Escuela Politécnica Superior

Socialize. Sistema de sincronización dinámica de interfaces de usuarios acoplables a servicios de minería de datos.

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Michel Padrón Morales

Tutor/es:
Yoan Gutiérrez Vázquez

Septiembre 2016

Universitat d'Alacant
Universidad de Alicante

Agradecimientos

El desarrollo de este Trabajo de Fin de Grado tiene un significado especial para mí, ya que ha sido la culminación de muchos años de dedicación a la consecución de una de mis principales metas.

La experiencia adquirida acerca del mundo de la Ingeniería Informática se incorpora como parte de mi crecimiento profesional, tiene un valor incalculable y es la motivación para querer cada día más aprender y compartir los conocimientos acerca de esta profesión.

Quisiera agradecer a varias personas la ayuda prestada para alcanzar mis objetivos en este Trabajo de Fin de Grado.

Gracias a mi tutor, Yoan Gutiérrez, por confiar en mí y darme su apoyo incondicional.

Gracias a Ulises, Jorge, Paco y a ese futbolín desestresante... A todos los compañeros del departamento del GPLSI por su valiosa ayuda. Son todas maravillosas personas.

Gracias a mis compañeros de carrera, los de Alicante y los de Cuba. Siempre han sido un gran apoyo.

Gracias a todas las personas que de una forma u otra han sido de ayuda para llegar hasta aquí.

Gracias a mis dos universidades, el Instituto Superior Politécnico José A. E. de Cuba y la Universidad de Alicante en España, que me han visto crecer. Gracias por prepararme para la vida profesional.

Dedicatoria

A mi madre, guía de mi vida. Este es mi regalo para ti, el que siempre has deseado.

A mi nena. Siempre apoyándome… hasta en mis momentos de flaqueza. Siempre has confiado en mí y me has dado la fuerza necesaria para continuar.

A mi familia, en especial a mi abuelo José.

A mi Cuba… siempre estarás en mi corazón.

Índice

1. Introducción

Hoy en día se maneja cada vez más información en formatos no estructurados o semi-estructurados, como comentarios en redes sociales, mensajes de correo electrónico, notas de los centros de servicio al cliente, respuestas de encuestas con final abierto, fuentes de noticias, formularios web, entre otros. Se presenta como un problema a la hora de preguntarse cómo recopilar, explorar y aprovechar esta abundancia de información.

El proceso de analizar colecciones de materiales de texto con el objeto de capturar los temas y conceptos clave, descubrir las relaciones ocultas y las tendencias existentes sin necesidad de conocer las palabras o los términos exactos que los autores han utilizado para expresar dichos conceptos se conoce como **minería de textos**. La minería de textos y la acción de recuperar información son conceptos que a veces se confunden, aunque son bastante diferentes. Una recuperación precisa de la información y su almacenamiento supone un reto importante, pero la extracción y administración de contenido de calidad, de terminología y de las relaciones contenidas en la información son procesos cruciales y determinantes.

El análisis de texto, un tipo de análisis cualitativo, es la extracción de información útil del texto de manera que las ideas o los conceptos clave que contiene el texto pueden agruparse en una serie de categorías apropiadas.

La capacidad de extraer conceptos clave y crear categorías intuitivas para los orígenes de texto en un breve período de tiempo se obtiene mediante técnicas automáticas lingüísticas que aplican los principios de Procesamiento de Lenguaje Natural (PLN) [38].

1.1. Procesamiento de Lenguaje Natural

El PLN es una parte esencial de la Inteligencia Artificial que investiga y formula mecanismos computacionalmente efectivos que faciliten la interrelación hombre-máquina y permiten una comunicación mucho más fluida y menos rígida que los lenguajes formales utilizados tradicionalmente.

El PLN no trata de la comunicación por medio de lenguajes naturales de una forma abstracta, sino de diseñar mecanismos para comunicarse que sean eficaces

computacionalmente. Los modelos aplicados se enfocan no solo a la comprensión del lenguaje de por sí, sino a aspectos generales cognitivos humanos y a la organización de la memoria.

Todo sistema de PLN intenta simular un comportamiento lingüístico humano; para ello debe tomar conciencia tanto de las estructuras propias del lenguaje, como de un conocimiento general acerca del universo de discurso. [41]

1.2. Problemática

El Grupo de Procesamiento de Lenguaje Natural y Sistemas de Información (GPLSI) cuenta en estos momentos con una plataforma de procesamiento de contenido social (twitter, facebook), la cual ofrece diversos servicios de analítica y minería de textos.

Estos procesos permiten analizar dichas fuentes y obtener en tiempo real un conjunto gigantesco de datos agregados como las entidades que intervienen (nombre de personas, lugares, empresas, entre otros), la polaridad sentimental e intensidad asociada a dichas entidades, la geolocalización o referencias a lugares no descritos implícitamente en el texto, los términos más representativos implicados en el texto, fechas mencionadas en el texto independientemente del modo en que se ha hecho (ej. ayer, 10 de julio, semana pasada), dominios (ej. Deporte, Medicina, entre otros.) relevantes que enmarcan el contexto del contenido analizado, así como la obtención de resúmenes que sintetizan las fuentes de diferentes selecciones de contenido.

El problema científico de este trabajo está basado en que dichos procesos se han concebido como Servicios Webs[1], pero carecen de interfaces visuales que se adapten a tal número de información, las cuales podrían ser representadas de muy diversas formas (gráficos de barras, pastel, radial, entre otros) y ser sindicadas[2] en Webs de terceros.

[1] Tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

[2] Reenvío o reemisión de contenidos desde una fuente original (sitio web de origen) hasta otro sitio web de destino (receptor) que a su vez se convierte en emisor de dichos contenidos.

1.3. Objetivo General y Objetivos Específicos

Una vez identificada el problema científico, se plantea el **objetivo general** de este trabajo. Dicho objetivo consiste en el desarrollo de un sistema informático, que permita la generación de interfaces visuales adaptables a la información generada por los Servicios Webs de PLN ofrecidos por el GPLSI, las cuales puedan ser representadas de muy diversas formas y sindicadas en Webs de terceros.

Para llevar a cabo el objetivo general se plantean los siguientes **objetivos específicos**:

- Realizar un análisis minucioso del Estado del Arte.
- Estudiar y seleccionar metodologías existentes para el correcto desarrollo del sistema.
- Estudiar y seleccionar herramientas y tecnologías de desarrollo modernas, de gran alcance, que sean útiles para el rápido desarrollo y la obtención de calidad del software.
- Estudiar, seleccionar e implementar arquitecturas de software que definan la estructura para la construcción del proyecto.
- Incorporar visualizaciones que deberán utilizar los datos que se obtienen de los servicios de procesamiento de lenguaje natural desarrollados por el GPLSI.
- Desarrollar el sistema propuesto en el objetivo general.
- Estudiar y establecer un modelo de negocio adecuado al sistema a desarrollar.
- Evaluar el sistema obtenido y mostrar los resultados de la evaluación.
- Elaborar las conclusiones sobre el desarrollo del trabajo realizado.

2. Estado del Arte

El objetivo de este capítulo es proporcionar la información analizada en la investigación sobre las herramientas existentes hasta la fecha que podrían dar solución a los objetivos planteados. De cada herramienta investigada se hará un resumen del concepto y propósito y se realizará una valoración atendiendo a sus ventajas y desventajas.

Es una realidad la inexistencia de visualizaciones adaptables que muestren los datos agregados resultantes de los servicios de procesamiento de lenguaje natural del GPLSI. Por tanto se ha decidido investigar las tecnologías existentes hoy en día que pudieran dar solución a esta necesidad.

Entre muchas otras este trabajo se ha centrado en las siguientes tecnologías y/o herramientas:

- Kibana[3]
- Grafana[4]
- Social Analytics[5]
- Atribus[6]
- Sentiment Viz[7]
- Meaning Cloud[8]

A continuación se detallaran las características de cada una y se realizará una valoración atendiendo a sus ventajas y desventajas.

2.1. Kibana

Plugin[9] de código abierto de visualización de gráficos que permite interactuar con *ElasticSearch*[10]. Kibana trae los datos a la vida con efectos visuales como histogramas o geomapas, que se pueden combinar en cuadros de mandos personalizados. [17]

[3] https://www.elastic.co/products/kibana
[4] http://grafana.org/
[5] http://socialanalytics.gplsi.es/
[6] http://www.atribus.com/
[7] https://www.csc.ncsu.edu/faculty/healey/tweet_viz/
[8] https://www.meaningcloud.com/es
[9] Aplicación (o programa informático) que se relaciona con otra para agregarle una función nueva y generalmente muy específica.
[10] https://www.elastic.co/products/elasticsearch

2.1.1. ¿Qué es ElasticSearch?

Es un motor de búsqueda distribuida y análisis en tiempo real creado por Shay Banon en el 2010 [16]. Explora datos a una gran escala y velocidad. Se utiliza para la búsqueda de texto completo, búsqueda estructurada, análisis, y los tres en combinación. [17]

Ventajas de Kibana:

- Es gratuito
- De código Abierto
- Plataforma de visualización con flexibles analíticas.
- Resúmenes en tiempo real y gráficos de flujo de datos.
- Inclusión de cuadros de mandos con visualizaciones acopladas y posibilidad de compartir para sindicación en aplicaciones de terceros. [17]

Desventajas de Kibana:

- Al ser un plugin de Elasticsearch lo hace dependiente del mismo.
- Solo funciona conectado a un Elasticsearch que le proporcione datos.
- Carece de protocolos de privacidad y seguridad. Para solventar este problema los mismos creadores implementaron otro plugin llamado Shield, el cual es de pago [17].

En la investigación Kibana se valoró para su utilización dentro de la propuesta de este trabajo pues cumplía algunos objetivos planteados. Con la idea de añadirle características según las necesidades del GPLSI se realizó una investigación más profunda y se llegó a la conclusión de no utilizar la misma. Los motivos se describen en las desventajas de la herramienta. El motivo principal fue la privacidad y seguridad casi nulas.

2.2. Grafana

Grafana es una herramienta de código abierto que habitualmente trabaja con Graphite[11] , InfluxDB[12] y OpenTSDB[13]. Las versiones más recientes también pueden trabajar con otras fuentes de datos, tales como Elasticsearch [18].

[11] Herramienta para monitorización de métricas [22].
[12] Servidor de base de datos de series temporales [20].
[13] Base de Datos de Series Temporales distribuida y escalable [19].

Esencialmente, es un reemplazo rico en características de Graphite-web[14], que ayuda a los usuarios a crear y editar fácilmente cuadros de mando. Grafana es más comúnmente utilizado para la visualización de los datos de series temporales y la analítica de métricas de infraestructura en Internet y de las aplicaciones, pero muchos lo utilizan en otros dominios que incluyen sensores industriales, automatización del hogar, el clima y el control de procesos [21].

Ventajas de Grafana:

- Es gratuito
- Código Abierto
- Gestiona roles, con lo cual existe un protocolo de privacidad de la información.
- Plataforma de visualización con flexibles analíticas.
- Resúmenes en tiempo real y gráficos de flujo de datos.
- Puede trabajar con múltiples bases de datos de series temporales.

Desventajas de Grafana:

- Creado para la monitorización de métricas, con lo cual las visualizaciones que posee están pensadas en exponer estadísticas tipo métricas.
- La herramienta es complicada de entender para los usuarios finales que no están relacionados con la informática.

Con la investigación de Grafana se encuentran una serie de ventajas que no existen en Kibana como la independencia de una fuente de datos o la privacidad incorporada a través de la gestión de roles. En este caso las desventajas han sido contundentes, Grafana está pensado para usuarios que provengan del mundo de la informática y estén capacitados para las tareas de analítica e inteligencia de negocio. Para usuarios sin la experiencia comentada sería complejo de entender.

2.3. Social Analytics

Social Analytics es una aplicación que recupera tweets de la red social de Twitter sobre un tema en concreto y, de forma automática, valora las opiniones expresadas en los

[14] Herramienta de visualización de datos parte de Graphite.

mensajes. Esto permite realizar un seguimiento de las opiniones de las personas sobre diferentes temas. De esta forma, y gracias a esta herramienta se puede llegar a un análisis y predicción de opiniones y tendencias. Este análisis puede representarse tanto geográficamente en un mapa como en gráficos [23].

Incorpora minería de opiniones[15] que permite rastrear a través de Internet y de las redes sociales los comentarios que hacen las personas sobre cualquier tema y valorar qué emoción expresa, si es positiva o negativa e, incluso, la intensidad. Se centra en monitorizar los temas definidos en el sistema y, para cada tweet relacionado con dichos temas, valora la opinión de las personas.

Social Analytics incorpora un *Back-End*[16] donde utiliza un descargador de tweets y los almacena en una base de datos ElasticSearch [17]. A su vez incorpora un *Api Rest*[17] para exponer y consumir a través de la web los datos indexados del ElasticSearch.

La otra parte de la herramienta es su *Front-End*, el cual trata de cubrir objetivos parecidos a los planteados en este proyecto. En estos momentos gestiona usuarios y colecciones de entidades por usuario con el objetivo de mostrar los datos recogidos sobre estas entidades en un cuadro de mando estático, ofreciendo una serie de visualizaciones predefinidas. La herramienta permite filtrar, cambiar a modo histórico o a tiempo real [26].

Ventajas del Front-End de Social Analytics:

- Creado y mantenido por el GPLSI.
- Herramienta intuitiva y fácil de utilizar.
- Ofrece visualizaciones personalizadas con el objetivo de que el usuario obtenga información útil de las mismas.

Desventajas del Front-End de Social Analytics:

- No se pueden crear, editar o eliminar las visualizaciones; estas ya vienen predefinidas.
- El cuadro de mando o dashboard donde se insertan las visualizaciones no es dinámico, por tanto no se puede cambiar de tamaño y posición los elementos contenidos en el mismo.
- Al ser parte del proyecto Social Analytics está creado con una dependencia total de su Back-End.

[15] La Minería de Opiniones es también llamada Análisis de Sentimientos [24].
[16] Front-end y back-end son términos que se refieren a la separación de intereses entre una capa de presentación y una capa de acceso a datos, respectivamente
[17] El Servicio Web tipo REST [25].

- No es posible sindicar visualizaciones webs de análisis y minería de textos.

Mientras que se puede apreciar que esta herramienta tiene un buen acabado y está en constante evolución hacia las necesidades del GPLSI [26], las carencias de Social Analytics han sido el motivo principal por el cual se desarrolla este trabajo.

2.4. Atribus

Atribus busca lo que se dice acerca de un contexto determinado en redes sociales, foros, blogs y webs. Una vez introducidas tus búsquedas en Atribus la herramienta rastrea y clasifica automáticamente todas las menciones en diferentes archivos para poderlos descargar y archivar, además de gráficas de fácil interpretación.

Atribus monitoriza y analiza numerosas variables según tus intereses o necesidades: qué personas han participado más, tweets más retwitteados, personas por retweets, enlaces más compartidos, palabras más empleadas o personas más influyentes que hablan sobre tu marca, entre otras muchas variables.

Ventajas de Atribus:

- Análisis de contenido en diversas fuentes de datos (Redes Sociales, Blogs, Foros, entre otros)
- Permite la incorporación de visualizaciones adaptables, cambiando tamaño y posiciones en la plantilla base.
- Permite la descarga en formato de imagen o PDF de los gráficos creados.

Desventajas de Atribus:

- Algo sobrecargado. El excesivo volumen de información en pantalla exige un esfuerzo añadido a la hora de sacar provecho a la aplicación.

Atribus es una herramienta bastante completa donde puedes hacerte una visión global, útil para la toma de decisiones respecto a tu reputación online, estrategias de marketing o comunicación.

Se destaca por ser líder en España para la gestión y monitorización de la información en internet y cuenta con una poderosa cartera de clientes. Si existe un rival a batir, sin dudas, se llama Atribus.

2.5. Sentiment Viz

Sentiment Viz es un proyecto de investigación de la Universidad Estatal de Carolina del Norte, Estados Unidos. Este proyecto estudia formas de calcular y visualizar el sentimiento en fragmentos de textos cortos. Actualmente está enfocado en obtener el sentimiento de tweets publicados en Twitter. Posee una aplicación web donde muestra su enorme potencial[18].

Ventajas de Sentiment Viz:

- Además de definir la polaridad sentimental del texto es capaz detectar las emociones generadas en el mismo.
- La alianza con personas del mundo de la psicología ha permitido que se apliquen técnicas innovadoras para el análisis y representación de emociones.

Desventajas de Sentiment Viz:

- La principal desventaja de Sentiment Viz es que no posee la opción de crear visualizaciones adaptables ni cuadros de mandos que permitan al usuario utilizar los beneficios de la herramienta para sus necesidades.

Sentiment Viz es un proyecto meramente investigativo. Las gráficas que implementa destacan su potencialidad en el análisis con algoritmos de PLN que identifican las emociones.

2.6. Meaning Cloud

Meaning Cloud, propiedad de s|ngular[19], posee un compendio de herramientas y *APIs* de análisis semántico que se proporcionan en modo *SaaS*[20] y *on-premises*[21]. Aunque Meaning

[18] https://www.csc.ncsu.edu/faculty/healey/tweet_viz/tweet_app/
[19] http://sngular.team/es/-/servicios/software
[20] Modelo de distribución de software donde el soporte lógico y los datos que maneja se alojan en servidores de una compañía de tecnologías de información, a los que se accede vía Internet desde un cliente.
[21] Se instala y se ejecuta en los ordenadores de los locales

Cloud no incorpora herramientas de monitorización del lado del cliente se ha realizado un estudio de sus capacidades de analítica de textos. Algunas de sus *APIs* son las siguientes:

- Extracción de Tópicos e Identificación de Idioma
- Clasificación y Clustering de Texto
- Análisis de Sentimiento
- Lematización, Análisis Morfológico y Sintáctico
- Reputación Corporativa

Ventajas de Meaning Cloud:

- Amplia gama de productos y soluciones centrados en la analítica de textos.
- Posee una licencia gratuita bastante interesante.
- Cuenta con un conjunto de *APIs* para desarrolladores.

Desventajas de Meaning Cloud:

- La principal desventaja es que no cuenta con interfaces gráficas que puedan mostrar sus capacidades.

2.7. Conclusiones Parciales

A lo largo de este capítulo se ha realizado un análisis minucioso del Estado del Arte, lo cual ha servido para conocer las tecnologías punteras relacionadas con este trabajo. De cada tecnología y/o herramienta estudiada se han especificado los puntos positivos y negativos. Por tanto, se concluye que ninguna de las tecnologías y/o herramientas analizadas resuelve nuestro problema científico, más bien cada una por separado aporta una pequeña parte de la solución y servirán de punto de partida para la consecución de los objetivos planteados.

3. Metodologías

En este capítulo se estudian y seleccionan las metodologías para la correcta planificación y desarrollo del sistema. Esto es necesario dada la imperiosa necesidad de que el sistema llegue al éxito y obtener un producto de gran valor para nuestros clientes. Es por eso la importancia de utilizar una metodología robusta que ajustada en un equipo cumpla con sus metas, y satisfaga más allá de las necesidades definidas al inicio del proyecto.

El éxito del proyecto depende en gran parte de la metodología escogida por el equipo, ya sea tradicional o ágil, donde los equipos maximicen su potencial, aumenten la calidad del producto con los recursos y tiempos establecidos [3].

3.1. Metodologías Tradicionales

Teniendo en cuenta la filosofía de desarrollo de las metodologías, aquellas con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales o Pesadas.

Las Metodologías Tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada. [4]

Este tipo de metodologías son más eficaces y necesarias cuanto mayor es el proyecto que se pretende realizar respecto a tiempo y recursos que son necesarios emplear, donde una gran organización es requerida. Además, las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno, donde los requisitos no pueden predecirse o bien pueden variar. [4]

Entre las metodologías tradicionales se pueden citar:

- Proceso Unificado de Racional (RUP)
- Microsoft Solution Framework (MSF)
- Win-Win Spiral Model
- Microsoft Operation Framework.

3.1.1. Proceso Unificado de Racional (RUP)

El Proceso Unificado de Desarrollo [20] fue creado por el mismo grupo de expertos que crearon UML, Ivar Jacobson, Grady Booch y James Rumbaugh en el año 1998. El objetivo que se perseguía con esta metodología era producir software de alta calidad, es decir, que cumpla con los requerimientos de los usuarios dentro de una planificación y presupuesto establecidos. Como se expresaba anteriormente, esta metodología concibió desde sus inicios el uso de UML como lenguaje de modelado.

Es un proceso dirigido por casos de uso, este avanza a través de una serie de flujos de trabajo (requisitos, análisis, diseño, implementación, prueba) que parten de los casos de uso; está centrado en la arquitectura y es iterativo e incremental. Además cubre el ciclo de vida de desarrollo de un proyecto y toma en cuenta las mejores prácticas a utilizar en el modelo de desarrollo de software. [5]

- Desarrollo de software en forma iterativa.
- Manejo de requerimientos.
- Utiliza arquitectura basada en componentes.
- Modela el software visualmente.
- Verifica la calidad del software.
- Controla los cambios. [5]

Para apoyar el trabajo con esta metodología ha sido desarrollada la herramienta CASE (Computer Assisted Software Engineering) Rational Rose, creada por la Compañía norteamericana Rational Corporation en el año 2000. Esta herramienta integra todos los elementos que propone la metodología para cubrir el ciclo de vida de un proyecto. [4]

3.1.2. Microsoft Solution Framework (MSF)

MSF es una metodología desarrollada por Microsoft Consulting Services en conjunto con varios grupos de negocios de Microsoft y otras fuentes de la industria. MSF provee los principios, modelos y disciplinas para un correcto desarrollo de proyectos en cualquier plataforma.

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

MSF se centra en:

- Alinear los objetivos de negocio y de tecnología
- Establecer de manera clara los objetivos, los roles y las responsabilidades
- Implementar un proceso iterativo controlado por hitos o puntos de control
- Gestionar los riesgos de manera proactiva
- Responder con eficacia ante los cambios [6]

MSF tiene las siguientes características:

- **Adaptable**: Es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- **Escalable**: Puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.
- **Flexible**: Es utilizada en el ambiente de desarrollo de cualquier cliente.
- **Tecnología Agnóstica**: Puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Gobernanza (ver figura 1), Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación.

Figura 1. Seguimiento en el Modelo de Gobernanza

3.1.3. Win-Win Spiral Model

El modelo en espiral (ver figura 2) fue propuesto por primera vez por Boehm en 1986 [7]. Es un modelo de proceso de software evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial.

A diferencia del modelo de proceso clásico que termina cuando se entrega el software, el modelo en espiral puede adaptarse y aplicarse a lo largo de la vida del software de computadora. Una visión alternativa del modelo en espiral puede ser considerada examinando el eje de punto de entrada en el proyecto. Las regiones de tareas que componen este modelo son:

- **Comunicación con el cliente**: las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.
- **Planificación**: las tareas requeridas para definir recursos, el tiempo y otras informaciones relacionadas con el proyecto. Son todos los requerimientos.
- **Análisis de riesgos**: las tareas requeridas para evaluar riesgos técnicos y otras informaciones relacionadas con el proyecto.

- **Ingeniería**: las tareas requeridas para construir una o más representaciones de la aplicación.
- **Construcción y adaptación**: las tareas requeridas para construir, probar, instalar y proporcionar soporte al usuario.
- **Evaluación del cliente**: las tareas requeridas para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementación durante la etapa de instalación.

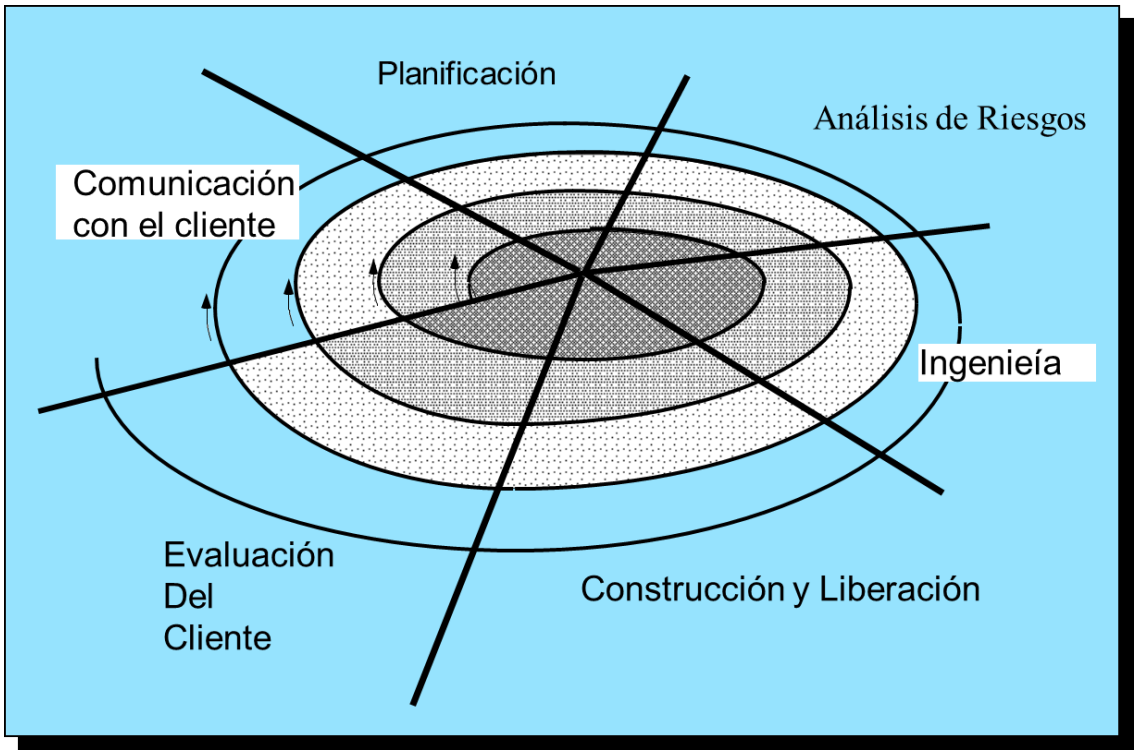

Figura 2. Modelo en espiral

3.1.4. Microsoft Operation Framework.

El modelo de proceso MOF (ver figura 1) está formado por cuadrantes, revisiones de la administración de las operaciones y revisiones de la administración de los servicios.

Figura 3. Ciclo de Microsoft Operations Framework

MOF se desplaza en sentido de las agujas del reloj y se divide en los cuatro cuadrantes integrados siguientes:

- Cambios
- Operaciones
- Soporte técnico
- Optimización

Estos cuadrantes forman un ciclo de vida en espiral que se aplica a las operaciones de TI, desde una aplicación específica hasta un entorno de operaciones completo con varios centros de datos. El modelo de proceso está respaldado por funciones de administración de servicios (SMF), y un modelo de equipo y un modelo de riesgos integrados. Cada cuadrante se complementa con una revisión de la administración de operaciones (también conocida como hito de revisión), durante la cual se evalúa la eficacia de las SMF de ese cuadrante. Se debe tener en cuenta que, aunque el modelo describe los cuadrantes de MOF de forma secuencial, las actividades de todos los cuadrantes pueden ocurrir al mismo tiempo. [10]

3.2. Metodologías Ágiles

Las metodologías ágiles son métodos de desarrollo de software en los que las necesidades y soluciones evolucionan a través de una colaboración estrecha entre equipos

multidisciplinarios. Se caracterizan por enfatizar la comunicación frente a la documentación, por el desarrollo evolutivo y por su flexibilidad.

En febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace el término "ágil" aplicado al desarrollo de software. En esta reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software [9].

Tras esta reunión se creó *The Agile Alliance*[22], una organización, dedicada a promover los conceptos relacionados con el desarrollo ágil de software. El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía "ágil" [9].

Según el Manifiesto Ágil se valoran:

- **Individuos e interacciones** sobre procesos y herramientas.
- **Software funcionando** sobre una amplia documentación.
- **Colaboración con el cliente** sobre negociación de contratos.
- **Respondiendo al cambio** sobre seguir un plan.

Mientras que hay valor en los elementos de la derecha, se valoran más los elementos de la izquierda [1].

Los valores anteriores inspiran los doce principios del manifiesto. Son características que diferencian un proceso ágil de uno tradicional. Los dos primeros principios son generales y resumen gran parte del espíritu ágil. El resto tienen que ver con el proceso a seguir y con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo.

Los 12 principios del Manifiesto Ágil:

1. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.
2. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

[22] https://www.agilealliance.org/

3. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
4. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
5. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
7. El software que funciona es la medida principal de progreso.
8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
9. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
10. La simplicidad es esencial.
11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
12. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

Siguiendo estos principios se explican algunas de las metodologías ágiles más populares entre la comunidad de desarrolladores:

- eXtreme Programming (XP)
- Scrum
- Kanban
- Iconix

3.2.1. Extreme Programming (XP)

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones

implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico [9].

A continuación se presentan las características esenciales de XP organizadas en los tres apartados siguientes: historias de usuario, roles, proceso y prácticas.

3.2.1.1. Las Historias de Usuario

Son la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Beck en su libro [11] presenta un ejemplo de ficha (customer story and task card) en la cual pueden reconocerse los siguientes contenidos: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas y una lista de seguimiento con la fecha, estado cosas por terminar y comentarios.

A efectos de planificación, las historias pueden ser de una a tres semanas de tiempo de programación (para no superar el tamaño de una iteración). Las historias de usuario son descompuestas en tareas de programación (task card) y asignadas a los programadores para ser implementadas durante una iteración.

3.2.1.2. Proceso XP

A grandes rasgos el ciclo de desarrollo consiste en los siguientes pasos [9]:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, pues se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

3.2.1.3. Práctica XP

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione.

Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas, las cuales se refuerzan entre sí (ver figura 4).

- **El juego de la planificación**. Hay una comunicación frecuente entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración. Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema.
- **Metáfora**. El sistema es definido mediante una metáfora o un conjunto de metáforas. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema.
- **Diseño simple**. Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
- **Pruebas**. La producción de código está dirigida por las pruebas unitarias. Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.
- **Refactorización**. Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.

- **Programación en parejas**. Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores).
- **Propiedad colectiva del código**. Cualquier programador puede cambiar cualquier parte del código en cualquier momento.
- **Integración continua**. Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
- **40 horas por semana**. Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.
- **Cliente in-situ**. El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.
- **Estándares de programación**. XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

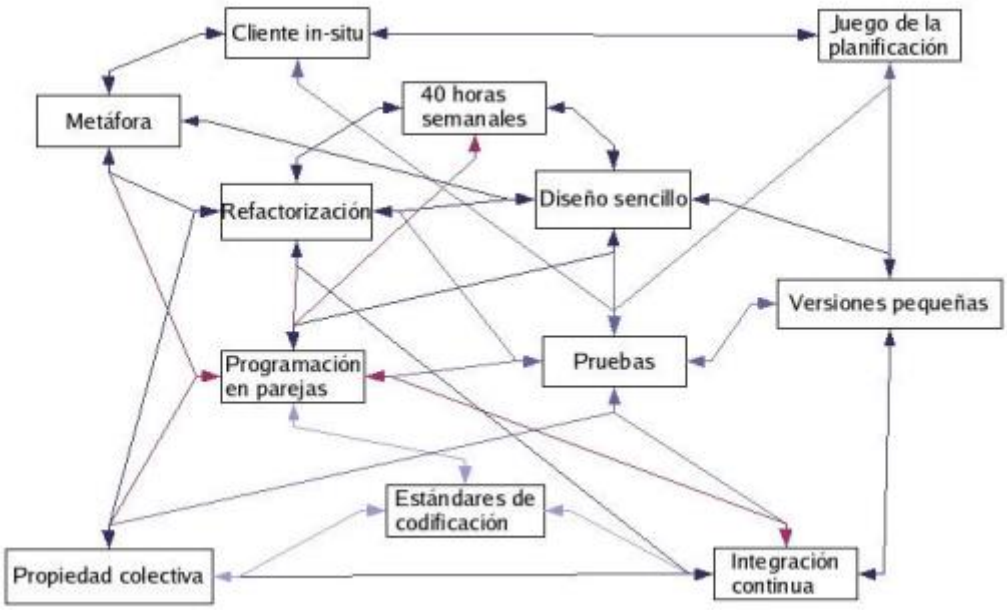

Figura 4. Las prácticas se refuerzan entre sí.

3.2.2. Scrum

En Scrum el desarrollo de software se realiza mediante iteraciones, denominadas Sprints[23], con una duración fija. El resultado de cada Sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración [9].

3.2.2.1. Cómo funciona el Proceso

En primer lugar se define el Product Backlog, lo que nos permitirá realizar nuestros Sprints más adelante [12].

- **Product Backlog**: Es una lista de deseos sobre las funcionalidades del producto. Es elaborado por el Product Owner (Dueño del Producto) y las funciones están priorizadas según lo que es más y menos importante para el negocio. El objetivo es que el Product Owner responda la pregunta "¿Qué hay que hacer?".
- **Sprint Backlog**: Es un subconjunto de tareas del Product Backlog, que son seleccionados por el equipo para realizar durante el Sprint sobre el que se va a trabajar. El equipo establece la duración de cada Sprint.
- **Sprint Planning Meeting**: Esta reunión se hace al comienzo de cada Sprint y se define cómo se va a enfocar el proyecto que viene del Product Backlog las etapas y los plazos. Cada Sprint está compuesto por diferentes features. Por ejemplo, se decide que los features del primer Sprint son: diseño del logo, definición colores y contenido multimedia.
- **Daily Scrum o Stand-up Meeting**: Es una reunión breve que se realiza a diario mientras dura el periodo de Sprint. Se responden individualmente tres preguntas: ¿Qué hice ayer?, ¿Qué voy a hacer hoy?, ¿Qué ayuda se necesita? El Scrum Master debe tratar de solucionar los problemas u obstáculos que se presenten.
- **Sprint Review**: Se revisa el sprint terminado, y ya debería haber un avance claro y tangible para presentarlo al cliente.

[23] El Sprint o iteración es el período en el cual se lleva a cabo el trabajo en sí.

- **Sprint Retrospective**: El equipo revisa los objetivos cumplidos del Sprint terminado. Se anota lo bueno y lo malo, para no volver a repetir los errores. Esta etapa sirve para implementar mejoras desde el punto de vista del proceso del desarrollo.

3.2.2.2. Participantes

- **Product Owner**: Habla por el cliente, y asegura que el equipo cumpla las expectativas. Es "el jefe" responsable del proyecto.
- **Scrum Master**: Lidera las reuniones y ayuda al equipo si es que tienen problemas. Además, minimiza los obstáculos para cumplir el objetivo del Sprint, es un "facilitador" pero no es un gestor.
- **Scrum Team**: Son los encargados de desarrollar y cumplir lo que les asigna el Product Owner.
- **Cliente**: Recibe el producto y puede influir en el proceso, entregando sus ideas o comentarios respecto al desarrollo

3.2.3. Kanban

El método Kanban en el desarrollo de software maneja equipos de proyectos para visualizar el flujo de trabajo, limita el trabajo en progreso (*WIP*) en cada etapa del flujo de trabajo y mide el tiempo de ciclo (*lead time*) como se muestra en la tabla 1.

Característica	Descripción
Visualiza el flujo de trabajo	Divide el trabajo en bloques, escribe cada elemento en una tarjeta y se coloca en el tablero. Utiliza columnas con nombre para ilustrar dónde está cada elemento en el flujo de trabajo.
Limita el trabajo en curso (WIP)	Asigna límites concretos al número de elementos que pueden estar en progreso en cada estado del flujo de trabajo.
Mide el tiempo en ciclo (lead time) medio para completar un elemento	Optimiza el proceso para que el lead time sea tan pequeño y predecible como sea posible.

Tabla 1. Principales características de Kanban

El tablero Kanban (ver figura 5) proporciona visibilidad del proceso del software, en el muestra el trabajo asignado para cada desarrollador, comunica claramente las prioridades y resalta los cuellos de botella, así el equipo se concentra en resolver los problemas que bloquean el proceso y restauran el flujo productivo. En la figura 3 se muestra un ejemplo de un tablero básico Kanban.

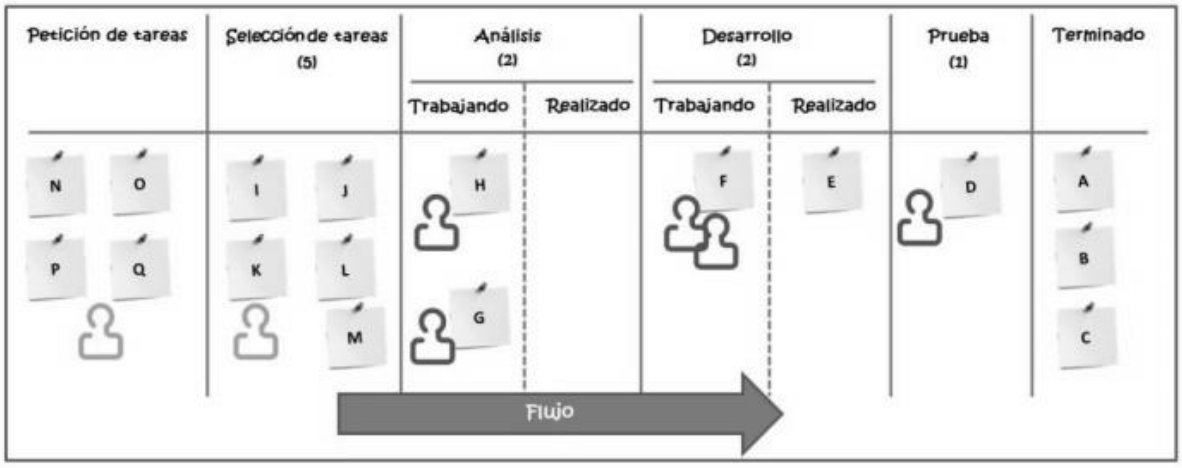

Figura 5. Tablero Kanban

Kanban limita el trabajo en curso de acuerdo a la capacidad del equipo, que equilibra la demanda contra el rendimiento del trabajo liberado por el equipo. Esto ayuda a visualizar los problemas del proceso, minimiza los defectos y mantiene un flujo estable. Al limitar el trabajo en curso se consigue un ritmo de desarrollo sostenible, elevando la calidad de los productos y un mayor rendimiento de los integrantes del equipo. El flujo estable y la calidad en el producto ayuda a reducir el tiempo de ciclo (lead time), generando la liberación de entregables de forma más regular que incrementa la confianza del cliente hacia la empresa desarrolladora de software [14].

3.2.4. Iconix

Es una metodología pesada-ligera de desarrollo del software que se halla entre RUP (Rational Unified Process) y XP (eXtreme Programming), unifica un conjunto de métodos de orientación a objetos con el objetivo de tener un control estricto sobre todo el ciclo de vida del producto a realizar.

Fue elaborado por Doug Rosenberg y Kendall Scott a partir de una síntesis del proceso unificado de los "tres amigos" Booch, Rumbaugh y Jacobson y que ha dado soporte y conocimiento a la metodología ICONIX desde 1993. Presenta claramente las actividades de cada fase y exhibe una secuencia de pasos que deben ser seguidos [8].

Las tres características fundamentales de ICONIX son:

- **Iterativo e incremental**: Varias interacciones ocurren entre el modelo del dominio y la identificación de los casos de uso. El modelo estático es incrementalmente refinado por los modelos dinámicos.
- **Trazabilidad**: Cada paso está referenciado por algún requisito. Se define la trazabilidad como la capacidad de seguir una relación entre los diferentes artefactos producidos
- **Dinámica del UML**: La metodología ofrece un uso dinámico del UML como los diagramas del caso de uso, diagramas de secuencia y de colaboración [4].

Las tareas que se realizan en la metodología ICONIX son:

- Análisis de requisitos
- Análisis y diseño preliminar
- Diseño
- Implementación

3.3. Comparación entre metodologías

En este apartado se realizará una comparación general entre las metodologías ágiles y tradicionales (ver tabla 2)

Puntos	**Metodologías Ágiles**	**Metodologías Tradicionales**
Fuente	Heurísticas provenientes de prácticas de producción de código.	Normas provenientes de estándares seguidos por el entorno de desarrollo.
Cambios	Especialmente preparadas para cambios durante el proyecto.	Cierta resistencia a los cambios.

Imposición	Impuestas internamente (por el equipo).	Impuestas externamente.
Restrictividad	Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
Contrato	No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
Cliente	El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Tamaño de grupos	Grupos pequeños (se recomienda <10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Artefactos	Pocos artefactos.	Más artefactos.
Roles	Pocos roles.	Más roles.
Arquitectura	Menos énfasis en la arquitectura del software. Se va definiendo en el desarrollo.	La arquitectura del software es esencial y se expresa mediante modelos desde el principio.

Tabla 2. Diferencias entre metodologías ágiles y no ágiles

3.4. Metodologías empleadas

El análisis anterior fue necesario para poder comprender y escoger las metodologías necesarias, aquellas que se ajustaran más al proyecto.

Los principales factores que intervinieron fueron:

- El tiempo disponible para la creación y puesta en marcha del proyecto era reducido.
- El proyecto estaba sujeto a cambios desde el principio debido a la volatilidad de los requerimientos.

- La necesidad de una fuerte comunicación entre el cliente (tutor) y el equipo de trabajo (en este caso, el tutelado).
- La necesidad de acoplarse al entorno de trabajo del departamento del GPLSI, desde donde se tutela el proyecto y se trabaja con una metodología bien definida.
- El tamaño del equipo muy pequeño.

Kniberg y Skarin en su libro [15] comentan:

¡Mezcla y combina las herramientas que necesites! Me cuesta imaginar un exitoso equipo Scrum que no incluye la mayoría de los elementos de XP, por ejemplo. Muchos equipos Kanban hacen reuniones diarias (una práctica de Scrum). Algunos equipos Scrum escriben algunos de sus elementos de la pila como Casos de Uso (una práctica de RUP) o limitan el tamaño de las colas (una práctica de Kanban). Usa lo que sea que funcione para ti.

Miyamoto Musashi (Samurai del siglo 17º, famoso por su técnica de lucha de la doble espada) dijo:

¡No te ciñas a una única arma o a una única escuela de lucha!

Dado el caso particular de Socialize, utilizar una metodología y seguirla al pie de la letra es casi imposible. Lo ideal es escoger prácticas y/o procesos de varias metodologías que se pudieran adaptar fácilmente, de manera natural y no fuesen luego una carga más, en vez de una ayuda.

Por tanto, para tareas organizativas,

- Se toma Scrum como metodología base.
- Se decide hacer un seguimiento mediante iteraciones cortas (Sprints) de dos semanas.
- Las reuniones diarias (Daily Scrum) carecen de sentido, por el contrario la realización de una reunión semanal es necesaria. De esta manera se hace un seguimiento de las iteraciones.
- El seguimiento de tareas de la iteración se hace mediante un tablero Kanban en el cual se emplean las columnas Backlog, Por hacer, En Progreso, Fin.

- Al final de cada iteración se realiza un Sprint Review donde se le presenta una Demo al cliente del producto y un Sprint Retrospective al mismo tiempo donde se comenta hasta donde se llegó y que faltó del Sprint.
- Por último se organiza una nueva iteración luego de que el cliente decide las prioridades del proyecto dentro del Product Backlog.

Para el análisis,

- Se utiliza el lenguaje de modelado UML para el modelado de casos de uso, diagramas de navegación y diagramas de clases.

Para el desarrollo,

- Se utiliza el Desarrollo Dirigido por Tests (Test Driven Development), técnica de diseño e implementación de software incluida dentro de la metodología XP.
- Se refactoriza el código para hacerlo cada vez más legible, aplicando refactorización de XP
- Se aplican procesos de Integración Continua, con lo cual, desde etapas muy tempranas con un único comando se puede desplegar el proyecto en su versión más reciente.

3.5. Conclusiones Parciales

En este capítulo se han estudiado las metodologías existentes con el objetivo de escoger la adecuada para guiar el desarrollo del sistema. En el estudio realizado se separan las metodologías ágiles y las tradicionales para compararlas entre sí. En la fase de selección se concluye que las metodologías ágiles son las más adecuadas dadas las características de este trabajo y se decide combinar diferentes prácticas y/o métodos de Scrum, XP, Kanban y UML para la consecución de los objetivos trazados.

4. Herramientas y Tecnologías de Desarrollo

En este capítulo se listan las herramientas y tecnologías de desarrollo utilizadas en el proyecto. Se realiza una breve descripción y se enfatiza en las ventajas de cada una. Además, se realiza una conclusión parcial acerca de los beneficios de usar las herramientas y tecnologías escogidas.

4.1. Herramientas y Tecnologías

El siguiente listado es la selección del compendio de herramientas y tecnologías usadas para el desarrollo de este trabajo.

Webstorm[24]: Entorno de desarrollo integrado de JavaScript enfocado en dicho lenguaje. Es escogido por su fácil integración con frameworks JavaScript, herramientas de gestión de paquetes, control de versiones y automatizadores de tareas. Incorpora un servidor web interno para el rápido despliegue en pruebas de los proyectos.

Google Chrome[25]: Navegador utilizado para el desarrollo del proyecto. Contiene buenas herramientas de testeo de código, debug y monitorización de la red.

Firebase[26]. Perteneciente a Google, Firebase es un proveedor de servicios en la nube. Ofrece un servicio de Back-end, el cual se utiliza en el proyecto para el control de la lógica [2]. Hasta mayo de 2016 Firebase contaba con tres principales servicios:

- *Firebase Realtime Database*. Base de datos alojada en la nube. Los datos se almacenan en formato *JSON*[27] y se sincronizan en tiempo real (ver figura 6) con cada cliente conectado.
- *Firebase Authentication* proporciona servicios de *Back-End*, *SDK*[28] fáciles de usar para autenticar usuarios en la aplicación. Admite autenticación con contraseñas,

[24] https://www.jetbrains.com/webstorm/
[25] https://www.google.com/chrome/
[26] https://firebase.google.com/
[27] Formato de texto ligero para el intercambio de datos. Debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.
[28] Un kit de desarrollo de software o SDK es generalmente un conjunto de herramientas de desarrollo de software que le permite al desarrollador de software crear aplicaciones para un sistema concreto.

proveedores de identidades federadas populares, como Google[29], Facebook[30] y Twitter[31], y más opciones.

- *Firebase Hosting*. Proporciona *hosting*[32] estático rápido y seguro para la aplicación web.

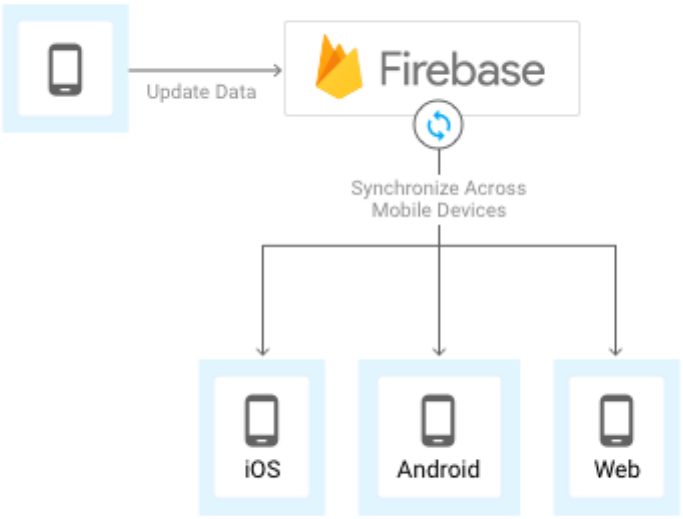

Figura 6. Esquema de sincronización de datos con Firebase

En mayo de 2016 Google anuncia Firebase 3 donde se mantiene lo anterior y muestra ampliaciones interesantes (ver figura 7).

[29] https://plus.google.com/
[30] https://es-es.facebook.com/
[31] https://twitter.com/
[32] El alojamiento web (en inglés: web hosting) es el servicio que provee a los usuarios de Internet un sistema para poder almacenar información, imágenes, vídeo, o cualquier contenido accesible vía web.

Figura 7. Firebase 3 muestra sus ampliaciones.

Ampliaciones en Firebase:

- *Cloud Messaging*. Solución multiplataforma que te permite enviar, de forma gratuita y segura, mensajes y notificaciones.
- *Storage*. Se creó para desarrolladores de aplicaciones que necesitan almacenar y proporcionar contenido generado por el usuario, como fotos o vídeos.
- *Remote Config*. Servicio en la nube que te permite cambiar el aspecto y el comportamiento de tu App sin pedirles a los usuarios que descarguen una actualización de la App.
- *Test Lab*. Prueba tu App en dispositivos alojados en un centro de datos de Google.
- *Crash Reporting*. Información completa e interactiva para ayudar a diagnosticar y solucionar problemas en tu App.
- *Notifications*. Es un servicio gratuito para desarrolladores de aplicaciones para dispositivos móviles que permite enviar notificaciones orientadas a los usuarios. Está basado en el SDK de Cloud Messaging.
- *App Indexing*. Anteriormente Google App Indexing, lleva tu App a la Búsqueda de Google. Si tu App ya está instalada cuando los usuarios buscan contenido relacionado, podrán iniciar tu App directamente desde los resultados de la búsqueda. Esta características solo tiene sentido en aplicaciones móviles.

- *Dynamic Links*. Son URLs inteligentes que cambian de comportamiento de forma dinámica para proporcionar la mejor experiencia en diferentes plataformas.
- *Invites*. Solución multiplataforma para enviar invitaciones personalizadas por correo electrónico y SMS, incorporar usuarios y medir el impacto de las invitaciones.
- *AdWords*. Integración sencilla con Google AdWords. Llega a clientes potenciales con anuncios en línea gracias a las analíticas de Firebase.
- *AdMob***.** Creado por Google es un método sencillo para monetizar aplicaciones a través de publicidad orientada integrada en las mismas. Está disponible en el SDK de Firebase para su fácil integración.

JavaScript. Lenguaje de programación interpretado, dialecto del estándar ECMAScript[33]. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). El crecimiento del desarrollo web con este lenguaje de programación ha ido en aumento constante. Destaca la gran cantidad de frameworks, plugins y módulos basados en JavaScript que han tenido éxito entre la comunidad de desarrolladores (ejemplos: NodeJS, AngularJS, entre otros)

AngularJS[34]: *Framework* de código abierto para JavaScript que incorpora patrones como SPA y MVC con el objetivo de hacer el desarrollo y las pruebas más sencillas. Está mantenido por una amplia comunidad y goza de gran popularidad (ver figura 8).

[33] https://github.com/tc39/ecma262
[34] https://angularjs.org/

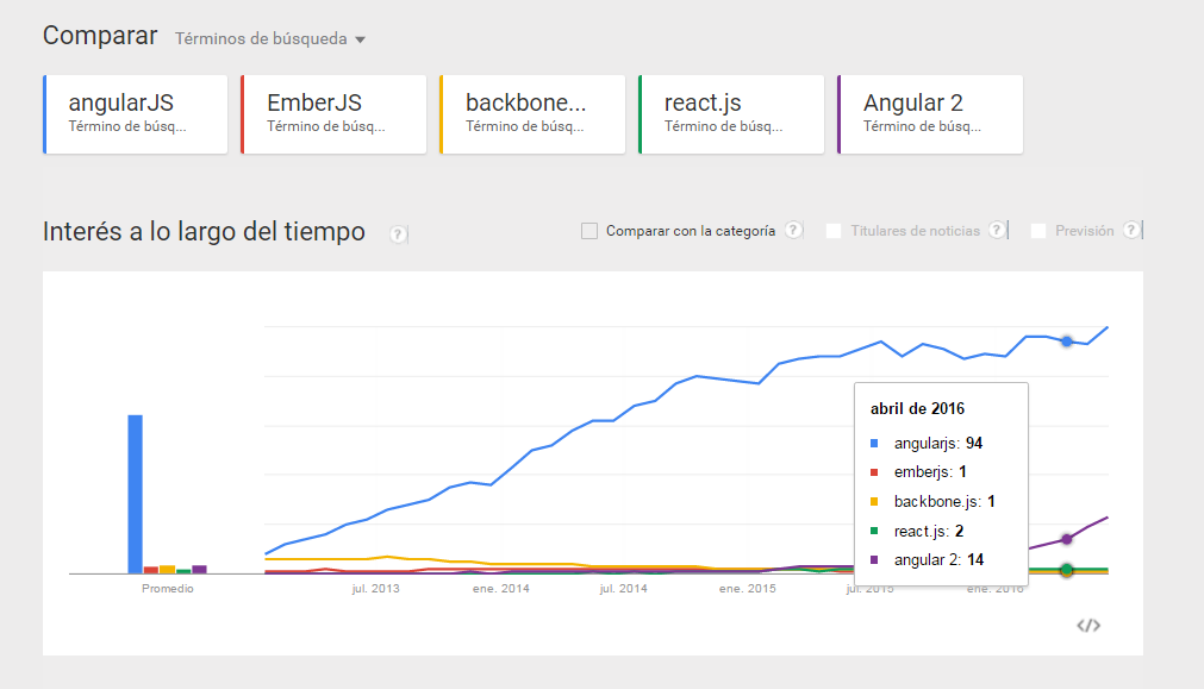

Figura 8. Popularidad en Google Trends de frameworks JavaScript

Bootstrap[35]: *Framework*[36] o estructura de herramientas de código abierto para el diseño de aplicaciones sobre *HTML*[37] y *CSS*[38]. Bootstrap es uno de los proyectos más famosos (ver figura 9) y con mayor comunidad de Github[39]. Diseñado para funcionar en multi-plataformas.

[35] http://getbootstrap.com/
[36] En el desarrollo de software, un framework o infraestructura digital, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software.
[37] Lenguaje de marcas de hipertexto en inglés HyperText Markup Language.
[38] Hoja de estilo en cascada en inglés Cascading Style Sheets.
[39] GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git.

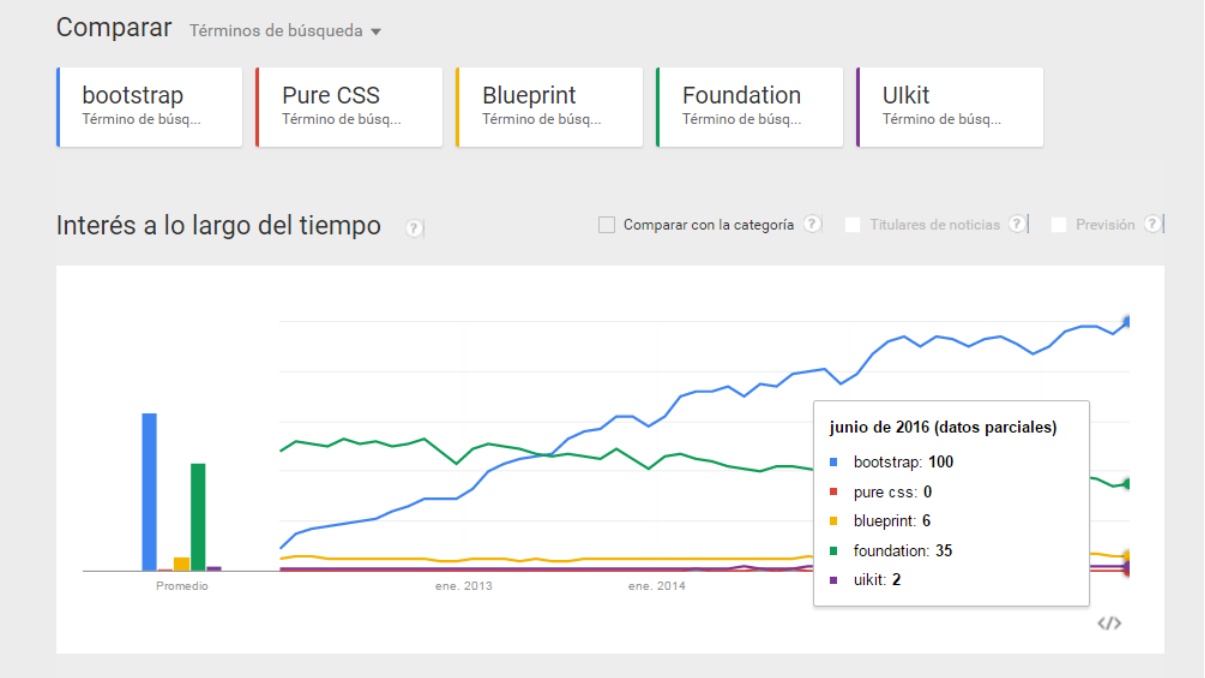

Figura 9. Popularidad en Google Trends de *frameworks* de diseño

Trello[40]: Gestor de tareas que permite el trabajo de forma colaborativa mediante tableros compuestos de columnas que representan distintos estados. Se basa en el método Kanban (ver sección 3.2.3) para gestión de proyectos, con tarjetas que viajan por diferentes listas en función de su estado.

4.2. Conclusiones Parciales

En este capítulo se han estudiado y seleccionado herramientas y tecnologías de desarrollo modernas, útiles para el rápido desarrollo y la obtención de calidad del software. Las herramientas y tecnologías escogidas tienen un gran alcance, son muy populares dentro de la comunidad de desarrollo y marcan una tendencia a seguir creciendo. Además, con la selección hecha se dan los primeros pasos para la consecución de un sistema escalable, seguro, de alta disponibilidad y fuerte fiabilidad.

[40] https://trello.com/

5. Desarrollo del Sistema

A continuación se describe cómo se desarrolla el proyecto según los objetivos planteados al inicio, requisitos funcionales y no funcionales detectados, enfoques utilizados, diseño y arquitectura del software, ampliaciones incorporadas y planificación realizada. Por último, se mencionan algunos problemas encontrados durante el desarrollo del proyecto y las soluciones aplicadas.

5.1. Fase Inicial

En las fases de inicialización del proyecto (a partir de ahora Socialize[41]), se quería desarrollar solo la capa cliente para dar solución a los objetivos planteados. Crear un Back-End significaba dedicar esfuerzo al desarrollo del mismo, con lo que el proyecto tomaría una gran envergadura y complejidad.

Lo que se quería hacer era centrar todo el esfuerzo en tener una cartera de visualizaciones que fueran compatibles con la cartera de servicios del GPLSI y se pudieran acoplar en un cuadro de mandos central.

5.1.1. Social Observer[42]

El GPLSI cuenta con varias herramientas de PLN las cuales están concebidas como Servicios Webs pero carecen de interfaces visuales capaces de mostrar la información obtenida de dichos servicios. Una de estas herramientas es Social Observer, el cual fue elegido como el proveedor de servicios de analítica de textos que se debía utilizar en Socialize.

De esta forma, se pretendía que la información generada por los servicios de PLN del GPLSI estuviera disponible para su representación visual a través de Socialize.

[41] Socialize fue el nombre que se decidió para el proyecto en esta fase.
[42] https://gplsi.dlsi.ua.es/gplsi13/es/node/249

5.2. Análisis de requisitos

Los objetivos del proyecto estaban bien definidos pero dejaban muchas puertas abiertas que se debían controlar. Era necesario dar mayor valor a la idea inicial sin cambiarla o perder su esencia. Así se enfocaron los esfuerzos en realizar un resumen del sistema e ir entrando en detalles en forma de módulos. Horas de investigación, reuniones semanales con el cliente (ver figura 10 y 11) y la colaboración de los equipos de Social Observer y Social Analytics fueron dándole una aspecto renovado a la idea inicial, aportando mayor valor.

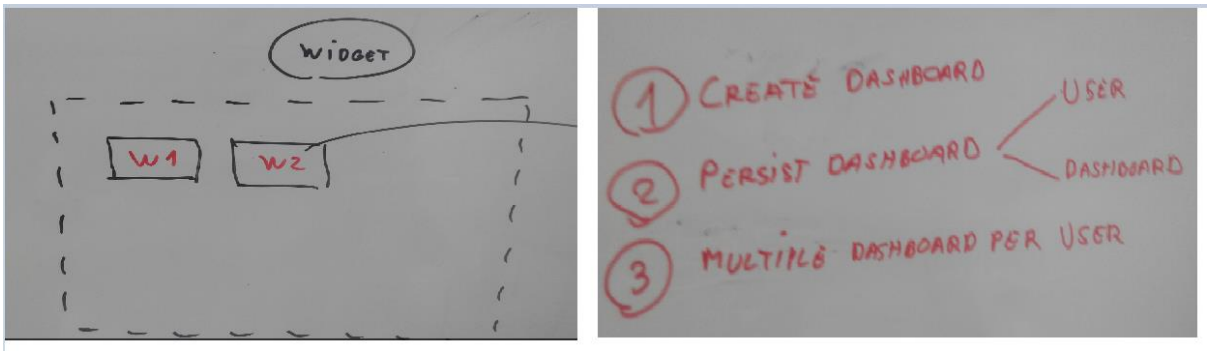

Figura 10. Dibujos en reuniones Scrum.

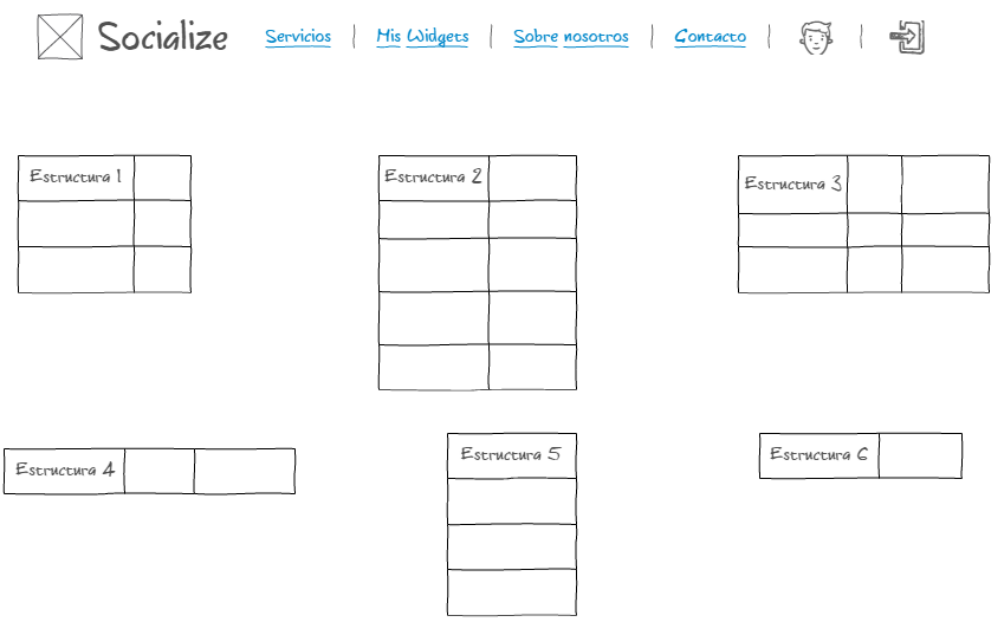

Figura 11. Se muestran los primeros Mockups[43] al cliente

[43] Boceto básico y de baja calidad del desarrollo de una página web o el diseño de una interfaz

5.2.1. Ampliaciones de la idea inicial

Después de realizado un análisis de requisitos se amplió la idea inicial con los siguientes aspectos.

- **Incorporación de Firebase**[44] al proyecto. Esto daba solución a las cuestiones referentes a la necesidad de tener Back-End y persistencia de datos propios para el control de los mismos. A la vez, no rompía con la idea de no dedicar tiempo a la construcción de un *Back-End* desde cero. Firebase es un proveedor de servicios *BAAS*[45] y su misión es brindar todos los servicios necesarios para abstraer esta capa en su totalidad, lo que suponía concentrar los esfuerzos solamente en el *Front-End*.
- **Gestión de usuarios**. Socialize incorporaría la gestión de usuarios, con lo cual tendría sentido controlar cuadros de mandos y visualizaciones que pertenecen a un usuario. Sería mucho más intuitiva la aplicación.
- **Mocks de Servicios**: Social Observer estaba en pleno desarrollo y no tenía finalizada el API[46] para poder consumir sus servicios desde Socialize. Por tanto, se llegó a un acuerdo en centrar las primeras iteraciones en la obtención de visualizaciones acopladas a *Mocks*[47] de los servicios. Con esta idea, al finalizar el desarrollo del API por parte del equipo de Social Observer, sería sencilla la integración del mismo.

  Para la primera versión se identificaron los siguientes módulos del sistema,

  - Control de usuarios
  - Visualizaciones por usuario
  - Cuadros de mandos por usuario

5.2.2. Historias de usuario

Las historias de usuario son un recordatorio de las reuniones con el cliente, un acuerdo formal de mínimos para dar por buena la funcionalidad descrita y esperada. Las historias de usuario y las tareas que surgen para lograr sus criterios de aceptación se incorporan al Product Backlog (ver sección 3.4).

[44] Véase apartado Herramientas - Firebase
[45] Backend como servicio
[46] La interfaz de programación de aplicaciones, abreviada como API (del inglés: Application Programming Interface), es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
[47] Objetos simulados (mock object, objetos de pega) a los objetos que imitan el comportamiento de objetos reales de una forma controlada.

A continuación se detallarán las historias de usuario siguiendo la estructura de sentencias de la siguiente forma: **Como, Quiero, Para**.

1. Autenticación.

   Como *usuario no validado*

   Quiero *poder autenticarme de diversas formas*

   Para *acceder a las funcionalidades ofrecidas en el sistema a usuarios validados*

   Criterios de Aceptación:

   - Quiero poder registrarme a través de Facebook, Twitter y email.
   - Si estoy registrado quiero poder acceder a través de Facebook, Twitter y email.
   - Quiero que la aplicación recuerde mi configuración al volver a acceder.
   - No quiero que otros usuarios puedan ver mis datos.

2. Visualizaciones.

   Como *usuario validado*

   Quiero *crear visualizaciones*

   Para *analizar a través de estas los datos obtenidos de servicios de PLN del GPLSI.*

   Criterios de Aceptación:

   - Quiero poder crear una visualización asociada a un servicio de PLN del GPLSI y editar la forma de visualizar la misma.
   - Quiero poder editar la configuración de una visualización, cambiándole el servicio de PLN asociado y la forma de visualizarlo.
   - Quiero poder eliminar una visualización.
   - Quiero poder listar las visualizaciones creadas.
   - No quiero que se pueda asociar más de un servicio de PLN a una misma visualización.

3. Cuadros de mando.

   Como *usuario validado*

   Quiero *integrar visualizaciones en cuadros de mando*

   Para *poder realizar un análisis global sobre los datos visualizados.*

Criterios de Aceptación:

- Quiero poder crear cuadros de mando que se puedan asociar a colecciones de entidades a analizar.
- Quiero poder integrar visualizaciones en cuadros de mando, las cuales pueda cambiar de posiciones y tamaños.
- Quiero poder editar los cuadros de mando, cambiando la colección asignada y las visualizaciones integradas.
- Quiero poder listar los cuadros de mando creados.
- Quiero poder eliminar un cuadro de mando creado.
- No quiero que un cuadro de mandos se asocie a más de una colección de entidades a analizar.

4. Sindicación de cuadros de mando.
   ***Como** usuario validado*
   ***Quiero** sindicar cuadros de mando*
   ***Para** poder incrustar las visualizaciones en webs de terceros.*

   Criterios de Aceptación:

   - Quiero que exista una opción de compartir un cuadro de mandos a través de una ruta web.
   - Quiero que se pueda compartir un cuadro de mandos para la sindicación en webs de terceros a través de un *iframe*[48] .
   - Quiero que cualquiera pueda ver un cuadro de mandos compartido a través de la ruta web o el *iframe*.
   - No quiero que otros puedan ver el cuadro de mandos a través de la ruta web o el *iframe* cuando lo deje de compartir.

[48] Elemento HTML que permite insertar o incrustar un documento HTML dentro de un documento HTML principal.

5.2.3. Requisitos funcionales

Después de la fase inicial se identifican los requisitos funcionales a tener en cuenta en el desarrollo de Socialize.

- Autenticación a través de distintos medios (email, redes sociales).
- Crear, modificar, listar y eliminar visualizaciones.
- Conectar visualizaciones con servicios de PLN del GPLSI.
- Crear, modificar, listar y eliminar cuadros de mando.
- Integrar, cambiar tamaños, cambiar posiciones y eliminar visualizaciones en cuadros de mando.
- Opción para la sindicación de visualizaciones a través de los cuadros de mando.

5.2.4. Requisitos no funcionales

En este apartado se identifican los requisitos no funcionales que deben existir para que el sistema sea atractivo, usable, rápido y confiable.

- **Desempeño**. Se requiere que el sistema ejecute sus funcionalidades de forma rápida, sin demoras.
- **Escalabilidad**. Se requiere que el sistema reaccione de forma adecuada a un incremento en los usuarios y en los datos que estos gestionan.
- **Usabilidad**. El sistema debe ser fácil de usar e intuitivo.
- **Seguridad**. Los datos de los usuarios deben estar protegidos.
- **Disponibilidad**. El sistema debe estar disponible el 100% del tiempo.

5.3. Diseño y Arquitectura

Desde su concepción Socialize sería un proyecto *Front-End* sin dependencias de *Back-End*, el cual estaría relacionado con los datos a través de servicios. Con la evolución del proyecto y dado el aspecto cambiante de los requisitos funcionales surgió la necesidad de tratar datos propios como usuarios, esquemas de visualizaciones, cuadros de mando, entre otros.

Con el objetivo de darle solución a este problema surgió la idea de integrar Firebase (ver sección 4.3), servicio de *Back-End*, al proyecto.

La incorporación de AngularJS en el *Front-End* y Firebase en el *Back-End* dio lugar a una arquitectura de tres capas y dos niveles (ver figura 12). Se divide en **Presentación | Lógica** en el nivel cliente y **Lógica | Datos** en el nivel servidor [42].

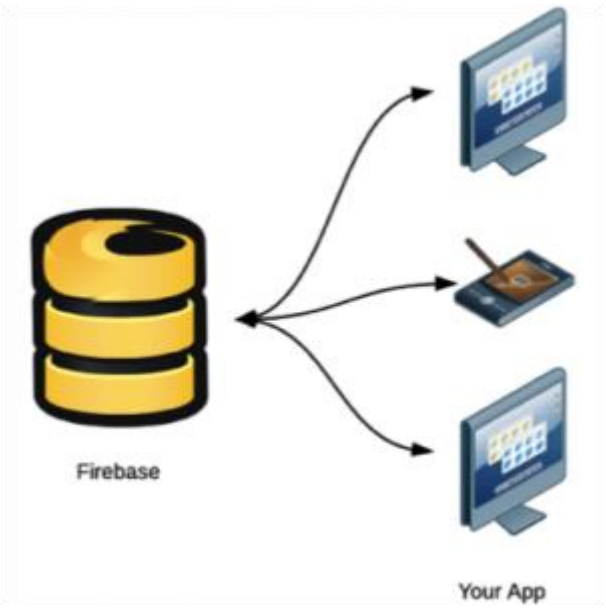

Figura 12. Arquitectura de dos niveles

5.3.1. Patrones

De los patrones estudiados se escogieron los siguientes patrones de arquitectura, estructuración y diseño.

Patrones de Arquitectura:

- **MVC**[49]. Este patrón separa en 3 capas la aplicación. El objetivo es tener un código más claro, reutilizable y fácil de testear.

Patrones de Diseño:

- **Single Page Application**[50]. El objetivo de este patrón es dar una experiencia más fluida al usuario, al estilo de una aplicación de escritorio. Todo el código se carga al inicio de la aplicación en una sola página y los recursos se visualizan dinámicamente según se requiera y se van agregando a la página.
- **Data Binding**[51]. Este patrón inherente en AngularJS nos permite la fácil comunicación y actualización de datos entre el modelo y la vista.
- **Module Pattern**[52]. Implementado en la aplicación con el objetivo de modularizar. Cada módulo cumple un objetivo específico. Así se separa la lógica de cuadros de mando, visualizaciones y usuarios.
- **Column Drop**. Utilizado con el objetivo de ser "Responsive" (adaptativo) y funcionar correctamente en multiplataforma. Un ejemplo gráfico del patrón puede ser el que se muestra en la figura 13:

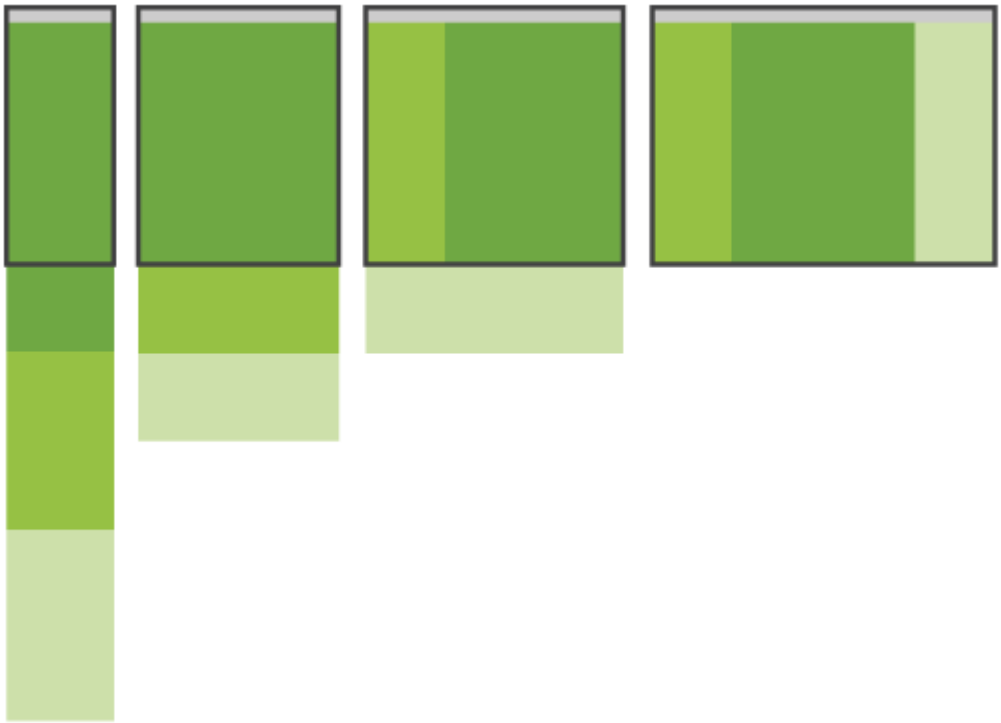

Figura 13. Patrón de Diseño Column Drop [29]

[49] Patrón Modelo-Vista-Controlador
[50] Conocido como SPA, el patrón de Aplicación de Página Única
[51] Data Binding [28]
[52] Module pattern [28]

A medida que se achique la resolución de la pantalla en el dispositivo utilizado, los componentes de la aplicación se encogen hasta un límite determinado y se posicionan en forma de pila. A su vez, a medida que la resolución de pantalla aumenta los componentes de la aplicación retoman su tamaño y posición original.

Patrones de Estructura de Componentes:

- **Patrón Específico**. Dada la envergadura del proyecto, fue necesario establecer un patrón para la organización del código. Se agrupan los controladores, directivas, filtros y servicios se agrupan en diferentes carpetas del proyecto [27].

5.3.2. Estructura de la Base de Datos

Firebase provee una base de datos no relacional orientada a documentos donde se almacenan los datos en formato *JSON* (ver sección 4.3). Se ha realizado un modelado UML[53] de clases, el cual se expone a continuación en la figura 14.

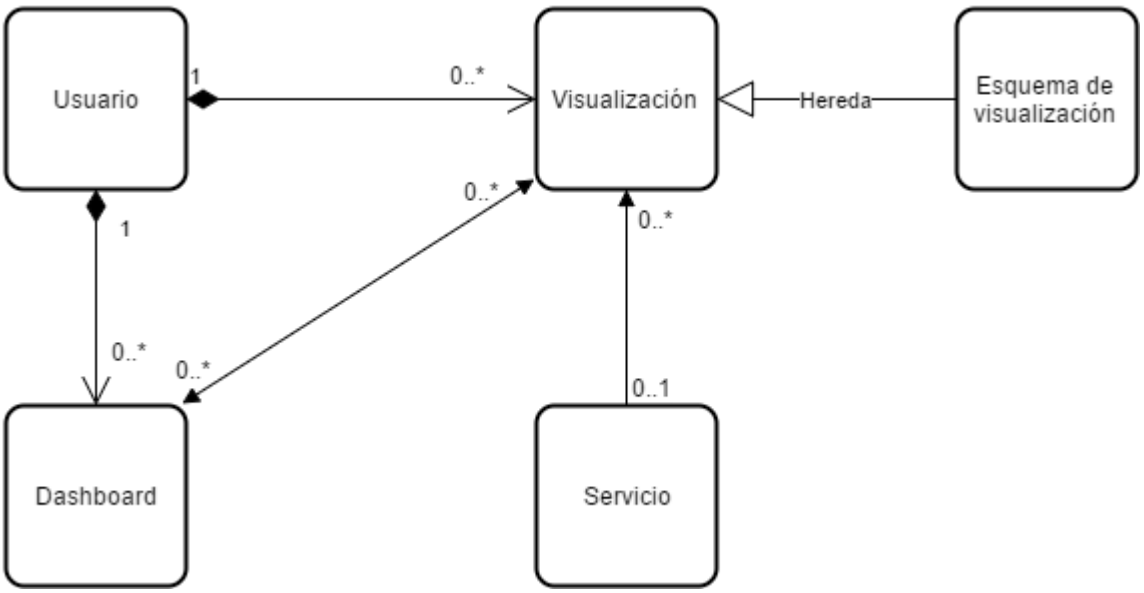

Figura 14. Diagrama UML de clases

[53] Lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad

Listado de Componentes:

- **Usuario**. Cualquier persona que interactúa con la aplicación.
- **Servicio**. Referencia a servicios de PLN integrados.
- **Esquema de visualización**. Estructura con componentes básicos de una visualización.
- **Visualización**. Los datos que contiene el objeto se convierten en características visuales con un objetivo específico al integrarlo en la aplicación.
- **Cuadro de mando**. Un cuadro de mando desde donde el usuario podrá interactuar con sus visualizaciones.

Relaciones entre componentes:

- **Usuario - Cuadro de mando.** Un usuario puede tener de cero a muchos cuadros de mando. A su vez un cuadro de mando pertenece sólo a un usuario. Se destaca que un cuadro de mando no puede existir sin un usuario asociado al mismo.
- **Usuario - Visualización**. Un usuario puede tener de cero a muchas visualizaciones. A su vez una visualización pertenece sólo a un usuario. Se destaca que una visualización no puede existir sin un usuario asociado a la misma.
- **Cuadro de mando - Visualización**. Un cuadro de mando puede tener de cero a muchas visualizaciones. A su vez una visualización puede estar en cero a muchos cuadros de mando.
- **Visualización - Esquema de Visualización**. Una visualización hereda características de un esquema de visualización.
- **Visualización - Servicio**. Una visualización puede tener asociado un servicio o ninguno. A su vez un servicio puede estar asociado a cero o muchas visualizaciones.

5.3.3. Casos de uso

A continuación se expondrán los diagramas de caso de uso que se crearon a partir de los módulos definidos en el sistema.

5.3.3.1. Módulo de Autenticación

Se separa la autenticación en un módulo para brindar mayor peso a la seguridad de los datos y los permisos del usuario.

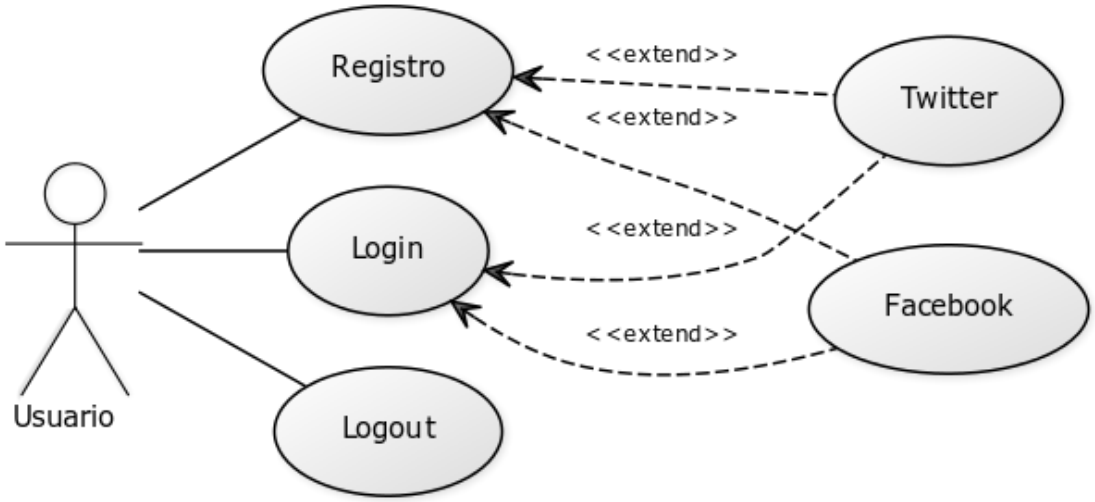

Figura 15. Diagrama de caso de uso de la Autenticación

5.3.3.2. Módulo de Visualización

El módulo de visualización contiene todas las funcionalidades para crear, modificar y eliminar una visualización.

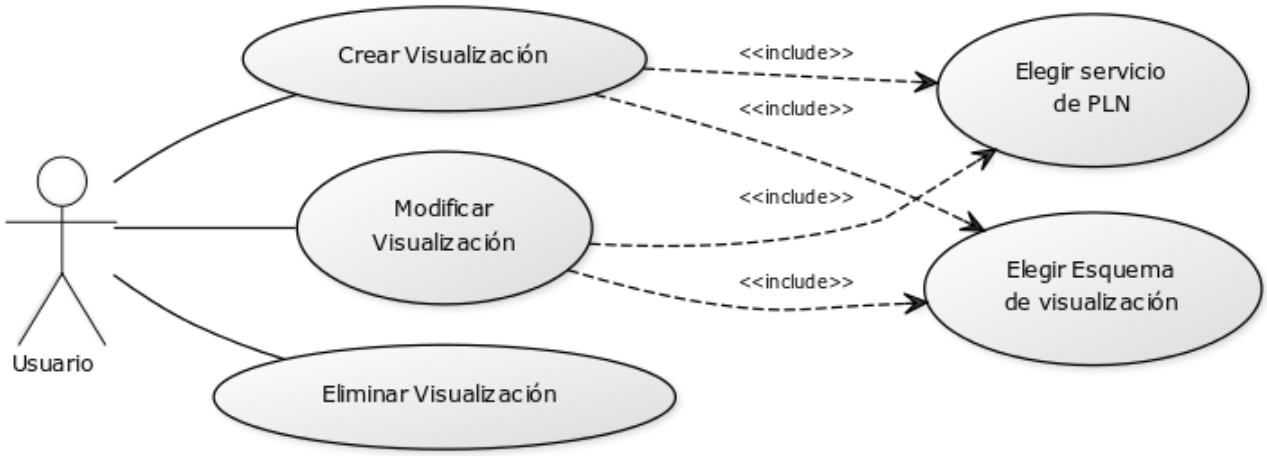

Figura 16. Diagrama de caso de uso del módulo Visualización

5.3.3.3. Módulo Cuadro de Mandos

El módulo de cuadros de mando contiene todas las funcionalidades para crear, modificar y eliminar cuadros de mando. Además, presenta otras funcionalidades como integrar visualizaciones, compartir y definir colecciones de entidades[54] dentro de cuadros de mando.

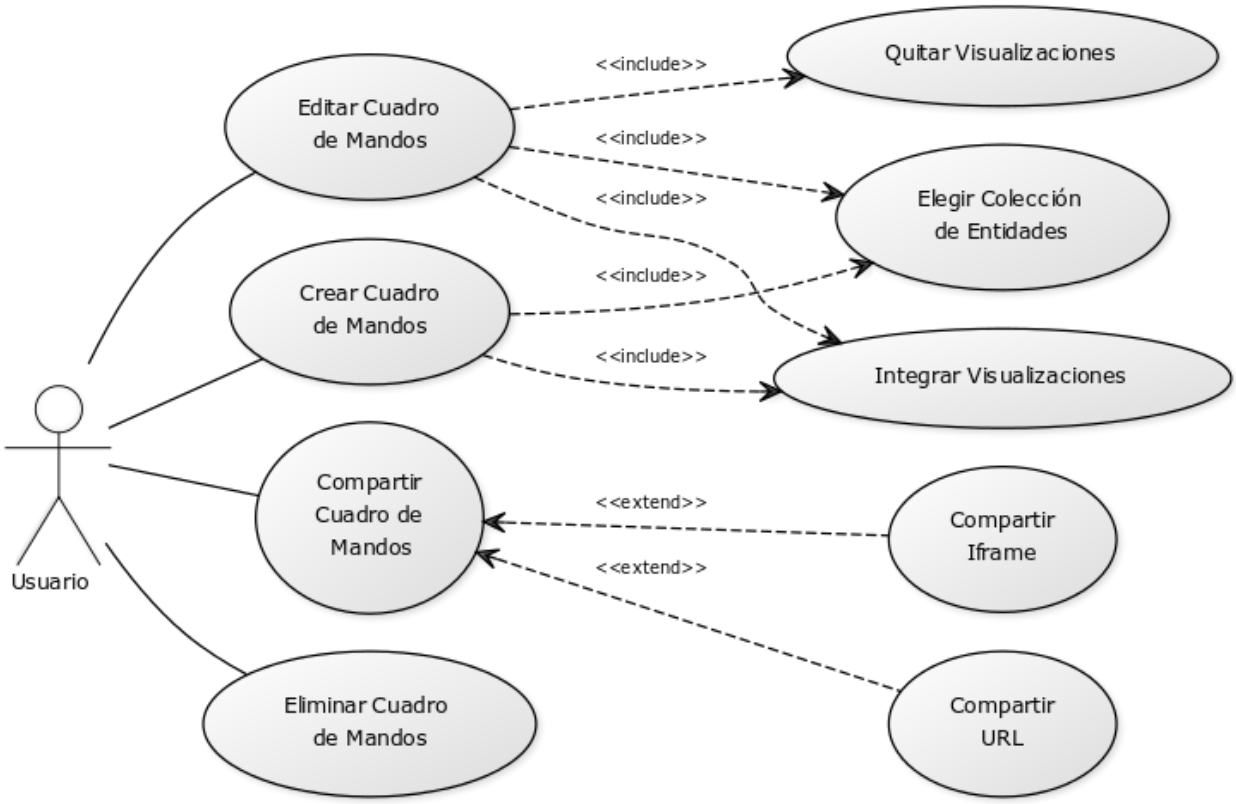

Figura 17. Diagrama de caso de uso del módulo cuadros de mando

5.3.4. Diagramas de navegación

Los diagramas de navegación tienen como objetivo definir cómo se le proporcionará a cada usuario del sistema el acceso a la información y la funcionalidad que le es relevante para llevar a cabo sus tareas y qué secuencias de caminos deberán seguir para conseguirlas.

[54] Las entidades son los conceptos sobre los que se realiza el análisis de texto en redes sociales.

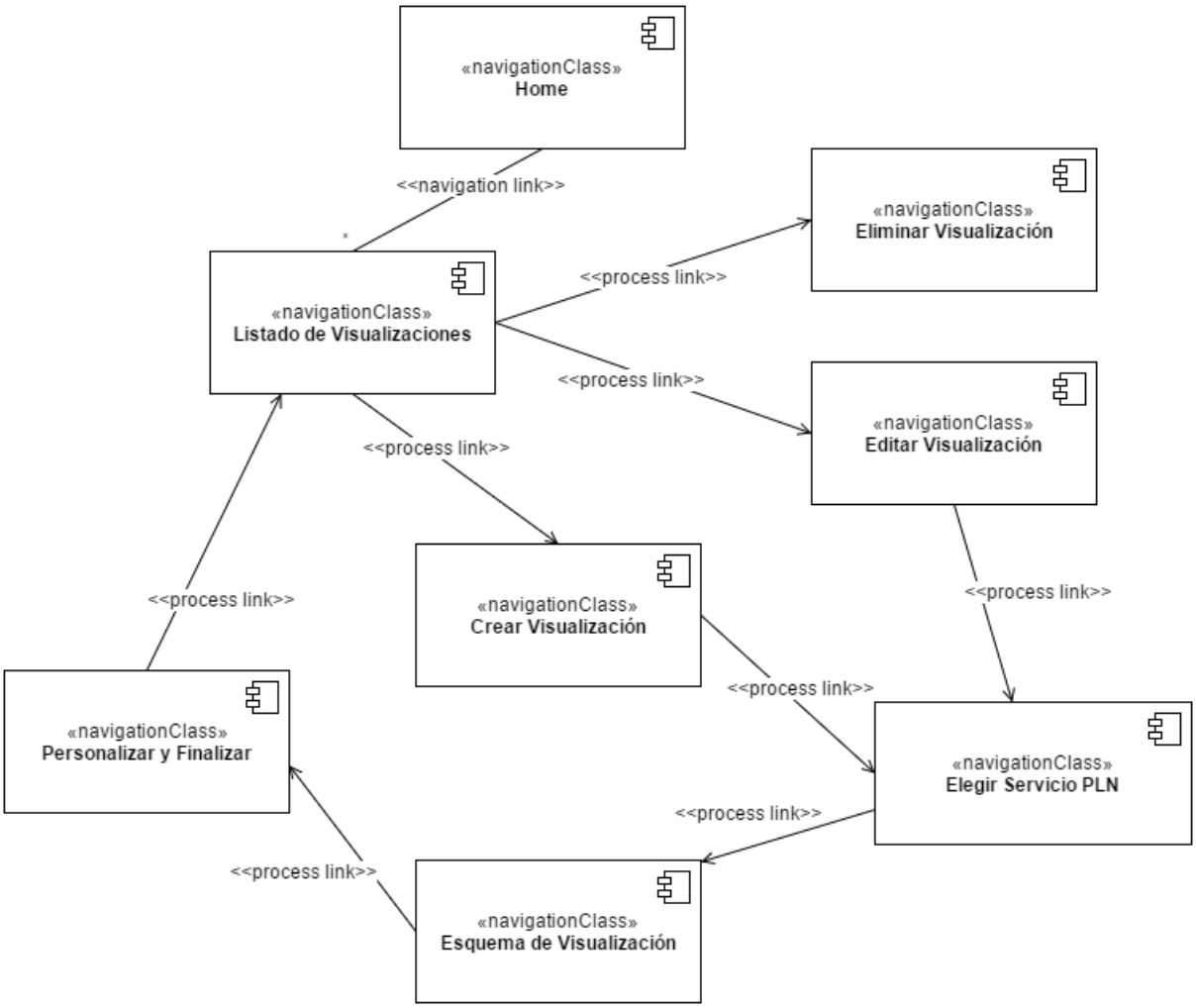

Figura 18. Diagrama de navegación del módulo de visualización

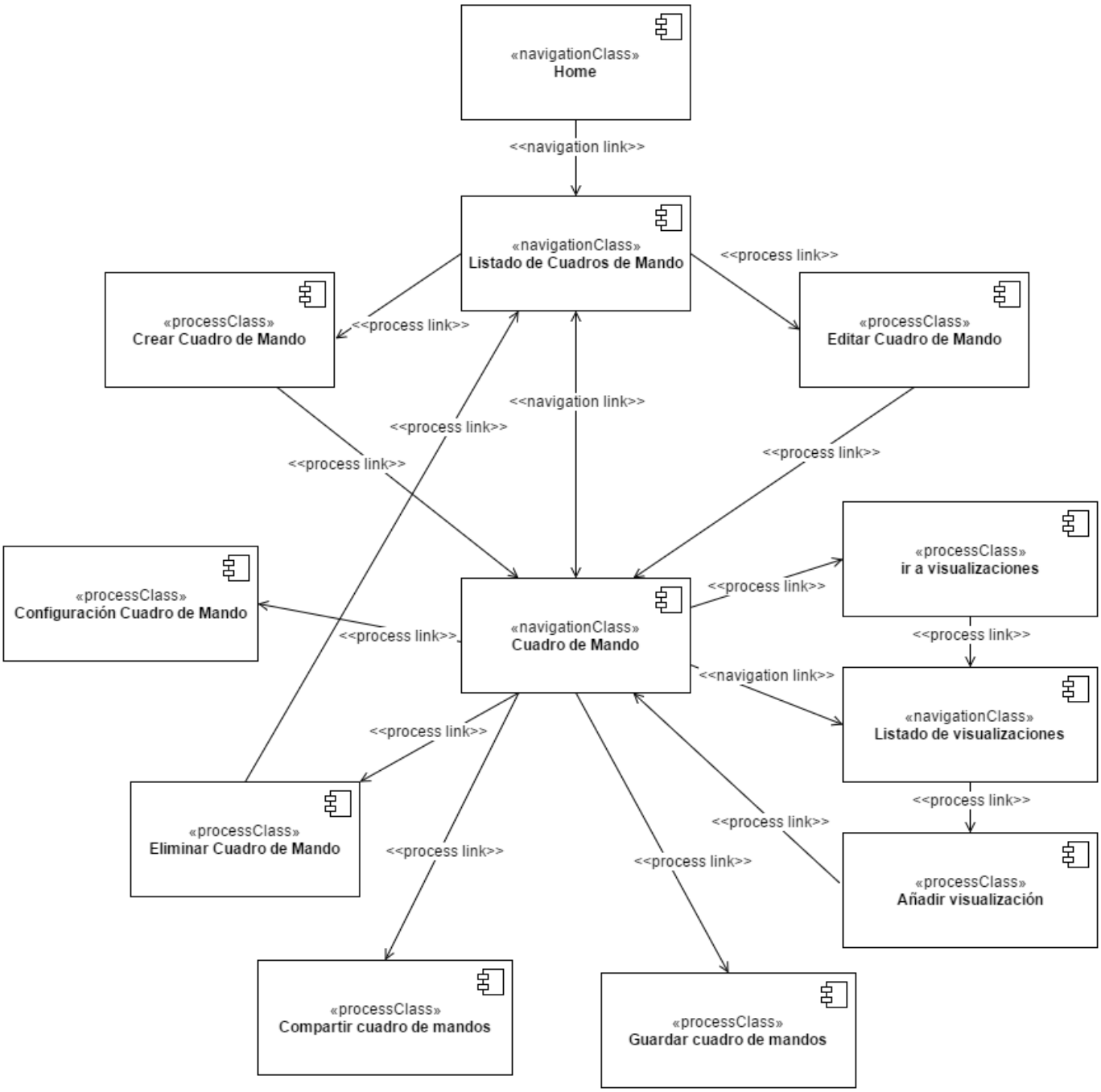

Figura 19. Diagrama de navegación del módulo de cuadros de mando

5.4. Planificación y desarrollo de iteraciones

La planificación de las iteraciones o Scrum Sprints y el desarrollo de los requisitos con mayor prioridad fueron guiando a Socialize por la ruta correcta para obtener un mayor valor en el menor tiempo posible.

La herramienta Trello [36] sirvió de apoyo para dar un seguimiento de los Sprints y las tareas derivadas de estos (ver figura 20).

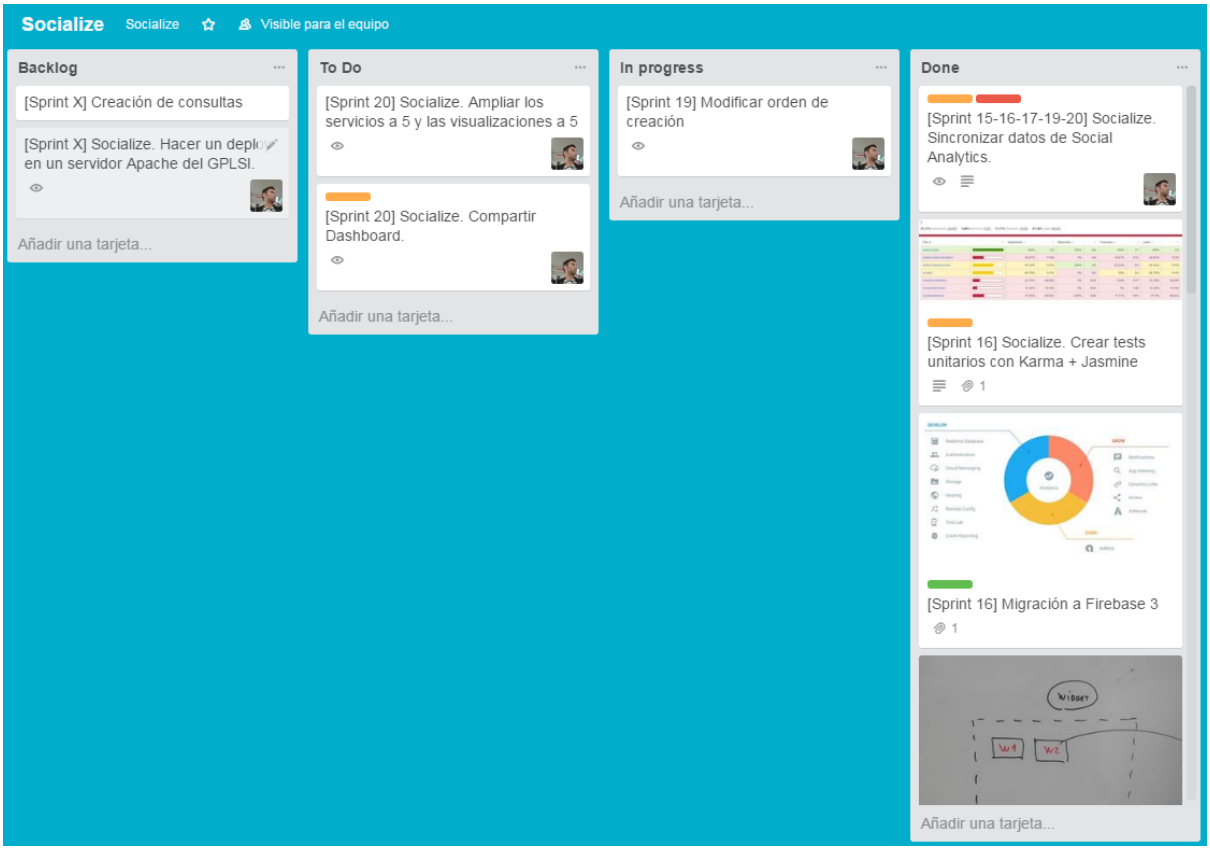

Figura 20. Seguimiento de tareas por Sprints en Trello

Se utiliza un enfoque Top-Down [34] para darle mayor valor a la idea inicial sin cambiarla o perder su esencia. Así se enfocaron los esfuerzos en realizar un resumen del sistema e ir entrando en detalles en forma de módulos.

Horas de investigación, reuniones semanales y la colaboración de los equipos de Social Observer y Social Analytics fueron dándole un aspecto renovado a la idea inicial, aportando mayor valor.

En los Sprints iniciales fue necesario realizar un análisis de requisitos (ver sección 5.2), definir el diseño y la arquitectura del sistema (ver sección 5.3) y decidir el compendio de herramientas y tecnologías a usar (ver sección 4).

Se necesitó un período puramente investigativo para explorar los sistemas de código abierto que existían hasta esos momentos que pudieran cumplir los objetivos definidos, explorar sus ventajas y desventajas y tomarlas como referencia para el desarrollo del sistema (ver sección 2).

Los siguientes Sprints (ver sección 12.1) se concentraron mayormente en el desarrollo del sistema. Algunas de las tareas que se planificaron se exponen a continuación.

- Elegir y adaptar plantilla web para el proyecto.
- Crear registro y logueo básico. Incorporar persistencia de usuarios con Firebase (ver sección 4).
- Incorporar opción de logueo con Twitter y Facebook
- Investigación de módulos de AngularJS para la creación de un cuadro de mandos.
- Automatización de la construcción y despliegue del proyecto.
- Investigación sobre posibles módulos de gráficos para la integración de visualizaciones.
- Crear la lógica[55] de cuadros de mando.
- Crear la lógica de visualizaciones.
- Integrar visualizaciones con cuadros de mando.
- Migrar Firebase (ver sección 4) y AngularFire[56] a la versión 3
- Sincronizar e integrar los servicios de PLN el Social Analytics
- Ampliar funcionalidades, compartir cuadros de mando para su sindicación en aplicaciones de terceros.

[55] Crear, modificar, listar y eliminar objeto
[56] Librería de código abierto que integra AngularJS con Firebase [37]

5.5. Problemas y Soluciones

A lo largo del desarrollo de Socialize han aparecido diferentes problemas a los cuales se le han buscado soluciones y/o alternativas sin desviarse de la ruta trazada. La aparición y resolución de los mismos han servido de experiencia para estar preparados y saber qué hacer en casos parecidos. A continuación se irán exponiendo los principales problemas y sus soluciones.

Idea inicial muy básica.

Al inicio del proyecto se aseguraba que no haría falta un Back-end. En teoría no se iban a manejar datos propios por tanto no era necesario controlar lógica alguna. A la idea inicial era necesario añadirle retoques para que Socialize fuera más que una simple lista de visualizaciones.

La solución fue realizar un análisis más profundo, utilizando el enfoque Top-Down, donde se ampliaron los requisitos y surgieron nuevos módulos (usuarios y cuadros de mando). Al añadir lógica de negocio, fue necesario replantearse la idea de una capa servidor sin perder el enfoque hacia la capa cliente. En este punto la solución fue Firebase (ver sección 4.3), Back-end como servicio, el cual cumplía con el rol de servidor administrando la lógica de negocio y a su vez requería un mínimo esfuerzo y poco tiempo para integrarlo al proyecto. Además, Firebase (ver sección 4.3) ofrecía muchas funcionalidades a tener en cuenta en un futuro.

Elección de módulos de gráficos.

La elección de uno o varios módulos de gráficos que pudieran integrarse en el proyecto, específicamente con el módulo de cuadros de mando Angular Gridster fue un quebradero de cabezas, pues hacer pruebas para comprobar la integración era algo tedioso.

La solución fue realizar un estudio de los módulos existentes, donde se revisaron investigaciones, foros, trabajos relacionados y se crearon indicadores para elegir el módulo correcto: de código abierto, angularizado, fuerte comunidad en Github, actualizado, fácil acoplamiento con Angular Gridster.

Al final solo quedaron tres módulos: angular-nvd3, Amchart, Highcharts. La selección de Highcharts fue basada en la fácil integración con Angular Gridster y su amplio repertorio de gráficos.

Integración de Highchart con Angular Gridster.

A pesar de que Highchart era el módulo de gráficos más adecuado, su integración en el cuadro de mandos de Angular Gridster fue complicada (ver figura 21). Los ítems de cuadro de mandos son especies de cajas que pueden ser modificadas dinámicamente alterando sus tamaños y cambiando sus posiciones. El objetivo era introducir gráficos que alterarán sus tamaños y posiciones junto con las cajas sin salirse fuera de las mismas.

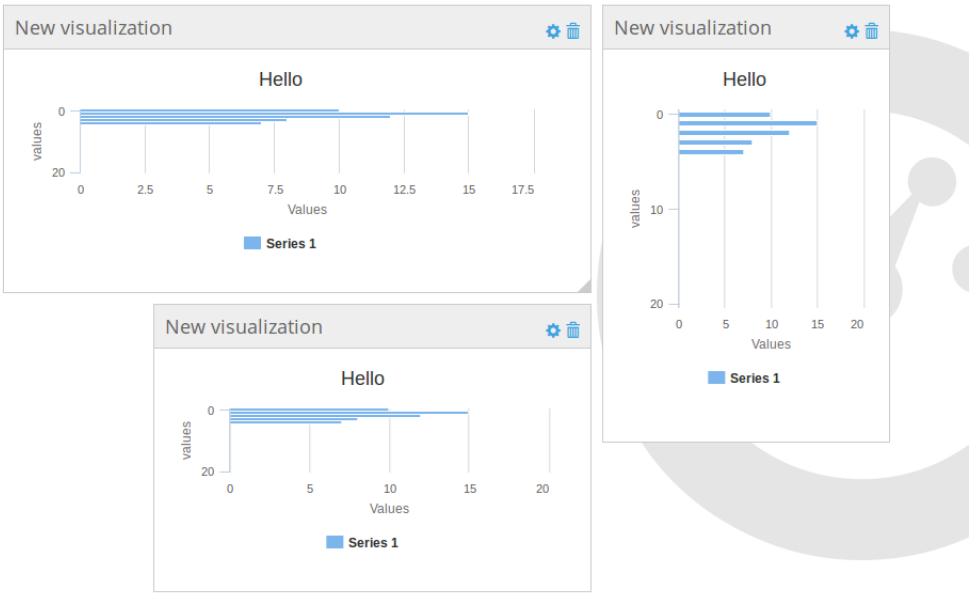

Figura 21. Gráficos Highchart dentro de ítems del cuadro de mando

La solución fue crear funciones asociadas a eventos que se disparaban al cambiar algún ítem del cuadro de mandos, las cuales a su vez modificaban los gráficos de Highchart (ver figura 22).

El configurador de cuadros de mando de Angular Gridster provee una serie de eventos para cuando una caja inicia el cambio, está cambiando o termina de cambiar de tamaño. Al evento que inicia se le asocia una función de ocultar el gráfico y al evento que finaliza se le asocia otra de cambiar el tamaño del gráfico de acuerdo a su contenedor (la caja), posteriormente se habilita el gráfico para su visualización.

```
    resizable: {
    enabled: true,
        handles: ['n', 'e', 's', 'w', 'ne', 'se', 'sw', 'nw'],
    start: function(event, $element, visualization) {VisualizationService.hideVisualization(visualization)},
    resize: function(event, $element, visualization) {}, // optional callback fired when item is resized,
    stop: function(event, $element, visualization) {VisualizationService.resizeVisualization(visualization)}
},
```

Figura 22. Eventos de renderizado para Angular Gridster

Cambio de servicios de procesamiento de lenguaje natural (PLN).

Los servicios de PLN del GPLSI se plantearon inicialmente para a ser consumidos mediante el Api Rest del Social Observer, el cual no estaba implementado aún. Debido a un cambio de requisitos dentro de Social Observer la implementación de esta funcionalidad fue perdiendo prioridad, con lo cual se tomó la decisión de cambiar el proveedor de servicios de PLN por el Social Analytics.

Desde Socialize se tomaron medidas a tiempo para prevenir este problema. La funcionalidad para conectar con los servicios de PLN quedó en espera y se crearon mocks o servicios simulados con datos constantes para realizar pruebas. Por último, al decidir cambiar de proveedor de servicios PLN fue necesaria una pequeña modificación en la estructura de la invocación a los servicios y la adaptación al cambio fue buena.

Integración con Servicios de Social Analytics.

En este caso fueron dos los problemas. Primero, los servicios del Social Analytics no estaban adaptados para ser consumidos desde un origen externo por lo que la integración con el mismo estuvo paralizada. Segundo, los servicios ofrecidos con protocolo http y Firebase Hosting (ver sección 4.3) solo permite comunicación con protocolos seguros HTTPS.

Para el primer caso fue necesaria la colaboración del equipo de Social Analytics para que se autorizara el consumo de sus servicios desde otro origen habilitando el CORS[57].

Para el segundo caso, el esfuerzo del equipo del Social Analytics para ofrecer todos sus servicios mediante el protocolo HTTPS era demasiado por lo que desde Socialize se buscaron alternativas para solucionar este problema. Se decidió cambiar de hosting a un servidor web del GPLSI con protocolo http, con lo cual la integración se pudo solventar de forma exitosa.

[57] Cross-Origin Resource Sharing [39].

Actualización a Firebase 3

A mediados de mayo de 2016, en pleno desarrollo del proyecto, Firebase (ver sección 4.3) anuncia la salida de la versión 3. De la mano de Google, Firebase ofrece ampliaciones interesantes y la comunidad de desarrolladores comienza a trabajar para actualizar el módulo AngularFire.

Gracias al manejador de paquetes Bower se actualiza la versión de AngularFire en Socialize. Además hubo que cambiar el sistema de autenticación debido a una variación con respecto a la versión anterior. Se realiza la actualización de forma exitosa en una semana.

5.6. Conclusiones Parciales

A lo largo de este capítulo se ha realizado el análisis de requisitos. Los requisitos funcionales y no funcionales se han obtenido gracias a las historias de usuario que surgieron en las constantes reuniones con el cliente.

Se ha estudiado, seleccionado e implementado la arquitectura del software. La selección de una arquitectura de 3 capas y dos niveles fue fundamental y sirvió para tomar decisiones de diseño que se traducen en atributos de calidad como Rendimiento, Seguridad, Coste de hacer un cambio, Confiabilidad, Escalabilidad, Usabilidad, entre otros.

Se ha comentado la planificación realizada en el desarrollo del sistema gracias a la guía de las metodologías ágiles elegidas (más detalles en la sección 10). De esta forma se han establecido las tareas prioritarias en forma de iteraciones y se ha alcanzado el objetivo de incorporar visualizaciones que utilizan los datos que se obtienen de los servicios de procesamiento de lenguaje natural desarrollados por el GPLSI.

6. Modelo de Negocio

En este capítulo se estudiará y seleccionará el modelo de negocio más adecuado y se identificará el mercado objetivo con dos ejemplos.

En la primera versión se tiene como objetivo dar a conocer la herramienta globalmente. Con ese propósito sólo se utilizarán datos que los servicios de Social Analytics ofrecen de forma pública y la licencia que se establece es gratuita.

6.1. Modelo Freemium

Para las siguientes versiones el modelo de negocio que más se adecua a Socialize es el Freemium, donde se ofrecen servicios básicos gratuitos, mientras se cobra por otros más avanzados, especiales o ampliados.

6.1.1. Licencias

Se expone a continuación un ejemplo de posibles licencias.

- **Gratis**: 1 cuadros de mando, 4 visualizaciones, 1 colección de 4 entidades. No se puede compartir el cuadro de mando.
- **Mínima**: 3 cuadros de mando, 20 visualizaciones, 1 colección con 7 entidades. Se puede compartir de forma embebida con un máximo de vistas de 500/día/cuadros de mando
- **Normal**: 6 cuadros de mando, 40 visualizaciones, 3 colecciones con 9 entidades por colección. Se puede compartir de forma embebida con un máximo de vistas de 1000/día/cuadros de mando
- **A medida**: Panel configurable donde tú eliges que quieres y se hace un cálculo del precio según lo elegido.

6.2. Mercado Objetivo

Socialize se orienta al sector B2B[58]. Es una plataforma de Business Intelligence[59] donde poder analizar y entender las opiniones de las personas en redes sociales acerca de uno

[58] Modelos de negocio en los que las transacciones de bienes o la prestación de servicios se producen entre dos empresas.

o varios temas correlacionados en un contexto. Es una herramienta útil para descubrir tendencias y necesidades, encontrar debilidades y ventajas e idear estrategias de marketing orientado.

6.2.1. Casos prácticos

A modo de ejemplo se pueden analizar los casos de una gran empresa y una pequeña empresa.

6.2.1.1. Caso de gran empresa

El Banco Sabadell[60] se prepara para el lanzamiento de un nuevo producto. Se trata de un crédito con intereses muy bajos el cual va orientado a conquistar el sector joven trabajador de la población. Antes necesitan definir estrategias de marketing.

Se utiliza Socialize como una de las herramientas para realizar un estudio de mercado y se analizan productos similares de otras entidades bancarias.

Se observa una tendencia a la negatividad. Se analizan las palabras más usadas en los comentarios y se descubre que existe una molestia por la burocracia que conlleva contratar un crédito, pues el cliente se tiene que desplazar hasta la oficina de la financiera y aportar mucha documentación.

Como respuesta el Banco Sabadell orienta su publicidad a la posibilidad de contratar el nuevo producto desde terminales móviles, sin tener que desplazarse a sus oficinas, con la posibilidad de adjuntar la documentación de forma digital.

6.2.1.2. Caso de pequeña empresa

La clínica de microinjerto capilar iHairMedical situada en el centro de Madrid ha sido inaugurada hace pocos meses. A pesar de mantener altos estándares de calidad y haber realizado una campaña publicitaria decente no tiene casi clientes.

Se decide realizar un análisis para comprender qué ha pasado. Al utilizar Socialize se observan tendencias negativas y las palabras más usadas dan a entender que se conoce muy poco acerca de los logros del doctor jefe de la clínica. Además se observa que uno de los

[59] Habilidad para transformar los datos en información, y la información en conocimiento, de forma que se pueda optimizar el proceso de toma de decisiones en los negocios.
[60] Por su tamaño es la cuarta entidad financiera española

usuarios que más escribe tiene una cierta reputación como bloguero del sector y sus comentarios son ofensivos.

La decisión que se toma es la de realizar una campaña publicitaria centrada en la experiencia del doctor jefe, donde se incluye su doctorado en Canadá y los logros obtenidos en los más de 15 años de experiencia en el sector. Además se realiza un trato personalizado con algunos blogueros de prestigio, donde se les enseña la clínica y tienen un diálogo personal con el doctor.

6.3. Conclusiones Parciales

En este capítulo se ha estudiado y establecido un modelo de negocio adecuado al sistema desarrollado. En la primera versión se estableció que el sistema fuera gratuito, para las siguientes versiones se establecerá una licencia Freemium. Además, se ha identificado el mercado objetivo donde se han descrito dos ejemplos de utilización de la herramienta.

7. Evaluación y Resultados

En este capítulo se abordan temas sobre la evaluación y la usabilidad. Además, se comentarán los resultados obtenidos en las evaluaciones de las distintas funcionalidades.

Se usaron diferentes métodos para la evaluación de la aplicación mediante los cuales se corrobora el correcto funcionamiento y la calidad del software desarrollado.

7.0.1. Pruebas Unitarias

Mediante los módulos JavaScript Karma[61], Jasmine[62] y Gulp[63] se automatizan las pruebas unitarias empleando la práctica TDD (ver sección 3.4).

Utilizando el comando `gulp tdd` comienza el proceso donde Gulp se queda observando cualquier cambio en el código y al encontrar alguno automáticamente ejecuta el repertorio de pruebas (ver figura 23 y 24).

```
  beforeEach(inject(function(_$rootScope_,_$controller_, _$state_){
    state = _$state_
    scope = _$rootScope_.$new()
    // Injectamos el controlador
    AuthController = _$controller_('AuthController', {
      $scope: scope
    });
  }));

describe('Login', function() {
  it('Fail login', function () {
    AuthController.user = {
      email: '',
      password: ''
    };
    AuthController.login().then(function(){
      expect(state.is('dashboard_open')).toBe(false);
    })
  });

  it('Success login', function () {
    AuthController.user = {
      email: 'gplsi@ua.es',
      password: 'gplsi'
    };
    AuthController.login().then(function(auth){
      console.log(auth)
      expect(state.is('dashboard_open')).toBe(true);
```

Figura 23. Pruebas unitarias con Karma y Jasmine.

[61] https://karma-runner.github.io/1.0/index.html
[62] http://jasmine.github.io/
[63] http://gulpjs.com/

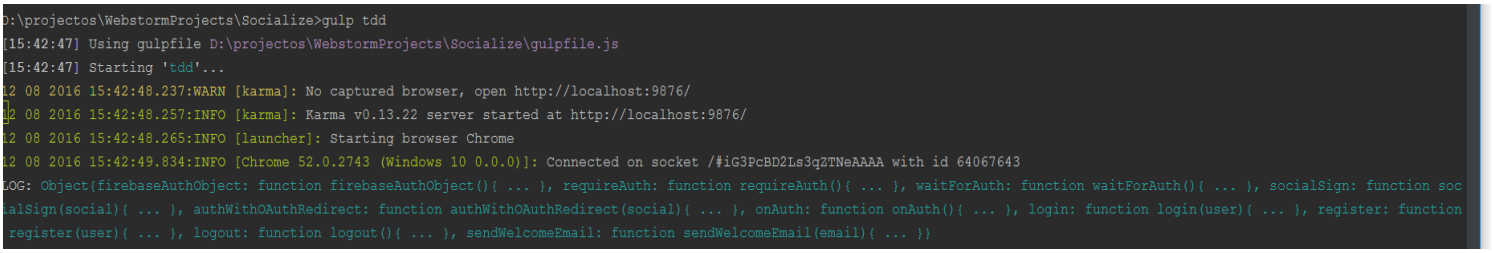

Figura 24. Automatización de pruebas unitarias con Gulp.

El análisis de los resultados y beneficios obtenidos gracias a las pruebas unitarias automatizadas empleando la práctica TDD se verá en la sección 7.1.

7.0.2. Pruebas de Cobertura

La cobertura es la cantidad de código (medida porcentualmente) que está siendo cubierto por las pruebas. Si se tiene una alta cobertura, significa que gran parte del código está siendo probado y por consiguiente se tiene una certeza sobre el correcto funcionamiento de la aplicación.

El módulo karma-coverage [40] integrado en las evaluaciones de la aplicación permite obtener mediciones sobre la cobertura de las pruebas realizadas.

La figura 25 es el resultado de una prueba de cobertura en Socialize en la mitad del desarrollo del mismo. Se listan todas las carpetas que contienen ficheros con extensión .js y, para cada una, se realiza una medición de cantidad y porcentaje de declaraciones (statements), ramas (branches), funciones (functions), lines (líneas de código) probadas.

/

31.11% Statements 196/630 **1.64%** Branches 2/122 **11.11%** Functions 22/198 **31.16%** Lines 196/629

File		Statements		Branches		Functions		Lines	
auth/scripts/		100%	2/2	100%	0/0	100%	1/1	100%	2/2
auth/scripts/controllers/		36.67%	11/30	0%	0/4	16.67%	2/12	36.67%	11/30
auth/scripts/services/		68.42%	13/19	100%	0/0	22.22%	2/9	68.42%	13/19
scripts/		68.75%	11/16	0%	0/2	50%	2/4	68.75%	11/16
scripts/controllers/		23.79%	49/206	0%	0/28	7.04%	5/71	23.79%	49/206
scripts/directives/		14.42%	15/104	0%	0/42	5%	1/20	14.42%	15/104
scripts/services/		37.55%	95/253	4.35%	2/46	11.11%	9/81	37.7%	95/252

Figura 25. Pruebas de cobertura con karma-coverage [40]

El análisis de los resultados y beneficios obtenidos gracias a las pruebas de cobertura se verá en la sección 7.1.

7.0.3. Pruebas de Usabilidad

Las pruebas de usabilidad implican una observación sistemática en condiciones controladas para determinar cómo las personas interactúan con un determinado producto. Además, pueden utilizarse conjuntamente como métodos para comprender mejor las motivaciones o percepciones de los usuarios, además de su interacción.

Se eligieron 15 usuarios para realizar las pruebas. El 20% de estos usuarios son estudiantes de Ingeniería Informática, un 60% son trabajadores profesionales en el área de finanzas y el 20% restante son usuarios poco implicados en temas relacionados con tecnologías.

La prueba consta de dos partes. La primera, después de una pequeña descripción de la aplicación se le indica al usuario que ejecute diferentes tareas. En la segunda parte, se entrega un formulario con preguntas relacionadas con la usabilidad de la aplicación.

El análisis de los resultados y beneficios obtenidos gracias a las pruebas de usabilidad se verá en la sección 7.1.

7.1. Resultados

A continuación se mostrarán las características y funcionalidades principales resultantes en el desarrollo de la primera versión de Socialize y evidencia la consecución de los requisitos funcionales establecida. Además, se mostrarán los resultados y beneficios de la evaluación con los tipos de pruebas realizados.

Funcionalidades principales:

- Autenticación a través de distintos medios (emails, redes sociales).
- Crear, modificar, listar y eliminar visualizaciones.
- Crear, modificar, listar y eliminar cuadros de mando.
- Añadir, cambiar tamaños, cambiar posiciones y eliminar visualizaciones en cuadros de mando.
- Compartir cuadros de mando a través de un enlace web.
- Opción a la incrustación de cuadros de mando en aplicaciones de terceros mediante un *iframe*.

Características principales:

- Diseño responsive (ver Figuras 33 y 34)
- Sistema escalable gracias a las bondades que ofrece Firebase (ver sección 4).
- Disponibilidad todos los días del año las 24 horas.

Resultados y beneficios de la evaluación:

- **Pruebas Unitarias**: La realización constante de pruebas en busca de errores en el sistema mediante la automatización de las mismas arroja como resultado un código con buena calidad. La solución de los problemas se realiza en etapas tempranas y no se corre el riesgo de encontrar graves errores que impliquen un cambio sustancial una vez liberado el producto.
- **Pruebas de Cobertura**: Este tipo de pruebas tiene como principal beneficio dar a conocer que cantidad del código se ha probado. De esta forma los desarrolladores pueden conocer qué partes del código faltan por probar y qué partes deben probarse con más profundidad. Los resultados de las últimas pruebas de coberturas realizadas son de un 70% de declaraciones, 50% de ramas, 60 % de funciones, 70% de líneas de código probadas.

- **Pruebas de Usabilidad:** Las pruebas de usabilidad han arrojado buenos resultados. De manera global se comprueba que la mayoría de usuarios encontró pocos problemas para ejecutar las principales funcionalidades de la aplicación. Se tiene en cuenta para cada caso de pruebas el porcentaje de usuarios que consideraron algunas características algo o muy complicadas de entender para mejorar la usabilidad del sistema. Las consideraciones de los usuarios son valoradas para incluir mejoras en siguientes versiones del software (ver sección 8.2 y 12.2).

A continuación se muestra un esquema del sistema:

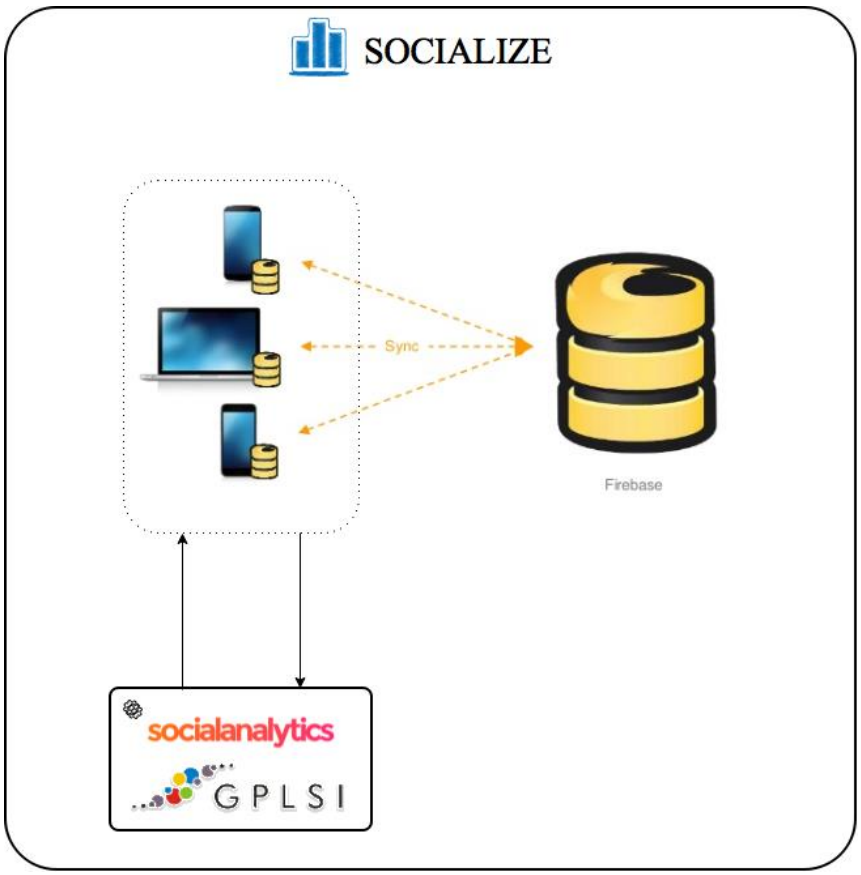

Figura 26. Esquema del sistema.

7.2. Conclusiones Parciales

A lo largo de este capítulo se ha evaluado el sistema desarrollado. Además, se muestran los resultados obtenidos gracias a las pruebas realizadas y se comentan los beneficios que tiene la constante evaluación del sistema.

8. Conclusiones generales y Trabajos futuros

Socialize se define como un sistema multiplataforma destinado al análisis inteligente del negocio. Incorpora cuadros de mandos con visualizaciones acopladas a servicios de minería de datos que están basados en el procesamiento de lenguaje natural del GPLSI y obtienen estadísticas asociadas a la polaridad sentimental y la intensidad de la misma. Estas visualizaciones ofrecen la posibilidad de combinar y representar la información de múltiples formas. A su vez, los cuadros de mando pueden ser sindicados para la reemisión de contenidos en aplicaciones de terceros. De esta forma se logra dar **solución al objetivo general**.

Para llegar a la solución del objetivo general fue necesaria la **consecución de los objetivos específicos** a continuación detallada.

Se ha realizado un análisis minucioso del Estado del Arte, lo cual ha servido para conocer las tecnologías punteras relacionadas con este trabajo. De cada tecnología y/o herramienta estudiada se han especificado los puntos positivos y negativos. Por tanto, se concluye que ninguna de las tecnologías y/o herramientas analizadas resuelve nuestro problema científico, más bien cada una por separado aporta una pequeña parte de la solución y han servido de punto de partida.

Se han estudiado las metodologías existentes con el objetivo de escoger la adecuada para guiar el desarrollo del sistema. En el estudio realizado se han separado las metodologías ágiles y las tradicionales para compararlas entre sí. En la fase de selección se concluye que las metodologías ágiles son las más adecuadas dadas las características del sistema y se decidió combinar diferentes prácticas y/o métodos de Scrum, XP, Kanban y UML.

Se han estudiado y seleccionado herramientas y tecnologías de desarrollo modernas, útiles para el rápido desarrollo y la obtención de calidad del software. Las herramientas y tecnologías que fueron escogidas tienen un gran alcance, son muy populares dentro de la comunidad de desarrollo y marcan una tendencia a seguir creciendo. Además, con la selección hecha se dieron los primeros pasos para la consecución de un sistema escalable, seguro, de alta disponibilidad y fuerte fiabilidad.

Se ha realizado el análisis de requisitos. Los requisitos funcionales y no funcionales se han obtenido gracias a las historias de usuario que surgieron en las constantes reuniones con el cliente.

Se ha estudiado, seleccionado e implementado la arquitectura del software. La selección de una arquitectura de 3 capas y dos niveles fue fundamental y sirvió para tomar decisiones de diseño que se traducen en atributos de calidad como Rendimiento, Seguridad, Coste de hacer un cambio, Confiabilidad, Escalabilidad, Usabilidad, entre otros.

Se ha comentado la planificación realizada en el desarrollo del sistema gracias a la guía de las metodologías ágiles elegidas (más detalles en la sección 10). De esta forma se han establecido las tareas prioritarias en forma de iteraciones y se ha alcanzado el objetivo de incorporar visualizaciones que utilizan los datos que se obtienen de los servicios de procesamiento de lenguaje natural desarrollados por el GPLSI.

Se ha estudiado y establecido un modelo de negocio adecuado al sistema desarrollado. En la primera versión se decide que el sistema sea gratuito, para las siguientes versiones se establecerá una licencia Freemium. Además, se ha identificado el mercado objetivo donde se han descrito dos ejemplos de utilización de la herramienta.

Por último, se ha evaluado el sistema desarrollado. Además, se han mostrado los resultados obtenidos gracias a las pruebas realizadas y se detallan los beneficios que tiene la constante evaluación del sistema.

Gracias a la disciplina sobre la metodología planteada, la incorporación del cliente como parte del proyecto, las evaluaciones y pruebas sistemáticas y el trabajo en equipo se han conseguido satisfactoriamente el objetivo general y los objetivos específicos planteados al inicio del proyecto.

8.1. Trabajos Futuros

Socialize es un proyecto ambicioso que ha dado sus primeros pasos. Se han cumplidos los objetivos planteados al inicio pero para versiones futuras se quiere continuar ampliando, mejorando y escalando este producto que goza de una potencialidad inherente.

Algunas ampliaciones y mejoras se listan a continuación:

- Incluir módulo de licencias de pago.
- Mejora de la experiencia de usuario con un acercamiento al usuario final basado en pruebas de usabilidad.
- Implementación de un tutorial interactivo para el usuario.
- Incorporación de otras fuentes de servicios de minería de datos, por ejemplo: Social Observer, Meaning Cloud, entre otros.
- Integración de visualizaciones basadas en la geolocalización.
- Captar nuevos usuarios a través de un Landing Page[64].
- Temporizador automático para obtener datos en tiempo real.

[64] En español Página de Aterrizaje. El objetivo principal es convencer al usuario de obtener un producto o servicio determinado.

9. Referencias

1. Cunningham, 2001. Manifiesto por el Desarrollo Ágil de Software [en línea] <http://www.agilemanifesto.org/iso/es/> [Consulta: 03 julio 2016].
2. Firebase, 2016. Firebase - BaaS [en línea]<https://firebase.google.com/docs/> [Consulta: 03 julio 2016]
3. Roberth, 2008. Metodologías Ágiles vs Metodologías Tradicionales [en línea] <https://adonisnet.files.wordpress.com/2008/06/articulo-metodologia-de-sw-formato.doc> [Consulta: 08/07/2016]
4. Brito, 2009. Selección de metodologías de desarrollo para aplicaciones web en la Facultad de Informática de la Universidad de Cienfuegos <http://www.eumed.net/libros-gratis/2009c/584/584.zip> [Consulta: 08/07/2016]
5. Rational, 1998. Best Practices for Software Development Teams. <https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf> [Consulta: 08/07/2016]
6. Microsoft, 2016. Descripción general de Microsoft Solutions Framework [en línea] <https://msdn.microsoft.com/es-es/library/jj161047(v=vs.120).aspx> [Consulta: 09/07/2016]
7. Boehm, 1986. A Spiral Model of Software Development and Enhancement, ACM SIGSOFT Software Engineering Notes, ACM <http://dl.acm.org/citation.cfm?doid=12944.12948> [Consulta: 09/07/2016]
8. EcuRed, 2016. Iconix [en línea] <http://www.ecured.cu/index.php/ICONIX#Car.C3.A1cter.C3.ADsticas_de_Iconix> [Consulta: 09/07/2016]
9. ISSI, 2003. Metodologías Ágiles en el Desarrollo de Software [en línea] <http://issi.dsic.upv.es/archives/f-1069167248521/actas.pdf> [Consulta: 09/07/2016]
10. Microsoft, 2016. Microsoft Operations Framework [en línea] <https://technet.microsoft.com/es-es/library/bb232042(v=exchg.80).aspx> [Consulta: 10/07/2016]
11. Beck, K., 1999. Extreme Programming Explained. Embrace Change, Pearson Education. Traducido al español como: Una explicación de la programación extrema. Aceptar el cambio, Addison Wesley, 2000

12. Gutiérrez, 2014. ¿Para qué sirve el Scrum en la Metodología Ágil? [en línea] <http://www.i2btech.com/blog-i2b/tech-deployment/para-que-sirve-el-scrum-en-la-metogologia-agil/> [Consulta: 11/07/2016]
13. D. J. Anderson, 2010. Kanban: Successful Evolutionary Change for Your Technology Business. Sequim, WA: Blue Hole Press, (2010).
14. Ibarra, 2014. Metodología ágil scrumban en el proceso de desarrollo y mantenimiento de software de la norma moprosoft. <http://www.rcs.cic.ipn.mx/2014_79/Metodologia%20agil%20Scrumban%20en%20el%20proceso%20de%20desarrollo%20y%20mantenimiento%20de%20software%20de%20la%20norma.pdf> [Consulta: 11/07/2016]
15. Kniberg & Skarin, 2010. Kanban y Scrum – obteniendo lo mejor de ambos <http://www.proyectalis.com/documentos/KanbanVsScrum_Castellano_FINAL-printed.pdf> [Consulta: 12/07/2016]
16. Banon, 2013. The Future of Compass & ElasticSearch.<https://web.archive.org/web/20130827121405/http://www.kimchy.org/the_future_of_compass/>[Consulta: 21/07/2016].
17. Elastic, 2016. Web Oficial <https://www.elastic.co>. [Consulta: 21/07/2016].
18. Yigal, 2016. Grafana vs Kibana<http://logz.io/blog/grafana-vs-kibana/> [Consulta: 21/07/2016].
19. OpenTSDB, 2016. Web Oficial <http://opentsdb.net/>. [Consulta: 21/07/2016].
20. Influx Data, 2016. Web Oficial <https://influxdata.com/>. [Consulta: 21/07/2016].
21. Grafana, 2016. Web Oficial <http://grafana.org/>. [Consulta: 21/07/2016].
22. Graphite, 2016. Web Oficial <http://graphiteapp.org/>. [Consulta: 21/07/2016].
23. GPLSI, 2016. Social Analytics. <https://gplsi.dlsi.ua.es/gplsi13/es/node/251>. [Consulta: 21/07/2016].
24. Pang & Lee, 2008. Opinion mining and sentiment analysis. <http://www.cs.cornell.edu/home/llee/omsa/omsa.pdf>. [Consulta: 21/07/2016].
25. Navarro, 2007. Rest vs WebServices. <http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>. [Consulta: 21/07/2016].
26. Social Analytics, 2016. Web Oficial <http://socialanalytics.gplsi.es/>. [Consulta: 21/07/2016].

27. Genbeta, 2016. Angularjs. Módulos y arquitectura. <http://www.genbetadev.com/desarrollo-web/angular-js-modulos-y-arquitectura>. [Consulta: 21/07/2016].
28. Gechev, 2016. AngularJS in Patterns <http://blog.mgechev.com/2014/07/05/angularjs-in-patterns-part-3/>. [Consulta: 21/07/2016].
29. UX HTML, 2016. Responsive Design: Patrones de Diseño. <http://uxhtml.blogspot.com.es/2013/06/responsive-design-patrones-de-diseno.html>. [Consulta: 21/07/2016].
30. Github, 2016. Any Theme, código fuente. <https://github.com/start-angular/ani-theme>. [Consulta: 21/07/2016].
31. Github, 2016. Angular Gridster, código fuente. <https://github.com/ManifestWebDesign/angular-gridster>. [Consulta: 21/07/2016].
32. Amcharts, 2016. Web Oficial <https://www.amcharts.com/>. [Consulta: 21/07/2016].
33. GPLSI, 2016. Social Observer. <https://gplsi.dlsi.ua.es/gplsi11/content/gplsi-social-observer>. [Consulta: 21/07/2016].
34. ITSZO, 2016. Curso Remedial de Estructuras Algorítmicas, técnicas de diseño. <https://programacionfacil.wikispaces.com/file/view/Parte_4.pdf>. [Consulta: 21/07/2016].
35. Highcharts, 2016. Web Oficial <http://www.highcharts.com/>. [Consulta: 21/07/2016].
36. Trello, 2016. Tutorial de trello. <https://trello.com/c/RZExuHxu/6-que-es-trello-y-como-se-usa>. [Consulta: 21/07/2016].
37. Firebase, 2015. Angularfire. <https://www.firebase.com/docs/web/libraries/angular/guide/intro-to-angularfire.html>. [Consulta: 21/07/2016].
38. Asociación Mexicana Para el Procesamiento de Lenguaje Natural, 2016. Procesamiento de Lenguaje Natural. <http://www.cicling.org/ampln/NLP.htm>. [Consulta: 21/07/2016].
39. W3C, 2016. Cross-Origin Resource Sharing <http://www.w3.org/TR/cors/>. [Consulta: 21/07/2016].
40. NPM, 2016. Karma-Coverage. <https://www.npmjs.com/package/karma-coverage>. [Consulta: 21/07/2016].

41. GPLSI, 1998. Procesamiento del Lenguaje Natural. <http://gplsi.dlsi.ua.es/gplsi/areasf.htm> [Consulta: 21/07/2016].
42. López, 2016. Arquitectura de n capas y niveles. <http://www.academia.edu/10102692/Arquitectura_de_n_capas> [En línea]

10. Anexos

En este apartado se aborda con más profundidad la consecución de objetivos a través de las iteraciones en el desarrollo del proyecto. Luego, se muestran las pruebas de usabilidad realizadas. Por último, se anexan el Manual de Usuario en donde se describen las funcionalidades del sistema en forma gráfica y el Manual de Desarrollador para la continuación del proyecto.

10.1. Principales Iteraciones

A continuación se detallaran los Sprints más relevantes en el desarrollo de Socialize.

10.1.1. Sprint 9

Para este Sprint se decide centrar los esfuerzos el control de usuarios y la plantilla de la aplicación.

El Sprint Backlog se conforma por:

- Elegir nombre para el proyecto.
- Elegir y adaptar plantilla web para el proyecto.
- Crear registro y logueo básico. Incorporar persistencia de usuarios con Firebase (ver sección 4.3).
- Incorporar opción de logueo con Twitter y Facebook

Al final se revisa lo que se ha hecho:

- Se elige el nombre Socialize, su traducción es socializar. Se escogió este nombre porque el proyecto no tendría sentido sin la socialización o la interacción de las personas a través de las redes sociales. Se escogió en inglés para que la aplicación tuviera un alcance universal.
- Se elige la plantilla web Ani Theme Edition[65] la cual es adaptada al proyecto con la incorporación de Bower como manejador de paquetes, Gulp para automatización de tareas y el patrón SPA para la incorporación de AngularJS.
- Se incorpora la opción de registro y logueo básico con persistencia a través de Firebase (ver sección 4.3).

[65] Adaptada a AngularJS esta plantilla se puede encontrar como código abierto en Github [30]

En la retrospectiva del Sprint se comenta la necesidad de un cambio de alcance de la funcionalidad de logueo con Twitter y Facebook.

10.1.2. Sprint 10

Tareas correspondientes al Sprint:

- Incorporar opción de login con Twitter y Facebook.
- Elección de dos servicios del Social Observer a utilizar.
- Investigación de módulos de AngularJS para la creación de un cuadro de mandos.
- Automatización de la construcción y despliegue del proyecto.
- Investigación sobre posibles módulos de gráficos para la integración de visualizaciones.

Al finalizar el Sprint:

- Se finaliza el módulo de Autenticación (ver Figura 27)
- Se agrega opción de login con usuario de Twitter y Facebook.

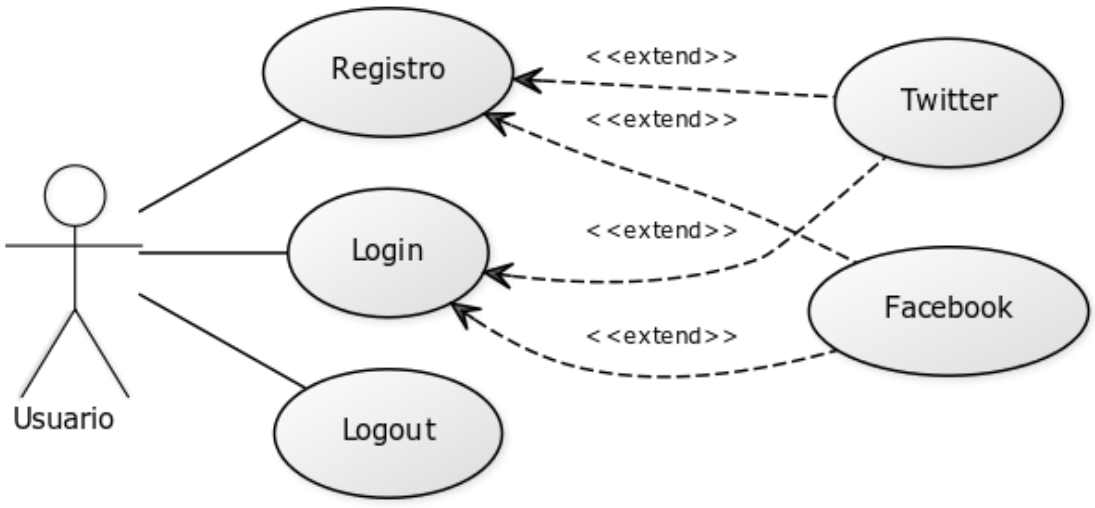

Figura 27. Diagrama de caso de uso de la Autenticación

- Se eligen los servicios de detección de polaridad de sentimientos y la identificación de emociones sobre entidades para desarrollar los primeros módulos sobre ellos.
- Se elige el módulo Angular Gridster [31] como base para crear cuadros de mandos.
- Utilizando Gulp se automatiza la construcción del proyecto. Se realizan pruebas de estructuración de código y se minifica[66] la totalidad del código. Luego se copian los ficheros a la carpeta de distribución.
- El despliegue se hace a través de la consola de Firebase (ver sección 4.3), indicando en la configuración que la carpeta DIST contiene la distribución del proyecto a desplegar. Con solo una línea de comandos se despliega el proyecto a un contenedor web de Firebase (ver sección 4.3), desde donde versionar los despliegues.
- Se investiga Amchart [32] para utilizarlo como módulo de visualizaciones.

En la retrospectiva del Sprint se comenta sobre realizar un cambio de alcance para la tarea de investigación de módulos de gráficos. Se decide desistir de Amchart ya que encuentran inconvenientes a la hora de integrar los módulos de gráficos con el cuadro de mandos Angular Gridster.

10.1.3. Sprint 13

Tareas correspondientes al Sprint:

- Crear la lógica[67] de cuadros de mandos.
- Crear la lógica de visualizaciones.

Al finalizar el Sprint:

- Se logra finalizar la lógica de cuadros de mandos, faltando pequeños detalles con respecto a la forma de visualizar los mismos.
- La lógica de visualizaciones queda a un 75%, faltando la opción de editar las mismas. (ver Figura 28 y 29).

[66] Minificación (también minimización o reducción al mínimo), es el proceso de eliminación de todos los caracteres innecesarios del código fuente sin cambiar su funcionalidad.
[67] Crear, modificar, listar y eliminar objeto

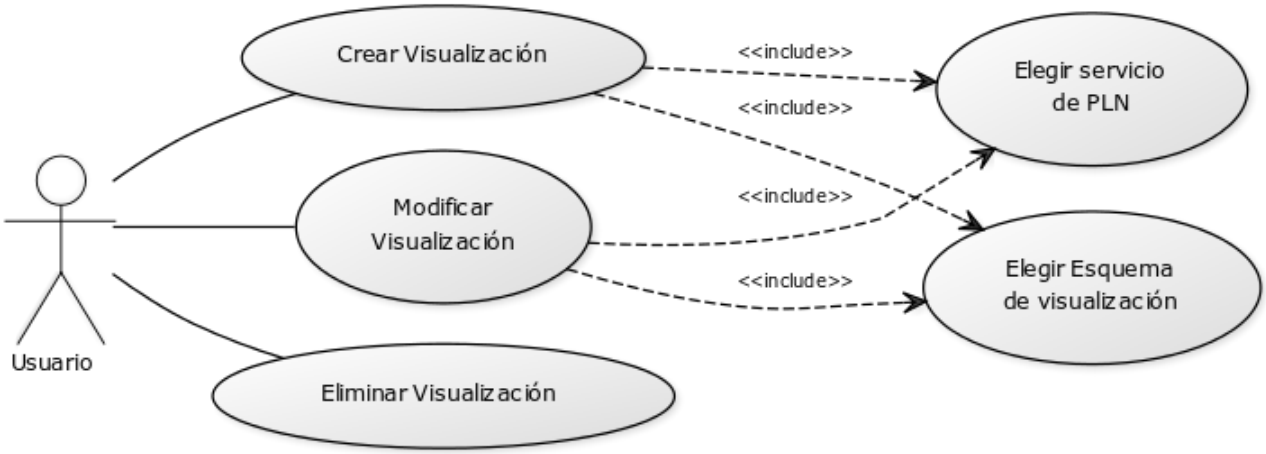

Figura 28. Diagrama de caso de uso del módulo Visualización

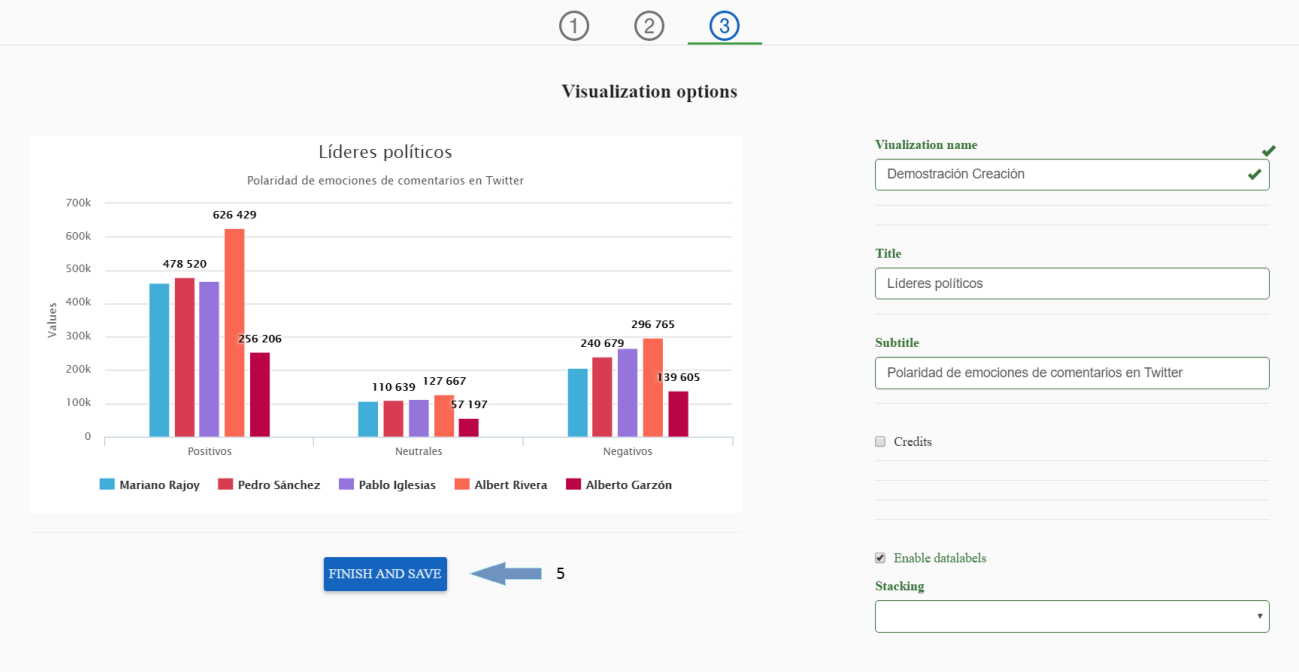

Figura 29. Crear una visualización personalizada

En la retrospectiva del Sprint se comenta la dificultad para integrar al cuadro de mandos el módulo de gráficos escogido[68] . Además se realiza un cambio de alcance respecto a la tarea de lógica de visualizaciones para terminarla en el siguiente Sprint.

[68] Highcharts, módulo de gráficos para JavaScript [35]

10.1.4. Sprint 15

Tareas correspondientes al Sprint:

- Integrar visualizaciones con cuadros de mando.
- Migrar Firebase (ver sección 4.3) y AngularFire[69] a la versión 3

Al finalizar el Sprint:

- La integración de las visualizaciones con el cuadro de mandos (ver figura 30 y 31) avanza lentamente. Se ha realizado un 50% de la tarea, con lo cual se realiza un cambio de alcance para darle fin en el próximo Sprint.

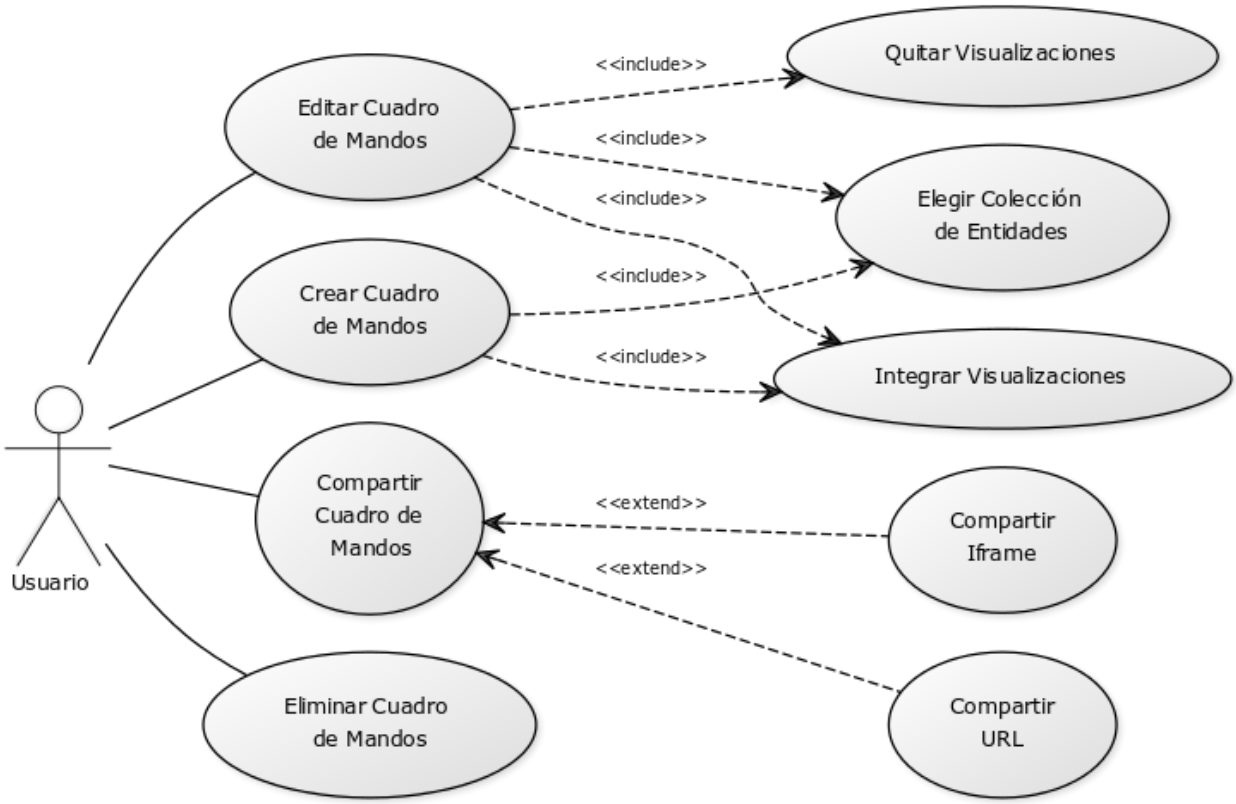

Figura 30. Diagrama de caso de uso del módulo cuadros de mando

[69] Librería de código abierto que integra AngularJS con Firebase [37]

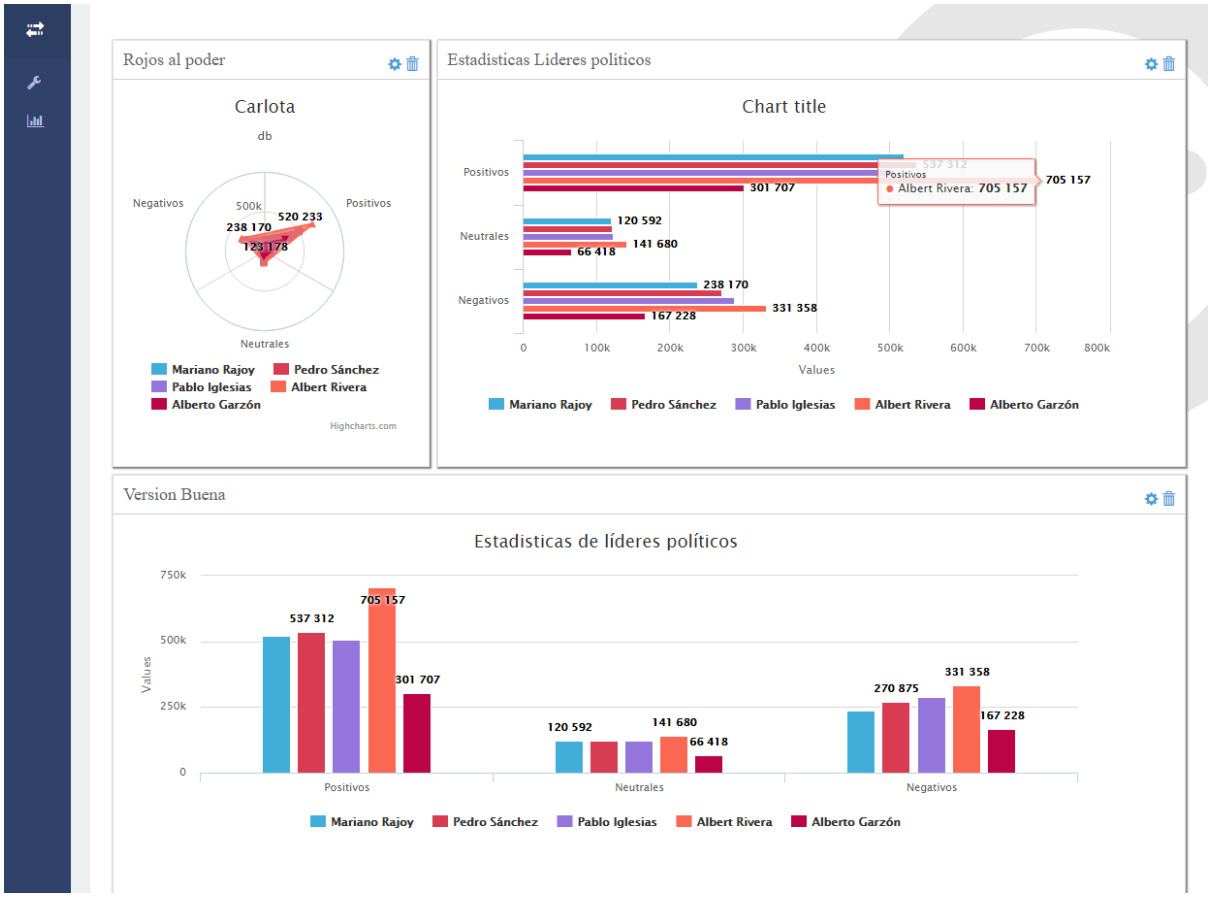

Figura 31. Integración de visualizaciones en cuadros de mando

- La migración a Firebase 3 ha sido un éxito, dando la posibilidad de explotar sus ventajas.

En el Sprint Review se comenta la complejidad de integrar y adaptar las visualizaciones con el cuadro de mandos.

10.1.5. Sprint 20

Tareas correspondientes al Sprint,

- Sincronizar e integrar los servicios de PLN del Social Analytics
- Ampliar funcionalidades, compartir cuadros de mando para su incrustación en aplicaciones de terceros.
- Mejorar el diseño para móviles

Al finalizar el Sprint,

- Se sincronizan correctamente los servicios del Social Analytics.
- Se crea la funcionalidad de compartir el cuadro de mandos mediante la opción de copia de url o mediante un *iframe* para la incrustación en aplicaciones de terceros (ver figura 32).
- Diseño responsive para mejorar la visualización en móviles (ver figura 33 y 34)

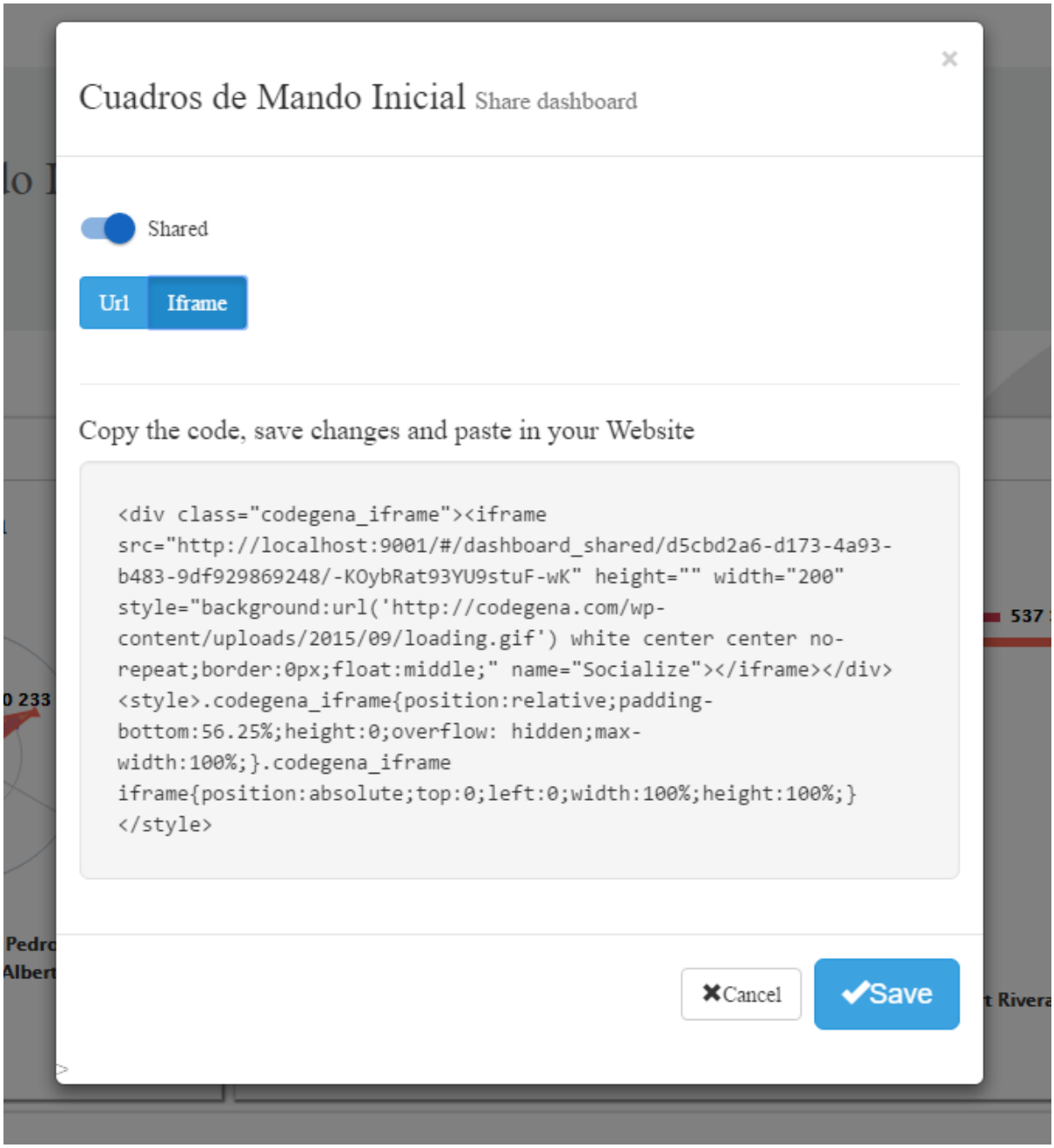

Figura 32. Compartir cuadro de mandos

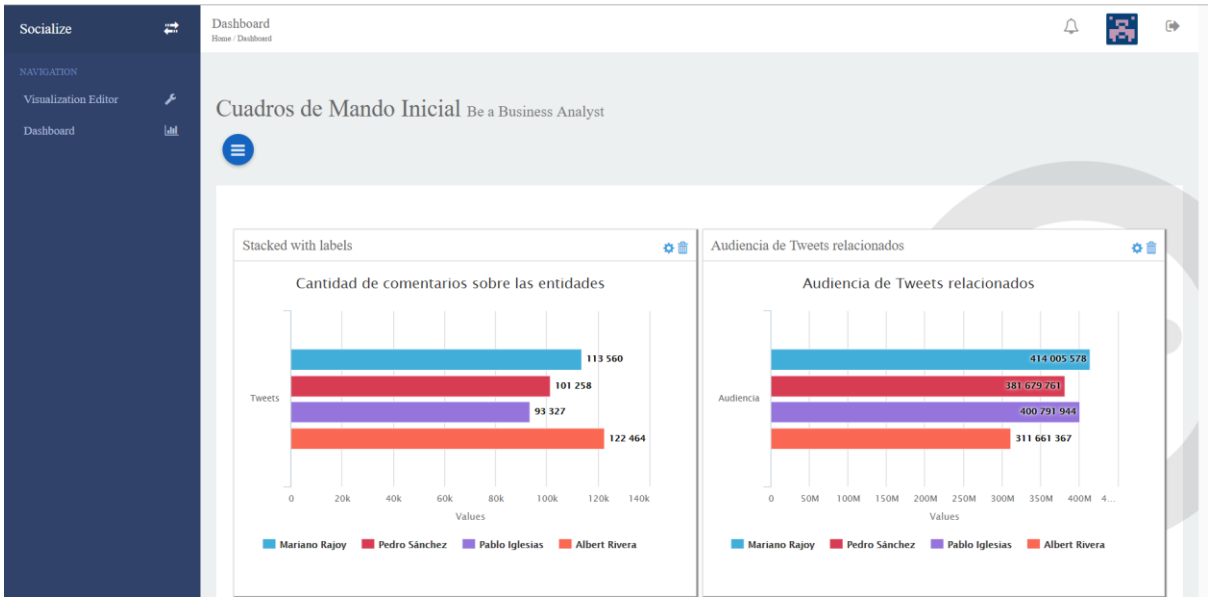

Figura 33. Diseño responsive para ordenador

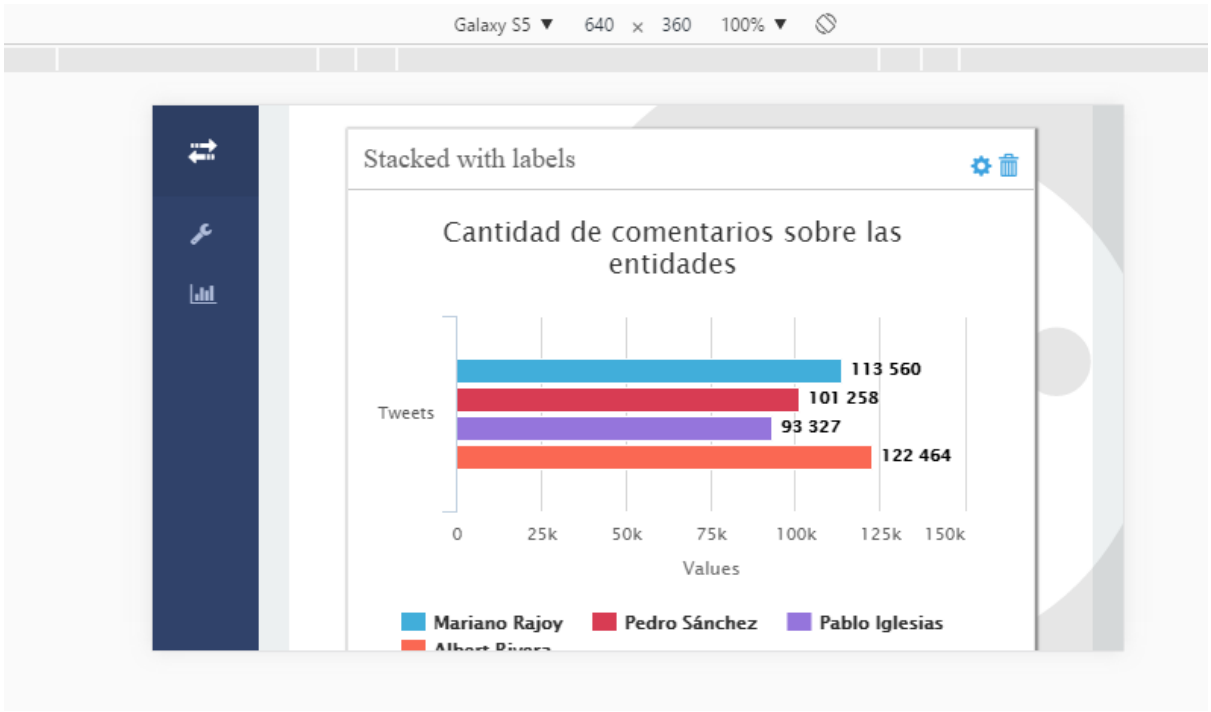

Figura 34. Diseño responsive para móviles

En el Sprint Review se comenta sobre la ventaja de haber creado mocks de los servicios de PLN del Social Analytics con datos estáticos, integrar los servicios reales fue rápido y sencillo.

10.2. Pruebas de Usabilidad

A continuación se mostrarán las estadísticas relacionadas con las respuestas de los usuarios en las pruebas de usabilidad.

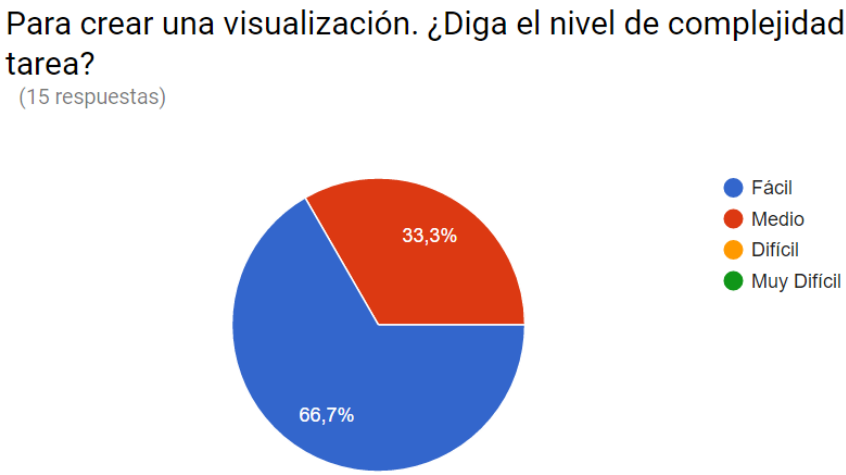

Figura 35. Pregunta 1. Pruebas de Usabilidad

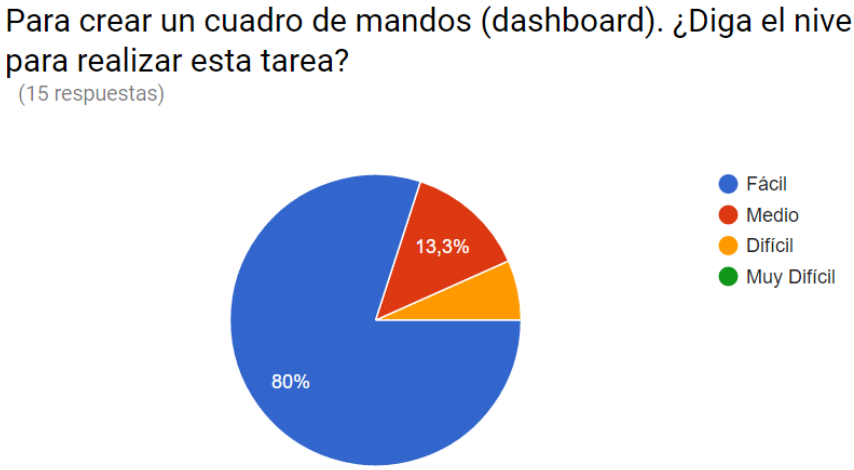

Figura 36. Pregunta 2. Pruebas de Usabilidad

Figura 37. Pregunta 3. Pruebas de Usabilidad

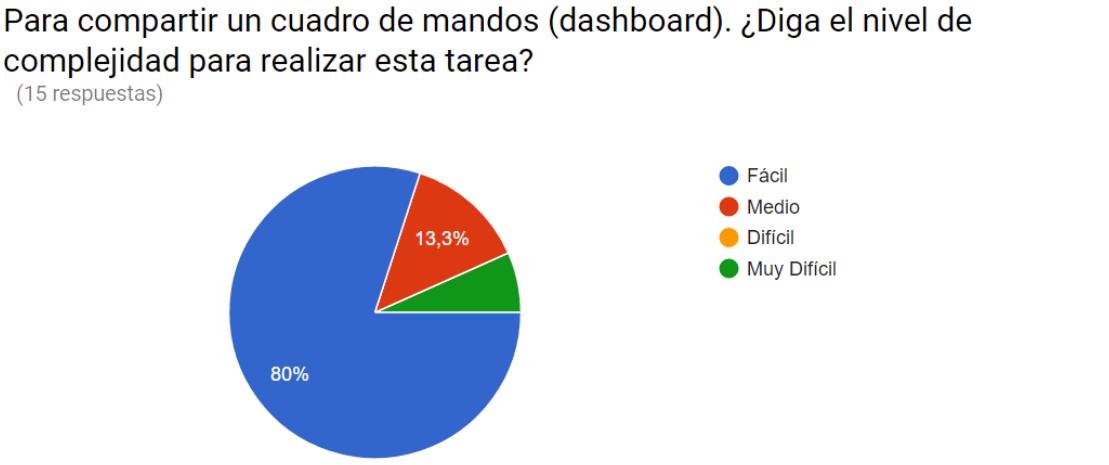

Figura 38. Pregunta 4. Pruebas de Usabilidad

Figura 39. Pregunta 5. Pruebas de Usabilidad

Figura 40. Pregunta 6. Pruebas de Usabilidad

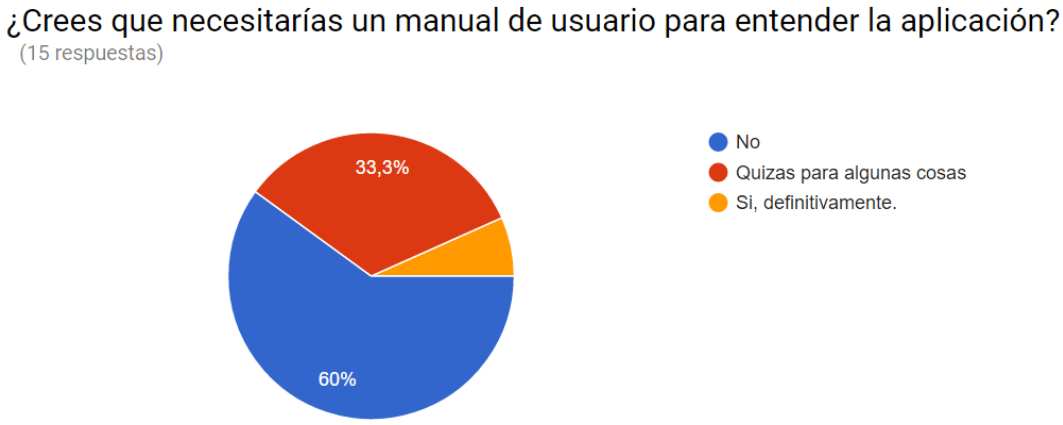

Figura 41. Pregunta 7. Pruebas de Usabilidad

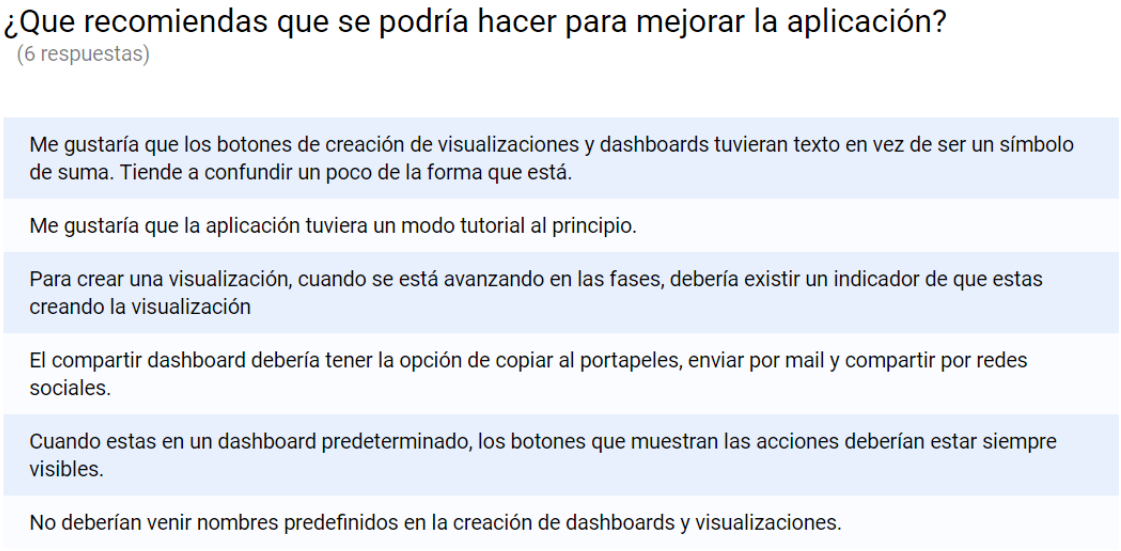

Figura 42. Pregunta 8. Pruebas de Usabilidad