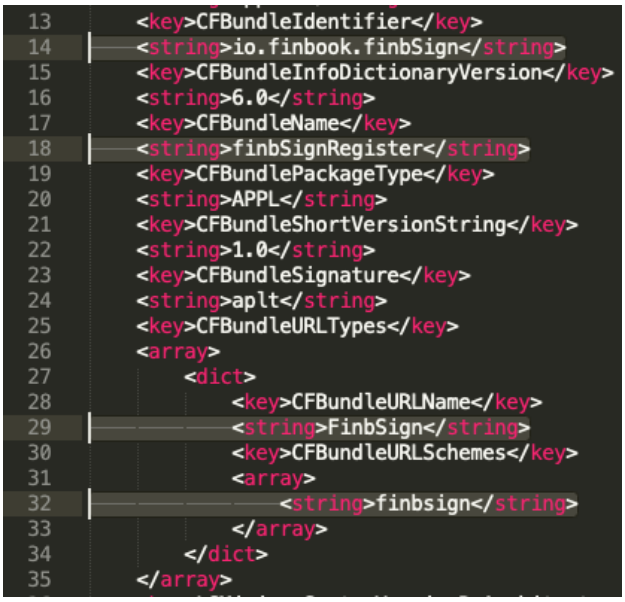
La creación del protocolo que necesitamos en MacOS es más compleja de realizar comparado con la creación en Windows, no obstante, el funcionamiento final es el mismo. Para realizarla tenemos que abrir el Editor de Scripts de Apple y crear un nuevo documento donde tendremos que escribir lo de la imagen siguiente. La ruta que definamos en el script tiene que ser absoluta y apuntar al ejecutable de nuestro script programado en C.

Ilustración 14: Fichero para crear un protocolo en MacOS

Fuente: (Seres, 2019)

Seguidamente, guardamos el archivo con "*FinbookSign.app*", y luego accedemos a la ruta donde lo hemos guardado, ya que aún tenemos que hacer unos cambios. Una vez dentro, hacemos doble clic sobre el fichero que acabamos de crear, y otro clic sobre "Mostrar contenido del paquete". Ahora tenemos que abrir al fichero llamado "Info.plist" que se encuentra en la carpeta *Contents*.

Ilustración 15: Modificación del fichero "Info.plist" para la creación del protocola en MacOS



Fuente: (Seres, 2019)

Cuando lo tengamos abierto, tenemos que modificar las líneas que se señalan en la ilustración anterior. La más importante es la línea 32, ya que es la clave por la que se reconocerá el protocolo. Finalmente, guardamos los cambios, y hacemos doble clic sobre el fichero anteriormente creado, *FinbookSign.app*, para que se instale el protocolo y el navegador pueda reconocerlo.

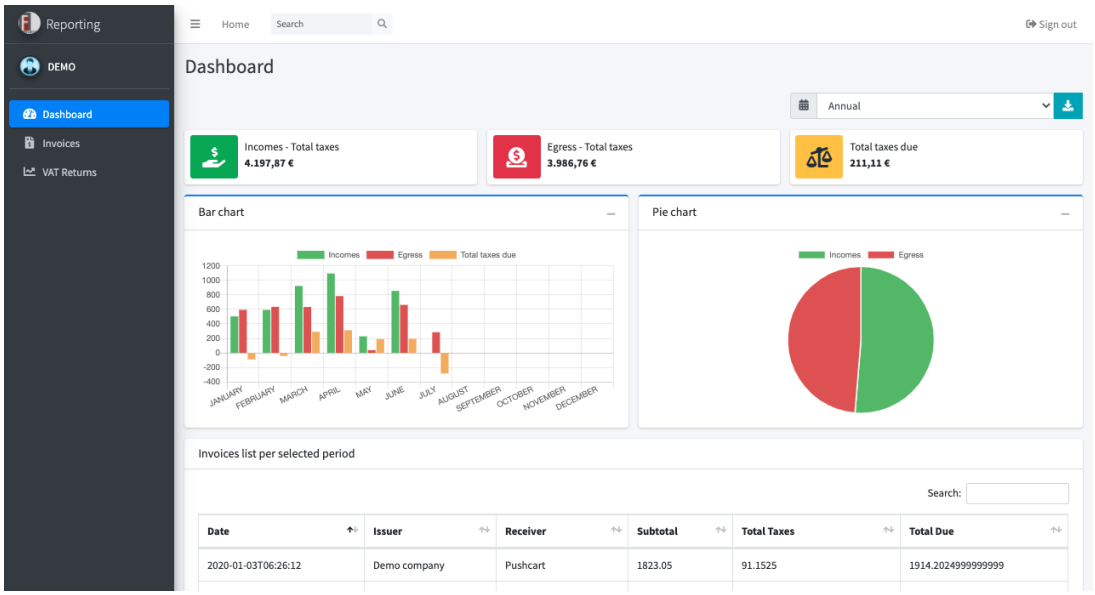
6.3.2. Representación de los datos

Con el fin de que la visualización de los datos más importante estuviera disponible con un solo vistazo, se ha optado por mostrar en una sola página (*dashboard*) los datos analizados de las facturas del usuario autenticado que están almacenadas en la base de datos. En dicha pantalla se muestra los ingresos, egresos y diferencia del IVA, lo que viene a ser el IVA devengado y el IVA soportado y la cantidad resultante de la resta entre ambos. Estos datos se pueden observar separados en tres cuadros y también con dos gráficas, una de barras y una de queso. Para añadir, esa vista también contiene una tabla con la lista de facturas correspondientes al periodo del análisis, pudiendo filtrar por mes (por defecto), trimestre, semestre y año actual.

La carga de datos en la tabla se hace a través de JavaScript, usando jQuery y peticiones Ajax, lo que permite que la página no tenga que estar refrescándose todo el tiempo, sino que solo se actualizará el contenido necesario, permitiendo así optimizar y obtener un menor uso de datos, que en versiones móviles puede ocasionarle al usuario un gasto extra en su consumo.

Además, dicha página contiene un botón que abre un modal para que se introduzca una dirección de correo electrónico, y así poder recibir un informe del IVA correspondiente al período seleccionado. Se hablará más adelante la generación de informes y su envío por correo electrónico.

Ilustración 16: Captura de la página Dashboard



Fuente: Elaboración propia

Con respecto al funcionamiento del JavaScript para el cambio de datos según el periodo seleccionado, se ha creado una función que se llama al cambiar el valor del *select*, ya que al cargar la página se establece un *listener* para capturar cuando cambiar el valor.

Ilustración 17: Listener on change of datepicker

```
// Fill out all dashboard
let datepicker = $('#datepicker');
if (datepicker.length){
    ajaxChangeReportPeriod( valueOfSelect: "monthly");
}
// If datepicker changes
datepicker.on('change', function() {
    ajaxChangeReportPeriod(this.value);
});
```

Fuente: Elaboración propia

La función para traer los datos con los que rellenar la página recibe el valor del periodo seleccionado y mediante una petición Ajax le pasa el valor al servidor, para que este nos devuelva todos los datos, en formato JSON, para cargar el *dashboard* nuevamente. Cabe destacar, que al acceder a la página *dashboard*, siempre hace una llamada a esta función con el valor “monthly” para rellenar la vista con los valores del mes actual.

Ilustración 18: Función para traer los datos del servidor y rellenar el dashboard mediante Ajax

```
function ajaxChangeReportPeriod(valueOfSelect){
    $.ajax( e: {
        url: "/admin/reporting/ajax-datepicker",
        method: "POST",
        data: { datepicker_value : valueOfSelect },
        dataType:"JSON",
        success: function(data){
            $('#incomes').text(data.incomes);
            $('#egress').text(data.egress);
            $('#totalTaxesDue').text(data.totalTaxesDue);
            drawBarChart(data.barChart);
            drawPieChart(data.pieChart);
            fillOutTableBodyInvoiceList(data.invoicesShortedList);
            format_amounts();
        },
        error: function () {
            Toast.fire({
                icon: 'error',
                title: 'Something was wrong!'
            })
        }
    });
}
```

Fuente: Elaboración propia

En el servidor el método para la obtención de los datos y su posterior envío a la vista recibe dos parámetros, el id del usuario autenticado y el periodo seleccionado. Con ello, calcula la fecha de inicio a través del periodo y la fecha final con el momento actual. Seguidamente, en función de periodo se establece una cantidad de meses que van a entrar en el cálculo, esto se realiza para dibujar el “bar chart” en consecuencia.

Con los datos obtenidos, comienzan las búsquedas en la base de datos para traer las listas de facturas de tipo ingreso en las que el usuario es emisor, y las de tipo egreso en las que usuario es receptor, ya que unidas estás dos, tenemos todas las facturas en las que el usuario ha ganado dinero y, por tanto, calcular el IVA devengado. El mismo proceso es para obtener el IVA soportado, solo que ahora se buscará en las facturas de tipo ingreso y egreso en las que el usuario sea receptor y emisor, respectivamente.

Una vez se han obtenido la lista de facturas y se ha realizado los cálculos, estos se añaden al mapa, con su respectiva clave – valor, que le será devuelto a la vista como un objeto JSON.

Ilustración 19: Método para la obtención de los datos necesarios para el dashboard

```
public static JSONObject getDataPerPeriod(String currentUserId, String period) {
    Map<String, Object> data = new HashMap<>();
    LocalDateTime startDate = getStartDate(period);
    LocalDateTime endDate = Utils.getCurrentDate();
    int amountMonths;

    switch (period) {
        case TRIMESTER:
            amountMonths = 3;
            break;
        case SEMESTER:
            amountMonths = 6;
            break;
        case ANNUAL:
            amountMonths = 12;
            break;
        default:
            // Monthly by default
            amountMonths = 1;
    }

    List<Invoice> invoicesIncomes = getInvoicesListPerPeriodAndType(currentUserId, InvoiceType.INCOME, startDate, endDate);
    Double incomesTaxes = invoiceService.getSumTotalTaxes(invoicesIncomes);

    List<Invoice> invoicesEgress = getInvoicesListPerPeriodAndType(currentUserId, InvoiceType.EGRESS, startDate, endDate);
    Double egressTaxes = invoiceService.getSumTotalTaxes(invoicesEgress);

    data.put("incomes", incomesTaxes);
    data.put("egress", egressTaxes);
    data.put("totalTaxesDue", incomesTaxes - egressTaxes);

    data.put("barChart", drawBarChart(currentUserId, startDate, amountMonths));
    data.put("pieChart", drawPieChart(incomesTaxes, egressTaxes));

    data.put("invoicesShortedList", getInvoicesShortedListByDate(invoicesIncomes, invoicesEgress));

    return new JSONObject(data);
}
```

Fuente: Elaboración propia

Realizado lo anterior, ya se pueden dibujar los gráficos, el de barra y el de queso, los cuales se añadirá posteriormente al mapa que se devolverá. Al de barra se le pasará el usuario actual, la fecha de comienzo del período y la cantidad de meses a mostrar.

Ilustración 20: Método para dibujar un BarChart

```
private static JSONObject drawBarChart(String currentUserId, LocalDateTime startDate, int amountMonths){
    BarChart barChart = new BarChart();
    barChart.getData().setLabels(getMonthsNamesForChart(startDate, amountMonths));

    barChart.getData().addDataset( label: "Incomes",  backgroundColor: "#5cb85c",
            getIncomesPerMonth(currentUserId, startDate, amountMonths));

    barChart.getData().addDataset( label: "Egress",  backgroundColor: "#d9534f",
            getEgressPerMonth(currentUserId, startDate, amountMonths));

    barChart.getData().addDataset( label: "Total taxes due",  backgroundColor: "#f0ad4e",
            getTotalTaxesDuePerMonth(currentUserId, startDate, amountMonths));

    return barChart.toJSON();
}
```

Fuente: Elaboración propia

Con respecto gráfico de queso, se le pasará como parámetros los ingresos y egresos totales del IVA.

Ilustración 21: Método para dibujar un PieChart

```
private static JSONObject drawPieChart(Double incomesTaxes, Double egressTaxes){
    PieChart pieChart = new PieChart();

    JSONArray backgrounds = new JSONArray();
    backgrounds.put("#5cb85c");
    backgrounds.put("#d9534f");

    pieChart.getData().addOneLabel("Incomes");
    pieChart.getData().addOneLabel("Egress");

    JSONArray pieData = new JSONArray();
    pieData.put(incomesTaxes);
    pieData.put(egressTaxes);

    pieChart.getData().addPieDataset(backgrounds, pieData);

    return pieChart.toJSON();
}
```

Fuente: Elaboración propia

Solo queda añadir al mapa la lista de facturas que se mostrarán en la vista, y que están ordenadas por fecha de mayor a menor. Para ello al método se le pasan las dos listas de facturas que se obtuvieron anteriormente, la de ingresos y la de egresos. En este se creará una nueva lista y se añadirán las listas que se le pasan por parámetros. Luego, con la función de

ordenación se organizan con un comparador por el campo “InvoiceDate” y se le añade la función “reversed” para que se establezcan de mayor a menor.

Ilustración 22: Método para ordenar dos listas de facturas por fecha

```
private static List<Invoice> getInvoicesShortedListByDate(List<Invoice> invoicesIncomes, List<Invoice> invoicesEgress){
    List<Invoice> invoicesList = new ArrayList<>();
    invoicesList.addAll(invoicesIncomes);
    invoicesList.addAll(invoicesEgress);
    invoicesList.sort(Comparator.comparing(Invoice::getInvoiceDate).reversed());
    return invoicesList;
}
```

Fuente: Elaboración propia

Finalmente, tras haber realizado todos los pasos anteriores y tener el mapa rellenado con todos los datos necesarios para cargarlos en el *dashboard*, el mapa es devuelvo a la vista en formato JSON y, esta con cada clave – valor rellenará el código HTML de la página.

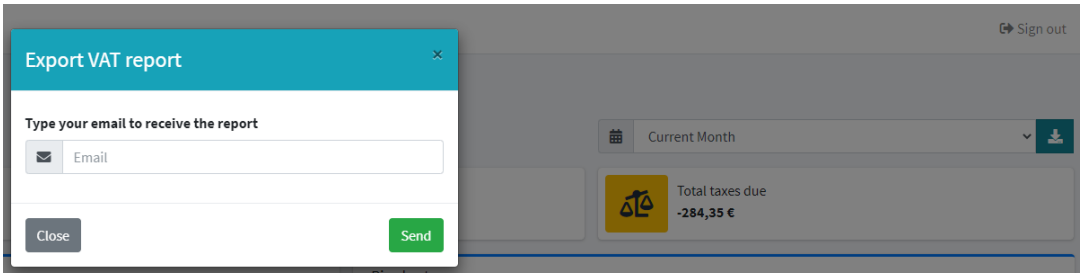
6.3.3. Generación de informes

En este trabajo se ha querido implementar una forma de extraer los datos de la aplicación, por ello se ha desarrollado la generación de informes en formato PDF, y su posterior envío por correo electrónico.

Actualmente el proyecto cuenta con tres tipos de informes que se pueden generar, uno para los datos que se estén mostrando en el *dashboard*, y dos para facilitar la liquidación del IVA ante la Agencia Tributaria Canaria, ya que estos dos formatos están basados en los modelos 420 y 425 para que un asesor o empresa pueda rellenar una parte de modelo con los datos obtenidos.

Para la generación del informe del IVA con los datos y periodo seleccionado en el *dashboard*, se ha habilitado un botón con un símbolo de descarga/extracción, el cual cuando se hace clic sobre él abre un modal en el que nos pedirá que escribamos el email al que queremos que nos llegue el informe.

Ilustración 23: Frangmento del dashboard donde se muestra el modal y el botón que lo abre



Fuente: Elaboración propia

Nuevamente se vuelve a usar peticiones Ajax para realizar está comunicación entre la vista y el servidor, y la ejecución de función definida.

En la función de JavaScript se valida mediante una expresión regular que el email sea correcto, ya que el input a no ser usada como un formulario normal, pues, aunque se le ponga el tipo de email, este no funciona.

Ilustración 24: Función que devuelve si un email es valido o no

```
function isEmail(email) {
    let regex = /^([a-zA-Z0-9_\.\-\+])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+$/;
    return regex.test(email);
}
```

Fuente: Elaboración propia

Como sistema para mostrar mensajes de éxito o errores al usuario, se ha definido y hecho uso de Toast de la librería (*SweetAlert2*, s. f.), ya que es una forma discreta y muy visual, a la vez que se puede seleccionar el tipo de mensaje que se va a mostrar y su descripción.

Ilustración 25: Definición del Toast y tipos de mensajes

Fuente: Elaboración propia

Siguiendo con la función principal, si el email es correcto obtendrá el valor del periodo seleccionado y se le pasará al servidor como parámetro junto al email.

Ilustración 26: Función para la generación y envío del informe del IVA

```
function ajaxSendIVAReport(){
    let email = $('#report_email').val();

    if(isEmail(email)){
        let valueOfSelect = $('#datepicker').val();
        $.ajax({
            url: "/admin/reporting/ajax-send-report",
            method: "POST",
            data: { period : valueOfSelect, email : email },
            dataType:"JSON",
            success: function(data){
                $('#close_modal').trigger('click');
                $('#report_email').val('');
                Toast.fire({
                    icon: 'success',
                    title: 'Report sent successfully!'
                });
            },
            error: function () {
                Toast.fire({
                    icon: 'error',
                    title: 'Something was wrong!'
                });
            }
        });
    }else {
        Toast.fire({
            icon: 'error',
            title: 'Wrong email format!'
        });
    }
}
```

Fuente: Elaboración propia

Una vez recibida la respuesta del servidor en la vista, se cierra el modal, se limpia el campo del email, y se le notifica al usuario que el envío se ha realizado correctamente.

Para que el servidor devuelva lo que se espera, se usa el método “sendIVAReport” de la clase *ReportingCommand*, el cual obtiene los datos con relación al periodo seleccionado, tal como se hace para cargar la página del *dashboard*, solo que, en este caso, esos datos se le pasa a la clase *IVAReport* que es la que se encarga de generar el informe en formato PDF.

Ilustración 27: Método para la generación y envío de informes del IVA

```
public static JSONObject sendIVAReport(String currentUserId, String period, String email) {
    Map<String, Object> content = new HashMap<>();
    LocalDateTime startDate = getStartDate(period);
    LocalDateTime endDate = Utils.getCurrentDate();

    String filename = createFilename(currentUserId, endDate);

    List<Invoice> invoicesIncomes = getInvoicesListPerPeriodAndType(currentUserId, InvoiceType.INCOME, startDate, endDate);
    Double incomesTaxes = invoiceService.getSumTotalTaxes(invoicesIncomes);

    List<Invoice> invoicesEgress = getInvoicesListPerPeriodAndType(currentUserId, InvoiceType.EGRESS, startDate, endDate);
    Double egressTaxes = invoiceService.getSumTotalTaxes(invoicesEgress);

    content.put("filename", filename);
    content.put("currentUserId", currentUserId);
    content.put("period", startDate.toLocalDate().toString() + " | " + endDate.toLocalDate().toString());
    content.put("incomesTaxes", incomesTaxes.toString());
    content.put("egressTaxes", egressTaxes.toString());
    content.put("totalTaxes", Double.toString( d: incomesTaxes - egressTaxes));
    content.put("invoicesShortedList", getInvoicesShortedListByDate(invoicesIncomes, invoicesEgress));

    try {
        IVAReport ivaReport = new IVAReport();
        ivaReport.create(content);
    } catch (IOException e) {
        e.printStackTrace();
    }

    sendMailWithFiles(email, filename);

    return new JSONObject();
}
```

Fuente: Elaboración propia

A la clase de generación del informe también se le pasa el nombre del fichero que sigue la estructura del id del usuario actual y la fecha y hora actual. Como los nombres de archivos no pueden tener los dos puntos de las horas, estos se suprimen.

Ilustración 28: Método para crear un nombre de archivo con el id del usuario y momento actual

```
private static String createFilename(String currentUserId, LocalDateTime endDate){
    return currentUserId + "_" + endDate.toString().replace( target: ":", replacement: "");
}
```

Fuente: Elaboración propia

Para la elaboración de los informes, se ha creado una interfaz llamada *PDFGenerate* que define un método créate, además de dos constantes, una especifica la ruta donde se guardará el informe generado y la otra, la extensión de los informes.

Ilustración 29: Interfaz PDFGenerate

```
public interface PDFGenerate {

    String PATH = "src/main/resources/public/finbook/files/temp/";
    String FILE_EXTENSION = ".pdf";

    void create(Map<String, Object> content) throws IOException;

}
```

Fuente: Elaboración propia

Para el caso del informe del IVA, se ha creado una clase que implementa esta interfaz y el método *create*. En dicho método se usa la librería (*iText 7*, s. f.) con la que se define la estructura del PDF a crear y añadiendo los datos que corresponden en su lugar.

En primer lugar, se inicializan dos variables con los tipos de letras a usar, y luego se completa la ruta final del archivo que se conforma con la ruta definida en la interfaz, el nombre del fichero recibido y la extensión que se encuentra también definida. Seguidamente se comprueba que no existe un archivo con el mismo nombre en la ruta, y en caso de que esto ocurra, se elimina.

Ilustración 30: Fragmento de código del método create de la clase IVAReport

```
PdfFont timesRoman = PdfFontFactory.createFont(StandardFonts.TIMES_ROMAN);
PdfFont helveticaBold = PdfFontFactory.createFont(StandardFonts.HELVETICA_BOLD);

String pathToSave = PATH.concat(content.get("filename").toString()).concat(FILE_EXTENSION);

try {
    Path fileToDeletePath = Paths.get(pathToSave);
    Files.deleteIfExists(fileToDeletePath);
} catch (IOException e) {
    e.printStackTrace();
}
```

Fuente: Elaboración propia

Luego se abre el documento en el que se irá insertando la estructura y los datos. Para nos mostrar todo el código ya que es extenso, se mostrará fragmentos de partes comunes:

Ilustración 31: Fragmento de código en la que se crea un parrafo y se añade al documento

```
Paragraph p = new Paragraph( text: "Finbook report")
        .setTextAlignment(TextAlignment.CENTER).setFont(helveticaBold).setFontSize(20).setUnderline();
document.add(p);
```

Fuente: Elaboración propia

Ilustración 32: Fragmento de código en la que se crea una lista y se añade al documento

```
// Create a List
List list = new List()
        .setSymbolIndent(12)
        .setListSymbol("\u2022")
        .setFont(timesRoman);
// Add ListItem objects
list.add(new ListItem("ID: ".concat(content.get("currentUserId").toString())))
        .add(new ListItem("Reporting period: ".concat(content.get("period").toString())));
// Add list to the document
document.add(list);
```

Fuente: Elaboración propia

Ilustración 33: Fragmento de código en la que se crea un tabla y se añade al documento

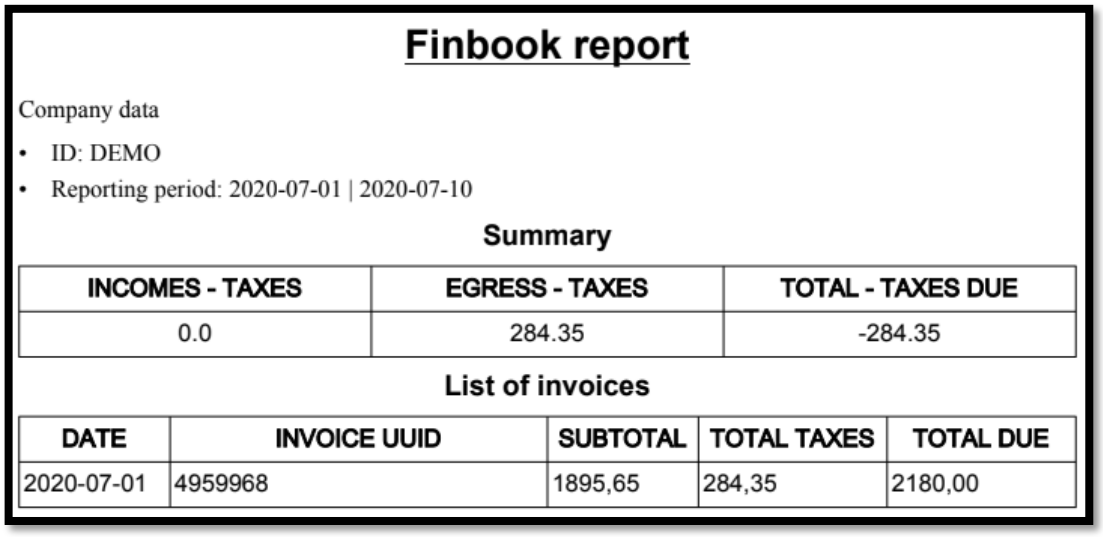
```
Table table = new Table(UnitValue.createPercentArray( size: 3)).useAllAvailableWidth();
table.addCell(headerCell( textToAdd: "INCOMES - TAXES"));
table.addCell(headerCell( textToAdd: "EGRESS - TAXES"));
table.addCell(headerCell( textToAdd: "TOTAL - TAXES DUE"));
table.addCell(centerCell(content.get("incomesTaxes").toString()))
        .addCell(centerCell(content.get("egressTaxes").toString()))
        .addCell(centerCell(content.get("totalTaxes").toString()));
document.add(table);

p = new Paragraph( text: "List of invoices")
        .setTextAlignment(TextAlignment.CENTER).setFont(helveticaBold).setFontSize(14);
document.add(p);
```

Fuente: Elaboración propia

En el código mostrado faltaría dos métodos, uno cuya función es la de crear una celda si se quiere como encabezado de una tabla y centrado, y otro que crea una celda centrada. No es necesario su uso, se separó en métodos apartes para no repetir código.

Una vez todo el documento ha sido formado con la estructura que se quiere y los datos incluidos, no hay que olvidar cerrar el documento. Tras hacerlo el método habrá guardado el PDF en la ruta definida al principio y terminará, dando paso al envío del correo electrónico del cual veremos su funcionamiento en el punto siguiente, donde después de enviar el documento lo borrará de la ruta. El resultado final del informe del IVA quedaría de la siguiente forma:

Ilustración 34: Ejemplo de un informe de IVA generado



Finbook report

Company data

- ID: DEMO
- Reporting period: 2020-07-01 | 2020-07-10

Summary

INCOMES - TAXES	EGRESS - TAXES	TOTAL - TAXES DUE
0.0	284.35	-284.35

List of invoices

DATE	INVOICE UUID	SUBTOTAL	TOTAL TAXES	TOTAL DUE
2020-07-01	4959968	1895,65	284,35	2180,00

Fuente: Elaboración propia

Para la realización de los informes basados en los modelos 420 y 425 de la Agencia Tributaria Canaria, se ha seguido el mismo proceso, pero usando en cada caso la clase del modelo al que correspondan.

Ilustración 35: Fragmento de modelo 425 de la Agencia Tributaria Canaria

Agencia Tributaria Canaria

IMPUESTO GENERAL INDIRECTO CANARIO
DECLARACIÓN - RESUMEN ANUAL

Modelo **425**

1. - Ejercicio

Ejercicio

Gran empresa ___ Registro de devolución mensual ___

Régimen especial del pequeño empresario o profesional ___

Declaración sustitutiva _____

Declaración sustitutiva por rectificación de cuotas en caso de concurso de acreedores _____

N.º de justificante

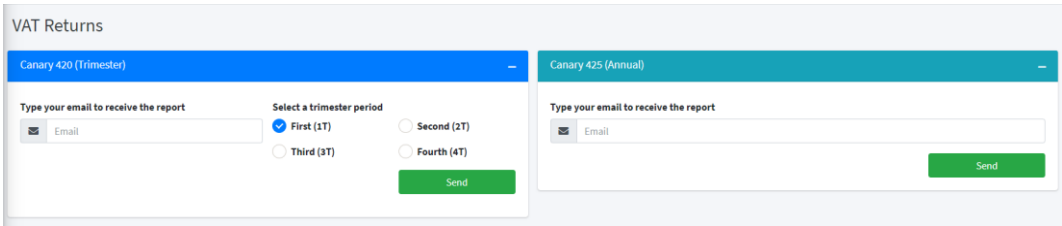
2. - Datos identificativos

N.I.F.	Apellidos y nombre / Razón social					
S.G.	Nombre de la vía pública	Número	Esc.	Piso	Puerta	Teléfono
Provincia	Municipio			Código postal		Fax

Fuente: (Agencia Tributaria Canaria, 2020)

La petición de generación de los informes basados en los de la Agencia se realizan a través de la página “VAT Returns” (European Commission, s. f.), lo que en español significa devolución del IVA. También se le puede llamar *VAT Declaration*, que es como normalmente se conoce, declaración del IVA.

Ilustración 36: VAT Returns page



Fuente: Elaboración propia

El proceso es el mismo que el seguido anteriormente, exceptuando que a estoy informes se les ha añadido un color a las celdas de una tabla, ya sean para un encabezado principal o uno secundario. Si se trata de un encabezado principal se hace un uso del color celeste RGB(20,255,225), en cambio si se trata de uno secundario, se utilizará el gris RGB(115,163,163) y en este caso, además la letra se pondrá en color blanco. Se ha elegido estos colores para que tuviera especial relación con los modelos oficiales de la Agencia Tributaria Canaria.

Ilustración 37: Código para especificar el color de los encabezados de los modelos 420 y 425

```
private Cell topHeaderCell(String textToAdd) {
    Cell cell = new Cell( rowspan: 1,  colspan: 0).add(new Paragraph(textToAdd));
    cell.setBackgroundColor(new DeviceRgb( r: 20,  g: 225,  b: 225));
    cell.setBold();
    return cell;
}

private Cell nextTopHeaderCell(String textToAdd) {
    Cell cell = new Cell( rowspan: 1,  colspan: 0).add(new Paragraph(textToAdd));
    cell.setBackgroundColor(new DeviceRgb( r: 159,  g: 163,  b: 163));
    cell.setFontColor(new DeviceRgb( r: 255,  g: 255,  b: 255));
    cell.setBold();
    return cell;
}
```

Fuente: Elaboración propia

El resultado final de los documentos generados, tanto para el modelo 420 como para el 425, han sido los siguientes:

Ilustración 38: Ejemplo de un informe generado basado en el modelo 420

IGIC (Mod. 420)

1. - Periodo de liquidación			
EJERCICIO	2020	PERÍODO	3T
2. - Datos identificativos			
DOMICILIO FISCAL			
N.I.F.		DEMO	
3. - Ingreso		**6. - Sujeto pasivo**	
Importe	-284.35	Fecha	2020-07-10T15:05:15.502579400
7. - Liquidación			
I.G.I.C. DEVENGADO			

	Base imponible	**Tipo de gravamen %**	**Cuota devengada**
	0.0	6.50	0.0
Total cuotas devengadas			0.0

I.G.I.C. DEDUCIBLE Y RESULTADO AUTOLIQUIDACIÓN

	Base	**Cuota**
IGIC deducible en operaciones interiores corrientes	1895.65	284.35
Total cuotas deducibles		284.35
Diferencia		-284.35
Resultado de la autoliquidación		-284.35

Fuente: Elaboración propia

Ilustración 39: Ejemplo de un informe generado basado en el modelo 425

IGIC (Mod. 425)

1. - Ejercicio	
Ejercicio	2020
Periodo	2020-01-01 | 2020-07-10
2. - Datos identificativos	
N.I.F.	DEMO
4. - Datos del representante y firma de la declaración	
PERSONAS FÍSICAS y COMUNIDADES DE BIENES (REPRESENTANTE)	
Fecha	2020-07-10T15:06:58.139586700
5. - Operaciones realizadas en régimen general	
BASE IMPONIBLE, TIPOS y CUOTAS	

	Base imponible	Tipo %	Cuota
Régimen ordinario	28975.97	6.50	4197.87
Total cuotas devengadas			4197.87

DEDUCCIONES

	Base	Cuota
IGIC deducible en operaciones interiores corrientes	29607.5	3986.76
Total cuotas deducibles		3986.76

RESULTADO DE LAS AUTOLIQUIDACIONES	
Resultado régimen general	211.10999999999967

Fuente: Elaboración propia

6.3.4. Envío de informes por email

El desarrollo de esta función se ha realizado mediante la creación de una clase llamada Mail y utilizando los métodos de la librería JavaMail para formar el mensaje a enviar. En la clase creada se inicializan en el constructor las propiedades que contiene todos los datos de acceso a la dirección de correo con el que se quiere enviar el mensaje.

Las propiedades que se necesitan establecen son las que se puede ver en la siguiente imagen, y los valores están como constantes ya que no variaran con la ejecución del programa.

Ilustración 40: Propiedades necesarias para mandar un correo electrónico

```
private void setUpProperties() {
    properties.put("mail.smtp.host", HOST);
    properties.put("mail.smtp.starttls.enable", "true");
    properties.put("mail.smtp.port", PORT);
    properties.put("mail.smtp.auth", "true");
    properties.put("mail.smtp.user", SENDER_USER);
    properties.put("mail.smtp.clave", PWD);
}
```

Fuente: Elaboración propia

La clase Mail contiene un método para enviar correos adjuntando un archivo en el que se recibe por parámetros la dirección de correo electrónico a la que se quiere enviar el correo y el nombre del archivo a adjuntar, que ha tenido que ser previamente creado.

Ilustración 41: Método para enviar correos electrónicos adjuntando un fichero

```
public void sendMailAttachFile(String userMail, String filename) {
    String pathToFile = "src/main/resources/public/finbook/files/temp/".concat(filename);
    Session session = Session.getDefaultInstance(properties);

    try {
        MimeMessage message = new MimeMessage(session);
        message.setFrom(new InternetAddress(SENDER_EMAIL));
        message.addRecipient(Message.RecipientType.TO, new InternetAddress(userMail));
        message.setSubject("Finbook reporting");

        Multipart multipart = new MimeMultipart();
        multipart.addBodyPart(formatMimeBodyPart(getMainMessage()));
        multipart.addBodyPart(formatMimeBodyPart(getEnviromentMessage()));
        multipart.addBodyPart(attachFile(pathToFile, filename));

        message.setContent(multipart);
        transport(session, message);

        deleteFile(Paths.get(pathToFile));

    } catch (MessagingException | IOException me) {
        me.printStackTrace();
    }
}
```

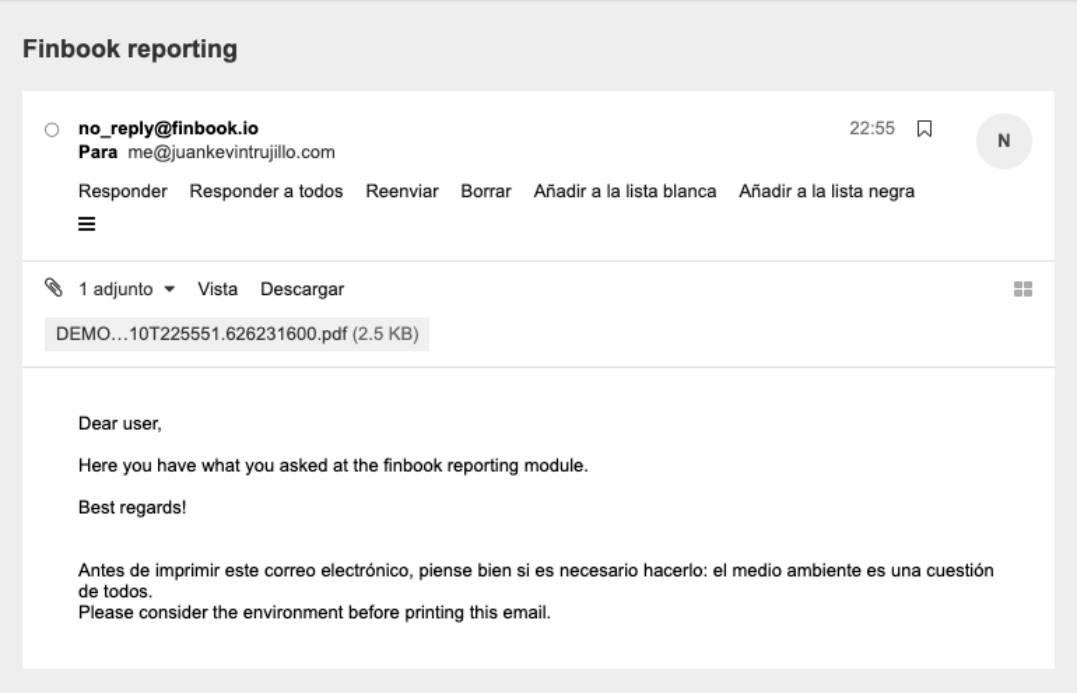
Fuente: Elaboración propia

En dicho método se crea una sesión con las propiedades definidas en el constructor, y luego se crea un objeto *MimeMessage* al que se le pasa la sesión y se añaden el email del emisor, y el email del receptor, así como el asunto del correo. En este caso el email del emisor se ha utilizado una dirección de correo creada con el dominio que se adquirió para este proyecto uso del dominio (finbook.io) adquirido para crear una dirección "*no_reply*".

Por otra parte, en este caso se añaden tres partes diferenciadas, que corresponden al mensaje principal del correo, una segunda que es un mensaje diciéndole al usuario que solo imprima el correo si es necesario con el fin de preservar el medioambiente, la tercera que adjuntar el archivo.

Una vez terminado el mensaje, se le pasa al *transport* que se conectará al host por SMTP, enviará el mensaje, y cerrará la conexión. Y finalmente, como se dijo anteriormente en la generación de informes, el fichero adjuntado y enviado se elimina.

Ilustración 42: Ejemplo de correo electrónico recibido



Fuente: Elaboración propia

6.3.5. Implementación del subscriber para la carga la base de datos

Para poder cargar la base de datos mediante un subscriber de la plataforma de datos, hemos creado una clase que se inicializa al arrancar la aplicación y que estaría comprobado todo el rato si hay facturas nuevas para su posterior inserción en nuestra base de datos.

Ilustración 43: Clase Subscriber

```
public class Subscriber {

    private static final String TERMINALS_PATH = "src/reporting/temp/terminals/";
    private static InvoiceService invoiceService;

    public static void init() {
        invoiceService = new InvoiceService();

        JmsConnector connector = new JmsConnector(
                brokerUrl: "failover:(tcp://localhost:63001)", //BrokerURL
                user: "io.finbook.datamart",
                password: "io.finbook.datamart",
                clientId: "reporting.finbook-datamart",
                new File(TERMINALS_PATH)
        );
        connector.start();
        DatamartTerminal dt = new DatamartTerminal(connector);
        dt.subscribe(Subscriber::processedInvoice, subscriberId: "reporting.finbook");
    }

    private static void processedInvoice(ProcessedInvoice e){
        // Parse Instant to LocalDateTime
        LocalDateTime date = LocalDateTime.ofInstant(e.date(), ZoneOffset.UTC);

        // Encode content file to Base64
        String xmlFileEncode = Utils.encodeStringToBase64(e.xml());

        Invoice invoice = new Invoice(e.uUID(), date, e.pC(), e.type(), e.use(),
                e.issuerName(), e.issuerRFC(), e.receiverName(), e.receiverRFC(), e.concept(), e.subtotal(),
                e.taxRate(), e.total(), e.currency(), xmlFileEncode
        );

        invoiceService.add(invoice);
    }
}
```

Fuente: Raúl Lozano y Juan Kevin Trujillo

En la clase *Subscriber* se crear un conector especificando la ruta, usuario y contraseña, como también el id del cliente que tiene que ser único para todos los *subscribers*. Luego se inicia la conexión y se crea el *datamart* terminal al que nos vamos a suscribir, donde se específica el método que se ejecutará en caso de recibir una factura nueva, en este caso el *processedInvoice*.

En el método *processedInvoice* se crea un *LocalDateTime* a partir de la fecha de la factura añadiendo la zona UTC, y se pasa a Base64 el contenido del archivo XML, ya que lo que viene es el contenido total del documento y así no es como se guarda en la estructura de la base de

datos creada en MongoDB. Finalmente se crea un objeto de la clase *Invoice* con los datos requeridos, y se llama al método de añadir del *Service* para que inserte la factura en la base de datos.

6.4. Página web global de Finbook con GitHub Pages

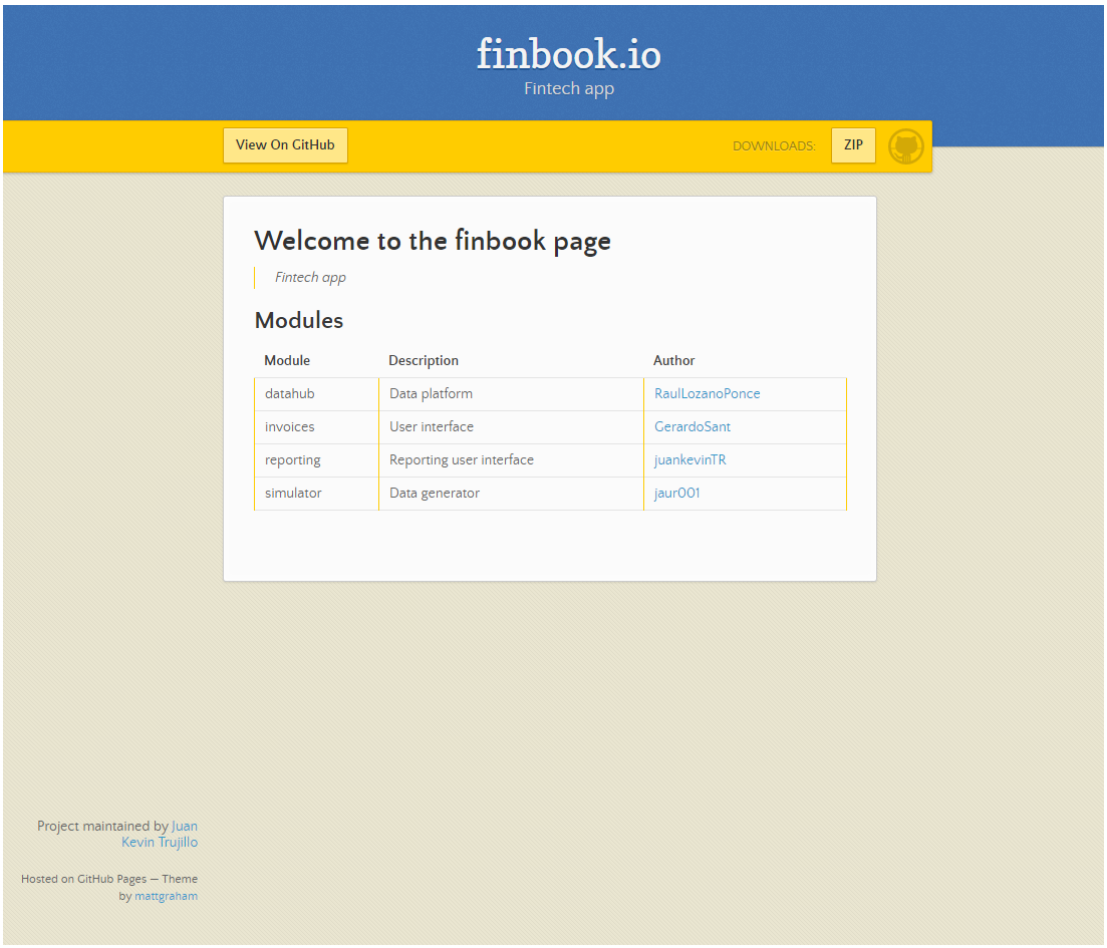
Con se dijo en el punto anterior, para este proyecto se ha comprado el dominio ***finbook.io*** y además ***finbook.es***, no obstante, este está redireccionado al ".io". El dominio se compró en IONOS y se maneja desde la parte de administración de esta página.

Como aún no se dispone de un servidor o hosting donde alojar la página web del proyecto entero, se ha decido hacer uso de la herramienta que nos proporciona GitHub, y que nos permite crear y alojar una página web con el nombre de nuestro grupo de trabajo, en este caso, finbook-io.

La web se puede consultar accediendo a la https://finbook-io.github.io/ o haciendo uso del dominio principal comprado http://finbook.io/ ya que está redireccionado al de GitHub.

Para realizarla se creado un nuevo repositorio con el nombre que se especifica en GitHub Pages, y luego se ha hecho uso de la plantilla (*Leap day*, s. f.).

Ilustración 44: Página web de Finbook.io creado con GitHub Pages



Fuente: Elaboración propia

7. Justificación de las competencias

La realización de este Trabajo de Fin de Título lleva aparejada unas competencias generales y específicas del Grado en Ingeniería Informática de las que se han cubierto:

- **CII01**: "Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente." (Universidad de Las Palmas de Gran Canaria, s. f.)
  Con el desarrollo de este proyecto, se ha diseñado e implementado un módulo de una aplicación más grande que tenía que ser integrada a otros módulos, por lo que en su realización se ha tenido en cuenta que la aplicación cumpla los requisitos de fiabilidad, seguridad y calidad.
- **CII08**: "Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados." (Universidad de Las Palmas de Gran Canaria, s. f.)
  La aplicación que se ha desarrollado es segura dado que se ha implementado un sistema de autenticación mediante certificado digital, y para el proyecto se utilizó java como lenguaje de programación por su robustez y su extensión en el desarrollo de diversos programas.
- **IS04**: "Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales." (Universidad de Las Palmas de Gran Canaria, s. f.)

  Con la realización de la aplicación y la posterior documentación del trabajo elaborado, explicando el objetivo a resolver, las tecnologías usadas y su funcionamiento, esta competencia quedaría cubierta.

- **TFG01**: "Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas." (Universidad de Las Palmas de Gran Canaria, s. f.)
  La realización de este trabajo de fin de título y la defensa de este ante el tribunal, estaría cubriendo esta competencia.

8. Conclusiones

8.1. Punto de vista personal

Participar en un proyecto junto a otros compañeros, en el que cada uno se ha encargado de un módulo para lograr un objetivo final más grande, me ha supuesto una mejora y una visión más clara de la forma de trabajar en equipo, lo que supone una ventaja de cara al desarrollo de la actividad profesional en una empresa. A su vez, el desarrollo de este trabajo de fin de título me ha servido para conocer de primera mano como se desarrolla un proyecto desde la idea inicial hasta su implementación.

Dada la envergadura de Finbook, confío en que no hubiera sido posible realizarlo individualmente por el tiempo que se requería para acabarlo, por lo que ha salido adelante gracias al trabajo en grupo que he realizado junto a mis compañeros y al apoyo de nuestro tutor que nos ha ido guiando hasta la finalización del proyecto.

Por otro lado, siento que he podido utilizar y aprender diferentes conceptos y herramientas que en la carrera no se ven, como son las tecnologías de Big Data y sus implementaciones, lo que me ha hecho ver sus virtudes e inconvenientes, y también me ha ayudado a ser más autodidacto a la hora de buscar soluciones para aquellos problemas que fuesen surgiendo. Además, el desarrollo de Finbook me ha permitido aprender, utilizar y mejorar conceptos con dos menciones del grado en Ingeniería Informática, que son la de Tecnologías de la Información, en la cual me he especializado, y la de Ingeniería del Software, por lo que el grado de aprendizaje y confianza personal ha aumentado tras este trabajo.

8.2. Punto de vista de contribución

Este trabajo de fin de título se ha realizado como un experimento para entender las ventajas del uso de la factura digital para inversores y pequeñas empresas, y aportar una aplicación informática para el análisis y visualización de los datos mediante una página web como también al generar informes.

Por otra parte, esto puede ser un aliciente para que la Agencia Tributaria invierta en desarrollar, implementar y extender el uso de la factura digital, ya que la utilización de las tecnologías Big Data, permitirían ahorrar costes y prevenir el fraude fiscal.

8.3. Posibles continuaciones de Finbook

Este trabajo permite que su escalabilidad siga aumentando con nuevas funciones que pueden ser desarrolladas en un futuro y que aportarían más valor al proyecto:

- **Análisis por tipo de uso de la factura:** en el que se calcularán los ingresos y gastos por categorías y, así poder sacar informes para llevar un mejor control de las finanzas.
- **Alertas por tipo de uso de la factura:** se tendrá en cuenta los cálculos y la media de todos los meses y, en caso de que una categoría tenga un pico de gastos, se le mostrará una alerta al usuario.
- **Analizar productos:** en relación con el estudio de los datos en Big Data, se podría añadir una función para poder analizar los productos/conceptos de las facturas, y así saber qué productos son más vendidos, por ejemplo.
- **Implementación de la aplicación en varios idiomas:** se añadiría una opción para que el usuario pueda elegir el idioma de preferencia en el que quiere que se muestre la aplicación. Esto se realizaría mediante diccionarios, un fichero por cada idioma, en los que se definirán unas claves con sus respectivos valores.
- **Gestión de un grupo de empresas:** actualmente solo se permite la visualización de los datos y generación de informes de una única empresa, pero en el futuro se podría implementar que un usuario pueda tener acceso a varias empresas o, que sea un grupo de empresas, donde la aplicación le mostraría los datos filtrado por cada una de ellas, así como los datos totales del grupo entero.
- **Utilización de varias monedas:** en las facturas y los informes que se generan en nuestro proyecto la moneda utilizada es el euro, por lo que se añadiría que una empresa pueda tener facturas en diversas monedas, pudiendo realizar entonces una consulta en el mercado de valores, para posteriormente hacer la conversión a la moneda que se quiera en tiempo real.

9. Referencias

Agencia Tributaria Canaria. (2020). *IGIC - Agencia Tributaria Canaria*. http://www.gobiernodecanarias.org/tributos/portal/jsf/publico/asistenciaContribuyente/modelos/listado.jsp?tributo=IGIC&jftfdi=&jffi=listado.jsp

Alves Fernández, Marcelo Luis; Botelho Junqueira, Álvaro Ribeiro; Hama, Renato; Hernándes Júnior, Nelson; Mitsutani, Heitor; Tsuha, S. (2010). Un nuevo paradigma para el control y la supervisión de los contribuyentes minoristas. *Revista de Administración Tributaria CIAT/AEAT/IEF*.

European Commission. (s. f.). *VAT returns*. Recuperado 10 de julio de 2020, de https://ec.europa.eu/taxation_customs/business/vat/eu-vat-rules-topic/vat-returns_en

Herring, M. (2015). *SparkJava: Separating routing and resources*. https://www.deadcoderising.com/sparkjava-separating-routing-and-resources/

iText 7. (s. f.). Recuperado 10 de julio de 2020, de https://itextpdf.com/es/products/itext-7

Leap day. (s. f.). Recuperado 10 de julio de 2020, de https://github.com/pages-themes/leap-day#usage

Lozano Ponce, R. (2020a). *Firma*. https://github.com/finbook-io/Firma

Lozano Ponce, R. (2020b). *Sign-Verifier*. https://github.com/finbook-io/sign-verifier

Sánchez, M. (2019). *El impulso de la facturación electrónica en Europa y España*. https://www.generixgroup.com/es/blog/impulso-facturacion-electronica-europa-espana

Santana Franco, G. (2020). *FinbSignRegister*.

Seres. (2019). *Launching applications using custom browser protocols*. https://support.shotgunsoftware.com/hc/en-us/articles/219031308-Launching-applications-using-custom-browser-protocols

SweetAlert2. (s. f.). Recuperado 10 de julio de 2020, de https://sweetalert2.github.io/

Universidad de Las Palmas de Gran Canaria. (s. f.). *Objetivos y Competencias del GII*. Recuperado 10 de julio de 2020, de https://www2.ulpgc.es/archivos/plan_estudios/4008_40/ObjetivosyCompetenciasdelGII.pdf

Agradecimientos

Quiero agradecer a todos los profesores de la universidad que me han ayudado durante mi aprendizaje, y en especial mención a José Daniel por haberme guiado en la planificación y realización del Erasmus que realicé en Maribor, Eslovenia, ya que esa experiencia fue y será siempre muy enriquecedora en mi vida personal y profesional, y por otro lado, a mi tutor de este trabajo, José Juan, por haberme enseñado diversas versiones sobre cómo realizar un trabajo, y hacer hincapié en el desarrollo de software de calidad.