UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Formrunner: una solución multiplataforma para el despliegue de formularios inteligentes en centros médicos

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Óscar Ferrando Sanchis
Tutor: Santiago Escobar Román
Tutor externo: Jorge Hortelano Otero

Curso 2017-2018

Resum

L'objectiu del present treball és el disseny i desenvolupament d'una aplicació que permeta millorar la recollida de dades dels clients de centres mèdics. El desenvolupament s'ha dut a terme durant unes pràctiques d'empresa en *BiiT Sourcing Solutions, S.L.* El present treball permetrà a l'empresa millorar el desplegament dels seus formularis agilitzant el procés.

Donat el nivell de desenvolupament assolit, *BiiT Sourcing Solutions, S.L.* planeja integrar l'aplicació dins de l'ecosistema d'aplicacions de l'empresa, permetent que a partir d'una especificació d'un formulari, aquest sigui representat i gestionat convenientment. L'aplicació serà desplegada a clients reals, en aquest cas centres mèdics. Actualment es troba en proves en un client situat als Països Baixos.

Paraules clau: formulari, Angular, API, Node, servici web, Android

Resumen

El objetivo del presente trabajo es el diseño y desarrollo de una aplicación que permita mejorar la recogida de datos de los clientes de centros médicos. El desarrollo se ha llevado a cabo durante unas prácticas de empresa en *BiiT Sourcing Solutions, S.L.* El presente trabajo permitirá a la empresa mejorar el despliegue de sus formularios agilizando el proceso.

Dado el nivel de desarrollo alcanzado, *BiiT Sourcing Solutions, S.L.* planea integrar la aplicación dentro del ecosistema de aplicaciones de la empresa, permitiendo que a partir de una especificación de un formulario, éste sea representado y gestionado convenientemente. La aplicación será desplegada en clientes reales, en este caso centros médicos. Actualmente se encuentra en pruebas en un cliente situado en los Países Bajos.

Palabras clave: formulario, Angular, API, Node, servicio web, Android

Abstract

The objective of the present project is the design and development of an application that allows to improve the collection of data from the clients of medical centers. The development has been carried out during a company internship in textitBiiT Sourcing Solutions, S.L. This project will allow the company to improve the deployment of its forms speeding up the process.

Given the level of development achieved, textitBiiT Sourcing Solutions, S.L. plans to integrate the application within the ecosystem of applications of the company, allowing that from a specification of a form, this form is represented and managed conveniently. The application will be deployed in real clients organizations, in this case medical centers. Actually is being tested in a client located in the Netherlands.

Key words: form, Angular, API, Node, web service, Android

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

El uso de formularios es una de las formas más comunes para la recogida de datos, así como para su posterior tratamiento y análisis. Actualmente se hacen uso de ellos para cualquier tipo de acción cotidiana en la web, como crearse una cuenta en un servicio concreto, rellenar una solicitud, utilizar un buscador etc. Además, los formularios web tienen la ventaja que pueden aplicar validaciones a las entradas de datos haciéndolos más seguros y eficientes.

Como en casi todos los ámbitos lo digital va ganando posiciones sobre el papel y con los formularios la cosa no cambia. Casi todos los servicios que anteriormente se contrataban rellenando formularios a papel, como abrir una cuenta en un banco, hoy en día se hacen a través de la web. Es por eso por lo que el desarrollo de servicios que se encarguen de generarlos/analizarlos son de gran importancia.

1.1 Motivación

BiiT Sourcing Solutions, S.L. implementa un sistema de *Bussiness Inteligence*, el cual está formado por tres etapas: la extracción de datos, la transformación de los mismos para poder ser gestionados y el estudio de estos para poder tomar decisiones respecto a los resultados de una forma más eficiente. Este proyecto se centra en la primera etapa, la extracción o recogida de datos.

Existen muchas aplicaciones en la actualidad que se encargan de generar y gestionar formularios como se explicará en el Capítulo 2, pero ninguna de ellas se adapta completamente a las necesidades de la empresa. Por ello se requiere diseñar un software que consiga romper esta barrera obteniendo distintas ventajas: modificación, adaptabilidad, venta, etc.

Otro de los motivos del desarrollo de este TFG es conseguir integrarlo con las otras aplicaciones de la empresa, consiguiendo así un sistema multiplataforma.

1.2 Objetivos

El objetivo de este trabajo es complementar aplicaciones ya existentes en la empresa utilizadas para facilitar la toma de datos de pacientes, así como sus examinaciones médicas. Para ello se pretende crear una librería que se encargue de la gestión y representación de los formularios, así como el posterior envío de los datos a las otras aplicaciones de la empresa. Dicha librería permitirá realizar acciones como previsualizar, guardar, rellenar,

etc. tanto en la parte cliente, como en la parte servidor. El nombre elegido para la aplicación es: *Formrunner* como se indica en el título del proyecto.

1.3 Estructura de la memoria

La memoria se estructura de la siguiente forma:

- Primero (Capítulo 2) se explicará el contexto en el que se desarrolla el presente trabajo.
- Después (Capítulo 3) se realizará una breve introducción a las metodologías y tecnologías utilizadas.
- A continuación, se expondrán (Capítulo 4-5) los requisitos del sistema, así como el diseño y arquitectura de este.
- Se continuará (Capítulos 6-7) con la integración de la solución desarrollado con las diferentes aplicaciones de la empresa y las pruebas realizadas para comprobar que se satisfacen los requisitos inicialmente propuesto.
- En el Capítulo 8 se realizará un breve recorrido por un caso de uso práctico.
- Para concluir se analizará (Capítulo 9) el estado final del trabajo, posibles mejoras y cumplimiento e objetivos.

CAPÍTULO 2
Contexto

Este proyecto ha sido realizado durante las prácticas de empresa en *Biit Sourcing Solutions, S.L.* La empresa trabaja ofreciendo servicios para centros médicos situados generalmente en los Países Bajos. Actualmente la empresa dispone de varias aplicaciones desarrolladas por ellos mismos.

Una de las aplicaciones desarrollada por la empresa permite el diseño de formularios con flujos complejos y capaces de obtener información de otras aplicaciones mediante el uso de servicios web. Hasta ahora la empresa se proveía de un servicio propietario, para la representación y gestión de los formularios. Dicho servicio tiene un coste de licencia, no es particularmente eficiente y además no es suficientemente flexible para las necesidades de la empresa. Por eso se decide desarrollar una aplicación propia que se pueda adaptar a sus necesidades.

Por tanto, este proyecto tiene como objetivo el diseño y desarrollo de una aplicación que a partir de la especificación de formularios de entrada de datos dada por otras aplicaciones de la empresa sea capaz de mejorar la funcionalidad ofrecida por el servicio que usa actualmente. De esta forma se poseerá una librería que podría integrarse en las aplicaciones web, móviles y otros sistemas. Al pasar a ser una aplicación propietaria de la empresa se consiguen las ventajas inherentes a la misma:

1. La posibilidad de venta de esta en forma de licencia.
2. Facilidad para modificar o añadir nuevas funcionalidades.
3. Al estar desarrollada para esa empresa mejor y más fácil integración con las aplicaciones de la empresa, así como con la imagen corporativa de los clientes gracias a la posibilidad de cambiar sus estilos.

2.1 ¿Quién es BiiT?

El desarrollo de este proyecto se ha llevado a cabo en colaboración con la empresa BiiT Sourcing Solutions,S.L. [3]. BiiT es una empresa situada en Valencia cuyas siglas provienen de *Bussines Inteligence* [4] que consiste en transformar en los datos en información y la información en conocimiento, de forma que se pueda optimizar el proceso de toma de decisiones en los negocios, e *Information tecnology* [5] que es el uso de ordenadores, alamcenamiento, redes u otros sistemas físicos, infraestructura y procesos para crear, procesar guardar e intercambiar todo tipo de datos digitales.
Se encarga de ofrecer soporte y desarrollar aplicaciones y actualmente está enfocada en la gestión de centros médicos privados generalmente en el mercado holandés. Algunos ejemplos de dichas aplicaciones se verán en la siguiente sección.

2.2 Aplicaciones de la empresa

A continuación, se van a citar sólo algunas aplicaciones que ha desarrollado *BiiT* y que tienen relación con este proyecto.

Webforms

Permite el diseño de formularios con flujos complejos y capaces de obtener información de otras aplicaciones mediante el uso de servicios web. Dichos formularios se usan para la toma de datos para los pacientes o clientes que van a un centro médico.

Agile Bussiness sCenario Designer (ABCD)

Se encarga de aportar la lógica de los formularios definidos con *Webforms*. Para ello se generan reglas que se procesarán posteriormente en *Drools* [7] que es un sistema de gestión de reglas de negocio que utiliza un motor de reglas basado en inferencia de encadenamiento hacia adelante (*forward chaining*) y de encadenamiento hacia atrás (*backward chaining*).

***Ik ga voor gezondheidswinst* (IGROW)**

Es una aplicación móvil de la empresa que ha sido desarrollada como TFG por un antiguo alumno, Alejandro Melcón [1]. Esta aplicación que permite a los clientes tener retroalimentación después de haber pasado por el centro médico, observar su progreso y trabajar todos los días en mejorar su salud. Un sistema al que cualquier cliente de una clínica sea capaz de acceder sin complicaciones extras y con el cual se pueda motivar el desarrollo de cada individuo de manera personalizada. Está disponible para su descarga para los principales sistemas, como *Android* e *iOS*.

Sport Medi Score

Abreviado *SMS*, es el eje central donde convergen todas las anteriores. Es la aplicación que gestiona los procesos de un centro médico, como, por ejemplo, generar citas, generar informes con los resultados de las examinaciones, comunicar con los clientes, generar facturas etc. A esta aplicación es a la que mi aplicación enviará los datos para iniciar los procesos de negocio. Así mismo también contiene los datos de los clientes que los servicios web podrán obtener.

2.3 Estado del arte

En la actualidad existen muchos servicios que permiten la gestión/creación de formularios, haciendo uso de una interfaz gráfica. Vamos a describir tres de ellos entre los que se encuentra el usado por la empresa.

Vaadin [33]

Vaadin se ejecuta en modo servidor y requiere de una gran conexión. Cada vez que ocurre un cambio en la interfaz, éste se comunica con el servidor ralentizando el proceso. En nuestro caso requiere la integración con la aplicación móvil y si se está situado en una zona con una conexión más lenta el rendimiento se verá afectado. Así mismo los datos móviles pueden ser limitados haciendo que los posibles se muestren reticentes al uso de la misma. En al figura 2.1 se muestra una imagen de *Vaadin Designer*.

Formularios de Google

Presenta menos problemas que Vaadin, ya que tiene parte de ejecución en el cliente, pero sigue presentado comunicación con el servidor. En la figura 2.2 se muestra una captura.

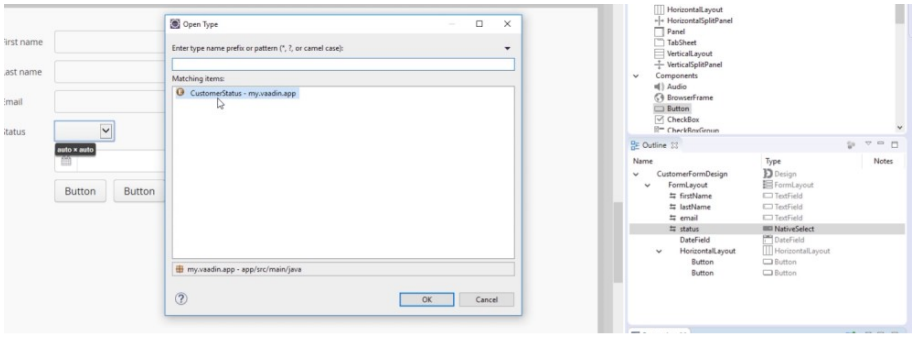

Figura 2.1: Imagen Vaadin Designer

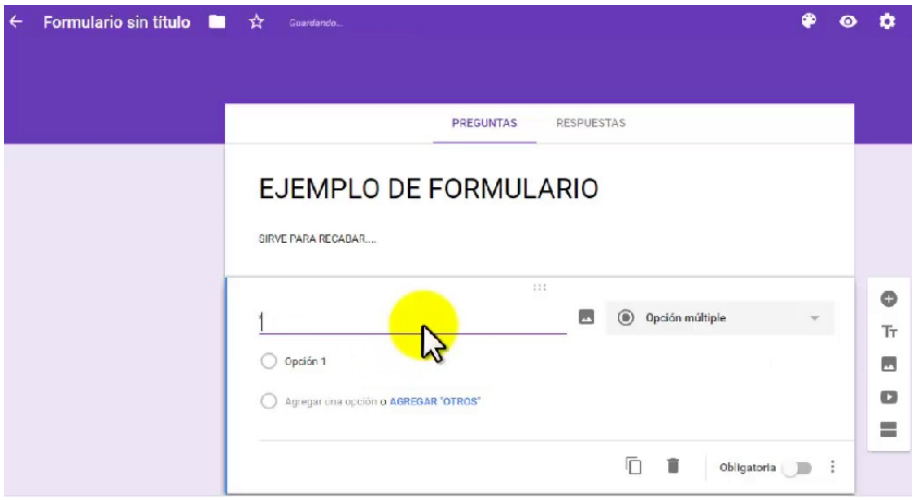

Figura 2.2: Imagen Google Forms

Estos dos servicios tienen en común que para poder generar formularios hay que saber usar las plataformas, cosa que puede no resultar sencilla para una persona que no posea conocimientos técnicos. Así mismo, servicios como *Google Forms*, no permiten la ejecución de reglas de *Drools* [7], crucial en la transformación de datos para la toma de decisiones. Además, *Google forms* no permite la creación de distintos flujos en los formularios, es decir, dependiendo de las respuestas que se muestren u oculten distintas preguntas llegando a distintos finales.

Actualmente la empresa hace uso del servicio *Orbeon* [6] para la gestión/generación de formularios. Dicho servicio es una solución para construir y desplegar formularios con sistemas de validación complejos haciendo uso del estándar *Xforms* [8]. Este servicio es más potente que los anteriormente descritos ya que permite la ejecución de reglas de *Drools* y la creación de distintos flujos, es por ello que tiene un coste de licencia elevado. No es particularmente eficiente, ya que igual que con los anteriores servicios se comunica con cada cambio con el servidor, lo cual consume ancho de banda y ralentiza la obtención de resultados. Además no es suficientemente flexible para las necesidades de la empresa. Un ejemplo de esto último es la imposibilidad de personalizar los estilos utilizando por ejemplo los colores corporativos de la empresa donde se despliegue el servicio. En al figura 2.3 se muestra una capturra de Orbeon.

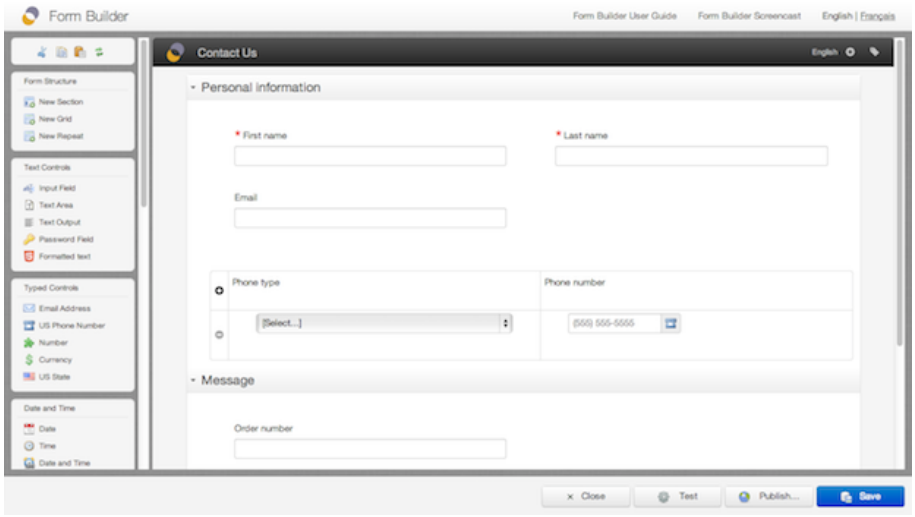

Figura 2.3: Imagen Orbeon

2.4 Problemática

El inicio de todo el proceso de negocio de los servicios que la empresa ofrece a los centros médicos empieza con un potencial cliente que rellena un formulario web con sus datos personales junto con otros posibles para que el centro médico reciba la solicitud y empiece todo el proceso, es decir, crear el cliente, crear una cita programada para ese cliente, realizar las examinaciones que ha seleccionado a través del formulario... Como se ha mencionado anteriormente la aplicación utilizada por la empresa y por que se paga una subscripción no se adapta a las necesidades de los clientes y carece de la flexibilidad necesaria. Tres de los principales problemas de la aplicación usada actualmente serían:

- Imposibilidad de cambiar los estilos y adaptarlos a la imagen corporativa de los clientes.
- Baja velocidad en la resolución de los flujos (ver Sección 5.3.2) de los formularios. Esto es debido como se ha explicado anteriormente a la comunicación con el servidor.
- Además añadir nueva funcionalidad resulta especialmente complejo ya que hay que adaptarse a las especificicaciones del software contratado o en algunos es casos imposible si no está implementado en dicho software.

CAPÍTULO 3
Metodología y tecnología

La empresa cuenta con una metodología de trabajo específica para la gestión interna. En esta sección también se presentarán diversas tecnologías utilizadas en el presente trabajo y el porqué de su elección.

3.1 Metodología

Scrum

Scrum [35] es un medología ágil [20] preparada para trabajar en proyectos complejos y entornos dinámicos. Se aplican una serie de buenas prácticas que permiten mejorar la colaboración del equipo implicado en el desarrollo y obtener los mejores resultados.

Durante la realización de este proyecto se han ido realizando *sprints* o iteraciones, los cuales consisten en períodos normalmente de entre dos y tres semanas de duración durante los cuales se desarrollan nuevas funcionalidades de la aplicación y en los que al final de los mismos se entrega un producto usable.

Durante cada *sprint* se realizaban las cuatro fases principales:

-Planificación: Se lleva a cabo antes de cada *sprint* y en ella se prepara qué se va a desarrollar durante el *sprint*.

-Reunión diaria: Diariamente, normalmente al empezar el día, el quipo de desarrollo se reune y comenta qué se ha realizado durante el día anterior, qué se va a realizar el mismo día y si se ha tenido algún problema.

-Revisión: Se realiza al finalizar el *sprint* y consiste en una reunión con el cliente en la que se le muestran los requisitos y funcionalidades completados durante el *sprint*.

-Reunión de retrospectiva: También se realiza al finalizar el *sprint* y en ella el equipo analiza cómo ha sido la forma de trabajar durante el mismo. Qué cosas han podido ir mal, qué puede mejorarse, por qué se han conseguido o no los objetivos marcados durante la planificación, etc.

3.2 Tecnología

Durante el desarrollo del proyecto se ha hecho uso de distintas tecnologías utilizadas por la empresa en las que se ha tenido que ir adquiriendo conocimiento de las mismas.

3.2.1. JavaScript

JavaScript [21], es un lenguaje de programación ligero e interpretado. Está oriendado a objetos basados en prototipos y es débilmente tipado. Su uso está limitado normalmente en los navegadores o lado del cliente pero también se usa en otros si navegador como Node (ver Sección 3.2.4)

3.2.2. TypeScript

TypeScript [22] extiende la sintaxis de *JavaScript* (es un super conjunto de este), es decir, que cualquier código *JavaScript* existente debería funcionar sin problemas. En este caso se utilizará junto con el *framework* de Angular (ver Sección 3.2.3). La principal ventaja de *TypeScript* viene dada por su nombre y es la posibilidad de uso de tipos de datos y clases.

3.2.3. Angular

Angular [24] es un *framework* para aplicaciones web desarrollado en *TypeScript*, el cual se utiliza para crear y mantener aplicaciones web de una sola página. Haciendo uso del Modelo Vista Controlador permite que el desarrollo y las pruebas sea más sencillo.

3.2.4. Node

Node.js [23] es un entorno en tiempo de ejecución multiplataforma, el cual se utiliza normalmente en la capa del servidor. Es asíncrono y utiliza una entrada y salida de datos sin bloqueos y orientado a eventos. Pese a ser código JavaScript no se ejecuta en el lado del cliente o navegador sino en el servidor.

3.2.5. Verdaccio

Verdaccio [34] se encarga de gestionar los paquetes o los *release* en *Node*. Es un registro local de *npm* [25] simple y que no necesita configuración. En este registro es donde se publican las dependencias para que puedan ser descargadas y utilizadas por otros proyectos.

3.2.6. Git

Git [26] es un sistema de control de versiones que está diseñado para mejorar la eficiencia y confiabilidad en el mantenimiento de distintas versiones de código fuente. Permite llevar un control de los cambios que se realizan sobre el mismo.

Existen una serie de comandos básicos para utilizarlo:

Git add
Marca el archivo que se le pasa como parámetro a staged y lo deja listo para un commit, si hay más de un archivo y se quieren marcar todos se usará el *flag* "–all".

Git commit
Confirma todos los archivos en stado *staged* o listos para el *commit* y crea un punto de guardado. Normalmente con el *flag* -m <mensaje>se asocia un pequeño mensaje descriptivo de qué se ha hecho o qué cambios se han realizado en ese *commit*.

Git push
Sube todos los cambios realizados en el directorio de trabajo local al servidor remoto. Esto permite que estén disponibles para otros desarrolladores.

Git merge
Combina los cambios de dos ramas. Normalmente para crear una nueva funcionalidad, solucionar un error, etc. Se creará una rama nueva, se realizarán los cambios y test pertinentes en la misma y posteriormente se fusionará con la rama principal.

3.2.7. Jenkins

Jenkins [27] es un software de integración continua [30] y desarrollado en Java. Es la herramienta que la empresa utiliza para realizar los test de integración, unitarios y permite automatizar los despliegues de las aplicaciones. Esto último se explicará en más profundidad en capítulos posteriores (Capítulos 5 y 7).

CAPÍTULO 4
Análisis de requisitos

4.1 Introducción

Durante el desarrollo de este capítulo se van a especificar los diferentes requisitos del sistema, tanto funcionales como no funcionales siguiendo el estardar IEEE 830 [16], el cual especifica una serie de prácticas y pasos, que se expondrán a continuación, a seguir para una correcta especificación de requisitos.

Propósito

El sistema que se ha de implementar tiene como propósito la mejora en la toma de datos por parte de los clientes para su posterior proceso en los centros médicos así como mejorar el despliegue de los formularios.

Alcance

Se debe diseñar e implementar el servicio *Formrunner* que permitirá iniciar el proceso de negocio en el sistema *Sport Medi Score* (ver Sección 2.2). Cualquier cliente debe poder acceder al *Formrunner* a través de la red.

4.2 Descripción general

En esta segunda sección se describen las principales funcionalidades del sistema de forma general.

Perspectiva del producto

La principal característica del *Formrunner* es poder representar los formularios complejos que vendrán a partir de una especificación de *Webforms* (ver Sección 2.2). El *Formrunner* estará diseñado para poder trabajar tanto en entornos web como móviles. Para poder ser utilizado de forma rápida, eficaz y sencilla.

Funciones del producto

Las principales funcionalidades exigidas por la empresa para la aplicación son:

- Visualización de formularios provenientes de *Webforms*.
- Gestión de las respuestas para que puedan ser interpretadas por *Sport Medi Score*.
- Gestión de las peticiones a los servicios web de *Sport Medi Score*.
- Validación de las respuestas para prevenir errores y asegurar que los datos enviados tienen el formato correcto.

Restricciones

Los requisitos extra exigidos por la empresa son:

- La interfaz podrá ser usada desde la web y la aplicación móvil.
- Tecnologías: *Angular 4*, *Node*, *JavaScript* y *TypeScript* (ver Sección 3.2).
- Se utilizará el *Component Based Development* [17] ya que facilita el desarrollo y la adición de nuevos componentes al tener bajo acoplamiento.
- Al tratarse datos personales confidenciales, no se gestionarán contraseñas y se usará un sistema de firma digital.

Suposiciones y dependencias

La empresa quisiera que también se tuvieran en cuenta los siguientes requisitos adicionales.

- Se genera una librería que podrá exportarse y usarse en otros sistemas.
- El servidor web deberá estar disponible para que la aplicación web funcione correctamente.
- El uso del producto en todos los aspectos será llevado a cabo por usuarios sin conocimientos técnicos por lo tanto se requiere sencillez de uso.

4.3 Requisitos específicos

4.3.1. Interfaces

Interfaces con el usuario.

Se debe poder navegar por las categorías de los formularios, así como gestionar el envío.

Interfaces con el software

Para que las aplicaciones con las que se va a integrar la librería (cliente y aplicación móvil) puedan interactuar con ella se deberá contar con una API.

Interfaces de comunicación

Para gestionar las peticiones a los servicios web y el envío de datos ser hará uso del protocolo *HTTP* [18] o en caso de envío de datos sensibles su contraparte más segura *HTTPS* [19].

4.3.2. Funcionales

Los requisitos funcionales se encargan de definir un comportamiento específico de un sistema software. Se han presentado los siguientes:

- Leer la especificación del formulario: a partir del documento JSON generado por Webforms, éste será cargado en la libería (ver Capítulo 6).
- Categorías: El modelo de presentación principal serán las categorías, es decir, se mostrarán categorías completas.

- Textos descriptivos: Se podrán mostrar textos para explicar cualquier función o pregunta.
- Pregunta tipo *input*: Se podrán representar preguntas de tipo *Input*.
- Pregunta textarea: Similares a las preguntas de tipo *Input* pero de mayor tamaño.
- Pregunta dropdown o lista: Se podrán mostrar listas de selección unica.
- Pregunta *multicheckbox* o elección múltiple: Se podrán representar preguntas de selección múltiple.
- Pregunta *radio button*: Se podrán representar preguntas de tipo *radio button*.
- Pregunta slider: Se podrán representar preguntas de tipo *slider* o barra de selección.
- *SystemFields*: Existe un tipo especial de pregunta oculto que se rellena automáticamente con parametros que se le envían a la aplicación (ver Sección 5.3.3).
- Grupos: Las distinstas preguntas/textos se podrán agrupar en grupos.
- Grupo repetibles: Existe un tipo especial de grupo que se puede duplicar al pulsar un botón.
- Subrespuestas: Las respuestas de las preguntas de tipo radio y selección múltiple podrán mostrar más respuestas si la respuesta padre está seleccionada.
- Valor por defecto: Las preguntas podrán mostrar un valor por defecto proveniente de la especificación.
- Tablas: Las preguntas tendrán un tipo de representación especial en formato de tabla.
- Navegación entre las categorías: Se podrá navegar por las distintas categorías.
- Validación de las distintas preguntas: Las preguntas podrán poseer validadores especiales que se comprobarán automaticamente (ver sección 5.3.1)
- Descripciones: Las preguntas y respuestas podrán poseer una pequeña descripción que ayude a rellenarlas.
- Servicios web: Algunas preguntas podrán lanzar llamadas a servicios web para rellenar partes de los formularios automáticamente.
- Flujo: Los formularios podrán ocultar o mostrar categorías/grupos/preguntas en función de las respuestas(ver Sección 5.3.2)
- Exportar respuestas: Las respuestas se exportarán al formato aceptado por *Sport Medi Score* para su posterior análisis y gestión.
- Leer respuestas: Se podrán visualizar las respuestas de un formulario previamente rellenado y enviado.
- Personalizar estilos: Dependiendo de la organización donde esté desplegada la aplicación se podrán personalizar los estilos de la misma, por ejemplo el logo y los colores corporativos de la organización.
- Idiomas: mediante un parámetro se podrá especificar el idioma de la interfaz (castellano, inglés, holandés...)

- Pistas sobre el valor de las preguntas: Si una pregunta tiene un tipo específico de datos aceptados (número positivo) se mostrará una pequeña pista.
- Desactivar la interfaz: Mediante un parámetro se podrá desactivar la interfaz.
- Desactivar preguntas: Algunos campos rellenados automáticamente se podrán desactivar para evitar su modificación.
- Desplegar formularios: Se podrán desplegar formularios automáticamente desde *Webforms*.
- Previsualizar: Se podrán previsualizar los formularios generados desde *Webforms*.

Requisitos de rendimiento

- Los envíos se han de realizar en menos de 7 segundos.
- Los flujos deberán de resolverse en menos de 0,5 segundos.

4.3.3. Atributos

Disponibilidad

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

Portabilidad

La librería y el cliente se tiene que poder integrar con otras aplicaciones por lo tanto ha de usarse un lenguaje de programación que la facilite. La interfaz se adaptará a la resolución y tamaño de las pantallas donde se ejecute.

Verificación

Para poder asegurar la calidad y el correcto funcionamiento del sistema se implementarán pruebas automáticas.

CAPÍTULO 5
Diseño y arquitectura

5.1 Diseño

Una vez especificados los requisitos se va a procedor al diseño del modelo de datos de la aplicación. En la siguiente figura se puede ver el diagrama de clases de la aplicación.

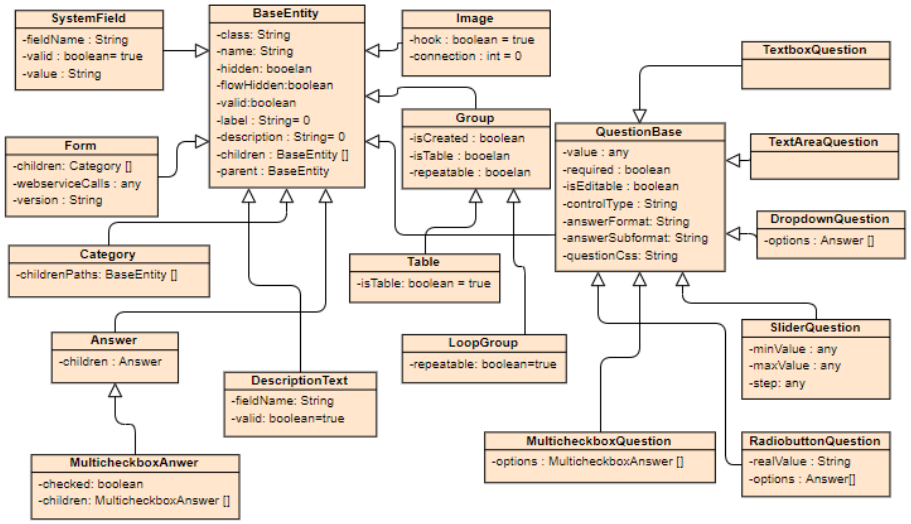

Figura 5.1: Diagrama de clases

5.2 Arquitectura

Una vez especificados los requisitos del sistema y llevado a cabo un diseño de éste, el proyecto se dividirá en tres subsistemas.

1. Una librería que se encargará de la gestión/representación de los formularios, que podrá importarse para su uso en IGROW.

2. Una aplicación autocontenida o cliente que haciendo uso de la librería permitirá, haciendo uso de una interfaz rellenar los formularios, enviarlos y con ello iniciar el proceso de negocio descrito anteriormente.

3. Un servidor que gestionará las llamadas a servicios web, así como los accesos a la base de datos de la aplicación.

Como se ha explicado anteriormente se ha usado la metodología *Scrum* para el desarrollo del proyecto por lo que la realización del mismo se ha realizado por *sprints* coincidiendo con los distintos subsistemas.

Durante el primer sprint se realizó una pequeña formación en las distintas tecnologías explicadas anteriormente, así como la preparación del proyecto.

Para la creación de una base para el proyecto se ha usado el *framework* de *Angular-CLI* [9] el cual permite mediante una simple orden `ng new Formrunner` crear un esqueleto vacío, pero funcional de una aplicación con la siguiente estructura.

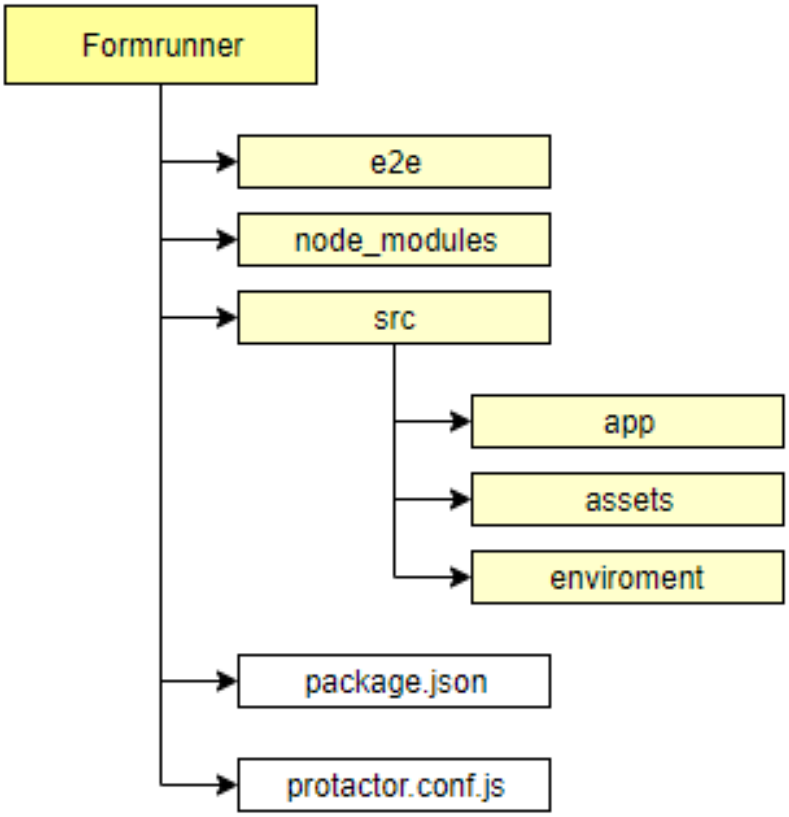

Figura 5.2: Estructura proyecto Angular-CLI

En la Figura 5.2 se puede ver una vista simplificada de un proyecto creado con Angular-CLI. En el directorio e2e es donde se encuentrar los test de integración (ver Capítulo 7), en la carpeta *node modules* se encuentran todas las dependencias de la aplicación. En *src* o *source* es donde estará situado el código fuente. El archivo *package.json* especifica en nombre del proyecto, versión, tipo de licencia, dependencias, etc.

5.3 Librería

Una vez creada la estructura del proyecto, se pasa a integrarla con el ciclo de vida de las aplicaciónes de la empresa para conseguir lo que se conoce como *continuous integration* o integración continua. La integración continua permite, a base de hacer integraciones automáticas muy a menudo, detectar fallos antes y solucionarlos. Esto se consigue mediante la herramienta *Jenkins* (ver Sección 3.2.7). Para ello se crea una tarea en *Jenkins*, como se ha mencionado en el Capítulo 3, que cada cierto tiempo revisa si una rama en

concreto del proyecto (*development* en este caso) en *git* tiene algún cambio. Una vez se detecta un cambio se lanza la tarea que consiste en pasar los dos tipos de test definidos, unitarios y de integración (ver Capítulo 7), aumentar la versión del proyecto y publicar la librería en el repositorio *npm* privado de la empresa *Verdaccio* para después poder descargarse como dependencia en los demás proyectos donde quiera integrarse (aplicación y aplicación móvil).

Después de generar el proyecto e integrarlo con el ciclo de vida se procede a la creación de los modelos que se encargarán de gestionar todo el formulario, descritos al inicio de esta capítulo.

La librería será el núcleo central de la aplicación. Es decir, en ella se gestionarán la mayor parte de los servicios que proveerá el *Formrunner*. Para ello se ha dividido el desarrollo de la misma en distintos componentes y servicios que se encargarán de gestinarlo todo correctamente.

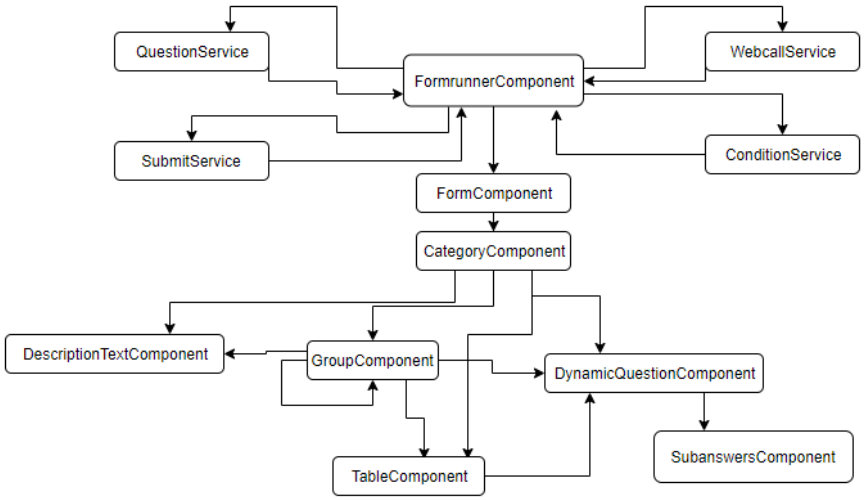

Figura 5.3: Diagrama de interacción de componentes y servicios de la librería

Primero es necesario crear el servicio de Angular que se encargará de leer la especificación del formulario a partir de un *JSON* proveniente de la aplicación *Webforms*. Esto se consigue mediante una llamada de tipo *HTTP* a la carpeta *assets* del propio proyecto la cual es accesible mediante este tipo de llamadas. Debido a que se trata de una llamada *http* su ejecución no es síncrona, es decir, puede tardar un poco en resolverse por lo tanto ha de tratarse como tal y hacer uso de **Observables** [10]. Estos permiten mediante un método subscribirse al resultado de la petición permitiendo gestionar las llamadas asíncronas fácilmente.

Una vez se consigue leer la especificación hay que representar las preguntas provenientes de ésta por lo tanto se crea otro servicio *Angular* el cual a partir del objeto leído anteriormente y haciendo uso de los modelos de datos explicados, genera un objeto de tipo *Form* el cual se propagará a los demás componentes.

Cada uno de los componentes se encarga de gestionar su propio objeto. El componente padre es el *FormComponent* al cual se le pasa como argumento el objeto de tipo *Form* y un número que indica la categoría que se tendrá que mostrar en ese momento. A partir de entonces observando el tipo de cada uno de los hijos o *children* del objeto se van pasando a cada uno de los componentes correspondientes de su gestión.

En última instancia quedan dos tipos de objetos por representar, las preguntas y los textos descriptivos. La gestión de las preguntas es competencia del *DynamicQuestionComponent* que observando el tipo de pregunta representa cada uno de los cinco tipos

disponibles. Durante el desarrollo de este componente se encontraron dificultades para la gestión de preguntas de tipo selección múltiple y se tuvo que cambiar el modelo de estas para poseer una propiedad *checked* o marcado que permitiera cada vez que se cargara el componente representar correctamente las preguntas seleccionadas. Esto fue debido a la imposibilidad de usar la propiedad de angular *NgModel* que permite gestionar más fácilmente las respuestas de un formulario.

Como se ha mencionado en el Capítulo 4, las categorías/grupo/preguntas podían tener imágenes. Estas imágenes se especifican en *Webforms*, se les pude especificar el ancho y largo a mostrar y pueden ser de dos formatos:

1. A partir de una URL donde está alojada la imagen.
2. Una imagen en la especificación en formato Base64.

Durante el desarrollo de esta funcionalidad se encontraron diversos problemas. Uno de ellos fue que cuando la imagen provenía de una url no se mostraba ya que saltaba un error de *URL not safe*. Para poder solucionarlo se tuvo que crear un *pipe* [31] o tubería que se encargara de sanear la *url*.

Una vez representados todos los tipos de objetos: grupos, grupos repetibles, preguntas, textos, etc. se pasa a la validación de estos.

5.3.1. Validación de campos:

La validación automática de los campos es uno de los pilares para la gestión correcta de cualquier formulario. *Webforms* permite especificar distintos tipos de validadores específicos que permiten que los valores enviados posteriormente a *Sport Medi Score* (ver Sección 2.2) son correctos y no causarán ningún problema ya que la validación de estos es competencia del *Formrunner*.

Cada tipo de pregunta posee validadores propios, pero existen preguntas que no se puede especificar la respuesta que se tiene que seleccionar ya que no tendría sentido que apareciera la misma. Un ejemplo de éstas serían las preguntas de tipo *RadioButton*, en las cuales, no se puede especificar qué respuesta se ha de seleccionar, pero en el caso de ser obligatoria, deberá tener seleccionado un valor. Lo mismo ocurre con las de selección múltiple, listas, *sliders* y las áreas de texto. Las preguntas de tipo *Input* sí que podrán poseer validadores al tipo de respuesta.

Como se ha explicado anteriormente *Webforms* posee varios tipos de validadores. Se puede especificar primero un formato de respuesta y un subformato del formato. En la figura 5.4 se incluye una captura del proceso de crear reglas de validación.

Formato tipo *text*: Permite cualquier tipo de carácter, carácter especial, número etc.

1. Text: al igual que el formato permite cualquier tipo de carácter.
2. Email: solamente se validarán campos con el formato <usuario>@<test>.<dominio>
3. Phone: permite solamente el uso de número y en este caso del caracter especial + para los prefijos internacionales.
4. IBAN: Valida las entradas al formato <ZZ00 0000 0000 0000 0000 0000>

Formato tipo *Number*: Solamente se aceptan números.

1. Number: permite cualquier tipo de número.

2. Float: solo se permiten números con coma flotante o punto.
3. Positive Number: solo se aceptan números positivos.
4. Negative Number: solo se aceptan números negativos.
5. Positive Float: solo se aceptan números con coma flotante positivos.
6. Negative Number: solo se aceptan números con coma flotante negativos.

Formato tipo *Date*: Permite formato del tipo DD-MM-AAAA. Esto se consigue con una etiqueta especial de *HTML* que crea una pregunta con un calendario para seleccionar una fecha.

1. Date: Se acepta cualquier fecha siempre que siga en formato anteriormente descrito.
2. Past: solo se aceptan fechas anteriores al día de hoy.
3. Future: solo se aceptan fechas posteriores al día de hoy.

Formato tipo *PostalCode*: Acepta formato de código postal. Debido a que este formato cambia entre países este validador no es muy fuerte ya que no se puede comprobar todos los formatos.

1. PostalCode: Acepta un formato de código postal.

La figura 5.4 muestra como se pueden seleccionar las reglas de validación en *Webforms*.

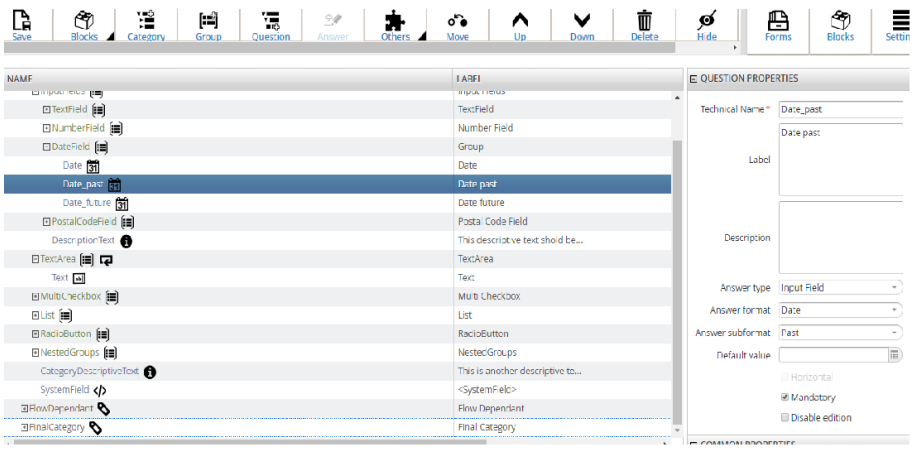

Figura 5.4: Creación de reglas de validación

Las validaciones se consiguen mediante la aplicación de expresiones regulares o *regex* [11] a las respuestas. Cada vez que se produce un cambio en el valor de una pregunta se lanza el método de validar de la pregunta en cuestión. En el caso de que la preguntar no sea obligatoria, si se rellena también se comprueba la validez del campo.

5.3.2. Flujo del formulario:

Otro de los pilares de la aplicación es la posibilidad de no solo representar formularios con una continuidad lineal, es decir aparecen todas las preguntas independientemente de las respuestas, sino que también permite que determinados campos, grupos, incluso categorías enteras no se muestren si el flujo no lo permite. Es decir, el flujo de un formulario

simple tendría una forma recta y el de un formulario con flujo una estructura en forma de árbol.

Como en los casos anteriores el flujo viene dado también en el *JSON* con la especificación del formulario proveniente de *Webforms*.

Los distintos tipos de flujo que se pueden especificar se explicarán a continuación. Existen dos tipos grandes que engloban todos los tipos de flujo que puede seguir una pregunta, los de tipo normal que es el mayor número de ellos y los de tipo *END FORM* que implican la llegada al fin de formulario de manera prematura. En los primeros se ha de especificar un origen y un destino mientras que en los segundos el destino se omite. Además, se pueden especificar condiciones para que se dé un flujo, es decir, se pueden especificar valores para que se siga un camino u otro. Los distintos tipos de condiciones son:

1. ==: El valor de una pregunta es igual a otro predefinido.
2. <: El valor de una pregunta es menor que uno predefinido.
3. >: El valor de una pregunta es mayor que uno predefinido.
4. >=: El valor de una pregunta es mayor o igual que uno predefinido.
5. <=: El valor de una pregunta es menor o igual que uno predefinido.
6. <>: El valor de una pregunta es distinto que uno predefinido.
7. BETWEEN: El valor de una pregunta se encuentra dentro de un rango.
8. DATE year: El valor de una fecha hasta hoy en años es mayor, menor, igual.
9. DATE month: El valor de una fecha hasta hoy en meses es mayor, menor, igual.
10. DATE day: El valor de una fecha hasta hoy en días es mayor, menor, igual.

La figura 5.5 muestra cómo se puede crear una regla de flujo.

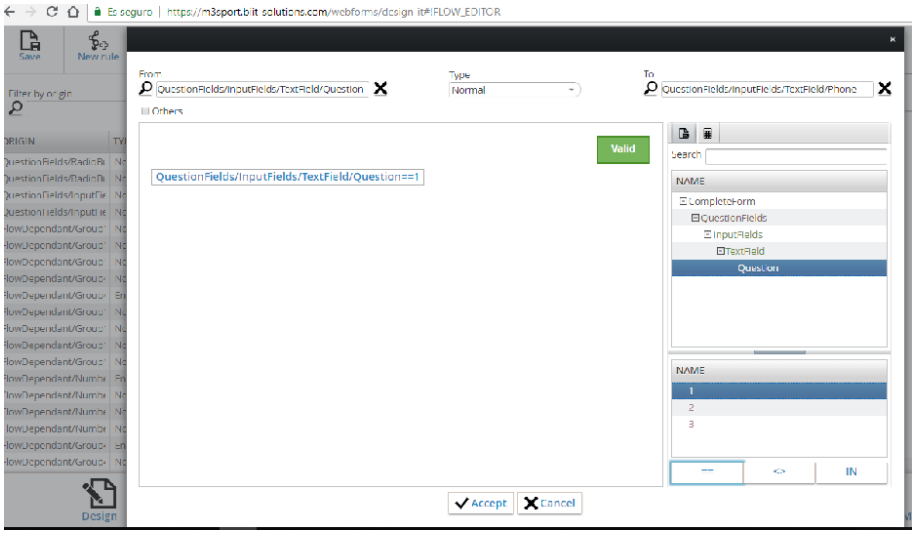

Figura 5.5: Creación de reglas de flujo

También se puede aplicar la negación **NOT** a una condición anterior. Otra de las cualidades de Webforms es que permite también concatenar condiciones mediante los operadores lógicos ***AND*** y ***OR***, así como agrupar los grupos mediante paréntesis. A continuación, se muestran dos imágenes, una con distintos tipos de flujos y otra con condiciónes anidadas.

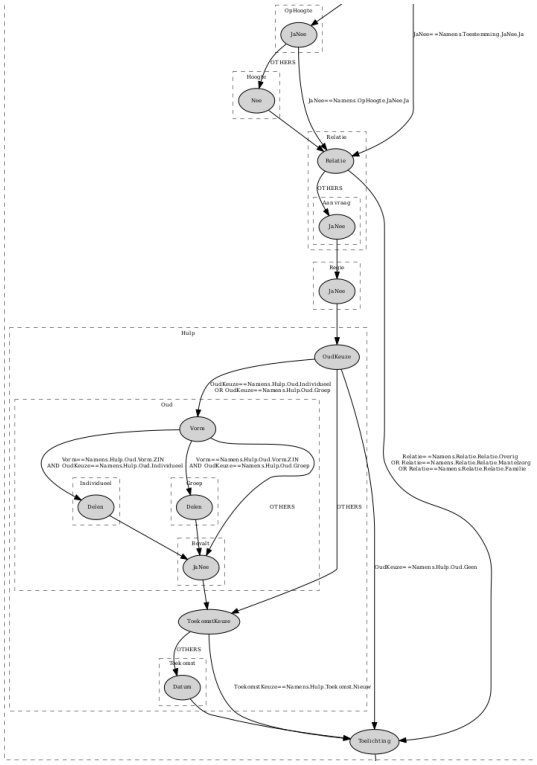

Figura 5.6: Ejemplo de flujo

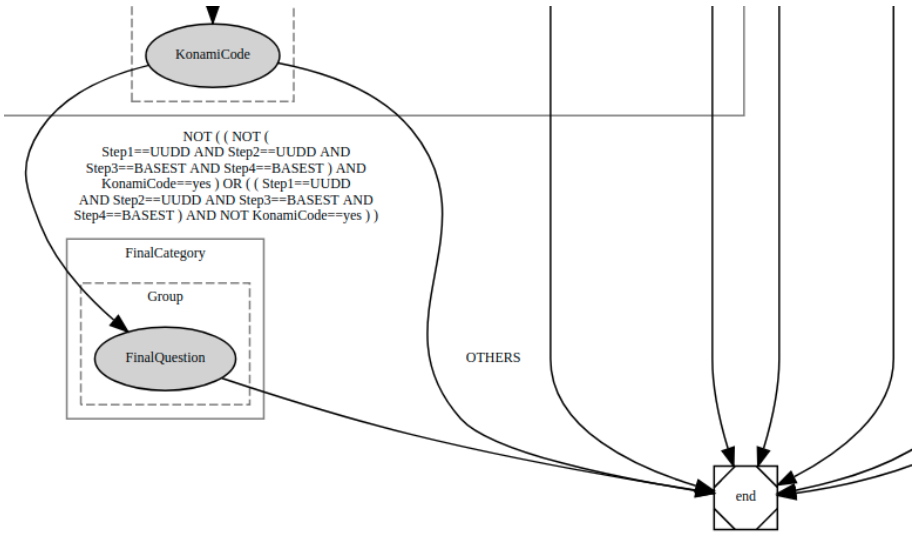

Figura 5.7: Flujo con multiples condiciones

También se pueden especificar flujos sin condición, es decir producir un salto en el flujo normal. En la especificación apararecerían en el siguente orden: pregunta de 'sí/no', pregunta en caso de 'sí', pregunta en caso de 'no' y pregunta siguiente a mostrar independientemente de las respuestas. En ese caso si se contestara 'sí' se necesitaría un salto en el flujo para evitar que apareciera la pregunta de respuesta 'no'.

Para no tener que especificar todos los posibles escenarios de condiciones *Webforms* posee un tipo de flujo especial llamado *others* que permite que "seguir un camino en caso de **no** cumplirse ninguna otra condición". Esto también se aprecia en la figura 5.7 donde no hay que especificar la condición contraria a las anidadas que se muestran.

Cualquier pregunta puede ser origen de flujo o condición de otro, pero siempre afectará a preguntas posteriores a la misma ya que si no se podrían crear bucles infinitos. De esto se encarga el validador de formularios de *Webforms*. En ambos casos, ya sea origen o condición se ha de lanzar el algoritmo desarrollado para resolverlo. El siguiente diagrama muestra cómo se resuelve el flujo desde el momento que se detecta un cambio en el valor de una pregunta hasta que se llega a un estado final.

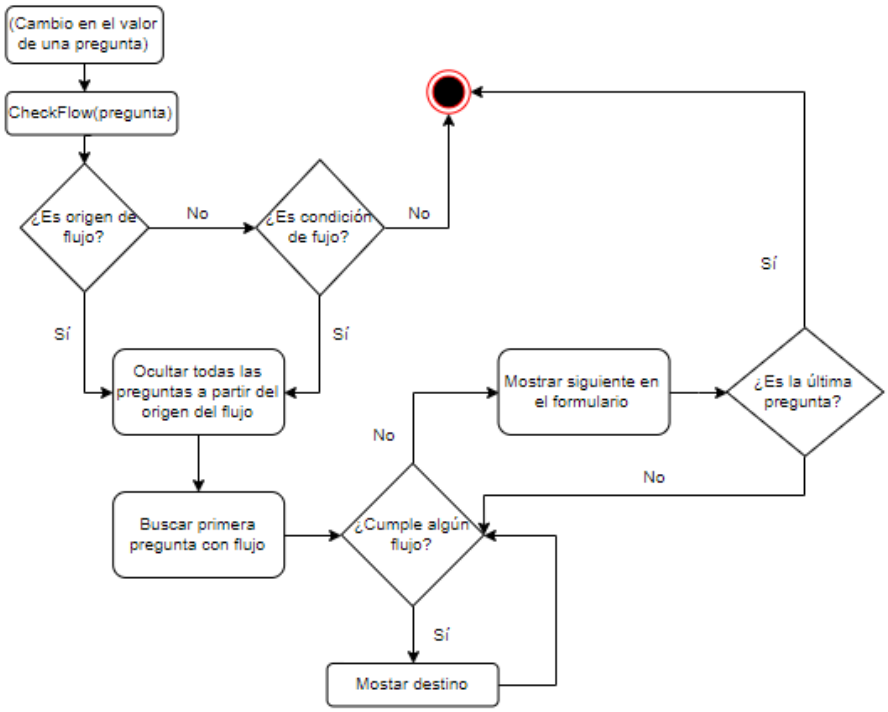

Figura 5.8: Diagrama resolución de flujo

5.3.3. Comunicación de los servicios web

El tercer punto fuerte de la librería es que permite la ejecución de llamadas a servicios web, las cuales también vienen en la especificación del formulario. En este caso el formato de una llamada a un servicio web es el siguiente:

- Una ruta a la pregunta que será el *trigger* o disparador del servicio web.
- Un *endpoint* donde se hará la llamada al servicio web.
- Un conjunto de las posibles respuestas que se recibirán, especificando en cada una de ellas el *path* o camino a la pregunta que hacen referencia.

En el caso de realizarse la llamada correctamente en la especificación del servicio también puede ponerse si una vez rellenado el campo automáticamente se deshabilita la edición de este. Es decir, por ejemplo, si se auto rellena el campo del email, la edición debería estar habilitada ya que este puede cambiar con el tiempo pero si se autor rellena la fecha de nacimiento se puede deshabilitar ya que esta no cambia y se previenen errores de cambios no deseados.

Los servicios web permiten agilizar el rellenado de los formularios a los clientes. Un cliente, la primera vez que se registra para una examinación en el centro médico ha de rellenar todos sus datos personales para que una vez enviados los datos, se cree el objeto cliente en la aplicación *Sport Medi Score*. Pero si el cliente vuelve otra vez pasado un tiempo al centro médico existen varias opciones para verificar que es el mismo cliente y no volver a crear otro objeto en la base de datos de *Sport Medi Score*.

Uno de ellos es el uso de un identificador único que posea el cliente como podría ser el uso de DNI o BSN como se llama en los Países Bajos. La problemática es que en España la entrega del número del DNI es considerado algo bastante cuotidiano mientras que en Holanda son bastante reacios a su entrega. Otra forma sería mediante el uso del formato usuario/contraseña, pero también implica añadir más dificultad a la gestión del formulario además de obligar al cliente a gestionar un usuario y una contraseña más. Como solución se crea un sistema de autenticación temporal, algo parecido a una firma digital. Mediante la aplicación móvil, se consigue un número único y autogenerado con una duración limitada, en este caso dos min, de seis dígitos que se pondrá en un campo del formulario. De esta forma se consigue de forma fácil (no hay que recordar ni usuarios ni contraseñas) y segura (únicamente el usuario posee el número temporal).

En el Capítulo 2 se ha mencionado la existencia de un tipo de pregunta especial llamada *SystemField*. Estos campos se autorrellenan con valores. Un ejemplo de ellos sería combinado con el uso de los servicios web. Si la llamada al servico web del user guard es exitosa hay un *SystemField* en el formulario que se rellena con el identificador único del cliente y es enviado a *Sport Medi Score*. De esta forma la aplicación sabe que ese cliente ya existe y lo sobreescribe con los nuevos datos.

En la figura 5.9 se observa el funcionamiento de los servicos web y en la 5.10 cómo se obtiene la clave expirable para su uso en el *Formrunner*.

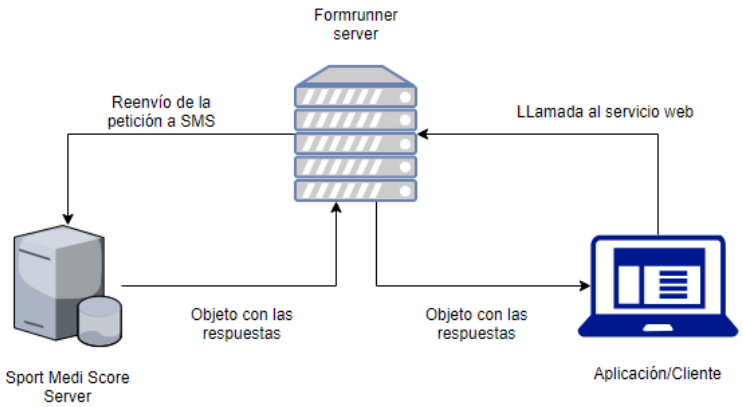

Figura 5.9: Llamada a los servicios web de SMS

El paso de la petición a través del servidor de la aplicación en lugar de realizarla directamente es porque así se evita gestionar las claves de autorización de uso de los servicios web en el cliente, pudiendo estas quedar expuestas. De esta forma al ser el servidor seguro es este el que se encarga de mantenerlas.

Por último cabe recalcar una ventaja que se consigue frente a *Orbeon*, está la conservación de los valores previamente escritos. Por ejemplo, una categoría es visible y se rellena. Si cambia un valor de una categoría anterior que hace que la categoría mencionada al principio se oculte. Si se vuelve a escribir el valor inicial, o cualquier otro que lleve a que esa categoría se muestre, los valores escritos anteriormente se conservarán.

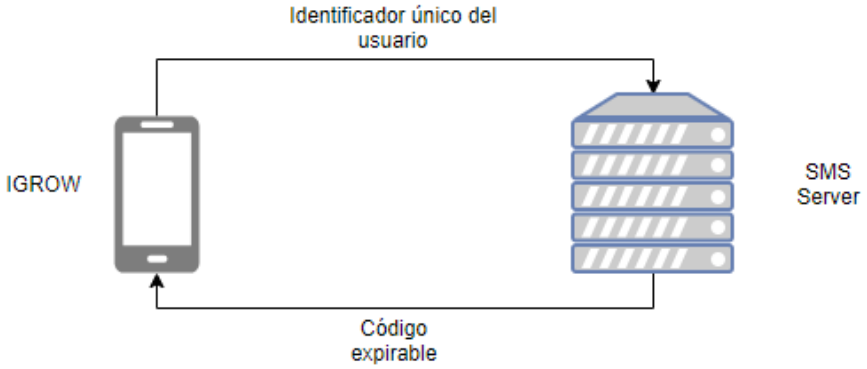

Figura 5.10: Obtención de la firma digital

5.4 Aplicación

El segundo componente del trabajo es el desarrollo de la aplicación que funcionará como una aplicación autocontenida. El cliente será el encargado de mostrar la interfaz de los formularios con los botones para permitir la navegación, así como el envío. El desarrollo del cliente se llevó a cabo durante el cuarto *sprint*.

De la misma forma que en la librería, haciendo uso de *Angular-CLI*, se ha generado un proyecto nuevo. El primer paso, una vez generado el proyecto, es importar la librería creada anteriormente. Para ello en el *package.json* se añade la dependencia de la libreria del *Formrunner*. El ordenador donde se desarrolla el proyecto ha de tener acceso al repositorio privado *npm* de la empresa (ver Sección 3.2.5) donde la librería está publicada. Con todo esto se ejecuta `npm install` y se descargan las dependencias. Una vez descargadas se añade el componente de la librería al *HTML* donde se quiera mostrar y ya está listo para usarse.

Los distintos formularios disponibles para su carga estarán disponibles en la carpeta *assets* del proyecto. Se podrá seleccionar cual cargar mediante un parámetro en la *URL* que llamará a la aplicación.

Se crea una interfaz simple que permita la navegación y envío de los formularios. Los botones destinados a la navegación y al envío están o no habilitados dependiendo de la validez del formulario o la categoría en la que se encuentra. Es decir, si una categoría esta correctamente cumplimentada el botón de siguiente se habilitará siempre y cuando no sea la última. En caso de no estar correctamente cumplimentada se deshabilitarán la navegación a la siguiente categoría. Únicamente se podrá enviar el formulario se todas las categorías que no estén ocultas por el flujo están correctamente cumplimentadas.

Del mismo modo y haciendo uso de un parámetro se podrá especificar la lengua en la que se quiere mostrar la interfaz (castellano, inglés, holandés). Esto se consigue haciendo uso del módulo *in18*.

Como se ha mencionado al principio de la memoria, uno de los problemas de *Orbeon* era su inflexibilidad en cuanto al tema de modificación de los estilos. Este por normal general no gusta a los posibles clientes que vayan a usar/comprar la aplicación ya que siempre se prefiere algo más personalizado. Mediante un parámetro en la *URL* especificando la organización a la que se van a enviar los datos, el cliente se encarga de cargar una hoja de estilos personalizada que modifica la apariencia de la aplicación. Es este caso se han creado distintos estilos haciendo uso de los colores corporativos de los logotipos de las organizaciones. De la misma forma se ha añadido el propio logotipo al cliente en

la parte superior izquierda. Esto se puede apreciar en la siguiente figura que muestra la versión final del cliente.

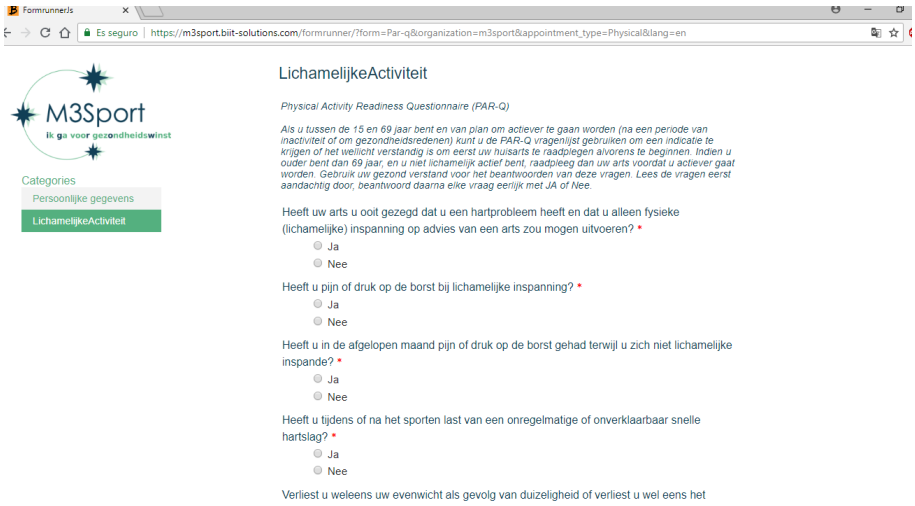

Figura 5.11: Figura con captura de la versión final del cliente

Al pulsar el botón de enviar la respuestas de los campos no ocultos se transforman al formato aceptado por *Sport Medi Score*. De la misma forma a partir de la categoría de datos personales se extraen los datos que se enviarán junto con las respuestas para que *Sport Medi Score* las procese.

5.5 Servidor de la aplicación

La última pieza del sistema es el servidor, encargado de gestionar las peticiones que recibe el cliente, así como los envíos a la aplicación, carga de respuestas, accesos a la base de datos, etc. El desarrollo de esta ha sido llevado a cabo durante el quinto *sprint*. Para el desarrollo del servidor se ha hecho uso de *Node* y *JavaScript*. El servidor estará dividido en cuatro archivos:

Logger
El *logger* permite visualizar la actividad del servidor. Mostrando mensajes en distintas situaciones, lo que ayuda a resolución de errores. Se usan distinos tipos de log. En este caso los más genéricos son: llamada a un servicio web y resultado de esa llamada.

Archivo de comunicación con la BD
Se encarga de gestionar los accesos a la base de datos realizando las consultas a la misma mediante llamadas *SQL* [14]. Así como de generar las tablas de la base de datos. En este caso se generan tres tablas. Una donde se guardan los resultados de los envíos de formularios, una donde se almacenan los formularios a presualizar y otra donde se guardan los formularios a los que la aplicación puede acceder. Esto se explicará en más profundidad en el Capítulo 6.

Servidor (index.js)
Es donde se gestionan todas las llamadas a servicios web de la aplicación, desde el envío, llamadas de carga de formularios, carga de respuestas etc.

Configuración
En el fichero de configuración se guardan las dirrecciones específicas donde está desplegada la aplicación para que esta sepa a qué dirección hacer la llamadas.

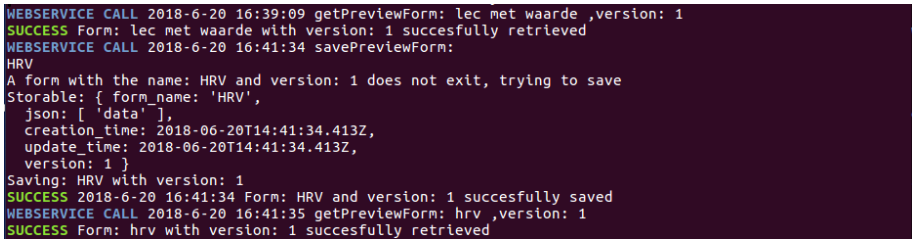

Figura 5.12: Imagen de los logs

CAPÍTULO 6
Integración con otras aplicaciones

Al inicio del proyecto y en el título del mismo se menciona que el sistema es multiplataforma, es decir, se puede integrar para su uso en otras aplicaciones. En este capítulo se explicará el proceso de integración de estas y los resultados obtenidos.

Webforms

Como se ha mencionado anteriormente, en versiones anteriores se usaba la carpeta *assets* del proyecto para guardar y cargar los formularios, pero este hacía que para desplegar y visualizar nuevos formularios, hiciera falta desplegar de nuevo el cliente. Una vez desarrollado el servidor de la aplicación mediante un parámetro y una llamada a un servicio web el cliente busca primero en la base de datos si existe un formulario con ese nombre y versión y en caso de no encontrarlo ya busca en la carpeta de *assets*. En la siguiente figura se muestra el proceso de guardado en las tablas de la base de datos dol *Formrunner* para las acciones de *Preview* y *Publish*.

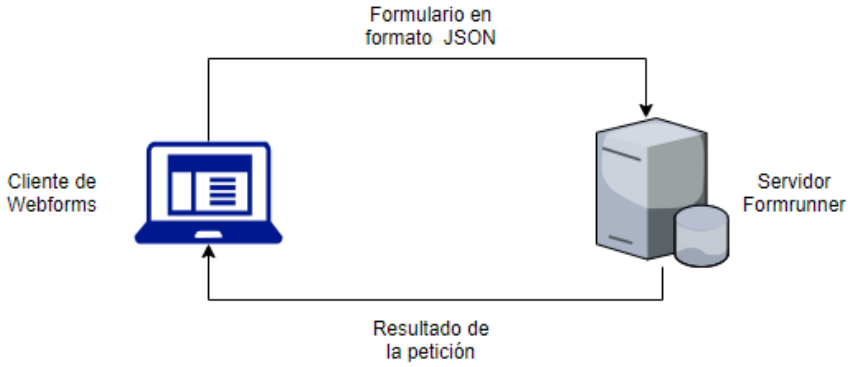

Figura 6.1: Diagrama de guardado para *Preview* y *Publish*

La aplicación *Webforms* possee una sección que permite mostrar un grafo [15] el cual representa el formulario y los distintos caminos que se pueden seguir. Se ha integrado con el cliente para que mediante una llamada a un servicio web del servidor, el formulario seleccionado en *Webforms* se cargue en la base del datos de previsualización del *Formrunner* y se abra una ventana que permita visualizarlo e interaccionar con él. Facilitando la comprobación y el correcto funcionamiento del mismo. Para ello mediante un servicio web se le envia al servidor del *Formrunner* el JSON con la especificación del formulario. A partir de esto el servidor guarda en la base de datos un objeto con el nombre del formulario, su versión y el JSON con la especificación. En caso de encontrarse un objeto con el mismo nombre y versión se actualiza en valor anterior. Para prevenir duplicados en la

base de datos en el caso que dos personas intentaran previsualizar el formulario a la vez se hace uso de transacciones *SQL* [32] que permiten ejecutar de forma atómica cada una de las funciones internas. Una vez guardado correctamente el objeto se llama al cliente con un parametro de preview para desactivar el envío de datos y el nombre y versión del formulario a mostrar como puede verse en la siguiente figura.

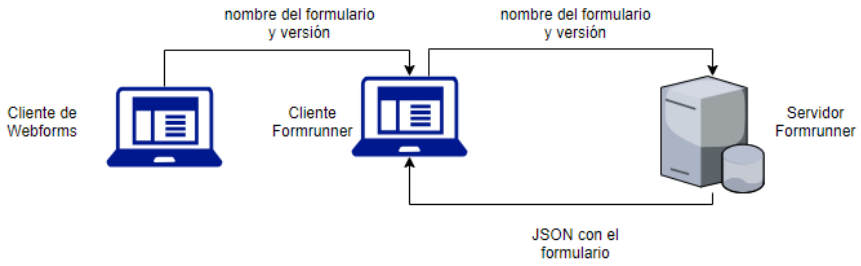

Figura 6.2: Diagrama de previsulización

En la siguiente figura se observa el en modo de presualización en *Webforms* en el que se muestra el formulario a previsualizar en una ventana del navegador.

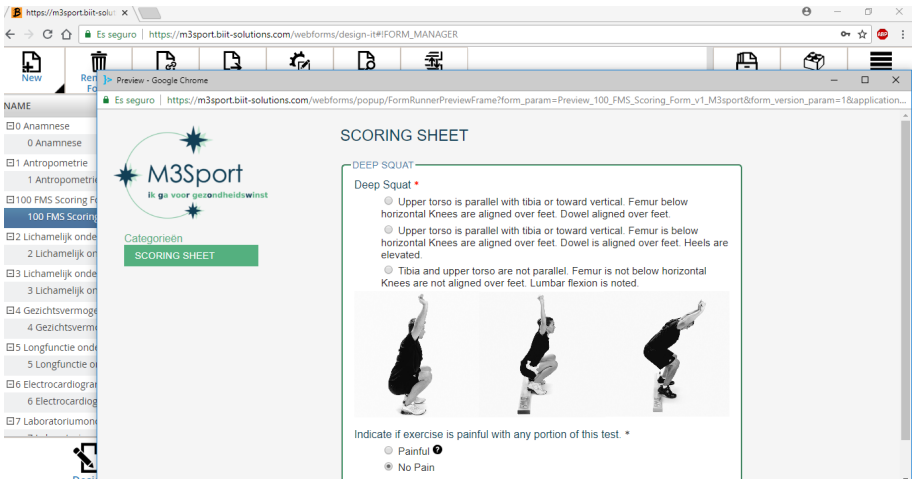

Figura 6.3: Imagen de previsualización en *Webforms*

Gracias a lo mencionado anteriormente de poder cargar de la base de datos los formularios, se añade otra funcionalidad a *Webforms* que permite el despliegue automático en la base de datos del *Formrunner*. De esta forma, con un simple botón y sin interacción media ningún especialista técnico se pueden desplegar los formularios y empezar a usarlos cambiando solamente el parámetro *form* de la *URL* al nombre del nuevo formulario. Al igual que se ha explicado anteriormente esto se consigue con una llamada a un servicio web. En este caso existen dos formas de visualizar el formulario. En el caso de especificar solamente el parámatro *form* con el nombre del formulario, se cargará la última versión de dicho formulariode la base de datos. Si se especifica la versión se cargará dicha versión del formulario. La figura 6.4 explica el funcionamiento del *publish*.

Sport Medi Score

En referencia a la parte de *Sport Medi Score* el *Formrunner*, como se ha mencionado en los primeros capítulos, el principal objectivo del *Formrunner* es iniciar el proceso de negocio permitiendo el envío de datos al sistema de *SMS*. Enviando los datos en el formato aceptado por *SMS*.

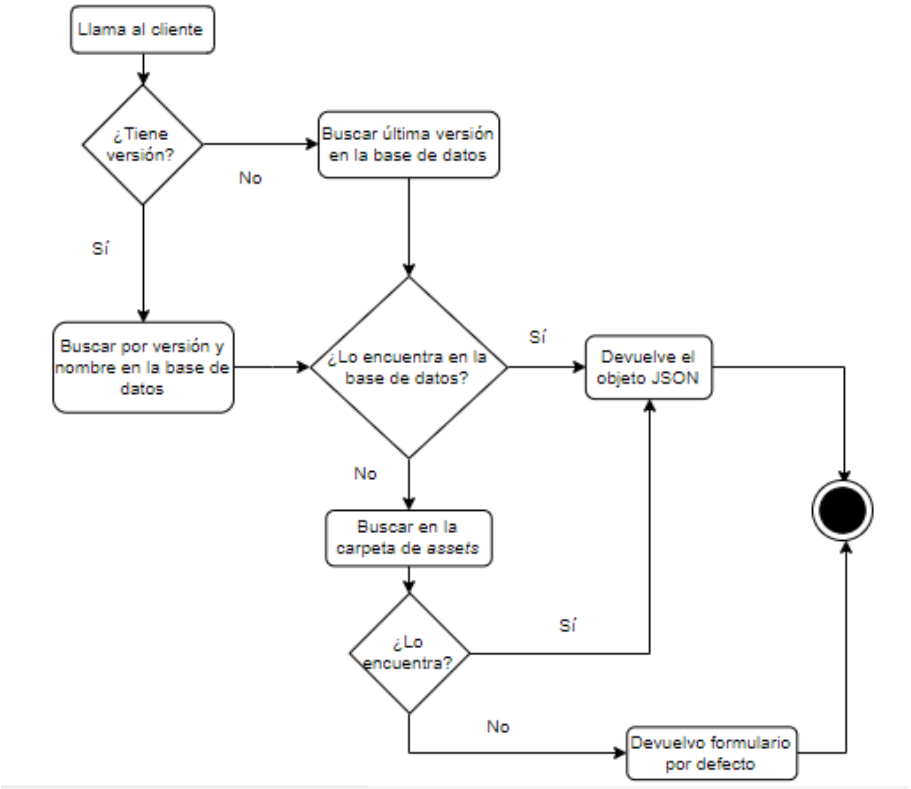

Figura 6.4: Diagrama de carga de formularios publicados

Otra funcionalidad integrada en *Sport Medi Score* (como se ha mencionado en la especificación de requisitos) es la posibilidad de recuperar las respuestas para visualizarlas. En este caso no se puede modificar el valor de éstas ya que se necesita la versión enviada por el cliente por temas de seguridad.

En el momento del envío, los resultados del mismo son guardados en la base de datos del *Formrunner* con la generación de una clave única para poder identificarlos. Esta clave es enviada a *Sport Medi Score* y en caso de querer recuperar los resultados, simplemente hay que enviar mediante un parámetro al cliente el identificador único y el nombre del formulario para que entre los dos se autorellenen las respuestas y se muestre el resultado del mismo. La figura 6.5 muestra el proceso seguido para visualizar las respuestas.

En la figura 6.6 se muestra como es la vista de visualización de las respuestas en *Sport Medi Score*.

IGROW

Como se ha mencionado en las aplicaciones de la empresa, *IGROW* es una aplicación móvil que permite agilizar la comunicación entre los centros médicos y sus clientes. En este caso se integra la librería desarrollada en la aplicación para poder enviar los formularios desde la misma. Como al configurar la aplicación móvil esta adquiere todos los datos personales del cliente una vez rellenado el formulario, se envían los datos del mismo junto con las respuestas evitando el uso de un navegador web en un navegador y agilizando y facilitando el procceso.

Esta funcionalidad todavía se encuentra en fase *beta* de desarrollo ya que la aplicación móvil no es competencia total de este trabajo. La figura 6.7 muestra el funcionamiento de la librería integrada con la aplicación móvil.

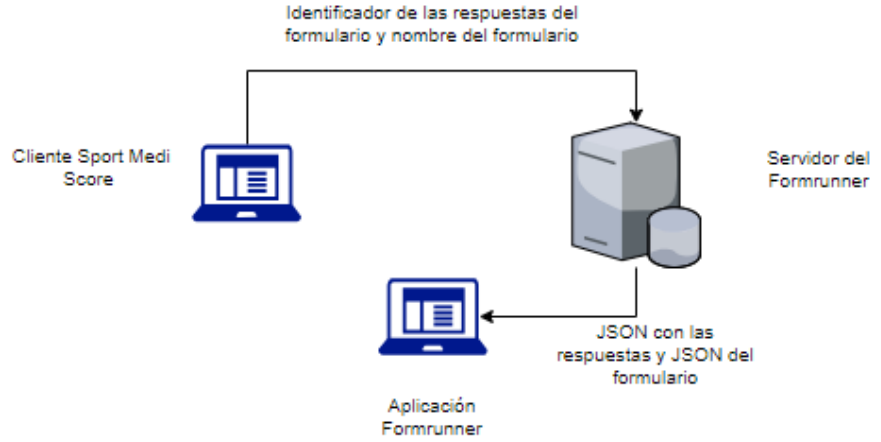

Figura 6.5: Diagrama de funcionamineto de la visualización de respuestas

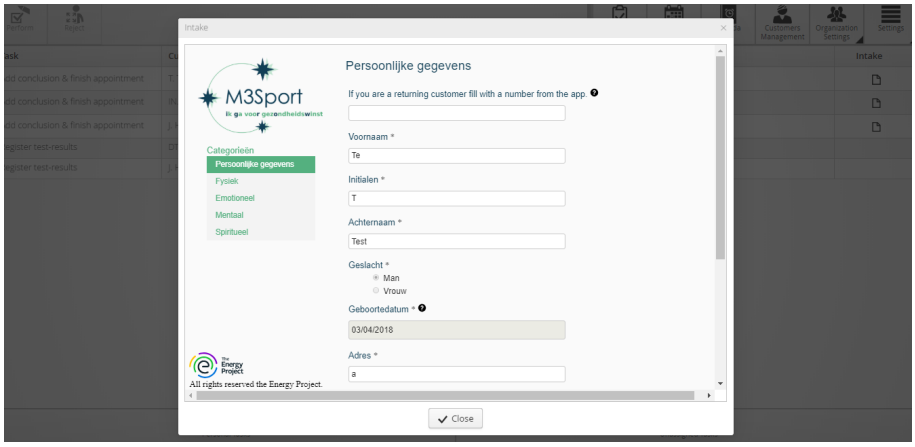

Figura 6.6: Previsualización de respuestas

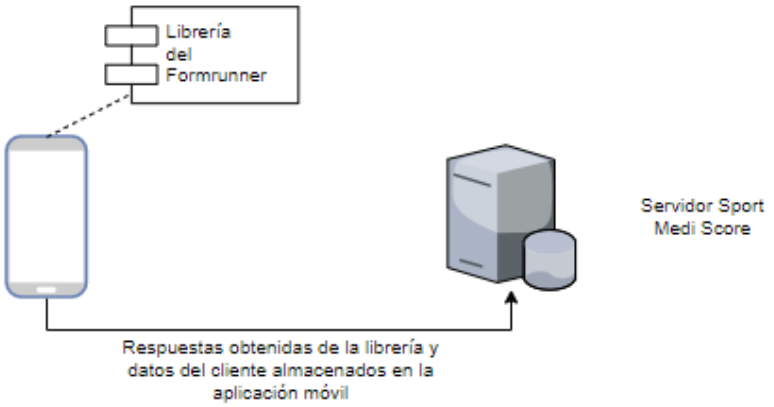

Figura 6.7: Diagrama de integración con IGROW

CAPÍTULO 7
Pruebas

Las pruebas es una de las etapas más importantes, si no la más, del ciclo de vida del software. Permiten asegurar que los requisitos se cumplen, que la aplicación se comporta como está especificada. Los test han de ser claros y sencillos para permitir una buena depuración del sistema. El libro *Clean Code* [2] cita textualmente que la importancia de unos buenos test es crucial para el desarrollo de un sistema tanto que su no uso o mal uso pueden abocar el resultado al fracaso.

Existen diferentes tipos de pruebas, en este proyecto se han usado dos de las más importantes, los test unitarios y los test de integración. Ambos permiten mediante combinaciones de entradas, por muy improbables que puedan ser probar la estabilidad del sistema ante las mismas teniendo como resultado y producto robusto.

Citando otra vez al libro, un conjunto de test no está completo hasta que por lo menos uno de ellos haga fallar el sistema y este pueda gestionar ese fallo. Así mismo también pueden comprobar el correcto funcionamiento de este.

Para la plataforma sobre la que se enjutarán los test automáticos se ha elegido el *Chrome Headless* [29] que permite ejecutar tanto los unitarios como los de integración es un entorno sin interfaz gráfica permitiendo así su automatización en sistemas como *Jenkins* que se explicarán posteriormente. En un principio se optó por el uso de *PhantomJS* [28] que ya había sido utilizado en el testeo de otros proyectos en la empresa, pero se cambió a *Chrome Headless* debido que el primero estaba obsoleto y sin soporte.

7.1 Test unitarios

Se encargan de comprobar cada componente de forma individual, asegurando que este se comporta como está especificado. La composición de un test unitario tiene tras partes: unos valores de entrada, unos valores de salida y unos valores esperados. Si los valores de salida no concuerdan con los valores esperados el test falla. Se puede comprobar la seguridad de un componente al cier por cien si se tienen en cuenta todas las posibles ejecuciones

Para la ejecución de los test unitarios Angular hace uso del *framework Jasmine* [12] y al ejecutar el comando ng test se lanza el *Karma test runner* [13]. En la figura de abajo se observa un test que comprueba si una pregunta es válida o no dependiendo de la entrada.

Actualmente hay definida una batería de 59 test unitarios y mediante el comando `ng test` se puede observar el resultado de estos.

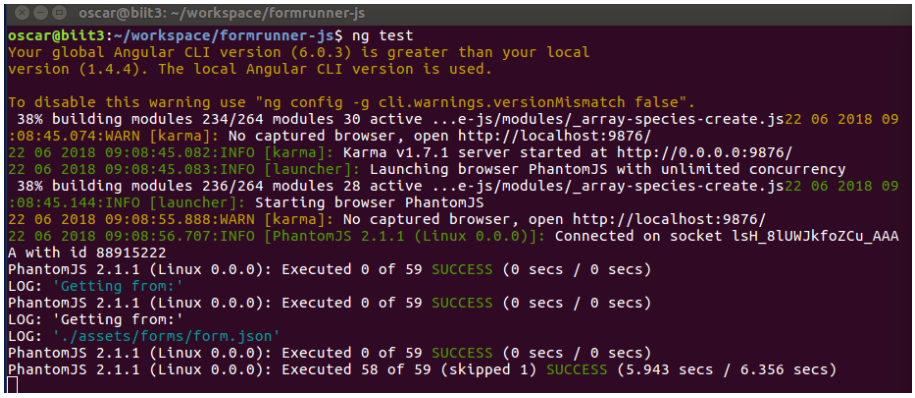

Figura 7.1: Ejecución test unitarios

7.2 Test de integración

Son los responsables de asegurar la integridad del sistema (todos sus componentes), es decir que cado uno de los elementos se comunica correctamente con los demás. Para ello se pone a prueba todo el sistema como un bloque.

Estas pruebas simulan la interacción real de un usuario con el sistema, mediante clics, cambios de ventana, rellenar datos etc … Permiten capturar comportamientos imprevistos por parte de los usuarios y gestionarlos de manera correcta.

Así mismo se pueden usar como pruebas de regresión ya que cualquier nueva funcionalidad añadida volverá a pasar los mismos test y se asegurará que la funcionalidad antigua sigue funcionando correctamente, permitiendo añadir nueva funcionalidad sin miedo a que la antigua deje de funcionar y de esta forma mejorando la integridad del sistema.

Su principal inconveniente respecto a los unitarios es el tiempo de ejecución, debido en parte a la necesitad de recrear una interfaz gráfica virtual.

De la misma forma que los unitarios en este caso una de las pruebas de integración se encarga de ir rellenando un formulario y cambiando valores para asegurar que el flujo de este se satisface correctamente en este caso comprobando la visibilidad de las preguntas, la validación de estas, si los botones han o no de estar activados etc.

Estos test se ejecutan con el comando `ng e2e` y el resultado de la ejecución es el siguiente.

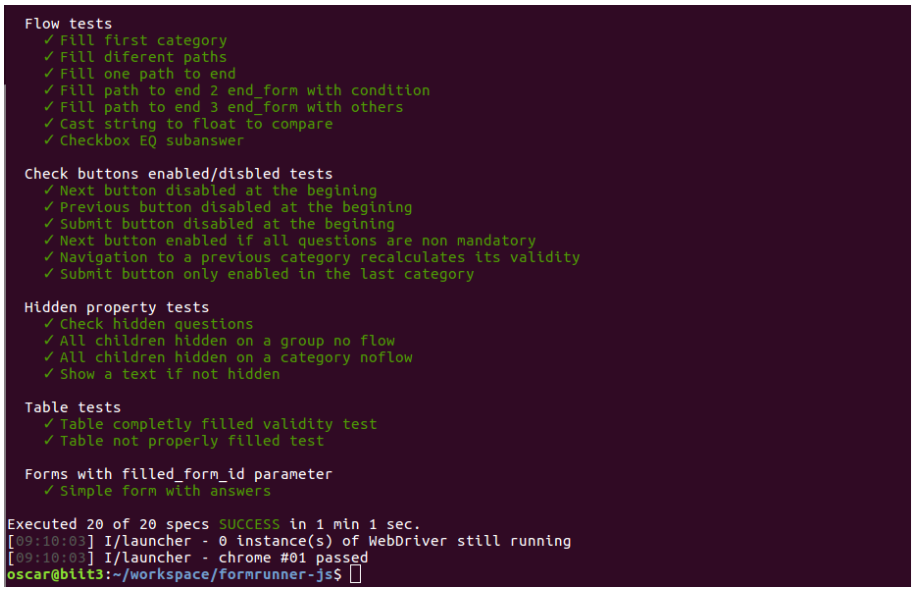

Figura 7.2: Ejecución test integración

CAPÍTULO 8
Ejemplo de uso

En este capítulo se va a mostrar un ejemplo de un caso de uso completo de la aplicación desde el inicio hasta crear una cita programada en Sport Medi Score.

Primero hay que seleccionar el tipo de examinación que se desearía tomar. Actualmente existen dos de tipo físico y una de tipo mental. Para ello en la página web de la organización a la que se quiere acudir existirían los distintos tipos de enlace para cada una de ellas que llevarían a la siguiente figura.

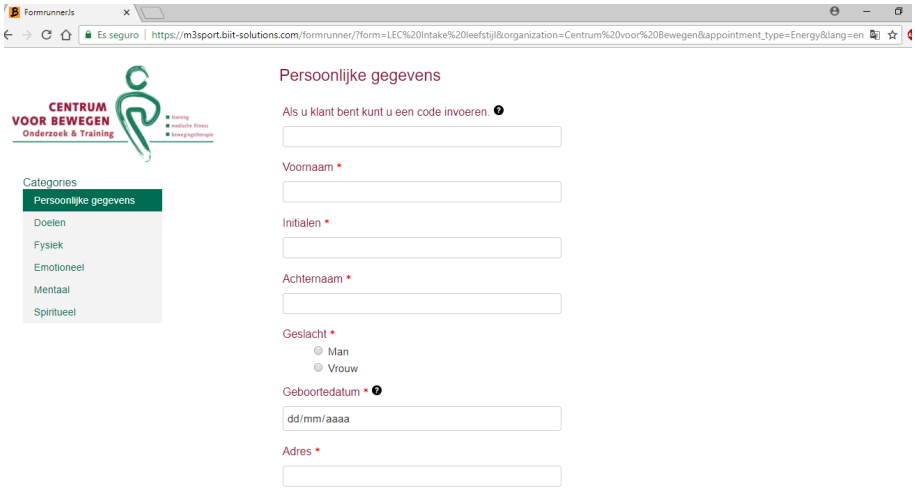

Figura 8.1: Figura del cliente vacío

Una vez aquí se podría empezar a rellenar el formulario con normalidad. Como se ha explicado en la Sección 4.3.1 si se es un cliente que vuelve al centro médico se puede usar la aplicación móvil para obtener el *user guard* y así rellenar automaticamente los datos, desbilitanto campos que no se pueden modificar, como es el caso de la fecha de nacimiento. En la figura 8.2 se muestra una categoría de la aplicación rellena. En este caso la última que es la única en la que se permite enviar el formulario.

Una vez se pulsa el botón enviar, se informa al cliente si el envio del formulario ha sido satisfactorio o ha habido algún error. Esto se muestra en la figura 8.3.

En el caso de ser satisfactorio se reenviará a una página donde se informará que el centro médico se pondrá en contacto con el cliente para la cita programada. Esto se muestra en la figura 8.4.

Una vez enviado el formulario, en la aplicación Sport Medi Score, en la sección de citas aparecerá una nueva cita por asignar con los datos del cliente que acaba de enviar el formulario como puede verse en la siguiente figura 8.5.

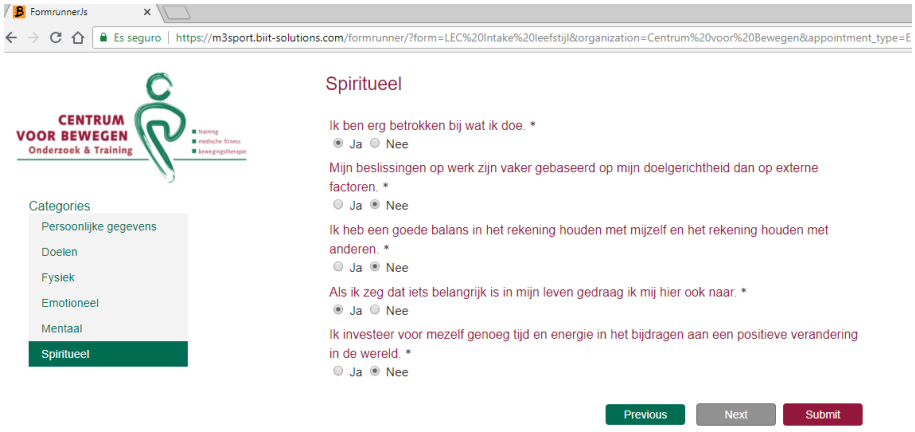

Figura 8.2: Figura formulario relleno

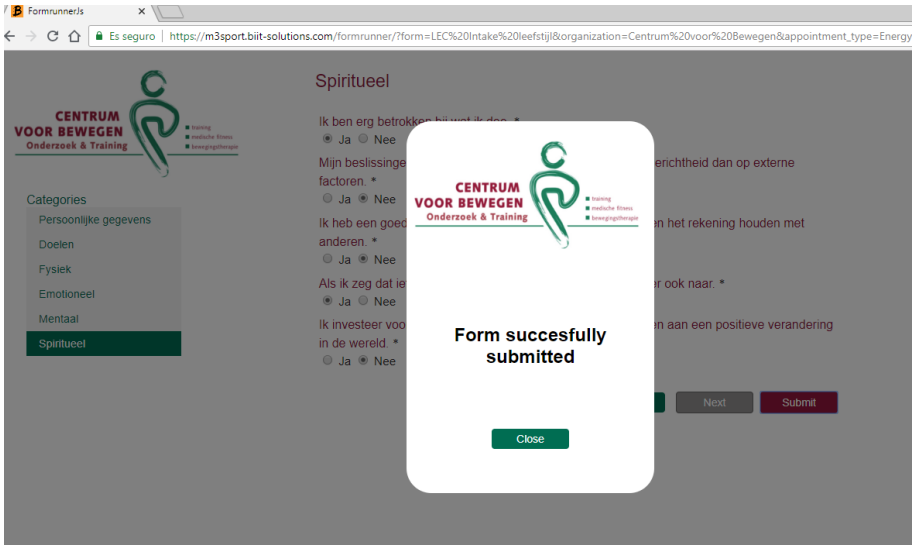

Figura 8.3: Figura de un envío correcto

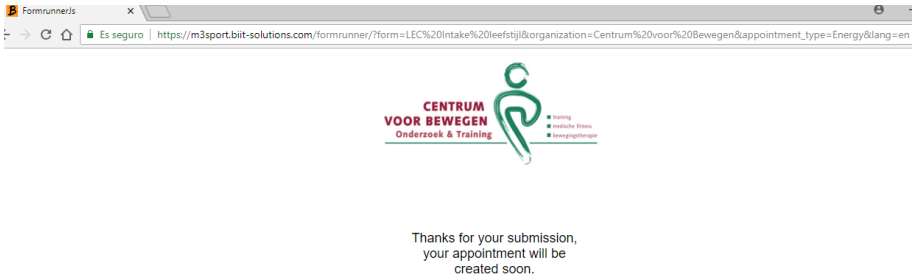

Figura 8.4: Figura de creación de aviso de que el centro médico se pondrá en contacto

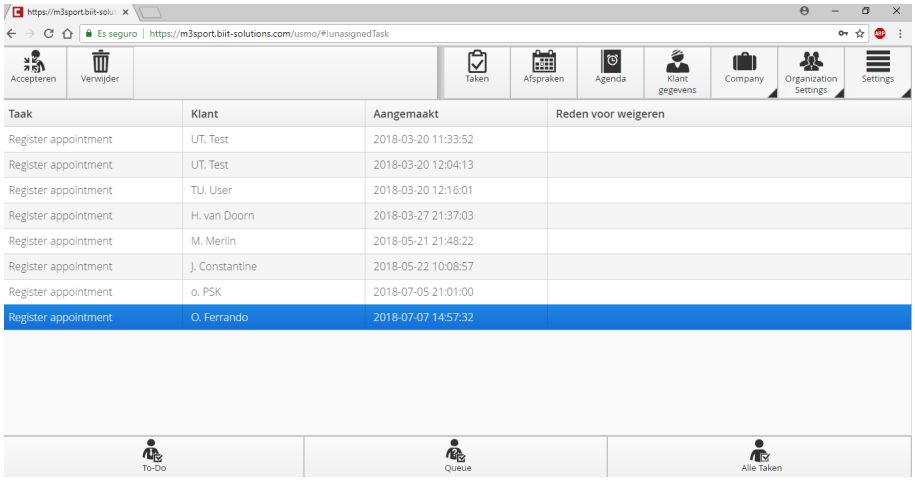

Figura 8.5: Figura de cita creada en *Sport Medi Score*

CAPÍTULO 9
Conclusiones

En este trabajo se ha visto el progreso de desarrollo del *Formrunner*, se ha demostrado que es una aplicación multiplataforma ya que se puede desplegar tanto para aplicaciones móviles como para aplicaciones web, además utiliza las últimas tecnologías del mercado como *Angular* y *Node*. En este trabajo se ha demostrado que la aplicación es estable, cosa que se comprueba como se ha descrito anteriormente con el uso de test unitarios y de integración.

Durante el desarrollo del proyecto han ido surgiendo problemas debido a que se trata de un proyecto complejo, pero se han podido solucionar todos. Se ha conseguido un sistema que se adapta perfectamente a las necesidades de la empresa y que cumple con los requisitos inicialmente propuestos. Mejorando en muchos casos los del sistema que se usaba inicialmente para la gestión de los formularios.

Debido a que el sistema ya se encuentra en pruebas en un entorno real como se explicará en el siguiente apartado, se han podido ir realizando pequeñas mejoras, así como encontrando errores no detectados durante el desarrollo gracias a la retroalimentación o *feedback* que proveían los clientes que lo usaban.

9.1 Casos de éxito

Esta aplicación está desplegada en un escenario real *Centrum Voor Bewegen*, un centro médico situado en la localidad de *Veenendaal* en los Países Bajos, en el que en este momento existen 20 usuarios en pruebas y se espera que a finales de verano pase a producción. También se espera que una vez terminada la migración a la nueva versión de la aplicación de gestión de centros (*Sport Medi Score*) en *Orbis Sport*, otro centro médico situado en *Sittard-Geleen*, Países Bajos.

Como se ha mencionado al inicio de esta memoria este trabajo ha sido realizado en el contexto de unas prácticas de empresa y han desembocado en un contrato laboral en la misma, lo que me permitirá seguir trabajando en esta aplicación.

9.2 Trabajo futuro

La aplicación todavía puede mejorarse añadiendo nuevas funcionalidades. Durante el desarrollo del proyecto han ido surgiendo nuevas ideas:

- Nuevos tipos de representación del formulario (aparte del normal y tabla).
- Facilitar la creación de distintos estilos para las organizaciones o centros.

- Mejorar el despliegue para que lo formularios también se desplieguen en *Sport Medi Score* ya que hasta ahora se necesitan conocimientos técnicos para su despliegue.

Bibliografía

[1] Melcón Álvarez, Alejandro. Diseño e implementación de una aplicación multiplataforma para la presentación y seguimiento de programas médicos. Septiembre 2017.

[2] Robert C. Martin. *Clean Code*. Prentice Hall.

[3] BiiT Sourcing Solutions, S.L. Consultado el 15 de mayo de 2018 http://biit-solutions.com/.

[4] Bussiness inteligence. Consultado el 16 de mayo de 2018 https://www.sinnexus.com/business_intelligence/.

[5] Information Technology. Consultado el 16 de mayo de 2018 https://www.lifewire.com/introduction-information-technology-817815.

[6] Orbeon. Consultado el 20 de abril de 2018 https://www.orbeon.com/.

[7] Drools. Consultado el 25 de abril de 2018 https://www.drools.org/.

[8] Xforms. Consultado el 29 de mayo de 2018 https://es.wikipedia.org/wiki/XForms.

[9] Angular-CLI. Consultado el 10 de octubre de 2017 https://cli.angular.io/.

[10] Observable. Consultado el 25 de octubre de 2017 https://angular.io/guide/observables.

[11] Expresión regular. Consultado el 15 de noviembre de 2017 https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions.

[12] Jasmine. Consultado el 20 de octubre de 2017 https://docs.angularjs.org/guide/unit-testing#jasmine.

[13] Karma Test Runner. Consultado el 20 de octubre de 2017 https://karma-runner.github.io/2.0/index.html.

[14] SQL. Consultado el 15 de enero de 2018 https://www.w3schools.com/sql/.

[15] Grafo. Consultado el 10 de junio de 2018 https://es.wikipedia.org/wiki/Grafo.

[16] IEEE. Consultado el 20 de junio de 2018 http://dis.unal.edu.co/~icasta/GGP/xDBD/2013_02_18_ieee830/GGP_IEEE_830_vL4.pdf.

[17] Component Based Development. Consultado el 16 de junio de 2018 https://www.techopedia.com/definition/31002/component-based-development-cbd.

[18] HTTP. Consultado el 19 de enero de 2018 https://developer.mozilla.org/en-US/docs/Web/HTTP.

[19] HTTPS. Consultado el 19 de enero de 2018 https://www.instantssl.com/ssl-certificate-products/https.html.

[20] Métodos ágiles. Consultado el 5 de junio de 2018 https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/.

[21] Javascript. Consultado el 10 de enero de 2018 https://www.javascript.com/.

[22] Typescript. Consultado el 10 de octubre de 2017 https://www.typescriptlang.org/.

[23] Node. Consultado el 10 de enero de 2018 https://nodejs.org/es/.

[24] Angular. Consultado el 10 de octubre de 2017 https://angular.io/.

[25] NPM. Consultado el 12 de octubre de 2017 https://www.npmjs.com/.

[26] Git. Consultado el 10 de octubre de 2017 https://git-scm.com/.

[27] Jenkins. Consultado el 11 de octubre de 2017 https://jenkins.io/.

[28] PhantomJs. Consultado el 20 de octubre de 2017 http://phantomjs.org/.

[29] Chrome Headless. Consultado el 20 de mayo de 2017 https://cvuorinen.net/2017/05/running-angular-tests-in-headless-chrome/.

[30] Integración continua. Consultado el 10 de junio de 2018 https://www.thoughtworks.com/continuous-integration.

[31] Pipe. Consultado el 20 de diciembre de 2017 https://angular.io/guide/pipes.

[32] Transacción SQL. Consultado el 20 de enero de 2018 https://angular.io/guide/pipes.

[33] Vaadin. Consultado el 6 de octubre de 2018 https://vaadin.com/.

[34] Verdaccio. Consultado el 20 de abril de 2018 https://www.verdaccio.org/.

[35] Scrum. Consultado el 4 de octubre de 2017 https://proyectosagiles.org/que-es-scrum/.

APÉNDICE A
IDE's de desarrollo y pruebas

Durante el desarrollo del proyecto se han usado distintas interfaces de desarrollo y plugins para facilitar la realización de este.

A.1 Visual Studio Code

Como se ha desarrollado la mayor parte del proyecto en *Typescript* y *Javascript*, se ha usado *Visual Studio Code* como editor de código. Es un editor ligero y rápido pero potente que permite trabajar con distintos lenguajes como los mencinados anteriormente y tiene el *git* integrado.

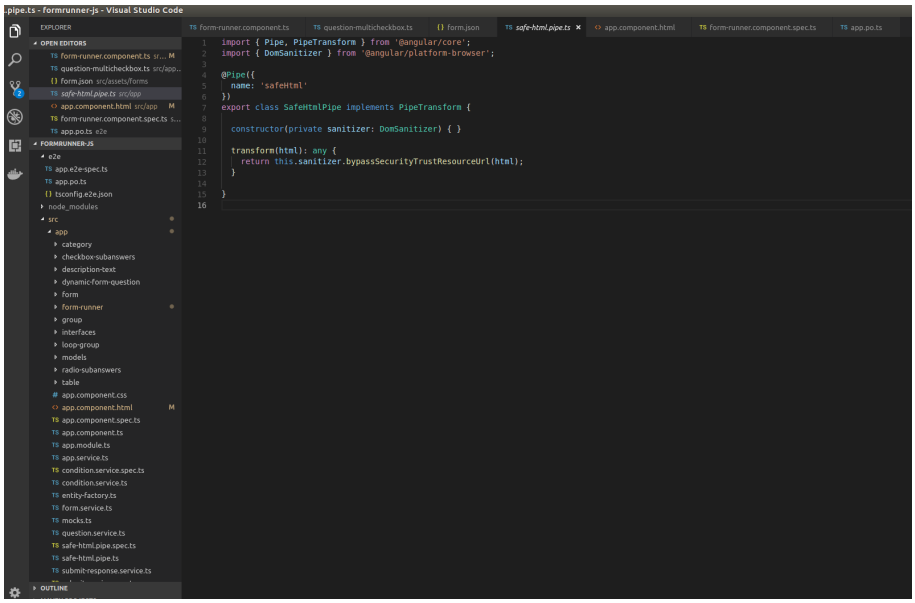

Figura A.1: Figura del VS Code

A.2 Eclipse

La aplicaciónes web de la empresa con las que se ha integrado la aplicación (ver Capítulo 6) están desarrolladas en Java se ha optado por el uso de Eclipse Oxygen, una herramienta *open source* y la más utilizada para el desarrollo en Java.

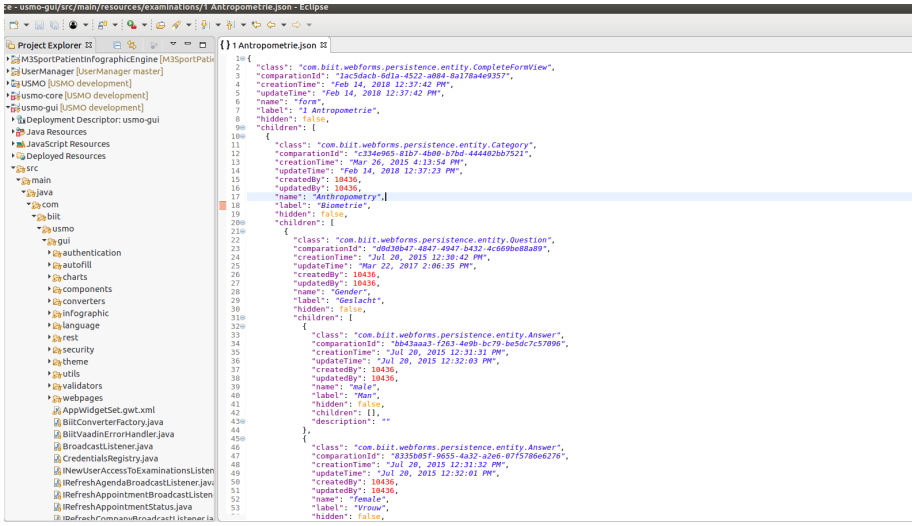

Figura A.2: Figura del Eclipse Oxygen

A.3 Chrome Reslet

Dado que se ha implementado un servidor que gestiona llamadas a servicios web, se ha hecho uso del plugin de *Chrome*, *Reslet*, el cual permite realizar peticiones y observar el resultado de la respuesta de las mismas permitiendo comprobar que se ejecutan de manera correcta.

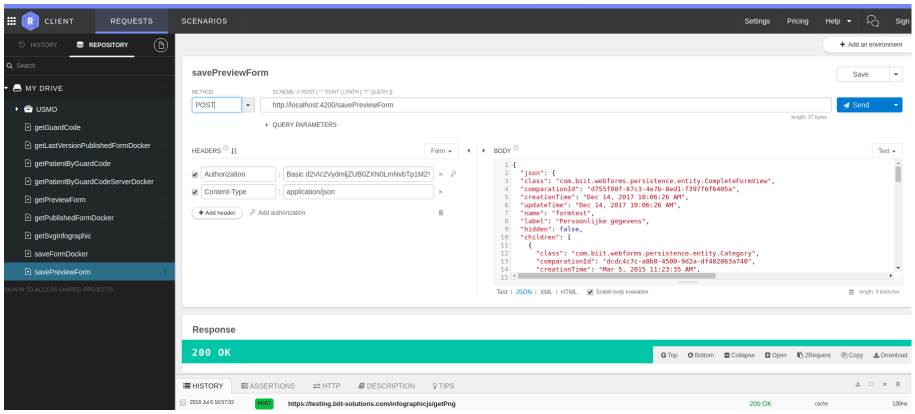

Figura A.3: Figura del Reslet de Chrome