FIB

TRABAJO FINAL DE GRADO, MODALIDAD B

Titulación: Grado en Ingeniería Informática

Especialidad: Ingeniería del Software

Empresa: Safelayer Secure Communications S.A

Estudio, propuesta y desarrollo de componentes avanzados para una PKI Híbrida

Autor:

Ben Bouker Hmaddouch, Soufiane

Director:

Ciurana, Óscar

(Safelayer Secure Communications S.A)

Ponente:

Jordán, Francisco

(Departamento de Arquitectura de Computadores)

Barcelona, 9 de Abril del 2018

Glosario

- **Software**: Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.
- ***On-premises***: Software que se instala y se ejecuta en computadoras de las instalaciones propias de la persona u organización que usa el software, en lugar de una instalación remota como una granja de servidores o en la nube.
- ***As-a-service***: Software como un servicio, un modelo de distribución de software donde el soporte lógico y los datos que maneja se alojan en servidores de una compañía de tecnologías de la información y comunicación, a los que se accede vía Internet desde un cliente.
- ***PKI*:** Una infraestructura de clave pública o PKI es una combinación de *hardware, software,* políticas y procedimientos de seguridad que permiten la ejecución con garantías de operaciones criptográficos como el cifrado, la firma digital y el no repudio de transacciones electrónicas.
- **Híbrida**: En desarrollo web, una aplicación híbrida es una forma de integración y reutilización. Ocurre cuando de una aplicación web es usada o llamada desde otra aplicación, con el fin de reutilizar su contenido y/o funcionalidad.
- **Certificado digital**: Un certificado digital o certificado electrónico es un fichero informático generado por una entidad de servicios o certificación que asocia unos datos de identidad a una persona física, organismo o empresa, confirmando de esta manera su identidad digital en Internet.
- **Testing**: El testing y/o las pruebas de software es un método para evaluar la funcionalidad y calidad del software.
- **Proxy**: Un proxy es un servicio que hace de intermediario en las peticiones de recursos que realiza un cliente a otro servidor.
- **API**: Una API es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción.

Resumen

En los últimos años se ha producido un gran avance en las tecnologías dentro del área de la Seguridad Informática, las cuales integran tecnologías de la información con herramientas cotidianas para mejorarnos la gestión, ofrecernos más funcionalidades y mejorar nuestra comodidad y/o experiencia en el uso de dichos dispositivos. El simple hecho de gestionar cuentas bancarias, poder acceder a cualquier dispositivo industrial o también poder firmar digitalmente mediante un smartphone, hace que este campo de la informática esté en pleno auge.

En concreto, este proyecto se centra en el campo de las firmas digitales, sistemas seguros de autenticación, verificación de firmas digitales, etc. Es por eso que un sistema de tal magnitud requiere de alguna infraestructura que gestione y garantice toda la seguridad necesaria en un sistemas tan crítico en términos de seguridad y/o privacidad.

Dicha infraestructura, es la PKI, y en concreto este proyecto tiene como principal objetivo la creación de componentes avanzados (tanto *frontend* como *backend*) que permitirán un sistema de registro orientado a usuarios de una Autoridad de Registro PKI Híbrida. Donde esencialmente por Híbrido entendemos que el sistema pueda desplegarse y funcionar en un entorno *on-promises* (máquinas en casa de la organización) y un entorno como servicio (máquinas en un proveedor de servicio o en la nube). Incluso que pueda desplegarse y funcionar en un entorno mixto o híbrido.

Abstract

In recent years there has been a great advance in the technologies within the area of Computer Security, which integrates information technologies with daily tools to improve our management, offer us more functionalities and improve our comfort and / or experience in the use of such devices. The simple fact of managing bank accounts, being able to access any industrial device or also being able to digitally sign with a smartphone, makes this field of computing in full swing.

Specifically, this project focuses on the field of digital signatures, secure authentication systems, verification of digital signatures, etc. That is why a system of this magnitude requires some infrastructure that manages and guarantees all the necessary security in a system so critical in terms of security and / or privacy.

This infrastructure is the PKI, and in particular this project has as its main objective the creation of advanced components (both frontend and backend) that will allow a registration system oriented to users of a Hybrid PKI Registration Authority. Where essentially by Hybrid we understand that the system can be deployed and operate in an on-promises environment (machines in the organization's home) and an environment as a service (machines in a service provider or in the cloud). Even it could be deployed and operate in a mixed or hybrid environment.

Índice de contenido

Índice de tablas

Índice de figuras

Capítulo 1: Introducción

1.1 Descripción del proyecto

Este proyecto se realiza como Trabajo Final de Grado (TFG) de Ingeniería Informática, especialidad en Ingeniería del Software, en la Facultad de Informática de Barcelona, Universidad Politécnica de Catalunya (UPC) conjuntamente con la empresa española Safelayer Secure Communications, en la cual estoy realizando las prácticas curriculares. Safelayer está especializada en fabricar software de seguridad para soluciones de infraestructura de clave pública (PKI), sistemas de autenticación multifactor, firma electrónica, cifrado y transacciones seguras [1]. Su tecnología se usa en proyectos de identificación electrónica de personas (ámbito corporativo o ciudadano), dispositivos conectados (*hardware* o *software*) y en la generalización de servicios de confianza en redes telemáticas tales como Internet y móviles.

Este proyecto está formulado por la propia empresa Safelayer. Este tiene como principal objetivo la creación de componentes avanzados (tanto *frontend* como *backend*) que permitirán un sistema de registro orientado a usuarios de una Autoridad de Registro PKI Híbrida, donde por usuario se entiende la persona, el dispositivo o el sistema para quién se emiten los certificados (operarios y usuarios finales) y por sistema de registro, toda la parte de gestión de usuarios, es decir, todo el sistema de autentificación. Por híbrida calificamos dos características esenciales de la nueva Autoridad de Registro. La primera característica es que pueda desplegarse y funcionar en un entorno on-premises (máquinas en casa de la organización) y un entorno como servicio (máquinas en un proveedor de servicio o en la nube). Incluso que pueda desplegarse y funcionar en un entorno mixto o híbrido.

1.2 Formulación del proyecto

Una vez visto el proyecto que se realiza, es conveniente saber qué pretende resolver el proyecto. Por ahora, el producto sólo está pensado para despliegues on-premises. Es posible desplegarlo en la nube como cualquier otro producto, pero el producto no lo facilita, se requiere que el cliente utilice herramientas de terceros para hacer el despliegue. Esto es un problema, pues otros sistemas implementan la operativa de infraestructuras de clave pública en la nube, es decir, como servicio.

Hay que destacar que una parte del proyecto ya ha sido diseñada y implementada por otras empresas, y es por ese motivo que conviene adaptarse a las nuevas funcionalidades introducidas por la competencia para seguir siendo una empresa competitiva. Tal y como está ahora implementado el producto, la Autoridad de Registro requiere del despliegue de una Autoridad de Registro Local en los puestos de registro. Esto hace que, en esencia, se requiera de un mantenimiento en cada puesto de registro. Sin embargo, si el proyecto se realiza correctamente no sería necesario desplegar software adicional en el puesto de registro, consiguiendo que se pueda operar la Autoridad de Registro de una forma completamente segura desde un smartphone o cualquier dispositivo con acceso a internet, pues es uno de los principales objetivos del proyecto, poder desplegarse en un entorno mixto o híbrido.

Además, la parte innovadora es el diseño del componente de gestión de usuarios que posteriormente se integrará en un portal web que permitirá el despliegue de infraestructuras de clave pública tanto on-premises como as-a-service. De esta manera, el cliente de nuestra empresa podría ofrecer un servicio de PKI gestionada. A su vez, los clientes de la PKI gestionada podrían acceder a su PKI desde el mismo frontal, el cual accedería a la PKI correspondiente. Esto es el concepto de multi-tenant en la nube, donde el frontal web podría tener varias PKI.

1.3 Objetivos del proyecto

Este proyecto tiene unos objetivos bastante ambiciosos basándonos en el marco el tiempo establecido para el desarrollo del proyecto, pero se irán priorizando y adaptando en función del avance. Dicho esto, a largo plazo la Autoridad de Registro deberá:

- ❏ Permitir a los clientes registrarse y gestionar sus cuentas para poder acceder a la plataforma de gestión de certificados.

- ❏ Permitir a las organizaciones / corporaciones registrarse y gestionar sus cuentas y la de sus clientes y sus operadores (Oficiales de Registro) para poder acceder a la plataforma de gestión de certificados.

- ❏ En ambos casos se deberá estudiar y decidir qué opciones de registro se permitirán, ya sean registros locales y/o mediante RRSS (redes sociales).

- ❏ Desplegarse y funcionar en un entorno on-premises y un entorno como servicio (máquinas en un proveedor de servicio o en la Nube). Incluso que pueda desplegarse y funcionar en un entorno mixto o híbrido, esto es, parte de la Autoridad de Registro corporativo y parte como servicio.

1.4 Stakeholders

Los stakeholder son todas aquellas personas, grupos y entidades que tienen intereses en cualquier tipo en una empresa y se ven afectados por su actividad. En nuestro caso nos centraremos en nuestro caso nos encontraremos con dos grandes grupos de stakeholders:

- **Clientes**: Por un lado, tenemos a los clientes, por clientes entendemos todas esas personas que usen la infraestructura de clave pública para su propio uso. A continuación clasificamos por roles el tipo de clientes :

    - **Oficiales de seguridad**: son los que definen quienes son los Oficiales de Registro y también quienes definen cómo y con qué credenciales se autentican dichos oficiales de registro y los usuarios finales.
    - **Responsables de Marketing**: se encargan de mantener la imagen corporativa. Esto significa que el proyecto debe ser personalizable según los requisitos del puesto de registro.
    - **Administradores de sistemas**: mantienen el software del puesto de registro.
    - **Operadores de registro**: son los encargados de operar el puesto de registro mediante procesos de registro F2F (face-to-face).
    - **Usuarios**: son los receptores del certificado/tarjeta, tanto en procesos F2F como procesos remotos.
    - **Desarrollador**: encargado de realizar el proyecto de la empresa, que a la vez es un TFG, así que también se va a ver beneficiado por este proyecto ya que le hará falta para acabar el grado universitario y recibir el título.

- **Competencia**: Por otro lado, los agentes implicados en este proyecto encontramos la competencia. La competencia son todas aquellas empresas o organizaciones que se dedican a ofrecer la operatividad de infraestructuras de clave pública tanto en máquinas on-premises o en máquinas en un proveedor de servicios. La realización de este proyecto podría repercutir negativamente para ellos, pues podrían perder parte de sus clientes ya que nuestra solución permitiría combinar nuestra PKI on-premises con otras PKI desplegadas como servicio mediante un mismo portal web.

Empresas competidoras serían, por ejemplo, Nexus Group [6] (ofrecen una PKI multi-tenant), Primekey [7] y/o Entrust [8], la cual ofrece PKI tanto on-premises como as-a-service [5]. Por último, añadir que esta competencia es a nivel internacional.

Capítulo 2: Contextualización

2.1 Contexto

Este proyecto surgió como iniciativa de la empresa Safelayer Secure Communications, S.A. en Julio de 2017 y está siendo realizado en el marco del proyecto *Distributed cloud-based service Architecture for managing billions of deViceIDs (DAVID) [2],* subvencionado por el Ministerio de Energía, Turismo y Agenda Digital dentro de la Acción Estratégica Economía y Sociedad Digital (AEESD) 2016-2017 con referencia TSI-100200-2016-26.

El proyecto *DAVID* pretende definitivamente popularizar y democratizar la tecnología PKI basada en certificados digitales de forma que ésta nunca más se considere una traba en el despliegue de servicios de información altamente seguros y confiables. Para ello, es fundamental el uso y aplicación de las nuevas tecnologías de virtualización y *Cloud*.

Safelayer Secure Communications, S.A. es una compañía destacada en el mercado de seguridad y confianza para las TIC (Tecnologías de la Información y la Comunicación). Dispone de múltiples proyectos en el ámbito de la tecnología para la autenticación, autorización, integridad y confidencialidad de la información.

Además de contextualizar el proyecto, en los puntos anteriores se han introducido los motivos de la creación de este proyecto, el problema general que intenta resolver y los objetivos, es decir, los problemas concretos que tiene que resolver este proyecto. Ahora, es necesario ver el estado del arte y quiénes son los actores implicados, es decir todo aquello que se beneficiará (o no) de la creación de los componentes de este portal web.

2.2 Estado del arte

En este proyecto, a diferencia de otros con un propósito parecido pero con un enfoque distinto, se requiere de una cierta asimilación y formación de los distintos conceptos. Dado que este proyecto está en el campo de la Seguridad Informática, se aconseja tener unas ciertas base, las cuales están explicadas en el Capítulo 5.

Una vez dicho esto, esta sección pretende mostrar lo que hay hasta ahora, ver qué aportaría mi proyecto y si se podría llegar a aprovechar algún software ya diseñada por algún tercero. Es por eso que para la realización de esta sección ha hecho falta un amplio estudio de mercado. Antes de empezar, es importante saber que, a nivel nacional, no se ha encontrado ninguna competencia que ofrezca los mismos servicios que ofrece la empresa Safelayer. Sin embargo, a nivel internacional, hay empresas competidoras como hemos visto en la sección de stakeholders implicados en el proyecto.

Como se ha visto en secciones anteriores, hay otras organizaciones o empresas que ofrecen operatividad con una PKI mediante un servicio. También hay otras empresas que ofrecen tanto la operatividad de sus PKI *on-premises* como las de PKI desplegadas *as-a-service*. Justamente por eso, la realización del proyecto se respalda en la capacidad que han tenido otros para hacerlo y, por lo tanto, se sabe de la viabilidad de poder hacerlo. Hasta aquí podemos ver que no se está innovando, pues se está intentando desarrollar algo que ya está en el mercado, adaptado por supuesto al producto de la empresa.

La parte novedosa y que requiere de un diseño es la siguiente: la idea del proyecto es la de poder crear una aplicación que permita la operatividad con distintas PKI de diferentes fuentes, desplegadas *as-a-service* y la operatividad con PKI *on-premises* de una forma segura [5]. Esto aumenta las fronteras de la empresa, ya que con el mismo portal web se podría operar con distintas PKI, independientemente de quién las haya creado. Además, se podría usar el producto PKI que ofrece la empresa Safelayer. Todo esto se conseguiría mediante el uso de

una API comuna. Está claro que hacer todo esto en 6 meses es prácticamente inviable, pero el proyecto está creado con la idea de ser finalizado en más tiempo del establecido para el TFG.

Para los componentes que se van a desarrollar ya existen servicios de autentificación y/o gestión de usuarios que lo ofrecen, como Auth0 [3], el cual se explica con más detalle en el apartado 5.4.3, se procederá a analizar las distintas soluciones para poder adaptarlo al sistema y/o productos de Safelayer. Sin embargo, para poder operar con diferentes PKI en el *cloud*, así como para usar nuestro producto desplegado *on-premises* se requiere de una solución nueva. Es por todos estos motivos que este proyecto requiere de un diseño de arquitectura importante. En esencia, esta es la justificación de por qué es prácticamente inviable terminar todo este proyecto en seis meses, pero sí puede servir de ayuda a Safelayer para poder empezar a encaminar este proyecto.

Capítulo 3: Alcance

3.1 Alcance del proyecto

El alcance de este proyecto es concreto y como hemos visto hasta ahora, es un proyecto que supondrá una duración superior a los seis meses. Es por eso que se ha hecho una previa planificación para la ejecución de este. Para la realización de dicho proyecto se estipulan unas fases a seguir las cuales son citadas a continuación:

- ❏ Análisis y estudio de los frameworks web y lenguajes más actuales y avanzados para el desarrollo de los componentes *frontend* y *backend*.

- ❏ Definición del protocolo de comunicación entre el *frontend* y el *backend* del subcomponente de la autoridad de registro para la gestión de usuarios*.*

- ❏ Desarrollo del software que implemente y permita la construcción de dicha Autoridad de Registro en entornos *on-premises.*

- ❏ Despliegue *as-a-service* por parte de prestadores de servicios de certificación.

El alcance de la última fase de determinará durante la ejecución del proyecto. No obstante, se prevé que haya dos sub-fases: La primera sub-fase sería el primer prototipo y la segunda el desarrollo final del sistema. En el caso de alargarse el proyecto más allá de 6 meses (o duración formal de la carga lectiva del TFG y las prácticas curriculares), el primer prototipo sería el objeto del TFG y la finalización del sistema final se llevaría a cabo ya como ingeniero dentro de Safelayer.

3.2 Posibles obstáculos

En esta subsección, se pretende poner en contexto situaciones y/o motivos que pueden llegar a desviar el alcance del proyecto:

- **Pérdida de datos parcial:** El contenido del proyecto local del ordenador puede perderse debido a fallo del SSD (Solid State Disk), interrupción de write del sistema operativo en el disco, cierre por crash del programa de edición u otro motivo que afecte al sistema.

    - Soluciones:
        - Guardar de forma periódica el contenido del trabajo y al finalizar la jornada laboral guardar el contenido en un servidor externo.
        - Pedir al personal IT de Safelayer que te proporcione el último *backup* hecho del disco local.

- **Pérdida de datos total:** El contenido del repositorio externo se pierde o quede completamente inutilizable debido a una rotura de los discos de almacenaje o destrucción del hardware por alguna calamidad.

    - Soluciones:
        - Se podría hacer *mirroring* del contenido del servidor en otro servidor localizado en una zona geográfica distinta.
        - Pedir al personal IT de Safelayer que te proporcione el último *backup* hecho del disco local.

- **Incumplimiento del contrato por parte de la empresa contratista:** la empresa contratista, Safelayer Secure Communications, cambia de planes o sufre quiebra económica y los fondos para la realización del proyecto desaparecen.

- Solución:
  - A nivel empresarial el proyecto quedaría cancelado, pero de forma autónoma el estudiante podría continuar con dicho proyecto con el permiso de la empresa.

- **Tiempo limitado**: la finalización del proyecto en un marco de tiempo de seis meses puede llegar a ser un obstáculo para la realización de todos los objetivos mencionados anteriormente.

  - Solución:
    - Mediante la metodología de trabajo se irá hablando periódicamente con el cliente sobre las tareas prioritarias y así poder acortar el alcance de dicho proyecto y presentar una solución válida y funcional.

Capítulo 4: Metodología

4.1 Metodología de trabajo

La metodología de trabajo utilizada en la empresa para el desarrollo del proyecto es el desarrollo ágil de software, en concreto una metodología Scrumban [9]. Se utilizan iteraciones de aproximadamente 2 semanas. En cada una de estas iteraciones se planifican pequeños objetivos a conseguir durante la iteración, se implementan las funcionalidades y/o soluciones para lograr esos pequeños objetivos, se realiza la corrección de errores de iteraciones pasadas y se genera el informe de dicha iteración. Todo esto con el fin de tener un feedback constante y en pequeños intervalos de tiempo.

Respecto a la organización de la empresa, Safelayer dispone de diferentes equipos de desarrolladores en función del producto, es por eso que se ha requerido de una constante comunicación entre los miembros de este. Esta comunicación ha sido dada mediante el uso del correo interno de la empresa, *daily-scrums* y demostraciones cada dos semanas presentando el trabajo que cada miembro del equipo ha desarrollado. Estas han sido las herramientas más usadas a nivel de comunicación entre los miembros del equipo. Dicho esto, el resto del equipo está implementando otras funcionalidades que por ahora no tienen relación alguna con el proyecto que se me ha asignado, debido a eso, la comunicación que voy a realizar con los demás equipos será muy limitada. Para trabajar en el código del proyecto usamos el sistema de repositorios de código Stash, que es el nombre del servidor interno en el que hay instalado una versión de BitBucket [10], un producto de Atlassian que se encarga de la gestión de versiones del código.

A nivel administrativo y de gestión de proyectos, se usa la famosa herramienta Jira [11]. Jira sirve de apoyo para la gestión de requisitos, seguimiento del estado del proyecto y de errores. El *Product Manager* de la empresa asigna las tareas a hacer en cada iteración y se deja constancia en Jira, donde cada

programador puede ver cuales son sus próximas tareas y/o las más prioritarias. Esta herramienta es muy importante de cara al desarrollo del proyecto, pues es la principal fuente de comunicación con el director del proyecto ya que le permite saber en cada momento las tareas que estoy realizando.

Además de Jira, se usa otro producto de Atlassian llamado Confluence [12], que sirve para la gestión de la documentación corporativa en forma de web. Dicha web contiene información sobre los procedimientos de desarrollo, diseño de los productos, guías de programación, material de formación, manuales de los productos, etc.

Por último, el ponente se encargará de la parte administrativa del proyecto y de la validación del trabajo, respetando la normativa de formato de la UPC.

4.2 Validación del proyecto

Para la realización de la validación del proyecto, se parte de la base de que el proyecto que se va a desarrollar es sólo una prueba de concepto de un producto de la empresa, ya que realizar todo el proyecto en el marco de tiempo establecido para el TFG es inviable. Dicho esto, la fuente de validación más usada en la empresa a nivel de software es la de testing y a nivel de gestión de proyecto es la validación por parte del project manager y/o tutor.

Cuando se habla de testing a nivel de software, se refiere tanto a la prueba de todos los componentes y/o módulos desarrollados (test unitarios) como de la prueba de las funcionalidades y/o cumplimiento de las interacciones entre dichos componentes (test funcionales) para el software que se está llevando a cabo.

Partiendo de que se irán desarrollando los componentes independientemente, la metodología que se seguirá en dicho proyecto es la realización de dichos tests a medida que se van desarrollando los componentes

avanzados y sus tests correspondientes. De este modo, se irá haciendo una validación progresiva de la implementación del software.

Referente a la validación del proyecto, gracias a que el tutor designado ejerce como profesional dentro de la empresa, habrá un flujo continuo de comunicaciones para que el proyecto se vaya validando y gestionando a medida que se está desarrollando. Antes de cualquier entrega de documentación, el tutor hará la correspondiente revisión con un cierto margen de tiempo para la posterior corrección del documento si es necesario. También, mencionar que gracias a la metodología de trabajo de la empresa también se irán realizando reuniones de seguimiento y validación del trabajo realizado.

Capítulo 5: Formación y puesta en marcha del proyecto

En este proyecto, a diferencia de otros con un propósito parecido pero con un enfoque distinto, se usan muchos conceptos que requieren de una cierta asimilación y formación. Dado que este proyecto está en el campo de la Seguridad Informática, se aconseja tener unas ciertas bases que esta sección pretende explicar. Por eso, la mayor parte de los primeros meses se ha estado asimilando todos estos conocimientos para poder ejecutar el proyecto de la mejor manera posible.

A continuación se explicarán los conceptos y las tecnologías que se irán viendo a lo largo del proyecto, serán el núcleo de éste y tendrán gran importancia para entender cada parte de él. Se empezará por explicar algunos conceptos en el ámbito de la Seguridad Informática como la Criptografía, la Infraestructura de Clave Pública y la Firma Digital. Seguidamente se procederá a explicar conceptos acerca de las páginas y servicios webs actuales y la implementación de estos.

Y finalmente se explicarán conceptos acerca de las *best practices* a seguir a la hora de desarrollar y desplegar el *software* en el *cloud* conjuntamente con las tecnologías / herramientas usadas para satisfacer dichas prácticas.

5.1 Introducción a la Seguridad Informática

Generalmente, la seguridad informática consiste en garantizar que el material y los recursos de software de una organización se usen únicamente para los propósitos para los que fueron creados y dentro del marco previsto. De una forma más técnica y usando su definición, la seguridad informática es una disciplina que se encarga de proteger la integridad y la privacidad de la información almacenada en un sistema informático. Pero, ¿cómo podemos saber que nuestro sistema es seguro? ¿cuándo se garantiza que éste está protegido? Estas preguntas son muy

difíciles de responder, y hay que decir que no existe ninguna técnica que permita garantizar la invulnerabilidad del sistema, pero sí existen términos que se usan para describir el grado de seguridad en un entorno como el que concierne a este proyecto. Estos términos son citados y explicados a continuación:

- **Confidencialidad**: esta propiedad asegura que los datos transmitidos sólo pueden ser interpretados por su destinatario, es decir, tienen que ser legibles únicamente para los usuarios autorizados.

- **Integridad**: la integridad de un sistema informático garantiza que los datos transmitidos no pueden ser modificados sin que esta modificación no pueda detectarse. Sólo personal autorizado podrá modificar estos datos.

- **No repudio**: la irrefutabilidad de un sistema garantiza que el autor de los datos transmitidos no pueden rechazarlos ni negar sus acciones.

- **Disponibilidad**: la última propiedad garantiza que los datos están disponibles en cualquier momento para su uso.

Así pues, observamos que para hacer servir estas propiedades, la información o los datos con los que tratamos deben estar protegidos y/o cifrados. Y es aquí donde entra en juego la criptografía.

5.2 Criptografía

La criptografía es el estudio de los métodos mediante los cuales se puede convertir la información, de su forma normal y comprensible, en otra forma que resulta incomprensible si no se dispone del conocimiento necesario. Este conocimiento necesario es en el que se basa la fiabilidad de los métodos criptográficos.

En el marco de este proyecto se denomina a este conocimiento clave y la seguridad de que la información es segura depende de que el conjunto de claves permanezcan a su vez seguras, ocultas y sean de confianza. Así pues la criptografía surge de la necesidad de intercambiar mensajes de forma que solamente sus destinatarios sean capaces de comprenderlos. Cabe destacar que también engloba a otras disciplinas como el criptoanálisis que se encarga de atacar un mensaje cifrado con el objetivo de conseguir descifrarlo. Cuando esto no es posible se considera que es un método seguro y que en ausencia de la clave no se podría recuperar la información.

Para el cifrado de la información se usan algoritmos matemáticos que convierten el texto en criptogramas (o secuencias de información cifrada) que resultan ilegibles y sólo pueden ser descifrados por quien conozca la clave. En las siguientes secciones se describen los dos métodos de criptografía que más se usan en el ámbito informático.

5.2.1 Criptografía Simétrica

La criptografía simétrica consiste en usar la misma clave tanto para cifrar (emisor) como para descifrar el mensaje (receptor), por eso se denomina simétrica. Su funcionamiento consiste en aplicar una serie de transformaciones al mensaje basándose en una clave, de forma que sea posible invertir las transformaciones si la clave es conocida. Los implicados en la comunicación deben ponerse de acuerdo de antemano sobre qué clave usar. Cuando ambos conocen la clave, el remitente cifra un mensaje usándola y lo manda al destinatario que lo descifra con la misma que él conoce. Para verlo de una forma más matemática se pueden usar las funciones de cifrados y descifrado:

$$P = e(k, M)$$
$$M = d(k, P)$$

La función de cifrado *e()* usa el parámetro *k*, que es la clave usada para la conversión del mensaje *M* en un nuevo mensaje *P*. El criptograma *P* sólo puede descifrarse con la función de descifrado *d()*, que dará como resultado el mensaje original *M* siempre y cuando se conozca la clave *k*.

Por lo que a la seguridad se refiere, un buen sistema de cifrado pone toda la seguridad en la clave y ninguna en el algoritmo. En otras palabras, no debería ser de ninguna ayuda para un atacante conocer el algoritmo que se está usando. Sólo si el atacante obtuviera la clave, le serviría para conocer el algoritmo. Dado que toda la seguridad está en la clave, es importante que sea muy difícil adivinar el tipo de clave.

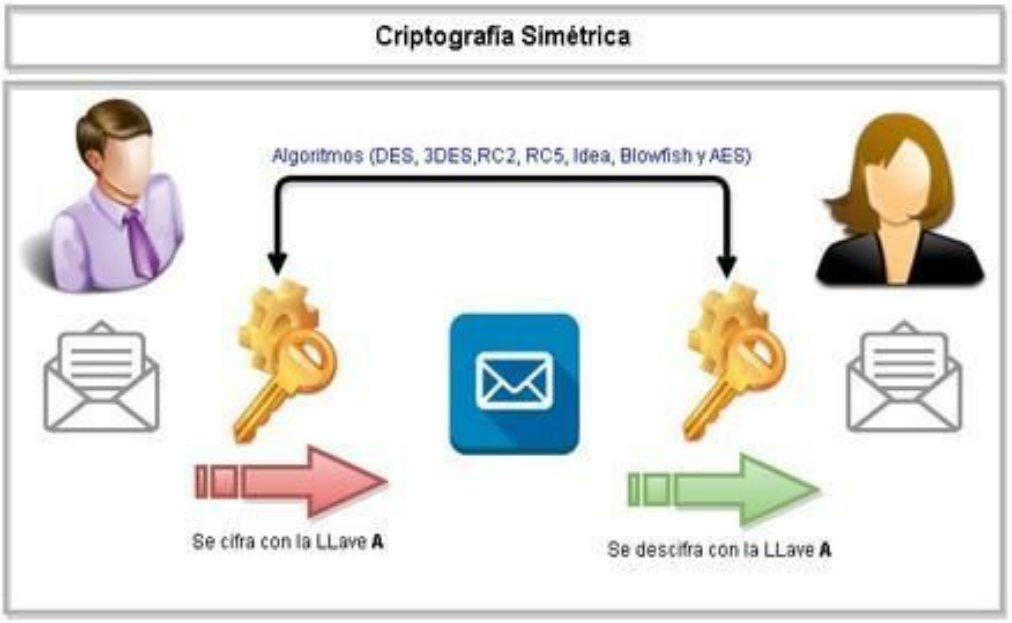


Figura 1. Cifrado con Criptografía Simétrica

Este método, aunque garantiza la confidencialidad de la información transmitida, presenta varios problemas:

- **Compartición de la clave:** Dado que, tanto el emisor como el receptor deben conocer la clave que se usará para el cifrado/descifrado del mensaje, es necesario que antes de iniciar la comunicación hayan intercambiado dicha clave a través de un medio de comunicación seguro.

- **Gran cantidad de claves:** Para cada grupo de entidades que deseen realizar una comunicación segura se deben acordar una clave compartida y por lo tanto son necesarias una gran cantidad de claves.

- **Falta de autenticación:** Dado que tanto emisores como receptores comparten la clave es imposible conocer quién ha sido el emisor de un mensaje o si el receptor es el que se esperaba.

- **Debilidad de algunos algoritmos:** Algunos algoritmos criptográficos simétricos son vulnerables a ataques de fuerza bruta o descifrado con métodos criptoanalíticos usando la potencia de los equipos actuales.

5.2.2 Criptografía Asimétrica

En criptografía asimétrica, cada entidad dispone de un par de claves, de las cuales una de ellas se publica para que todo el mundo la pueda conocer (clave pública) y la otra se almacena de forma segura (clave privada). Ambas claves están relacionadas matemáticamente y se generan a la vez, pero computacionalmente es imposible, a partir de la clave pública, hallar la privada. La gran mejora de esta técnica es que la clave privada e solo conocida por el propietario, mientras que la pública puede distribuirse al resto de los usuarios, sin que ello comprometa de ningún modo la confidencialidad de la clave privada. Así mismo, las funciones de cifrado y descifrado se puede representar matemáticamente como el caso anterior:

$$P = e\ (k'_{pub}, M)$$
$$M = d\ (k'_{prv}, P)$$

La función de cifrado *e()* usa la clave pública *k' pub* del destinatario para la conversión del mensaje *M* en un nuevo mensaje *P*. El criptograma *P* sólo puede descifrarse con la función de descifrado *d()*, usando la clave privada *k' prv*, que sólo debería conocer su propietario.

En este tipo de cifrados, uno de los inconvenientes es la eventual pérdida de la clave privada, ya que imposibilita el posterior descifrado de los mensajes. Así mismo, para una misma longitud de clave y mensaje se necesita mayor tiempo de proceso. Otra desventaja es que las claves deben ser de mayor tamaño que las simétricas (generalmente son cinco o más veces de mayor tamaño que las claves simétricas) y por lo tanto el mensaje cifrado ocupa más espacio que el original.

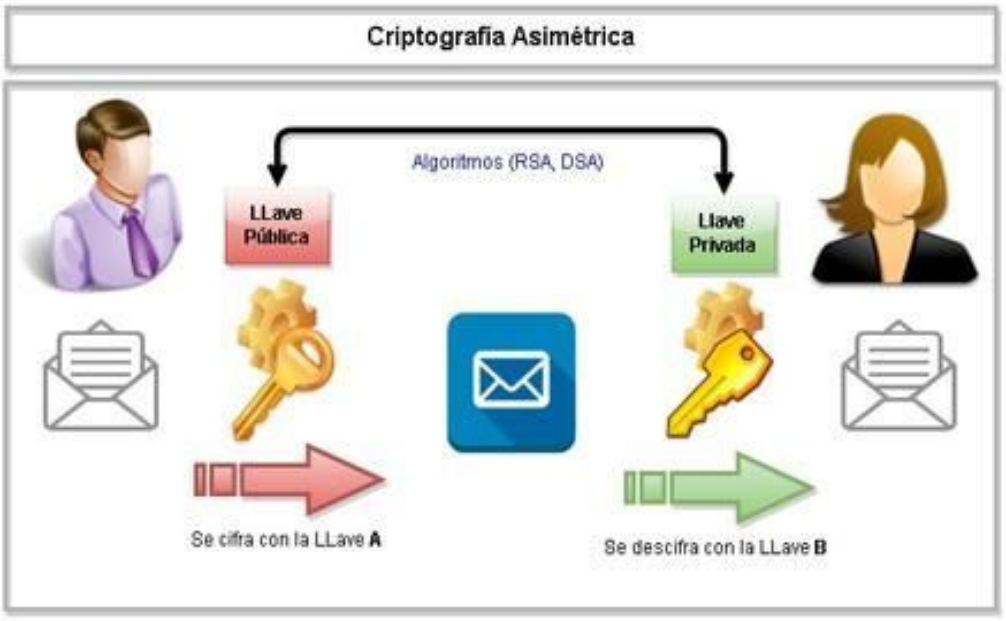


Figura 2: Cifrado con Criptografía Asimétrica

Una vez vistos los dos tipos de cifrado y las ventajas e inconvenientes de cada uno, podemos comparar los dos tipos de criptografía:

- **No intercambio de la clave:** En criptografía asimétrica, sólo es compartida la clave pública por lo que la clave privada se mantiene realmente en secreto. Además solo es necesario una clave pública y una privada por usuario a

diferencia del número de claves que son necesarias en la criptografía simétrica.

- **Integridad, autenticidad y no-repudio:** En criptografía asimétrica, con la infraestructura adecuada, es posible asegurar la confidencialidad además de permitir identificar los remitentes, garantizar el no-repudio del mensaje y asegurar que los datos se han mantenido inalterados durante el envío.

- **Mayor tiempo de proceso:** En criptografía asimétrica, para una misma longitud de clave, como los métodos matemáticos son más complejos es necesario un mayor tiempo de cálculo. Además el menaje una vez cifrado con este método es de mayor tamaño que el original, a diferencia de la criptografía simétrica.

Una vez visto qué tipos de criptografía existen, en qué se basan y las principales semejanzas y diferencias entre ellas, es hora de dar paso a la infraestructura de clave pública, un sistema de encriptación basado en un clave pública y en una clave privada.

5.2.3 Infraestructura de clave pública (PKI)

En términos de criptografía, una infraestructura de clave pública (Public Key Infrastructure) es una combinación de hardware, software y políticas y procedimientos de seguridad, que permiten la ejecución con garantías de operaciones criptográficas. Este sistema, permite vincular las claves públicas con sus respectivas entidades. De este modo, un organismo externo en el cual confían las partes implicadas garantiza que una clave pública pertenece a una identidad. Las operaciones criptográficas de clave pública utilizan unos algoritmos de cifrado conocidos y accesibles para todos. Por eso, la seguridad proporcionada por la tecnología PKI, está mayormente ligada a la privacidad de la clave privada y la políticas de seguridad aplicadas.

La tecnología PKI permite a los usuarios autenticarse frente a otros usuarios y usar la información de los certificados de identidad (por ejemplo, las claves públicas de otros usuarios) para cifrar y descifrar mensajes, firmar digitalmente información, garantizar el no repudio de un envío, u otros usos.

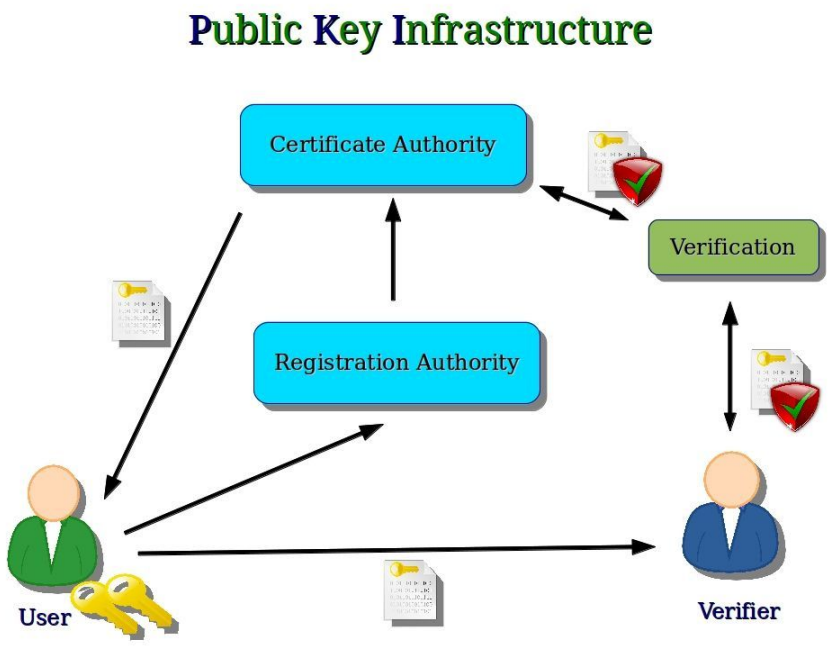


Figura 3: Infraestructura de Clave Pública

A continuación se explicarán con un poco más de detalle algunos elementos que componen esta infraestructura:

5.2.3.1 Certificado digital

Un certificado digital es una estructura de datos usada para transmitir la información relacionada con una entidad, en especial su identidad y clave pública, donde una tercera parte de confianza da fe de esa relación, en nuestro caso una Autoridad de Certificado o CA (la cual vamos a explicar seguidamente). Existen varios formatos de certificados, pero el más extendido es el estándar UIT-T X509, comúnmente codificados en DER (Distinguished Encoding Rules) o PEM (Privacy-enhanced Electronic Mail). Los certificados digitales deben estar firmados por la CA que los ha emitido para asegurar su validez. Solo las CA denominadas raíz firman el certificado con la misma entidad que están representando.

Los certificados digitales, contienen más información en él además de la firma del mismo. A continuación se cita y explica esta información:

- **Número de serie:** actúa de identificador único del certificado

- **Nombre del titular:** el titular subject expresado como *DN (Distinguished Name)* el cual está formado por el *CN (Common Name)*, *OU (Organizational Unit)*, *O (Organization)* y *C (Country)*.

- **Información sobre el certificado**: información sobre su versión, la CA firmante del certificado (*Issuer*), las fechas de emisión y expiración del mismo, la metodología usada para la emisión del certificado, etc.

- **Clave pública:** La clave pública para cifrar un mensaje para este usuario o validar una firma del mismo así como el algoritmo usado para crearla.

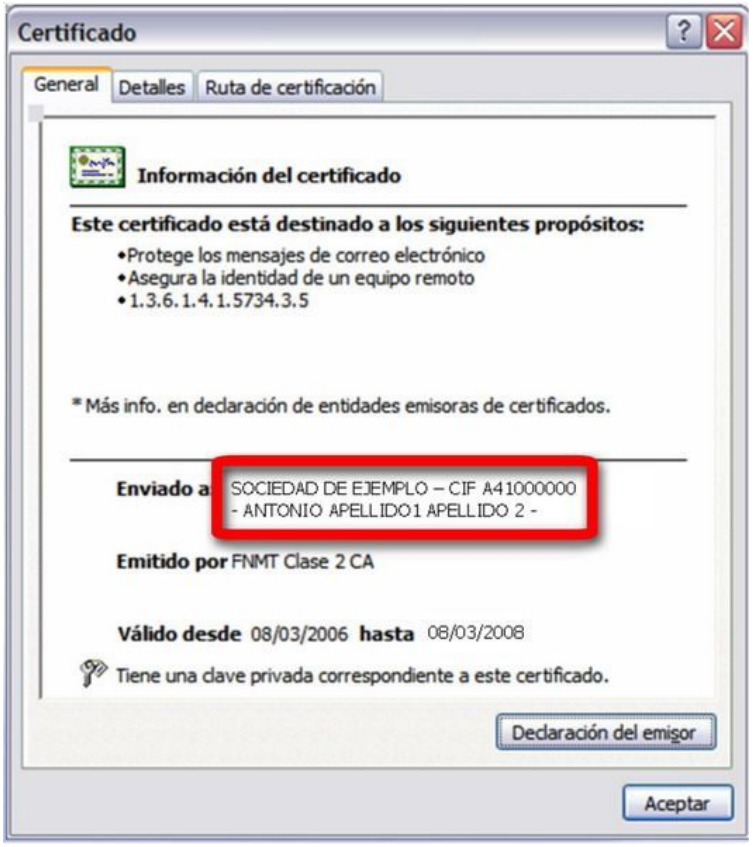


Figura 4: Visualización de un Certificado Digital

5.2.3.2 Autoridad de Certificación

La Autoridad de Certificación o CA es la que tiene como principal objetivo emitir y revocar certificados digitales. También se encarga de renovar los certificados cuando estos caducan o son revocados. Es por eso, que la confianza hacia la autoridad de certificado es el fundamento de toda infraestructura PKI.

Cuando se quiere confiar en una CA, es suficiente con instalarse en el sistema de la entidad final el certificado de la CA y, sólo a partir de ese momento, el sistema confiará en esa. Además, la CA dispone de su propio certificado y claves pública y privada. Las usa para firmar los certificados solicitados y, a la vez, las CAs se pueden organizar de manera jeràrquica, donde el primer elemento de cualquier jerarquía se denomina CA raíz.

5.2.3.3 Autoridad de Registro

La autoridad de Registro o RA es la autoridad que tiene como principal objetivo conectar o comunicar las entidades que solicitan certificados con las autoridades de certificación. Debe verificarse los datos proporcionados por las entidades solicitantes y tramitar la petición del certificado a la CA correspondiente. Una vez el certificado se haya generado deberá entregarlo a la entidad solicitante. Para la solicitud de certificados una RA envía una petición *PKCS#10 (Public-Key Cryptography Standards)*, que como su nombre en inglés define, es un estándar que define los formatos que deben tener los datos o la información que esté de una manera u otra ligada con la infraestructura de clave pública, es decir, incluye la información del solicitante, su clave pública y un fragmento firmado con su clave privada.

5.2.3.4 Repositorio de Certificados

El Repositorio de Certificados, como su propio nombre indica, es un repositorio que contiene los certificados válidos para que las entidades que lo requieran puedan descargarlos. Suele estar implementado como directorios X.500 accesible mediante LDAP (LightWeight Directory Access protocol) [4], que es un

conjunto de protocolos abiertos usados para acceder a información guardada centralmente a través de la red.

5.2.3.5 Entidad final

Por entidad final, se entiende al organismo, individuo o sujeto que dispone de un certificado con sus propias clavesa para hacer uso de la infraestructura de clave pública o PKI. Por claves entendemos la pública, visible por todo el mundo y la privada que sólo el organismo o sujeto conoce. El uso de PKI ya puede ser para firmar un certificado, validarlo, etc.

5.2.4 Firma digital

El proceso de firma digital es muy parecido al de firmar un documento por escrito. En el último caso coges un bolígrafo y pones tu firma, por ejemplo. En el caso digital pasa algo parecido pero en vez de un bolígrafo físico se usa un proceso diferente que a continuación se explica. Partimos de un mensaje M que queremos enviar:

- El emisor calcula el hash* del mensaje M y lo cifra con su clave privada.
- El emisor envía el mensaje M junto con la firma del hash al receptor.
- El receptor, a través de la clave pública del emisor, descifra el hash que ha recibido cifrado.
- EL receptor, una vez descifrado, obtiene un hash.
- A continuación el receptor calcula el hash del mensaje M y si coincide con el hash obtenido al descifrar, se considera el mensaje como auténtico. Si no, significa que ha sido alterado.

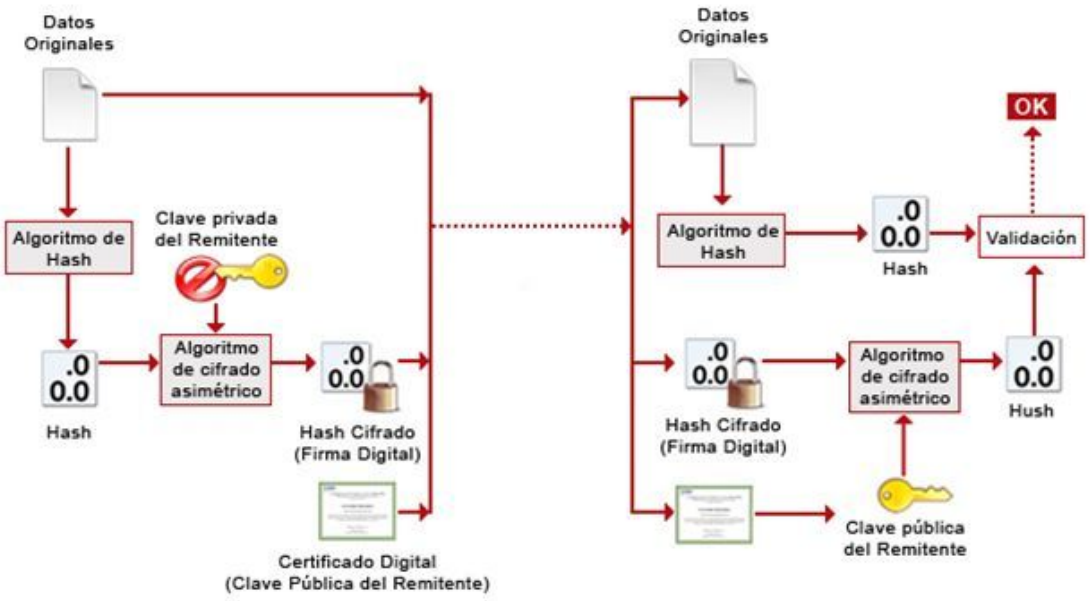


Figura 5: Proceso de firma digital

---

*Los hash o funciones de resumen son algoritmos que consiguen crear a partir de una entrada una salida alfanumérica de longitud normalmente fija que representa un resumen de toda la información que se le ha dado [13].

5.2.4.1 Tipos de firma

Como hemos visto anteriormente, un certificado se firma digitalmente por una entidad o persona. Ahora bien, hay diferentes maneras o formas de hacer el proceso de firmado, donde básicamente, se relaciona la ubicación de la firma con la de los datos. A continuación veremos los principales tipos de firma:

- **Firma separada (detached):** El contenedor de la firma digital o firmas digitales no incluye el archivo de datos que se ha firmado.

- **Firma envolvente (enveloping):** El contendor de la firma digital o firmas digitales incluye el archivo de datos que se ha firmado.

- **Firma incrustada (enveloped):** El valor de la firma está incluido en el archivo que contiene los datos firmados.

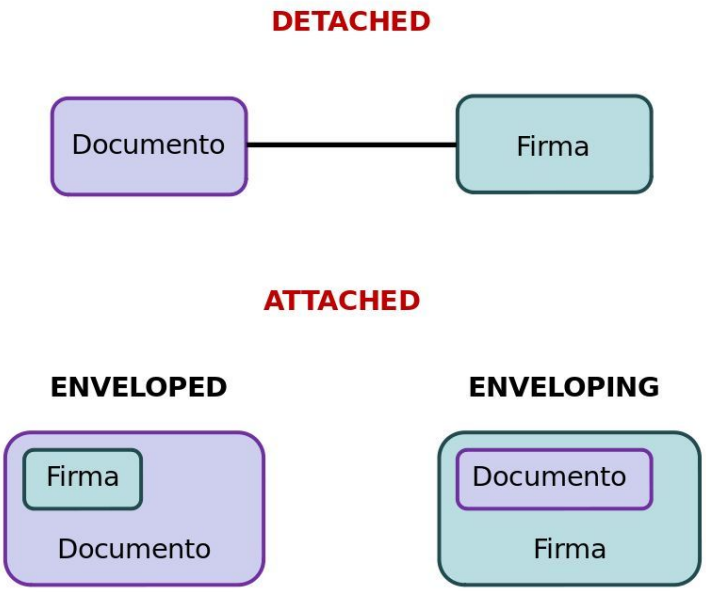


Figura 6: Tipos de firma digital

Una vez vistos los principales tipos de firma, cabe recalcar que existen también las multifirmas, donde l ainformación se firma por más de una entidad o persona. En este tipo de firmas, cuando cada entidad sólo firma los datos, hablamos de firma en paralelo. En cambio, si cada entidad firma los documentos firmados por las entidades anteriores (excepto la primera entidad que no tendrá firma anterior), se dice que es una firma en serie.

5.2.5 Cifrado

Por último, para cerrar esta breve introducción a la seguridad informática hablaremos del proceso de cifrado de información. Como se ha visto anteriormente, el cifrado de los datos es un proceso crucial para garantizar la integridad de los datos, donde sólo el usuario receptor de esos datos ser capaz de procesar esa información. Para poderlo garantizar, existen unos pasos a seguir:

- ❏ El emisor consigue la clave pública del receptor y cifra los datos con dicha clave.
- ❏ El emisor envía los datos cifrados.
- ❏ Una vez recibido el mensaje, el receptor descifra los datos con su clave pública.

Hasta aquí la introducción a la seguridad informática y a sus conceptos. Hemos visto de qué se trata, los fundamentos básicos de la criptografía y de las infraestructuras de clave pública. Además, también se ha repasado el concepto de firma digital, y por último, el de cifrado.

5.3 Aplicaciones Web

Una aplicación web es un programa informático que realiza una función específica mediante el uso de un navegador web como cliente en vez de ser ejecutados por el sistema operativo como en el caso de las aplicaciones de escritorio. La aplicación puede ser tan simple como un tablero de mensajes o un formulario de contexto en una página web o tan compleja como un procesador de texto [14].

Las aplicaciones web son populares debido a que presentan muchos beneficios como podrían ser la ligereza de la aplicación ejecutada en el navegador web, a la independencia del sistema operativo, así como a la facilidad de actualizar y mantener dichas aplicaciones sin distribuir e instalar software a miles de usuarios potenciales.

Para desarrollar dichas aplicaciones normalmente se usa un lenguaje scriptado en el servidor (Python, Ruby, Go, etc) para desarrollar todo tipo de servicios necesarios para la interfaz como podrían ser servicios de autentificación y/o comunicación con la base de datos. Dicho lenguaje en el servidor se combina con un lenguaje scriptado en el cliente como podrían ser HTML, Javascript, etc para generar toda la estructura e estilado de la interfaz como la interacción del usuario.

5.3.1 Single Page Applications

Las aplicaciones de página única o Single Page Applications (SPA de ahora en adelante) es un nuevo concepto y/o práctica que ha surgido en los últimos años en el mundo del desarrollo de las aplicaciones web. Las SPA son aplicaciones web donde la mayoría de interacciones con el usuario están manejadas en la parte del

cliente sin la necesidad de estar siempre comunicándose con el servidor, dando fluidez a la experiencia del usuario. [15]

Aunque se tenga la percepción de estar navegando por páginas separadas de la aplicación, la página no se encarga en ningún punto del acceso ni se transfiere a otra página. Visualmente solo se actualizan los datos obtenidos por peticiones hechas al servidor y no se realiza un procesado de toda la página web, es aquí donde se introduce el nuevo concepto del contenido dinámico. Por tanto, las SPA son aplicaciones web que contienen todo su código HTML en una sola página.

5.3.2 React & Redux

Cuánto más interactividad se produce en el cliente, más dependencia a código JavaScript se produce para garantizar un funcionamiento correcto y/o fluido en el navegador web. También hay que tener en cuenta que cuánto más código se deba escribir mejor y más importante será tener un código limpio y bien estructurado. Y ese es el problema que intentan solventar cada uno a su manera los frameworks Javascript.

Hay una larga lista de frameworks Javascript en el mercado como podrían ser React, Angular, Ember.js, Vue.js etc.. Pero me voy a centrar principalmente en el primero de ellos debido a que la empresa anteriormente a mi incorporación realizó un estudio sobre las tecnologías y creyó conveniente seguir esta línea.

5.3.2.1 React

React fue implementado por Facebook para solventar problemas de rendimiento, eficiencia, reusabilidad, etc. en las interfaces de usuario en aplicaciones web de gran escala [16]. La proposición de dicho framework era muy diferente de las tendencias a las que se dirigía el resto de la comunidad JavaScript en ese momento. Mientras que otros autores de frameworks se enfocaban en aplicar el patrón MVC para escribir aplicaciones y tener una separación clara de código asociado a la vista y otras partes de la aplicación web, React propuso unirlas

y simplificar el código mediante la composición y otros paradigmas funcionales a través de componentes.

React fue diseñado con el concepto de los componentes reusables, en los que se puede llegar a definir pequeños componentes y/o acabar juntando varios de ellos para formar componentes más grandes y/o complejos.

Todo componente en React dispone de una propiedad llamada **props** que se podría definir como los atributos de configuración para dicho componente que son recibidos desde un nivel superior y por definición son inmutables, es decir, un componente no puede cambiar sus propias props.

También disponen de un estado, **state**, que podríamos definir como una representación del componente en un punto en el tiempo, una instantánea del componente. El **state** se iniciará con un valor por defecto pero este valor si podrá mutar durante la vida del componente. No tenemos porqué definir estados para un componente, son opcionales y al contrario que ocurre con las props, un componente puede gestionar su propio estado.

Todo componente también dispone de un ciclo de vida, hay principalmente tres fases: una fase donde se crea el componente, una intermedia en la que las propiedades y el estado del componente cambian y finalmente una fase en la que el componente se desmonta.

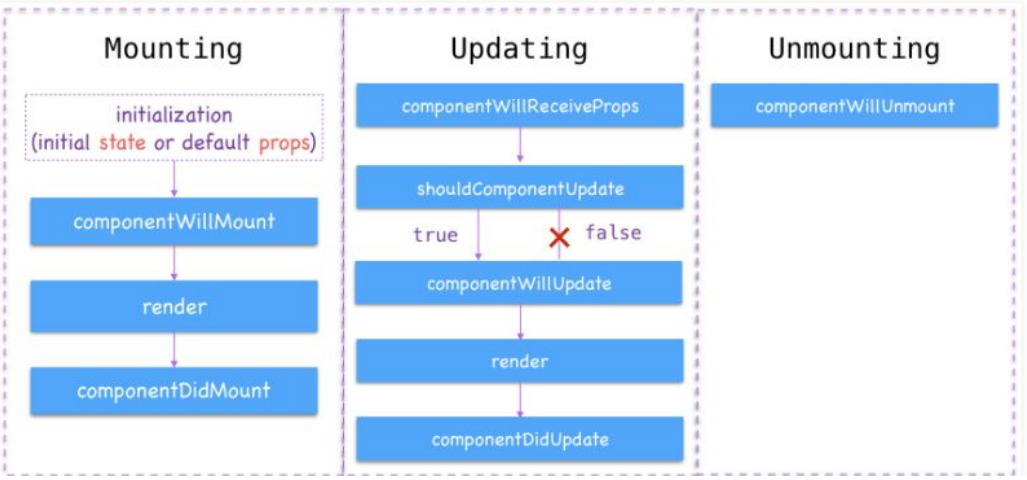


Figura 7: Ciclo de vida de un componente de React

Desde el nacimiento de React otros frameworks populares como Angular y Ember.js también han adoptado una arquitectura basada en componentes, así como otras ideas similares como el DOM virtual, que es como React aplica los cambios en la vista de la SPA.

El enfoque general de React para crear aplicaciones es diferente a Angular y Ember porque principalmente solo se encarga de las vistas, no proporciona una forma de enrutar el contenido en el cliente o incluso una forma de llamar y/o cargar datos desde el servidor. Esto significa que para hacer implementar una aplicación simple pero totalmente funcional (que le hagan falta las características que hemos comentado anteriormente) se tendrá que buscar otras librerías de terceros.

Sin embargo, no se tiene que descartar dicho framework por la dependencia de librerías externas debido a que gracias al auge actual de participación en proyectos open source hay muchas bibliotecas populares respaldadas por la comunidad que incluso han sido adoptadas por equipos dentro de Facebook.

5.3.2.2 Redux

Entre dichas librerías externas está Redux, que sirve para controlar el estado de nuestras aplicaciones web fácilmente, de una forma consistente entre cliente y servidor, testeable y con una gran experiencia en desarrollo [17]. Redux está en gran parte influenciado por la arquitectura Flux propuesta por Facebook para las aplicaciones de React, pero también se puede utilizar con otros frameworks/librerías JavaScript como Angular, Backbone, etc. Redux se basa en tres principios:

- **El estado es de solo lectura**: La única forma de modificar el estado es emitir una acción que indique qué cambió.

```
/**
 * Devuelve una acción de tipo ADD_TODO
 * @param  {String} text Texto del TODO
 * @return {Object}      Objecto de acción
 */
function addTodo(text) {
    return {
        type: 'ADD_TODO',
        payload: {
            text,
        },
    };
}
```

Figura 8: Ejemplo de una acción

- **Los cambios se hacen mediante funciones puras**: Para controlar como el store es modificado por las acciones se usan reducers puros. En la siguiente imagen podemos observar un ejemplo de reducer:

```
// cargamos el método de Redux para
// poder combinar reducers
import { combineReducers } from 'redux';

function todos(state = [], action) {
  switch(action.type) {
  case 'ADD_TODO':
    // creamos una copia del state actual
    const copy = Array.from(state);
    // modificamos lo que necesitamos
    copy.push(action.payload.text)
    // retornamos el nuevo state
    return copy;
  default:
    // si el action.type no existe o no concuerda
    // con ningunos de los casos definidos
    // devolvemos el estado sin modificar
    return state;
  }
}

// combinamos nuestros reducers
// los keys que usemos para nuestros reducers
// van a ser usados como keys en nuestro store
// en este ejemplo sería: { todos: [], }
const reducers = combineReducers({
    todos,
});

export default reducers;
```

Figura 9: Ejemplo de un reducer de Redux

- **Una sola fuente de la verdad**: Todo el estado de tu aplicación está contenido en un único store.

```
// cargamos la función para crear un store
import { createStore } from 'redux';
// cargamos nuestros reducers (ya combinados)
import reducers from './reducers.js';

// definimos el estado inicial
const initialState = {
    todos: [],
};

// creamos el store
const store = createStore(reducers, initialState);

export default store;
```

Figura 10: Ejemplo de una store de Redux

Seguidamente mostraremos el funcionamiento que sigue Redux usando los conceptos clave que acabamos de explicar cuando se realiza una interacción en un componente de la aplicación web. Una vez vista la tecnología que utilizaremos en nuestra aplicación web, hablaremos del concepto de los servicios web y de su comunicación con la SPA.

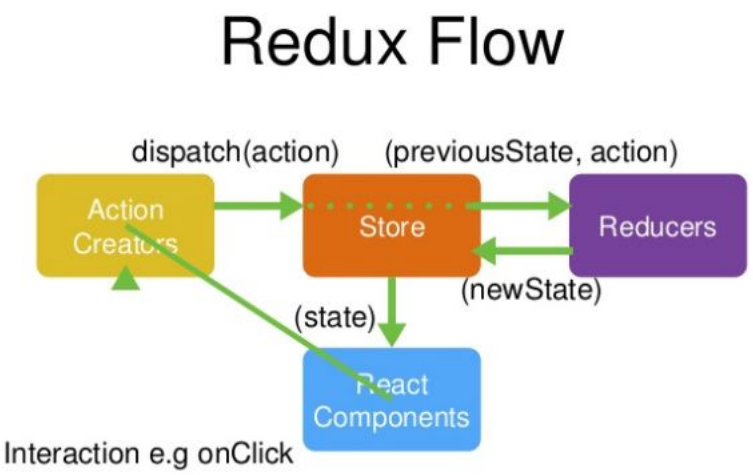


Figura 5: Arquitectura Redux

5.4 Servicios Web

Existen múltiples definiciones sobre lo que son los Servicios Web, lo que muestra su complejidad a la hora de dar una adecuada definición que englobe todo lo que son e implican. Una posible sería hablar de ellos como un conjunto de aplicaciones y/o tecnologías con capacidad para interoperar en la Web. Estas aplicaciones y/o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitar un servicio llamando a estos procedimientos a través del cliente Web [18].

Se pueden utilizar para integrar aplicaciones escritas en diferentes lenguajes y que se ejecuten en distintas plataformas. Estos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para mostrar información dinámica al usuario. Para proporcionar

interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura orientada a servicios.

Actualmente existen muchos mecanismos de comunicación estándares por lo que vamos a contemplar los más relevante para nuestro proyecto:

- **XML (Extensible Markup Language)**: formato estándar para los datos que se vayan a intercambiar.

- **SOAP (Simple Object Access Protocol)**: protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

- **HTTP (Hypertext Transfer Protocol)**: protocolo de comunicación que permite las transferencias de información en la *World Wide Web.*

- **REST (Representational State Transfer)**: arquitectura que haciendo uso del protocolo HTTP proporciona una API (*Application Programming Interface*) que utiliza cada uno de sus métodos (GET, POST, PUT, DELETE, etc.) para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio web y el cliente (navegador web).

5.4.1 Arquitectura de los servicios web

Como hemos mencionado anteriormente, para garantizar y/o proporcionar interoperabilidad y extensibilidad entre los servicios es necesaria una arquitectura. En el mundo del desarrollo de servicios web principalmente existen dos alternativas principales: la arquitectura monolítica o la arquitectura de microservicios [19].

5.4.1.1 Arquitectura monolítica

En la arquitectura monolítica, la aplicación (servicio web) está implementado y/o se ejecuta en una única unidad. Para realizar cualquier cambio en el sistema, el

desarrollador debe compilar y desplegar toda la parte del servidor (*back-end*), por lo que en un sistema de gran envergadura puede ser muy susceptible a errores y a problemas a la hora de garantizar una alta disponibilidad. A parte del inconveniente mencionado, cabe mencionar otros problemas que soluciona la arquitectura de microservicios:

- ❏ Las aplicaciones monolíticas pueden evolucionar hasta una magnitud en la que ningún desarrollador entienda la totalidad de la aplicación.

- ❏ Se produce una reutilización limitada.

- ❏ Escalar aplicaciones monolíticas a menudo puede ser un desafío para el desarrollador y/o grupo de desarrolladores.

- ❏ Por definición, las aplicaciones monolíticas se implementan utilizando un único conjunto de tecnologías (JEE, .NET..), esto puede llegar a limitar al equipo de desarrollo.

5.4.1.2 Arquitectura de microservicios

Las capacidades de microservicio se expresan formalmente con API's orientadas a los negocios. Encapsulan una capacidad empresarial básica, y como tal son activos valiosos para el negocio. La implementación del servicio, que puede implicar integraciones con sistemas de registro, está completamente oculta ya que la interfaz se define puramente en términos comerciales. El posicionamiento de los servicios como activos valiosos para el negocio los promueve implícitamente como adaptables para su uso en múltiples contextos. El mismo servicio se puede reutilizar en diferentes canales comerciales o puntos de contacto digitales, según las necesidades. Las dependencias entre los servicios y sus consumidores se minimizan aplicando el principio de acoplamiento flexible.

Al estandarizar los contratos expresados a través de API orientadas a los negocios, los consumidores no se ven afectados por los cambios en la

implementación del servicio. Esto permite a los propietarios del servicio cambiar la implementación y modificar los sistemas de registro o las composiciones de servicio que pueden estar detrás de la interfaz y reemplazarlos sin ningún impacto posterior.

Una arquitectura de microservicio, junto con las tecnologías de implementación en la nube, la administración de API y las tecnologías de integración, proporciona un enfoque diferente para el desarrollo de software. En su lugar, el monolito se divide en un conjunto de servicios independientes que se desarrollan, implementan y mantienen por separado. Esto proporciona las siguientes ventajas:

- ❏ Se recomienda que los servicios sean pequeños.

- ❏ Si las interfaces de los microservicios se exponen con un protocolo estándar, como una API REST-ful, pueden ser consumidos y reutilizados por otros servicios y aplicaciones sin acoplamiento directo.

- ❏ Los servicios existen como artefactos de implementación independientes y se pueden escalar independientemente de otros servicios.

- ❏ Desarrollar servicios independientes permite a los desarrolladores utilizar las tecnologías de desarrollo apropiadas para la tarea en cuestión.

La compensación de esta flexibilidad es la complejidad. Gestionar una multitud de servicios distribuidos a escala es difícil por las siguientes razones:

- ❏ Los equipos de proyecto deben descubrir fácilmente los servicios como potenciales candidatos para la reutilización. Estos servicios deberían proporcionar documentación, tests, etc. para que la reutilización sea mucho más fácil que implementar el servicio de cero.

- Se deben monitorizar las interdependencias entre servicios. El tiempo de inactividad del servicio, las actualizaciones de los servicios, etc. pueden tener efectos en cascada y dicho impacto debe analizarse previamente.

Es importante asegurarse de que el despliegue de los microservicios se administre cuidadosamente y de que esté lo más automatizadamente posible. La falta de coordinación y automatización del equipo provocará más problemas que beneficios en la arquitectura de microservicios establecida.

5.4.2 Golang Programming Language (Go)

El lenguaje de programación Go fue lanzado en Noviembre 2009, desarrollado e ideado por Google. Es un lenguaje moderno, que ha adquirido lo mejor de muchos otros lenguajes. Entre sus diseñadores iniciales aparecen nombres como Rob Pike, ingeniero de software canadiense conocido por su trabajo en Laboratorios Bell donde junto con su equipo desarrolló el sistema operativo UNIX, y Kenneth Lane Thompson, pionero en las ciencias de la computación que se le atribuye también la creación de UNIX en los Laboratorios Bell.

Seguidamente se van a detallar las características principales de este lenguaje de programación:

- La sencillez es la característica principal de Go. Con una sintaxis clara y concisa, combina una sintaxis parecida a C con las características y facilidades de lenguajes dinámicos como Python. Lenguajes como C++, Java o C# son más pesados o voluminosos.

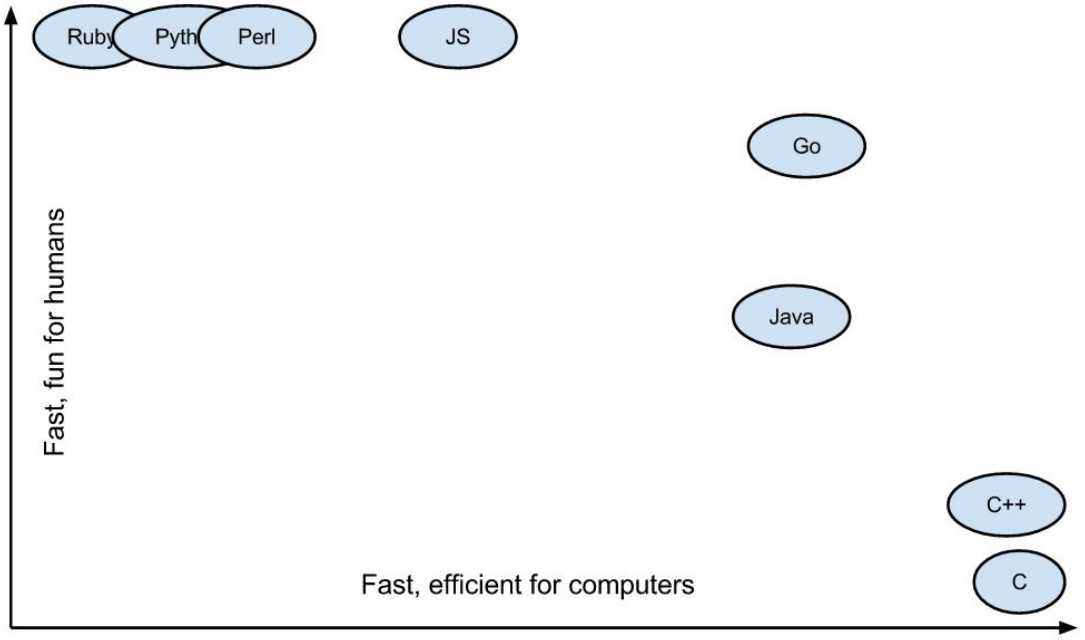


Figura 12: Gráfico comparación sencillez Go

- Aún estando diseñado para la programación de sistemas, provee de un recolector de basura (*garbage collector*), reflexión, potentes patrones de concurrencias y otras capacidades de alto nivel que lo convierten en un lenguaje muy potente.

- Go admite el paradigma de programación orientado a objetos, pero a diferencia de los lenguajes de programación más populares no dispone de herencia de tipos y tampoco de palabras clave que denoten claramente que soporta este paradigma.

- Un detalle que puede resultar confuso es que la definición de un tipo "clase" se realiza por medio de declaraciones separadas (*interfaces, structs, embedded values*..)

- Go permite el uso de delegación a través de *embedded values* y poliformismo por medio de interfaces.

- También cabe destacar que tiene el concepto de concurrencia incorporado, permite el paralelismo de una forma más fácil. Go, mediante el concepto de goroutines permite realizar tareas simultáneas y el concepto de canales para permitir tanto la comunicación como la sincronización.

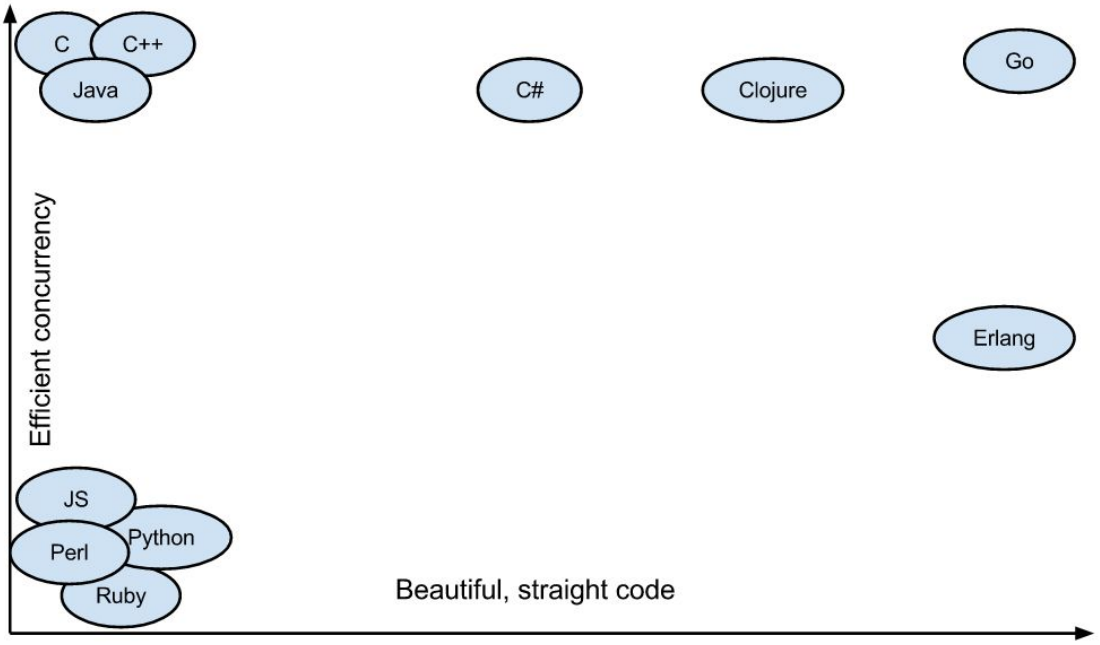


Figura 13: Gráfico comparación lenguaje Go

Como hemos podido ver, Go, es un lenguaje de programación que proporciona una serie de características que lo hacen posicionarse como uno de los lenguajes de programación más potentes de hoy en día, cabe decir, que aún no tiene detrás una gran comunidad como podría ser la de Java y/o C++ pero gracias a ser respaldada por Google y gracias a que la documentación es una de sus *features* principales, cada vez se irán sumando más y más desarrolladores.

5.4.3 Auth0: Servicio de autenticación

Auth0 es una empresa que proporciona un software de autenticación y autorización como servicio (SaaS). Ofrecen tanto a los desarrolladores como a las empresas los componentes básicos que necesitan para proteger sus aplicaciones sin tener que convertirse en expertos en seguridad ni entrar en detalles. Proporciona muchas ventajas y seguidamente vamos a detallar unas cuantas de ellas:

- Permite conectar cualquier aplicación, escrita en cualquier lenguaje informático, a Auth0 y definir los proveedores de identidad que se desean utilizar, es decir, cómo se desea que los usuarios inicien sesión en la aplicación.

- Dispone de distintos SDK en función de la tecnología de la aplicación o también una API para poder conectar la aplicación con dicho servicio. Y una vez configurada la aplicación, cualquier usuario que quiera autenticarse, Auth0 se encargará de verificar su identidad y enviar la información requerida a dicha aplicación.

- Permite a los usuarios autenticarse con un usuario/contraseña, es decir, gestionar una base de datos de usuario propia o autenticarse con cuentas de redes sociales (Facebook, Twitter y muchas más) y obtener los datos de su perfil de dicho usuario.

- Permite a organizaciones conectar su servicio de directorio empresarial (LDAP, Active Directory..) existente para permitir a los empleados que inicien sesión en las diversas aplicaciones con sus credenciales empresariales.

- Permitir tanto a aplicaciones web como aplicaciones móviles, en caso de que se desarrollen por separado, autenticarse con seguridad a la API / servicio web.

- Permitir acceder a una solución interactiva con una interfaz gráfica a la gestión de los usuarios que se identifiquen con tu aplicación.

- Permite forzar un segundo método de autenticación (*two-factor authentication*) cuando los usuarios quieran autentificarse en la aplicación y se considere que los datos o acciones a realizar son sensibles y/o importantes.

5.4.3.1 Estándares de identidad de Auth0

Cuando las máquinas eran sistemas independientes, toda la autenticación y los datos del usuario se almacenaban en una sola máquina. Los tiempos han cambiado y ahora se puede usar la misma información de inicio de sesión en múltiples aplicaciones y sitios. Esto fue gracias a los estándares de la industria de la identidad que fueron ampliamente adoptados en la web.

Se trata de un conjunto de especificaciones y protocolos abiertos que le indican cómo diseñar un sistema de autenticación y autorización. Le dicen cómo debe administrar la identidad, mover los datos personales de forma segura y decidir quién puede acceder a las aplicaciones y a los datos.

Auth0 utiliza muchos de estos estándares pero solo se van a mencionar los que se han creído relacionados con este proyecto:

- **OAuth1:** el estándar original para la delegación de acceso. Se utiliza como una forma para que un usuario otorgue a los sitios web acceso a su información en otros sitios web o aplicaciones, pero sin darles las credenciales.

- **OAuth2:** un estándar de autorización que permite a un usuario otorgar acceso limitado a su información de un sitio web a otro sitio web sin tener que exponer sus credenciales. Se está utilizando dicho estándar cada vez que inicia sesión en un sitio utilizando su cuenta de Google y se le pregunta si acepta compartir su dirección de correo electrónico y su lista de contactos con dicho sitio.

- **OpenID Connect:** es una capa de identidad que se encuentra en la parte superior de OAuth2 y permite una fácil verificación de la identidad del usuario,

así como la capacidad de obtener información básica del perfil del proveedor de identidad.

- **JSON Web Token (JWT):** un estándar que define una forma compacta y autónoma para transmitir información de forma segura entre distintos sitios como un objeto JSON. En la imagen siguiente podemos ver en la izquierda, apartado Encoded, un JWT codificado y en la parte derecha el mismo JWT descodificado y dándonos información del propietario de dicho token.

Figura 14: Ejemplo JSON Web Token

5.5 Containerization

El concepto de *containers* o contenedores, son una solución al problema de cómo hacer que el software se ejecute sin problemas cuando se traslada de un entorno informático a otro [20]. Esto podría ser desde la computadora portátil de un desarrollador a un entorno de prueba, desde un entorno de transición a producción y tal vez desde una máquina física en un centro de datos a una máquina virtual en una nube privada o pública.

De ahí surge el concepto de la contenedorización de aplicaciones, es un método de virtualización a nivel de sistema operativo utilizado para implementar y ejecutar aplicaciones distribuidas sin iniciar una máquina virtual (VM) completa para cada aplicación [21]. Múltiples aplicaciones o servicios aislados se ejecutan en un solo *host* y acceden al mismo núcleo del sistema operativo. Los contenedores funcionan en sistemas simples, instancias de nube y máquinas virtuales, tanto en sistemas Linux como en sistemas operativos Windows y Mac.

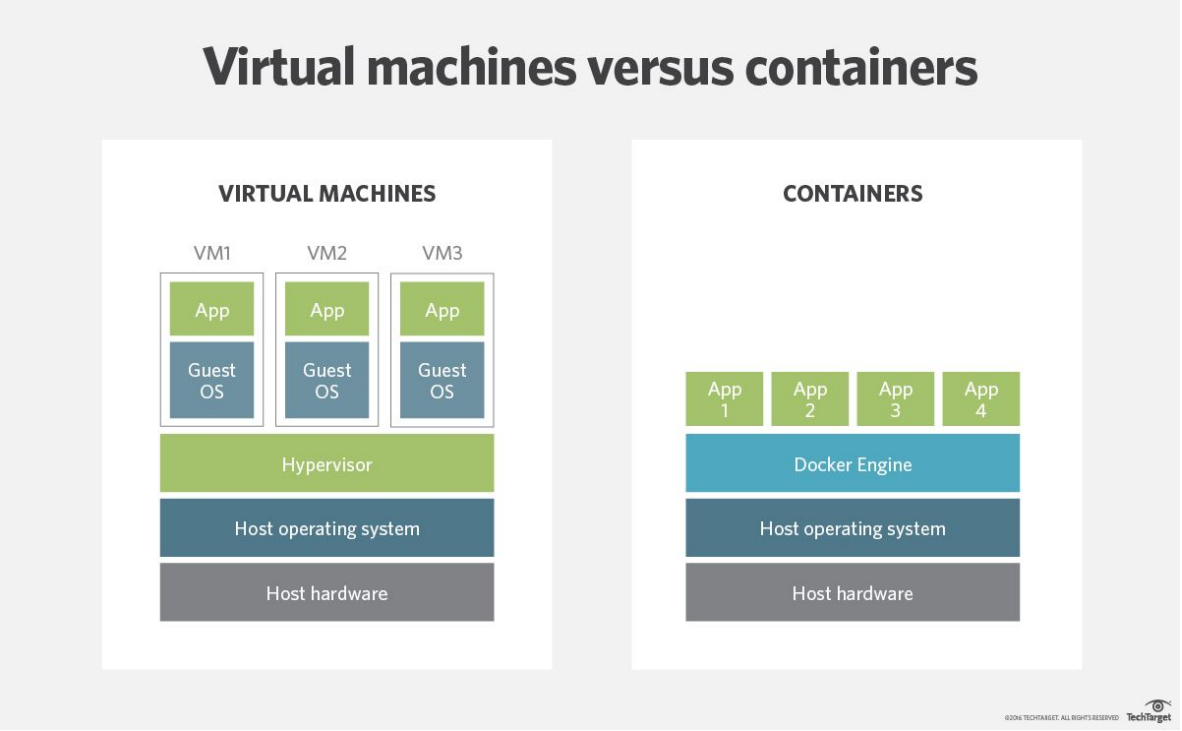


Figura 15: Comparación entre máquinas virtuales y contenedores

Todo esto es posible gracias a que un contenedor de aplicaciones consta de un entorno de tiempo de ejecución completo: una aplicación, todas sus dependencias, librerías, archivos binarios y los archivos de configuración necesarios para ejecutarlo, agrupado todo en un solo paquete o imagen. Al contener la plataforma de aplicaciones y sus dependencias, las diferencias en las distribuciones del sistema operativo y la infraestructura se abstraen. De este modo, los contenedores consumen menos recursos que una implementación comparable en máquinas virtuales porque los contenedores comparten recursos sin un sistema operativo completo para respaldar cada aplicación. [22]

5.5.1 Docker & Docker Hub

Docker es una herramienta diseñada para facilitar la creación, implementación y ejecución de aplicaciones mediante el uso de contenedores. [23] Como hemos mencionado anteriormente, los contenedores permiten a un desarrollador empaquetar una aplicación con todas sus dependencias. Al hacerlo, gracias al contenedor, el desarrollador puede estar seguro que la aplicación se ejecutará en cualquier sistema operativo, independientemente de las configuraciones personalizadas que la máquina pueda tener y que pueda interferir en el funcionamiento del software.

En cierto modo, Docker es parecido a una máquina virtual. Pero a diferencia de éste, en lugar de crear un sistema operativo virtual completo, Docker permite que las aplicaciones utilicen el mismo kernel de Linux que el sistema en el que se ejecutan y solo requiere que las aplicaciones se envíen con cosas que aún no se estén ejecutando en la máquina. De este modo Docker proporciona una mejora significativa en el rendimiento y reduce el tamaño de la aplicación.

Cabe mencionar que Docker es un proyecto *open source,* es decir, que cualquiera puede contribuir al proyecto y extender sus funcionalidades dependendiendo de las necesidades en caso de que no estén implementadas.

Dichas características hacen a Docker fácil de usar y único, le otorgan a los desarrolladores un acceso total a las aplicaciones, la capacidad de implementar el software rápidamente centrándose en el código y sin preocuparse del sistema operativo en el que se ejecutará, facilitar el control de las versiones y de su distribución ya que también existe una plataforma llamada Docker Hub [24] en la que puedes almacenar todas tus imágenes compiladas y compartirlas con los demás usuarios. Por todos estos motivos, se ha decidido seguir esta práctica y ejecutar los servicios web y el *frontend* de la aplicación con esta herramienta.

5.6 Continuous Integration

El concepto de *continuous integration* o integración continua [25] es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central, como Git, de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. La integración continua se refiere en su mayoría a la fase de creación o integración del proceso de publicación de software y conlleva un componente de automatización como Travis CI [26] y un componente cultural, como podría ser la práctica de integrar con frecuencia.

Dicho concepto surge del problema causado antiguamente en la que distintos miembros del equipo trabajaban aislados durante un largo periodo de tiempo y posteriormente combinasen los cambios en la versión final una vez que habían completado el trabajo.

Los objetivos clave y los beneficios de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones del software.

Existen distintas herramientas que permiten a los desarrolladores seguir esta práctica. En mi proyecto se ha decidido usar GitHub, como repositorio de código, y Travis CI, como herramienta para la automatización y la monitorización de las integraciones en el código.

5.6.1 GitHub & Gitflow

GitHub es un servicio web para alojar proyectos de código basado en el sistema de control de versiones Git. Además ofrece otras funcionalidades como control de acceso y varias funciones de colaboración, como seguimiento de errores, solicitudes de funciones, administración de tareas y documentación para cada proyecto.

Cuando un usuario está trabajando en un proyecto, tendrá varias funciones o ideas en progreso en un momento dado, algunas de las cuales están listas para empezar y otras no. Para administrar este flujo de trabajo existe un sistema de ramas (*branch* en inglés). Cuando se crea una nueva rama en el proyecto, se está creando un entorno en el que se puede probar a implementar nuevas funcionalidades sin que haya repercusiones en el código fuente original anterior a la creación de dicha rama. Se recomienda tener al menos tres tipos de ramas:

- **Master**: Es la rama principal. Contiene el repositorio que se encuentra publicado en producción, lo que que debe estar siempre estable.

- **Development**: Es una rama que creada a partir de Master. Es la rama de integración, todas las nuevas funcionalidades se deben integrar en esta rama. Posteriormente a la integración y a la comprobación de errores, se puede proceder a hacer una fusión (*merge*) sobre la rama master.

- **Features**: Cada nueva funcionalidad se debe utilizar en una rama nueva, específica para esa funcionalidad. Estas se deben sacar de development. Una vez que la funcionalidad esté desarrollada, se puede proceder a hacer un *merge* sobre la rama development, donde se integrará con las demás funcionalidades.

5.6.2 Travis CI

Travis CI es un servicio de integración continua alojado y distribuido utilizado para crear y probar proyectos de software alojados en GitHub. Se configura mediante un archivo llamado *.travis.yml* en la raíz del proyecto. Dicho archivo especifica mediante formato YAML, el entorno de compilación y los *tests* deseados, entre otros parámetros como pueden ser las dependencias del software para que se pueda compilar y/o ejecutar.

Cuando ya se ha activado Travis CI para un repositorio dado, Github lo notificará cada vez que se envíen nuevos commits a ese repositorio. También se puede configurar para que solo se ejecute para *branches* específicas. Travis CI revisará la rama correspondiente y ejecutará los comandos especificados en el fichero *.travis.yml* que generalmente compilan el software y ejecutan cualquier prueba automatizada. Cuando dicho proceso haya sido completado, se notifica al desarrollador de la manera en la que se configuró para hacerlo (p.ej: email).

Travis CI se puede configurar para ejecutar las pruebas en una variedad de máquinas diferentes, con diferentes programas instalados (como versiones anteriores de una implementación) y admite la compilación de software en una gran variedad de lenguajes informáticos.

5.7 Continuous Delivery

El concepto de *Continuous Delivery* o entrega contínua [27] es una extensión de la integración continua para garantizar que se puedan lanzar nuevos cambios a sus clientes de forma rápida y sostenible. Esto significa que, además de automatizar sus pruebas, también ha automatizado su proceso de lanzamiento y puede implementar su aplicación en cualquier momento haciendo clic en un botón.

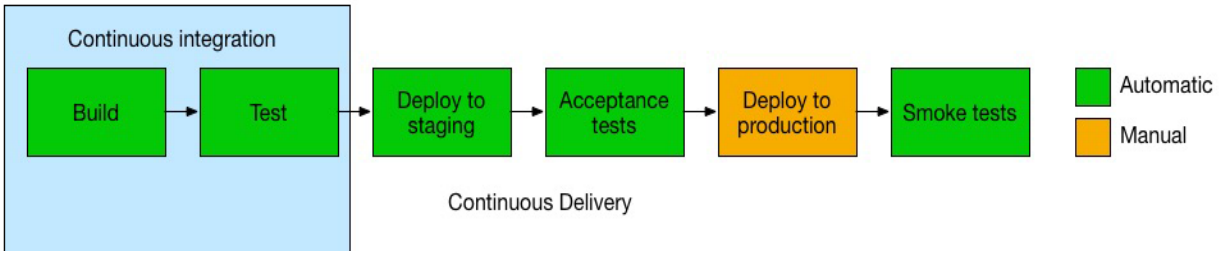


Figura 16: Procesos de la entrega contínua

Con la entrega continua, se puede decidir publicar el software diariamente, semanalmente o lo que se ajuste a los requisitos del negocio y/o cliente. Sin embargo, si se desea obtener los beneficios de la entrega continua, se debe implementar en producción lo antes posibles para asegurarse de desplegar el software en pequeños lotes, que son fáciles de solucionar en caso de que surja un problema.

5.7.1 Amazon Web Services

Tanto para el despliegue de las aplicaciones web como para el de los microservicios existen distintos proveedores y/o servicios que te permiten realizar dicho proceso de una forma más sencilla y/o automatizada. En nuestro caso, tanto para mi proyecto como para otros productos de la empresa se ha optado por el conjunto de servicios que proporciona Amazon Web Services.

Amazon Web Services [28], de ahora en adelante AWS, es una colección de servicios de computación propiedad de Amazon que en conjunto forman una plataforma en la nube con un amplio abanico de herramientas listas para usar en la gestión de diferentes elementos dentro de la empresa. Los servicios de AWS están preparados tanto para autónomos, como pequeñas y medianas empresas o grandes corporaciones, ya que existen posibilidades para escalar las instancias o el almacenamiento según vaya creciendo la empresa.

AWS está situado en distintas regiones geográficas repartidas por el mundo, cada región está totalmente contenida dentro de un solo país y todos sus datos y servicios permanecen dentro de la región designada.

Cada región tiene múltiples zonas de disponibilidad que son los diferentes centros de datos que proporcionan servicios de AWS. Las zonas de disponibilidad están aisladas unas de otras para evitar la propagación de cortes entre las zonas. Varios servicios operan a través de zonas de disponibilidad, mientras que otros pueden estar configurados para reproducirse a través de zonas para extender la demanda y evitar el tiempo de inactividad de los fallos.

Como hemos mencionado anteriormente, la *suite* de AWS ofrece una gran variedad de servicios, seguidamente vamos a definir y explicar un poco los más relevantes para el desarrollo de este proyecto:

- **Amazon Virtual Private Cloud (VPC):** este servicio permite aprovisionar una sección en la nube de AWS aislada de forma lógica, en la que se pueden lanzar otros servicios, como los que vamos a mencionar a continuación, en una red virtual que define el usuario/cliente. Permite configurar todos los aspectos del entorno de redes virtual como el rango de direcciones IP, la creación de subredes, etc…

- **Amazon Elastic Compute Cloud (EC2):** es un servicio web que proporciona un entorno de capacidad y tamaño modificable para el desarrollo, testeo y gestión de aplicaciones y programas en la nube. Con este servicio se reduce el tiempo necesario para obtener y arrancar nuevas instancias de servidor en cuestión de minutos, lo que permite escalar rápidamente la capacidad, ya sea aumentándola o reduciéndola, en función de las necesidades.

- **Amazon Elastic Load Balancing (ELB):** es un servicio que distribuye automáticamente el tráfico de aplicaciones entrantes a través de varios destinos, tales como instancias de EC2, contenedores y direcciones IP.

Elastic Load Balancing ofrece tres tipos de equilibradores de carga: el balanceador de carga de aplicaciones (equilibrio de carga del tráfico HTTP y HTTPS), un balanceador de carga de red (equilibrio de carga del tráfico TCP) y un balanceador de carga clásico (una mezcla entre los dos anteriores).

- **Amazon Relational Database Service (RDS):** es un servicio administrador de base de datos relacionales, es decir, permite configurar, utilizar y escalar una base de datos relacional en la nube de una forma muy sencilla. Igual que en la EC2, proporciona capacidad modificable para facilitar la escalabilidad y toda la automatización de la gestión de backups.

- **Amazon Simple Storage Service (S3):** es un servicio de almacenamiento de objetos creado para almacenar y recuperar cualquier volumen de datos desde cualquier cliente, ya sean sitios web, aplicaciones móviles o dispositivos IoT. Garantizando siempre un alto nivel de durabilidad, disponibilidad y escalabilidad de los datos almacenados.

- **Amazon Cloudfront:** es un servicio de entrega de contenido, *content delivery network (CDN),* global con la que puedes llegar a proporcionar datos, vídeos, aplicaciones web, etc. de forma segura a los usuarios finales ofreciendo una baja latencia y altas velocidades de transferencia. Se puede integrar fácilmente con otros servicios de Amazon como S3, Elastic Load Balancing o EC2 para utilizarlos como orígenes de aplicaciones.

- **Amazon Route 53:** es un servicio web DNS escalable y de alta disponibilidad en la nube. Ofrece un método de redirección de los usuarios finales a las aplicaciones en internet convirtiendo nombres legibles en direcciones IP numéricas, es decir, su función principal es conectar las solicitudes de los usuarios con la infraestructura en ejecución en AWS.

Capítulo 6: Desarrollo del proyecto

6.1 Especificación del software

Vistos los objetivos principales y los conceptos tecnológicos relacionados con este proyecto vamos a ver exactamente el comportamiento que se desea que realicen los componentes avanzados:

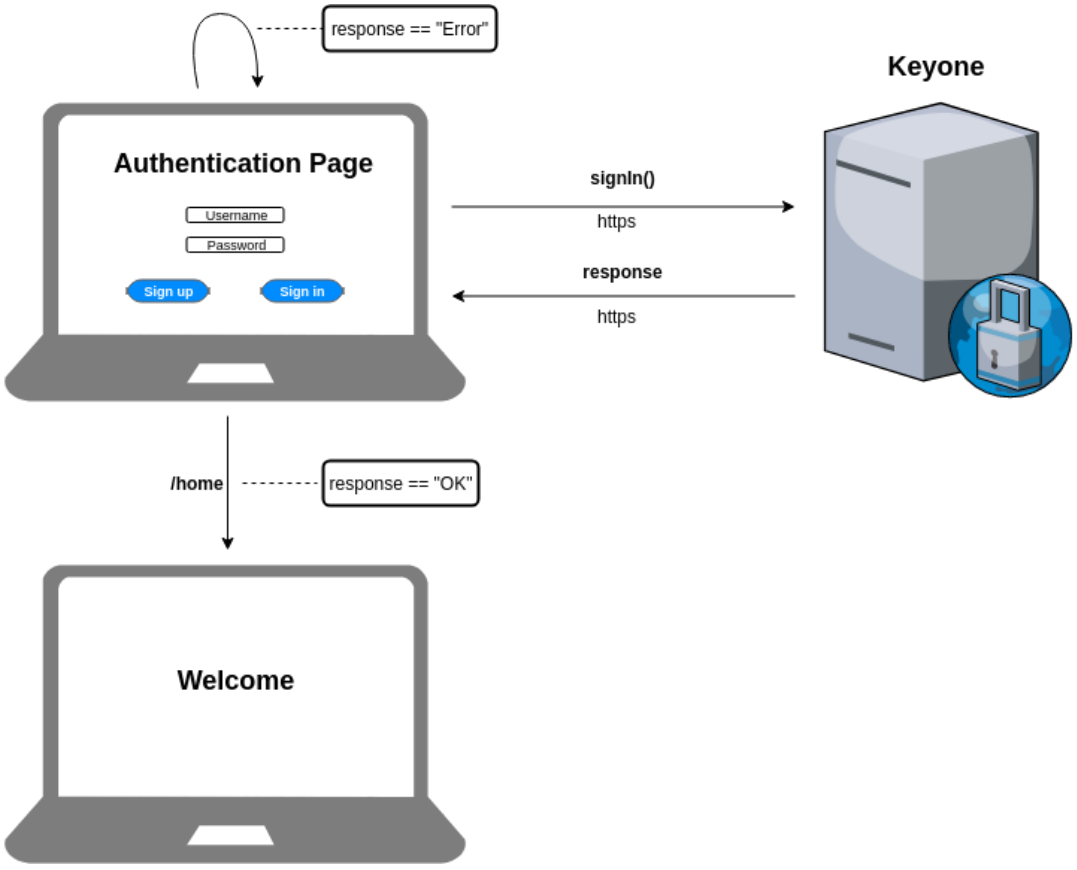


Figura 17: Comportamiento autenticación portal web

Como podemos ver en la imagen anterior la idea principal es muy simple, desde la página web se necesita poder autenticar y/o registrar a los usuarios del portal web con KeyOne, el software/producto de Safelayer encargado de gestionar los certificados. En caso de que la petición de autentificación sea incorrecta nos devolverá a la página web de autenticación, en caso contrario, permitirá al usuario acceder al portal web y a sus funcionalidades.

Después de la etapa de formación en la empresa y en sus productos vimos que existía una limitación tecnológica en el producto mencionado anteriormente y es que de momento solo acepta autenticaciones con certificados digitales que deben instalarse directamente en el navegador del cliente. Por lo que se ha tenido que pensar en cómo implementar una alternativa para que los componentes avanzados sigan satisfaciendo los principales objetivos del proyecto, es decir, permitir a los usuarios autenticarse y/o registrarse en la plataforma mediante redes sociales o mediante credenciales de su empresa/organización.

6.1.1 Requisitos funcionales de los componentes avanzados

Se define como requisito funcional las características requeridas del sistema o las funcionalidades expresadas de forma verbal. En este caso los componentes avanzados deben:

- ❏ Ser capaces de registrar usuarios mediante redes sociales
- ❏ Ser capaces de autenticar a usuarios mediante redes sociales
- ❏ Ser capaces de registrar usuarios de organizaciones / corporaciones
- ❏ Ser capaces de autenticar usuarios de organizaciones / corporaciones
- ❏ Ser capaz de visualizar la cuenta del usuario autenticado
- ❏ Definir un protocolo de comunicación entre el *frontend* y el *backend*
- ❏ Dar la posibilidad de desplegar la aplicación tanto *on-premises* como *as-a-service*

6.1.2 Requisitos no funcionales de los componentes avanzados

Por requisitos no funcionales se entienden aquellas exigencias que se imponen como restricciones a la hora de realizar la aplicación. Es decir, un requisito que especifica criterios que pueden usarse para juzgar la operación de una sistema en lugar de sus comportamientos específicos. La aplicación por lo tanto debe:

- ❏ Ser eficiente
- ❏ Ser usable, es decir, ofrecer una buena experiencia de usuario
- ❏ Seguir unas pautas de diseño
- ❏ Ser altamente segura
- ❏ Ser estable y fácil de mantener
- ❏ Ser internamente modular y escalable
- ❏ Asegurar una alta disponibilidad y rendimiento
- ❏ Ser automáticamente testeable.

Tanto los requisitos funcionales como los no funcionales se han ido debatiendo a lo largo del proyecto conjuntamente con el tutor y el director de este.

6.2 Diseño del software

6.2.1 Diseño de los componentes de la aplicación web

Como hemos dicho en la descripción del proyecto, actualmente, el portal web contiene una serie de funcionalidades que interactúan directamente con el software de Safelayer, KeyOne. Pero dicho portal web no contiene ningún método de autenticación sino que como hemos dicho anteriormente el usuario debe tener un certificado digital para poder realizar cualquiera de las acciones.

Es por ello que se ha tenido que diseñar una solución de como poder añadir dicho componente de gestión de credenciales para los usuarios y adaptar las funcionalidades de la aplicación para que las pruebas de integración existentes siguieran funcionando sin ningún problema. Y la solución propuesta ha sido la siguiente:

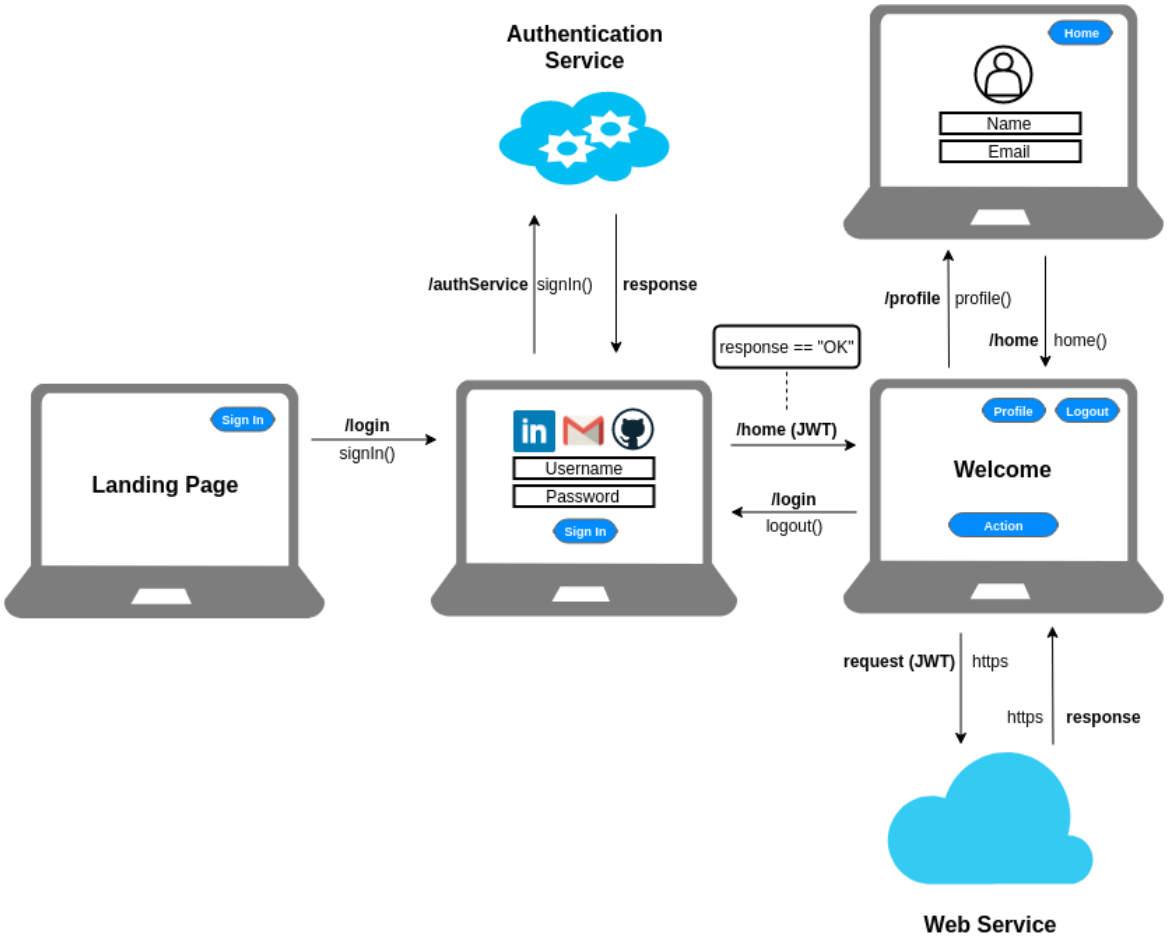


Figura 18: Proceso de autenticación en el portal web

Como podemos ver en la imagen anterior habrá que implementar una *landing page* la cual será una página web de bienvenida para los usuarios en la cual tendrán una opción para poder autenticarse.

En caso de que el usuario desee autenticarse dispondrá de una vista en la página web que le ofrecerá distintas opciones de autenticación, mediante redes sociales (LinkedIn, Gmail y/o Github) o mediante credenciales de su empresa. El funcionamiento de los componentes de dicha página web serán integrados y/o

configurados con el servicio de autenticación Auth0 mencionado anteriormente. Dicho servicio será el encargado de validar las credenciales tanto corporativas como las provenientes de las redes sociales.

Una vez el usuario haya introducido sus credenciales, en caso de que la validación sea satisfactoria se le permitirá acceder el portal web para poder visualizar su perfil y/o realizar cualquiera de las acciones disponibles en dicha plataforma. Todas las acciones implicarán realizar una petición HTTPS al servicio implementado encargado de autenticar todas las peticiones y que explicaré con detalle a continuación.

6.2.2 Diseño de los componentes del servicio web

Como hemos mencionado anteriormente me topé con un problema tecnológico entre la comunicación de la plataforma web y KeyOne, el producto de la empresa, que debía solventar para poder realizar la autenticación correctamente. Para ello se decidió implementar un servicio web que haría de intermediario entre las peticiones de la página web y dicho software empresarial, principalmente debía encargarse de procesar y/o formatear las peticiones para posteriormente mandarlas a KeyOne.

En vez de diseñar un servicio web monolito que tuviese muchas funcionalidades se ideó una alternativa siguiendo una arquitectura de microservicios y separar las funcionalidades para delegar el trabajo a realizar y mejorar así la escalabilidad y/o el rendimiento. Dichos microservicios, de ahora en adelante JWTProxy y DBProxy, se encargarán de realizar unas funcionalidades determinadas y a la vez funcionar como proxys inversos:

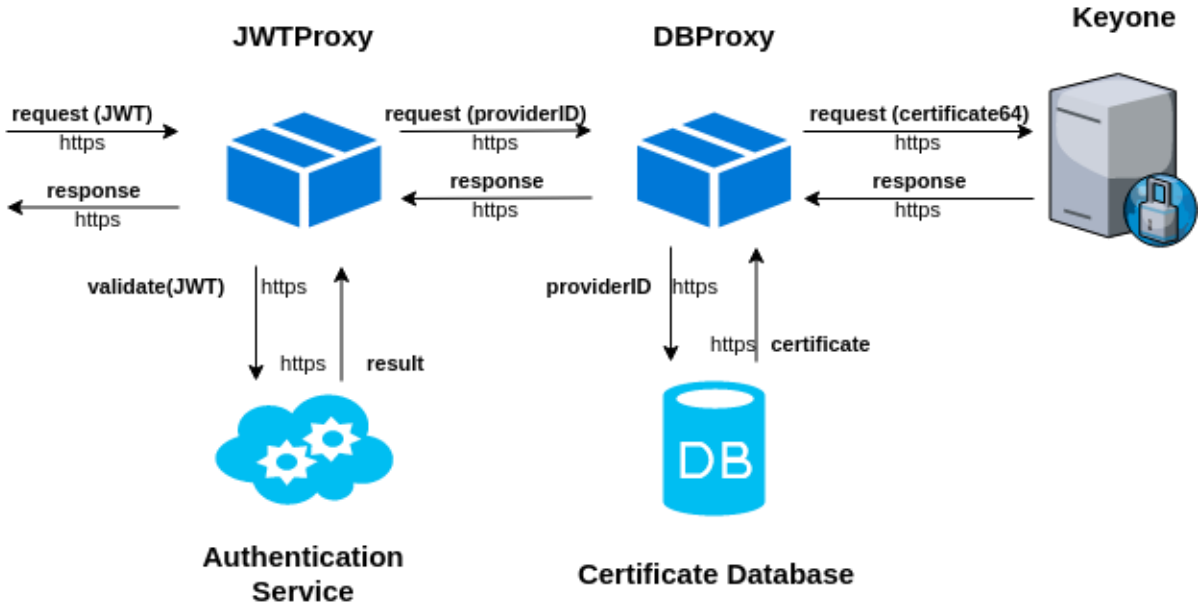


Figura 19: Proceso peticiones a los microservicios

Por un lado, JWTProxy será el servicio que se encargará de recibir peticiones HTTPS con un JWT (JSON Web Token) como cabecera. Su funcionalidad principal será la extracción de dicho token de la petición para poder validarlo con el servicio de autenticación y confirmar que el token ha sido firmado/generado por nuestra aplicación web y no otra, ya que cualquier usuario con un poco de conocimientos podría generar un JWT y realizar peticiones a nuestro servicio pero no todos los tokens serían válidos.

En caso de que la validación con el servicio de autenticación fuese erróneo se notificará al usuario que ha realizado la petición con un error detallado. En caso contrario, se descodificará dicho token para extraer de él un identificador único, que lo llamaremos providerID. Posteriormente se redirigirá la petición HTTPS original al segundo servicio diseñado sustituyendo el valor de la cabecera, que pasará a contener el valor de dicho identificador único.

Por otro lado, DBProxy será el servicio que se encargará de recibir peticiones HTTPS con un providerID como cabecera. Su funcionalidad principal será la extracción de dicho valor de la petición para posteriormente poder realizar una

petición a una base de datos. Dicha base de datos contendrá todos los certificados validados por KeyOne, el software de la empresa, y que estarán identificados mediante el providerID.

Una vez realizada la petición a la base de datos para consultar la existencia de dicho usuario, en caso de que no haya sido satisfactoria se notificará a dicho usuario. En caso contrario, se hará una conversión en base 64 de dicho certificado para poder redirigir la petición HTTPS original a KeyOne con el certificado en base 64 como cabecera.

Como hemos dicho anteriormente, dichos microservicios funcionan como proxys inversos así que una vez realizada la petición a KeyOne, la respuesta de dicha petición irá haciendo el camino inverso hasta llegar al cliente que estará utilizando la plataforma web.

6.3 Arquitectura del software

6.3.1 Arquitectura del Frontend

En la imagen posterior podemos ver un diagrama de la arquitectura del Frontend en la nube junto al flujo de lo que sucede cuando un usuario accede a la página web, primero de todo se explicará qué servicios se han debido usar en Amazon para lograr dicho funcionamiento y posteriormente se explicará paso a paso todo el procedimiento.

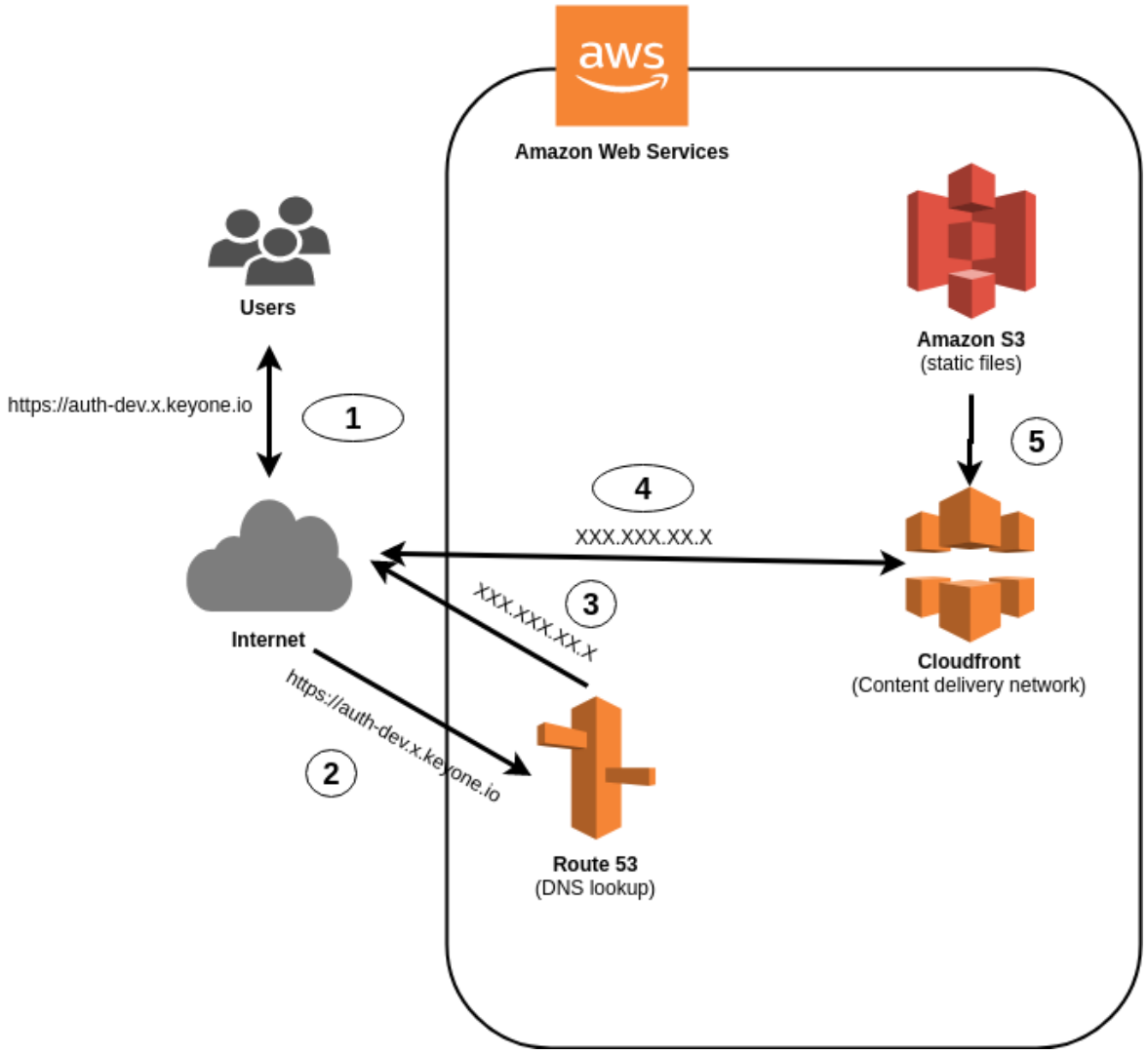


Figura 20: Arquitectura de los servicios del Frontend

Para la realización de esta arquitectura se tuvo que crear y configurar una serie de servicios de Amazon que son los siguientes:

- **S3:** se tuvo que crear un *bucket*, una unidad de almacenamiento de dicho servicio, en la que se almacenará el contenido (html, javascript, imágenes..) de nuestra página web.

- **Cloudfront:** para procesar y servir los ficheros almacenados en S3 se tuvo que crear una distribución de Cloudfront para que se encargará de ello, asociando dicha instancia al origen de datos, es decir, al *bucket* S3 creado anteriormente.

- **Route 53:** cuando se crea una distribución de Cloudfront, a éste se le asocia un *Domain Name,* es decir, un identificador en la red de forma aleatoria. Pero en nuestro caso nos interesaba acceder a la página web como "auth-dev.x.KeyOne.io", es por ello que se decidió utilizar este servicio para crear el enrutamiento de dicha URL a la distribución de cloudfront creada anteriormente.

6.3.1.1 Caso de uso: Usuario accede a la página web

1. El usuario introduce la URL en su navegador y realiza una petición HTTPS.
2. El proveedor de internet procesa el DNS de la URL introducida y la resuelve redirigiendo al servicio web DNS de Amazon llamado Route 53.
3. Route 53 resuelve el DNS y devuelve la dirección IP asociada.
4. Se redirige la petición a la dirección IP obtenida, asociada al servicio de red de entrega de contenido (CDN) de Amazon llamado Cloudfront.
5. Cloudfront obtiene el contenido asociado a la IP solicitada del servicio de almacenamiento llamado S3, procesa los archivos obtenidos y los devuelve al origen de la petición, es decir, el usuario visualiza en su navegador web el contenido de la aplicación web.

6.3.2 Arquitectura del Backend

Igual que hemos vistos anteriormente en la arquitectura del Frontend, a continuación podemos ver una imagen de la arquitectura del Backend junto al flujo de lo que sucede cuando un usuario hace una petición al servicio web que diseñaremos e implementaremos en lo siguientes apartados.

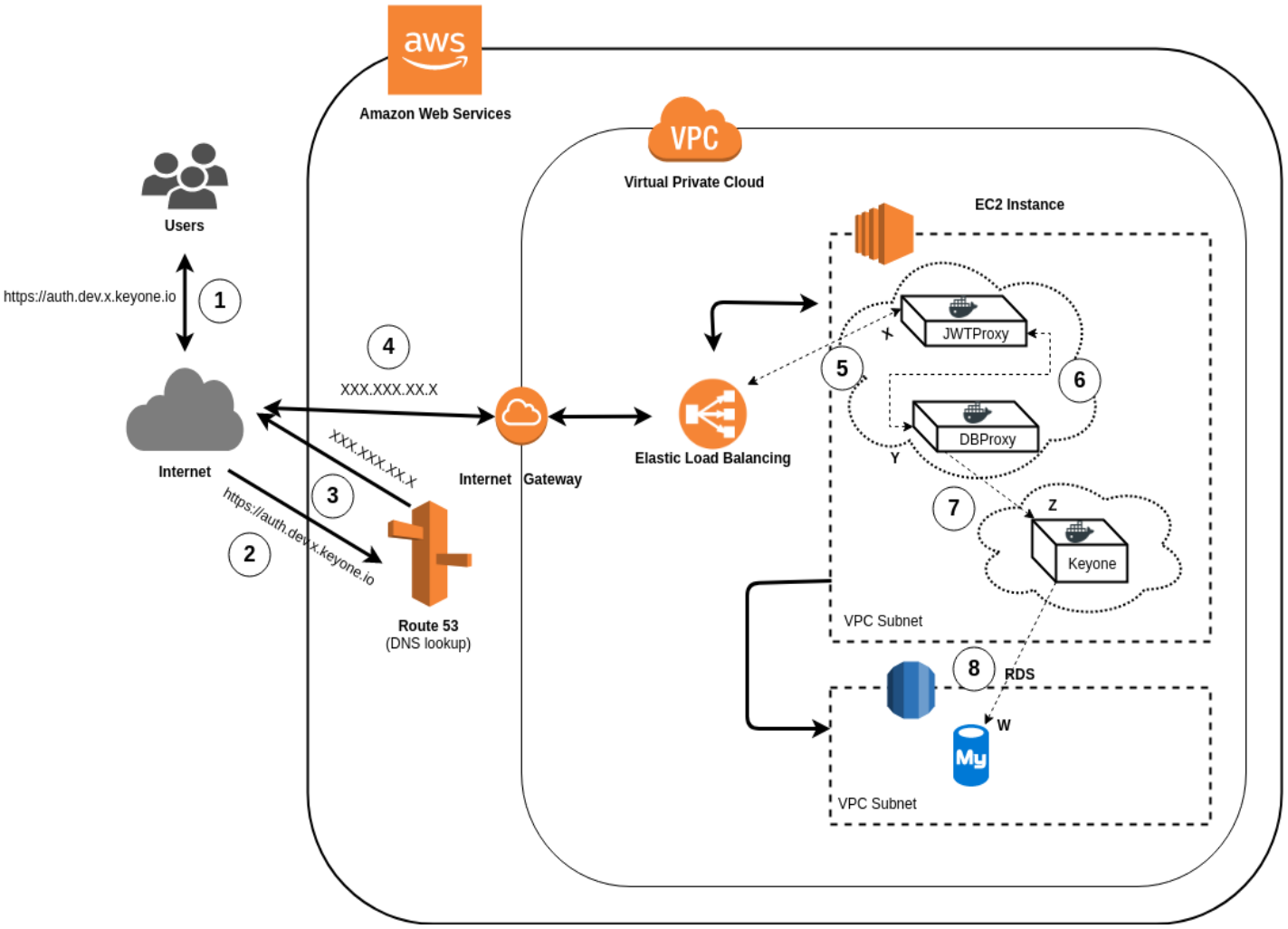


Figura 21: Arquitectura de los servicios del Backend

Igual que en el Frontend, para la realización de esta arquitectura se tuvo que crear y configurar una serie de servicios de Amazon que son los siguientes:

- **Amazon VPC:** como se necesitaba desplegar software en la nube junto a otros servicios de AWS y queríamos que fuese lo más seguro posible se tuvo que crear una red virtual privada la cual se configuró de tal manera que hubiese:
  - Control de acceso.
  - Un punto de enlace a Internet.
  - Tablas de enrutamiento.
  - Distintas subredes, una para los servicios/software y otra para la base de datos.
  - Duplicación de las subredes anteriores en distintas zonas geográficas para garantizar la alta disponibilidad de los servicios.

- **Amazon RDS:** el producto interno de la empresa, KeyOne, necesita de una base de datos relacional MySQL para funcionar. Para ello se creó una instancia de MySQL con este servicio y se configuró tal como indicó Safelayer para posteriormente ejecutar un backup también proporcionado por la empresa.

- **Amazon EC2:** como se necesitaba un entorno en el que desplegar software, es por ello que se tuvo que crear una instancia de EC2 utilizando una AMI (imágen de máquina de Amazon), que proporcionaba Safelayer en la que se indicaba la configuración de software de dicha instancia. También se proporcionó un script con instrucciones para desplegar KeyOne en Amazon. En el capítulo 8 se explicará el despliegue de los microservicios en dicha instancia.

- **Amazon Elastic Load Balancing:** como se ha mencionado anteriormente este servicio proporciona distintos tipos de balanceadores, en mi caso se utilizó un balanceador de carga de aplicaciones para redirigir las peticiones

HTTPS al servicio HTTP (que se explicará con más detalle en los próximos capítulos) que se desplegará en la instancia EC2 anterior.

- **Route 53:** como hemos mencionado anteriormente, este servicio se encarga de asociar un *Domain Name* a otro servicio de Amazon. En este caso, nos interesaba asociar "auth.dev.x.KeyOne.io" a las peticiones que se realizaran al servicio, por lo que se decidió asociar dicho *Domain Name* al Elastic Load Balancing creado anteriormente y así garantizar el acceso a los usuarios finales.

6.3.2.1 Caso de uso: Usuario hace una petición al servicio

1. El usuario hace una petición HTTPS deseando acceder al servicio, es decir, a https://auth.dev.x.KeyOne.io.
2. El proveedor de internet procesa el DNS de la URL introducida y la resuelve redirigiendo al servicio web DNS de Amazon llamado Route 53.
3. Route 53 resuelve el DNS y devuelve la dirección IP asociada.
4. Se redirige la petición a la dirección IP obtenida, asociada al servicio de balanceo de carga de Amazon llamado Elastic Load Balancing.
5. El balanceador realiza su función y procesa la petición HTTPS para redirigirla al puerto HTTP "X" donde se está ejecutando uno de los microservicios.
6. El servicio JWTProxy, que explicaremos detalladamente más adelante, procesa la petición y la redirige al segundo microservicio por el puerto "Y".
7. El servicio DBProxy, que también explicaremos a continuación más detalladamente, procesa la petición y la redirige a KeyOne, el software de la empresa, por el puerto Z.
8. KeyOne procesa la petición obteniendo los datos necesarios de la base de datos MySQL conectándose a ella por el puerto W. Y finalmente el resultado es devuelto al origen de la petición.

6.4 Implementación y testeo del software

6.4.1 Primera iteración

En esta primera iteración lo que se quiere hacer principalmente es la creación de todos los componentes / configuraciones necesarias del servicio de autenticación escogido, Auth0, para hacer funcionar la aplicación web y el servicio JWT, que se encargará de verificar el token que le llegue en la HTTPS request. Para ello se deberá acceder al portal de gestión de Auth0 y realizar las acciones que explicaremos detalladamente a continuación.

6.4.1.1 Implementación

Como hemos mencionado anteriormente se ha tenido que acceder al portal de gestión de Auth0 que está disponible en la web *https://manage.auth0.com*.

- **Creación API Auth0:**

En primer lugar se deberá configurar una API. Una API es una entidad que representa un recurso externo, capaz de aceptar y responder a las solicitudes de recursos protegidos realizadas por los clientes. Para ello, el cliente deberá proporcionar un token de acceso (JWT), dicho token se puede usar para acceder a los recursos de la API sin tener que autenticarse nuevamente, hasta que caduque. Haciendo click en el botón “Create Api” nos aparecerá el siguiente formulario para configurar nuestra API:

En dicho formulario podemos ver tres campos a rellenar/escoger:

- **Name**: Un nombre para la API. No afecta a ninguna funcionalidad.
- **Identifier**: Un identificador único para la API. Se recomienda utilizar una URL.
- **Signing Algorithm**: El algoritmo con el que se firmarán los tokens.

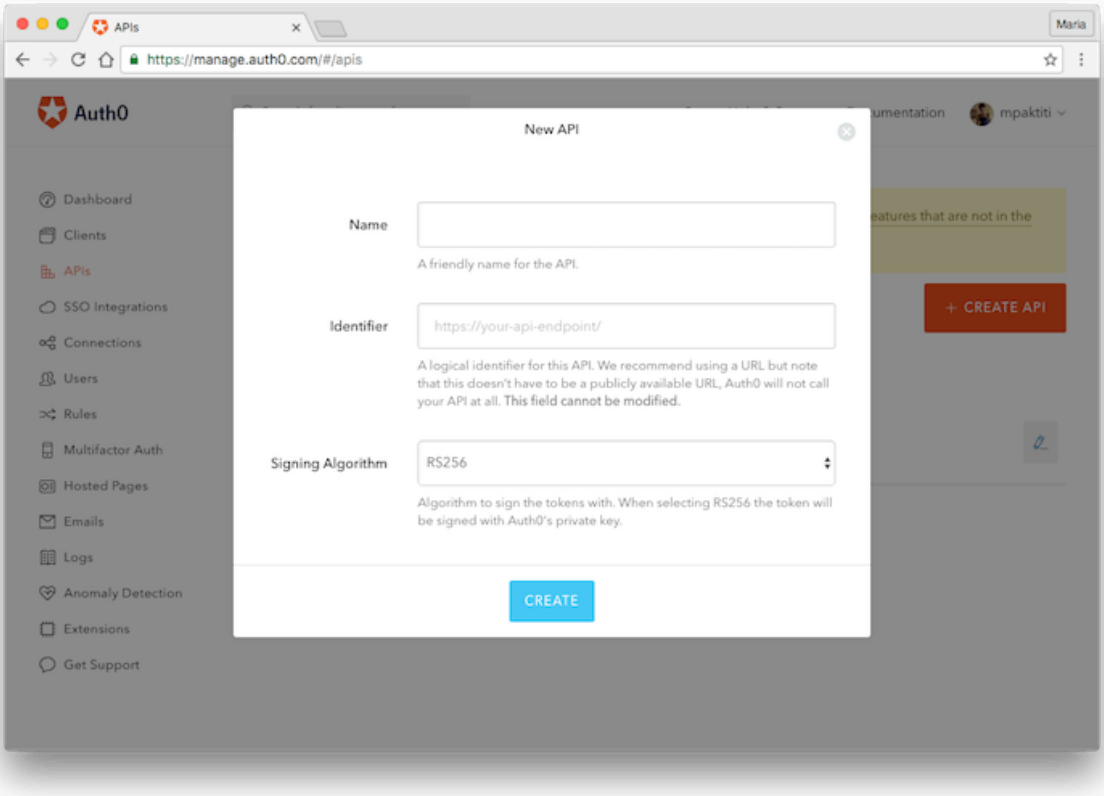


Figura 22: Formulario de creación de una API en Auth0

Una vez creada nuestra API ya estaremos listos para recibir tokens generados por Auth0. Veremos una ventana de *Quick Start* en la que nos dará una explicación y unas variables que se deberán incorporar en el servicio web de validación de tokens que se desee. De todas las variables que nos mostrará el panel nos deberemos quedar con el valor de tres de ellas: **JWKSUri**, **Audience** y **Issuer**. Dichas variables nos servirán para configurar nuestro servicio, JWTProxy, cuando lo tengamos que implementar.

- **Creación Cliente Auth0:**

En segundo lugar, se debe configurar un Cliente de Auth0, dicho Cliente representa la plataforma web y permite el uso de Auth0 para la autenticación de los usuarios. Cabe decir que el término Cliente no implica ninguna característica particular de implementación. Haciendo click en el botón “Create Client” en el apartado Clients nos mostrará un formulario parecido al de la imagen **“X”**. En dicho

formulario tendremos que introducir un nombre para nuestro Cliente y escoger el tipo de Cliente que se desea, en nuestro caso, una Single Page Web Application.

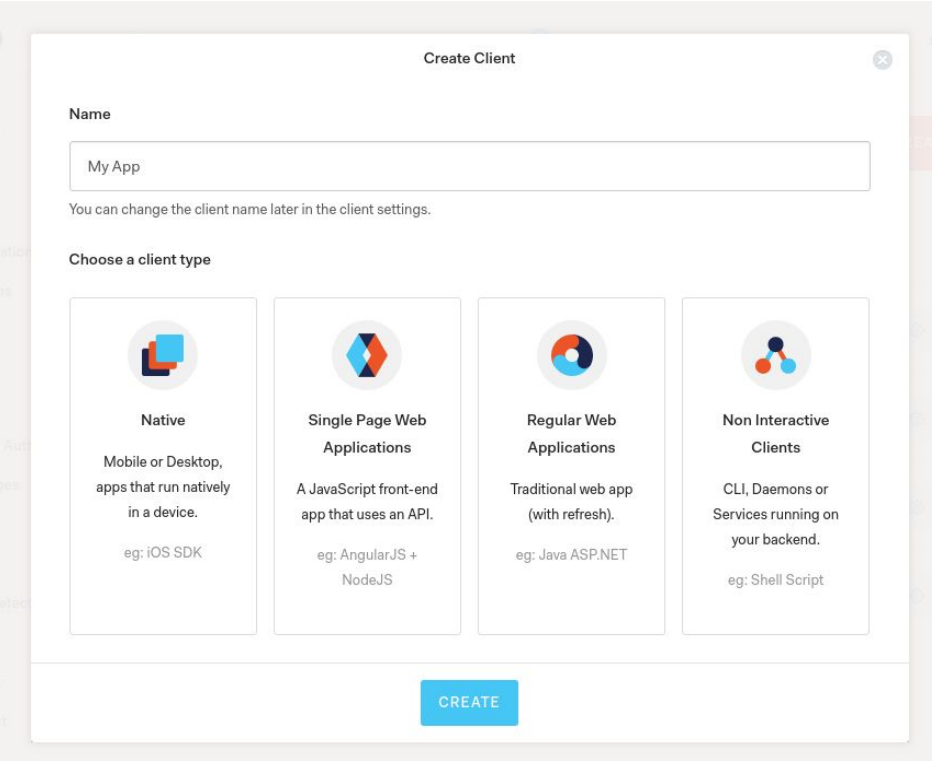


Figura 23: Formulario de creación de un Cliente en Auth0

Igual que con la API cuando hemos creado un Cliente, Auth0 nos mostrará un apartado *Settings* con datos específicos de dicho Cliente que deberemos almacenar y/o recordar dónde encontrarlos para cuando se implemente el componente de autenticación en la plataforma web. Las variables más relevantes serán: **Name**, **Domain**, **Client ID** y **Client Secret.**

❑ Creación conexiones a Cliente Auth0

Una vez creado nuestro Cliente deberemos configurarlo para que acepte autenticación con redes sociales y con las credenciales empresariales, ya que eran unos de los objetivos de este proyecto. Para ello en primer lugar, dentro del portal de gestión de Auth0 deberemos acceder al apartado Connections y luego al subapartado Social. Una vez dentro veremos una larga lista de redes sociales con

las que los usuarios de un cliente se pueden llegar a autenticar. En nuestro caso solo tenemos habilitadas Google, GitHub y LinkedIn:

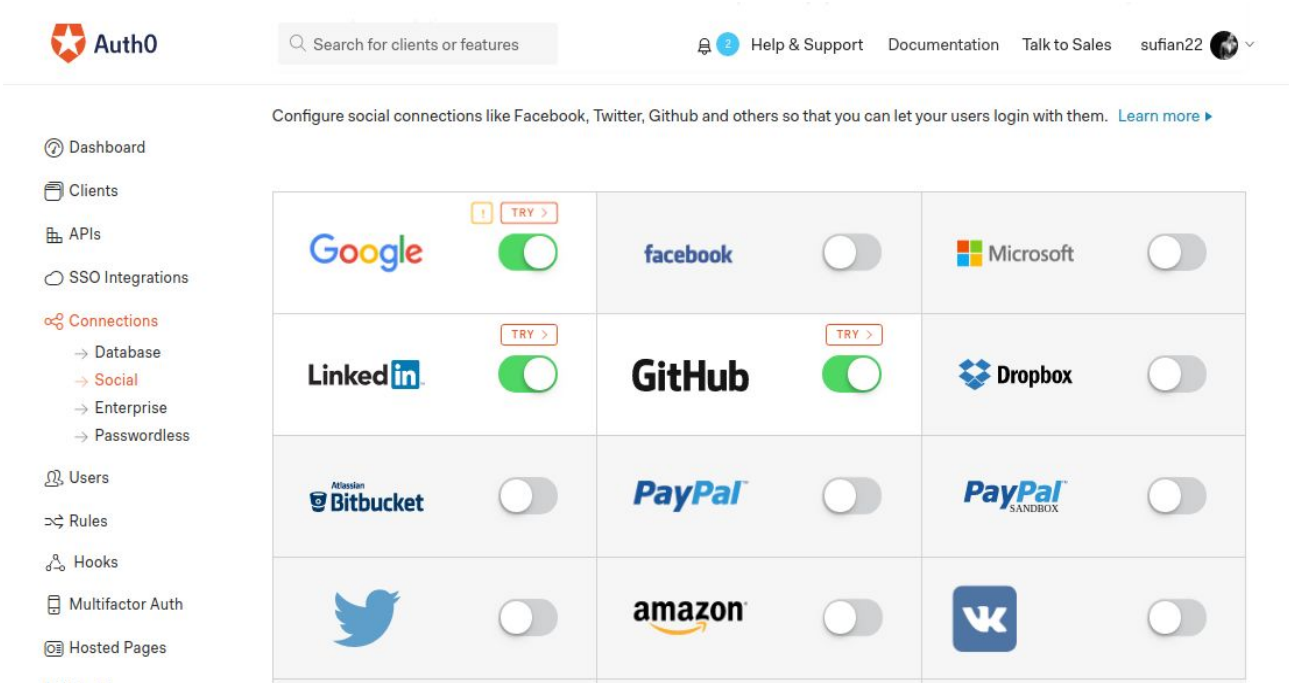


Figura 24: Conexiones con proveedores disponibles en Auth0

Si hacemos click en alguna de estas redes sociales veremos que nos abre un formulario parecido al de la siguiente imagen, en él, los valores más importantes son el Consumer Key y el Consumer Secret. Para obtenerlos se deberá acceder al apartado developers en la web oficial de de cada una de las redes sociales que queramos y crear una aplicación de autenticación para obtener dichos parámetros y poder asociar Auth0.

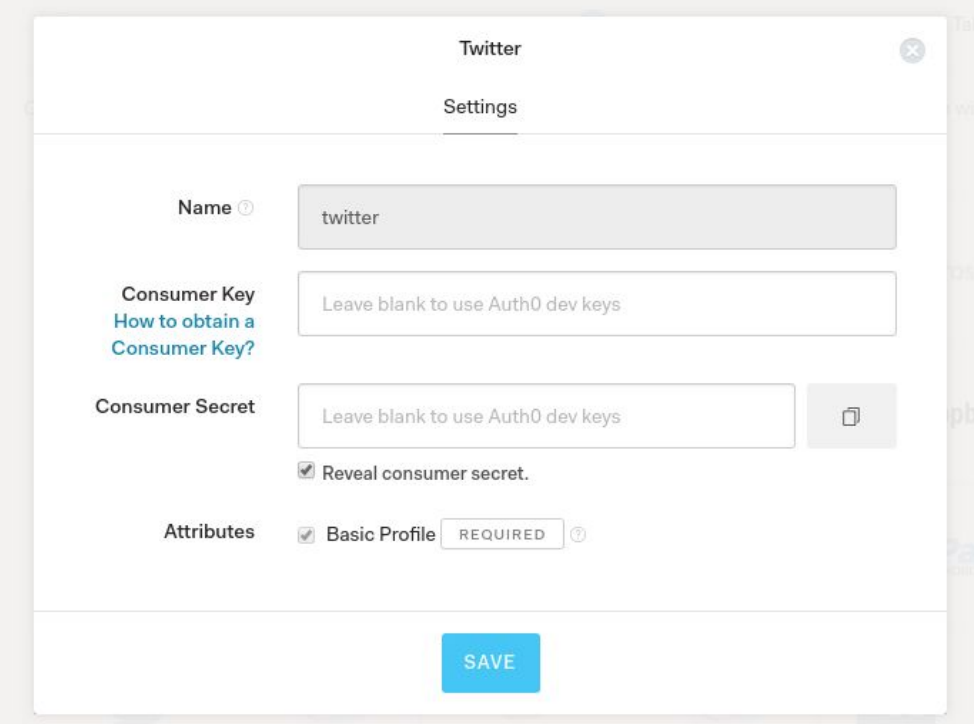


Figura 25: Configuración de una conexión a Twitter en Auth0

Una vez configuradas las conexiones a las redes sociales deseadas solo nos quedará configurar de algún modo la autenticación en Auth0 con las credenciales empresariales. Para ello accederemos al apartado Connections, siguientemente al subapartado Enterprise y haremos click finalmente en el elemento Active Directory / LDAP. Igual que con las redes sociales nos saldrá un formulario en el que nos bastará con solo poner un nombre a dicha conexión.

Una vez creada la conexión, Auth0 nos proporcionará un valor único que identificará a dicha conexión creada, el **Ticket URL**. Posteriormente a la creación de la conexión se deberá configurar con el LDAP de la empresa, para ello Auth0 nos proporcionará un manual de como hacerlo que se puede consultar en la siguiente web en caso de dudas: *https://auth0.com/docs/connector/install-other-platforms*. Para ello deberemos descargarnos de su repositorio de GitHub un servicio implementado por ellos en el que lo único que deberemos hacer es proporcionar el valor del Ticket URL que hemos mencionado anteriormente y substituir en un fichero

JSON el valor de los siguientes parámetros que corresponden a la configuración de un LDAP:

You will be prompted to edit the `config.json` configuration file with your LDAP connection and authentication details:

```
"LDAP_URL": "ldap://YOUR_LDAP_SERVER_FQDN",
```

Figura 26: Parámetros de configuración de un LDAP

Una vez configurado el servicio, como es un servicio hecho en NodeJS habrá que ponerlo en marcha y será tan fácil como ejecutar **npm start -s** y veremos si lo hemos configurado correctamente. Tendríamos que ver un resultado parecido al siguiente:

```
sufian.ben_bouker@ravioli /opt/auth0-adldap sudo npm start -s
[sudo] password for sufian.ben_bouker:
[2018-04-09 08:14:08] Reading CA certificates from OPENSSLDIR
[2018-04-09 08:14:08] Reading CA certificates from /usr/lib/ssl/certs
[2018-04-09 08:14:08] Adding 154 certificates
[2018-04-09 08:14:08] Loading settings from ticket: https://sufian22.eu.auth0.com/p/ad/ZQRKIDmH/info
[2018-04-09 08:14:09] Local settings updated.
[2018-04-09 08:14:09] Certificates already exist, skipping certificate generation.
[2018-04-09 08:14:09] Configuring connection testingADLDAP.
[2018-04-09 08:14:09]  > Posting certificates and signInEndpoint: http://ravioli:4000/wsfed
[2018-04-09 08:14:09] Connection testingADLDAP configured.
[2018-04-09 08:14:09] Connector setup complete.
[2018-04-09 08:14:09] Cache enabled
[2018-04-09 08:14:09] Connecting to wss://sufian22.eu.auth0.com/lo/hub.
[2018-04-09 08:14:09] auth0: Agent accepted.
[2018-04-09 08:14:10] latency test took avg: 162.19 ms, max: 186.93 ms, min: 144.98 ms
```

Figura 27: Resultado de una conexión al LDAP corporativo

Si el resultado en la terminal es correcto. En nuestra aplicación veremos que se ha habilitado la opción para autenticarnos con nuestras credenciales empresariales cada vez que hagamos click en el botón Login que implementaremos en las siguientes iteraciones.

❑ **Configuración 2FA en Auth0**

Una vez tenemos configuradas las conexiones a nuestra aplicación con redes sociales y credenciales empresariales, procedemos a configurar una segunda capa de protección con el protocolo de 2FA (two-factor authentication). Para ello deberemos ir al menú izquierda del panel de gestión de Auth0 y hacer click en la opción *Multifactor Auth.* Veremos que dispone de dos métodos de verificación de identidad, mediante notificaciones en el móvil o mediante mensaje SMS. En nuestro caso hemos decidido utilizar las notificaciones, para ello será tan fácil como habilitar el switch:

Figura 28: Autenticación 2FA en Auth0

Una vez habilitada esta opción, directamente en todas nuestras aplicaciones configuradas en Auth0 nos pedirá una segunda verificación cada vez que iniciemos sesión. Dicho esto, no necesitaremos tampoco ninguna modificación en el código ya que Auth0 se encarga de ello mediante su servicio.

6.4.1.2 Inspección y validación

Como dijimos en el apartado de Metodología y siguiendo con las mejores prácticas a la hora de desarrollar software, todo lo que se implementa / configura se debe testear. El servicio de Auth0 ya está pensado para ello y es por eso que proporciona opciones de probar todas las configuraciones de las que hemos hablado anteriormente y podremos ver una simulación del resultado obtenido identificándonos tanto con las redes sociales como con las credenciales de nuestra empresa, en nuestro caso Safelayer.

Para ello volvemos al apartado *Connections* del panel de gestión de Auth0 y hacemos click en lo que deseamos probar, Social (para redes sociales) o Enterprise (como dice su nombre en inglés, para credenciales empresariales). En el caso de las redes sociales veremos como una opción en naranja llamada *TRY:*

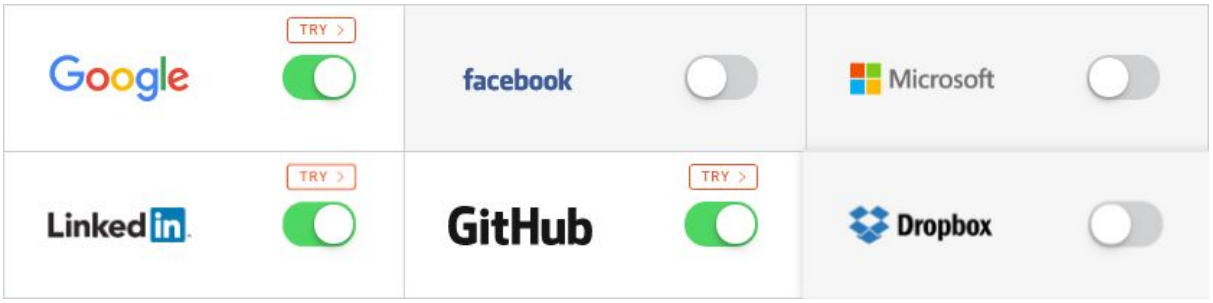


Figura 29: Conexiones habilitadas en Auth0

Una vez hagamos click en cualquiera de las redes sociales nos redirigirá a la página web de dicha red social para autenticarnos con nuestras credenciales. Una vez introducidos nuestros datos deberíamos ver un resultado parecido al siguiente:

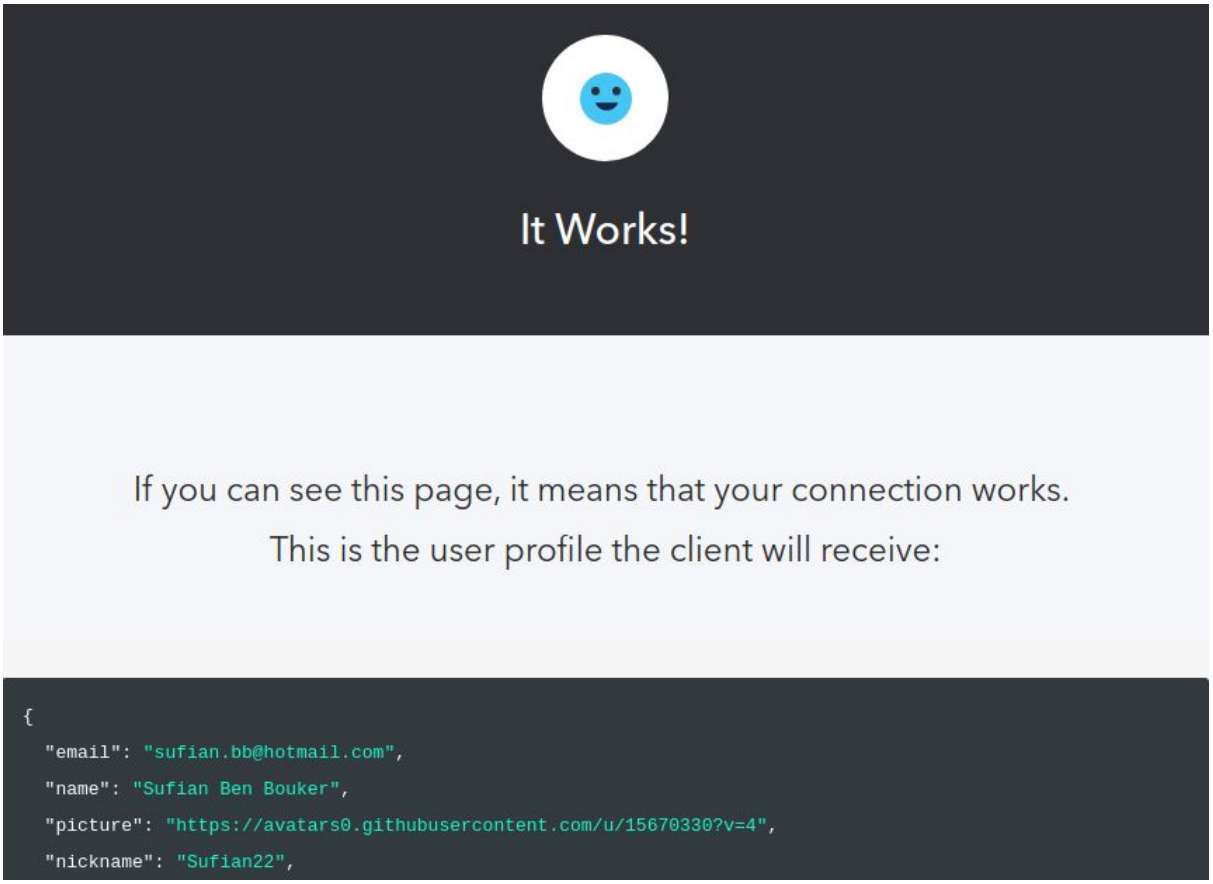


Figura 30: Confirmación autenticación con red social en Auth0

En el caso de la autenticación corporativa deberemos ir al apartado *Enterprise* y una vez ahí seleccionar el subapartado *Active Directory / LDAP.* Nos saldrá una lista con todas las conexiones creadas, las activas saldrán con un círculo de color verde y las inactivas con un círculo rojo:

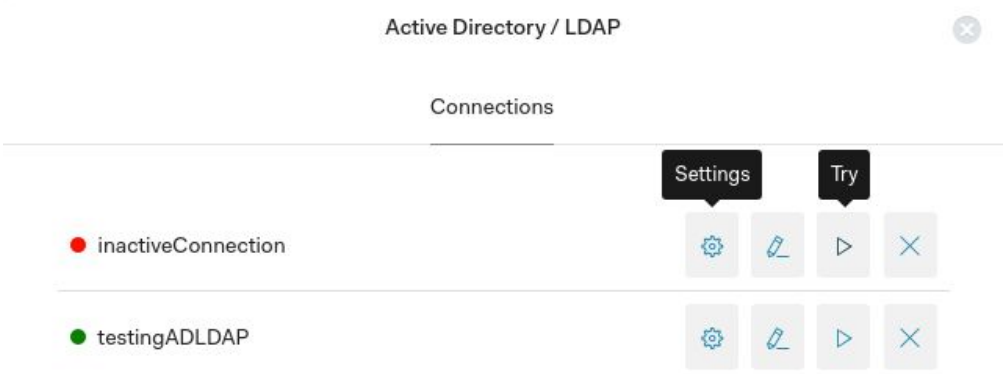


Figura 31: Conexiones LDAP disponibles en Auth0

Igual que las redes sociales, las conexiones corporativas también disponen de una opción *Try* para probar dicha configuración. Si hacemos click en dicha opción nos saldrá un panel igual al mencionado anteriormente en la **imagen X.** Una vez introducidas nuestras credenciales veremos en la web los siguientes resultados, parecidos a cuando nos hemos autenticado con alguna red social.

También, podemos ver la confirmación de autenticación en el servicio que hemos configurado y ejecutado tal como se ha mencionado anteriormente:

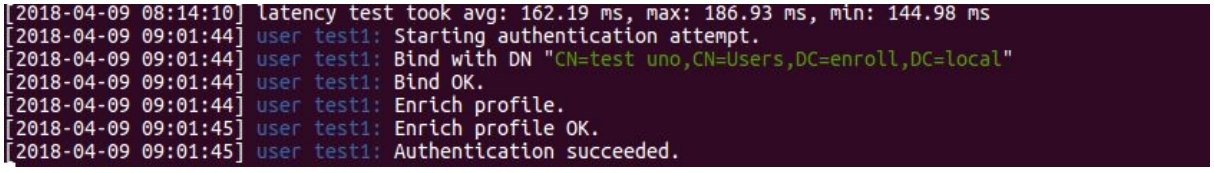


Figura 32: Confirmación autenticación LDAP en Auth0

Si la configuración de la doble autenticación también ha sido hecha correctamente, después de autenticarnos con las credenciales de nuestra red social escogida nos debería aparecer la siguiente ventana:

Como podemos ver nos sugiere instalar en nuestro teléfono móvil una aplicación para la generación de las claves de la doble autenticación (Google Authenticator o similar). Si coincide la clave de 6 dígitos nos mostrará la ventana de resultado que hemos mencionado anteriormente con nuestros datos.

Así que de este modo quedan testeadas todas nuestras opciones de autenticación, solo quedaría implementar el componente de autenticación en la plataforma web.

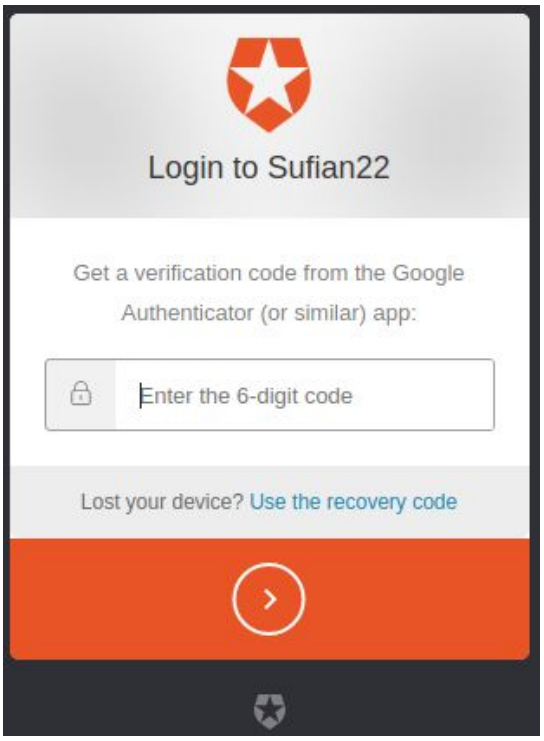


Figura 33: Two Factor Authentication (2FA)

6.4.2 Segunda iteración

En esta segunda iteración el principal objetivo será el desarrollo del servicio web que se encargará de validar el JWT procedente de las peticiones HTTPS con nuestra API de Auth0 anteriormente configurada y también como hemos dicho anteriormente se encargará de redirigir la request al siguiente servicio.

6.4.2.1 Implementación

En primer lugar se tuvo que crear un *reverse proxy,* es decir, un proxy que escuchara por un puerto y redirigiera las peticiones por otro puerto. Como hemos dicho anteriormente, utilizamos Golang (Go) para implementar dichos servicios, Go ya dispone de muchas utilidades y documentación de como configurar y implementar uno. Se puede consultar en la siguiente web: https://golang.org/pkg/net/http/httputil/#ReverseProxy.

```
const (
    redirectTo = "http://dbproxy.internal:4444"
    listenPort = 8080
)

(...)

// NewCustomProxy returns a proxy supporting concurrent connections using keep alive
func NewCustomProxy(destURL *url.URL) *httputil.ReverseProxy {

    (...)

    return &httputil.ReverseProxy{Director: proxy.Director, Transport: tp}
}
```

Figura 34: Creación y configuración de un *reverse proxy*

En segundo lugar se tuvo implementar un *middleware* de Auth0 para supervisar todas las peticiones que le llegaran al servicio y así poder autenticar las peticiones y saber quién estaba pidiendo qué. Para ello se utilizaron las variables de la configuración de la API de Auth0 que hemos mencionado en la iteración anterior.

```
// Auth0 Variables
var jwksURI = os.Getenv("JWKS_URI")
var issuer = os.Getenv("ISSUER")
var audience = []string{os.Getenv("AUDIENCE")}
```

Figura 35: Variables de configuración de Auth0

En tercer y último lugar, una vez verificado y validado el token JWT procedente de la *request* se procedió a cambiar la cabecera de la petición y así poder redirigir la petición al siguiente servicio.

```
// If the token is valid and we have the right scope, we'll pass through the middleware
r.Header.Del("jwt")
r.Header.Add("providerID", providerID)
h.ServeHTTP(w, r)
```

Figura 36: Redirección de una petición HTTP

6.4.2.2 Inspección y validación

Como se dijo en la metodología de trabajo y también siguiendo los principios de software de calidad, el código de este servicio también se tiene que testear. En la siguiente imagen se pueden obtener los resultados obtenidos:

Figura 37: Validación del código de los microservicios

Como podemos observar el porcentaje no es del 100% sino un 65,3% debido a que hay una parte del software producido que es de librerías externas y otra parte que es código que no hace falta su testeo. Pero la herramienta de supervisión de tests de Golang revisa línea por línea todo el código existente, así que con la aprobación de Safelayer se dá por válido y suficiente dicho valor obtenido.

6.4.3 Tercera iteración

En esta tercera iteración el principal objetivo será el desarrollo del servicio web que se encargará de validar la autorización al producto interno de la empresa, KeyOne. Para ello deberá procesar las peticiones HTTPS que le lleguen y extraer lo necesario para hacer la comprobación con la base de datos interna.

6.4.3.1 Implementación

En primer lugar, igual que al anterior servicio, se tuvo que crear un *reverse proxy* ya que el funcionamiento es parecido. Tal como hemos visto anteriormente, la creación del proxy será bastante parecido, pero con ciertas modificaciones. Cabe decir que la redirección del proxy, lógicamente no será la misma ya que no redireccionará al otro servicio sino al software de la empresa desplegado en la nube:

```
const (
    redirectTo = "https://keyone.internal:45000"
    listenPort  = 4444
)
```

Figura 38: Constantes necesarias para la redirección HTTPS

En segundo lugar, tal como hemos ido diciendo anteriormente, este segundo servicio web se encargará de la extracción del valor del providerID proveniente del JWTProxy, verificar la identidad de dicho usuario con la base de datos y extraer su certificado para posteriormente modificarlo en base 64 y redirigirlo a KeyOne.

```
//HandlerProxy redirects depending on body value
func HandlerProxy(proxy *httputil.ReverseProxy) func(http.ResponseWriter, *http.Request) {
    return func(w http.ResponseWriter, req *http.Request) {

        providerID := req.Header.Get("providerID")

        (...)

        if err != nil {
            log.Printf("Error reading cert: %s", err)
        }

        certificateBase64 := base64.StdEncoding.EncodeToString(certificate)

        (...)

        req.Header.Del("providerID")
        req.Header.Add("X-Client-Cert", certificateBase64)

        proxy.ServeHTTP(w, req)
    }
}
```

Figura 39: Porción de código del microservicio

6.4.3.2 Inspección y validación

Tal como hemos mencionado y visto anteriormente el código de dicho servicio también ha estado sujeto a un proceso de calidad mediante la validación de los tests. En la imagen del *coverage* del servicio anterior vimos como el porcentaje adquirido era de un 65,3%. En dicho porcentaje se incluía la parte del código de este servicio también ya que los tests incluyen las funcionalidades de ambos servicios y la verificación del proceso completo hasta llegar al producto de Safelayer desplegado en la nube, KeyOne. Por estos motivos, se dió por buena la implementación de este servicio y desde Safelayer se dá el visto bueno.

6.4.4 Cuarta iteración

En esta última iteración se ha centrado en la implementación del componente web dentro de la interfaz diseñada en el apartado 6.2.

6.4.4.1 Implementación

Para ello se ha tenido que reestructurar la web un poco debido a que es necesario añadir una vista principal en la que se fuerce a los usuarios a autenticarse para acceder al contenido. Por eso se ha creado un componente (siguiendo los principios de React) llamado Main en que la su visualización es la siguiente:

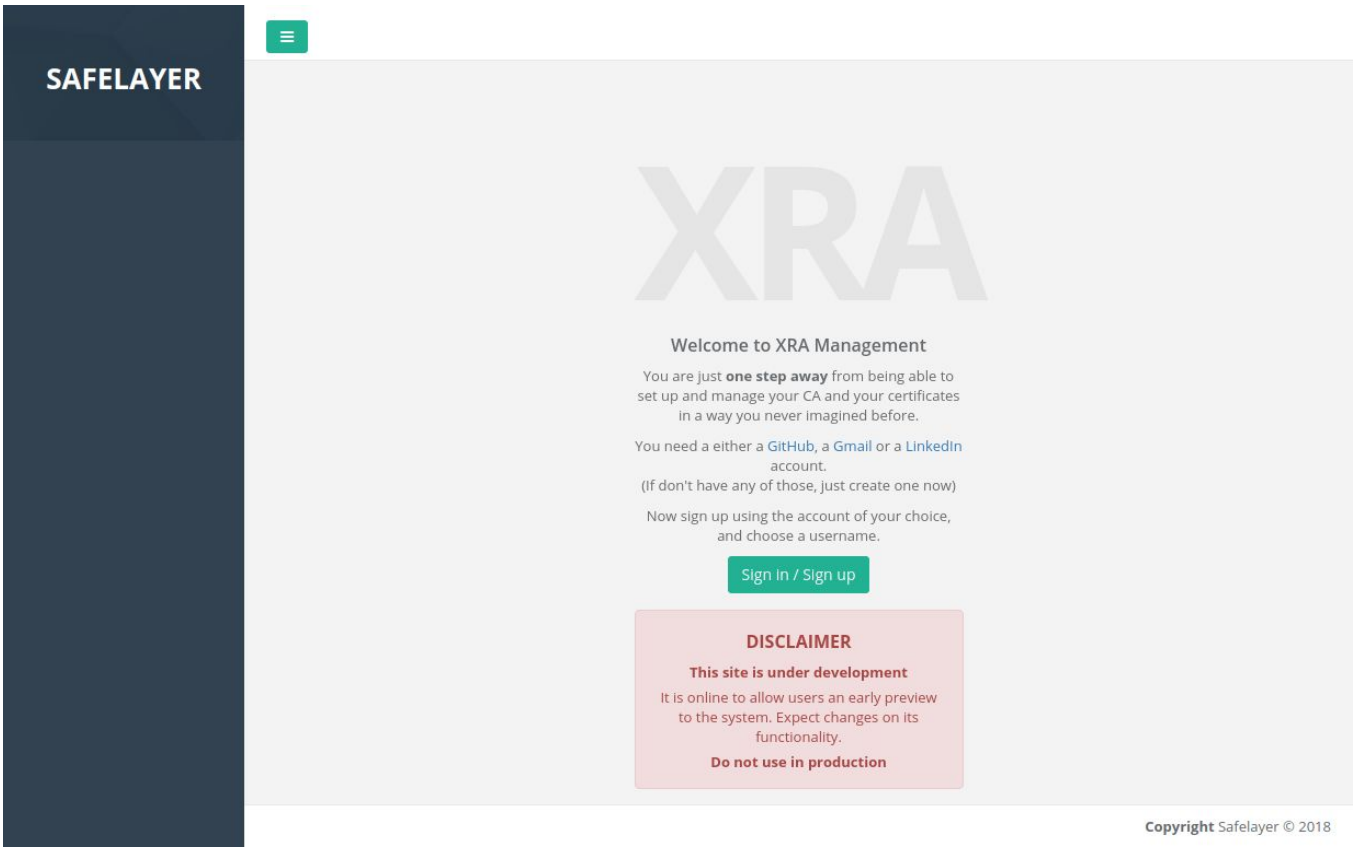


Figura 40: Visualización principal de la plataforma web

En dicho componente se ha introducido un componente de autenticación con toda la lógica de Auth0, para ello han sido necesarias algunas de las variables que se han mencionado anteriormente en la configuración de un Cliente Auth0 (sub apartado 6.4.3.1) como el identificador y el dominio de dicho cliente (almacenadas como variables de entorno dentro del código):

```
const CLIENT_ID = `${CLIENT_ID}`;
const CLIENT_DOMAIN = `${CLIENT_DOMAIN}`;

let auth = new auth0.WebAuth({
  clientID: CLIENT_ID,
  domain: CLIENT_DOMAIN
});

export function login() {
  auth.authorize({
    responseType: 'token id_token'
  });
}
```

Figura 41: Función encargada de ejecutar el servicio de Auht0

Siguiendo los principios de React & Redux, al hacer click en el botón Sign In/Sign Up del componente inicial se produce un evento onClick que es capturado por la acción que ejecuta la función login() de la imagen anterior. Dicha función se encarga de renderizar visualmente el componente de autenticación de Auth0 y posteriormente gestionar el resultado obtenido:

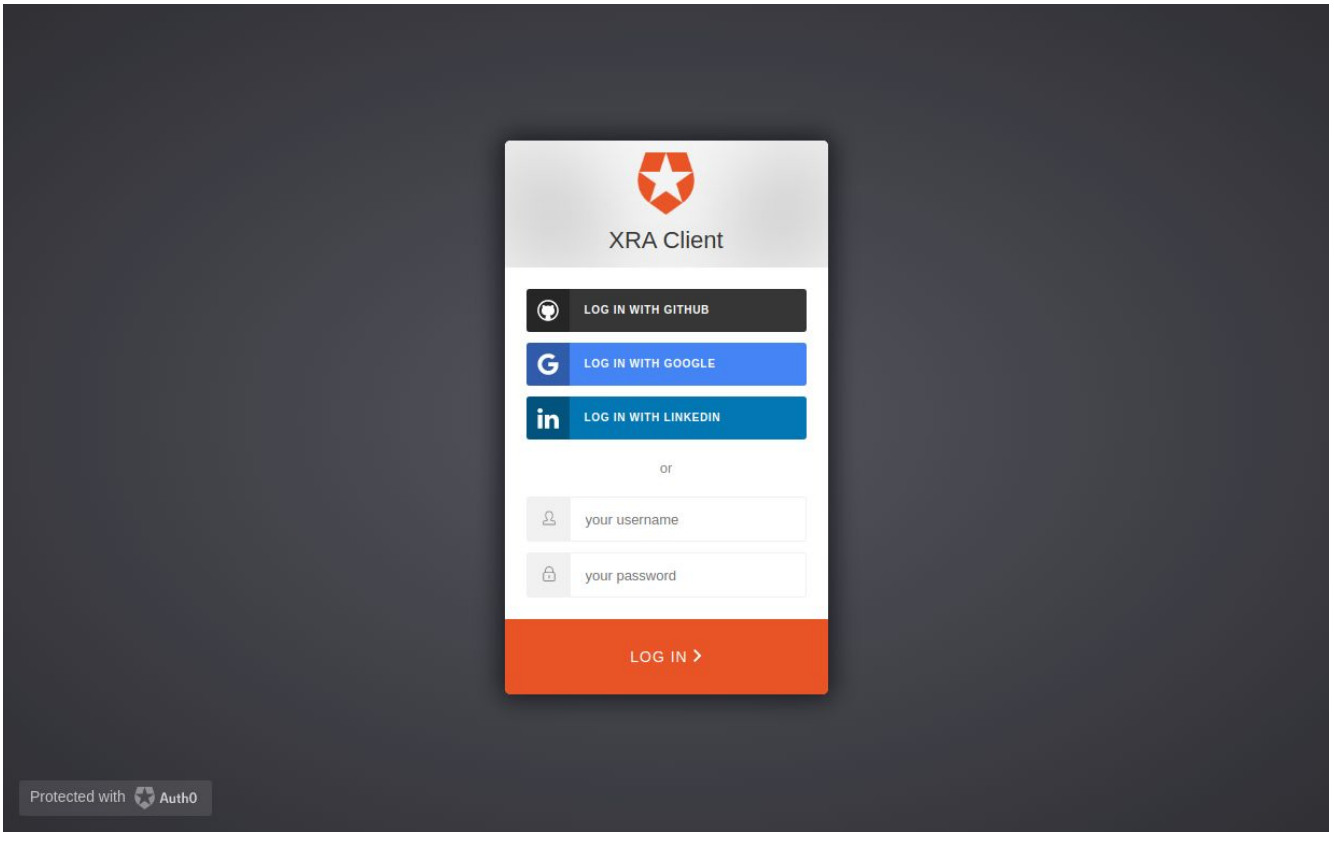


Figura 42: Servicio de Auth0 en la plataforma web

El componente de la imagen anterior se puede visualizar como se ofrece a los usuarios la posibilidad de autenticarse con múltiples redes sociales (Google, Github & LinkedIn) y o también un formulario en el que se puede introducir un usuario y una contraseña, dichos campos son para introducir las credenciales del personal de la empresa. Una vez introducidas las credenciales nos aparece la ventana correspondiente a la segunda verificación en la cual necesitaremos tener nuestro *smartphone* a mano para poder acceder.

Una vez completado el proceso de autenticación podemos ver que se han habilitado las funcionalidades principales de la plataforma web en la barra lateral junto a las opciones de consultar los datos de nuestro perfil y la posibilidad de salir de nuestra cuenta en la esquina superior derecha:

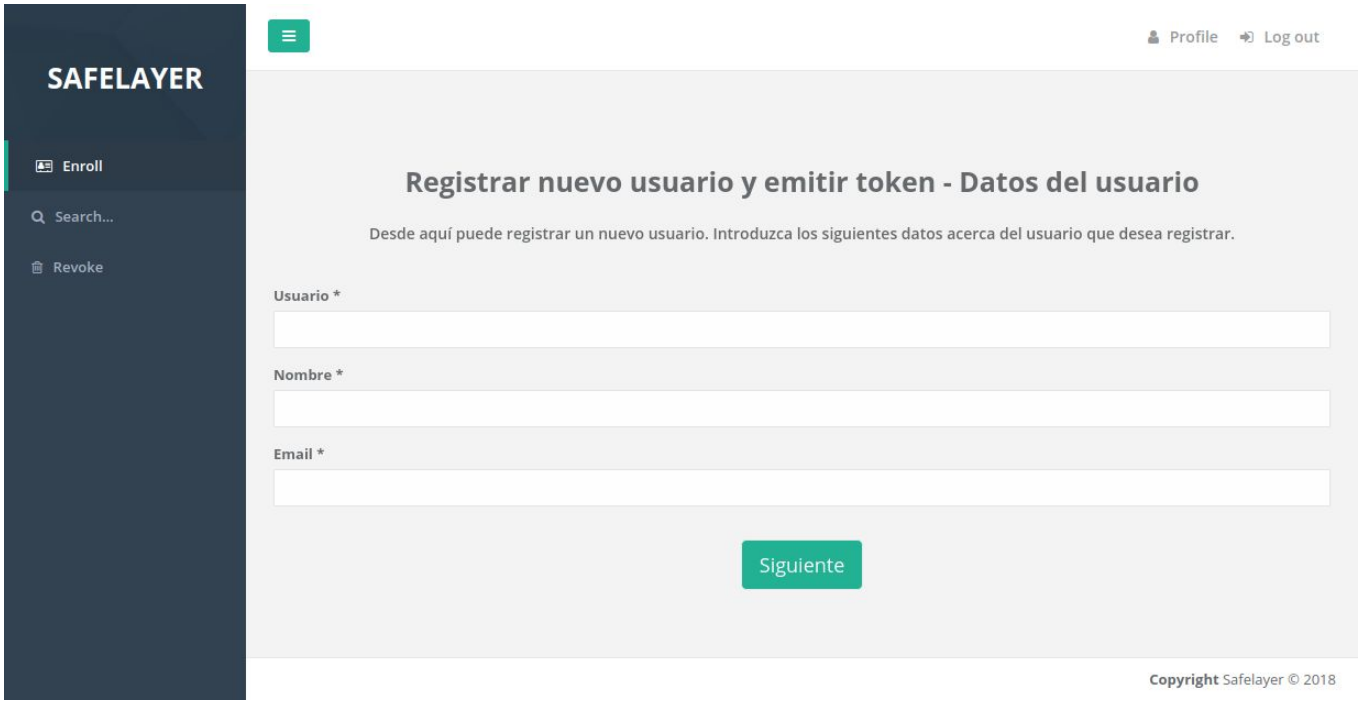


Figura 43: Visualización principal de los usuarios autenticados

También se ha tenido que modificar la lógica de la plataforma web debido a que antiguamente todas las funcionalidades de la plataforma hacían peticiones HTTPS directamente al software de la empresa, es decir, a KeyOne. Ahora, dicho software ha quedado protegido de tal modo que solo se puede acceder a él mediante los servicios explicados e implementados anteriormente:

```
static fetchAPI = (param,url) => {
    return fetch('https://auth.dev.x.keyone.io'+ url + '?' + querystring.stringify( param ),
    {
      method: 'GET',
      headers: {
        'Authorization': 'Bearer ' + getAccessToken(),
      },
    })
    .then (function(response) {
      if (response.status === 401) {
        logout()
      }
      else if (response.status === 200) {
        browserHistory.push("/enroll")
      }
      else {
        toastr.error("Unauthorized. Contact the administrator to have permissions.")
        logout()
      }
    })
    .then (function(json) {
        if (json.errorId) throw json
        else return json
    })
    .catch(function(ex) {
        throw ex
    })
}
```

Figura 44: Código encargado de la realización de las nuevas peticiones

Como podemos ver en la imagen anterior, se ha creado una función genérica para casi todas las peticiones realizadas al servicio web, en las que se le añade una cabecera con el token JWT devuelto por el servicio de Auth0. De éste modo también es más sencillo procesar todas las posibles respuestas devueltas por nuestro servicio. La url de https://auth.dev.x.KeyOne.io es el DNS creado para el servicio JWTProxy, en el capítulo 8 hablaremos más en detalle sobre el despliegue de los servicios y la plataforma web en la nube.

También se ha creado un componente para la visualización del perfil del usuario. Los datos devueltos por el proveedor / por la conexión configurada en Auth0 son susceptibles a la configuración del usuario en éste, por ello solo son datos de lectura y no se permite la opción de modificarlos. Siguiendo la lógica que hemos mencionado anteriormente, al hacer click en el apartado Profile, se produce una acción que se encarga de renderizar el componente ProfileModal que incluye toda la renderización visual y la lógica de éste.

```
{ this.state.showProfile ?
    <ProfileModal
        showProfile={this.state.showProfile}
        handleProfileModal={this.handleProfileModal}
    />
: null
}
```

Figura 45: Código encargado de la visualización del perfil del usuario

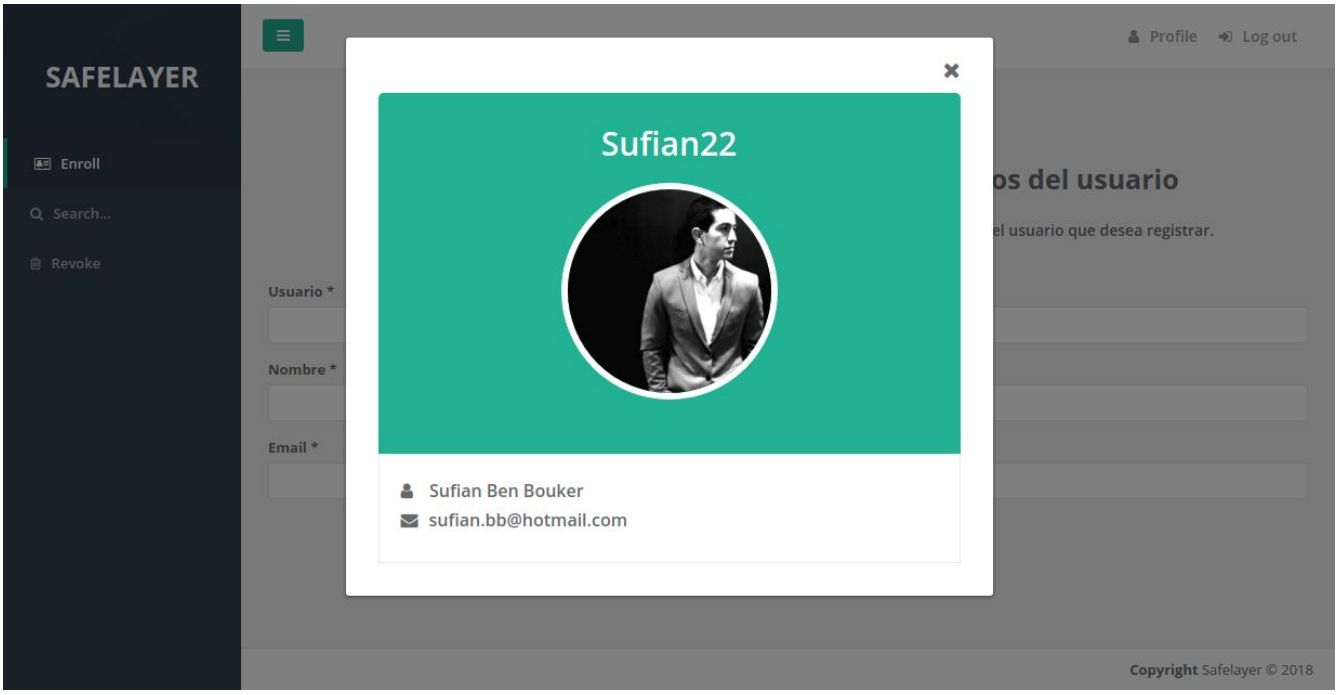


Figura 46: Visualización del perfil del usuario autenticado

6.4.4.2 Inspección y validación

Igual que los microservicios implementados, el código correspondiente al frontend, es decir, a la interfaz de la plataforma web también debe pasar los controles de calidad. En este caso no era tan sencillo ya que se partía de un código y unos tests unitarios ya implementados así que en un principio se tuvo que ver todo lo que hay hecho y probado para que la incorporación del nuevo componente no produjera ningún error y todo funcionase correctamente. Y éstos han sido los resultados:

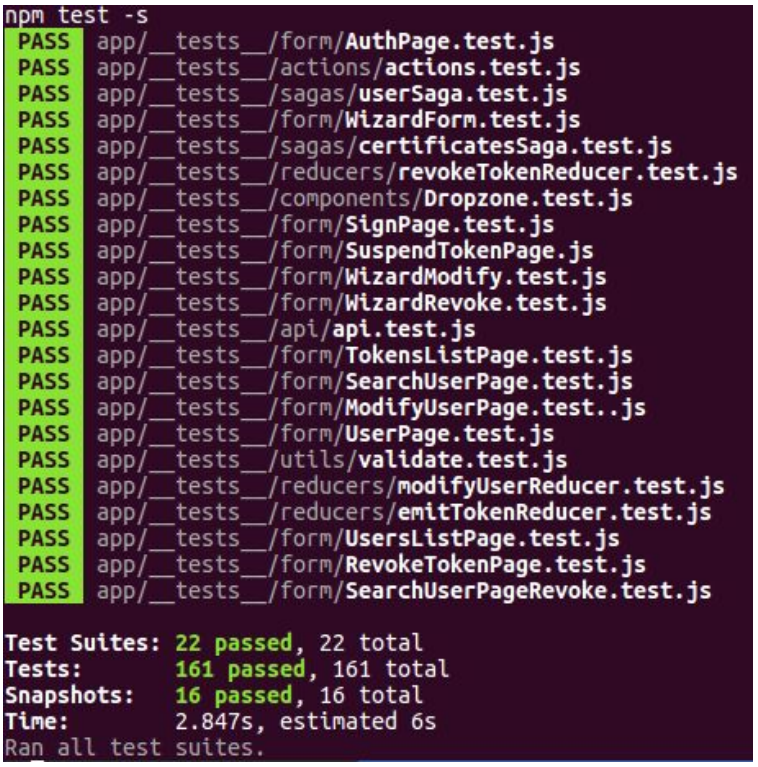


Figura 47: Ejecución de los pruebas realizadas en el código de la plataforma web

Como podemos ver se han pasado todos los tests del proyecto, tanto los antiguos como los nuevos, un total de 161 tests en 22 escenarios. Los tests implementados para este proyecto han sido AuthPage.test.js y UserPage.test.js, el test modificado ha sido el api.test.js.

De este modo quedan implementados, testeados y se dan por validados por parte de Safelayer todos los componentes que añaden la funcionalidad de autenticación a la plataforma web ya existente. En el siguiente capítulo procederemos a explicar los procedimientos de cómo desplegar los componentes mediante los servicios de Amazon incluyendo los principios del *continuous delivery.*

Capítulo 7: Despliegue del software

En este capítulo se explicará cómo se logra hacer que cada pequeño cambio en la interfaz web o en los servicios web los usuarios logren ver los cambios en producción, es decir, como se han seguido los principios del *continuous delivery* en tanto en el Frontend como en el Backend.

7.1 Despliegue de la aplicación web

En la siguiente imagen se pueden visualizar los procesos que se realizan para satisfacer los principios del *continuous delivery* en el frontend. Antes de todo hace falta comentar que para ello se ha debido de crear una asociación entre el repositorio de Frontend con la herramienta Travis CI, se puede hacer de forma muy sencilla desde la página web de GitHub. También, que se le han adjudicado a Travis CI las credenciales de Amazon para que pueda acceder al servicio S3.

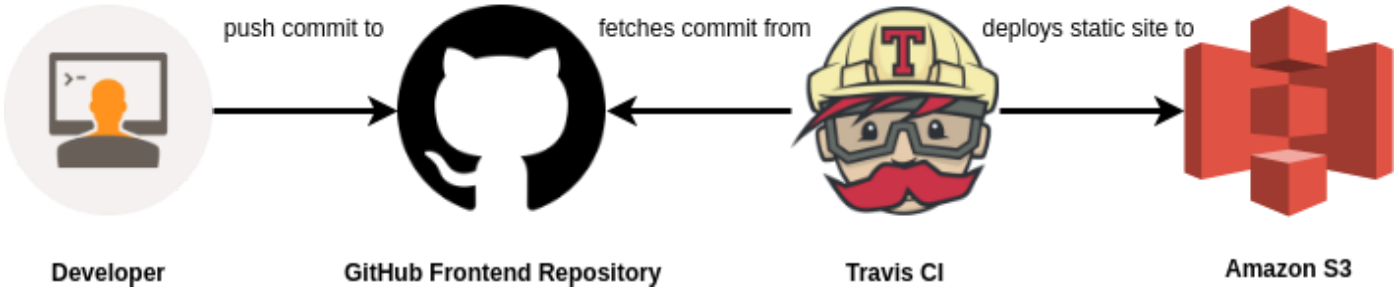


Figura 48: Procesos automáticos del continuous delivery en el Frontend

Dicho esto, como podemos observar, cada vez que un desarrollador hace un *commit*, es decir, un cambio en el código y lo sube al repositorio correspondiente de GitHub, Travis se encarga de realizar las acciones que se le han establecido en el fichero de configuración que hay almacenado en el repositorio. Dicho fichero en nuestro caso establece que cada vez que Travis detecte un *commit*, se realicen los *tests* pertinentes y que si son superados con éxito, compile el contenido y se conecte al servicio S3 para que almacene los datos en el *bucket* indicado.

De esta forma como se ha mostrado anteriormente en la arquitectura del Frontend, dicho *bucket* del servicio S3 está asociado a una distribución de Cloudfront, encargada de procesar el contenido de éste, por lo que los usuarios siempre estarán viendo el último cambio hecho en la página web cuando accedan desde sus navegadores.

7.2 Despliegue de los microservicios

En la siguiente imagen se pueden visualizar los procesos que se realizan para satisfacer los principios del *continuous delivery* en el backend. Antes de todo hace falta comentar que para ello, igual que en el caso del Frontend, se ha tenido que crear una asociación entre el repositorio de Backend con la herramienta Travis CI. También, que se le han adjudicado a Travis las credenciales de Docker para que pueda acceder al servicio de Docker Hub.

Como veremos a continuación, para el despliegue de los microservicios implementados se ha tenido que dividir en dos fases, una fase automática y otra que es necesaria la interacción manual del desarrollador.

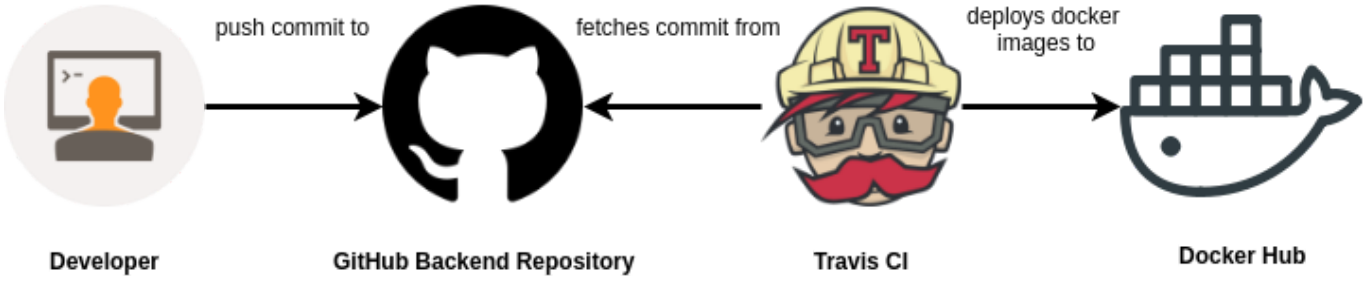


Figura 49: Fase automática del continuous delivery en el Backend

En la imagen anterior podemos ver la primera fase, igual que en el Frontend, cada vez que el desarrollador realiza un *commit* en el repositorio de Backend, Travis CI se encarga de realizar las funciones establecidas en el fichero de configuración. A diferencia del Frontend, en dicho fichero de configuración se le ha establecido a Travis CI que después de la ejecución de los *tests* correspondientes, que mediante Docker ejecute la compilación de los microservicios y la creación de las respectivas

imágenes en un contenedor individual. Finalmente, dichas imágenes son subidas al Docker Hub gracias a las credenciales de Docker otorgadas.

Figura 50: Fase manual del despliegue de los microservicios

En la segunda fase, una vez subidas las imágenes de los microservicios a Docker Hub. El desarrollador de la empresa debe conectarse mediante SSH a la instancia de EC2 que se está ejecutando en Amazon, la cual tiene Docker instalado, y ejecutar un script en el que se le ordena descargar las dos imágenes de Docker Hub y ejecutarlas. Lo cual implica la creación de un contenedor Docker para cada microservicio dentro de una misma network en la cual solo el puerto en el que se está ejecutando el JWTProxy es accesible, tal como se ha mostrado en la arquitectura del Backend (apartado 6.2).

De éste modo, gracias a la configuración de los servicios de Amazon ya hecha el usuario final cada vez que haga una petición al DNS correspondiente a la API de KeyOne obtendrá el resultado correspondiente a la última implementación de los servicios.

Capítulo 8: Planificación

Esta sección es muy importante de cara a hacer un buen proyecto ya que es la que se hizo justo después de haber hecho el análisis de mercado y haber entendido la problemática del proyecto. Como bien dice el título de este capítulo, su principal objetivo es planificar todas las fases del proyecto y estructurarlo en base al tiempo. Hablaremos también de qué desviaciones y alternativas puede sufrir y en esencia cuáles he sufrido durante el proyecto y de qué recursos disponemos, tanto a nivel material como humano. Por último se mostrará un diagrama de Gantt respecto a la planificación inicial, el cual expondrá el tiempo de dedicación previsto para las diferentes tareas o actividades a lo largo del tiempo total determinado así como el diagrama de Gantt final que ha resultado de hacer el proyecto.

8.1 Fases del proyecto

Seguidamente se van a detallar las distintas en fases que se han llevado a cabo a lo largo del proyecto, desde la formación en las tecnologías y conceptos de la empresa, pasando por toda la planificación / diseño del proyecto y finalizando con la explicación de todo el proceso de desarrollo y puesta en producción del software.

8.1.1 Formación y puesta en marcha

Esta primera fase principalmente fue dedicada a adquirir conocimiento sobre el dominio y contexto del proyecto ya que era necesario para el desarrollo del mismo, la preparación del entorno de trabajo y la consolidación de conocimientos sobre las tecnologías que fueron necesarias para la implementación de los componentes (tanto *frontend* como *backend)*, *testing* y despliegue del *software* en la nube.

8.1.2 Planificación del proyecto

La segunda fase, ha englobado todas las entregas que se han llevado a cabo en la asignatura de GEP. Es una de las fases más importantes del proyecto ya que en ella se han definido todos los aspectos del proyecto en cuanto a qué, cómo y cuándo se va a realizar, junto a la gestión económica y sostenibilidad del proyecto.

Una vez finalizada esta fase del proyecto ya quedaron explicados todos los aspectos no relacionados directamente con el proyecto y el trabajo que quedó por hacer fuer todo el contenido relacionado con el proyecto.

8.1.3 Desarrollo del proyecto

La tercera fase, fue dedicada principalmente a especificar los distintos procesos que hay que seguir a la hora de desarrollar cualquier tipo de software.

- ❏ Análisis y diseño del software

La primera fase en el desarrollo del *software* consistió en analizar la solución, determinando lo que se pretendía solucionar y/o implementar. Para ello se necesitó recoger y especificar los requisitos del proyecto conjuntamente con el equipo y/o el cliente. También se detallaron todos los objetivos, características y casos de uso que el software tenía que satisfacer.

Una vez acabado el análisis se procedió a hacer un diseño técnico de cómo iban a ser los componentes de dicho software, es decir, se generó la arquitectura de software, interacción de los componentes, explicación de los casos de uso, diseño de la interficie de los componentes, etc..

- Implementación y testeo del software

Como se mencionó anteriormente, en el capítulo 4, durante el desarrollo de este proyecto se siguieron los principios del Scrumban. Dicha metodología estipula que el desarrollo de todo software será planificado en iteraciones de aproximadamente dos semanas. Cada iteración tiene que proporcionar un resultado completo, un incremento del software susceptible de ser entregado con el mínimo esfuerzo al cliente. Las actividades que se llevaron a cabo en cada una de estas iteraciones fueron: la planificación de la iteración, la ejecución de dicha iteración y finalmente, la inspección y adaptación de las tareas realizadas. Así que en esta segunda fase también se realizó toda la parte de validación y/o supervisión de las tareas que se fueron realizando.

Al final fueron necesarias cuatro iteraciones para el desarrollo de todo el proyecto, tal como hemos podido ver en el apartado 6.4:

- La primera iteración se realizó todo lo correspondiente a la instalación y configuración de Auth0, el servicio de autenticación que se utilizó para la verificación y/o validación de las autenticaciones.

- La segunda iteración fue dedicada principalmente a la implementación del microservicio JWTProxy y de la creación y supervisión de sus *tests* para pasar los controles de calidad de software que pedía Safelayer.

- La tercera iteración, como hemos podido ver parecida a la segunda, fue dedicada principalmente a la implementación del microservicio DBProxy y de la creación y supervisión de sus respectivos tests.

- La cuarta y última iteración se realizó todo lo correspondiente a la implementación y/o integración del componente de autenticación en la plataforma web de Safelayer.

8.1.4 Despliegue del software en el cloud

Una vez desarrollados y testeados los distintos componentes implementados, se procedió a poner en producción dicho *software* en el *cloud*, permitiendo la accesibilidad y disponibilidad del producto a los clientes.

Para la realización de dicha fase, como hemos podido ver en el apartado 5.6.1, se utilizaron los servicios proporcionados por Amazon ya que eran los que se estaban utilizando actualmente en la empresa y proporcionaban todo lo necesario para el desarrollo de este proyecto también.

8.1.5 Memoria y preparación de la defensa

En esta última fase del proyecto se procedió a redactar la memoria final del proyecto y la preparación de la presentación ante el tribunal. Gracias a la metodología de trabajo que se ha llevado a cabo a lo largo del proyecto, a la hora de redactar dicha memoria se disponía de cortos informes y/o informes de las distintas fases y procedimientos que se han ido realizando, por lo que no fue muy difícil realizar la documentación final del proyecto.

La preparación de la presentación y/o defensa del proyecto también gracias a las reuniones periódicas que se han ido llevando a cabo durante la realización de este, en las cuales se debía explicar/presentar delante del coordinador y el *project manager* los avances que se iban realizando.

8.2 Calendario

8.2.1 Planificación estimada

El proyecto tenía una durada aproximada de seis meses, con el inicio el día 25 de Septiembre de 2017, fecha que coincide con el dia de contratación por parte de Safelayer, y con fecha de límite el 30 de Marzo de 2018, fecha de finalización del convenio. Con estas fechas se pudo disponer de un margen de unos 20 días antes de la presentación ante el tribunal para posibles desviaciones que pudiesen surgir y/o para la supervisión de la memoria y/o de la defensa del proyecto. Tiene una carga mínima de trabajo de 735 horas (mínimo establecido por la UPC para proyectos realizados en empresa) y máxima de 900 horas aproximadamente (máximo establecido por la UPC para proyectos realizados en empresa).

8.2.2 Carga de trabajo estimada

Seguidamente tenemos en forma de tabla la estimación aproximada hecha de las horas de trabajo de las distintas fases del proyecto:

Tarea	Cálculo	Horas
1. Formación y puesta en marcha	Diferencia entre el mínimo de horas y el máximo permitido por convenio.	165
2. Planificación del proyecto	3 ECTS (25 horas * 3)	75
3. Desarrollo del proyecto		
3.1 Análisis y diseño del software	Planificación aproximada	140
3.2 Implementación y testeo del software	3 sprints (80 horas cada uno aproximadamente)	240
4. Despliegue del software en el cloud	Planificación aproximada	100
5. Memoria y preparación de la defensa	Planificación aproximada	80
Total		**800**

Tabla 1. Estimación de horas

8.2.3 Diagrama de Gantt

El diagrama de Gantt es una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas a lo largo de un tiempo total determinado. En los anexos se muestra el diagrama correspondiente a la planificación inicial del proyecto y otro para la planificación final. Las dos primeras fases tienen una parte que se hace en paralelo debido a que mi incorporación en Safelayer fue posterior al comienzo de GEP.

Al final de la planificación inicial se añadió una reserva de 100 horas para posibles desviaciones que pudiesen llegar a surgir a lo largo de las fases desarrollo o en las propias fases del proyecto.

8.3 Plan de acción y validación de las alternativas

Por un lado, como podemos ver en la *Tabla 1* se contaba con un primer margen de hasta 100 horas hasta la finalización del convenio con la empresa. Por otro lado, también se disponía de un segundo margen de hasta 23-27 días desde la finalización del convenio hasta la presentación del proyecto. Este segundo margen podía ser anulado si se notificaba que las fechas de defensa del proyecto delante del tribunal iban a ser anteriores a las establecidas por la facultad.

Se podían llegar a producir desviaciones temporales tanto en la fase de aprendizaje y/o formación en las tecnologías, conceptos y herramientas de la empresa como en las distintas fases del desarrollo del *software* y/o iteraciones. Estas desviaciones se podían llegar a suceder en consecuencia de haber dedicado más o menos tiempo de lo estimado inicialmente en las distintas tareas.

Las medidas correctivas que se podían llegado a aplicar son o bien, cancelar la tarea que se estaba desarrollando, o bien, modificarla de tal manera que sea más fácil de tratar. En caso de que hubiese sido alguna tarea crítica se podía llegar a cancelar y / o dejar para una segunda versión a desarrollar como ingeniero en la

empresa para que no se llegase a perjudicar la calidad de los componentes deseados para el proyecto.

Finalmente para la realización del proyecto hizo falta una cuarta iteración que no se tenía prevista en un principio y como hemos mencionado anteriormente se disponía de 100 horas adicionales, así que se pudo realizar sin ningún problema. De estas 100 horas se destinaron a la nueva iteración 80 horas, es decir, un periodo de dos semanas, igual que las otras iteraciones. Las 20 horas restantes se destinaron al diseño de los componentes y de la arquitectura del proyecto. En la siguiente tabla se puede ver el desglose de horas final dedicadas al proyecto:

8.3.1 Total de horas dedicadas

Tarea	Horas
Formación y puesta en marcha	165
Planificación del proyecto	75
Análisis y diseño del software	160
Implementación y testeo del software	320
Despliegue del software en el cloud	100
Memoria y preparación de la defensa	80
Total	**900**

Tabla 2. Total de horas dedicadas al proyecto

También comentar que por motivos personales y/o familiares ajenos al proyecto tuve que ausentarme del trabajo durante un periodo de aproximadamente 20 días durante el mes de Noviembre, por lo que no se realizó ni documentación ni implementación de software. Como se disponía de un margen extra de días desde la finalización del contrato hasta la presentación del proyecto se pudo aplazar todo el trabajo a realizar previsto en un inicio. Igual que la incorporación de una nueva iteración, este acontecimiento se podrá ver reflejado en el diagrama de Gantt final disponible en los anexos.

8.4 Recursos para el desarrollo del proyecto

En este apartado estableceremos los recursos, tanto humanos como materiales, que han sido necesarios para el completo desarrollo del proyecto.

8.4.1 Recursos humanos

Referente a los recursos humanos se ha incluído el personal de la empresa implicado en el proyecto:

Rol	Funciones
Desarrollador (Yo)	Planificar, diseñar, implementar, testear y desplegar el software.
Arquitecto de Software & Technical Leader	Dirección del proyecto y encargado de las tecnologías que se usan en el desarrollo.
Product Manager	Encargado de dirigir el equipo de desarrollo.

Tabla 3: Recursos humanos

8.4.2 Recursos materiales

Para los recursos materiales se han tenido en cuenta tanto los recursos de *hardware* como los de *software:*

Material	Descripción
PC sobremesa	Máquina Ubuntu 16.04 LTS
Git	Controlador de versiones del código.
Visual Studio Code	Entorno de desarrollo del proyecto.
Herramientas Google Drive	Gestión de los documentos del proyecto.
Herramientas Atlassian (Confluence, Jira..)	Gestión de los documentos y/o comunicados de la empresa.

Tabla 4: Recursos materiales

Capítulo 10: Gestión económica

9.1 Identificación de los costes

Partiendo de que este proyecto se ha realizado en la empresa Safelayer Secure Communications con un contrato de 900 horas, la mayoría de los costes de este proyecto vendrán definidos por el personal de RRHH (Recursos humanos) y por los recursos mencionados anteriormente. Por temas de privacidad la empresa no ha podido proporcionar dichos costes, por lo que se han utilizado estimaciones basadas en estudios realizados. A continuación vamos a identificar los costes en distintas categorías:

Identificación de los costes del proyecto		
Directos	**Indirecto**	**Control de gestión**
Mano de obra	Consumo eléctrico de la oficina	Amortizaciones
- Ingeniero software y tester	Conexión a Internet	Contingencias
- Project Manager	Alquiler del local	
- Tech Leader	Transporte al lugar de trabajo	
Hardware		
- Dell Precision T3420		
- DELL 1907FP (x2)		
- Teclado y mouse DELL		
Software		
- Visual Studio Code		
- Jira		
- Confluence		
- Bitbucket		

Tabla 5: Identificación de los costes del proyecto

9.2 Estimación de los costes

En la siguiente tabla se muestran los salarios de cada rol. Por temas de privacidad la empresa no proporciona los sueldos de dichos roles, por lo que se han basado en el estudio de remuneración de Michael Page del año 2017 [29].

Rol	Sueldo bruto (€/h)
Ingeniero de Software y Tester (IST)	10
Project Manager (PM)	26
Tech Leader (TL)	28

Tabla 6: Asignación de salarios

9.2.1 Costes directos del proyecto

9.2.1.1 Coste recursos humanos

Fase	Duración total (horas)	Dedicación por rol (horas)	Coste (€)
Formación y puesta en marcha	205	(IST) - 165 (PM) - 40	2690
Planificación del proyecto	99	(IST) - 75 (PM) - 16 (TL) - 8	1390
Análisis y diseño del software	180	(IST) - 160 (TL) - 40	2720
Implementación y testeo del software	272	(IST) - 320 (PM) - 32	4032
Despliegue del software en el cloud	140	(IST) - 100 (TL) - 40	2120
Memoria y preparación de la defensa	120	(IST) - 80 (PM) - 24 (TL) - 16	2152
TOTAL	**15.104 €**		

Tabla 7: Coste de recursos humanos

Las horas dedicadas como Ingeniero de Software y Tester, es decir, por mi mismo, han sido extraídas de la planificación temporal. A esta, se le deben sumar las horas de dedicación que tienen el personal restante implicado en este proyecto.

9.2.1.2 Coste de herramientas de hardware

Material	Unidades	Coste (€)
DELL Precision T3420	1	1600
Monitor DELL 1907FP	2	150
Teclado DELL	1	35
Mouse DELL	1	15
TOTAL	**1.800 €**	

Tabla 8: Coste de las herramientas de hardware

Los precios de las herramientas anteriores son proporcionados por personal de la empresa ya que es una configuración y precio a medida hecho por parte de DELL [30]. El coste de los recursos materiales se obtiene de realizar el cálculo de amortización del equipo personal, el hardware: *(Precio de compra del equipo personal / (años de amortización * días laborables al año * horas de uso diario)) * Horas de contrato con la empresa* = (1800 / 4*230*8)) * 900 = **220,10 €**

9.2.1.3 Coste de herramientas de software

Software	Licencias	Coste (€)
Visual Studio Code	1	0
Jira	1	42
Confluence	1	30
Bitbucket	1	30
TOTAL	**102 €**	

Tabla 9: Coste de las herramientas de software

Los productos de Atlassian se pagan por licencia de usuario reservada. El coste resultante de las herramientas de software se obtienen de aplicar el coste unitario de licencia al mes por la duración del proyecto, que en mi caso son aproximadamente 6 meses, tal como se mencionó en la planificación de la entrega anterior: *(Precio licencia Jira + precio licencia Bitbucket + precio licencia Confluence) * duración del proyecto = (7 + 5 + 5) € /mes * 6 meses =* ***102 €.***

9.2.2 Costes indirectos del proyecto

9.2.2.1 Consumo eléctrico de la oficina

A partir de la factura eléctrica de la empresa se ha calculado un coste fijo de 0,13€/h y un coste por energía consumida de 0,14€/KWh. Teniendo en cuenta que el equipo personal tiene un consumo de 240 W y que las horas totales de uso son 800, el coste del consumo de electricidad derivado por el equipo personal es: *(coste fijo + (coste energía consumida * consumo equipo))* horas de contrato con la empresa*= (0,13+ (0,14*0,240))*900 = **130,88 €.**

9.2.2.2 Conexión a internet

La cuota de acceso a Internet es de 250€/mes, por tanto, el coste de conexión a Internet es: *(cuota acceso Internet/días mes/horas día) * horas de contrato con la empresa= (250/30/24) * 900* = **277,78 €.**

9.2.2.3 Alquiler del local

El coste mensual del alquiler del local es de 5000€/mes, calculado de un precio fijo de 20€/m2 y con un total de 250 m2 que tiene la oficina. La empresa cuenta con 53 trabajadores, por tanto, el coste de alquiler del local para este proyecto es de: *(coste del local / número de trabajadores / horas laborables al mes)* número de personas trabajando en el proyecto* horas de contrato con la empresa* = (5000/53/240) * 1 * 900 = **314,46 €.**

9.2.2.4 Transporte

El coste trimestral de ir al lugar de trabajo en mi caso son 199,20 € [31] ya que mi domicilio está a 3 zonas del lugar del trabajo. Con la previsión aproximada de una durada total de, 6 meses, el coste del transporte al lugar de trabajo es: *precio por tarjeta * número de tarjetas = 199,20 * 2* = **398,4 €.**

9.2.3 Costes de control de gestión

9.2.3.1 Amortizaciones

En este trabajo se ha considerado una amortización del hardware de 4 años, ya incluida en el cálculo de costes de recursos materiales. Así mismo también el software queda amortizado, ya que Visual Studio Code es gratuito y los productos de Atlassian utilizados en la gestión del proyecto se usan en todos los ámbitos de la empresa para todos los trabajadores. En esencia, sólo se pagan licencias por usuario. Por lo tanto tanto, el coste de amortizaciones es **0 €**.

9.2.3.2 Contingencias

Tal como se mencionó previamente, se destinaron 100 horas para imprevistos y desviaciones de las tareas. Estas horas se han contabilizado como una tarea en la asignación de recursos humanos, en concreto, en la fase de desarrollo. Este coste no se verá afectado por los recursos destinados a herramientas de software ya que las licencias de usuario se pagan mensualmente y dichas horas ya están incluídas dentro de la fase de desarrollo.

Tampoco se tendrán en cuenta el coste de las contingencias en hardware ni el coste de las contingencias de los costes indirectos, ya que el el cálculo hecho en los respectivos apartados se han incluido las horas previstas para imprevistos y desviaciones.

Por lo tanto, el coste total de las contingencias es: *coste de contingencias de software + coste contingencias de hardware + contingencias de los costes indirectos (internet, alquiler, consumo y transporte) + coste de contingencias de recursos humanos = 0 + 0 + 0 + (100 horas * 10 euros/hora)* = **1000 €**.

9.3 Coste total

Una vez analizados los costes del proyecto, solo hay que sumar los costes directos, los costes indirectos y los costes del control de gestión para saber el precio total del proyecto, cabe mencionar que en la siguiente tabla se han añadido las contingencias sufridas dentro del coste de recursos humanos:

Costes	Concepto	Precio (€)
Costes directos	Coste recursos humanos	15.104
	Coste herramientas hardware	220,10
	Coste herramientas software	102
Costes indirectos	Consumo eléctrico	130,88
	Conexión a Internet	277,78
	Alquiler del local	314,46
	Transporte	398,4
Control de gestión	Amortizaciones	0
Coste Total	**16.547,62 €**	

Tabla 10: Coste total del proyecto

Capítulo 10: Sostenibilidad y compromiso social

10.1 Evaluación de sostenibilidad

En esta parte del informe, evaluaremos cómo afecta nuestro proyecto a las diferentes dimensiones de las que está compuesta la sostenibilidad:

- ❏ Dimensión económica
- ❏ Dimensión social
- ❏ Dimensión ambiental

10.1.1 Dimensión económica

- ❏ ¿Existe una evaluación de los costes, tanto recursos materiales como humanos? ¿Se ha tenido en cuenta el coste de actualización y reparación de los equipos durante la vida útil del proyecto? ¿Se podría realizar un proyecto similar en menos tiempo/recursos y por lo tanto menor coste?

Los recursos tanto materiales como humanos están especificados en el apartado uno. Respecto a la segunda pregunta no se ha tenido en cuenta l actualización y reparación de los equipos ya que como se ha mencionado anteriormente el proyecto tiene una previsión de duración de seis meses, en dicho tiempo, nunca ha habido ni se prevé que hay una actualización de equipo ya que ha sido recientemente adquirido por parte de la empresa.

Se podría llegar a realizar en mentor tiempo ya que existe una fase de adaptación a los productos de la empresa y/o tecnologías, así que con un personal más especializado se podría llegar a conseguir un menor coste en herramientas de software y/o hardware. No obstante, conseguir un personal más especializado aumentaría los costes de recursos humanos, así que probablemente encarecería el proyecto a pesar de hacerlo en menor tiempo.

10.1.2 Dimensión social

- ¿Hay una necesidad del proyecto? ¿En qué mejora la calidad de vida de los consumidores este proyecto? ¿En qué cambiará la vida del usuario? Alguien se ve perjudicado por este proyecto?

Como vimos en la primera entrega la necesidad de realizar el proyecto nace de la capacidad que han tenido otras empresas (competidoras) en realizar algo muy parecido a este proyecto. Ahora bien, la novedad y lo que mejorará la calidad de vida de los consumidores de este proyecto recae en la innovación de poder gestionar diferentes PKI independientemente de la fuente de origen de esta en un mismo portal web.

Esta innovación hará mucho más cómoda y fácil la gestión de las PKI y hasta ahora, no existe nada igual. Los usuarios finales y/o usuarios de corporaciones podrán acceder fácilmente desde distintos sistemas de autentificación a la plataforma web que les permitirá gestionar fácilmente la documentación y todas las gestiones pertinentes que necesiten de una identidad digital y/o certificado de autentificación. Las únicas personas que se verán perjudicadas con este proyecto son las empresas competidoras ya que no ofrecen dicho servicio.

10.1.3 Dimensión ambiental

- ¿Qué consumo tendrán los recursos durante la fase de desarrollo del proyecto y posteriormente durante su vida útil? ¿Se han tenido en cuenta criterios para facilitar su posterior reciclaje? ¿Las fuentes de extracción de los materiales usados en tu proyecto tienen alguna implicación ética? ¿Con tu proyecto, reducirás o aumentarás la pisada ecológica? ¿Qué recursos se necesitarán en las diferentes fases del proyecto? ¿Qué consumo tendrán estos recursos durante el desarrollo del proyecto y posteriormente durante su puesta en marcha y vida útil? ¿Cuál es el impacto ambiental de este consumo (medido en toneladas de CO2, por ejemplo?)

Principalmente los recursos que voy a consumir durante el proyecto serán eléctricos. Esto es debido a que el hardware que uso requiere de ello para su funcionamiento. No obstante, durante las fases iniciales y finales (planificación del proyecto, análisis y diseño, desarrollo de la memoria, etc) se usará bastantes hojas de papel.

En segundo lugar, el principal consumo que tendrá mi proyecto durante y posterior a la fase de desarrollo será eléctrico debido a que el hardware que uso principalmente funciona mediante energía eléctrica. Por lo que concierne a la tercera pregunta, por motivos de privacidad de la empresa no podemos proporcionar información sobre los métodos de reciclaje de los componentes usados en el proyecto, pero sabemos que los componentes que no se pueden reutilizar finalmente son destinados a plantas de reciclaje.

Los materiales usados para fabricar componentes electrónicos (para ordenadores sobremesa y/o pantallas, etc) son comunes, excepto por el coltán que es bien conocido por sus implicaciones éticas (explotación infantil, conflictos bélicos, malas condiciones de trabajo, etc) en países como la República Democrática del Congo [32]. Por último, este proyecto pretende reducir el coste económico y a su vez ecológico de la actividad de operar con diferentes PKI en un mismo servicio (portal web) en la nube, ya que se ahorra una instalación para cada máquina que necesite el servicio. Así mismo, mediante el uso de transporte público al lugar de trabajo se contribuye positivamente a la reducción de la pisada ecológica.

10.2 Matriz de sostenibilidad

	PPP	Vida Útil	Riesgos
Ambiental	Consumo del diseño	Huella ecológica	Riesgos ambientales
	7	17	-3
Económico	Factura	Plan de viabilidad	Riesgos económicos
	7	18	-5
Social	Impacto personal	Impacto social	Riesgos sociales
	8	18	-3
Rango Sostenibilidad	22	53	-11
	64		

Tabla 11: Matriz de sostenibilidad

Capítulo 11 - Identificación de leyes y regulaciones

En este capítulo se pretende mostrar aquellas leyes y regulaciones que mi proyecto abarca. Comentar que son leyes impartidas por la Directiva Europea y no se entrará en detalle en estas pues está fuera del alcance del proyecto. Sin embargo, es información útil de cara al proyecto y brevemente citaré dichas leyes y regulaciones y las pondré como referencias al final de la memoria.

La familia de productos KeyOne implementa una solución de Infraestructura de Clave Pública (PKI), que cumple con la Directiva Europea 1999/93/CE [33] sobre firma electrónica y con la Regulación 910/2014 sobre identidad electrónica y servicios de confianza, conocida como Regulación eIDAS [34], aplicable a partir del 1 de julio de 2016.

Además, la solución KeyOne ha sido diseñada para cumplir todos los requisitos de seguridad de los sistemas de gestión de certificado de firma electrónica especificados en CWA 14167-1 [35] y CEN TS 419 261 . El documento CWA 14167-1 es reconocido por la Comisión Europea como la normativa técnica para los productos de firma electrónica. El documento CEN TS 419 261 describe los requisitos de seguridad y componentes tecnológicos que deben cumplir los Proveedores de Servicios de Confianza que emitan certificados electrónicos y/o sellos de tiempo electrónicos en el marco regulatorio de eIDAS .

Por último, comentar que los productos KeyOne están certificados según los Criterios Comunes con nivel EAL4+ [36] declarando conformidad con el Perfil de Protección NSA/NIST Certificate Issuing and Management Components (CIMC) de nivel de seguridad 3.

Capítulo 12: Conclusiones y futuras mejoras

12.1 Conclusiones

Las conclusiones a las que se han llegado al finalizar este proyecto junto a Safelayer Secure Communications S.A han sido las que comentaremos seguidamente, las dividiremos en distintos apartados: dificultades durante el proyecto, integración de conocimientos y conclusiones que se han sacado del prototipo realizado:

12.1.1 Integración de conocimientos

Durante la carrera los estudiantes aprendemos gran multitud de conocimientos para, una vez finalizada, llegar a aplicarlos en situaciones reales. Este proyecto es un claro ejemplo de ello, ya que no aporta ningún tipo de conocimiento nuevo a la sociedad, sino que adapta e integra técnicas ya existentes para dar soluciones. Los conocimientos que se han aplicado, algunos de ellos ya explicados en el capítulo 5 han sido los siguientes:

- **API - HTTP/JSON:** nos ha permitido de forma simple comunicar la aplicación con un servidor y/o servicios externos a través de Internet sin la necesidad directa de uso de sockets IP/TCP. Este protocolo se ha aprendido en la asignatura de Aplicaciones y Servicio Web (ASW).

- **Aplicaciones web:** es el principal objetivo del proyecto, la integración del componente en una interfaz web. Para ello se han tenido que tener claros conceptos tanto del *frontend* como del *backend*. Conceptos que se han aprendido también en ASW.

- **Gestión de proyectos:** en todo el ciclo de vida de un proyecto software la gestión que se hace sobre él es muy importante y primordial para un desarrollo correcto. Es por eso que asignaturas cursadas como Gestión de

Proyectos Software (GPS) nos ayudan a planificar un proyecto y gestionarlo de una forma correcta junto al equipo.

- **Desarrollo de un proyecto:** es importante tener conocimientos de gestión y conocimientos tecnológicos. Ahora bien, también es muy importante tener fluidez desarrollando software y afrontando proyecto reales y de gran dimensión. Es por eso que la asignatura de Proyectos de Ingeniería de Software (PES), la cual ha sido cursada también, nos prepara para encarar y desarrollar un proyecto de una forma más eficaz.

- **Testeo de aplicaciones:** en todo el ciclo de vida de un proyecto *software* para asegurar todos los requisitos de calidad y funcionales del mismo, tal como hemos visto con la práctica de la entrega contínua (*continuous delivery).* Conceptos como el de *testing* han sido adquiridos gracias a asignaturas como Gestión de Proyectos Software (GPS) y Proyecto de Ingeniería del Software (PES).

- **Bases de datos:** otro aspecto muy importante en la implementación de una aplicación web y sus servicios es el concepto de base de datos. En concreto, cómo gestionar una base de datos muy grande de una forma eficiente y con un elevado rendimiento, junto a los distintos tipos de base de datos existentes. En asignaturas como Diseño de Base de Datos (DBD) se adquieren dichos conocimientos.

12.1.2 Dificultades

- **Lenguaje de programación Go**: una de las principales dificultades que se han encontrado en este proyecto ha sido el aprendizaje de un nuevo lenguaje de programación como Go, el cual presentó un cambio de sintaxis que para mi fue relevante debido a que no estaba acostumbrado a ese tipo de lenguajes. Pero una vez se adquirió dicho conocimiento, fue fácil implementar lo que se necesitaba.

- **Arquitectura del software del proyecto:** otra de las principales dificultades que se han encontrado a lo largo del proyecto fue la utilización de los servicios que proporcionaba Amazon debido a que tienen una gran variedad de servicios que realizan funciones muy específicas. En un principio parece muy complejo debido a que se necesitan conocimientos en múltiples áreas de la informática y no solo de Ingeniería del Software. Pero gracias a la formación recibida en la empresa se fue adquiriendo práctica en la utilización de muchos de estos servicios.

12.1.3 Conclusiones

En esta sección pretendo explicar qué conclusiones saco de este proyecto, así como intentaré explicar aquellas cosas que no se han podido conseguir y qué me ha aportado personalmente el hecho de hacer este proyecto dentro de una empresa. Una vez visto si se han cumplido o no los objetivos explicaré la parte más personal, es decir, qué conclusiones me llevo a nivel personal de este proyecto y qué cosas he aprendido.

Una vez llegados a este punto, como se ha podido ver a lo largo de la memoria, el principal objetivo ha sido al alcanzado con éxito, pues se ha completado e integrado de forma satisfactoria el componente de autenticación en la plataforma y es plenamente funcional tanto para usuarios como para los trabajadores de una organización.

Además se ha conseguido poner en práctica todos los conceptos del *continuous delivery* de tal forma que se ha logrado desplegar la aplicación *on-premises*, es decir usando máquinas de casa de la organización así como en el *cloud* usando servicios de un proveedor como ha sido Amazon. Por lo tanto, los principales objetivos se han cumplido satisfactoriamente.

Sin embargo, hay que comentar que ciertos objetivos, menos importantes para mi TFG no se han llegado a alcanzar debido al marco de tiempo establecido en este proyecto. Entre ellos todo el tema de gestión de certificados digitales, es decir,

los certificados digitales que verifica el segundo microservicio (DBProxy), pues para ello hace falta una nueva solución que implica la creación de todo un portal web de administración y el diseño de una arquitectura funcional y escalable. Pero dichas mejoras se pretenden resolver en un futuro, donde la empresa Safelayer deberá decidir si apostar o no en este proyecto. Por lo tanto, el primer prototipo del componente de autenticación ha sido completado con éxito.

Una vez vistos los objetivos más técnicos del proyecto y las conclusiones técnicas del mismo, es hora de explicar un poco más las conclusiones personales que yo saco de este proyecto. Para empezar, y siguiendo un poco el hilo con el párrafo anterior, este proyecto me ha servido para formarme en tecnologías muy importantes hoy en día en el área de la Seguridad Informática. Entre ellas, la infraestructura de clave pública o PKI, que cada vez está más implementada en la sociedad actual.

Otros conceptos como el despliegue de aplicaciones web y microservicios en el *cloud* y/o las prácticas actuales en el software como la integración y la entrega continua son esenciales para cualquier ingeniero que quiera dedicarse al desarrollo de *software.* Además, conceptos como testeo o arquitectura de una aplicación, tanto a nivel de *frontend* como *backend,* y la utilización de servicios y/o soluciones por terceros han sido adquiridos gracias a este proyecto.

Para finalizar, no hay que olvidar que es un proyecto que se realiza dentro de una empresa, y por lo tanto, hay otras competencias a tener en cuenta. Estas competencias o habilidades las he ido aprendiendo a lo largo que desarrollaba el proyecto, y es que el hecho de estar usando una metodología ágil te ayuda a resolver problemas más fácilmente. También he aprendido que dentro de una empresa hay diferentes roles y que cada uno tiene unas funciones definidas, por no hablar de la importancia de la comunicación dentro de un equipo. Dicho esto, hemos podido comprobar que este proyecto cumple su cometido y además, a nivel personal me ha aportado mucho, por lo tanto valoro muy positivamente esta experiencia en Safelayer y estoy muy satisfecho con el trabajo realizado.

12.2 Futuras mejoras

A continuación se comentarán las futuras mejoras que se pretenden realizar en la prueba piloto que se ha realizado en este proyecto junto a las posibles mejores de la plataforma web que hay detrás del componente de autenticación implementado:

- ❏ Creación de un portal web de administración para la gestión de certificados digitales de usuarios válidos en la autenticación.

- ❏ Pensar en el modelo de datos y en la implementación de los microservicios necesarios para gestionar dichos certificados.

- ❏ Seguir ampliando las funcionalidades que se ofrecen en la plataforma web ajustándose al potencial de KeyOne, el software de Safelayer desplegado en la nube.

- ❏ Una vez implementadas estas nuevas funcionalidades se necesitará contratar a un experto en interfaces gráficas para que diseñe una aplicación lo más usable e intuitiva para los usuarios. Dicho diseño deberá ser implementado por los desarrolladores de la aplicación. De éste modo Safelayer estará más cerca de su principal objetivo, llegar a ofrecer la PKI como servicio.

Hasta ahora, se ha conseguido desplegar la Autoridad de Registro (RA) tanto en máquinas de la propia organización como en máquinas alojadas en la nube. Sin embargo, no deja de ser una PKI que ofrece en este caso Safelayer. Es decir, de una infraestructura de clave pública que necesita de ciertos elementos para ser funcional pues hace falta configurar la jerarquía de certificación, validación, mantenimiento, etc. Esto significa que los clientes que estén interesados en el producto deben tener personas dedicadas a la gestión de toda la infraestructura PKI, así como de formación en el producto que ofrece Safelayer. Como se ha

podido observar, el despliegue de una PKI supone un problema que los clientes tienen, pues implica tener recursos tanto a nivel humano, tecnológico como económico.

En este punto es donde surge la ide idea de ofrecer estas PKI como un servicio. Es decir, Safelayer se encargaría de toda la gestión de las PKI e infraestructura y los clientes sólo contratarían este servicio. Esto reduce notablemente los recursos a intervertir por parte de los clientes ya que no haría falta personal que mantenga la PKI ni destinar tantos recursos en formarse (pues son tecnologías y conceptos complejos que tienen una curva de aprendizaje bastante elevada). Si se decide apostar por esta iniciativa, el portal web podría ser común para cualquier cliente y/o organización que contrata el servicio de gestión de PKI y se podría unificar con aquellos clientes que decidan seguir desplegando la PKI de Safelayer en un entorno *on-premises.* En esencia, esta parte no es trivial de implementar y es una opción a valorar de cara al futuro de este proyecto.

Referencias

[1] Safelayer Secure Communications S.A. *"About the company"*, consultada el 18 de Septiembre de 2017. Fuente*:* https://www.safelayer.com/en

[2] Referencias Proyectos I+D de Safelayer, consultada el 18 de Octubre de 2017. Fuente: https://www.safelayer.com/es/referencias/proyectos-id

[3] Software de autentificación Auth0, consultada el 18 de Octubre de 2017. Fuente: https://auth0.com/

[4] Protocolo ligero de acceso a directorios (LDAP), consultada el 18 de Octubre de 2017. Fuente: http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ldap.html

[5] Inhouse PKI vs Managed PKI, consultada el 28 de Setiembre de 2017. Fuente: https://www.globalsign.com/en-sg/inhouse-pki-vs-managed-pki/

[6] Identity and security company Nexus Group, consultada el 18 de Octubre de 2017. Fuente: https://www.nexusgroup.com/

[7] Primekey Solutions, consultada el 18 de Octubre de 2017. Fuente: https://www.primekey.com/

[8] Entrust Datacard Corporation, consultada el 18 de Octubre de 2017. Fuente: https://www.entrustdatacard.com/

[9] Metodología Scrumban, consultada el 18 de Setiembre de 2017. Fuente: https://www.obs-edu.com/es/blog-project-management/temas-actuales-de-project-management/la-metodologia-scrumban-cuando-y-por-que-utilizarla

[10] Bitbucket: La solución Git para equipos profesionales, consultada el 18 de Octubre de 2017. Fuente: https://es.atlassian.com/software/bitbucket

[11] Jira: Software de seguimiento de de proyectos e incidencias, consultada el 18 de Octubre de 2017. Fuente: https://es.atlassian.com/software/jira

[12] Confluence: Software de colaboración para equipos, consultada el 18 de Octubre de 2017. Fuente: https://es.atlassian.com/software/confluence

[13] Qué son y para qué sirven los hash, consultada el 25 de Octubre de 2017. Fuente:https://www.genbetadev.com/seguridad-informatica/que-son-y-para-que-sirven-los-hash-funciones-de-resumen-y-firmas-digitales

[14] What Exactly is a Web Application?, consultada el 25 de Octubre de 2017. Fuente: https://www.lifewire.com/what-is-a-web-application-3486637

[15] Single-page Applications, consultada el 25 de Octubre de 2017. Fuente: https://www.codeschool.com/beginners-guide-to-web-development/single-page-applications

[16] React - A Javascript library for building user interfaces, consultada el 25 de Octubre de 2017. Fuente: https://reactjs.org/

[17] Introducción a Redux.js, consultada el 25 de Octubre de 2017. Fuente: https://medium.com/react-redux/introducci%C3%B3n-a-redux-js-8bdf4fe0751e

[18] Web Services Architecture, consultada el 25 de Octubre de 2017. Fuente: https://www.w3.org/TR/ws-arch/

[19] Microservices vs Monolithic Architecture, consultada el 25 de Octubre de 2017. Fuente: https://www.mulesoft.com/resources/api/microservices-vs-monolithic

[20] What is containerization?, consultada el 26 de Octubre de 2017. Fuente: https://searchservervirtualization.techtarget.com/definition/container-based-virtualization-operating-system-level-virtualization

[21] Containerization vs Virtualization, consultada el 26 de Octubre de 2017. Fuente: https://jaxenter.com/containerization-vs-virtualization-docker-introduction-120562.html

[22] What are containers and why do you need them?, consultada el 26 de Octubre de 2017. Fuente: https://www.cio.com/article/2924995/software/what-are-containers-and-why-do-you-need-them.html

[23] ¿Qué es Docker?, consultada el 26 de Octubre de 2017. Fuente: https://www.redhat.com/es/topics/containers/what-is-docker

[24] Docker Hub, consultada el 26 de Octubre de 2017. Fuente: https://hub.docker.com/

[25] Integración continua del software | Pruebas automatizadas, consultada el 27 de Octubre de 2017. Fuente:https://aws.amazon.com/es/devops/continuous-integration/

[26] Travis CI - Test and Deploy with Confidence, consultada el 27 de Octubre de 2017. Fuente: https://travis-ci.com/

[27] Continuous integration vs continuous delivery vs continuous deployment, consultada el 27 de Octubre de 2017. Fuente: https://www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd

[28] Amazon Web Services (AWS) - Cloud Computing Services, consultada el 28 de Octubre de 2017. Fuente: https://aws.amazon.com/

[29] Estudio de Remuneración 2017, consultado el 8 de Octubre de 2017. Fuente: https://www.michaelpage.es/sites/michaelpage.es/files/PG_ER_IT.pdf

[30] Ordenador sobremesa Precision T3420, consultado el 8 de Octubre de 2017. Fuente:http://www.dell.com/es/empresas/p/precision-t3x20-series-workstation/pd?oc=xctop3420sffemea&model_id=precision-t3x20-series-workstation

[31] Tarjeta de transporte T-Jove, consultado el 8 de Octubre de 2017. Fuente: https://www.tmb.cat/es/barcelona/tarifas-metro-bus/abonos/t-jove

[32] Consumo responsable.. ¿es posible con la electrónica? , consultado el 9 de Octubre de 2017. Fuente: https://www.isf.es/blog/consumo-responsable-es-posible-con-la-electronica/

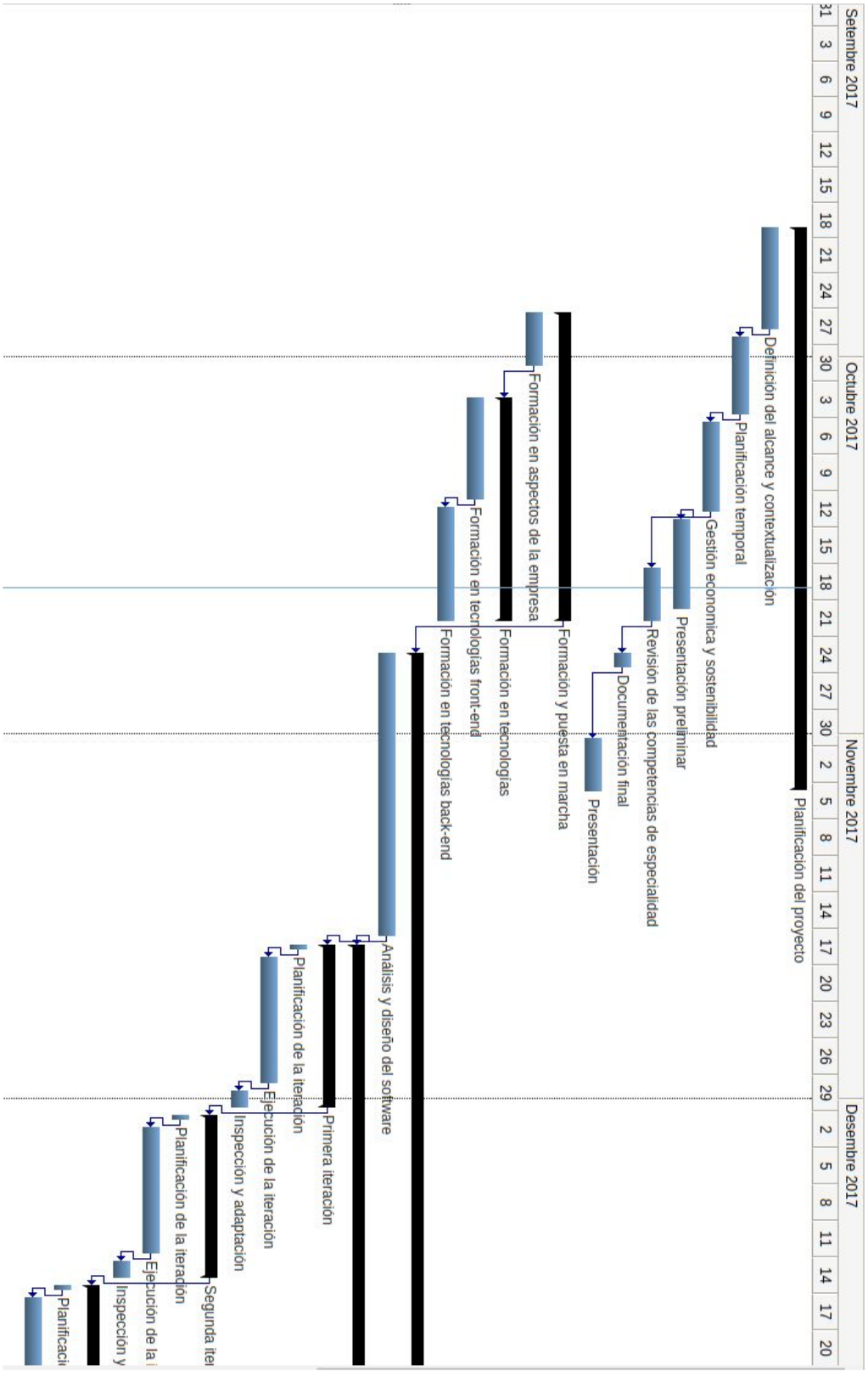
[33] Directiva Europea 1999/93/CE sobre firma electrónica, consultado el 2 de Abril de 2018. Fuente: https://www.boe.es/buscar/doc.php?id=DOUE-L-2000-80059

[34] Regulación eIDAS, consultado el 2 de Abril de 2018. Fuente: https://www.sede.fnmt.gob.es/documents/10445900/10526844/Reglamento_UE_910_2014.pdf

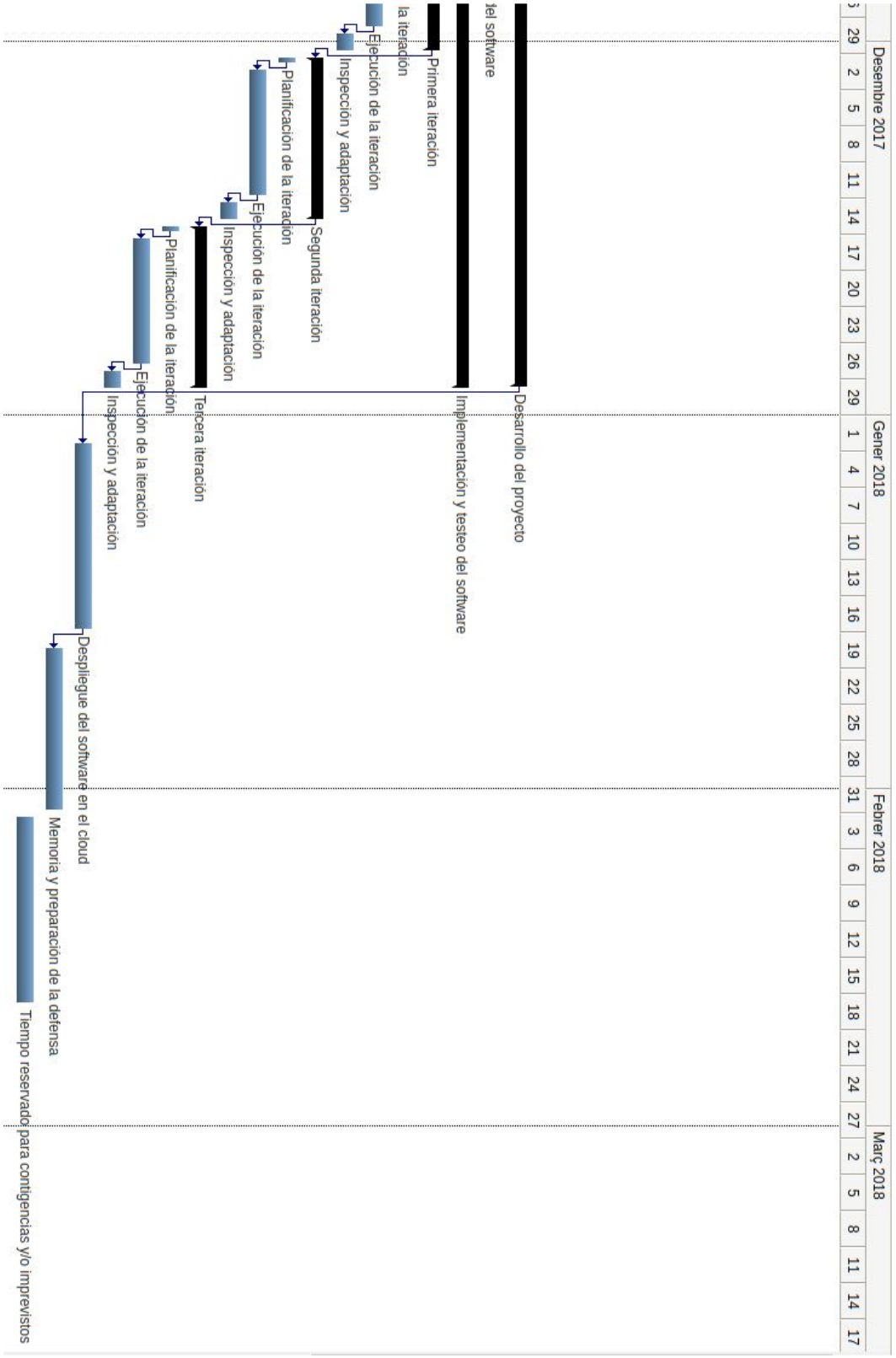
[35] Security Requirements for Trustworthy Systems Managing Certificates for Electronic Signatures, consultado el 2 de Abril de 2018. Fuente: ftp://ftp.cenorm.be/PUBLIC/CWAs/e-Europe/eSign/cwa14167-01-2003-Jun.pdf

[36] Evaluation Assurance Level, consultado el 2 de Abril de 2018. Fuente: https://en.wikipedia.org/wiki/Evaluation_Assurance_Level

Anexo A: Diagrama de Gantt Inicial



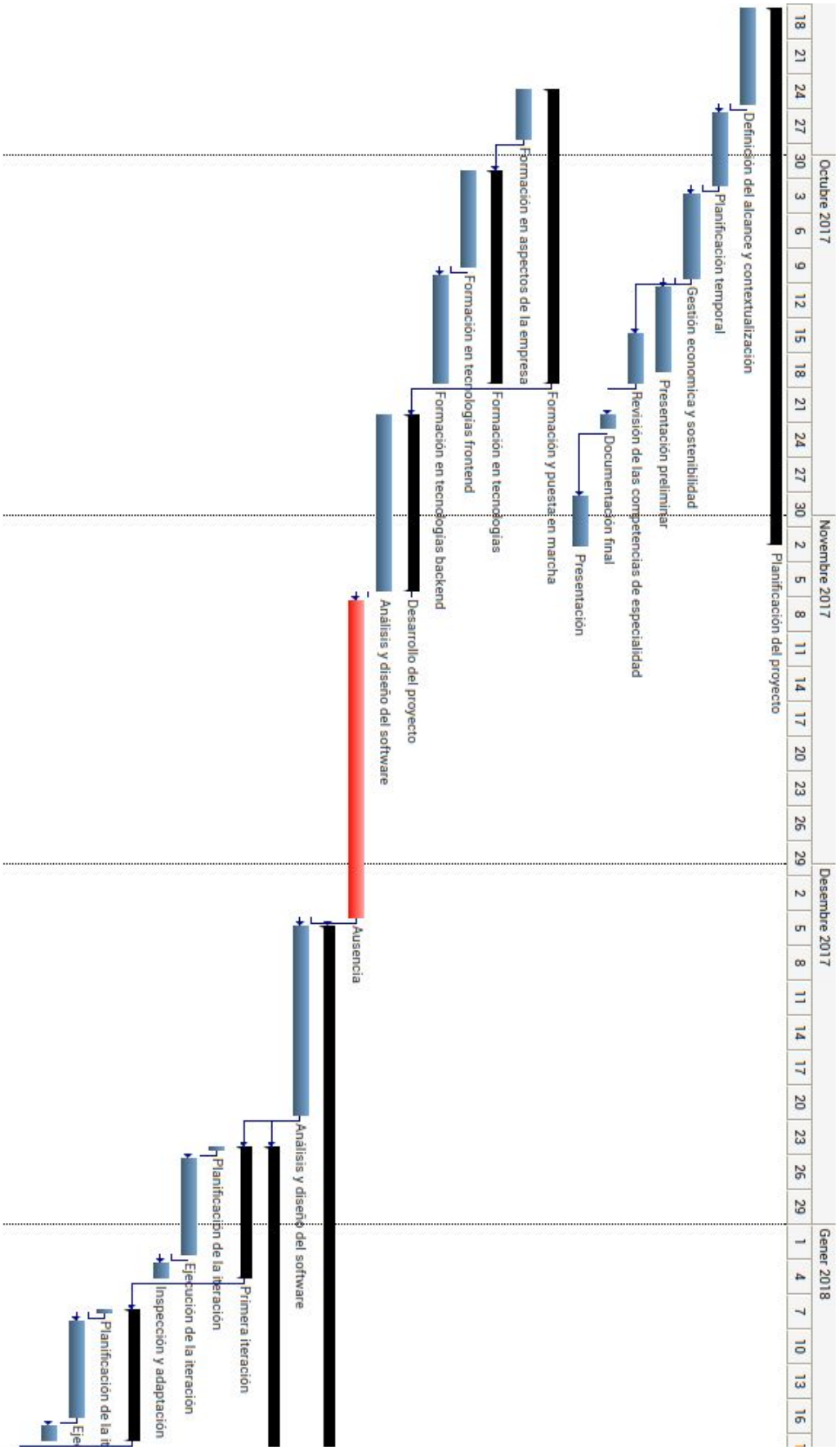
Anexo A: Diagrama de Gantt Inicial (II)



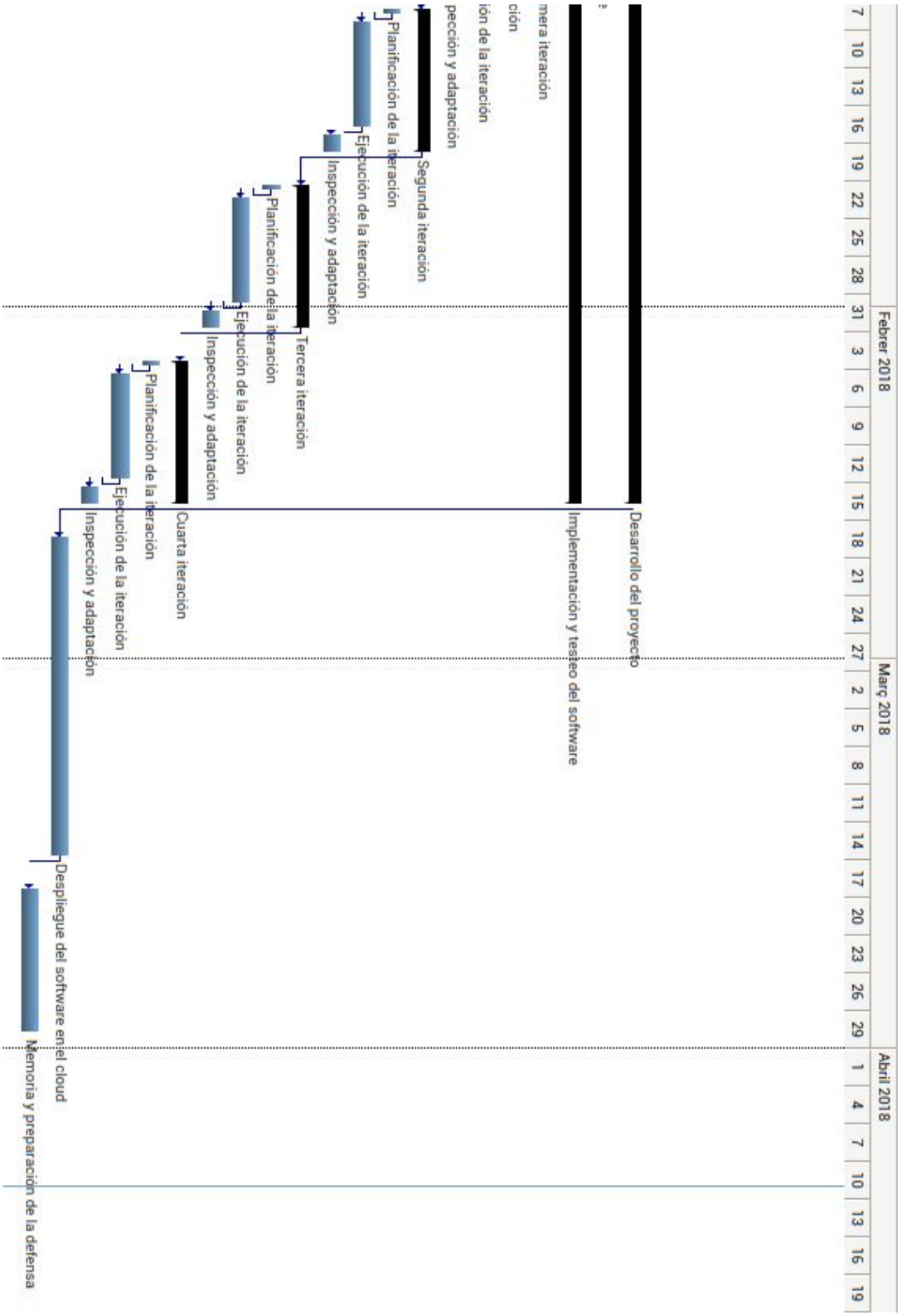
Anexo B: Tareas del diagrama de Gantt Inicial

	EDT		Nom	Durada	Inici	Fi	Predecessors
1	**1**		**Planificación del proyecto**	35d?	**18/09/2017**	**03/11/2017**	
2	1.1		Definición del alcance y contextualización	7d?	18/09/2017	26/09/2017	
3	1.2		Planificación temporal	5d?	27/09/2017	03/10/2017	2
4	1.3		Gestión economica y sostenibilidad	6d?	04/10/2017	11/10/2017	3
5	1.4		Presentación preliminar	6d?	12/10/2017	19/10/2017	4
6	1.5		Revisión de las competencias de especialidad	5d?	16/10/2017	20/10/2017	4
7	1.6		Documentación final	1.5d?	23/10/2017	24/10/2017	6
8	1.7		Presentación	5d?	30/10/2017	03/11/2017	7
9	**2**		**Formación y puesta en marcha**	20d?	**25/09/2017**	**20/10/2017**	
10	2.1		Formación en aspectos de la empresa	5d?	25/09/2017	29/09/2017	
11	**2.2**		**Formación en tecnologías**	15d?	**02/10/2017**	**20/10/2017**	**10**
12	2.2.1		Formación en tecnologías front-end	7d?	02/10/2017	10/10/2017	
13	2.2.2		Formación en tecnologías back-end	8d?	11/10/2017	20/10/2017	12
14	**3**		**Desarrollo del proyecto**	48d?	**23/10/2017**	**27/12/2017**	**9**
15	3.1		Análisis y diseño del software	18d?	23/10/2017	15/11/2017	
16	**3.2**		**Implementación y testeo del software**	30d?	**16/11/2017**	**27/12/2017**	**15**
17	**3.2.1**		**Primera iteración**	10d?	**16/11/2017**	**29/11/2017**	**15**
18	3.2.1.1		Planificación de la iteración	1d?	16/11/2017	16/11/2017	
19	3.2.1.2		Ejecución de la iteración	7d?	17/11/2017	27/11/2017	18
20	3.2.1.3		Inspección y adaptación	2d?	28/11/2017	29/11/2017	19
21	**3.2.2**		**Segunda iteración**	10d?	**30/11/2017**	**13/12/2017**	**17**
22	3.2.2.1		Planificación de la iteración	1d?	30/11/2017	30/11/2017	
23	3.2.2.2		Ejecución de la iteración	7d?	01/12/2017	11/12/2017	22
24	3.2.2.3		Inspección y adaptación	2d?	12/12/2017	13/12/2017	23
25	**3.2.3**		**Tercera iteración**	10d?	**14/12/2017**	**27/12/2017**	**21**
26	3.2.3.1		Planificación de la iteración	1d?	14/12/2017	14/12/2017	
27	3.2.3.2		Ejecución de la iteración	7d?	15/12/2017	25/12/2017	26
28	3.2.3.3		Inspección y adaptación	2d?	26/12/2017	27/12/2017	27
29	4		Despliegue del software en el cloud	12d?	01/01/2018	16/01/2018	14
30	5		Memoria y preparación de la defensa	10d?	18/01/2018	31/01/2018	29
31	6		Tiempo reservado para contigencias y/o imprevistos	12d?	01/02/2018	16/02/2018	

Anexo C: Diagrama de Gantt Final



Anexo C: Diagrama de Gantt Final (II)



Anexo D: Tareas del diagrama de Gantt Final

		Nom	Durada	Inici	Fi	Predecessors
1		**Planificación del proyecto**	35dies?	18/09/2017	03/11/2017	
2		Definición del alcance y contextualización	7dies?	18/09/2017	26/09/2017	
3		Planificación temporal	5dies?	27/09/2017	03/10/2017	2
4		Gestión economica y sostenibilidad	6dies?	04/10/2017	11/10/2017	3
5		Presentación preliminar	6dies?	12/10/2017	19/10/2017	4
6		Revisión de las competencias de especialidad	5dies?	16/10/2017	20/10/2017	4
7		Documentación final	1.5dies?	23/10/2017	24/10/2017	6
8		Presentación	5dies?	30/10/2017	03/11/2017	7
9		**Formación y puesta en marcha**	20dies?	25/09/2017	20/10/2017	
10		Formación en aspectos de la empresa	5dies?	25/09/2017	29/09/2017	
11		**Formación en tecnologías**	15dies?	02/10/2017	20/10/2017	10
12		Formación en tecnologías frontend	7dies?	02/10/2017	10/10/2017	
13		Formación en tecnologías backend	8dies?	11/10/2017	20/10/2017	12
14		**Desarrollo del proyecto**	12dies?	23/10/2017	07/11/2017	9
15		Análisis y diseño del software	12dies?	23/10/2017	07/11/2017	
16		Ausencia	20dies?	08/11/2017	05/12/2017	14
17		**Desarrollo del proyecto**	53dies?	06/12/2017	16/02/2018	16
18		Análisis y diseño del software	13dies?	06/12/2017	22/12/2017	16
19		**Implementación y testeo del software**	40dies?	25/12/2017	16/02/2018	18
20		**Primera iteración**	10dies?	25/12/2017	05/01/2018	18
21		Planificación de la iteración	1dia?	25/12/2017	25/12/2017	
22		Ejecución de la iteración	7dies?	26/12/2017	03/01/2018	21
23		Inspección y adaptación	2dies?	04/01/2018	05/01/2018	22
24		**Segunda iteración**	10dies?	08/01/2018	19/01/2018	20
25		Planificación de la iteración	1dia?	08/01/2018	08/01/2018	
26		Ejecución de la iteración	7dies?	09/01/2018	17/01/2018	25
27		Inspección y adaptación	2dies?	18/01/2018	19/01/2018	26
28		**Tercera iteración**	10dies?	22/01/2018	02/02/2018	24
29		Planificación de la iteración	1dia?	22/01/2018	22/01/2018	
30		Ejecución de la iteración	7dies?	23/01/2018	31/01/2018	29
31		Inspección y adaptación	2dies?	01/02/2018	02/02/2018	30
32		**Cuarta iteración**	10dies?	05/02/2018	16/02/2018	28
33		Planificación de la iteración	1dia?	05/02/2018	05/02/2018	
34		Ejecución de la iteración	7dies?	06/02/2018	14/02/2018	33
35		Inspección y adaptación	2dies?	15/02/2018	16/02/2018	34
36		Despliegue del software en el cloud	20dies?	19/02/2018	16/03/2018	17
37		Memoria y preparación de la defensa	10dies?	19/03/2018	30/03/2018	36