Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de un sistema multiplataforma para la generación automática de infografías

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Anna Llorens Roig

Tutores: Eliseo Marzal Calatayud

Jorge Hortelano Otero

Julio de 2017

Resumen

El objetivo principal de este Trabajo de Fin de Grado es mejorar la experiencia de los usuarios a la hora de presentar datos en sus aplicaciones web y móvil. Los datos que deban ser presentados al usuario se mostrarán mediante una representación que mezcla datos con elementos visuales llamada infografía. Para ello se ha desarrollado un sistema de generación de infografías que se integrará en diversas plataformas. Este sistema permite diseñar infografías personalizadas para cada usuario, que puedan ser accesibles a otras aplicaciones utilizando un formato de imagen.

Palabras clave: infografía, SVG, JSON, JavaScript, Node, librería, servidor web, API.

Abstract

The main objective of this Final Degree Project is to improve the user experience when presenting data in their web and mobile applications. The data that should be presented to the user must mix texts and visual elements. This is called infographic. For this scope, an infographic generation system has been developed and integrated into several platforms. This system allows you to design customized infographics for each user, get them through requests to a web server and visualized in different image formats.

Keywords: infographic, SVG, JSON, JavaScript, Node, library, web server, API

Agradecimientos

En primer lugar, agradecer el apoyo incondicional mostrado por mis padres durante todos mis estudios, gracias a ellos estoy ahora mismo aquí. A Jorge, mi tutor de empresa, por todo lo que me ha enseñado sobre el desarrollo de software, su paciencia, apoyo y amistad. A mi tutor de la UPV Eliseo por guiarme y aconsejarme durante el desarrollo de este TFG. Y por último a mis compañeras de piso Garazi y Laura por el interés mostrado durante toda la evolución de la presente memoria.

Anna

Índice

Índice de Figuras

1 Introducción

Hay muchos autores que afirman que el 90% de la información que percibimos en nuestro cerebro es visual [1]. Un niño, por ejemplo, es capaz de asociar un gráfico y su significado mucho antes de comprender el idioma, además nuestro cerebro procesa los elementos visuales como las gráficas, tablas o dibujos mucho más rápido que cuando lee la misma idea en un texto escrito.

Esto es especialmente importante cuando queremos mostrar una gran cantidad de datos a un receptor. Algo cada día más común en aplicaciones web o móviles. La forma de mostrar estos datos toma gran importancia ya que esta debe estar organizada y ser fácilmente entendible para el usuario final.

En este proyecto se pretende mejorar la experiencia del usuario y para ello los datos, informes y resultados que requieran mostrarse en aplicaciones se presentarán mediante **infografías**.

Por último, destacar en ésta introducción que este trabajo ha sido realizado en colaboración con la empresa BiiT Sourcing Solutions S.L., y su desarrollo se ha llevado a cabo durante 6 meses de prácticas en el departamento de desarrollo de la empresa. Finalmente, esta librería está siendo utilizada en una aplicación desplegada en el hospital Orbis situado en la ciudad de Sittard-Geleen (Países Bajos).

1.1 ¿Qué es una infografía?

El término infografía deriva del acrónimo información + grafía. Es por tanto una representación simplificada de un conjunto de datos complejos, normalmente acompañado de componentes visuales que permitan una mayor compresión de la idea transmitida. Las infografías facilitan la comunicación y transmisión de información de un modo diferente al lenguaje escrito u oral. Es por esto que son más que útiles si buscamos es un formato visual para explicar información compleja [18].

1.1.1 Aplicaciones de una infografía

Actualmente, con el desarrollo de las tecnologías se ha incrementado su uso. Las infografías se utilizan en ambientes muy variados. En el ámbito **educativo** con infografías pedagógicas destinadas a la enseñanza o infografías científicas con enciclopedias y textos científicos ilustrados. En el ámbito **empresarial** ya sea para los manuales de instrucciones de los productos de las empresas o para mostrar datos de sus informes anuales de ventas, beneficios, etc. También podemos encontrar infografías en el ámbito

periodístico, ya que el periodismo es un canal para divulgar conocimiento y en el las infografías juegan un papel importante. Por último cabe destacar la importancia de las infografías en el ámbito **publicitario** en el que se utilizan para expresar más claramente los beneficios o características de un producto o servicio [18].

1.1.2 Características de una infografía

A continuación, para explicar las características principales en las que éstas se basan, tomaremos una infografía a modo de ejemplo diseñada por *Subcutaneo creative* [2] y analizaremos las partes más importantes.

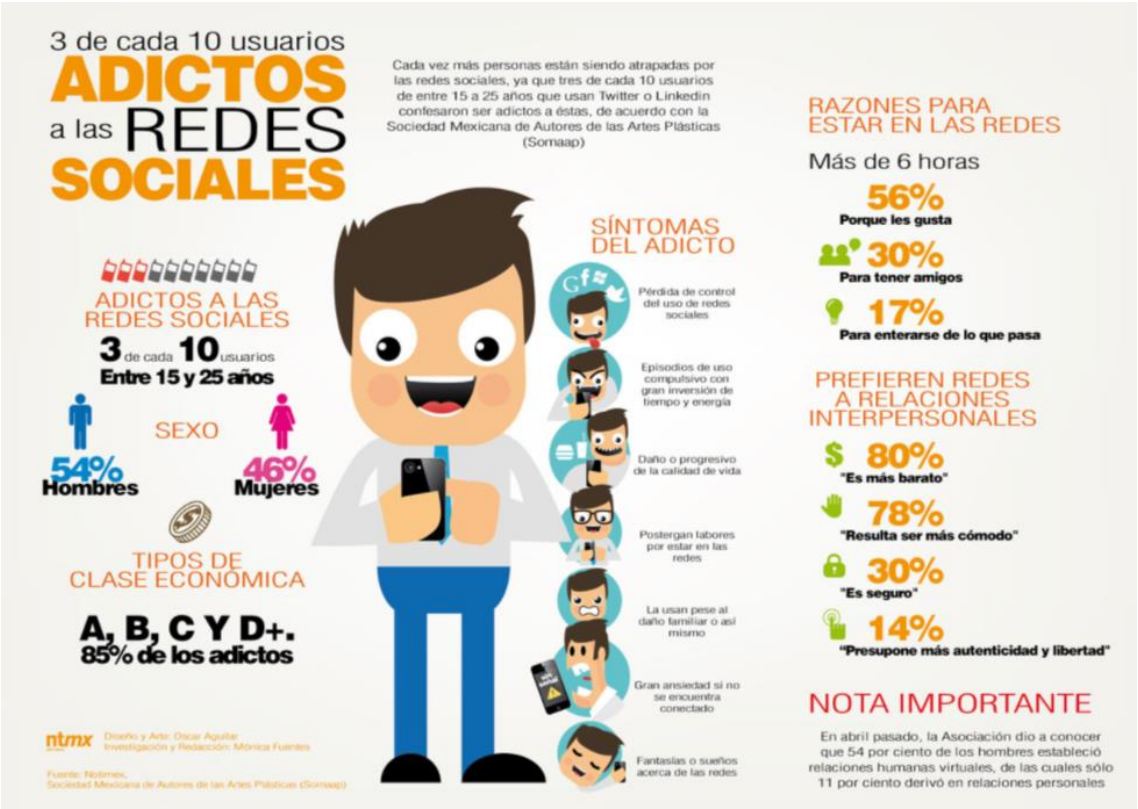

Figura 1: Ejemplo de infografía

Una de las figuras que más llama la atención en esta imagen es la **figura central**, se trata de un personaje con un móvil. El propósito de la imagen central es aproximar al lector a los elementos o ideas que se van a tratar en la infografía.

Otra de las características de las infografías es el **título.** Este debe resumir la información visual y textual que se presentará al receptor. En el ejemplo, el título "3 de cada 10 usuarios adictos a las redes sociales", presenta la infografía de manera directa, breve y expresa.

El **cuerpo** de las infografías contiene información visual y puede presentarse mediante gráficos, mapas, cuadros estadísticos, imágenes, etc. Como se puede ver en la imagen la infografía saca información de las mismas imágenes y las presenta en pequeños y breves textos que hacen su lectura más rápida y ágil. Como ejemplo, en la parte superior izquierda para explicar que 3 de cada 10 usuarios son adictos a las redes, hay 3 teléfonos de color rojo y los otros 7 de color gris. También vemos en la parte derecha como se ha presentado los datos estadísticos "razones para estar en las redes" y "prefieren redes a relaciones interpersonales" de forma visual.

Por último, en las infografías que traten con datos y estadísticas deben indicarse sus **fuentes** y **autor**. En el ejemplo, se ha incluido en la esquina inferior izquierda.

1.2 Objetivos

Una vez explicado el concepto de infografía y la importancia del mismo, procederemos a explicar los objetivos que se pretenden en este TFG.

Concretamente, en este proyecto se pretende definir una librería para la **obtención de infografías**, que posteriormente será integrada tanto en aplicaciones servidor para generar *Dashboards* [5], como en aplicaciones del lado del cliente para visualizar resultados o búsquedas.

Durante el desarrollo del proyecto han ido apareciendo objetivos derivados del objetivo principal que consisten en: dominar los lenguajes de programación utilizados en este tipo de aplicaciones y los *frameworks* que utilizan, comprender la estructura del Modelo de Objetos de Documento (DOM) y familiarizarse con metodologías ágiles de trabajo para la planificación, estructuración y control del proyecto.

2 Contexto

2.1 Contexto de la empresa

La implementación y desarrollo de este TFG se ha llevado a cabo en colaboración con la empresa BIIT Sourcing Solutions.

BIIT es una empresa con más de cuatro años de antigüedad que, a través de una variedad de ambientes funcionales, ofrece desarrollo, implementación y soporte personalizado de aplicaciones y servicios para clientes del sector médico privado. La empresa tiene sede en Valencia, pero su mercado está orientado a Holanda.

Los productos que ofrece se basan en su mayoría en "*Business Inteligente (BI)"* [19] . El objetivo del BI consiste en recabar información de distintas fuentes, procesar e informar las conclusiones con la finalidad de ayudar en la toma de decisiones, todo ello basado en técnicas informáticas.inteligencia_empr

Recientemente, la empresa está realizando el salto de aplicaciones para servidores, a aplicaciones multiplataforma, incluyendo especialmente los dispositivos móviles como plataforma cliente. Esto último condicionará la tecnología utilizada, así como los lenguajes de programación escogidos para el proyecto.

2.1.1 Problemática

Las aplicaciones web y móvil que ofrece la empresa están destinadas a la gestión de centros médicos y sus pacientes. Entre sus funciones más importantes para este proyecto destacaremos: la realización de test médicos, generación de estadísticas que muestran la evolución de los placientes y la creación de **informes médicos** para el paciente.

Un informe médico es un documento emitido por un profesional en el que se informa del estado de salud. En el informe se suelen incluir los datos del paciente, resumen de su historial clínico, la actividad asistencial prestada, el diagnóstico y las recomendaciones terapéuticas entre otros [25].

El problema de estos informes es que pese a ser uno de los documentos de mayor uso en el ámbito sanitario, los pacientes suelen tener dificultades para interpretar estos informes ya que contienen datos con los que no están acostumbrados a tratar. Podemos concluir que los informes actuales carecen de una interfaz atractiva y entendible para el usuario final, y es aquí por tanto en donde las infografías entrarán en juego.

2.1.2 Motivación

Se requiere desarrollar un software que gestione la generación de informes de una manera más visual para el usuario. Se decide utilizar para este propósito hacer uso de las infografías.

Ya hemos visto que los pacientes tienen dificultades para entender informes médicos por lo que las infografías deberán presentar la información de manera simple mediante gráficas, tablas, texto o diagramas que son algunos de los elementos hemos definido en las características de una infografía.

Otro de los problemas que se planteaba es la integración. La librería deberá poder integrarse en aplicaciones web como en aplicaciones móvil, es decir, un sistema multiplataforma que gestione la comunicación entre los distintos entornos y facilite la obtención de infografías personalizadas para cada usuario.

2.2 Estado del arte

Diseñar una infografía no es solamente poner datos en un gráfico, se debe sintetizar la información, interpretar los datos de una manera diferente y aportar valor al usuario con esa información [36].

En esta sección realizaremos un análisis de las herramientas existentes que facilitan el diseño de infografías diferenciando dos tipos: editores online y librerías para el tratamiento gráfico. A continuación, explicaremos cada una de ellas indicando también porque no serán válidas para nuestro propósito.

2.2.1 Editores online

Existen multitud de editores online de tipo *Drag and drop* [12] con interfaz gráfica que facilitan la obtención de infografías.

Estos editores ayudan a sus usuarios a la hora de diseñar infografías. Generalmente proporcionan plantillas prediseñadas que los usuarios pueden editar, cuentan con multitud de librerías con imágenes de distintas temáticas, proporcionan herramientas para añadir y editar textos, permiten que los usuarios puedan subir sus propias imágenes, facilitan la generación de gráficos, etc.

Algunos ejemplos de editores online: *Easel.ly, Infogram, Canva, Picktochart, etc*.

En la Figura 2 se muestra uno de los editores online más conocidos *Easel.ly*. Éste cuenta con recursos gratuitos, diferentes estilos de texto, herramientas para generar gráficos muy visuales y una multitud de plantillas diseñadas por la comunidad.

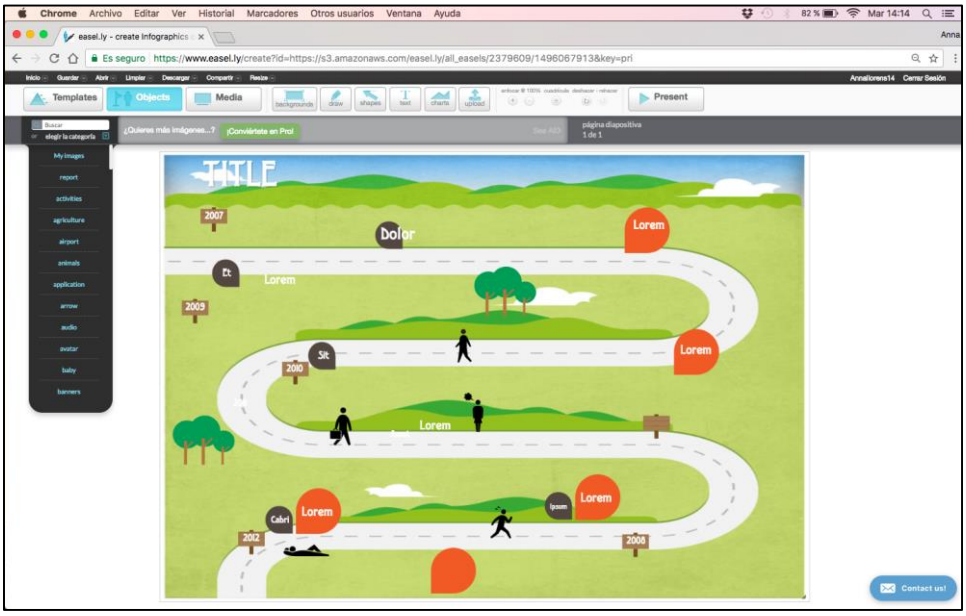

Figura 2: Captura de pantalla de la aplicación easel.ly

Otro ejemplo que nos ha parecido interesante es *Canva*:

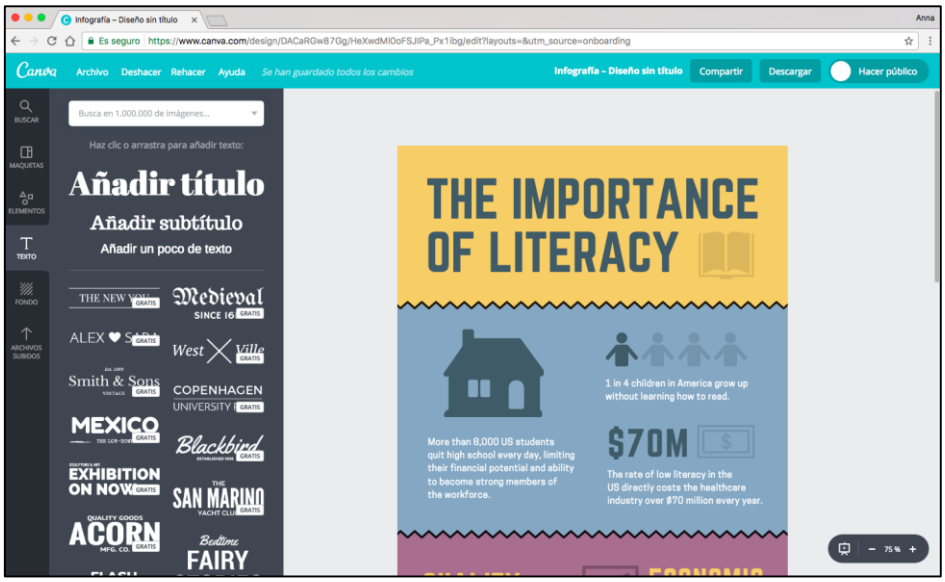

Figura 3: Captura de pantalla de la aplicación Canva

Sin embargo, el resultado que se obtiene con estos editores no es válido para el fin que se requieren las infografías en este proyecto. En los editores online, cuando descargamos la infografía no nos es posible cambiar su contenido, además estas herramientas no se pueden integrar en las aplicaciones de la empresa.

2.2.2 Librerías para tratamiento gráfico

Dado que una infografía es un elemento visual, vamos a utilizar el estándar SVG. Existen librerías que facilitan la generación de elementos gráficos basando su implementación en documentos SVG. Este formato de documento permite visualizar gráficos bidimensionales.

Se explicará con más detalle de que trata el estándar SVG y que ventajas conlleva utilizarlo para la obtención de infografías en otras secciones de la memoria.

Ejemplos de librerías con este propósito: *d3*, *snap.svg*, *raphael.js*, etc.

Figura 4: Librerías desarrolladas en Node

En un primer momento, se trató de integrar estas librerías como dependencia de las aplicaciones de la empresa sin éxito. El resultado que se obtuvo estaba muy limitado y era poco flexible. Las aplicaciones requieren de funcionalidades muy específicas que estas librerías no ofrecían, por lo que finalmente se decidió implementar una librería desde cero.

3 Especificación de requisitos

3.1 Introducción

La presente sección tiene como propósito la especificación de los requisitos presentados en el proyecto siguiendo el estándar IEEE 830 destinado a la especificación de requerimientos del software.

3.1.1 Propósito

El propósito del documento es ofrecer una visión detallada del proyecto, describiéndolo de forma general, enumerando sus funcionalidades y definiendo los conceptos fundamentales para su planteamiento.

A partir de la presente especificación será posible proceder al diseño de la arquitectura e implementación del sistema.

3.1.2 Ámbito

El sistema deberá facilitar la obtención de infografías a distintas aplicaciones tanto aplicaciones web como móvil.

3.1.3 Definiciones acrónimos y abreviaturas

API: Siglas de "*Application Programming Interface*". Conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas [3].

Base64: Sistema de numeración posicional que usa 64 como base. En la aplicación se usará para codificar las imágenes en formato PNG [6].

CSS: Siglas de "*Cascading Style Sheets*". Es el lenguaje utilizado para describir la presentación de documentos HTML o XML [8].

Dashboard: "Panel digital" en español. En el ámbito de las empresas se utilizan para mostrar de una manera visual los resultados, estadísticas, pronósticos, etc. [5]

DOM: Siglas de "*Document Object Model*". Es una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos en XML [11].

Drag and drop: "*Arrastrar y soltar*" en español. Es un dispositivo señalador en el que el usuario selecciona un objeto virtual "agarrándolo" y arrastrándolo a una ubicación diferente [12].

Framework: Estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de *software* [7].

Jenkins: Es un servidor de automatización de código abierto y autónomo que puede utilizarse para automatizar todo tipo de tareas, como el desarrollo, la realización de pruebas y despliegue de software [22].

JSON: De las siglas "*JavaScript Object Notation*". Es un formato ligero de texto para el intercambio de datos. Está inspirado en el lenguaje de programación JavaScript, pero puede ser usado perfectamente en otros contextos [23].

Grid: Cuadricula en español. Un grid es tabla con un número de filas y columnas definidos. Los grids se utilizarán en sistema para diseñar y estructurar las infografías.

Layout: "*Diseño o disposición"* en español. Este término, en el proyecto será utilizado para definir la posición de los distintos elementos que formarán una infografía.

PNG: De las siglas "Portable Network Graphics". Formato de archivos gráfico basado en un mapa de bits.

Software: Soporte lógico de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas [32].

SVG: Siglas de "*Scalable Vector Graphics"*. Formato de gráficos vectoriales bidimensionales, tanto estáticos como animados en formato XML [38].

Template: Para el desarrollo de la aplicación se ha utilizado el término "*Template"* como una plantilla de diseño con un estilo predefinido para las infografías.

TFG: Trabajo de Final de Grado.

Taiga: Es un software gratuito que permite gestionar proyectos de forma ágil y accesible, en la que se van estableciendo tareas y a realizar por los miembros del proyecto [39].

W3C: Siglas de "*World Wide Web Consortium".* Consorcio internacional que genera recomendaciones y estándares que aseguran el crecimiento de la web [43].

XML: De las siglas "*eXtensible Markup Language"*. Es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por W3C y utilizado para almacenar datos en forma de árbol [44].

3.1.4 Referencias

Para la elaboración de la especificación se ha seguido el estándar IEEE especificado en:

IEEE Std 830 - *IEEE Guide to Software Requirements Specifications*. *IEEE Standards Board. 345 Eas 47 th Street*. New York, NY 10017, USA.1998.

3.2 Descripción general

En este apartado se describe el producto de forma abstracta y general, sin entrar en aspectos técnicos.

3.2.1 Perspectiva del producto

La principal función del sistema es poder interpretar una infografía definida por otra aplicación y representarla como una imagen. Para ello, será necesario implementar una librería capaz de diseñar infografías y posteriormente establecer un sistema de comunicación compatible entre aplicaciones.

Este TFG pretende desarrollar un software para la generación de infografías orientado a diferentes plataformas simultáneamente, tanto en aplicaciones móvil, aplicaciones de web y/o aplicaciones de escritorio.

3.2.2 Funciones del producto

El producto deberá satisfacer las funcionalidades que se enumeran a continuación:

- Proporcionar un sistema que permita definir una infografía y representarla como imagen (usaremos para ello principalmente el formato PNG, si bien otros formatos también pueden ser aceptados).

- Gestionar peticiones web para establecer comunicación entre aplicaciones.

- Gestionar *templates* por defecto para las infografías.

- Manual de usuario documentado para su uso.

3.2.3 Características de los usuarios

En este apartado hay que destacar el hecho de que la aplicación está orientada a usuarios desarrolladores de software que estén familiarizados con lenguajes de

programación. No está orientada a usuarios convencionales ya que se trata de un producto para facilitar la visualización de datos de otras aplicaciones. Es decir, la aplicación no constará de una interfaz de usuario en la que el usuario convencional está acostumbrado a interaccionar.

3.2.4 Restricciones

Se deberán contemplar y cumplir una serie de condiciones a la hora de desarrollar el producto:

- Multiplataforma. Para poder gestionar la comunicación con distintas aplicaciones de la empresa.

- Lenguajes de programación: JavaScript, Java.

- La aplicación deberá contar con una infraestructura de entrega continua [14] e integrarse en Jenkins junto con los demás proyectos de la empresa.

- La parte lógica de la aplicación deberá ser gestionada por un sistema automatizado que incluya su despliegue en el servidor.

- Deberá tener una API compatible con otras aplicaciones de la empresa.

3.2.5 Suposiciones y dependencias

En caso de que se incumplan o modifiquen los siguientes factores podrían derivar en el incumplimiento de los requisitos. Estos son:

- Se espera la utilización del producto por parte de un usuario desarrollador de software, familiarizado con Java, JavaScript y objetos en JSON.

- El servidor web debe ser adecuado para el correcto funcionamiento de la aplicación.

- Se necesita un banco de imágenes adecuado a la temática de la infografía a generar.

3.3 Requisitos específicos

Este apartado describe el producto de forma más detallada, abordando aspectos técnicos. Esto nos permite conocer qué funciones debe cumplir la aplicación para ser considerada como plenamente funcional.

3.3.1 Interfaces

3.3.1.1 Interfaces con el usuario

No aplicable.

3.3.1.2 Interfaces con el software

- Se deberá contar con una API para que otras aplicaciones puedan usar la librería.
- La comunicación entre otras aplicaciones con el servidor deberá realizarse mediante servicios web utilizando objetos en JSON. En el caso de utilizar la librería como una API, se usarán también objetos en JSON.
- La librería debe tratar las peticiones que reciba y devolver una infografía en SVG al servidor.
- El servidor debe devolver las infografías en formato imagen (PNG, JPG o SVG) según requiera la aplicación cliente.

3.3.1.3 Interfaces de comunicación

Las peticiones que se realicen al servidor deberán hacerse mediante el protocolo http (o su versión segura https).

3.3.2 Requisitos funcionales

Los requisitos funcionales definen el comportamiento del sistema. El sistema deberá permitir:

Generación de infografías

Nombre	**Inicialización de un documento SVG**
Código	FR-01
Función	Generar un documento.
Descripción	La aplicación debe permitir al usuario elegir las dimensiones de la infografía en el momento de su creación.

Nombre	**Añadir fondo**
Código	FR-02
Función	Añadir un fondo a la infografía.
Descripción	Se deberá seguir con los estilos de fondo de CSS o elegir una imagen prediseñada.

Nombre	**Incrustar imágenes**
Código	FR-03
Función	Incrustar imágenes en base64 a las infografías.
Descripción	Se deberán poder añadir imágenes en formato JPG Y PNG al documento.

Nombre	**Anidar SVG**
Código	FR-04
Función	Anidar otras imágenes en formato SVG.
Descripción	Dentro del documento principal SVG, se podrán anidar otros documentos en SVG ya sean creados por la propia aplicación o por una aplicación externa.

Nombre	**Incluir elementos geométricos**
Código	FR-05
Función	Incluir elementos geométricos vectoriales.
Descripción	Se trata de las formas básicas como son: rectas, círculos, polígonos, etc. Estos elementos están definidos en el estándar de SVG y la aplicación deberá poder gestionarlos.

Nombre	**Incluir texto**
Código	FR-06
Función	Añadir elementos de texto.
Descripción	Las infografías deberán poder contener texto por lo que la aplicación deberá poder añadir elementos de texto al documento. Gestionar los textos, saltos de líneas, máximo de palabras por línea, alineación del texto, etc.

Nombre	**Incluir propiedades de estilo**
Código	FR-07
Función	Facilitar al usuario de la aplicación una lista con propiedades de estilo.
Descripción	Los elementos que formarán las infografías podrán tener un estilo propio.

Nombre	**Definición de un free Layout**
Código	FR-08
Función	Facilitar al usuario de la aplicación la creación de un documento ordenado.
Descripción	Los elementos en el Layout free se podrán añadir en cualquier parte del documento. Cada elemento tendrá definida una posición y unas dimensiones dentro del documento.

Nombre	**Definición de un grid Layout**
Código	FR-09
Función	Facilitar al usuario de la aplicación la creación de un documento mediante celdas.
Descripción	El usuario podrá definir el número de filas y columnas que tendrá el grid. Los elementos en el grid se podrán añaden en indicando el número de fila y columna y se le podrán definir márgenes.

Nombre	**Tratar objetos JSON**
Código	FR-10
Función	Tratar y validar los objetos definidos en JSON.
Descripción	La definición de la infografía deberá estar en JSON, lo que facilitará la comunicación de la librería con otras aplicaciones.

API

Nombre	**Creación de un paquete importable por otros proyectos**
Código	FR-11
Función	Integración del generador de infografías en otras aplicaciones de la empresa.
Descripción	Está API deberá contener todos los objetos necesarios para la creación de una infografía.

Nombre	**Serialización**
Código	FR-12
Función	Serializar los objetos de la API en formato JSON.
Descripción	Se serializarán todos los objetos de la API en JSON, para que el generador de infografías pueda leer la definición.

Servidor

Nombre	**Exportación**
Código	FR-13
Función	Exportación de las infografías en PNG y SVG
Descripción	Según el formato que requiera la aplicación que realiza una petición al servidor de infografías, se deberá poder devolver la infografía en formato PNG o en SVG.

Nombre	**Petición para la obtención de una infografía en SVG**
Código	FR-14
Función	Procesamiento de una solicitud y respuesta al cliente.
Descripción	Servicio RESTful que reciba mediante el método POST un documento en formato JSON y devuelvan a la aplicación que lo solicite una imagen (en SVG).

Nombre	**Petición para la obtención de una infografía en formato imagen**
Código	FR-15
Función	Procesamiento de una solicitud y respuesta al cliente.
Descripción	Servicio RESTful que reciba mediante el método POST un documento en formato JSON y devuelvan a la aplicación que lo solicite una imagen (en PNG).

3.3.3 Requisitos lógicos de la base de datos

No aplicable.

3.3.4 Requisitos de rendimiento

El sistema deberá tener un tiempo de respuesta no superior a un segundo, de acuerdo con *Nielsen* [29]. Sin contar con los tiempos de comunicación.

3.3.5 Restricciones de diseño

El aspecto más importante a la hora de diseñar el sistema deberá ser la claridad descriptiva y funcional de todas las especificaciones que necesitará un usuario a la hora de interaccionar con la aplicación. Por ello, es importante remarcar la importancia de generar una API bien documentada.

3.3.6 Atributos

Disponibilidad

Se establecerá un acuerdo de nivel de servicio (SLA) superior al 99%. Si el servicio no esté disponible se responderá con un estado HTTP 503 (service unavailable).

Fiabilidad

La aplicación deberá ser fiable. La probabilidad de su buen funcionamiento deberá tender a 1, es decir, el sistema deberá realizar sus funciones correctamente de acuerdo con las especificaciones requeridas. En caso de fallo se tratará en la medida de lo posible y, en caso de no poder ser tratado, deberá ser notificado al usuario.

Portabilidad

La aplicación deberá estar disponible en varias plataformas, por lo que se debe utilizar un lenguaje de programación que facilite dicha portabilidad.

4 Metodología y tecnologías utilizadas

En esta sección se explicará la metodología empleada para estructurar, planificar y controlar el proyecto, y se introducirán las tecnologías implicadas en la fase de implementación entre las que se incluyen: herramienta para el control de versiones, lenguajes de programación, *frameworks* y librerías.

4.1 Metodología ágil: SCRUM

En esta sección se detalla la manera de trabajar de la empresa de cara a la realización de un proyecto. La empresa toma un enfoque ágil en el desarrollo del software.

Este enfoque ágil se lleva a cabo mediante **Scrum**, sin llegarse a adoptar de forma estricta en este proyecto [9].

Scrum es un proceso de gestión y control que reduce la complejidad para centrarse en la creación de software satisfaciendo las necesidades del negocio. Los equipos entregan el software de trabajo de forma incremental y empírica con la máxima productividad posible [37].

Scrum trabaja con el marco temporal del *sprint*. Un ***sprint*** es un bloque temporal cuya duración suele estar entre dos semanas y un mes. Su objetivo principal consiste en desarrollar un producto completamente funcional y apto para ser puesto en producción.

Dentro de cada *sprint* se definen cuatro etapas principales:

- Reuniones de **planificación**: reunión que se realizara al inicio del *sprint* para planificar sus tareas.

- **Scrum diario**: cada miembro del equipo realiza una pequeña exposición de que ha realizado desde la reunión del día anterior, comunica cuales son los impedimentos con los que se encuentra y actualiza el estado de la lista de tareas de la iteración (Sprint backlog) .

  La lista de objetivos a complementar en cada iteración, la gestionamos mediante un **tablón de tareas** (Scrum taskboard) concretamente con el *framework* Taiga [39].

- **Revisión** del sprint: se revisa el sprint una vez ha sido terminado. Se pretende determinar si los objetivos se han cumplido o no.

- **Retrospectiva** del Sprint: última reunión del sprint y tiene como objetivo inspeccionar los elementos más importantes en cuanto a relaciones, personal, herramientas y procesos, y mejoras con respecto a cómo el equipo de Scrum hace su trabajo.

En la Figura 5 se muestra una captura de pantalla del tablón de tareas de un *sprint* del proyecto (del 21 de marzo al 4 de abril):

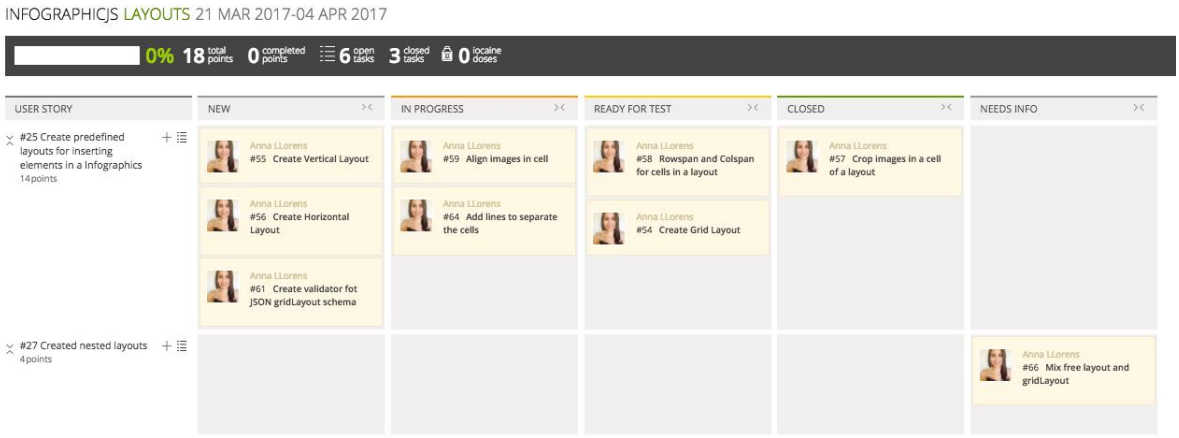

Figura 5: Captura de pantalla del "Scrum taskboard"

Como se observa en la Figura 5 al lado de cada objetivo se especifican las tareas necesarias para completarlos, y se van moviendo hacia la derecha para cambiarlas de estado. Los estados son: pendientes de iniciar, en progreso, preparadas para testear, hechas o con falta de información.

4.2 Control de versiones

Un sistema de control de versiones es una herramienta que registra todos los cambios hechos en un proyecto y permite guardar las versiones del producto en todas sus fases de desarrollo. Para el desarrollo de este proyecto hemos hecho uso de **Git**.

Git es un sistema de control de versiones desarrollado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código [16]. Durante el desarrollo del proyecto se ha utilizado un repositorio git privado de la empresa.

Sobre los detalles técnicos de este, destacaremos los comandos básicos utilizados en el desarrollo del proyecto:

- *git add*: Marca los archivos modificados, para que sean subidos al repositorio en la siguiente sincronización (commit).

- *git commit*: Guarda todos los archivos marcados en el repositorio local.

- *git push*: Guarda los cambios del repositorio local en el repositorio remoto.

- *git pull*: Actualiza el repositorio actual al commit más nuevo.

Internamente el sistema de git utiliza un sistema de ramas. Al tratarse de un proyecto en el que no han intervenido más desarrolladores, para las aplicaciones que se incluyen en este TFG, se han creado dos ramas de git:

- **Master**: Es la rama que se crea por defecto al inicializar un nuevo proyecto. En ella se encuentra una versión funcional, probada y estable del código.

- **Development**: En esta rama se han ido testeando y añadiendo funcionalidad a las aplicaciones.

El comando *git branch* lista las ramas presentes en el proyecto:

```
allorens@developer3:~/workspace/InfographicJS$ git branch
* development
  master
allorens@developer3:~/workspace/InfographicJS$
```

Figura 6: Ramas del proyecto InfographicJS establecidas en Git

4.3 JavaScript

Para la implementación de la parte lógica de la aplicación se ha utilizado el lenguaje interpretado de JavaScript junto con Node.

JavaScript es un lenguaje de programación ligero orientado a objetos basados en prototipos. Se trata de un lenguaje de scripting multi-paradigma y dinámico. Entre sus capacidades dinámicas se incluyen la construcción de objetos en tiempo de ejecución, listas variables de parámetros, variables que pueden contener funciones, creación de scripts dinámicos y recuperación de código fuente. La sintaxis básica es similar a Java o C++. Las construcciones del lenguaje tales como sentencias condicionales o bucles funcionan de la misma manera que en estos lenguajes.

Aunque tradicionalmente ha sido reducido a realizar tareas en el navegador como mejoras en la interfaz de usuario y páginas web dinámicas, es actualmente un lenguaje de programación tan capaz como cualquier otro lenguaje tradicional como C++ o Java

por lo que es usado cada vez más en entornos sin navegador como Node o Apache CouchBD [21].

4.4 Node

Node es una plataforma de software dedicada a crear aplicaciones de red rápidas, flexibles y escalables. Proporciona un entorno de ejecución de E/S de gran rendimiento y ofrece a los programadores la posibilidad de ejecutar código JavaScript del lado del servidor ya que está basado en el motor V8 de la máquina virtual de Google [30].

En Node el código se organiza en módulos. Los módulos son como los paquetes o librerías que utilizan otros lenguajes como Java. En el núcleo de Node se incluyen un conjunto de módulos básicos. Estos módulos se encuentran en el directorio "*node_modules"* y toman el nombre de **módulos nativos**, ejemplos de estos son *http* o *fs*.

4.4.1 Gestor de paquetes: NPM (Node Package Manager)

Adicionalmente, a lo largo del desarrollo de una aplicación con Node nos veremos obligados a utilizar **módulos de terceros** creados por la gran comunidad que apoya esta plataforma.

Npm es uno de los gestores de paquetes de Node. Un paquete contiene todos los ficheros necesarios para un módulo [31].

Mediante la siguiente instrucción se instala un nuevo módulo en nuestra aplicación:

```
#  npm install [nombre_del_módulo]
```

Npm permite crear aplicaciones desde cero con la instrucción:

```
#  npm init
```

Cuando ejecutamos esta orden se inicia un asistente en consola al que se le indican el nombre y la versión de la librería y de manera opcional el repositorio git, el autor, etc.

Finalmente se crea el fichero "*package.json*" en el directorio raíz de la aplicación. Los módulos Node que se requieran para la aplicación se añaden como dependencias él fichero cuando se instalan.

La Figura 7 se muestra un fragmento del fichero "*package.json*" de la librería *infographic-js* en el que se listan los módulos npm que tiene instalados como dependencias.

```
"devDependencies": {
    "ajv": "^4.11.8",
    "chai": "^3.5.0",
    "chai-string": "^1.3.0",
    "gulp": "^3.9.1",
    "gulp-bump": "^2.6.1",
    "gulp-if": "^2.0.2",
    "gulp-mocha": "^3.0.1",
    "gulp-util": "^3.0.8",
    "mocha": "^3.2.0",
    "yargs": "^6.6.0"
  },
  "dependencies": {
    "ajv": "^4.11.8",
    "xmldom": "^0.1.27"
  }
```

Figura 7: Fragmento fichero package.json

Repositorio privado de npm

Para la gestión de los nuevos módulos de Node que se han sido desarrollados en este proyecto hemos utilizado **sinopia** [35].

Sinopia es un módulo para la gestión de repositorios privados de npm. Permite tener un registro local configurado desde cero. El módulo de sinopia está alojado en un servidor privado de la empresa. La configuración de este, queda fuera del alcance de este proyecto.

4.5 JSON

"*JavaScript Object Notation*". Es un estándar basado en texto plano para el intercambio de datos. Está basado en un subconjunto del lenguaje de programación de JavaScript. Json posee un formato predefinido y no depende de ningún lenguaje de programación, permitiendo así, la comunicación entre aplicaciones de manera transparente y sin problemas. [23].

Json está constituido por dos estructuras:

- Una colección de pares de nombre/valor.
- Una lista ordenada de valores.

4.6 DOM: Document Object Model

El modelo de objetos de documento en español, es una interfaz de plataforma que proporciona un estándar para representar principalmente objetos en XML, SVG y HTML.

El DOM está definido y administrado por W3C por lo que los distintos navegadores aplican las especificaciones definidas en el éste para dar soporte en sus aplicaciones. A lo largo de la historia de los navegadores se han ido aplicando en mayor o menor medida las características del DOM por lo que puede que haya algunas que todavía no se encuentren implementadas [11].

El DOM sigue una estructura de nodos en la que se define un nodo raíz y se le van anidando los demás nodos hijo:

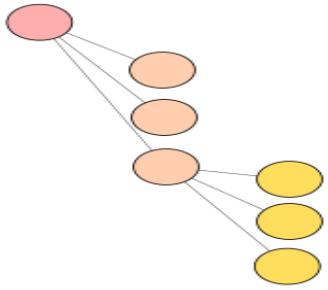

Figura 8: Anidación de los nodos en el DOM

4.6.1 SVG

SVG (Scalar Vector Graphics) es un lenguaje para describir gráficos bidimensionales en formato XML cuya especificación es un estándar abierto desarrollada por W3C. Ha sido expresamente diseñado para trabajar conjuntamente con otros estándares del W3C tales como CSS, DOM y SMIL [38].

Como se ha nombrado anteriormente SVG tiene un modelo de objetos de documento (DOM) por lo que mediante lenguajes de *scripting* como JavaScript se puede acceder a todos sus elementos, atributos y propiedades.

Las infografías que genera la aplicación son un conjunto de archivos SVG renderizados que forman una imagen.

¿Por qué elegir SVG?

Se trata de un formato gráfico independiente de la resolución. Los gráficos no pierden calidad si son ampliados o redimensionados, permitiendo tener un mismo gráfico para distintas resoluciones de pantalla.

SVG basa toda su implementación en tres tipos de objetos gráficos: elementos geométricos vectoriales, imágenes de bits y texto.

Los **elementos geométricos vectoriales** son caminos o trayectorias formadas por rectas y curvas delimitadas por un área. En el estándar de SVG vienen definidos: rectángulos, círculos, elipses, líneas, polígonos y trayectorias. Cada uno de ellos con sus correspondientes atributos.

Figura 9: Elementos geométricos definidos por el estándar SVG

Las **imágenes de bits** son una estructura de datos representada con una rejilla rectangular de píxeles o puntos de color denominada matriz. Esta matriz se puede añadir al DOM del fichero SVG y ser interpretada como imagen.

Los elementos de **texto** se representan como otros elementos gráficos. Por lo tanto, las transformaciones del sistema de coordenadas y el estilo se aplican a los elementos de texto de la misma manera que se aplican a las formas tales como trazados y rectángulos.

Todos estos objetos pueden ser agrupados, estilizados y compuestos en objetos previamente renderizados. Todos ellos se encuentran enmarcados en una ventana con un ancho y un alto determinados. Esta ventana tiene un sistema de coordenadas cuyo origen está situado en la parte superior izquierda orientado a la derecha y hacia abajo respetivamente.

Adicionalmente, se pueden añadir atributos de **estilo** a los objetos. SVG utiliza las propiedades de estilo para describir muchos de sus parámetros de documento. Éstas, definen la forma en que se deben representar los elementos gráficos (objetos) del contenido SVG. Muchas de las propiedades se comparten con CSS y XSL. SVG utiliza propiedades de estilo para:

- Elementos geométricos que son claramente visuales, se incluyen todos los atributos que definen como un objeto es “pintado”, los colores de relleno y de trazo, anchos de línea, estilos de guion, etc.

- Parámetros relacionados con el estilo del texto como es la fuente, su tamaño, espacio entre las palabras del texto, etc.

4.7 Java

Java es un lenguaje de programación orientado a objetos, rápido, seguro y fiable que se usa para el desarrollo de aplicaciones en entornos muy variados.

Una de las principales características es que es un lenguaje independiente de la plataforma, es decir, Java no se ejecuta en el sistema operativo sino en su máquina virtual JVM. Esta independencia del S.O ha contribuido a que se hayan desarrollado números productos y se haya aumentado la potencia y rendimiento de las aplicaciones para Java [20].

En el proyecto se ha creado una librería en Java gestionada por **Maven** [26] para la integración de la aplicación con otras aplicaciones de la empresa desarrolladas en JavaEE.

5 Arquitectura del proyecto

Una vez realizado un análisis de las tecnologías que se utilizarán para la implementación del sistema, en esta sección, explicaremos como se han relacionado entre ellas para conseguir un sistema multiplataforma de generación de infografías.

La arquitectura de un sistema determina la organización fundamental de éste desarrollando un plan general que asegurará que los requisitos del proyecto se cumplen y las aplicaciones se integraran correctamente.

En la arquitectura se definen los componentes, sus interfaces y las relaciones que se establecerán entre ellos. Los componentes que hemos definido representan los tipos de elementos que entrarán en la fase de desarrollo del generador de infografías y los hemos agrupado según su función para poder simplificar su implementación.

Para el desarrollo del sistema hemos utilizado una arquitectura cliente/servidor. Este modelo pretende lograr un mejor rendimiento de los componentes y conocer el tipo de responsabilidad que tiene cada uno de ellos.

En la Figura 10 se muestra el resultado de nuestro planteamiento:

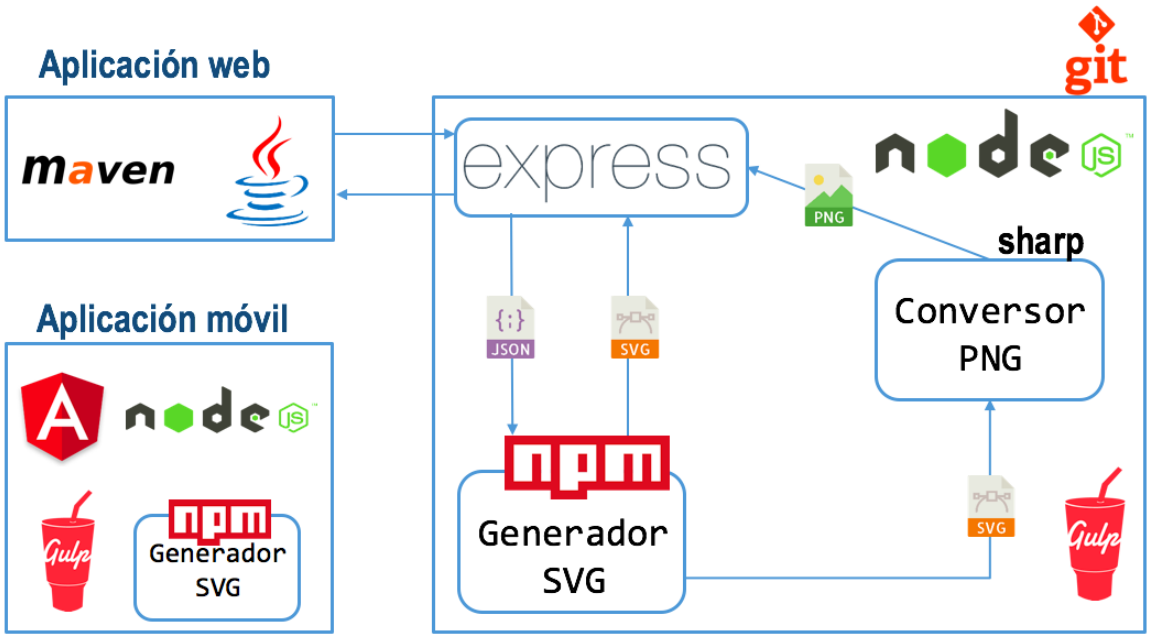

Figura 10: Arquitectura del proyecto

Tal y como se especificaba en los requisitos integraremos el sistema en aplicaciones web y plataformas móvil. Dado que estos utilizan tecnologías distintas la forma de integrarlas también cambiará.

En cuanto a las aplicaciones móvil se desarrollará un nuevo módulo en Node que permita obtener una infografía. El módulo deberá ser capaz de leer la definición de una infografía en formato JSON y devolver una imagen en formato SVG (Generador SVG en la Figura 10). Este módulo se añadirá como dependencia a la aplicación móvil.

Por otra parte, las aplicaciones web de la empresa también necesitan utilizar el generador SVG. Estas aplicaciones están desarrolladas en Java y no se puede integrar directamente como ocurre con las aplicaciones móvil.

Para que el generador SVG y las aplicaciones web puedan comunicarse se deberá instalar el generador en un servidor desarrollado en Node (Express) y facilitarle una API a la aplicación web que le permita realizar las peticiones *RESTful* al servidor.

Adicionalmente, las infografías deberán poder devolverse al cliente en formato PNG por lo que el servidor dispondrá de un conversor a PNG. Según el tipo de petición que realice la aplicación cliente se le devolverá en un formato u otro. El conversor finalmente escogido se llama Sharp.

Diagrama de clases

El generador SVG mostrado en la Figura 10 contiene la parte más importante del sistema, en él se ha implementado la lógica que nos permite generar infografías para servir a las aplicaciones que lo requieran.

Para especificar la estructura del generador se ha diseñado un **diagrama de clases**, en éste se muestra las clases del generador sus atributos y sus relaciones.

En la Figura 11 se muestra el diagrama de clases que sigue:

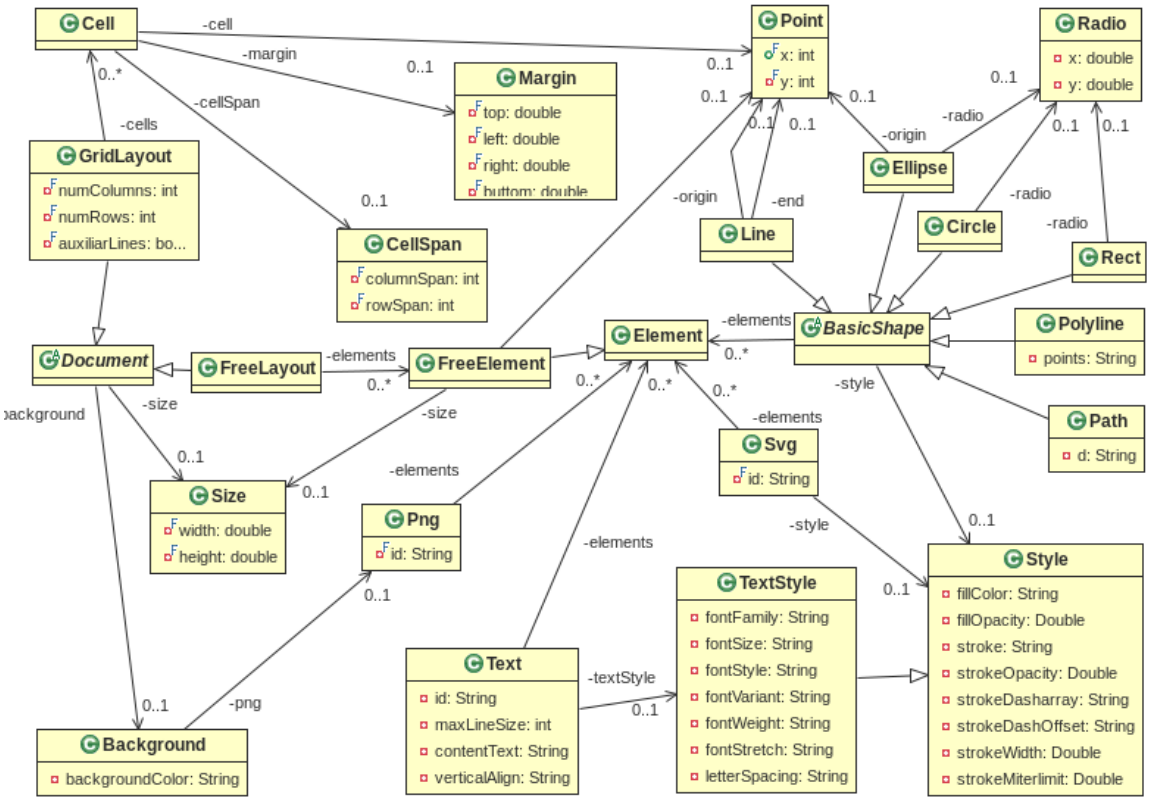

Figura 11: Diagrama de clases del generador de Infografías

Diagrama de Casos de Uso

Como ya se indicó en la sección destinada a los requisitos del software, el sistema no cuenta con una interfaz gráfica. El usuario de las aplicaciones web y móvil no interaccionan directamente con el generador de infografías. Las aplicaciones serán las encargadas de comunicarse con el generador y solicitar las infografías.

En la Figura 12 se muestra el caso de uso definido para la aplicación móvil:

Figura 12: Casos de uso para aplicación móvil

Cuando el usuario de la aplicación desee visualizar una infografía, la aplicación móvil según los datos que tenga en su base de datos del usuario será la encargada de comunicarse con el generador y obtener la infografía adecuada.

En la Figura 13 se muestra el caso de uso definido para la aplicación móvil:

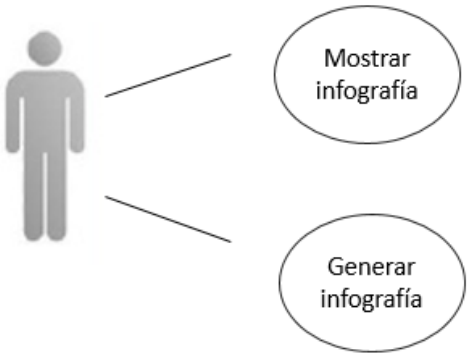

Figura 13: Casos de uso aplicación web

Al igual que en el caso de la aplicación móvil, la aplicación web será la encargada de comunicarse con el generador. En este caso, el usuario de la aplicación proporciona los datos de entrada que se muestran en las aplicaciones.

En este momento contamos con una estructura de clases que definen la lógica que sigue el sistema para generar infografías y por otra parte una visión global de la arquitectura que se ha definido para lograr un sistema multiplataforma.

En la siguiente sección explicaremos con detalle cómo hemos implementado cada uno de los componentes que forman nuestro sistema.

6 Implementación de la aplicación

Esta sección la hemos dividido en dos secciones principales: la primera sección la destinaremos a explicar cómo hemos implementado el generador de infografías y la segunda como se ha integrado en otras aplicaciones incluyendo la implementación del servidor.

6.1 Generador de infografías

Para el desarrollo de una librería capaz de generar infografías se optó por desarrollar un nuevo módulo de npm al que hemos denominado **infographic-js**. Esta librería de npm dada una definición de infografía en un objeto JSON puede leerlo y generar un documento SVG como resultado. Ha sido desarrollada principalmente en JavaScript con la plataforma Visual Studio Code (más información en el Anexo A).

Infographic-js contiene la parte más importante de todo el sistema. En él se encuentra la lógica de la aplicación que hace posible la creación de una infografía.

Este nuevo módulo de *npm* será almacenado en el repositorio de Sinopia [](repositorio privado de la empresa), permitiendo así que otras aplicaciones de la empresa lo descarguen y utilicen.

La estructura de directorios que hemos decido para el generador es la siguiente:

/node_modules	**Contiene los módulos propios de Node y los que se han utilizado para el desarrollo de la librería.**
/src/images	Se incluye una librería de imágenes prediseñadas en formato PNG y SVG que se pueden añadir a las infografías que se obtienen.
/src/lib	En este directorio se incluye todo el código JavaScript.
/src/lib/layouts	Incluye los Layouts de los que constará la librería.
/src/lib/svgManager	Incluye el código en JavaScript que trata los documentos SVG.
/src/lib/jsonManager	Validación de documentos JSON.
/src/test	Contiene los test realizados a la librería. Se explicarán en la sección destinada a los test unitarios.

Figura 14: Estructura de directorios

6.1.1 Infografía

Las infografías son imágenes que internamente están basadas en documentos SVG. A continuación, explicaremos como la librería genera los documentos SVG y anida los elementos al DOM para generar imágenes.

Las infografías que genera la librería se definen por:

- Una imagen de **fondo**. El fondo de las infografías se puede definir mediante elementos del estándar SVG, con imágenes predefinidas en la librería en formato PNG.

- Un conjunto de **elementos gráficos**. Entre los que se incluyen: elementos de texto, imágenes en formato SVG y PNG y figuras geométricas.

- Un conjunto de **Layouts**. Los Layouts permiten distribuir los elementos que se definan para una infografía.

6.1.2 Gestión de Layouts

La finalidad de los layouts es facilitar una herramienta al programador para el diseño automático y el posicionamiento del contenido gráfico. Mediante los layouts se podrá estructurar el documento y añadir los elementos de manera ordenada.

El estándar SVG tiene definida una propuesta muy interesante para la implementación de layouts, pero a día de hoy no se encuentra implementada. Se puede consultar en [45].

Para el desarrollo de esta librería se han implementado dos tipos de Layout:

- **Layout de tipo free:** Es el Layout con menos restricciones, pero también el más costoso de generar. Permite añadir los elementos en posiciones arbitrarias en el área de diseño. En él, se indican el tamaño que ocupara cada elemento en el Layout. Las posiciones se especifican con coordenadas horizontales y verticales. Se utilizará para añadir elementos concretos que se requiera definir sus posiciones y tamaño manualmente.

- **Layout de tipo grid:** Se trata de una cuadrícula en la que se indican el número de filas y columnas. Los elementos se van insertando en las celdas de la tabla. No es necesario indicar la dimensión de los elementos. La librería lo calcula en función del espacio que se le haya adjudicado a la celda. Adicionalmente, se les pueden definir márgenes y expansión a las celdas. Este tipo de Layout se utilizará para estructurar el documento en distintas secciones.

6.1.2.1 Combinación de Layouts

La librería permite combinar varios Layouts anidados para una misma infografía. Inicialmente se establece el tipo de documento con un Layout como principal. A éste, se le añaden tantos layouts como se necesiten para definir una infografía.

Según el tipo que se utilice se definen unas propiedades distintas ya que cada Layout trata sus elementos y espacio de forma distinta.

Adicionalmente, durante el diseño de la infografía se pueden "activar" líneas auxiliares que sirven de guía para el desarrollador y muestran los márgenes de los layouts. Una vez finalizado el diseño se pueden "desactivar".

Ejemplo práctico

En la Figura 15 se muestra una infografía a modo de ejemplo en la que se pretende ilustrar la combinación de los layouts.

Figura 15: Combinación de los layouts

Procedimiento

Para la inicialización del documento se han definido las dimensiones, se ha utilizado un Layout de tipo grid (3x1) y hemos activado el *flag* para visualizar las líneas auxiliares. Se trata de la disposición principal del documento que nos permite estructurarlo. Los layouts que se han anidado son:

1. Layout de tipo **grid** (1x1) en el que hemos anidado en su única celda un Layout de tipo **free** con un texto centrado y dos imágenes en formato SVG.

2. Layout de tipo **grid** (4x4): Este Layout está dividido en 4. En las celdas de la primera fila se ha añadido un elemento central con márgenes y las celdas de la segunda fila se han anidado 2 layouts de tipo grid (el 4 y 5).

3. Layout de tipo **grid** (1x1) en la que hemos anidado un Layout de tipo **free** y hemos insertado 3 images en formato SVG.

4. Layout de tipo **grid** (4x4) que forma parte del Layout 2.

5. Layout con la misma estructura que el Layout 4.

La elección del tipo de layouts es bastante flexible. Se podrían haber elegido otros layouts con los elementos definidos de forma distinta y se obtendría el mismo resultado.

6.1.3 Generador de documentos SVG

Los elementos gráficos que se definen en cada layout se tratan para generar un documento SVG.

Para explicar cómo la librería genera documentos SVG nos hemos servido del esquema definido en la Figura 16:

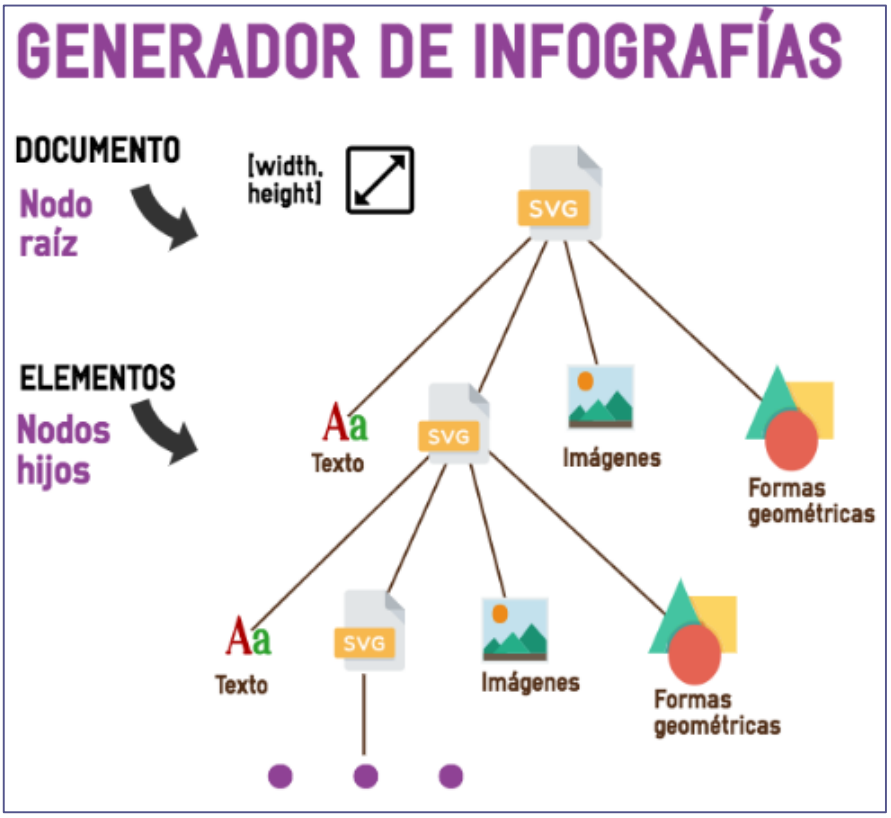

Figura 16: Funcionamiento del generador de infografías

Inicialmente en la Figura 16 encontramos el nodo raíz, en él, se define el documento como SVG y se establecen las dimensiones de la infografía. Adicionalmente se le añade un estilo de fondo que se aplica a todo el documento.

Para inicializar el documento se le añade el id root que facilitará la búsqueda de sus hijos a partir de su id, además se añaden los enlaces a w3c que permiten utilizar la definición del estándar SVG.

```
1  <svg id="root" width="400" height="400"
2  xmlns="http://www.w3.org/2000/svg"
3  xmlns:xlink="http://www.w3.org/1999/xlink"/>
```

Figura 17: Cabecera para interpretación de documentos SVG

Una vez tenemos inicializado el documento se podrán ir anidando los distintos elementos (nodos hijos). Todos ellos se encontrarán al mismo nivel con respecto al nodo padre. Cada uno, tiene unas propiedades y atributos distintos por lo que se han definido distintas funciones para anidarlos al documento.

Las **imágenes** en formato PNG, JPG, o GIF se transforman en *base64* para poder ser incrustadas en el DOM, una vez transformadas se añaden al documento raíz con la

etiqueta <image>. Para poder ser visualizadas se le debe indicar la posición y dimensiones que se deseen de la imagen.

```
1  <image id="png01" width="400" height="400" x="0" y="0"
2  xlink:href="
..  h3tsSBKDFR8kQc9sXfm2NdoELpCkiz8L6pCkiziVBORwqVkAAAxe23JjdvRr32
46 />
```

Figura 18: Fragmento de un fichero SVG con una imagen en formato PNG

La **formas geométricas** que trata la librería son: rectángulos, círculos, elipsis, líneas, polígonos y trayectorias, cada una de ellas tiene su etiqueta correspondiente: *<rect>* , *<circle>* , *<ellipse>*, *<line>*, *<polygon>* y *<path>* con sus atributos específicos.

```
1  <rect x="10" y="10" width="30" height="30" fill="red"/>
2  <circle cx="25" cy="75" r="20" fill="blue"/>
3  <ellipse cx="75" cy="75" rx="20" ry="5" fill="yellow"/>
4  <line x1="10" x2="50" y1="110" y2="150"/>
5  <polygon points="50 160 55 180 70 180 60 190 65 205 50 195 35
    190 30 180 45 180" fill="green" />
6  <path d="M20,230 Q40,205 50,230 T90,230" fill="blue" />
```

Figura 19: Definición de las distintas formas geométricas

Con respecto a la implementación de los elementos de **texto,** existen infinidad de atributos de estilo definidos, destacar que se han tenido ciertas limitaciones, ya que algunas de las funciones definidas en el estándar SVG para tratar los elementos de texto todavía no han sido implementadas por los navegadores, por lo que no se visualizan.

```
1  <text font-family="Verdana" font-size="55">foo text</text>
```

Figura 20: Definición de un elemento de texto

Por último, el elemento más completo que se puede añadir a nuestro documento SVG es otro **documento SVG**. La librería permite anidarlos, estableciendo así un tercer (cuarto, quinto. etc.) nivel en la jerarquía de nodos. Estos documentos a su vez, como se muestra en la imagen, podrán tener todos los elementos que hemos definido para nuestro generador de infografías. Ésta funcionalidad es de gran utilidad cuando se requieran infografías con mucho contenido en el que se podrán definir por partes y finalmente anidarlas al documento raíz. También se utiliza para insertar imágenes en formato SVG.

En ambos casos el generador sigue los siguientes pasos:

1. Primero lee los atributos *width* y *height* que definen las dimensiones del segundo documento y calcula las transformaciones que se deberán realizar a éste, para que ocupe la posición correspondiente en el documento raíz.

2. A continuación, se hace uso del elemento *<g>*. Este elemento es un contenedor en el que se pueden agrupar objetos. Todos los elementos del segundo documento quedarán agrupados dentro del contenedor.

3. Las transformaciones que se han calculado previamente se definen en el elemento *<g>* y se aplican sobre todos los elementos (hijos) del mismo.

   En el atributo *transform* se define la posición (*translate*) y tamaño (*scale*) que se deberán aplicar a las propiedades de los objetos para que se re-escalen con respecto a la posición que ocuparán en el documento principal.

```
1  <svg id="root" width="400" height="400" >
2     <g id="child1" transform="translate(0,20)scale(0.55,0.85)">
3       <rect width="500" height="220" x="30" y="50" fill="blue"/>
4     </g>
5  </svg>
```

Figura 21: Anidación de documentos SVG

Para recorrer y añadir elementos al documento SVG se ha hecho uso de los siguientes métodos para JavaScript que recorren los elementos del DOM:

- **`getElementById`** `(id)`
  - Devuelve el nodo del elemento cuyo *id* se le pasa como parámetro.
  - `Ej. fileId = rootDocument.getElementById('fileId')`

- **`getElementsByTagName`** `(name)`
  - Devuelve una lista ordenada con todos los nodos Element que tengan como nombre de etiqueta el *name* que se le pasa como parámetro.
  - `Ej. firstNode = parentSvg.getElementsByTagName('svg')[0]`

- **`createElement`** `(type)`
  - Crea un nodo de tipo Element del tipo especificado en *type*.
  - `Ej. rect = parentSvg.createElement('rect')`

- **`createTextNode`** `(contentText)`

        - Crea un nodo tipo Text con el valor *contentText.*
        - `Ej. parentSvg.createTextNode('Report')`

    - **`appendChild`** `(nodoNuevo)`
        - Añade el nodo que se le pasa como parámetro al nodo padre que se le indique.
        - `Ej. parentSvg.appendChild(rect)`

    - **`removeChild`** `(nodo)`
        - Elimina el nodo que se le pasa como parámetro del conjunto de hijos del nodo al que se aplica.
        - `Ej. parentSvg.removeChild(nodeToRemove)`

    - **`setAttribute`** `(nombre, valor)`
        - Añade al elemento un nuevo atributo *nombre,* estableciendo el *valor*.
        - `Ej. g.setAttribute("style", ”fill:#010595”)`

    - **`removeAttribute`** `(nombre)`
        - Elimina el elemento que se le pasa como parámetro a un determinado nodo.
        - `Ej. node[i].removeAttribute(attribute)`

En la Figura 22 se muestra la función en JavaScript que permite la inicialización de un documento en SVG:

```
1  exports.initializeSVG = function (width, height) {
2      var rootDocument = svgHandler.stringToSvg('<svg id="root"></svg>')
3      var attr = rootDocument.getElementById("root")
5      attr.setAttribute('width',width)
6      attr.setAttribute('height',height)
8      attr.setAttribute('xmlns', 'http://www.w3.org/2000/svg')
9      attr.setAttribute('xmlns:xlink', 'http://www.w3.org/1999/xlink')
10     return rootDocument;
11 }
```

Figura 22: Inicialización de un documento en SVG en JavaScript

En la Figura 23 se muestran las distintas funciones que utiliza la librería para añadir los objetos al DOM del documento SVG.

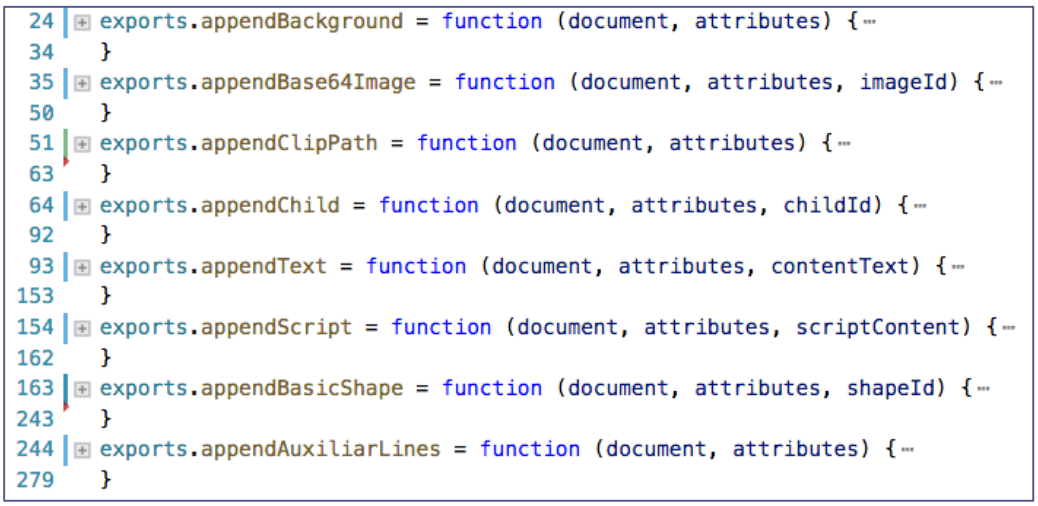

Figura 23: Funciones para anidar objetos a un documento SVG

Como se puede observar en la Figura 23, todas las funciones comparten un objeto en común: ***document***, en éste, se anidan los hijos que forman el documento final.

Todos los elementos definidos en la librería tienen atributos propios y algunos de ellos necesitan un tercer parámetro como es el caso de los elementos de texto en el que se le indicará el contenido del texto.

6.1.4 Validador JSON

La librería deberá leer la definición de una infografía en un objeto JSON proporcionado por una aplicación y generar como resultado una infografía en formato SVG. Para que pueda devolver el resultado esperado y sin errores es necesario determinar si los datos son válidos o no.

Cuando la librería recibe un objeto en JSON comprobará el objeto y si este es válido procederá a la generación, de lo contrario provocará un error y notificará al usuario.

Para la validación de los objetos JSON hemos utilizado **JSON-Schema** [41]. JSON-Schema es el estándar de los documentos JSON que describe la estructura y el tipo de datos permitidos en los objetos JSON.

En nuestra librería hemos definido distintos esquemas de documentos JSON que serán validados mediante **ajv**.

Ajv es el validador de JSON-Schema más rápido para JavaScript. Se trata de un módulo para npm por lo que para utilizarlo lo hemos instalado y añadido como dependencia del proyecto.

6.2 Integración con otras aplicaciones

Una vez tenemos definida la aplicación principal de nuestro sistema, el siguiente paso, siguiendo con los requisitos del proyecto será estudiar cómo integrarla en otras aplicaciones de la empresa concretamente en *Koobepop* y *Sport Medi Score*.

A continuación, explicaremos como se ha integrado en cada una de ellas ya que tal y como se explicó en la sección de arquitectura de la aplicación, al utilizar distintas tecnologías la manera de integrarlas también cambiará.

6.2.1 Integración con JavaScript

El generador de infografías está implementado en JavaScript y gestionado por *npm* por lo que para poder integrarlo con aplicaciones que usan JavaScript como código fuente y están gestionadas por npm solo será necesario instalar el generador como una dependencia más de la aplicación y hacer uso de sus funciones.

6.2.1.1 Caso práctico: Aplicación móvil (koobepop)

Koobepop es una aplicación móvil desarrollada por el alumno Alejandro Melcón en colaboración con la empresa. Se trata de otro TFG dirigido por la empresa que también será presentado próximamente.

La aplicación va dirigida principalmente a usuarios que quieran realizar un seguimiento de su actividad física, ya sea para mantenerse en forma, perder peso o prepararse para correr una maratón. Dicho seguimiento será controlado por un grupo de doctores. Entre las opciones de la aplicación, el usuario tiene la posibilidad de organizarse las actividades propuestas por los especialistas en su calendario, acceder a videos dedicados a explicar ejercicios, consultar sus informes médicos, ver su progreso en el tiempo con los *Dashboard* [5].

La aplicación móvil está desarrollada en Angular. Angular también utiliza Node para el desarrollo de sus aplicaciones por lo que la integración de nuestro módulo npm *infographic-js* ha sido muy fácil. Este es uno de los motivos por los que se decidió desarrollar el generador de infografías con Node.

Para integrar el generador de infografías con *Koobepop* solo hay que añadir el módulo *infographic-js* que se encuentra en el repositorio de Sinopia de la empresa y añadirlo como dependencia de la aplicación.

En la Figura 24 ilustramos como se instala en la aplicación móvil:

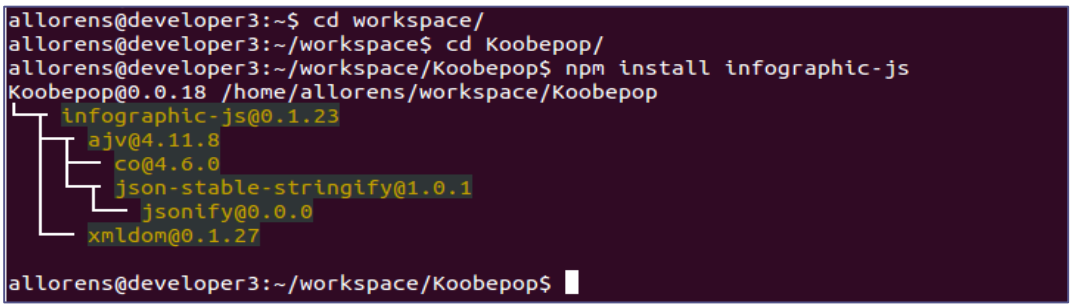

Figura 24: Instalación de la librería en la aplicación móvil

Al instalar el módulo *infographic-js* se instalan los módulos de los que depende. Para hacer uso de él, habrá que importarlo al archivo JavaScript que necesite utilizarlo.

```
1  var infographic = require (infographic-js)
```

Figura 25: Importación del módulo infographic-js

A partir de este momento, se hace uso de las funciones definidas para la obtención de infografías.

En la Figura 26 se muestra una captura de pantalla de la aplicación *Koobepop* en la que se ha hecho uso del generador para mostrar un informe médico al usuario de la aplicación.

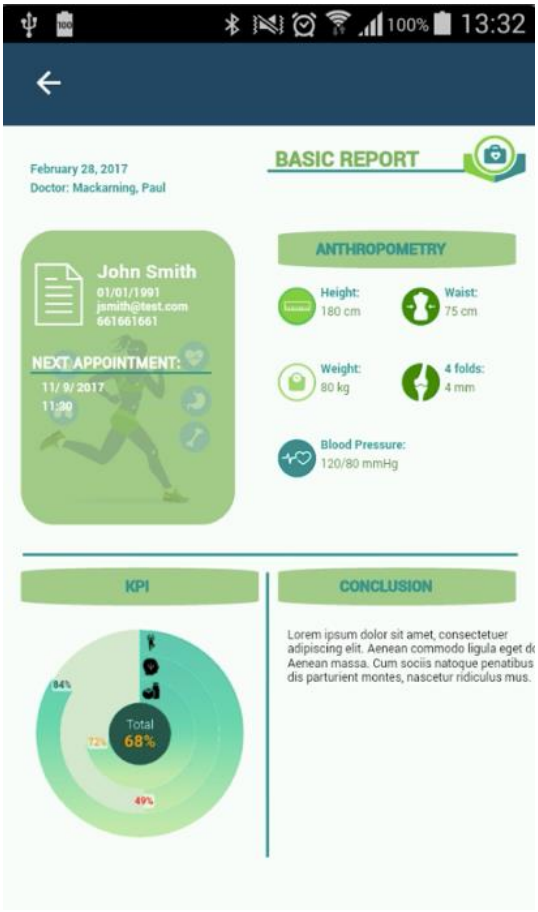

Figura 26: Infografía mostrada por la aplicación móvil

6.2.2 Integración con otros lenguajes de programación

La aplicación web a la que tiene que dar servicio el generador de infografías está basado en código Java por lo que no se pueden integrar directamente. Para que el sistema no este limitado a las aplicaciones basadas en código JavaScript hemos implementado un servidor en Node y como ejemplo una API-cliente compatible con aplicaciones Java, si bien, el servidor al estar basado en servicios web, será compatible con cualquier otro lenguaje que soporte estos servicios.

6.2.2.1 Servidor web

En esta sección veremos cómo se ha implementado el servidor de infografías. Para ello construiremos una API RESTful que pueda ser consumida desde una aplicación web.

Las peticiones RESTful siguen el estándar de los métodos de HTTP. Este principio básico establece una asociación uno-a-uno entre las operaciones de crear (POST), leer (GET), actualizar (PUT) y borrar (DELETE) definidas en el estándar HTTP. Para el desarrollo del servidor hemos hecho uso del método POST que permite incluir datos en la petición.

Para la implementación de este servicio hemos utilizado Node combinado con ***express.*** *Express* es un *framework* mínimo y flexible que funciona mediante Node y proporciona un conjunto robusto de características y permite la gestión de peticiones RESTful [15].

Funcionamiento

El servidor se encuentra activo esperando peticiones de los clientes. Cuando un cliente requiera una infografía realizará una petición al servidor enviando un objeto JSON con la definición de la infografía. El servidor procesa la solicitud del cliente y envía el objeto JSON al generador. El generador devuelve un documento en formato SVG al servidor que se encarga de responder al finalmente al cliente.

Implementación

El servidor es un nuevo proyecto de Npm que hemos denominado **infographicjs-server.** Para la implementación del servidor hemos instalado el módulo *express* de npm que gestionará las peticiones de los clientes y la librería *infographic-js* que nos permitirá generar las infografías.

Para hacer uso de las funciones de *express* y de la librería *infographic-js*, una vez instalados, los hemos incluido en un archivo en el directorio raíz del proyecto al que hemos denominado "*server.js*".

En la Figura 27 se muestra la importación de la librería y del servidor express:

```
1  var express = require('express');
2  var app = express();
3  var infographic = require('infographic-js');
```

Figura 27: Fragmento del fichero server.js

En este momento el servidor ya está listo para utilizarse. Para que la aplicación web se pueda conectar con el servidor hemos definido un *endpoint* con el método POST con la URI: /getSvg.

En la Figura 28 se muestra el funcionamiento de una solicitud *getSvg*:

Figura 28: Solicitud POST getSvg

Adicionalmente, la aplicación cliente podrá solicitar la infografía en formato PNG al servidor. Para que el servidor pueda devolver imágenes en formato PNG hemos instalado el módulo **sharp** [34] de npm en el servidor. Sharp permite convertir imágenes en formato SVG a formato PNG a gran velocidad. Hemos añadido un segundo endpoint en el servidor con el método POST con la URI: /getPng.

En la Figura 29 se muestra el funcionamiento de una solicitud *getPng*:

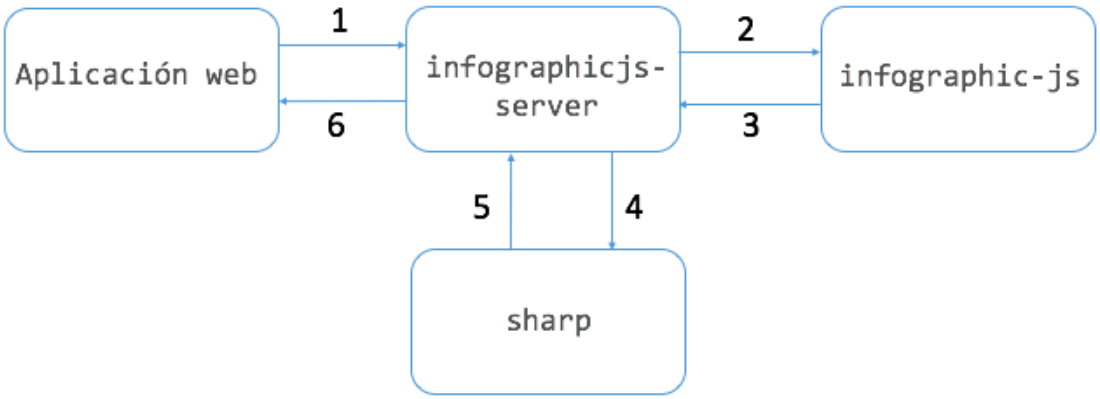

Figura 29: Solicitud POST getPng

Para el despliegue, hemos utilizado un servidor Nginx [28] , ya que es un servidor web muy ligero y más que suficiente para la ejecución de Node Express. Este servidor web a su vez está desplegado dentro de un contenedor Docker [10], lo que nos permite portarlo a cualquier servidor Linux del cliente, así como usarlo de forma local en las pruebas realizadas. Si bien, la metodología utilizada para el despliegue de la aplicación queda fuera del objetivo de este trabajo, por lo que se omite en este documento.

Para realizar las pruebas en el servidor y comprobar que devuelve los códigos http correspondientes hemos utilizado Restlet (Más información Anexo c).

6.2.2.2 API para el cliente

El servidor recibe peticiones con documentos JSON desde una aplicación web. Para facilitar que la aplicación web sea capaz de generar un objeto JSON bien formado, hemos implementado una API-Cliente. Esta API será añadida y gestionada como una dependencia más de la aplicación que la utilice.

La API del cliente podría considerarse un sub-proyecto del generador de infografías surgido de la necesidad de integración con otras aplicaciones en otros lenguajes de programación.

Para desarrollar nuestra API hemos creado un nuevo proyecto en Java al que hemos denominado **infographicJS-client**, y lo hemos gestionado con Maven. Para su desarrollo hemos utilizado la plataforma Eclipse (más información Anexo b).

Maven [26] es una herramienta que facilita la gestión y construcción de proyectos en Java. Ésta, proporciona manejo y administración de dependencias y plugins y una estructura estándar de directorios, la cual facilita la búsqueda de ficheros de código, librerías, etc.

Para la implementación de infographicJS-client hemos seguido la estructura de ficheros proporcionada por Maven. En la Figura 30 se muestra de manera simplificada.

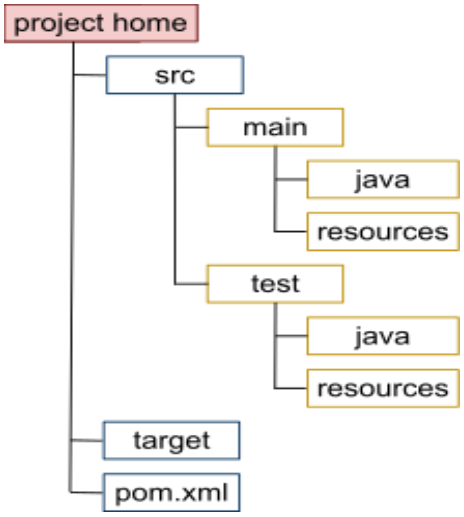

Figura 30: Estructura de ficheros en Maven

El primer paso que hay que seguir a la hora de trabajar con Maven es configurar el *pom.xml*. Este archivo se caracteriza por tener toda su definición bajo una estructura XML. El archivo contiene toda la información del proyecto y las

configuraciones que son utilizadas por Maven a la hora construir el proyecto, así como las dependencias de éste.

La configuración inicial del archivo *pom.xml* es la siguiente:

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2      <modelVersion>4.0.0</modelVersion>
3      <groupId>com.biit</groupId>
4      <artifactId>infographicjs-client</artifactId>
5      <packaging>jar</packaging>
6      <version>0.0.1-SNAPSHOT</version>
7      <name>InfographicJS Client</name>
8  </project>
```

Figura 31: Fichero pom.xml

Los elementos de la Figura 31 son los requisitos mínimos que deben aparecer en el POM. En él se indican la versión actual del POM, el id de grupo, que en este caso pertenece a la empresa, el nombre del "artefacto" (aplicación), el tipo de empaquetado que se desee para la aplicación y la versión de ésta. Posteriormente hemos ido configurando este archivo añadiendo distintas dependencias y plugins que se necesitan para su desarrollo y funcionamiento como son su repositorio en git, plugins para realizar test unitarios, etc.

A través de Maven, también desplegamos el artefacto final en un servidor Artifactory [4] privado, lo que permite que esté disponible como dependencia en cualquier aplicación de la empresa.

Para la generación de nuestra API se han creado distintas clases en Java con todos los objetos que utiliza el generador de infografías, los tipos de layouts y los elementos que pueden contener como son: las imágenes, los textos, las formas geométricas, etc.

Una vez definidas las clases que formaran una infografía se deberán serializar en formato JSON para que se puedan realizar peticiones. Para ello, hemos hecho uso de la librería **gson** [46].

Gson es una librería para Java desarrollada por Google de código libre que permite convertir los objetos Java en su representación en JSON. Para poder hacer uso de ella primero se debe definir como dependencia en el *pom.xml*.

En la Figura 32 se muestra como se añade gson como dependencia:

```xml
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.3.1</version>
</dependency>
```

Figura 32: Dependencia Maven gson

Una vez instalada como dependencia, hemos definido como se serializará cada uno de los objetos para que nuestro generador de infografías pueda interpretar el objeto en JSON correctamente y devolver la imagen.

Se ha definido una clase en Java por cada uno de los elementos que deberán aparecer en la definición de la infografía en formato JSON.

Ejemplo de uso

A continuación, mostraremos el funcionamiento de nuestra API mediante un caso práctico en el que definimos una infografía a modo de ejemplo, mostraremos la serialización en JSON que realiza la API y el resultado que obtendría el generador con esta definición.

```java
FreeLayout free = new FreeLayout(new Size(488, 687));
free.addElement(new FreeElement(new Point(0, 0), new Size(488, 687),
                new Svg("antropometry")));
Text title = new Text("Antropometry");
title.getStyle().setFontSize("55px");
title.getStyle().setFillColor("#ffffff");
title.getStyle().setFontWeight("bold");
free.addElement(new FreeElement(new Point(160, 660), null, title));
```

Figura 33: Definición de una infografía en Java

En la infografía de la Figura 33 hemos establecido un layout de tipo *FreeLayout* con un tamaño de (488x687) y hemos insertado dos elementos de tipo *FreeElement* en él. El primer elemento es una imagen SVG predefinida que ocupa todo el documento a la que hemos llamado “antropometry”. El segundo elemento que hemos insertado es un elemento de texto (title) y le hemos añadido estilo.

Una vez definida nuestra infografía llamaremos al serializador que creará un objeto JSON siguiendo con la especificación que se muestra en la Figura 34.

El objeto que crea el serializador es el siguiente:

```
{
  "type": "freeLayout", "width": 488, "height": 667,
  "elements": [
    {
      "content": {"type": "svg","id": "antropometry"},
      "origin": {"x": 0, "y": 0 },
      "size": {"width": 488,"height": 660 }
    },{
      "content": {"type": "text","contentText": "Antropometry",
        "style": "font-size:55px;font-weight:bold;fill:#ffffff;"},
      "origin": {"x": 160, "y": 650},
      "size": {"width": -1.0,"height": -1.0}
    }]
}
```

Figura 34: Definición de la infografía en JSON

Como se puede ver en la Figura 34, en el objeto JSON se han añadido los elementos que hemos definido para la infografía ejemplo. Nuestro generador *infographic-js* ya podrá leer el contenido de este objeto y devolverlo como una infografía en formato SVG. Este objeto JSON es el que la aplicación web enviaría al servidor.

La infografía no es estática, los textos y los valores se generan para cada infografía dependen de los datos de entrada. De esta manera obtenemos infografías personalizadas por cliente. Para ello definimos un conjunto de templates al que se la cambian los datos según el cliente.

En la Figura 35 se muestra la imagen en formato PNG que hemos definido a través de la API-Cliente:

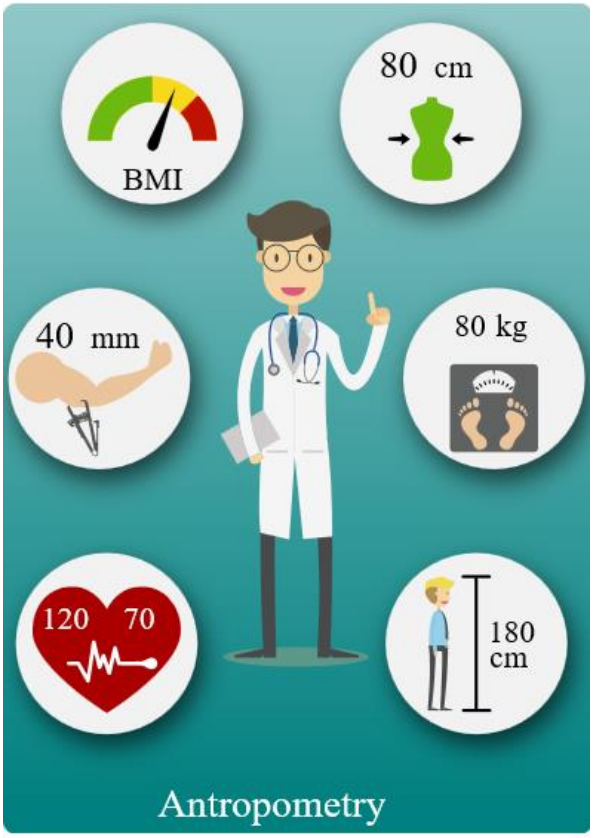

Figura 35: Infografía definida mediante la API

6.2.2.3 Caso práctico: Aplicación web (Sport Medi Score)

La segunda aplicación a la que da servicio nuestro generador de infografías es *Sport Medi Score*. Se trata de una aplicación web desarrollada en Java dirigida a la administración de clínicas médicas privadas. Esta aplicación se encuentra en funcionamiento desde Septiembre de 2015 en el Orbis Medical Center situado en Sittard-Geleen (Países Bajos).

La aplicación tiene definidos distintos tipos de usuario como son médicos, asistentes, recepcionistas o administradores. Entre las acciones que estos pueden realizar relacionadas con nuestro propósito destacaremos la creación de un informe médico en PDF que se le manda por correo al paciente. En este informe se incluye el resultado de distintos test realizados por el médico durante la consulta, recomendaciones por parte del médico y adicionalmente el médico puede establecer una serie de *KPI's (Key performance Indicator)* para seguir la evolución del paciente.

El propósito de integrar nuestro generador de infografías con esta aplicación es reemplazar el procedimiento actual de creación del informe médico por una infografía. Las infografías mostrarán estos informes de una manera más visual e intuitiva que el informe original en PDF. Además, se añadirán recomendaciones saludables, explicaciones sobre los test realizados, etc.

Sport Medi Score se gestiona con Maven por lo que para que pueda consumir los servicios web definidos en el servidor hemos añadido como dependencia de la aplicación la API del cliente en su fichero *pom.xml*.

Una vez instalada, la aplicación web podrá definir las infografías mediante clases en Java serializarlas a documentos JSON y realizar las peticiones a los métodos del servidor. Según el método POST que se utilice el servidor devolverá la imagen en formato PNG o en formato SVG.

En la Figura 36 se muestra una captura de pantalla de la aplicación *Sport Medi Score* en la que se ha hecho uso del generador para mostrar un informe al usuario de la aplicación.

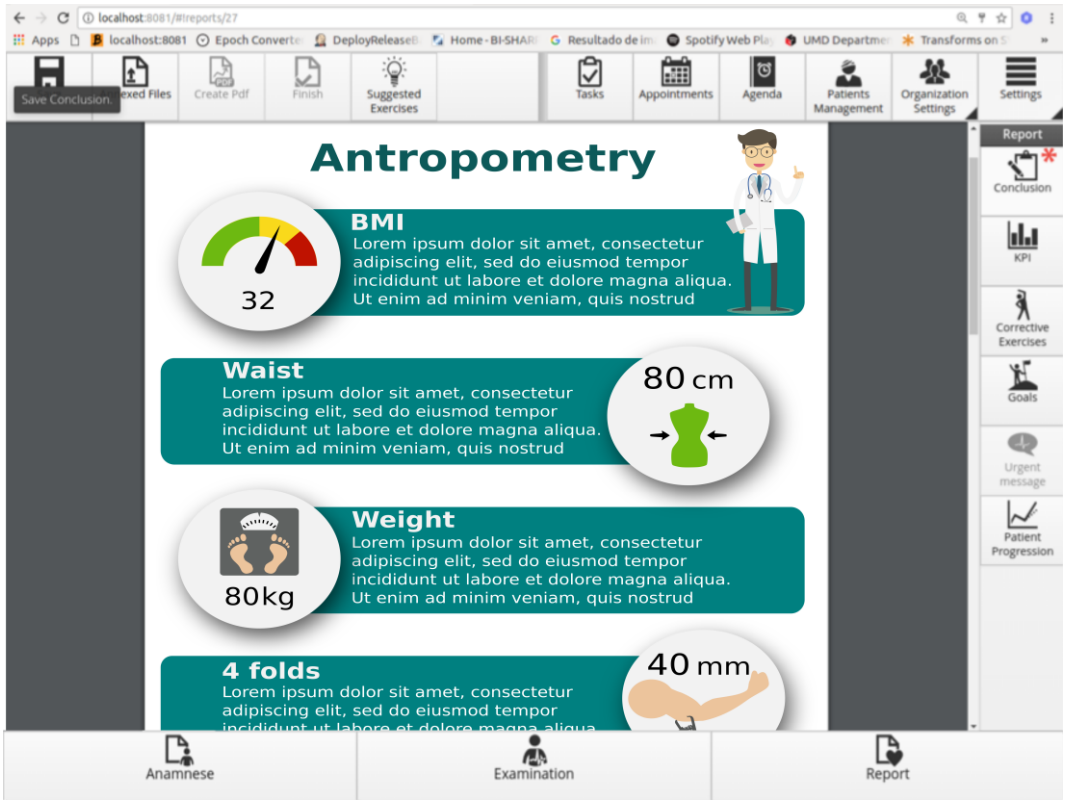

Figura 36: Infografía mostrada por la aplicación web

7 Pruebas de la aplicación

Las pruebas forman parte del ciclo de desarrollo y permiten verificar el buen funcionamiento y la calidad de una aplicación. Para ello, hemos aplicado una estrategia de pruebas que abarca tanto testeo automático como pruebas por parte del usuario.

Para realizar las pruebas hemos definido un conjunto de **pruebas unitarias** que se ejecutan durante el **despliegue** de la aplicación y nos aseguran que no se instala una nueva versión de la aplicación si no se pasan correctamente. Para ello, hemos definido un conjunto de ficheros con los resultados esperados por lo que para que se ejecuten correctamente se comprobarán los resultados con los datos de los ficheros definidos.

Tal y como hemos explicado en la sección destinada a la implementación de la aplicación, nuestro sistema tiene por una parte la lógica de la aplicación que permite generar infografías en JavaScript y, por otra parte, para su integración con otras aplicaciones hemos implementado una API en Java. Por ello, para realizar las pruebas unitarias de cada una de estas aplicaciones hemos utilizado tecnologías distintas. Concretamente Mocha [27] para realizar el testeo de las aplicaciones basadas en código JavaScript y para el código Java hemos utilizado TestNG [40].

En los siguientes apartados de esta sección trataremos de explicar el funcionamiento de las herramientas de testeo y como hemos realizado el despliegue del sistema.

7.1 Pruebas unitarias con Mocha

Mocha es un *framework* de pruebas para código JavaScript que nos proporciona una sintaxis y una estructura de pruebas con múltiples funciones definidas. Mocha hace uso de test asíncronos que se ejecutan de manera sencilla y en serie, lo que permite una presentación de informes flexibles y precisos.

Para hacer uso del *framework* de pruebas lo hemos instalado en la librería para la generación de infografías y ésta lo ha añadido como dependencia en su archivo *package.json* junto con las otras dependencias de desarrollo. El siguiente comando muestra cómo se instala mocha en una aplicación gestionada por npm:

```
# npm install mocha –-save-dev
```

Una vez instalado, en un fichero que hemos denominado *"infographic.test.js"* hemos añadido los distintos tests unitarios para probar los casos de uso de las funciones definidas en la librería.

Para definir las funciones de los test Mocha llamará a la función *describe* que le sirve como marco para ubicar los test. Esta función recibe como primer parámetro un String con el título del test y como segundo parámetro un *callback* con la función del test que queremos probar. Dentro de la función describe llamaremos a la función *"it ()"* tantas veces como test vayamos a realizar.

En la Figura 37 se muestra la función *describe* con uno de los test realizados a la aplicación en el que se comprueba que el documento SVG se inicializa correctamente.

```
1  describe('svgGeneratorTest', () => {
2    it('initializeSVG', () => {
3      xmlDoc = svgGenerator.initializeSVG(400, 400)
4      xmlText = svgHandler.svgToString(xmlDoc)
5      expectedResult = fileHandler.readFileFromTestResults('initializeSVG')
6      expect(xmlText).to.equalIgnoreSpaces(expectedResult)
7     })
8  })
```

Figura 37: Test unitario definido en infographic-js

Se han definido varios ficheros con los resultados esperados por los test por lo que en la ejecución de los test se deberán leer estos ficheros y comprobar si se obtienen los mismos resultados.

Una vez definidas todas las funciones que queremos testear de la aplicación con el comando *gulp test* podremos ejecutarlos y ver los resultados por consola.

La Figura 38 muestra el uniforme del resultado de la ejecución de los test:

```
allorens@developer3:~/workspace/InfographicJS$ gulp test
[13:51:13] Using gulpfile ~/workspace/InfographicJS/gulpfile.js
[13:51:13] Starting 'test'...
[13:51:13] Finished 'test' after 12 ms

  ✓ svgGeneratorTest initializeSVGTest: 6ms
  ✓ svgGeneratorTest appendBase64ImageTest: 4ms
  ✓ svgGeneratorTest appendTextTest: 1ms
  ✓ svgGeneratorTest combineDiferentElementsTest: 0ms
  ✓ svgGeneratorTest addBackgroundTest: 0ms
  ✓ svgGeneratorTest appendChildToExistingSvgTest: 0ms
  ✓ svgGeneratorTest combineDiferentElementsTest: 0ms
  ✓ mainFunctionsTest newLayout: 0ms
  ✓ validateJsonSchemasTest jsonFreeLayout: 0ms
  ✓ validateJsonSchemasTest jsonGridLayout: 0ms
  ✓ validateJsonSchemasTest jsonMixed: 0ms
  ✓ validateJsonSchemasTest newSvg: 0ms

  12 passing (18ms)

allorens@developer3:~/workspace/InfographicJS$
```

Figura 38: Ejecución de los test unitarios en infographic-js

7.2 Pruebas unitarias con TestNG

TestNG es un *framework* para pruebas y testing que trabaja con Java y Maven. Está basado en JUnit, pero introduce nuevas funcionalidades que lo hacen más poderoso y fácil de usar.

Los test que hemos implementado para la API crearán distintos objetos JSON con definiciones de infografías y comprobarán que el objeto obtenido es válido para el generador. Para ello, hemos definido un conjunto de ficheros con objetos JSON válidos y durante la ejecución de los test unitarios se leerán estos ficheros y se comprobará que los resultados coinciden.

TestNG se sirve de un conjunto de anotaciones para realizar los test. En las clases definidas para los test hemos añadido una anotación en la que se indica el grupo del test al que pertenece y para cada uno de los test hemos añadido la anotación *@Test* que marcará el método como parte del test.

En la Figura 39 se muestra uno de los test unitarios realizados en el que se comprueba que el objeto GridLayout de la API se inicia correctamente:

```java
@Test(groups = { "gridLayout" })
public class GridLayoutTests {
  @Test (enabled = true)
  public void initGridTest() throws IOException {
    GridLayout gridLayout = new GridLayout(getDim(), NUM_COLS, NUM_ROWS);
    String json = gridLayout.toJson();
    Assert.assertEquals(loadJsonFile("Grid.json"), json);
  }
}
```

Figura 39: Test unitario definido en infographicjs-client

Una vez definidas todas las funciones que queremos testear de la aplicación con el comando *mvn test* podremos ejecutarlos y ver los resultados por consola.

La Figura 40 muestra el uniforme que realiza Maven con el resultado de la ejecución de los test:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running TestSuite
Tests run: 17, Failures: 0, Errors: 0, Skipped: 1, Time elapsed: 0.488 sec - in TestSuite

Results :

Tests run: 17, Failures: 0, Errors: 0, Skipped: 1

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.790 s
[INFO] Finished at: 2017-06-16T11:01:59+02:00
[INFO] Final Memory: 11M/309M
[INFO] ------------------------------------------------------------------------
allorens@developer3:~/workspace/InfographicJS-client$
```

Figura 40: Ejecución de los test unitarios en infographicJS-client

7.3 Despliegue automático

El despliegue es la fase final del flujo de trabajo. Tras haber implementado el sistema y haber definido un conjunto de pruebas que comprueban que el se ejecuta correctamente, solo queda por realizar el **despliegue** de la aplicación.

Para el despliegue de las aplicaciones, siguiendo con la forma de trabajar de la empresa hemos utilizado Jenkins [22]. Jenkins es un servidor que automatiza todo tipo de tareas, desde el desarrollo a realización de pruebas y despliegue de software.

En esta fase final se sube el código al servidor Git, Jenkins comprueba las pruebas definidas para la aplicación y si se ejecutan correctamente, se genera una nueva versión de la aplicación que se desplegará en el servidor correspondiente (Sinopia para

aplicaciones JavaScript y Artifactory para Java). De este modo la aplicación queda disponible para su uso para cualquier miembro del equipo de desarrollo. Como hemos trabajado con un sistema de control de versiones el despliegue será relativamente sencillo.

7.3.1 Despliegue en Node

Para realizar el despliegue en Node inicialmente, analizamos las herramientas disponibles para Node. Entre ellas destacamos Gulp y Grunt. Tras comparar ambas herramientas y analizando las ventajas que supondría utilizar cada una de ellas se optó por Gulp [24].

Gulp es un *framework* que gestiona las dependencias del proyecto y automatiza el flujo de trabajo de manera eficiente. Tanto el servidor como el generador de infografías están gestionados por Gulp [17].

Para instalar Gulp en nuestro proyecto:

```
# npm install  gulp —-save-dev
```

A continuación, creamos el archivo *gulpfile.js* en el directorio raíz del proyecto para almacenar todas nuestras configuraciones Gulp.

El primer paso para usar Gulp es incluirlo en el archivo *gulpfile.js*:

```
1 var gulp = require('gulp');
```

Hemos usado esta variable Gulp para a escribir las tareas que se han necesitado en el desarrollo del módulo **infographic-js**. Estas tareas se ejecutarán de manera automática durante el despliegue de la aplicación. Las tareas definidas son:

- Cambio de la **versión** de la librería.

```
1 gulp.task('bump', () => {
2   gulp.src(['./package.json'])
3     .pipe(gulpif(argv.patch, bump({ type: 'patch' }))) // _._.X
4     .pipe(gulpif(argv.minor, bump({ type: 'minor' }))) // _.X._
5     .pipe(gulpif(argv.major, bump({ type: 'major' }))) // X._._
6     .pipe(gulp.dest('./'));
7 })
```

Figura 41: Tarea de Gulp para cambiar la versión

- Ejecución de los **test unitarios**.

```
gulp.task('test', () => {
  gulp.src('./test/infographic.test.js', {read:false})
    .pipe(mocha({reporter: 'list'}))
    .on('error', (err) => {
      console.log(err.toString())
      process.exit(1)
    })
})
```

Figura 42: Tarea de Gulp para la ejecución de los test unitarios

Las tareas definidas por Gulp se lanzan automáticamente por Jenkins cada vez que éste detecta un cambio en Git. Si Jenkins ejecuta los test correctamente se crea una nueva versión de la aplicación que se almacena en Sinopia [35].

7.3.2 Despliegue en Java

El despliegue para la API-Cliente es gestionado por Maven. Se ha explicado en secciones anteriores la instalación y funcionamiento de Maven.

Cuando Jenkins detecta un cambio en git lanza la orden *mvn test* en el proyecto. Si estos pasan correctamente Jenkins lanzará la orden *mvn reléase* que cambia automáticamente el número de versión y actualiza la aplicación en Artifactory [4].

8 Conclusiones finales

8.1 Objetivos cumplidos

El objetivo principal de este trabajo ha sido mejorar la experiencia de los pacientes de las empresas clientes de BiiT a la hora de utilizar sus aplicaciones. Originalmente, las aplicaciones web de la empresa contaban con un sistema de generación de informes médicos en PDF, pero estos eran poco visuales e intuitivos para el usuario final.

Para mejorar la experiencia del usuario se decidió sustituir el sistema de generación de PDF original por una infografía que mostrara los informes de una manera más visual y entendible para los usuarios.

El generador de infografías será incluido en una nueva aplicación móvil de la empresa que actualmente se encuentra en desarrollo por el alumno Alejandro Melcón y que también será entregada próximamente como TFG.

Finalmente, gracias a la implementación de los servicios web y la API hemos conseguido un sistema multiplataforma y el sistema no queda limitado a la integración de aplicaciones basadas en JavaScript.

8.2 Casos de éxito con clientes reales

La librería se encuentra desplegada en una aplicación web para la gestión del centro médico Orbis Sport en Sittard-Geleen (Países Bajos). Orbis Sports ofrece una gama completa para el cuidado de la medicina deportiva. Desde consultas y tratamientos a pruebas deportivas y diversos exámenes médicos. Todos los informes médicos que se realizan en el centro son enviados por email a los pacientes.

Hemos sustituido el sistema tradicional con el que contaba la empresa para la generación de informes en PDF por el sistema de obtención de infografías desarrollado en este proyecto para la generación de los informes médicos.

8.3 Trabajo futuro

A lo largo del desarrollo de la aplicación, han ido surgiendo una gran cantidad de ideas que podrían mejorar la aplicación y quedan fuera de los objetivos iniciales del proyecto.

En cuanto al sistema actual podemos establecer las siguientes:

- Desarrollo de **nuevos** tipos de **layouts** que permitan definir infografías más precisas al usuario de la aplicación. Se podrían definir por ejemplo un layout de tipo horizontal en el que se insertaran los elementos horizontalmente y otro de tipo vertical en el que se insertaran verticalmente.

- En las peticiones al servidor se podrían añadir **nuevos métodos POST** y en la solicitud del cliente, además del documento JSON un segundo parámetro con las imágenes que se deseen insertar en la librería. Actualmente, las imágenes se encuentran predefinidas en el servidor y las aplicaciones cliente utilizan estas imágenes en sus infografías.

Adicionalmente, se implementará un sistema *Drag and Drop* [12] en el que el usuario interactuaría directamente con una interfaz gráfica para diseñar su infografía. De esta manera cualquier tipo de usuario podría utilizar el sistema y no estaría limitado su uso a los desarrolladores.

8.4 Valoración personal

Mi valoración global sobre este proyecto es muy positiva. Este proyecto me ha ayudado a fomentar una base para la programación de aplicaciones y a conocer de cerca su ciclo de vida. Desde el análisis inicial a las pruebas y mantenimiento que se realizan durante el desarrollo de una aplicación. También he obtenido una base de conocimientos en tecnologías como Node y algunos de sus *frameworks*, que antes desconocía. La idea de utilizar JavaScript del lado del servidor supone trabajar de una forma distinta a la que había experimentado.

Cuando me propusieron este proyecto no tenía idea de todo lo implicaba su desarrollo. Esto, te hace comprender que para realizar un proyecto profesional, aparentemente sencillo, hay detrás mucho trabajo y detalles a tener en cuenta. Por todo esto me siento muy satisfecha con los resultados obtenidos.

Anexos

1. Plataformas de desarrollo

a. Visual Studio Code

Para el desarrollo de código en JavaScript hemos utilizado **Visual Studio Code** [42]. Se trata de un editor de texto, que incluye soporte para la depuración de código y está integrado con git. Trabaja con multitud de lenguajes de desarrollo web, como son PHP, HTML, JavaScript etc.

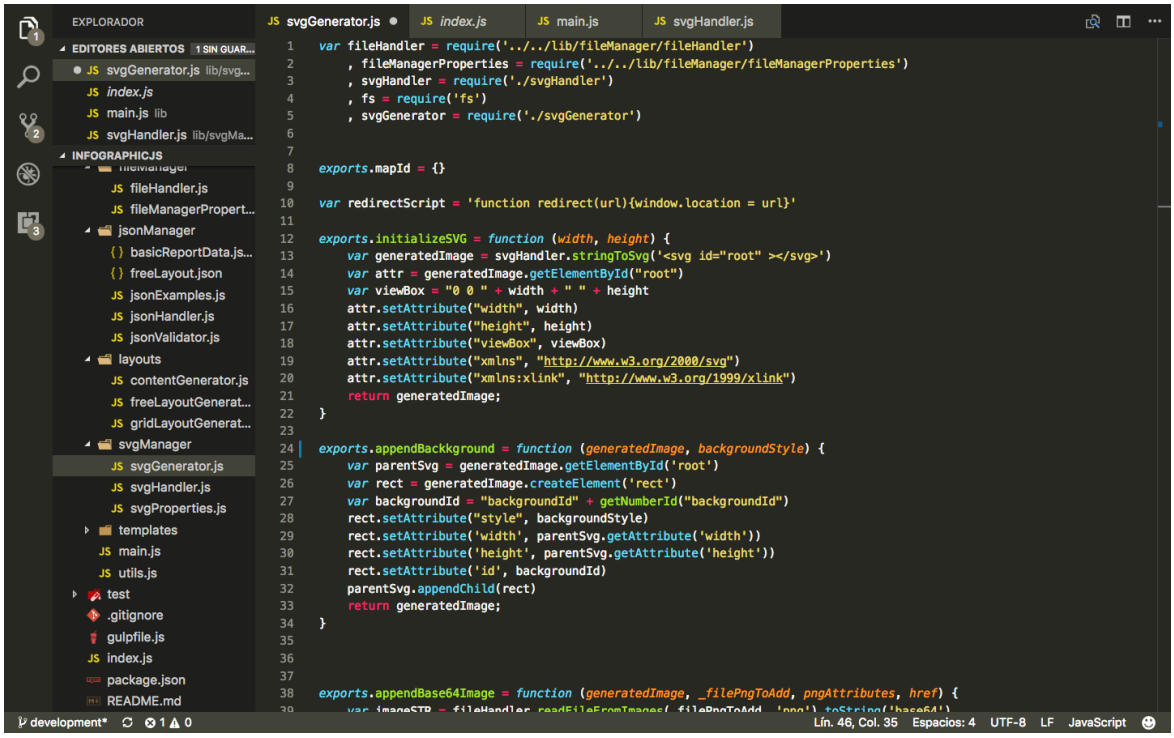

Figura 43: Captura de pantalla de Visual Studio

b. Eclipse

Para el desarrollo de código en Java se ha utilizado **Eclipse Luna** [13]. Eclipse es una plataforma de desarrollo muy completa, compuesta por herramientas de programación de código abierto. Se ha creado para el desarrollar entornos de desarrollo integrados como Java y el compilador (ECJ) que se entrega como parte de Eclipse. Además tiene plugins específicos que han facilitado el trabajo con Maven.

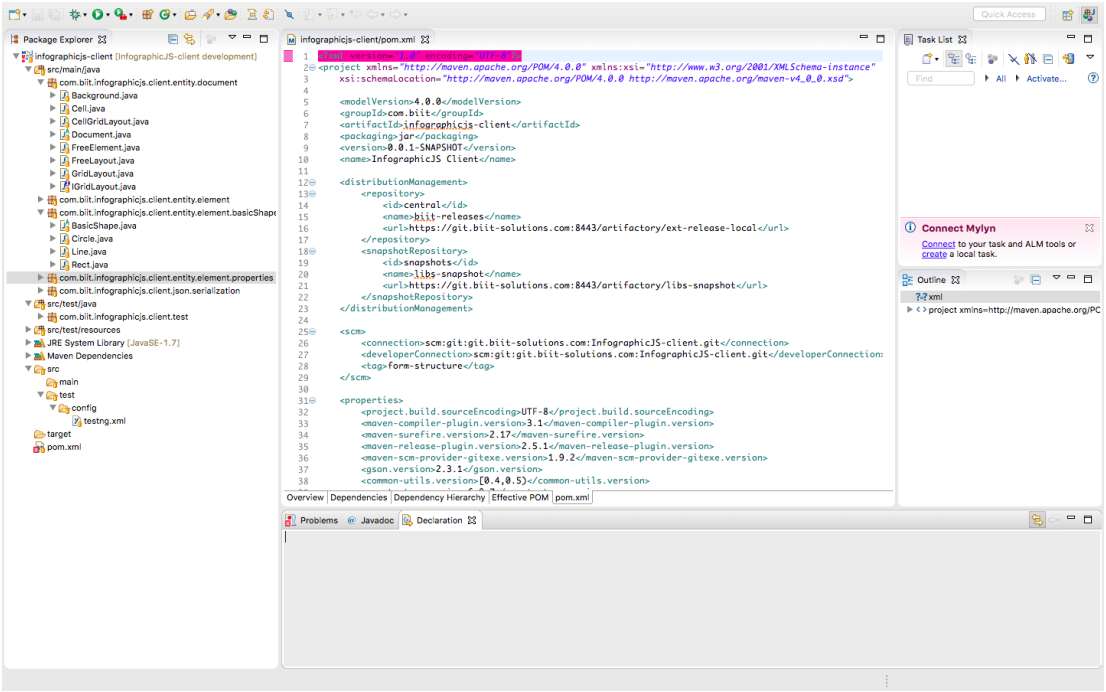

Figura 44: Captura de pantalla de Eclipse

c. Restlet

Para realizar las pruebas en el servidor y comprobar que gestiona las peticiones correctamente hemos utilizado **Restlet** [33]. Restlet es un *framework* ligero completo y de código abierto RESTful. Soporta el transporte de Internet, formato de datos y estándares de descripción de servicios como http Y HTTPS, XML, JSON, etc.

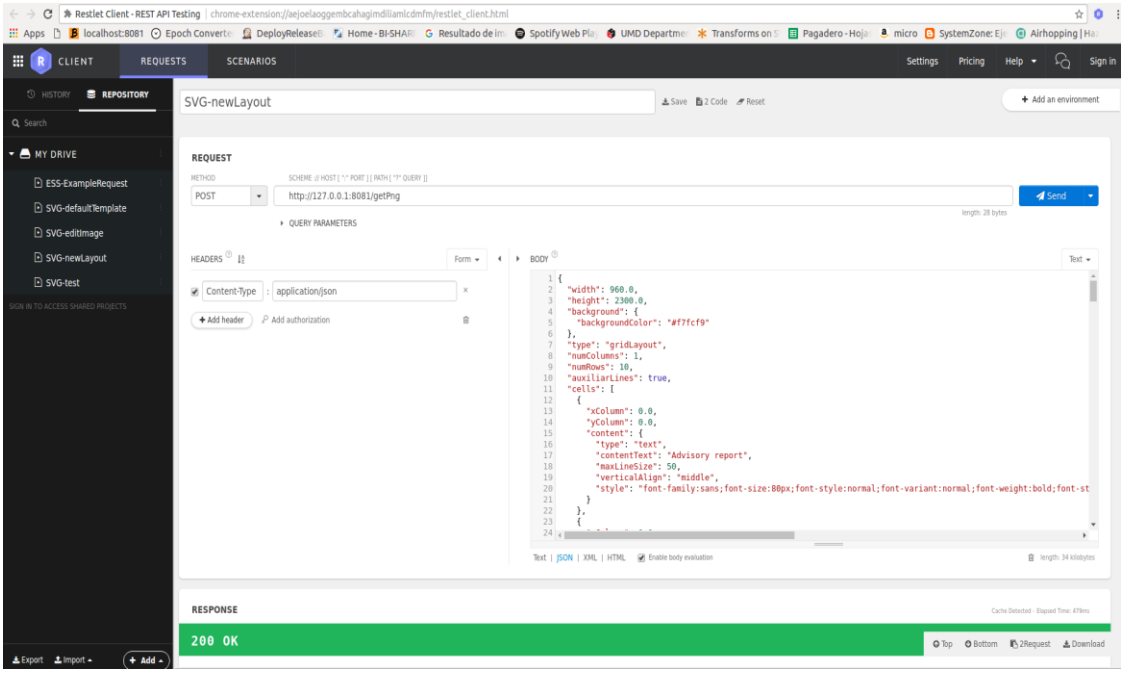

Figura 45: Captura de pantalla de Restlet

Bibliografía

[1] Álvarez, M. (2016). *Customer Experience: La fórmula del éxito para enamorar a los clientes.* Bacelona: Profit editorial.

[2] *Adictos a las redes sociales.* (s.f.). Recuperado el 25 de Abril de 2017, de Subcutaneo Creative: http://www.subcutaneocreative.com/

[3] API. (s.f.). *Wikipedia.* Recuperado el 20 de Marzo de 2017, de https://en.wikipedia.org/wiki/Application_programming_interface

[4] Artifactory. (s.f.). Recuperado el 20 de Junio de 2017, de https://www.jfrog.com/artifactory/

[5] Ascolese, G. (13 de Abril de 2016). *Qué es un dashboard de negocios y cuáles sus beneficios.* Recuperado el 10 de Junio de 2017, de We Are Marketing: https://www.wearemarketing.com/blog/que-es-un-dashboard-de-negocios-y-cuales-sus-beneficios

[6] Base64. (s.f.). *Wikipedia.* Recuperado el 5 de Mayo de 2017, de https://es.wikipedia.org/wiki/Base64

[7] Christensson, P. (7 de Marzo de 2013). *Framework Definition*. Recuperado el 10 de Mayo de 2017, de https://techterms.com/definition/framework

[8] CSS. (s.f.). *Mozilla Developer Network.* Recuperado el 30 de Mayo de 2017, de https://developer.mozilla.org/es/docs/Web/CSS

[9] Manifesto for Agile Software Development (s.f.). Recuperado el 25 de Mayo de 2017, de http://agilemanifesto.org/

[10] Docker. (s.f.). Recuperado el 21 de Junio de 2017, de https://www.docker.com/

[11] DOM. (s.f.). *W3C.* Recuperado el 1 de Junio de 2017, de https://www.w3.org/DOM/

[12] Drag and Drop. (s.f.). *Wikipedia.* Recuperado el 22 de Abril de 2017, de https://en.wikipedia.org/wiki/Drag_and_drop

[13] Eclipse. (s.f.). Recuperado el 30 de Junio de 2017, de https://eclipse.org/

[14] Entrega continua. (s.f.). Recuperado el 15 de Junio de 2017, de https://aws.amazon.com/es/devops/continuous-delivery/

[15] Express. (s.f.). Recuperado el 1 de Junio de 2017, de http://expressjs.com/

[16] Git. (s.f.). Recuperado el 30 de Mayo de 2017, de https://git-scm.com/

[17] Gulp. (s.f.). Recuperado el 30 de Mayo de 2017, de http://gulpjs.com/

[18] Infografía. (s.f.). *Ecured.* Recuperado el 20 de Abril de 2017, de https://www.ecured.cu/Infograf%C3%ADa

[19] Inteligencia empresarial. (s.f.). *Wikipedia.* Recuperado el 15 de Abril de 2017, de https://es.wikipedia.org/wiki/Inteligencia_empresarial

[20] Java. (s.f.). Recuperado el 30 de Mayo de 2017, de https://www.java.com/es/

[21] JavaScript. (s.f.). *Mozilla Developer Network.* Recuperado el 15 de Mayo de 2017, de https://developer.mozilla.org/es/docs/Web/JavaScript

[22] Jenkins. (s.f.). Recuperado el 20 de Junio de 2017, de https://jenkins.io/

[23] JSON. (s.f.). Recuperado el 2 de Mayo de 2017, de http://www.json.org/

[24] Liew, Z. *Automating Your Workflow.*

[25] Maria, E. (23 de Julio de 2016). *Informe medico.* Recuperado el 10 de Mayo de 2017, de Consumoteca: http://www.consumoteca.com/bienestar-y-salud/medicina-y-salud/informe-medico/

[26] Maven. (s.f.). Recuperado el 30 de Mayo de 2017, de https://maven.apache.org/

[27] Mocha. (s.f.). Recuperado el 20 de Junio de 2017, de https://mochajs.org/

[28] Nguix. (s.f.). Recuperado el 21 de Junio de 2017, de https://www.nginx.com/resources/wiki/

[29] Nielsen, J. (1993). *Usability Engineering.* Mountain View: Morgan Kaufmann.

[30] Node. (s.f.). Recuperado el 15 de Mayo de 2017, de https://nodejs.org/es/

[31] Npm. (s.f.). Recuperado el 15 de Mayo de 2017, de https://www.npmjs.com/

[32] Porto, J. P. (2008). *Definición de software*. Recuperado el 5 de Mayo de 2017, de http://definicion.de/software/

[33] Restlet. (s.f.). Recuperado el 30 de Junio de 2017, de https://restlet.com/

[34] Sharp. (s.f.). Recuperado el 20 de Junio de 2017, de http://sharp.dimens.io/en/stable/

[35] Sinopia. (s.f.). Recuperado el 20 de Mayo de 2017, de https://www.npmjs.com/package/sinopia

[36] Socialmood. (s.f.). *Cómo hacer infografías: 90 recursos útiles.* Recuperado el 2 de Mayo de 2017, de 40deFiebre: https://www.40defiebre.com/recursos-para-crear-infografias/

[37] Sutherland, J., & Schwaber, K. (s.f.). *What is Scrum?* Recuperado el 25 de Mayo de 2017, de http://www.scrumguides.org/

[38] SVG. (s.f.). *W3C*. Recuperado el 2 de Mayo de 2017, de https://www.w3.org/TR/SVG

[39] Taiga. (s.f.). Recuperado el 20 de Mayo de 2017, de https://taiga.io/

[40] TestNG. (s.f.). Recuperado el 20 de Junio de 2017, de http://testng.org/doc/

[41] Vepsäläinen, J. (23 de Enero de 2017). *ajv - The Fastest JSON Schema Validator - Interview with Evgeny Poberezkin.* Recuperado el 2 de Junio de 2017, de https://survivejs.com/blog/ajv-interview/

[42] Visual Studio Code. (s.f.). Recuperado el 30 de Junio de 2017, de https://code.visualstudio.com/

[43] W3C. (s.f.). *W3C*. Recuperado el 2 de Mayo de 2017, de https://www.w3.org/

[44] XML. (s.f.). *W3C*. Recuperado el 2 de Mayo de 2017, de https://www.w3.org/XML/

[45] Marriot, K. (13 de Febrero de 2009). *SVG Layout: Cases and Requirements*. Recuperado el 20 de Junio de 2017, de https://dev.w3.org/SVG/modules/layout/publish/SVGLayoutReqs.html

[46] Gson. (s.f.). Recuperado el 2 de Mayo de 2017, de https://sites.google.com/site/gson/gson-user-guide