Universitat de les Illes Balears

Escola Politècnica Superior

Memòria del Treball de Fi de Grau

Simulación de *APIs* en pro de la eficiencia en el desarrollo y mantenimiento de integraciones

Estudio y mejora de un producto de integración turística

Francisco Bernardo Bisquerra Castell

Grau de Enginyeria Informàtica

Any acadèmic 2018-19

DNI de l'alumne: 43226835Z

Treball tutelat per Beatriz Gómez Suárez
Departament de Matemàtiques i Informàtica

S'autoritza la Universitat a incloure aquest treball en el Repositori Institucional per a la seva consulta en accés obert i difusió en línia, amb finalitats exclusivament acadèmiques i d'investigació	Autor		Tutor	
	Sí	No	Sí	No
	X		X	

Paraules clau del treball:
Mock, API, integración

GRAU D'ENGINYERIA INFORMÀTICA

Simulación de *API*s en pro de la eficiencia en el desarrollo y mantenimiento de integraciones

Estudio y mejora de un producto de integración turística

FRANCISCO BERNARDO BISQUERRA CASTELL

Tutor

Beatriz Gómez Suárez

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 10 de septiembre de 2019

Quisiera agradecerle a Beatriz el esfuerzo y dedicación realizados en el desarrollo de este proyecto.

Índice general

Índice general **v**

Índice de figuras **vii**

Índice de cuadros **ix**

Glosario **xi**

Resumen **xiii**

1 Introducción **1**
1.1 Contexto . . . 2
1.2 Objetivos . . . 3
1.2.1 La Independencia del proveedor en el testeo del aplicativo . . . 3
1.2.2 El desarrollo en base a la definición del modelo y sin entorno . 4
1.3 Soluciones . . . 5
1.4 Planificación . . . 7

2 Tecnologías **9**
2.1 *Back-end* . . . 9
2.2 *Front-end* . . . 10
2.3 Diagrama tecnológico . . . 12

3 Análisis **15**
3.1 Integración turística . . . 15
3.2 Proveedor - cliente . . . 15
3.3 Proveedor - concentrador - cliente . . . 16
3.4 Integración a más bajo nivel . . . 17
3.4.1 Destinos . . . 17
3.4.2 Disponibilidad . . . 18
3.4.3 Reserva . . . 19
3.4.4 Consulta . . . 20
3.4.5 Cancelación . . . 21
3.5 *Wiremock* . . . 21
3.6 Peticiones de los proveedores . . . 24
3.6.1 *URI* . . . 25
3.6.2 Método *http* . . . 26
3.6.3 Cabeceras *http* . . . 26

3.6.4 Parámetros de consulta . . . 27
3.6.5 Cuerpo de la solicitud . . . 27
3.6.6 Escenarios . . . 29
3.6.7 Respuestas de los proveedores . . . 29
3.6.8 Estudio de acoplamiento a *Wiremock* . . . 30
3.7 Almacenamiento de los *stubs* . . . 32
3.8 *cgd-wiremock-frontal* . . . 33

4 Diseño e implementación 35
4.1 Diagramas de flujo y uso . . . 35
4.1.1 *cgd-wiremock* . . . 35
4.1.2 Wiremock . . . 36
4.1.3 *cgd-wiremock-frontal* creación, edición y borrado de *stubs* . . . 36
4.1.4 *cgd-wiremock-frontal* detalle y listado de *stubs* . . . 37
4.2 *StubFiles* . . . 38
4.3 *API cgd-wiremock* . . . 41
4.4 Aplicación *Java cgd-wiremock* . . . 41
4.5 Aplicación *JS cgd-wiremock-frontal* . . . 43
4.5.1 Listado de stubs . . . 43
4.5.2 Formulario de creación y edición de *stubs* . . . 44
4.5.3 Filtro de cuerpo de la solicitud . . . 45
4.5.4 Extructura de carpetas . . . 47

5 Resultados 49
5.1 *Mockeando* una petición . . . 49

6 Conclusiones 57
6.1 Mejoras a futuro . . . 58

Bibliografía 59

ÍNDICE DE FIGURAS

1.1 Esquema de interacción en el mundo turístico 1
1.2 Esquema de flujo entre cliente y proveedor 6
1.3 Esquema de flujo entre cliente y *cgd-wiremock* 6
1.4 Diarama de gantt del proyecto 7

2.1 Diagrama de la aplicación que muestra donde se aplica cada tecnología . . 13

3.1 Integración directa entre proveedor y cliente 16
3.2 Integración directa entre proveedor y cliente 16

4.1 Creación de un *mock* en *cgd-wiremock* 35
4.2 Interacción entre el usuario y *Wiremock* 36
4.3 Creación, edición y borrado de *stubs* en *cgd-wiremock* 37
4.4 Listado y detalle de *stubs* en *cgd-wiremock* 38
4.5 Estructura de carpetas en *Java cgd-wiremock* 41
4.6 Creación, edición y borrado de *stubs* en *cgd-wiremock* 42
4.7 Listado de *stubs* en *cgd-wiremock-frontal* 43
4.8 Formulario de creación y edición de *stubs* en *cgd-wiremock-frontal* 45
4.9 Filtros de cuerpo de solicitud en *cgd-wiremock-frontal* 46
4.10 Vista de texto plano en *cgd-wiremock-frontal* 47
4.11 Estructura de carpetas para los componentes en *cgd-wiremock-frontal* . . 48

5.1 Primera parte del formulario de creación de *stubs* en *cgd-wiremock-frontal* 51
5.2 Segunda parte del formulario de creación de *stubs* en *cgd-wiremock-frontal* 52
5.3 Registro de *stubs* en *cgd-wiremock-frontal* 53
5.4 Segunda parte del detalle de un *stubs* en *cgd-wiremock-frontal* 53
5.5 Segunda parte del detalle de un *stubs* en *cgd-wiremock-frontal* 54
5.6 Petición de reserva de hotel en *Postman* 55
5.7 Respuesta de reserva de hotel en *Postman* 56

Índice de cuadros

4.1 Acciones disponibles en la *API*. 41

Glosario

API Application Programming Interface

CGD Content Global Distributor

CRUD Create, Read, Update, Delete

CSS Cascading Style Sheets

DOM Document Object Model

ECMAScript Es una especificación de lenguaje de programación publicada por European Computer Manufacturers Association (ECMA) International

ECMA European Computer Manufacturers Association

HTML HyperText Markup Language

HTTP Portable Document Format

integración Concepto de establecer una comunicación directa entre dos de los participantes de la cadena de interacciones

JSF JavaServer Faces

JSON JavaScript Object Notation

mockear Acción de crear un mock, siendo mock la simulación de la respuesta de una API

mock Simulación

MVC Model View Controller

PDF Portable Document Format

programáticamente (Programar) Utilizando código fuente, en lugar de una interfaz de usuario

RQ Petición

RS Respuesta

SSJS Server-Side JavaScript

stub Conjunto de datos que define que dada una petición RQ debe devolverse una respuesta RS

W3C World Wide Web Consortium

WORA Write Once Run Anywhere

Resumen

Gracias a las tecnologías de la información se ha podido elaborar un complejo entramado de comunicaciones entre cada una de las partes de la cadena de compra-venta de un producto turístico: proveedores, concentradores y clientes. Dichas comunicaciones se llevan a cabo mediante programas informáticos llamados integraciones. Las mismas se deben desarrollar, mantener y mejorar para poder ofrecer el mejor servicio. En el desarrollo, mantenimiento y mejora de dichas integraciones, es importante la interacción entre las partes del entramado, y lamentablemente no siempre es posible mantener una comunicación constante. Esto afecta al desarrollo de tareas como: ejecución de tests de integración, ejecución de tests de rendimiento, el mismo desarrollo, mantenimiento y correctivo de estas plataformas.

Para dar solución a dicha problemática se plantean dos herramientas que independizan a los proveedores, concentradores y clientes de las otras partes. Hecho que se consigue mediante la simulación de las otras partes que intervienen en la comunicación. Se ha desarrollado una herramienta que permite definir qué respuesta debe devolver cuando se le haga una petición *X*, *cgd-wiremock*, de manera que se pueden crear simulaciones de los otros participantes, por ejemplo, el cliente puede simular al proveedor y así no requiere interactuar con él durante el desarrollo. También se ha implementado una interfaz de usuario para gestionar estas simulaciones de forma sencilla, *cgd-wiremock-frontal*.

De esta forma se solucionan los problemas en los test de integración y rendimiento ya que si un proveedor no dispone de entorno de test, porque esta caído, se define una simulación y se utiliza hasta que el entorno de test esté disponible de nuevo. También se consigue la paralelización de desarrollos entre las partes. Los clientes podrán simular los desarrollos de los proveedores, que están definidos, pero no implementados y disponibles.

CAPÍTULO 1

INTRODUCCIÓN

Desde que se implantaron las tecnologías de la información en el mundo turístico, la interacción entre los participantes del mismo se ha llevado a término mediante *Application Programming Interface (API)*s y servicios web. De este modo, se forma una cadena de comunicación entre el vendedor original del producto, el dueño del hotel y el comprador final del mismo (la familia que se va de vacaciones) a través de los intermediarios.

Todos los intermediarios asumen al menos uno de los tres roles descritos a continuación:

- *proveedor*: recolectar y agrupar el producto para venderlo.
- *cliente*: comprar el producto al proveedor para venderlo.
- *concentrador*: establece una comunicación entre al menos dos de los anteriores. Habitualmente agrega varios proveedores para facilitar la integración a los clientes.

En la figura 1.1 se observa la comunicación para la compraventa de un producto turístico entre el vendedor inicial, Hotelero, y el comprador final, Familia.

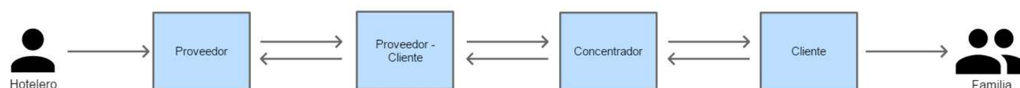

Figura 1.1: Esquema de interacción en el mundo turístico

La interacción entre las partes de la cadena es crucial en el desarrollo, mejora y

mantenimiento de las plataformas informáticas que dan soporte a la misma, donde es habitual que alguna de las partes no esté disponible por un tiempo. Este hecho impide la correcta ejecución de las siguientes tareas:

- Ejecución de tests de integración.
- Ejecución de tests de rendimiento.
- Desarrollo, mantenimiento y correctivo de nuevas integraciones.

Para dar solución a dicha problemática se plantea una herramienta que permita a los equipos de desarrollo de cualquiera de los intermediarios llevar a cabo las tareas mencionadas sin necesidad de la plataforma del siguiente eslabón de la cadena.

Esta herramienta permite *Acción de crear un mock, siendo mock la simulación de la respuesta de una API (mockear)*, (acción de crear un *Simulación (mock)*, siendo *mock* la simulación de la respuesta de una *API*) los servicios del intermediario. El objetivo es que dada una plataforma informática de un intermediario, una *API*, se pueda simular el comportamiento de dicha plataforma para que en caso de ser necesario se pueda prescindir de ella. Para ello se podrán *mockear* todas las respuestas de dicha *API* a las interacciones con la misma. Por lo tanto:

- Se solucionan los problemas de los tests de integración y rendimiento *mockeando* los entornos de tests de los intermediarios, entornos contra los cuales se lanzan dichos tests.
- Se da pie al desarrollo, mantenimiento y correctivo de las aplicaciones *mockeando* las *API*s actuales o las que están por desarrollar en base a sus documentos de definición. Pudiendo incluso paralelizar el desarrollo de la *API* de un intermediario que actúa como proveedor y la integración con la misma por parte de un intermediario que actúa como cliente.

1.1 Contexto

La herramienta se ha desarrollado a petición de la emphresa *Travel IMS* filial de *Amadeus IT Group.* [1]

Amadeus es una empresa líder en el sector turístico en la compraventa de productos y oferta turística ya sea vuelos, hoteles, paquetes vacacionales, etc. El último año adquirió la compañía *Hiberus Travel* y pasó a formar parte del conglomerado de *Amadeus* bajo el nombre *Travel IMS*. *Travel IMS* ofrece una aplicación *B2B* llamada *TravelIO* para el sector turístico que permite la gestión y venta de productos turístico de todos los tipos mencionados anteriormente. *TravelIO* es una aplicación personalizable y adaptable a las necesidades de su comprador. Cada uno de los clientes de la aplicación la personaliza a su gusto y añade sus propias reglas de negocio, además les posibilita la integración de su *TravelIO* con cualquier nuevo proveedor. La solución por la que se ha optado en *Travel IMS* para facilitar todas estas integraciones es desarrollar un concentrador de integraciones llamado *Content Global Distributor (CGD) (Content*

Global Distributor), es decir, una capa intermedia entre el proveedor del producto y el cliente. Dicho cliente en el caso de *Travel IMS* es *TravelIO*. De esta manera se consigue agilizar el trabajo y potenciar los beneficios ya que a partir de ahora *TravelIO* se integra con *CGD* y *CGD* con cada uno de los proveedores.

La operativa es la siguiente: *TravelIO* quiere vender hoteles y por lo tanto se integra con la parte de hoteles de *CGD*. Ahora bien, *El Corte Inglés*, cliente crucial de *TravelIO* quiere comprar hoteles a *HotelBeds* y se lo comunica a *Travel IMS*. Entonces se inicia el desarrollo de la integración sobre *CGD* y, como *TravelIO* ya está integrado con *CGD*, se consigue la venta de hoteles de *HotelBeds* a través del *TravelIO* de *El Corte Inglés*. Además si ahora otro cliente como *TUI* quisiera también producto, de *HotelBeds* sería suficiente con facilitarles una versión de *CGD* que incluya la integración con *TUI* ya que ya ha sido desarrollada anteriormente para *El Corte Inglés*.

Durante el desarrollo del nuevo concentrador de integraciones, *CGD*, se requería garantizar dos características cruciales en los proyectos de tecnologías de la información. La primera, que el cliente pudiese consumir el producto del que debía servir el nuevo concentrador pese a que la integración que debía servirlo estuviera aún en desarrollo. La segunda, testear todos los desarrollos realizados sobre el concentrador de forma independiente a los proveedores, evitando así añadir a los tiempos de procesado de datos la latencia de comunicación con el proveedor y además poder garantizar entornos de test estables y resilientes.

1.2 Objetivos

A continuación se desarrollan los cuatro objetivos a los que se le debe poner solución.

1.2.1 La Independencia del proveedor en el testeo del aplicativo

Existen dos tipos de tests cruciales dentro de una aplicación como *CGD*. Los tests de integración y los tests de rendimiento.

Test de integración

Se utilizan para garantizar que todo el flujo de integración sea correcto, desde el cliente que consume el contenido de *CGD* hasta la conexión con el proveedor. Para realizar estos tests se precisa de comunicación con los proveedores, quienes facilitan entornos de test. Habitualmente dichos entornos sufren de ciertas limitaciones como: no son fiables ni estables, tienen el número máximo de peticiones o sesiones muy acotado, el mismo proveedor los utiliza para realizar pruebas o cambios que afectan a *CGD* sin avisar y demás problemas que impiden usar estos entornos todo el tiempo y de forma segura.

Por lo tanto sería óptimo disponer de un entorno que simule la respuesta del proveedor y que a su vez siga permitiendo testear la comunicación con el mismo. Anteriormente se obviaba la comunicación con el proveedor en caso de que este fallase, sustituyéndola por una respuesta ya configurada. Esta forma de atacar el fallo del

proveedor supone un problema y es que se deja de testear la parte de código que realiza la comunicación con el proveedor.

Para seguir testeando todo el flujo incluyendo la comunicación del proveedor se precisa de una nueva forma de simulación del proveedor.

Test de rendimiento

CGD es un aplicativo de alta disponibilidad, es decir, recibe miles de peticiones por minuto y tiene que poder garantizar que estará listo para responder a todas ellas. Una de las formas de testear la disponibilidad de un aplicativo son los tests de rendimiento. Estos intentarán hacer que el aplicativo deje de funcionar, en el caso de *CGD* que no haya respuesta o se caiga el servidor mediante la realización de peticiones concurrentes al mismo. Para mostrar la importancia de los tests de rendimiento, es necesario explicar un poco más el funcionamiento interno de *CGD*.

El cliente envía una petición de disponibilidad de hoteles, es decir, cuántas habitaciones hay disponibles para un hotel en ciertas fechas y para 2 adultos. En *CGD* se efectúan los siguiente pasos:

1. Recibe la petición cliente.
2. Transforma esa petición a otra u otras peticiones que consultan lo mismo pero para el modelo de datos del proveedor.
3. Realiza la petición o peticiones al proveedor.
4. Transforma los datos de respuesta del proveedor al modelo de datos de *CGD*.
5. Devuelve la respuesta al cliente.

De entre los pasos que realiza *CGD*, el tercero implica al proveedor, quienes internamente también procesan datos y por lo tanto añaden tiempo y sobrecarga al sistema. Si se pudiese evitar la interacción con el proveedor y cambiarla por un sistema que devuelva la respuesta deseada pero de forma automática, se conseguiría eliminar de las pruebas de rendimiento el retardo añadido por la comunicación y procesamiento de datos por parte del proveedor.

1.2.2 El desarrollo en base a la definición del modelo y sin entorno

Durante el desarrollo existe otro caso en el que disponer de una herramienta que permita la simulación de las respuestas de un proveedor turístico es idílico. Este es el caso en el que la información o producto a solicitar o consumir no esté disponible, ya sea porque se está trabajando en el desarrollo de la plataforma o porque se está trabajando en el desarrollo de la integración y ese producto.

Desarrollo de la plataforma

Se plantea el caso en el que un cliente quiere consumir a través de *CGD* un producto que no está contemplado, se pone como ejemplo las entradas de cine. Desde *CGD* se

debe definir un modelo para vender el producto de entradas de cine, pero además, después de la definición del modelo se deben desarrollar las integraciones con los distintos proveedores de entradas de cine. Cabe recordar que se trata del esquema Proveedor - Concentrador - Cliente.

Para nuestro cliente, *TravelIO*, el proveedor es *CGD*, entonces, por qué no simular *CGD* de la misma manera que se han simulado los proveedores en los casos de testeo del aplicativo. De esta manera, una vez definido el modelo de datos, el cliente puede comenzar a desarrollar sobre la simulación de *CGD* mientras en *CGD* se desarrollan las integraciones.

Baja disponibilidad de los entornos de test de los proveedores

Como se ha comentado anteriormente, los proveedores tienen en muchas ocasiones entornos de test que no son fiables, no funcionan o son intermitentes. Para el desarrollo de integraciones es vital poder consumir de un servicio o *API* ya sea para desarrollar la parte que se encarga de realizar las peticiones como para facilitar el desarrollo de las partes que se encarga del procesado de datos.

En el caso de disponer de definiciones claras de la información que proporciona el proveedor de producto, al simularlo se consigue independencia de los entornos que no funcionan y por lo tanto ralentizan el desarrollo.

Para dar solución a los problemas citados anteriormente se decide implementar un conjunto de aplicaciones que permitan configurar la simulación de los proveedores y de *CGD*. Dichas aplicaciones se describen a continuación.

1.3 Soluciones

Se plantea el desarrollo de dos aplicaciones complementarias entre sí. Por un lado se implementará un simulador de *APIs*, que se focalizará en permitir la configuración de *stubs* (conjunto de datos que define que dada una petición *Petición (RQ)* debe devolverse una respuesta *Respuesta (RS)*). Estos *stubs* permiten a la aplicación:

1. Procesar una petición.

2. Compararla con las *RQ*s guardadas en los *stubs*.

3. Devolver la respuesta *RS* asignada a la *RQ* mas parecida a la recibida.

La herramienta se llamará *cgd-wiremock* y para facilitar la interacción con la misma y proporcionar una plataforma más usable para la gestión de estos *stubs* se desarrollará una aplicación llamada *cgd-wiremock-frontal*.Por lo tanto, utilizando *cdg-wiremock-frontal* se puede enviar una petición a *cdg-wiremock* definiendo qué respuesta debe devolverse a una petición determinada, definiendo así un *Conjunto de datos que define que dada una petición RQ debe devolverse una respuesta RS (stub)* y creando un *mock* de la respuesta de una *API*.

Por ejemplo, se puede definir que a una petición de tipo *requestA* se devuelva una respuesta de tipo *responseA* y que a una petición de tipo *requestB* se devuelva una respuesta de tipo *responseB*. El objetivo de la definición de estos *stubs* es la simulación de una secuencia de acciones a realizar en comunicación con una *API* para así poder desarrollar y testear de forma independiente a dicha *API*. A continuación se muestra en unos diagramas en qué punto se sitúa la herramienta desarrollada, *cgd-wiremock*. La figura 1.2 muestra la situación anterior al desarrollo de *cgd-wiremock*.

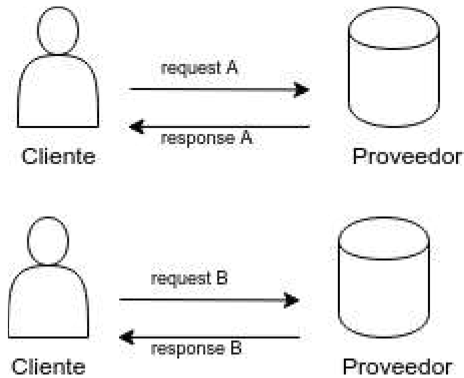

Figura 1.2: Esquema de flujo entre cliente y proveedor

Por otra parte la figura 1.3 muestra la situación posterior al desarrollo de *cgd-wiremock*, que permite el mismo flujo prescindiendo del proveedor.

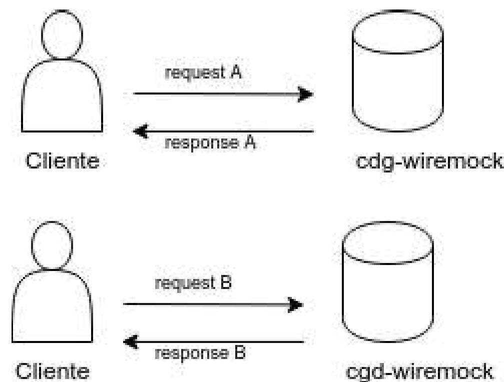

Figura 1.3: Esquema de flujo entre cliente y *cgd-wiremock*

De la siguiente forma se consigue independencia de la *API* para el desarrollo y testeo de aplicaciones que deban consumir de los mismos. Esta independencia es el objetivo principal de esta herramienta razón de dos motivos. El primer motivo es la inestabilidad de los entornos de test y desarrollo de estas *API*. Dichos entornos pese a ser inestables normalmente están documentados y ejemplificados. Con suficientes ejemplos y documentación se puede crear la suite de *stubs* que simula la *API* e independizar el desarrollo de nuestra Concepto de establecer una comunicación directa entre dos de los participantes de la cadena de interacciones (integración) de la misma. El segundo motivo es la paralelización de desarrollos. Previo desarrollo de cualquier proyecto in-

formático se documenta y define el mismo. En el caso de desarrollar una *API* por un lado y por otro un frontal que consuma de la misma, si se ha definido y ejemplificado la *API* correctamente, se podría crear la suite de *stubs* que simula el comportamiento de la misma y por tanto el frontal podría empezar a desarrollarse consumiendo los datos de *cgd-wiremock* mientras la *API* se está desarrollando paralelamente.

1.4 Planificación

En esta sección se explica la planificación del proyecto y las fases que contiene

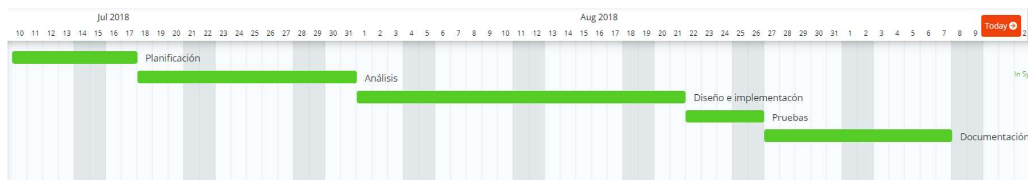

Figura 1.4: Diarama de gantt del proyecto

- **Planificación**: En esta fase se estudian los distintos *frameworks* y herramientas para *mockear* que hay disponibles, se divide el proyecto en fases de determinada duración con la finalidad de estructurar el desarrollo.

- **Análisis**: Se estudian las necesidades del proyecto *CDG* y el entorno del mismo para entender cuáles son los requisitos que deben cumplir las aplicaciones que se desarrollarán. Se analizará qué es una integración y cuáles son sus partes. También cómo son las peticiones y respuestas de los participantes en la cadena de compra-venta de productos turísticos, así como el almacenamiento de los datos que se generen, y las necesidades del frontal.

- **Diseño e implementación**: Se diseñarán los flujos de uso de las aplicaciones por parte del usuario y los flujos de interacción entre aplicaciones. Se definirán tanto la *API* como la petición de creación de *stubs*. Se implementarán las aplicaciones *Java* y *JavaScript* y se diseñará el frontal.

- **Pruebas**: Se probarán los flujos definidos mediante la creación de suites de *mocks* de las integraciones ya desarrolladas.

- **Documentación**: Se documentará todo lo realizado durante el desarrollo del proyecto.

CAPÍTULO 2

TECNOLOGÍAS

Se dividirán los desarrollos en dos partes. La primera parte denominada *back-end*, se refiere a toda la parte no visible por el usuario final. La segunda denominada *front-end* hace referencia a la interfaz gráfica de usuario.

2.1 *Back-end*

Durante la elección de las herramientas a utilizar en el desarrollo del *back-end* se barajaron varios lenguajes y *frameworks*. *Ruby on rails* [2], *framęwork* de *Ruby* muy conocido en Estados Unidos aunque menos utilizado en Europa, destaca por tener muy buenos estándares y rapidez durante el desarrollo, aunque a su vez es poco flexible. *Django* [3], un *framework* muy potente para *Python* que es otro lenguaje muy utilizado hoy en día en el desarrollo web, incluye por defecto un gran conjunto de herramientas para la gestión de base de datos, enrutamiento, etc. Muy útil para definir *Create, Read, Update, Delete (CRUD)*s (*Create, Read, Update, Delete*) ágilmente pero poco práctico para lo que requería el *back-end* de este proyecto. Finalmente se optó por *Java* sobre *Spring Boot 2* [4], por ser la herramienta más conocida en el equipo y por mantener coherencia con el resto de desarrollos del proyecto *CGD*.

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como *Write Once Run Anywhere (WORA)*, o *"write once, run anywhere"*), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para ejecutarse en otra. [5]

Spring Boot 2 es una herramienta que trabaja sobre *Spring* y añade un conjunto de configuraciones y dependencias de terceros, que los desarrolladores del mismo han

considerado oportunas para facilitar el desarrollo de aplicaciones complejas de forma sencilla. [6] *Spring* es el *framework* más utilizado por la comunidad de *Java*. La idea de *framework* surge de querer facilitar y estandarizar la forma de desarrollar, dando soluciones rápidas y eficaces a las situación y problemas más recurrentes, así como facilitando las tareas habituales y repetitivas. Gran parte del éxito de este *framework* surge de la funcionalidad más importante que aporta, la inversión de control, ya que descarga al programador de ciertas tareas y responsabilidades a la hora de codificar y diseñar las aplicaciones. *Spring* utiliza el patrón de diseño de Inyección de Dependencias para llevar a cabo esta inversión de control y tiene como finalidad conseguir un código más desacoplado, que facilita las cosas a la hora de hacer Tests y además permite cambiar partes del sistema más fácilmente en caso de que fuese necesario.

Spring Boot 2 en el caso de ambas aplicaciones se ha utilizado por la rapidez con la que se puede implementar y desplegar una aplicación web, ya que el *framework* gestiona todo el despliegue de servidores, configuraciones de puertos, etc.

Además se ha utilizado la herramienta *Wiremock* también desarrollada en *Java*. Dicha herramienta proporciona, en su versión embebida, una aplicación que permite la configuración de *stubs* de forma programática, así como un servidor al que se le pueden realizar peticiones y, si estas coinciden con alguno de los *stubs* definidos, devuelve la respuesta correspondiente. [7] Al añadir *Wiremock* a este proyecto se puede instanciar al arrancar la aplicación y utilizar en los métodos que sean necesarios de la *API* de la librería para definir los *stubs*.

Un ejemplo simple sería el siguiente, para la definición de una petición *GET* al *endpoint "some/thing"* se devuelva una respuesta con estado 200 que significa *OK*:

```
stubFor(get("/some/thing").willReturn(aResponse().withStatus(200)));
```

La aplicación *cgd-wiremock* se ha desarrollado como una capa que actúa sobre *Wiremock* acotando el gran número de posibilidades que ofrece ya que no todas son interesantes en la situación en la que se encuentra este proyecto y complican la tarea de definir *stubs*. En la próxima sección se expone cómo se ha implementado dicha capa y porqué.

Se valoraron alternativas a *Wiremock* y finalmente se descartaron por no ofrecer una versión libre del software que puede embeberse en esta aplicación. Las otras aplicaciones o tecnologías se ofrecían como un servicio en la nube en el que se podían definir las *APIs* que se querían simular. Por lo tanto desde nuestra empresa no se podía diseñar el software a medida para el proyecto.

2.2 *Front-end*

Para el desarrollo de la parte del cliente web en *cgd-wiremock-frontal* se han utilizado el lenguaje *JavaScript* en su versión *ES5*. *JavaScript* (abreviado comúnmente *JS*) es un lenguaje de programación interpretado, dialecto del estándar *Es una especificación de*

lenguaje de programación publicada por ECMA International (ECMAScript). Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. [8]

Se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Aunque existe una forma de *JavaScript* del lado del servidor (*Server-side JavaScript* o *Server-Side JavaScript (SSJS)*), finalmente no se ha utilizado porque en el diseño de la aplicación el servidor se ha implementado en *Java*. Su uso en aplicaciones externas a la web, por ejemplo en documentos *Portable Document Format (PDF)*, aplicaciones de escritorio (mayoritariamente *widgets*) es también significativo.

También se ha usado *HTML*, siglas en inglés de *HyperText Markup Language* (lenguaje de marcas de hipertexto), que hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código *HTML*) para la definición de contenido de una página web, como texto, imágenes, vídeos, juegos, entre otros. Es un estándar a cargo del *World Wide Web Consortium (World Wide Web Consortium (W3C))* o Consorcio WWW, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la *World Wide Web (WWW)*. Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.

Otra herramienta utilizada en el proyecto ha sido *Bootstrap*, es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en *HTML* y *Cascading Style Sheets (CSS)*, así como extensiones de *JavaScript* adicionales. A diferencia de muchos *frameworks* web, solo se ocupa del desarrollo *front-end*.

Para el desarrollo de la lógica necesaria en el frontal e interacción web se ha utilizado el *framework AngularJS*. Es una librería para el desarrollo con *Javascript* que permite el manejo del *Document Object Model (DOM)* (Modelo de Objetos del Documento) de forma sencilla, es decir, entender los inputs del usuario y actuar en consecuencia en base a lo que desee el diseñador de la web [9]. *AngularJS* (comúnmente llamado *Angular.js* o *AngularJS 1*), es un *framework* de *Javascript* de código abierto, mantenido por *Google*, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (*Model View Controller (MVC)*), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles. La librería lee el *HyperText Markup Language (HTML)* que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de *JavaScript*. Los valores

de las variables de *JavaScript* se pueden configurar manualmente, o recuperalos de los recursos *JavaScript Object Notation (JSON)* estáticos o dinámicos.

Se ha escogido *AngularJS* porque permite el desarrollo de interfaces web basadas en componentes, tendencia actual, ya que independiza cada una de las partes de la web y facilita el testeo al tener cada parte funcionalidades específicas y más sencillas. Además permite reutilizar estos componentes a lo largo de toda la página facilitando el mantenimiento y mejora de los mismos.

Se podría haber escogido un diseño monolítico de la aplicación desarrollando el frontal con algún *framework* para java como *primefaces* o *JavaServer Faces (JSF)*. La diferencia principal es que estos *frameworks* compilan la página en el servidor bajo demanda del cliente y la devuelven para que el navegador del cliente la muestre en pantalla. *AngularJS* por otra parte se descarga en el ordenador cliente y únicamente recupera datos del servidor procesando dicha información en cliente. De esta forma se libera al servidor de carga de trabajo, muy conveniente en el caso de webs ligeras.

Además, para la gestión de las tablas de datos en el frontal se ha utilizado la librería *DataTables*, un complemento para la biblioteca *jQuery*. Es una herramienta altamente flexible, construida sobre los cimientos de la mejora progresiva, que agrega todas estas características avanzadas a cualquier tabla *HTML*. Se encuentra un gran número de librerías para el manejo de tablas *HTML* como *AngularJS UI Grid* o *Smart Table* pero no existe actualmente una librería para manejo de datos en forma de tablas tan potente como *Datatables*. Además, esta proporciona integración con la librería de estilos escogida *Bootstrap3*, facilitando a su vez la paginación en servidor y la customización de celdas entre otras muchísimas posibilidades.

2.3 Diagrama tecnológico

A continuación, en la figura 2.1 se muestra un diagrama simple que ilustra en qué parte de la aplicación se utilizan las distintas tecnologías así como la comunicación entre las mismas.

Como se puede observar, las aplicaciones se dividen en dos servidores y el cliente web. El servidor *back-end* tiene a su vez embebido un servidor de *Wiremock* con el que es posible comunicarse (Programar) Utilizando código fuente, en lugar de una interfaz de usuario (programáticamente) desde el servidor *back-end* o via *Portable Document Format (HTTP)* desde fuera del mismo. El servidor *front-end* simplemente se encarga de enviar al Cliente *front-end*, por ejemplo un ordenador de sobremesa, la aplicación *AngularJS* que se ejecutará en el mismo. El Cliente *front-end* por su parte se comunicará con el servidor *back-end* para la definición y mantenimiento de las *API*s *mockeadas*. Finalmente cualquier usuario que quiera consumir de las *API*s simuladas, podrá hacerlo a través del Servidor *Wiremock*.

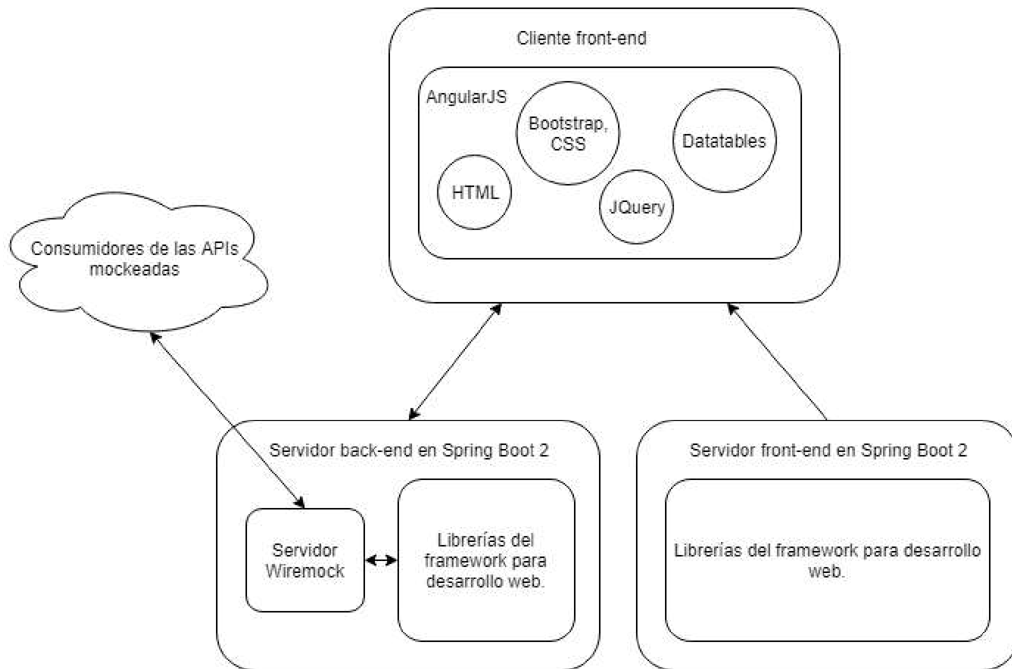

Figura 2.1: Diagrama de la aplicación que muestra donde se aplica cada tecnología

CAPÍTULO 3

ANÁLISIS

En esta sección se describa el análisis llevado a cabo para el desarrollo de las dos aplicaciones *cgd-wiremock* y *cg-wiremock-frontal* con el objetivo de implantar las soluciones propuestas a los problemas planteados. En primera instancia se estudiará el contexto, se entenderá mejor que es una integración y en qué punto de la cadena se encuentra *CGD*.

3.1 Integración turística

Primero se introducirá el concepto de integración turística, para después poder indagar en los problemas que esta presenta de cara al testeo y estudio de rendimiento del aplicativo. Integración es la palabra que se utiliza para referirse al concepto de establecer una comunicación directa entre dos de los participantes de la cadena de interacciones.

3.2 Proveedor - cliente

En la figura 3.1 se muestra el caso más sencillo. El proveedor ofrece un producto turístico y se lo vende al cliente mediante un canal informático directo, generalmente conocido como *API*. Puede haber otras tecnologías y protocolos similares. Para realizar la compra-venta el cliente se conecta al servicio de venta del proveedor y realiza el flujo preestablecido por el mismo para finalmente adquirir el producto deseado. Un posible flujo sería: disponibilidad ->confirmación ->consulta/cupón/cancelación.

Se puede observar que en esta operación de compra sólo se involucran el cliente que compra el producto y el proveedor que vende el mismo.

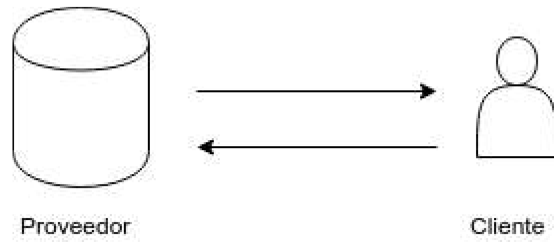

Figura 3.1: Integración directa entre proveedor y cliente

3.3 Proveedor - concentrador - cliente

Como se ve en la figura 3.2 se mantiene el flujo anterior, pero la principal diferencia es que en este caso se añade un agente intermedio. La función de este agente es integrarse con varios proveedores, que utilizan definiciones, flujos y reglas de integración diferentes. El agente intermedio aglomera todos estos distintos proveedores de producto bajo un único flujo, una única definición de modelo de datos y unas únicas reglas. De la siguiente manera la aplicación cliente debe integrarse con un único servicio para acceder a múltiples proveedores de producto.

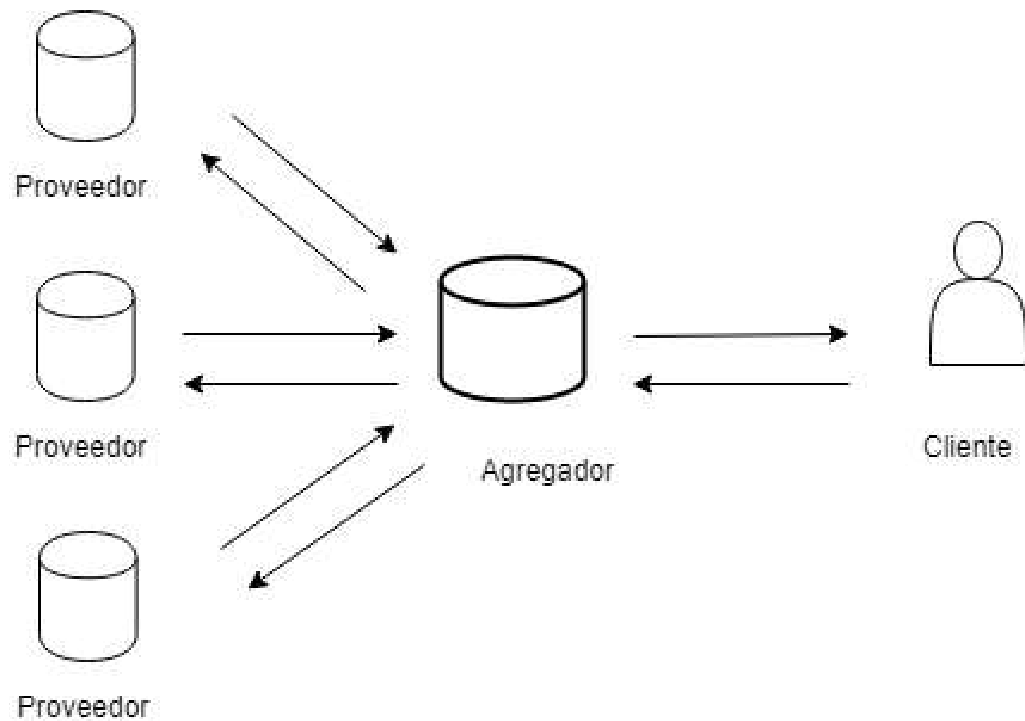

Figura 3.2: Integración directa entre proveedor y cliente

3.4 Integración a más bajo nivel

Hasta ahora se ha hablado del concepto de integración turística a alto nivel pero para comprender los beneficios que aporta *CGD* y por lo tanto entender mejor la necesidad de *cgd-wiremock* es necesario indagar ligeramente más y desarrollar en qué consiste una integración, cuál es el punto de partida de estas, qué papel desarrolla *CGD* y cómo lo lleva a cabo.

Los proveedores turísticos proporcionan plataformas informáticas, generalmente una *API*. Dichas plataformas constan de operaciones que permiten consultar, comprar, cancelar, etc. un producto turístico, por ejemplo, un hotel, y de un modelo de datos para cada operación, es decir, la definición de la petición a realizar y de la respuesta que se devolverá en caso de realizarla.

Para que resulte más sencillo de ahora en adelante se utilizará el ejemplo de la reserva de una habitación de hotel para dos personas durante una semana en el mes de septiembre. Y nuestro proveedor dispondrá una *API Rest* de cinco operaciones llamadas:

1. **Destinos**: Devuelve una lista de los hoteles disponibles.
2. **Disponibilidad**: Devuelve las habitaciones disponibles en base a ciertos filtros.
3. **Reserva**: Confirma la reserva de una de las opciones disponibles.
4. **Consulta**: Recupera información sobre una reserva.
5. **Cancelación**: Cancela una reserva.

Por lo tanto para realizar una reserva de un hotel con este proveedor se debe seguir el siguiente flujo. Hacer una petición de destinos, mediante la que se escoge el destino y se realiza la petición de disponibilidad para saber si hay habitaciones para los dos adultos en las fechas deseadas. Una vez se obtiene la respuesta de disponibilidad, se realiza la reserva con los datos necesarios, por ejemplo quién será el pasajero principal y por lo tanto el que se hará cargo del coste. Para consultar posteriormente la información de la reserva se realizará una llamada a **Consulta** con el localizador que se ha recuperado en reserva. Por último si se desea cancelar la reserva, se realizara la petición de cancelación.

Integrarse con este proveedor consiste en realizar una aplicación que de una forma u otra permita comprar un producto a través de la plataforma descrita que proporciona el proveedor. A continuación se describen las operaciones disponibles del proveedor que deberían seguirse en caso de querer integrarse con el mismo.

3.4.1 Destinos

Petición:

```
GET http://proveedor.com/rest/hoteles
```

Respuesta: **200 OK**

```
{
    "hoteles": [
        {
            "nombre": "Hotel tres playas",
            "codigo": "HTP",
            "descripcion": "Hotel frente al mar"
        },
        {
            "nombre": "Hotel Mont Blanc",
            "codigo": "HMB",
            "descripcion": "Hotel de montaña"
        }
    ]
}
```

3.4.2 Disponibilidad

Petición:

```
POST http://proveedor.com/rest/hoteles/HTP/disponibilidad
```

```
{
    "fechas": {
            "entrada": "09/09/2019",
            "salida": "12/09/2019"
        },
    "edades": ["09/07/1983", "31/02/1982"]
}
```

Respuesta: **200 OK**

```
{
    "habitaciones": [
        {
            "tokenDeReserva": "mcajklshffjkgkgufgasfnaw==",
            "nombre": "Habitación doble",
            "descripcion": "Habitación doble con ventana",
            "tipo": "HDv",
            "planes": ["soloHabitacion", "mediaPension", "pensionCompleta"],
            "precio": 200,
            "moneda": "EUR"
        },
        {
            "tokenDeReserva": "dfasdgdfsghthdthehaethht==",
            "nombre": "Habitación doble",
            "descripcion": "Habitación doble interior",
            "tipo": "HDi",
            "planes": ["soloHabitacion", "mediaPension", "pensionCompleta"],
            "precio": 150,
            "moneda": "EUR"
```

```
        },
        {
            "tokenDeReserva": "mcajklshflasdlñfgasfnaw==",
            "nombre": "Suite",
            "descripcion": "Habitación doble con vistas a la piscina y
                            salón particular",
            "tipo": "S",
            "planes": ["mediaPension", "pensionCompleta"],
            "precio": 400,
            "moneda": "EUR"
        }
    ]
}
```

3.4.3 Reserva

Petición:

```
POST http://proveedor.com/rest/hoteles/HTP
```

```
{
    "tokenDeReserva": "mcajklshflasdlñfgasfnaw==",
    "pasajeros": [
        {
            "fechaNacimiento": "09/07/1983",
            "nombre": "Pepe Viyuela",
            "email": "pepe.viyuela@email.com",
            "direccion": "C/ marques de caceres 18 2a",
            "principal": true
        },
        {
            "fechaNacimiento": "31/02/1982",
            "nombre": "Maria Viyuela",
            "principal": false
        }
    ]
}
```

Respuesta: **200 OK**

```
{
    "estado": "reservado",
    "localizador": "ABD2XYZ",
    "hotel": {
            "nombre": "Hotel tres playas",
            "codigo": "HTP",
            "descripcion": "Hotel frente al mar"
        },
    "habitacion": {
            "nombre": "Suite",
            "descripcion": "Habitación doble con vistas a la piscina y
                            salón particular",
            "tipo": "S",
```

```
            "planes": ["mediaPension", "pensionCompleta"],
            "precio": 400,
            "moneda": "EUR"
        },
    "pasajeros": [
        {
            "fechaNacimiento": "09/07/1983",
            "nombre": "Pepe Viyuela",
            "email": "pepe.viyuela@email.com",
            "direccion": "C/ marques de caceres 18 2a",
            "principal": true
        },
        {
            "fechaNacimiento": ""31/02/1982"",
            "nombre": "Maria Viyuela",
            "principal": false
        }
    ]
}
```

3.4.4 Consulta

Petición:

```
GET http://proveedor.com/rest/hoteles/reservas/ABD2XYZ
```

Respuesta: **200 OK**

```
{
    "estado": "reservado",
    "localizador": "ABD2XYZ",
    "hotel": {
            "nombre": "Hotel tres playas",
            "codigo": "HTP",
            "descripcion": "Hotel frente al mar"
        },
    "habitacion": {
            "nombre": "Suite",
            "descripcion": "Habitación doble con vistas a la piscina y
                 salón particular",
            "tipo": "S",
            "planes": ["mediaPension", "pensionCompleta"],
            "precio": 400,
            "moneda": "EUR"
        },
    "pasajeros": [
        {
            "fechaNacimiento": "09/07/1983",
            "nombre": "Pepe Viyuela",
            "email": "pepe.viyuela@email.com",
            "direccion": "C/ marques de caceres 18 2a",
            "principal": true
        },
        {
```

```
            "fechaNacimiento": "31/02/1982",
            "nombre": "Maria Viyuela",
            "principal": false
        }
    ]
}
```

3.4.5 Cancelación

Petición:

```
DELETE http://proveedor.com/rest/hoteles/reservas/ABD2XYZ
```

Respuesta: **200 OK**

Existen cientos de proveedores de producto turístico y la mayoría de ellos han definido su propia plataforma, es decir, comprar un producto implica integrarse con ellos y de una forma distinta a la que ya se había integrado con el anterior. A raíz de esto surge la idea de crear un concentrador de integraciones, una plataforma que defina un flujo, operaciones y modelo único y cuyo propósito sea integrarse con el mayor número de proveedores diferentes para poder vender su producto a través de la misma.

Una vez se ha comprendido el tipo de herramienta a la que dan soporte *cgd-wiremock* y *cgd-wiremock-frontal* se exponen los problemas que solucionan dichas aplicaciones y porqué facilitan el desarrollo tanto en *CGD* como a los clientes de *CGD*.

3.5 *Wiremock*

En primera instancia se estudia el funcionamiento de *Wiremock*. Dicha herramienta consiste en un servidor en el que programáticamente (versión embebida) o mediante una *API* de administrador se pueden crear *stubs*. Permite la configuración de un puerto de escucha en el despliegue de la aplicación,en cuyo caso tomará el 8089. Al realizar una petición contra ese puerto, *Wiremock* compara la petición con todos los *stubs* que se han cargado anteriormente y si encuentra uno que cumple los requisitos para considerarse igual, devuelve la respuesta definida para el mismo.

A continuación se ejemplifica el objetivo que se quiere conseguir con *cgd-wiremock*: *mockear* la respuesta a la siguiente petición. La petición consiste en solicitar los 5 primeros hoteles a un proveedor con el filtro por ciudad, Madrid.

```
GET http://proveedor.com/rest/hoteles?limit=5&offset=0
```

la respuesta a la misma es

```
{
    "hoteles": ['H1', 'H2', 'H3', 'H4', 'H5']
}
```

El primer paso es estudiar la petición puesto que el usuario de *cgd-wiremock* deberá poder decidir qué características de la misma se consideran importantes para definir el *stub*. En este momento se debe dar soporte a dos opciones para conseguir una herramienta flexible.

1. A una petición dada asignar una única respuesta.
2. A un número *N* de peticiones similares asignar una única respuesta.

La decisión de qué vía seguir debe tomarla el usuario de *cgd-wiremock*. La motivación de ofrecer esta característica es que para muchos entornos que consumen *cgd-wiremock* los datos no son relevantes. Por ejemplo, en el testeo de un algoritmo de ordenación en base a código de hotel que realiza dos peticiones paginadas, es necesario que los datos sean distintos. A la petición:

```
GET http://proveedor.com/rest/hoteles?limit=5&offset=0
```

la respuesta

```
{
    "hoteles": ['H1', 'H2', 'H3', 'H4', 'H5']
}
```

y a la petición

```
GET http://proveedor.com/rest/hoteles?limit=5&offset=5
```

la respuesta

```
{
    "hoteles": ['H6', 'H7', 'H8', 'H9', 'H10']
}
```

puesto que para comprobar el funcionamiento del algoritmo se necesita poder obtener el resultado

```
{
    "hoteles": ['H1', 'H2', 'H3', 'H4', 'H5', 'H6', 'H7', 'H8', 'H9', 'H10']
}
```

por otra parte si se quiere probar el rendimiento de un aplicativo que consume datos paginados y que realiza 10 peticiones de 5 en 5, los datos de respuesta de dichas peticiones son despreciables. De *Wiremock* sólo se desea que a una petición similar a

```
GET http://proveedor.com/rest/hoteles?limit=5&offset=0
```

se devuelva

```
{
    "hoteles": ['H1', 'H2', 'H3', 'H4', 'H5']
}
```

por lo tanto se debe poder crear un único *stub* que a estas 3 peticiones

```
GET http://proveedor.com/rest/hoteles?limit=5&offset=0
GET http://proveedor.com/rest/hoteles?limit=5&offset=5
GET http://proveedor.com/rest/hoteles?limit=5&offset=10
```

devuelva

```
{
    "hoteles": ['H1', 'H2', 'H3', 'H4', 'H5']
}
```

El proceso de creación del *stub* para la petición de ejemplo deberá ser el siguiente:

1. Se decide si se quiere igualar exclusivamente a la petición o a un conjunto de peticiones similares.
2. Se estudia la petición para entender qué características de la misma deben definirse en el *stub*, por ejemplo el método http o el filtro de búsqueda en el cuerpo de la petición.
3. Se define la respuesta que debe entregar *Wiremock* a una petición igualada por una definida en un *stub* cargado.
4. Se carga el *stub* en *Wiremock*.

Una vez se han realizado estos pasos, se puede levantar la máquina de *Wiremock*, por ejemplo en la *URI http://cgd-wiremock.test.com*, y al realizar una petición al puerto 8089 este intentara igualarla a algún *stub* configurado. Si lo encuentra devolverá la respuesta definida.

Se toma la petición y se *mockeará* como conjunto

```
GET http://proveedor.com/rest/hoteles?limit=5&offset=0
```

se estudia y decide que solamente se deberá igualar el método *http* y que se pueden obviar los parámetros de consulta. Se le asigna la siguiente respuesta

```
{
    "hoteles": ['H1', 'H2', 'H3', 'H4', 'H5']
}
```

y se define el *stub*. Una vez definido, al realizar las siguientes tres peticiones

```
GET http://proveedor.com/rest/hoteles?limit=5&offset=0
GET http://proveedor.com/rest/hoteles?limit=5&offset=5
GET http://proveedor.com/rest/hoteles?limit=5&offset=10
```

Wiremock siempre responde

```
{
    "hoteles": ['H1', 'H2', 'H3', 'H4', 'H5']
}
```

ya que solamente se le ha definido en el *stub* que a un *GET* a esta *URI /proveedor.com/rest/hoteles* devuelva

```
{
    "hoteles": ['H1', 'H2', 'H3', 'H4', 'H5']
}
```

Como se ha comentado anteriormente *Wiremock* proporciona más funcionalidad que la necesaria en *CGD* puesto que solo se trabaja con un nicho de mercado y por lo tanto las *API*s a simular son similares entre sí. Se analizarán qué tipos de peticiones se realizan a los clientes, así como las respuestas que se reciben de los mismos. De esta forma se acoplará *Wiremock* a los requisitos de *CGD*.

3.6 Peticiones de los proveedores

Se tiene en cuenta que hay que facilitar la mayor cantidad de filtros de petición posibles para así poder configurar la mayor cantidad de *stubs* dando soporte a simular la mayor cantidad de proveedores.

La comunicación con los proveedores actuales de *CGD* se realiza en tres formatos: *String*, *XML* y *JSON* lo que significa que al realizan peticiones a los proveedores se envían en uno de los tres formatos y que las respuestas que se reciben también están en uno de estos formatos. Por otra parte *CGD* está diseñado para recibir las peticiones en formato *JSON* y únicamente mediante método *POST*.

Se deben tener en cuenta estas características puesto que el primer paso que se realiza en el análisis es la definición de la información necesaria a almacenar para que una petición o conjunto de peticiones dado se pueda igualar inequívocamente.

Se diferencian tres tipos de implementaciones, es decir, formas en las que los proveedores han diseñado sus *API*s. La primera conocida como *SOAP* es un protocolo estándar que define cómo dos procesos pueden comunicarse mediante *XML*. La segunda, *REST* define un conjunto de principios arquitectónicos por los que se pueden diseñar servicios Web que se centran en los recursos de un sistema, lo que incluye la forma en que los estados de los recursos se dirigen y transfieren a través de *HTTP* por un amplio rango de clientes que están escritos en diferentes lenguajes. En el caso de *REST*, generalmente aunque no necesariamente, las comunicaciones se realizan mediante *JSON*. Finalmente se encuentran las *API*s propias, aquellas que han sido diseñadas por los mismos proveedores y no siguen ningún estándar.

3.6.1 *URI*

Todas las implementaciones anteriormente mencionadas precisan de una *URI* para funcionar es decir la dirección web del servidor que espera tales peticiones así como la *API*, puerto, o acción, en función de la implementación. A continuación se muestran ejemplos de las *URI*s estudiadas.

La primera *URI* que se estudió corresponde a un ejemplo de *API* implementada en *SOAP*. Su principal característica es indicar el servicio al que se está atacando en la última parte de la misma mediante *"/hotelBookingHandler"*, además de la versión del servicio en *"/v1_0rc1"*.

```
http://test.xtravelsystem.com/public/v1_0rc1/hotelBookingHandler
```

La segunda es un *endpoint* de una *API* implementada siguiendo el patrón *REST*. En la misma se puede ver la versión de la *API* *"/v3"*, el idioma en el que se quiere que se devuelva el contenido *"/es"* y después, siguiendo las directrices de la arquitectura *REST*, el tipo de objeto que se quiere recuperar *"/polizas"* el identificador de la póliza a recuperar *"/19229"* y la acción sobre la misma, en este caso las opciones, *"/opciones.json"*.

```
https://api-test.intermundial.com/v3/es/polizas/19229/opciones.json
```

Se denota que en ambas *URI*s hay información parametrizable. Para dar soporte a la funcionalidad de igualar exclusivamente a una petición se debe poder definir un *stub* para *mockear* la respuesta a

```
https://api-test.intermundial.com/v3/es/polizas/19229/opciones.json
```

que devuelve las opciones de la póliza 19229.

Por otra parte, si para el caso de uso que se quiere dar, no es importante a qué póliza hacen referencia las respuestas de opciones, sino que lo que importa es que a una petición de opciones de póliza se devuelva una respuesta con la estructura de opciones

de póliza pero sin importar los datos que contiene. En tal caso el trozo de la *URI* que hace referencia al identificador tiene que ser despreciable, indicando mediante *???* que el identificador de la póliza no se tendrá en cuenta en el proceso de identificar la petición. La *URI* toma la siguiente forma:

```
https://api-test.intermundial.com/v3/es/polizas/???/opciones.json
```

3.6.2 Método *http*

Debido a que existen proveedores que hacen uso de arquitectura *REST* será necesario poder diferenciar las peticiones en base a su método http. El motivo por el cual esta característica debe ser soportada es el siguiente.

En una *API REST* se utilizará la *URI*, para realizar todas las acciones disponibles sobre las habitaciones de un hotel con identificador id.

```
http://proveedor.com/rest/hoteles/{id}/rooms
```

es decir, si se quiere reservar habitaciones en el hotel se realizará la siguiente llamada,

```
POST http://proveedor.com/rest/hoteles/{id}/rooms
```

si se quieren realizar cambios sobre habitaciones ya reservadas

```
PUT http://proveedor.com/rest/hoteles/{id}/rooms/{idRoom}
```

y si se desea cancelar la reserva de alguna habitación

```
DELETE http://proveedor.com/rest/hoteles/{id}/rooms/{idRoom}
```

Se comprueba que con la *URI* no siempre es suficiente para saber qué respuesta se debe devolver puesto que la respuesta a las 3 acciones no debe ser la misma. Por lo tanto la posibilidad de tener en cuenta qué método http se utiliza al realizar la petición es necesaria.

3.6.3 Cabeceras *http*

Se detecta otra característica a soportar, puesto que existen *APIs* que en base a la cabecera devuelven una respuesta u otra, como ejemplo el proyecto padre en el que se sitúa la aplicación *cgd-wiremock*, *CGD*.

Al realizar una petición con la cabecera *"Accept: application/json"* se devuelve la respuesta en formato *JSON* pero si la cabecera es *"Accept: application/xml"* la respuesta

se devolverá en formato *XML*. En base a la cabecera se han generado dos posibles peticiones que se deben poder diferenciar.

3.6.4 Parámetros de consulta

Otra característica que ofrecen los proveedores es el filtrado de petición por parámetros de consulta, es decir, en la *URI*.

Según el proveedor en el siguiente *endpoint* se encuentran todos los tipos de habitaciones disponibles en el hotel con identificador id. Además como el registro de habitaciones disponibles es muy grande, el proveedor permite paginación en las peticiones mediante parámetros de consulta, donde *limit* indica cuántas habitaciones se quieren recuperar y *offset* desde qué punto se quieren recuperar.

Así, la consulta anterior recupera las diez primeras habitaciones disponibles en el hotel id

```
http://proveedor.com/rest/hoteles/{id}/rooms/catalogue?limit=10&offset=0
```

Si se quieren recuperar las 10 siguientes sería:

```
http://proveedor.com/rest/hoteles/{id}/rooms/catalogue?limit=10&offset=10
```

por lo tanto se encuentra que un posible factor diferenciador entre peticiones son los parámetros de consulta.

3.6.5 Cuerpo de la solicitud

En *API*s basadas en *SOAP* y en ocasiones en las de diseño propio, se encuentra el caso en el que el filtro de la petición está en el cuerpo de la solicitud. Es decir, la característica que va a indicar si se trata de una petición u otra va a estar en el cuerpo de la misma.

Se dan tres formatos en los cuerpos de solicitud a los que hay que dar soporte: cadena de caracteres planos, es decir, *String*, formato *JSON* y *SOAP*.

En el caso *SOAP* el cuerpo de la solicitud contiene prácticamente toda la información sobre la petición. Se aprecian tres características diferenciales en la petición. El *tag* *“<AvailabilityRQ>”* se conoce como *soap action* y hace referencia a la operación que se quiere realizar sobre un servicio *SOAP* por ejemplo al atacar al siguiente *endpoint* que hace referencia al servicio *hotelBookingHandler*, el caso de uso sería recuperar la disponibilidad de hoteles.

En el siguiente código se observa que existen *tags* dentro de *“<SOAP-ENV:Header>”* y *“<SOAP-ENV:Body>”*, cualquiera de esos *tags* puede ser una característica de la petición. En el caso que nos ocupa se observa como se indica un usuario y contraseña, ya que se puede querer simular respuestas distintas en base a esos campos. También se

observa el filtro por ciudad, Madrid, puesto que la respuesta simulada debe ser distinta si se pide disponibilidad en Madrid que si se pide en Barcelona.

```
http://test.xtravelsystem.com/public/v1_0rc1/hotelBookingHandler
```

```
<SOAP-ENV:Envelope>
    <SOAP-ENV:Header>
        <AvailabilityRQ>
            <User>user</User>
            <Password>password</Password>
        </AvailabilityRQ>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <City>Madrid</City>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Por otra parte en el caso en que el cuerpo de la solicitud sea una cadena de caracteres plana cualquier subcadena dentro de la misma puede ser diferencial y característica de la petición. En este caso el proveedor ha decidido que los filtros en sus peticiones de disponibilidad se implementen de la siguiente manera: conjuntos de dos cadenas de caracteres separadas por *“::”* que a su vez están separadas entre sí por *“<=>”*. Al no estar estandarizado como *JSON* o *XML* se tiene que estudiar cómo diferenciar peticiones en las que la solicitud entre dentro de este grupo.

```
'Ciudad::Madrid<=>Barrio::Malasaña'
```

Por último *JSON* sí que es una formato estandarizado y los proveedores lo utilizan de la siguiente manera en los cuerpos de las solicitudes. Como sucede con un cuerpo en formato cadena de caracteres, cualquier conjunto de clave y valor equivale a un filtro diferenciador de petición.

Se tiene en cuenta que para el caso *SOAP* y *JSON* es necesario, al igual que para el caso cadena de caracteres, dar soporte a que se puedan diferenciar peticiones en base a contenido parcial, por ejemplo.

Se tienen dos peticiones *JSON* donde para la clave Barrio se tienen los valores Barrio Bajo y Barrio Alto respectivamente. Por lo tanto se tiene que poder definir tanto que el contenido de la clave Barrio sea igual a Barrio Bajo o a Barrio Alto. Puesto que la clave Barrio contiene la palabra Barrio en su valor, se agrupan así las dos peticiones en una misma simulación.

```
{
"Ciudad":"Lisboa",
"Barrio":"Barrio Alto"
}
```

```
{
"Ciudad":"Lisboa",
"Barrio":"Barrio Bajo"
}
```

3.6.6 Escenarios

Otra característica a la que se debe dar soporte aunque no forma parte de una petición sino de un conjunto de petición o flujo es el concepto de escenarios. Se entenderá que los escenarios son como los estados de una máquina de estados y estos podrán ser cambiados cuando se realice una petición contra *Wiremcok* Son útiles para casos como el siguiente, que devuelve la información de la reserva 123

```
http://proveedor.com/rest/hoteles/{id}/reservas/123
```

se supone que se quiere *mockear* el siguiente flujo:

1. Consultar el estado de la reserva 123.
2. Si está en estado reservado, cancelarla.
3. Consultar que se ha cancelado la reserva 123

El flujo de peticiones sería el siguiente: como se observa, la primera y última petición son idénticas pero las respuestas que se deben *mockear*, son distintas puesto que en la primera el estado debe ser reservado y en la tercera cancelado.

```
GET http://proveedor.com/rest/hoteles/{id}/reservas/123
DELETE http://proveedor.com/rest/hoteles/{id}/reservas/123
GET http://proveedor.com/rest/hoteles/{id}/reservas/123
```

3.6.7 Respuestas de los proveedores

A continuación se definen los requisitos que deben cumplir las respuestas a las peticiones definidas en las simulaciones. Estas vienen impuestas por los entornos que van a utilizar *cgd-wiremock*. Sus requerimientos son:

1. Configurar el estado http de la respuesta, es decir: *200*, *404*, *401*...
2. Configurar el mensaje de estado de la respuesta, por ejemplo, para un estado *401: "el usuario no tiene acceso con los credenciales proporcionados"*
3. Incluir un cuerpo de respuesta en formato *String*, *JSON* y *XML*.
4. Configurar las cabeceras *http* de la respuesta.

3.6.8 Estudio de acoplamiento a *Wiremock*

Una vez se conocen las características de las peticiones a identificar y las respuestas a simular se procede a estudiar cómo se pueden acoplar dichas necesidades a las funcionalidades que ofrece *Wiremock* con el fin de simplificar y adaptar todo el abanico de posibilidades a los requisitos del proyecto.

Para la definición programática de los campos: método *http* de la petición, estado de la respuesta, mensaje de estado de la respuesta y cuerpo de la respuesta. Se definirán directamente utilizando la *API* de *Wiremock* puesto que no hay flexibilidad sobre estos campos.

A continuación los campos para los que sí que se ha adaptado *Wiremock* a las necesidades de *CGD*.

URI

Se ha utilizado la funcionalidad *"urlPathMatching()"* que acepta expresiones regulares sobre las *URI*s de manera que se consigue el objetivo analizado anteriormente de poder identificar dos peticiones diferentes como la misma. Para identificar estas dos peticiones podría hacerse de forma única si se incluye completa o de forma ambigua de la siguiente manera:

```
http://proveedor.com/rest/hoteles/{id}/reservas/123
http://proveedor.com/rest/hoteles/{id}/reservas/321

http://proveedor.com/rest/hoteles/{id}/reservas/([0-9]*)
```

Parámetros de consulta y cabeceras *http*

A la hora de definir los parámetros de consulta de la petición así como las cabeceras de la petición y la respuesta, se utilizará la funcionalidad *"matching()"* que ofrece *Wiremock*. Esta funcionalidad permite el uso de expresiones regulares. De esta forma se puede definir que se devuelva la respuesta X a cualquier petición que contenga una cabecera del tipo *"Accept: application/.*"* donde *".*"* puede ser cualquier cadena de texto. Se reduce al mínimo la restricción puesto que no se ha encontrado necesidad de no hacerlo.

Cuerpo de la solicitud

JSON

Para este campo en caso de ser de tipo *JSON* se utilizará la funcionalidad *"equalToJson()"*. Esta funcionalidad asume que el cuerpo entero de la solicitud es *JSON* y busca en el mismo un objeto *JSON* igual al definido. Además se configura para que ignore el orden en los *arrays* y los elementos extra.

Cuerpo de la solicitud

```
{
    "ciudad": "Madrid",
    "barrios": ["Malasaña", "Chueca"]
}
```

Se tiene que identificar el siguiente cuerpo de la solicitud que es para filtrar por ciudad y barrios. En caso de considerarse que todas las respuestas por ciudad deben ser la misma, es decir, que no importan los barrios, la siguiente definición sería la óptima.

```
{
    "ciudad": "Madrid"
}
```

Por otra parte, si la ciudad es indiferente pero los barrios no, se utilizará esta.

```
{
    "barrios": ["Chueca", "Malasaña"]
}
```

Se debe tener en cuenta que en el negocio del turismo el orden en las listas no es significativo, por ello se configura que se desprecie el orden en las listas y que por lo tanto al utilizar la segunda definición se entenderán como idénticas las peticiones que contengan

```
{"Barrios": ["Chueca", "Malasaña"]}
{"Barrios": ["Malasaña", "Chueca"]}
```

STRING

Para el caso de cuerpos de solicitud con formato en cadena de caracteres *Wiremock*, se escoge la opción más completa que es la funcionalidad *"containing()"*. Esta permite identificar peticiones en base a su contenido parcial. Las tres peticiones siguientes se identificarán como iguales cuando se defina la solicitud como *"Madrid"* puesto que la palabra Madrid está contenida en los tres cuerpos de solicitud. Si se desea identificar únicamente el primero, se debe ser explícito con toda la cadena de caracteres. En caso de no poder ser explícito, se debe buscar otra manera de identificar inequívocamente la petición.

1. *"Ciudad::Madrid<=>Barrio::Malasaña"*

2. *"Ciudad::Madrid"*

3. *"Barrio::Madrid central"*

SOAP

Se dividirá la petición en las tres partes mentadas en el apartado de análisis, y se utilizará *XPATH* para buscar el *soap action* en el árbol *xml*. Para la definición de campos en el *"<SOAP-ENV:Header>"* y *"<SOAP-ENV:Body>"* se permitirá filtrar por tantos como se desee. Para dar soporte a esta funcionalidad se utilizará también *XPATH* y los *tags* dentro de estos atributos podrían compararse de manera estricta o no mediante las funcionalidades *equalto()* o *contains()*. Se han establecido estas restricciones para aceptar la mayor cantidad de definiciones de cuerpos de solicitud en peticiones *SOAP*. En el siguiente cuerpo de solicitud se podrá identificar como única indicando que el *soap action* es *AvailabilityRQ* o bien que en el *tag Useranme* del *SOAP-ENV:Header* tiene que haber el valor *user*, también que en el *tag City* del *SOAP-ENV:Body* tiene que haber el valor Madrid. Se podrán añadir las tres restricciones a la vez. Cabe recordar que cuanto más se restringe, más inequívocamente se puede definir una petición.

```
<SOAP-ENV:Envelope>
    <SOAP-ENV:Header>
        <AvailabilityRQ>
            <User>user</User>
            <Password>password</Password>
        </AvailabilityRQ>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <City>Madrid</City>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Escenarios

Wiremock proporciona un conjunto de herramientas para definir estados y por lo tanto en la definición de un *stub* se puede definir el escenario en el que ese *stub* debe ser considerado igual a la petición y cómo deja configurada la máquina de estados una vez ha sucedido. Entonces se puede configurar que una vez recibida la primera petición se devuelva la respuesta con el estado de reserva en reservado, ya que la máquina de estados se encuentra en estado "inicial" y que luego se configure el escenario "reservado". A continuación se realiza la petición *DELETE* y no cambia el estado. Al realizar la tercera petición se ha configurado el *stub* para que la iguale siempre y cuando el estado de la máquina sea "reservado" y entonces se devuelve la respuesta con estado cancelado. Además se configura la máquina para volver al estado "inicial" y así poder repetir el flujo. El flujo de peticiones sería el siguiente:

```
GET http://proveedor.com/rest/hoteles/{id}/reservas/123
DELETE http://proveedor.com/rest/hoteles/{id}/reservas/123
GET http://proveedor.com/rest/hoteles/{id}/reservas/123
```

3.7 Almacenamiento de los *stubs*

Como restricción en el desarrollo de la aplicación *cgd-wiremock*, se impone por parte del departamento de arquitectura que se evite el uso de bases de datos y que se tiene

que conseguir la persistencia de las suites de *stubs* definidas. De esta manera se facilita el despliegue de una máquina *cgd-wiremock* en la que cargar toda la suite ya definida en otra máquina.

Para dar soporte a esta funcionalidad se diseñarán los *stubFiles*, es decir, los ficheros únicos que representan un *stub*. Se almacena el identificador del mismo y la petición que realizó el usuario para crear el *stub*. De esta manera mediante acción manual o un programa automático se podrá repetir la operación de generar cada *stub* en cualquier máquina de *cgd-wiremock*.

Esta solución pese a ser la tomada no es la más eficiente y en el apartado de mejoras se desarrollaran opciones más óptimas para el almacenaje de *stubs*.

3.8 *cgd-wiremock-frontal*

Hasta ahora no se ha analizado la forma en la que el usuario debe interactuar con *cgd-wiremock* para la creación y mantenimiento de *stubs*, así como la consulta de los mismos. En el siguiente apartado se analizarán los requisitos que debe tener la aplicación web que se utilizará para llevar a término las tareas mencionadas.

El primer requisito es que la aplicación web sea rápida y multiplataforma, es decir, que sea usable en múltiples dispositivos, ordenadores, tabletas y móviles. Para llevar a término esta funcionalidad se desarrollará la aplicación como *SPA (Single Page Application)*. Este tipo de aplicaciones web descargan todo el contenido web *JS, HTML y CSS* una única vez. Posteriormente realizan peticiones al servidor para recuperar la información o datos necesarios. De esta manera se evitan refrescos y los componentes de la aplicación aparecen y desaparecen al estilo aplicación de escritorio, dando una mejor experiencia de usuario. Para que la aplicación sea multidispositivo, es decir, que mantenga proporciones adecuadas ya sea en un ordenador como en un móvil y se reestructure si es necesario para ofrecer una mejor experiencia de usuario, se utilizará el *framework Bootstrap 3* que incluye toda una suite de estilos para dar soporte a dicha funcionalidad.

Se deben cumplir el resto de requisitos en el frontal:

1. En la aplicación el usuario debe poder configurar el servidor de *cgd-wiremock* que se usará.
2. La página de inicio de la misma será una vista de tabla con el registro de los *stubs* actuales. La tabla debe cumplir:
   - Ser paginable en grupos de 10, 25, 50 y 100 registros.
   - Soporte a ordenación por columnas.
   - Mostrar las columnas necesarias para identificar un *stub*.
   - Ser rápida al cargar.
   - Tener un botón de edición del *stub* y otro de detalle del mismo.
   - Tener un campo de búsqueda de texto en la tabla.

3. La página de formulario de creación y edición de *stubs* debe permitir realizar todas las opciones analizadas.

4. Permitir vista en texto plano del *stub* desde donde se pueda descargar en formato *JSON* el *stubFile*.

En el apartado de diseño se exponen los diagramas de uso e implementación del diseño del frontal.

CAPÍTULO 4

DISEÑO E IMPLEMENTACIÓN

A continuación se desarrolla como se han enfrentado los distintos apartados del análisis comenzando por la interacción entre cada una de las partes y los flujos de uso.

4.1 Diagramas de flujo y uso

4.1.1 *cgd-wiremock*

A continuación el diagrama de flujo en la figura 4.1 muestra la creación de un *stub* en *cgd-wiremock*

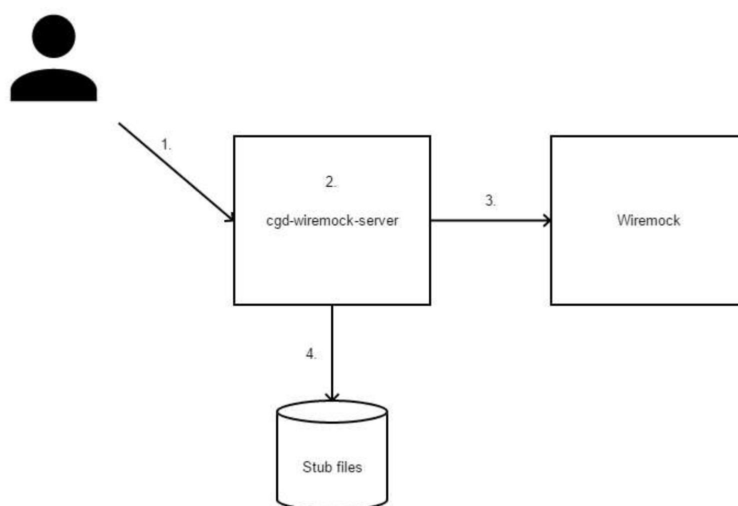

Figura 4.1: Creación de un *mock* en *cgd-wiremock*

1. El usuario envía una petición de creación de *stub* al servidor de *cgd-wiremock*.
2. *cgd-wiremock* procesa la petición y la transforma a una serie de instrucciones para programáticamente registrar el *stub* en *Wiremock*.
3. Registro del *stub* en *Wiremock*.
4. Almacenado en fichero del *stubFile* para poder replicar manual o automáticamente la creación del *stub*.

4.1.2 Wiremock

A continuación el diagrama de la figura 4.2 muestra la interacción entre usuario y *Wiremock* para consumir los *mocks* definidos mediante *stubs*.

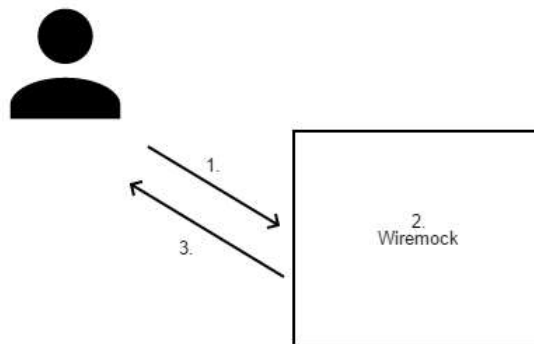

Figura 4.2: Interacción entre el usuario y *Wiremock*

1. El usuario realiza una petición a *Wiremock*.
2. *Wiremock* procesa la petición del usuario y busca entre los *stubs* si hay alguna definición de petición que iguale a la recibida.
3. En caso de encontrarla devuelve al usuario la respuesta configurada para esa petición en el *stub*.

4.1.3 *cgd-wiremock-frontal* creación, edición y borrado de *stubs*

A continuación el diagrama de la figura 4.3 muestra la interacción entre usuario y *cgd-wiremock-frontal* para la creación, edición y borrado de *stubs*.

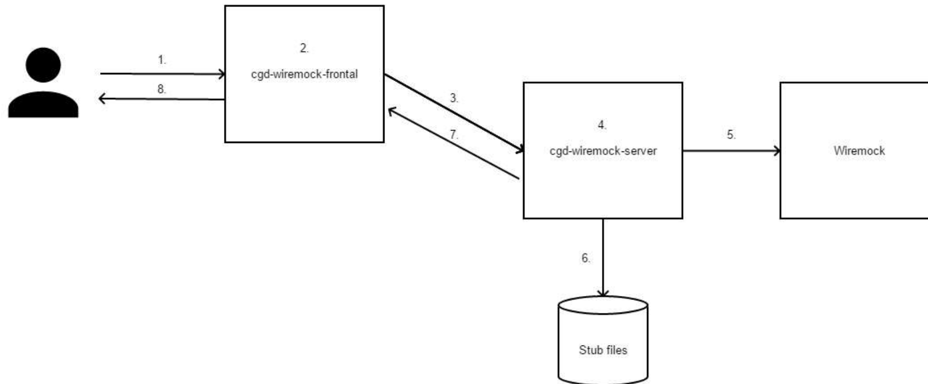

Figura 4.3: Creación, edición y borrado de *stubs* en *cgd-wiremock*

1. El usuario rellena, actualiza o presiona el botón de borrado de *stub* en *cgd-wiremock-frontal*.
2. El frontal procesa el evento y transforma los inputs de usuario a datos que puede enviar a *cgd-wiremock*.
3. Envía la petición a *cgd-wiremock*.
4. *cgd-wiremock* procesa la petición y la transforma a una serie de instrucciones para programáticamente registrar el *stub* en *Wiremock*.
5. Registro, actualización o borrado del *stub* en *Wiremock*.
6. Almacenado, actualización o borrado del *stubFile*.
7. Respuesta del estado del flujo al frontal, todo correcto o error en caso de error.
8. Muestra al usuario la información sobre si la acción deseada se ha llevado a cabo correctamente.

4.1.4 *cgd-wiremock-frontal* detalle y listado de *stubs*

A continuación el diagrama de la figura 4.4 muestra la interacción entre usuario y *cgd-wiremock-frontal* para el listado y detalle de *stubs*.

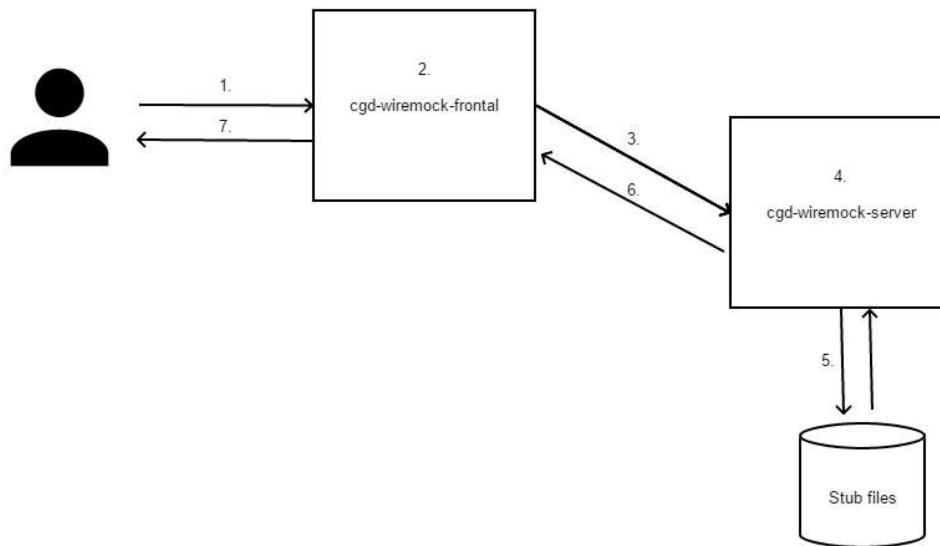

Figura 4.4: Listado y detalle de *stubs* en *cgd-wiremock*

1. El usuario accede a la página de listado de *stubs* o detalle de un *stub*.
2. El frontal procesa el evento y prepara la petición a enviar a *cgd-wiremock-server*.
3. Envía la petición a *cgd-wiremock-server*.
4. *cgd-wiremock-server* procesa la petición.
5. *cgd-wiremock-server* hace la lectura de los ficheros *stubFile*.
6. Devuelve los *stubFiles*.
7. *cgd-wiremock-frontal* muestra la tabla de registros o el detalle de un *stub* al usuario.

Una vez diseñados los flujos de la aplicación y comunicación entre ellas, se procede al diseño de los componentes que harán posible dicha comunicación, comenzando por los *subFiles*.

4.2 *StubFiles*

A continuación se define la estructura del fichero de almacenaje de *stubs* que a su vez es la estructura de la petición a realizar a *cgd-wiremock* para crear o actualizar un nuevo *stub*.

1. **filename**: Nombre del fichero, debe ser único y es responsabilidad del desarrollador.

2. **scenario**: Define el escenario para esta solicitud, permite diferentes respuestas a la misma solicitud dependiendo del escenario actual.
    a) **name**: Nombre del escenario.
    b) **whenStateIs**: Estado en el que se debe devolver esta respuesta.
    c) **triggerState**: Estado en el que debe quedar la máquina al devolver esta respuesta.
3. **request**: Define la petición a ser igualada.
    a) **urlPattern**: *URI* (campo versátil para mayor comodidad, puede contener expresiones regulares y parámetros de consulta si es necesario).
    b) **method**: Método *http*. por ejemplo: *GET*.
    c) **headers**: Filtros de cabeceras http en la forma *<key>: {"<predicate>": "<value>"}*.
    d) **queryParameters**: Filtros de parámetros de consulta en la forma *<key>: {"<predicate>": "<value>"}*.
    e) **bodyPattern**: Definición del cuerpo de la solicitud a igualar, solo uno de los tres atributos debe rellenarse.
        i. **jsonFilter**: Patrón para igualar a cuerpos en formato *JSON* en la forma *<key>: {"<predicate>": "<value>"}*.
        ii. **xmlFilter**: Patrón para igualar a cuerpos en formato *SOAP* o *XML*.
            A. **actionObjectName**: Nombre del objeto del cuerpo *SOAP*.
            B. **headerFilters**: Filtros en la cabecera *SOAP* con el patrón *<key>: {"<predicate>": {"<restriction>": "<value>"}}* Donde la restricción puede ser *equalTo* o *contains*.
            C. **bodyFilters**: Filtros en el cuerpo *SOAP* con el patrón *<key>: {"<predicate>": {"<restriction>": "<value>"}}* Donde la restricción puede ser *equalTo* o *contains*.
    f) **stringFilter**: Iguala un cuerpo de solicitud que contenga el *String* especificado.
4. **response**: Define la respuesta en caso de igualarse la petición.
    a) **status**: Estado http de la respuesta.
    b) **statusMessage**: Mensaje de estado http de la respuesta.
    c) **body**: Respuesta.
    d) **headers**: listado de las cabeceras de respuesta *<key>: {"<predicate>": "<value>"}*.

Durante la definición del *stubFile* se utilizan las expresiones *<key>: {"<predicate>": "<value>"}* o *<key>: {"<predicate>": {"<restriction>": "<value>"}}*. El significado de tales expresiones es: el campo *<key>* hace referencia al atributo de la petición, por ejemplo, al rellenar cabeceras *<key>* equivale a *headers*. *"<predicate>"* por su parte hace referencia al nombre del atributo o *tag* que se quiera filtrar se habla de cabeceras puede ser *"Accept"*. Por último *"<value>"*, es el valor del mismo, *"application/json"*. De esta forma si

se quiere filtrar la petición con una cabecera *"Accept: application/json"*, debería enviarse el campo *headers* de la siguiente manera:

```
headers: {"Accept": "application/json"}
```

Asimismo se puede añadir en algunos casos una restricción que permita indicar la igualdad estricta o solo que contenga lo que se quiere filtrar, volviendo al cuerpo *xml* del ejemplo en el apartado de análisis.

```
<SOAP-ENV:Envelope>
    <SOAP-ENV:Header>
        <AvailabilityRQ>
            <User>user</User>
            <Password>password</Password>
        </AvailabilityRQ>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <City>Madrid</City>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

En caso de querer filtrar esta petición en base a la ciudad se configuraría de la siguiente manera:

```
{
    "request": {
            "bodyPattern": {
                "xmlFilter": {
                    "actionObjectName": "AvailabilityRQ",
                    "headerFilter": {
                        "Password": {
                            "contains": "pass"
                        }
                    },
                    "bodyFilter": {
                        "City": {
                            "equalTo": "Madrid"
                        }
                    }
                }
            }
        }
}
```

Se diseña una pequeña *API* descrita en el próximo apartado para la integración con *cgd-wiremock*.

4.3 *API cgd-wiremock*

A continuación en la figura 4.1 Se muestran todos los endpoints de la *API*, incluyen el prefijo */mocks*

Método	Endpoint	Cuerpo	Respuesta	Descripción
GET	*/stubs*		*List<StubFile>*	Recupera todos los *stubs*
GET	*/stubs/{filename)*		*StubFile*	Recupera un *stub*
POST	*/stubs*	*StubFile*	*StubFile*	Crea un *stub*
PUT	*/stubs*	*StubFile*	*StubFile*	Actualiza un *stub*
DELETE	*/stubs/{filename)*			Borra un *stub*
POST	*/stubs/suites/load*			Carga una suite de *stubs**
*Para cargar una suite de *stubs* hay que depositar los *stubFiles* en la *wiremock/stubFiles* que debe situarse en la carpeta base desde donde se arranque el servidor.				

Cuadro 4.1: Acciones disponibles en la *API*.

Como se aprecia se trata de un *CRUD (Create, Read, Update, Delete)*, es decir permite la Creación, Lectura, Actualización y Borrado de *stubs*. También soporta la migración de una suite entera de *mocks* ya definidos en otro entorno a uno nuevo que se vaya a desplegar.

4.4 Aplicación *Java cgd-wiremock*

Para el desarrollo de la aplicación *Java cgd-wiremock* se han seguido las directrices que dicta *Spring Boot 2* en el diseño de aplicaciones web, como se ve en la figura 4.5.

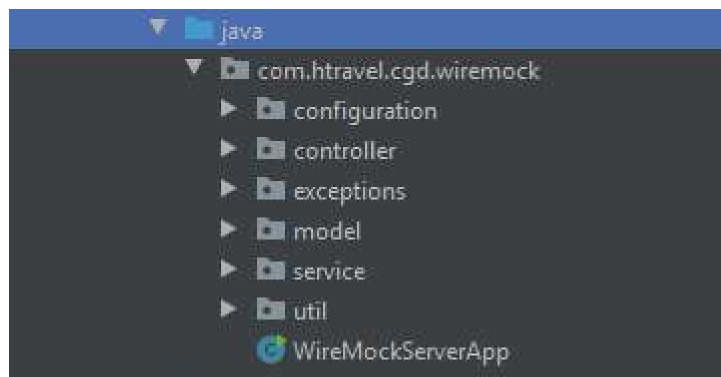

Figura 4.5: Estructura de carpetas en *Java cgd-wiremock*

De esta estructura de ficheros cabe destacar los siguientes puntos:

1. El modelo de datos se compone de objetos sin comportamiento que son manipulados por los servicios.

2. Los servicios se encargan de llevar a cabo toda la lógica del aplicativo y el procesado de datos.

3. Los controladores definen los endpoints que están disponibles y se encargan únicamente de recibir las peticiones, llamar al servicio correspondiente y devolver la respuesta. No contienen ninguna lógica.

Para la implementación de la lógica de la aplicación se han utilizado dos servicios, primero se ha diseñado una interfaz que soporte las operaciones necesarias, *MockService*, tal y como se aprecia en la siguiente figura 4.6

```
public interface MockService {

    JSONObject loadStoredStubs() throws MockServiceException;

    StubStoredInfoDTO getStub(String id) throws MockServiceException;

    StubStoredInfoDTO createStub(CreateStubRequest createStubRequest, boolean isWithMapping) throws MockServiceException;

    StubStoredInfoDTO updateStub(CreateStubRequest createStubRequest) throws MockServiceException;

    void deleteStub(String id) throws MockServiceException;

    List<StubStoredInfoDTO> listStubStoredInfo() throws MockServiceException;
```

Figura 4.6: Creación, edición y borrado de *stubs* en *cgd-wiremock*

De esta manera se podría implementar con otra tecnología el servicio y seguiría siendo compatible con la aplicación actual.

Se ha desarrollado también el servicio que se encarga del tratado y manejo de *stubFiles*, *StubManagerService*. Dicho servicio recibe *stubFiles* y se encarga de montar los objetos necesarios para luego poder crear los *stubs* en *Wiremock*. Por ejemplo, analiza el método http y comienza la definición en base a este:

```
WireMock.post(urlPathMatching(effectiveUrl(createStubRequest)));
```

o

```
WireMock.get(urlPathMatching(effectiveUrl(createStubRequest)));
```

Como ventaja destacar que descarga al programador de las operaciones más complejas en la definición programática de *stubs* al ofrecer utilidades que permiten añadir un filtro sobre el cuerpo de una petición *SOAP* con una simple instrucción:

```
buildBodyActionXpath(String actionTag)
```

Sin esta operación el desarrollador debería conocer el *XPATH* necesario para filtrar el *XML*.

4.5 Aplicación *JS cgd-wiremock-frontal*

Para el diseño e implementación del frontal se tomará el camino de diseño por componentes, los pasos a seguir en esta forma de diseño son lo siguientes.

1. Se plantean unos bocetos de cómo debe ser la web.

2. Se divide dicha web en trozos, componentes, que tendrán su propia responsabilidad y funcionalidad.

3. Se procede a la implementación.

4.5.1 Listado de stubs

En la figura 4.7 se puede apreciar la existencia una barra de navegación y una tabla que se puede refrescar. Además se encuentra la información del servidor *cgd-wiremock* al que se está atacando ya que debe ser configurable. En el rectángulo amarillo se tomará como componente base de la aplicación y contendrá únicamente la barra de navegación y los campos de configuración. Por otra parte el rectángulo azul irá cambiando y se tratará como contenido del componente base.

En este caso dentro del contenido encontramos el componente de tabla que incluye también el botón de refresco de la misma.

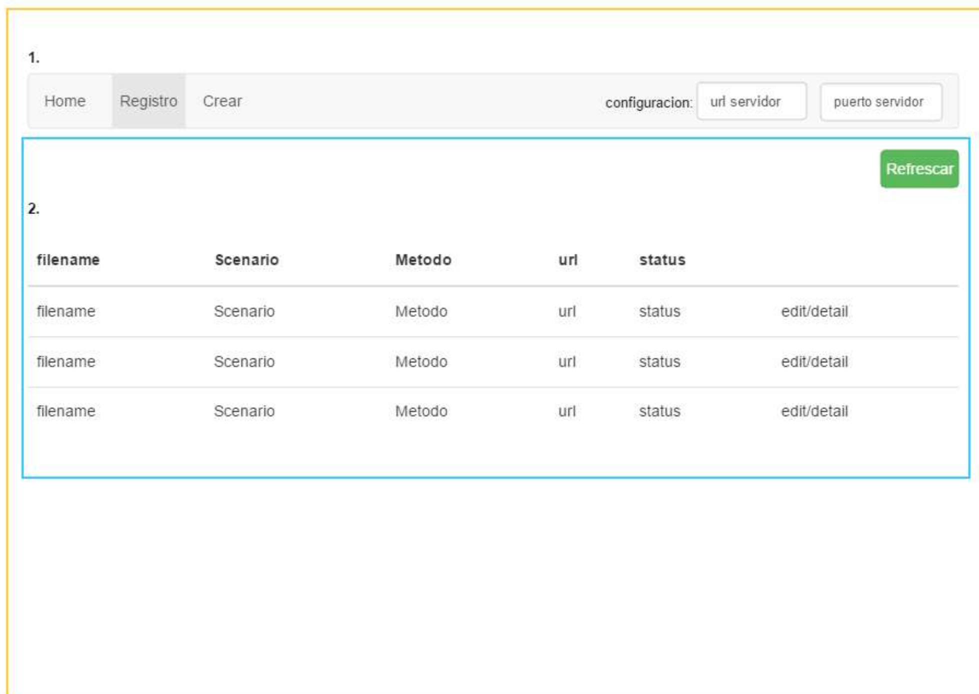

Figura 4.7: Listado de *stubs* en *cgd-wiremock-frontal*

4.5.2 Formulario de creación y edición de *stubs*

La figura 4.8 muestra la vista del formulario de creación y edición de *stubs* que se dividirá en tres componentes:

1. Componente base de formulario.
2. Componente de clave-valor.
3. Componente de filtro de cuerpo de petición, que se desarrollará más adelante.

Se aprecia la ventaja principal de la arquitectura de componentes en el desarrollo de aplicaciones web. Para el componente 2 que se reutiliza tres veces en esta página sólo se ha desarrollado un código y una maquetación y se puede reciclar a lo largo de la página.

Figura 4.8: Formulario de creación y edición de *stubs* en *cgd-wiremock-frontal*

4.5.3 Filtro de cuerpo de la solicitud

Para el filtro de cuerpo de solicitud de la figura 4.9 se tienen tres versiones en función del *tab* seleccionado. Se debe a que una petición sólo puede ser filtrada por uno de estos tres tipos de cuerpo. El primero para cuerpos de solicitud en formato *JSON* permite

añadir tantos filtros como se desee según las reglas descritas en el apartado de análisis. El segundo hace referencia a peticiones *SOAP* en las que el cuerpo puede filtrarse en base al nombre del objeto de acción, a cualquier *tag* en *SOAP-ENV:Header* o a cualquier *tag* en *SOAP-ENV:Body*. Permite especificar el tipo de restricción, mediante un selector que permite seleccionar *contais* o *equalTo*.

Finalmente el tercero ocupa los filtros en peticiones en las que el cuerpo de la solicitud es de tipo *String* y por tanto se buscará que la información introducida en el campo de texto esté contenida en el cuerpo de la solicitud de la petición a igualar.

Figura 4.9: Filtros de cuerpo de solicitud en *cgd-wiremock-frontal*

Finalmente se diseña la vista de detalle de un *stub* donde se podrá descargar el *stubFile* tal y como se muestra en la figura 4.10

Dowload

Stub:

Escenario

Nombre: **Estado actual:** **Estado siguiente:**

Definicion de la petición

Método: **Url:**

Cabeceras

Clave	Valor

Parametros

Clave	Valor

Cuerpo de la solicitud

Mostrando un cuerpo de tipo ...

Varia en funcion del cuerpo utilizando los componentes clave-valor y campo de texto

Definicion de la respuesta

Estado: **Mensaje de estado:**

Cabeceras

Clave	Valor

Cuerpo de la respuesta

Figura 4.10: Vista de texto plano en *cgd-wiremock-frontal*

4.5.4 Extructura de carpetas

En la figura 4.11 Se puede observar como la aplicación del frontal queda estructurada en los siguientes componentes

- **stub-app**: base de la aplicación, actúa como padre y maneja el estado de la misma.
- **stub-detail**: se encarga de mostrar en texto plano la información de un *stubFile*.
- **stub-dictionary-table**: se utiliza para mostrar los objetos de tipo clave-valor en el detalle, por ejemplo, las cabeceras de la petición.

- **stub-form**: se trata del componente que contiene el formulario de creación y actualización de un *stub*, así como la eliminación del mismo.
- **stub-info-available**: otro ejemplo de la agilidad que comporta la arquitectura de componentes, sirve para envolver cualquier otro componente y en caso de no tener información se muestra un mensaje advirtiendo que la información no está disponible.
- **stub-key-entry**: Se utiliza para los objetos de tipo clave-valor en los formularios.
- **stub-rq-xml-container**: En este caso se ha aplicado otra estrategia de la arquitectura de componentes. Cuando un trozo de la web tiene suficiente código o peso a alto nivel, se puede separar en un componente, de esta manera se garantiza mayor mantenibilidad y legibilidad. Se trata de la parte del formulario que hace referencia al cuerpo de solicitud en formato *SOAP*.
- **stub-table-container**: Este componente embebe el plugin de *jQuery DataTables* descrito en el apartado tecnológico, además permite el cacheo de datos. Si el usuario carga la tabla, esa información se guardará en memoria hasta que el usuario voluntariamente refresque la aplicación. Toda la lógica que conlleva este hecho está encapsulada en el código del componente y es una característica de cualquier tabla que se muestre mediante este componente, otra ventaja de la arquitectura.

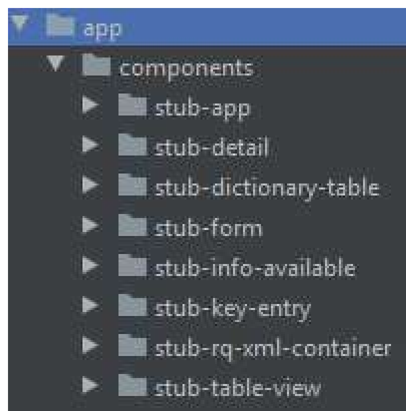

Figura 4.11: Estructura de carpetas para los componentes en *cgd-wiremock-frontal*

Capítulo 5

Resultados

5.1 *Mockeando* una petición

En este apartado se muestra el proceso por el cual se *mockea* una petición en *cgd-wiremock* mediante el uso de *cgd-wiremock-frontal*. Se comprobara que se ha *mockeado* correctamente buscando el *stub* en el listado de *stubs* y accediendo al detalle del mismo. Finalmente se realizará la petición utilizando *Postman* al *Wiremock* desplegado en *cgd-wiremock* asegurando así el correcto funcionamiento de los aplicativos.

Se tiene la siguiente petición del ejemplo del apartado 3.4.3 del Análisis. Se configurará un *stub* tal que a una petición que sea un *POST* a */proveedor.com/rest/hoteles/HTP* y en el cuerpo contenga *"tokenDeReserva": "mcajklshflasdlñfgasfnaw=="* devuelva la respuesta del ejemplo.

```
POST http://proveedor.com/rest/hoteles/HTP


{
    "tokenDeReserva": "mcajklshflasdlñfgasfnaw==",
    "pasajeros": [
        {
            "fechaNacimiento": "09/07/1983",
            "nombre": "Pepe Viyuela",
            "email": "pepe.viyuela@email.com",
            "direccion": "C/ marques de caceres 18 2a",
            "principal": true
        },
        {
            "fechaNacimiento": "31/02/1982",
            "nombre": "Maria Viyuela",
            "principal": false
        }
```

```
    ]
}
```


Respuesta: **200 OK**


```
{
    "estado": "reservado",
    "localizador": "ABD2XYZ",
    "hotel": {
            "nombre": "Hotel tres playas",
            "codigo": "HTP",
            "descripcion": "Hotel frente al mar"
        },
    "habitacion": {
            "nombre": "Suite",
            "descripcion": "Habitación doble con vistas a la piscina y
                             salón particular",
            "tipo": "S",
            "planes": ["mediaPension", "pensionCompleta"],
            "precio": 400,
            "moneda": "EUR"
        },
    "pasajeros": [
        {
            "fechaNacimiento": "09/07/1983",
            "nombre": "Pepe Viyuela",
            "email": "pepe.viyuela@email.com",
            "direccion": "C/ marques de caceres 18 2a",
            "principal": true
        },
        {
            "fechaNacimiento": "31/02/1982",
            "nombre": "Maria Viyuela",
            "principal": false
        }
    ]
}
```


El primer paso consiste en rellenar el formulario tal y como se muestra en la figura 5.1. Se le asigna un identificador al stub, el cual es aconsejable que sea descriptivo, en cuyo caso, se indica que es una prueba mediante la palabra *TEST*, que pertenece a la suite de *Proveedor* y al producto hotelero como se ve en el segundo trozo del identificador *ProveedorHoteles*. Tambien se indica la operación, en este caso *Reserva*. Se indica la *URI* así como que se trata de un *POST*.

En la segunda parte del formulario visualizado en la figura 5.2 se indica qué característica filtrar del cuerpo de la solicitud, incluyendo el token de reserva, así como la forma que tendrá la respuesta en caso de igualarse la petición.

Home Mocks registry New mock **cgd-wiremock configuration:** http://localhost 8080

Delete

Filename

TEST_ProveedorHoteles_Reserva

Scenario

Name

Name

Current state

Current state name

Triggered state

Triggered state name

Request to match

Url pattern

http://proveedor.com/rest/hoteles/HTP

HTTP Method

POST

Request headers

Add new request headers

Query parameters

Add new query parameters

Figura 5.1: Primera parte del formulario de creación de *stubs* en *cgd-wiremock-frontal*

Request body pattern

JSON XML STRING

Body filters

Predicate | **Value**
tokenDeReserva | mcajklshflasdlñfgasfnaw== | Remove

Add new body filters

Response

HTTP Status

200

Status message

OK

Response body


```
{
  "estado": "reservado",
  "localizador": "ABD2XYZ",
  "hotel": {
      "nombre": "Hotel tres playas",
      "codigo": "HTP",
      "descripcion": "Hotel frente al mar"
   },
  "habitacion": {
      "nombre": "Suite",
```


Response headers

Add new response headers

Figura 5.2: Segunda parte del formulario de creación de *stubs* en *cgd-wiremock-frontal*

A continuación se comprueba que el *stub* se ha creado correctamente mediante la vista de listado de *stubs*, figura 5.3 y también en el detalle del mismo se revisa que esté todo correcto, figuras 5.4 y 5.5

Figura 5.3: Registro de *stubs* en *cgd-wiremock-frontal*

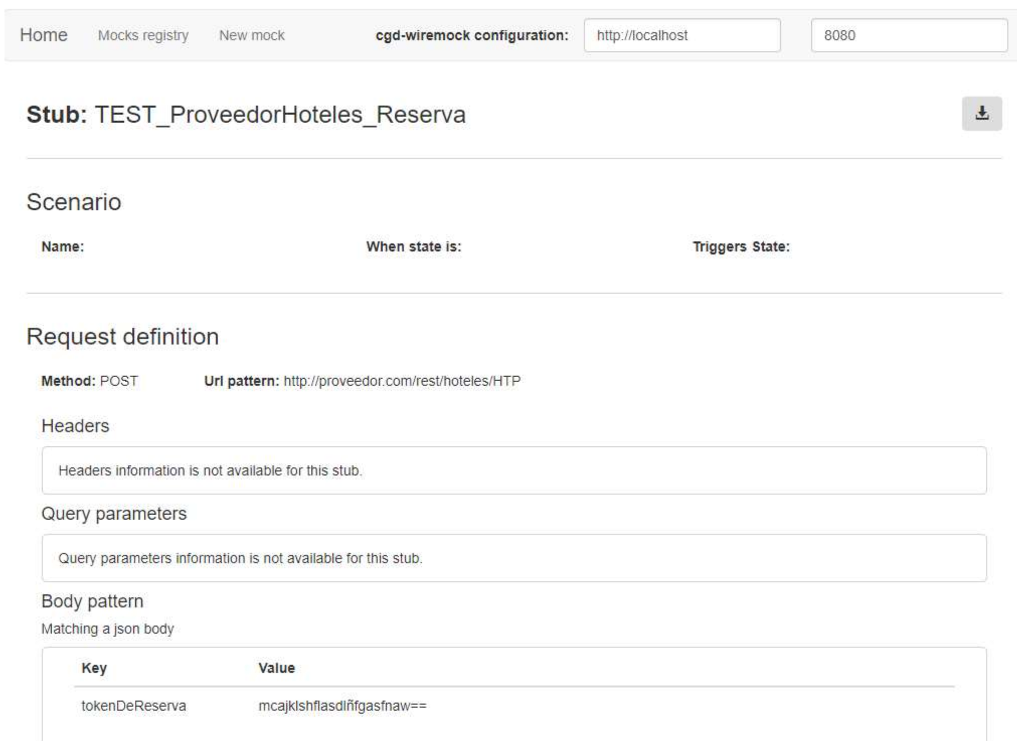

Figura 5.4: Segunda parte del detalle de un *stubs* en *cgd-wiremock-frontal*

Response definition

Status: 200 **Status message:** OK

Headers

Headers information is not available for this stub.

Response body

Notice: data might be truncated if it is to long to be printed. Click the download button to get the full information.


```
        {
    "estado": "reservado",
    "localizador": "ABD2XYZ",
    "hotel": {
            "nombre": "Hotel tres playas",
            "codigo": "HTP",
            "descripcion": "Hotel frente al mar"
        },
    "habitacion": {
            "nombre": "Suite",
            "descripcion": "Habitación doble con vistas a la piscina y
                            salón particular",
            "tipo": "S",
            "planes": ["mediaPension", "pensionCompleta"],
            "precio": 400,
            "moneda": "EUR"
        },
    "pasajeros": [
        {
            "fechaNacimiento": "09/07/1983",
            "nombre": "Pepe Viyuela",
            "email": "pepe.viyuela@email.com",
            "direccion": "C/ marques de caceres 18 2a",
            "principal": true
        },
        {
            "fechaNacimiento": ""31/02/1982"",
            "nombre": "Maria Viyuela",
            "principal": false
        }
    ]
}
```


Figura 5.5: Segunda parte del detalle de un *stubs* en *cgd-wiremock-frontal*

Por último, se prueba que funciona correctamente haciendo uso de *Postman*, un aplicativo para realizar llamadas *http*. Se cumplimenta el formulario para realizar la llamada que se ha *mockeado*, figura 5.6

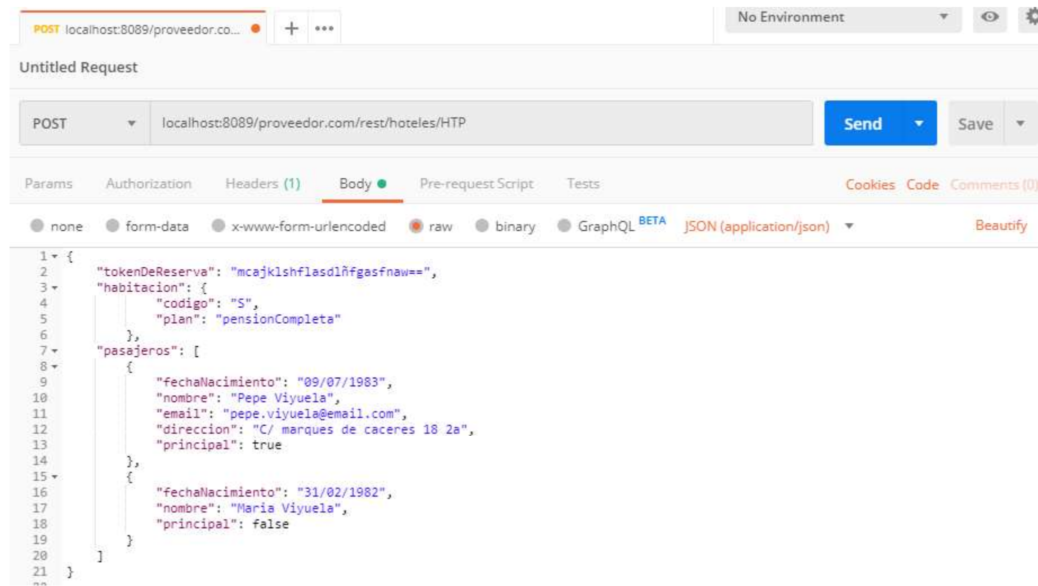

Figura 5.6: Petición de reserva de hotel en *Postman*

Y se comprueba que se recibe la respuesta configurada y que por lo tanto se ha creado correctamente el *stub* en *Wiremock* a través de *cgd-wiremock*, haciendo uso de *cgd-wiremock-frontal* para crearlo. Figura 5.7

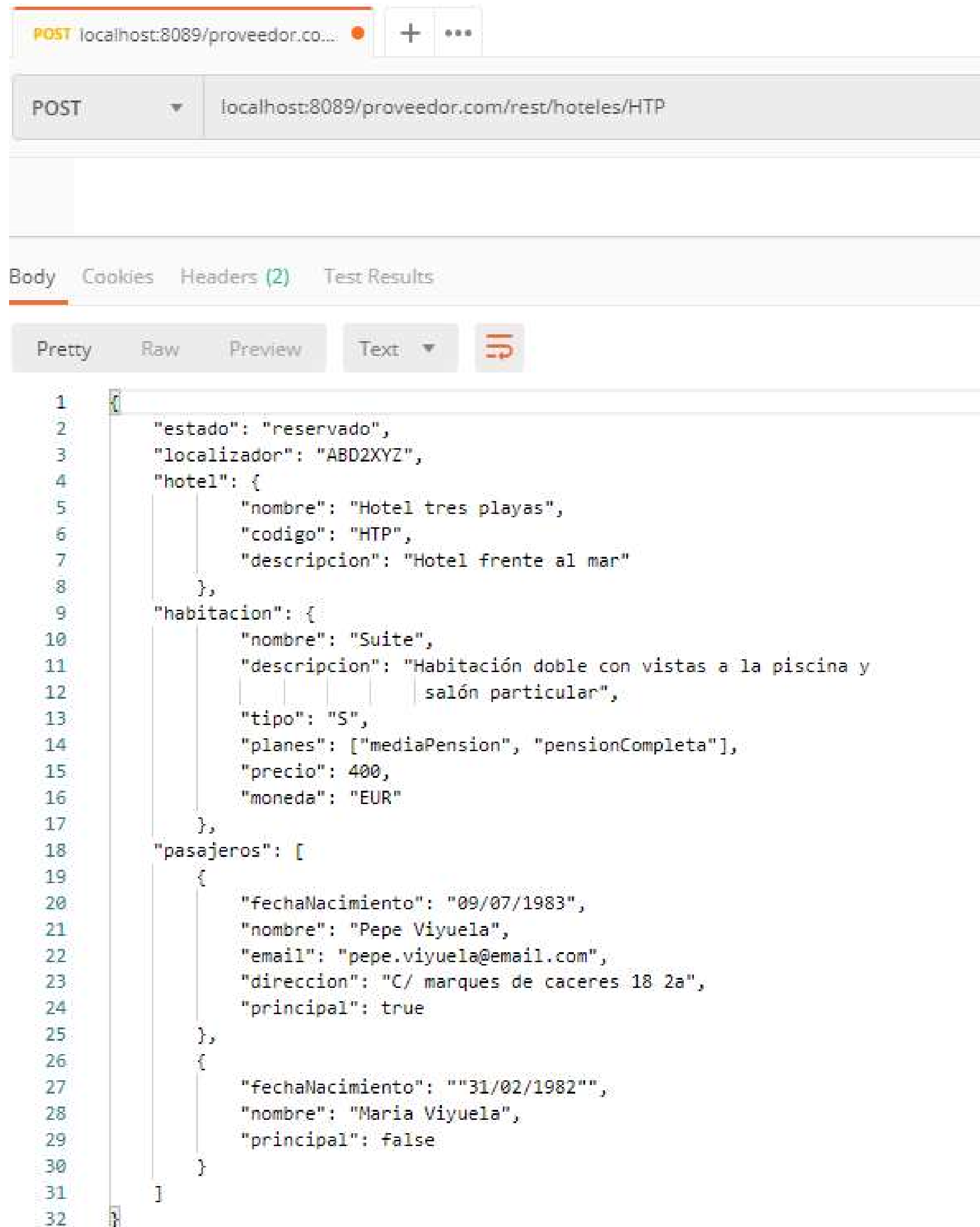

Figura 5.7: Respuesta de reserva de hotel en *Postman*

CAPÍTULO 6

Conclusiones

El objetivo del desarrollo de *cgd-wiremock* y *cgd-wiremock-frontal* era brindar la posibilidad de definir suites de *mocks* que simularan el comportamiento de tanto las *API*s de proveedores como *CGD*.

Actualmente se encuentran *mockeadas* más de 30 integraciones. Dentro del desarrollo de una integración se ha establecido como requisito el desarrollo de los *mocks* de la misma y se debe garantizar que los tests de integración, que aseguran el correcto funcionamiento de la aplicación, pasan sin errores al atacar a *Wiremock* en lugar de al proveedor.

Todos los tests de rendimiento de *CGD* se realizan sobre los *mocks* definidos con la aplicación *cdg-wiremock* y *cgd-wiremock-frontal*. De esta forma se evita lanzar miles de peticiones a los proveedores, por lo que se evitan así los bloqueos por IP o quedarse sin sesiones disponibles. Para los tests de integración también existe una configuración de *Wiremock* de forma que se pueden seguir ejecutando en caso de fallo del proveedor.

De esta manera se solucionan los dos primeros problemas planteados en los objetivos relacionados con los tests de integración y rendimiento. También se soluciona la baja disponibilidad en los entornos de tests de los proveedores puesto que una vez se tiene una suite de *mocks* de un proveedor se puede prescindir del mismo.

El uso de la aplicación se ha extendido a otros proyectos de la empresa que también están enfocados al sector turístico. Además *CGD* ofrece una instancia de la aplicación con una suite de *mocks* creada por cada producto que vende, cuyos *mocks* son editables y así los clientes pueden cambiar las respuestas en función de sus necesidades.Por último destacar que al tener una interfaz amigable y no necesitar de conocimientos técnicos, sino de negocio, los equipos de test funcional del aplicativo lo utilizan para realizar sus tests manuales. Suelen ser flujos complejos que involucran muchas partes de la aplicación. Se consiguen, pues, agilizar los desarrollos tal y como se deseaba puesto

que a pesar de no tener implementada la *API*, con *cgd-wiremock* y *cgd-wiremock-frontal* se puede *mockear* en base a las definiciones de la misma y dar servicio.

Finalmente recalcar que el proyecto me ha servido para enfrentarme por primera vez al desarrollo de un proyecto desde cero, a tener en cuenta todos los factores que entran en juego a la hora de planificar el desarrollo de un proyecto. Me ha servido para entender que es importante barajar las distintas opciones disponibles, que generalmente la opción mas sencilla es la mas resolutiva y que lidiar con las restricciones del entorno y usarlas a tu favor es necesario para la correcta implementación y uso de las herramientas que desarrollamos.

6.1 Mejoras a futuro

La aplicación se podría mejorar mediante la implantación de una base de datos en *cgd-wiremock*. La estructura seria la siguiente: los campos que forman parte de la definición de la petición y respuesta se podrían almacenar es una base de datos relacional y la respuesta como tal en una base de datos *NoSQL*, pensada para documentos grandes. Así se podrían hacer de forma eficiente filtros sobre los *stubs*, como recuperar todos los *stubs* de un proveedor o todos los que su estado de respuesta sea un 200. Además se podría paginar por servidor suavizando la carga de datos entre *back-end* y *front-end*.

Bibliografía

[1] "Amadeus it group," septiembre 2019. [Online]. Available: https://corporate.amadeus.com/ 1.1

[2] "Ruby on rails," julio 2019. [Online]. Available: https://rubyonrails.org/ 2.1

[3] "Django," julio 2019. [Online]. Available: https://www.djangoproject.com/ 2.1

[4] "Spring boot 2," septiembre 2019. [Online]. Available: https://spring.io/projects/spring-boot 2.1

[5] "Java," agosto 2019. [Online]. Available: https://www.java.com/es/ 2.1

[6] "Spring," septiembre 2019. [Online]. Available: https://spring.io/ 2.1

[7] "Wiremock," septiembre 2019. [Online]. Available: http://wiremock.org/ 2.1

[8] "Javascript," agosto 2019. [Online]. Available: https://www.javascript.com/ 2.2

[9] "Angularjs," agosto 2019. [Online]. Available: https://angularjs.org/ 2.2