



# Máster Universitario En Investigación En Ingeniería De Software Y Sistemas Informáticos

---

Generación Automática de Código

**Autor**

César Hugo Bárzano Cruz



TRABAJO DE INVESTIGACIÓN

—  
2017/2018



# Índice general



# Índice de figuras



# Capítulo 1

## Resumen

En el dominio de la WEB 2.0 el nuevo modelo de desarrollo al que tienden las empresas se basa en metodologías agile, donde se premia la rápida entrega de pequeñas funcionalidades junto con iteraciones mas seguidas con los equipos. Es común la creación de pequeños servicios, con funcionalidad muy específica para cubrir cierta necesidad de negocio y ademas de una manera dinámica y volátil. Esto puede suponer una carga de trabajo importante para los equipos de desarrollo, pues ademas de mantener los servicios funcionales han de crear nuevos constantemente, por este motivo se propone el siguiente trabajo basado en la generación dinámica de servicios web. La idea de este generador es la de reducir el tiempo de desarrollo de nuevos servicios procurando al desarrollador de un esqueleto o scaffolding funcional de un modelo de negocio muy concreto. La generación automática de servicios también puede resultar muy útil en el campo de la formación o capacitación ya que puede proporcionar el proyecto o estructura base para que el capacitador pueda mostrar los conceptos importantes en el desarrollo de servicios web a los alumnos o técnicos que reciban dicha capacitación.

El objetivo de esta investigación es el de automatizar mediante técnicas de generación automática de código la generación de servicios web base REST. El resultado de esta investigación serán un generador de servicios web en distintos lenguajes, así como un pequeño caso de uso a modo de conjunto de pruebas automáticas que permita validar el correcto funcionamiento del servicio generado.





## Capítulo 2

# Introducción

En los últimos años, la WEB 2.0 ha evolucionado de manera veloz debido a la gran demanda de información que tanto usuario como empresa han generado con el uso de las nuevas tecnologías. El desarrollo de aplicaciones web basadas en servicios ha marcado un antes y un después, haciendo que los datos tomen un nuevo valor. La transformación digital es un hecho que está cambiando el modo en que las empresas y usuarios interactúan con la información puesto que el acceso a los datos facilita la toma de decisiones, datos que hasta hace unos años no se pensaba que tendrían el valor social y económico que tienen hoy en día. Es notable como la información y los canales de comunicación crecen día a día satisfaciendo necesidades encubiertas que el usuario no conocía, automatizando procesos de negocio y mejorando tareas de gobierno.

El desarrollo de aplicaciones web normalmente va asociado a procesos de desarrollo ágil donde se premia la alta disponibilidad, la rápida creación y la integración de nuevos servicios de información con los ya disponibles. Estas metodologías son responsables en su justa medida del rápido crecimiento de lo denominado WEB 2.0 o infraestructura de información distribuida a la cual, todos podemos acceder con los innumerables dispositivos que nos rodean.

Estas metodologías, con aspectos muy beneficiosos para el ingeniería del software, tienen ciertas carencias con respecto a las metodologías de desarrollo tradicional. Si por un momento, dejamos de lado las tecnologías de la información y nos centramos en otros sectores donde el software cumple un papel vital, como por ejemplo proyectos en el ámbito espacial, defensa, automoción o aeronáutica, las líneas de desarrollo ágil son rechazadas por completo, puesto que el software resultante de estos proyectos ha sido validado y testeado durante largos años, sometidos a largos procesos de calidad y pruebas, procesos muy costosos y lentos para ser aplicados en el desarrollo de aplicaciones web o servicios cloud.

La gran demanda de información y soluciones cloud en ocasiones puede

ocasionar ciertos problemas ya que se disponibilizan servicios o versiones cuyo estado no es fiable, bajo el escudo de que los servicios de la información son de carácter volátil. Esta demanda produce una gran cantidad de software que ha de ser mantenido y actualizado, tarea complicada de manejar si se tiene en cuenta que cada solución puede ser implementada por distintos equipos, con distintos lenguajes de programación o tecnologías. Es aquí donde la generación automática de código marca la diferencia. Como se verá en los siguientes capítulos se propone la implementación de un generador de servicios cloud basados en REST cuyo objetivo sea la unificación del desarrollo de servicios. Dando un punto común y estructura base que permita a los desarrolladores avanzar rápidamente, produciendo de esta forma productos software de similar estructura, aunque desarrollen con distintas tecnologías.

Para ello, con el foco en el desarrollo ágil de aplicaciones se va a plantear una solución a como los servicios de la información han de exponerse al mundo, garantizando que su resultado final es el esperado aplicando técnicas de validación tradicional a los procesos de desarrollo ágil, siendo aquí donde la generación automática de código desempeñará un papel fundamental, ya que se intentará automatizar el proceso generativo del plan de validación de estos servicios. De esta manera, la solución final, verificará el desarrollo de aplicaciones sin incluir la dilación en el tiempo que conlleva la ejecución de un plan de validación completo sobre un proyecto software.

## Capítulo 3

# El Problema

El desarrollo basado en metodologías ágiles principalmente va enfocado a proyectos que precisan de una especial rapidez y flexibilidad en su proceso. En muchas ocasiones son proyectos relacionados con el desarrollo de software o el mundo de internet. En sectores constantemente cambiantes, las organizaciones necesitan desarrollar sus servicios rápidamente para ser altamente competitivos, y esto no es tarea fácil. Muchas veces es necesario ir probando las distintas funcionalidades del servicio sobre la marcha y medir si está funcionando o no para acabar ofreciendo una solución final.

Si se utilizan metodologías de desarrollo tradicionales, estas revisiones (o tests) pueden suponer un retraso en las fechas de entrega, aumento de costes y del volumen de trabajo. Además, también podría suceder que para cuando se tenga el producto final éste ya quede obsoleto. He aquí la importancia del desarrollo Agile.

Las metodologías ágiles se basan en un enfoque flexible. Los miembros del equipo trabajan en pequeñas fases y equipos sobre actualizaciones concretas del producto. Después, se prueba cada actualización en función de las necesidades del cliente. El producto final de un proyecto ágil puede perfectamente ser distinto al que se había previsto inicialmente. No obstante, durante los procesos de pruebas se sigue trabajando según los requerimientos del cliente, de forma que el producto final sigue respondiendo a sus necesidades. En las siguientes secciones se abordarán las metodologías ágiles más comunes y como afectan al desarrollo de servicios cloud bajo arquitecturas basadas en microservicios, y como la generación automática de código desempeña un gran papel.

### 3.1. Metodologías Ágiles

#### Scrum

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener

el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

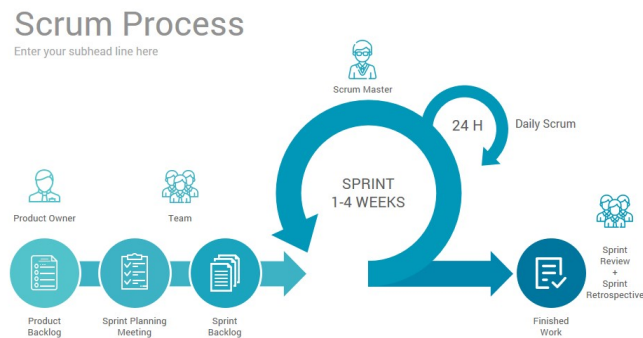


Figura 3.1: Scrum Process[?]

## Kanban

La metodología Kanban es una manera de gestionar el trabajo de forma fluida. La palabra Kanban proveniente de Japón, es un símbolo visual que se utiliza para desencadenar una acción. A menudo se representa en un tablero Kanban para reflejar los procesos de su flujo de trabajo.

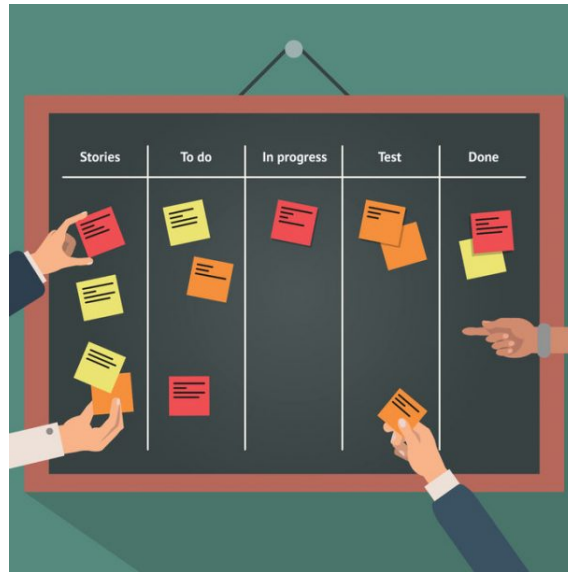


Figura 3.2: Kamban Table[?]

## Lean

Lean son una serie de principios enfocados a eliminar todas las tareas que no aporten valor sobre aquello que estamos realizando. Aplicado a un proyecto, aquello que no aporte valor al producto o servicio final. Para ello, visualiza todo el proceso que se produce en una cadena de valor a lo largo de un proyecto y, en todos aquellos puntos que se produce desperdicio, lo elimina.

### LEAN Process

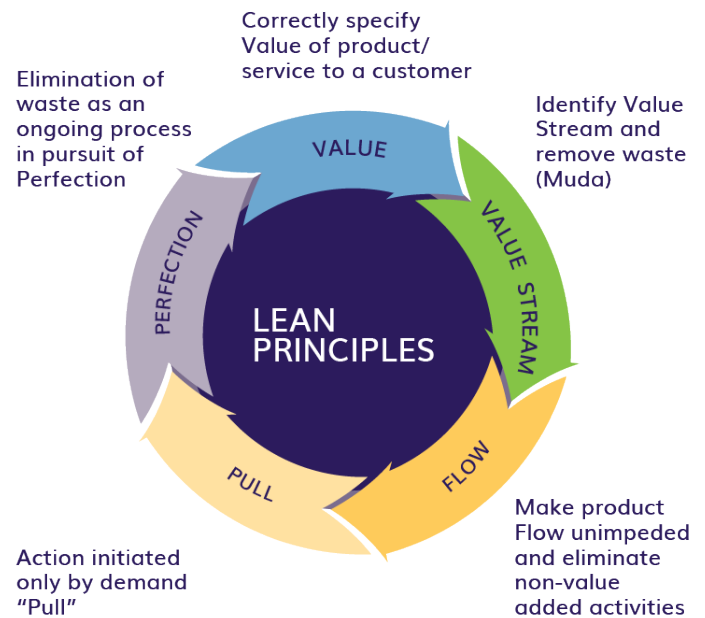


Figura 3.3: Lean Process[?]

### XP Programming

La programación extrema o eXtreme Programming (XP) es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, *Extreme Programming Explained: Embrace Change*[?]

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

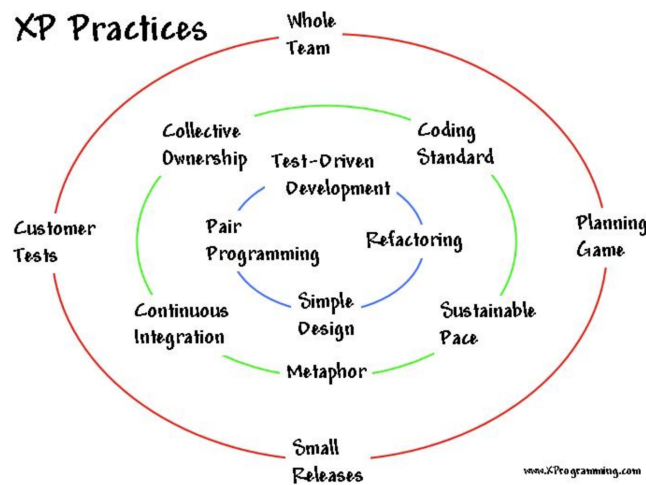


Figura 3.4: XP Practices[?]

## 3.2. Servicios web: REST

La Transferencia de Estado Representacional (REST - Representational State Transfer) fue ganando amplia adopción en toda la web como una alternativa más simple a SOAP y a los servicios web basados en el Language de Descripción de Servicios Web (Web Services Description Language - WSDL).

REST define un set de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. Los 4 principios de REST son:

1. Utiliza los métodos o verbos HTTP de manera explícita
2. No mantiene estado
3. Expone URIs
4. Transfiere XML, JavaScript Object Notation (JSON), o ambos

### 3.2.1. Arquitecturas Basadas en micro-servicios

El enfoque tradicional sobre el desarrollo de aplicaciones se centró en el monolito, donde todas las partes de la aplicación que se pueden implementar están contenidas en esa única aplicación. Esto tiene sus desventajas: cuanto más grande es la aplicación, más difícil resulta solucionar problemas nuevos

y agregar funciones nuevas con rapidez. Un enfoque basado en los microservicios para el diseño de las aplicaciones resuelve estos problemas e impulsa el desarrollo y la capacidad de respuesta.

Los microservicios se pueden considerar tanto un tipo de arquitectura como un modo de programar software. Con los microservicios, las aplicaciones se dividen en sus componentes más pequeños e independientes entre sí. A diferencia del enfoque tradicional y monolítico de las aplicaciones, en el que todo se integra en una única pieza, los microservicios son independientes y funcionan en conjunto para llevar a cabo las mismas tareas. Cada uno de estos elementos o procesos es un microservicio. Este enfoque sobre el desarrollo de software privilegia el nivel de detalle, la sencillez y la capacidad de compartir procesos similares en varias aplicaciones. Es un componente fundamental de la optimización del desarrollo de aplicaciones hacia un modelo nativo de la nube.

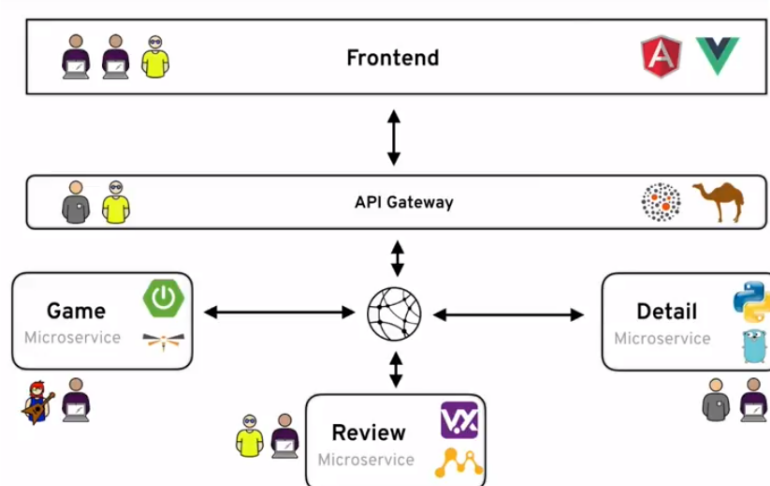


Figura 3.5: Arquitectura Basada en Microservicios[?]

### 3.3. GAC en Servicios Web

De las distintas metodologías ágiles utilizadas en el proceso de desarrollo de servicios web, se pone de manifiesto que el objetivo es cubrir rápidamente la demanda de nuevas funcionalidades o features que no estaban previstas inicialmente, evolucionando bajo demanda del cliente final e iterando con pequeñas entregas del producto. Por otro lado, se ha visto que la arquitectura orientada a microservicios está ligada a este tipo de productos e intenta descomponer en la medida de lo posible todas las funcionalidades o recursos rompiendo con las arquitecturas monolíticas tradicionales y mostrando el producto o sistema final como la suma de muchas piezas pequeñas que cumplen una funcionalidad concreta. Es aquí donde la generación automáti-



ca de código puede marcar la diferencia, ya que ante la demanda de nuevas funcionalidades es preferible la creación ( o generación) de nuevas piezas que intenten resolver el problema concreto de la manera más atómica posible.

Para ello, se propone en este trabajo la implementación de un generador de servicios web basado en REST que permita a los equipos de desarrollo generar nuevas funcionalidades asegurando que el código de los servicios generados sea todo lo genérico que se pueda, de manera que las reglas de codificación, Apis, estructura de los proyectos, separación entre capas, acceso a base de datos, pruebas y en general toda funcionalidad base esperada por un servicio web sea proporcionada por el generador de manera automática, transversal y común.



## Capítulo 4

# Desarrollo de la práctica

Visto el enunciado del problema en el capítulo anterior, se propone la implementación de un generador de servicios web basados en REST.

### 4.1. Recursos de programación utilizados

El generador ha sido desarrollado mediante el lenguaje de programación GO[?]. Go es un lenguaje de programación inspirado en C, compilado, estáticamente tipado, concurrente y con un enfoque de programación orientada a objetos. Principales características de GO:

1. Compilado: No es necesario instalar ningún programa para que el programa desarrollado funcione en el sistema operativo para el que se compiló.
2. Estáticamente Tipado: Las variables son tipadas de manera estática, así que si la variable x se define como entera, será entera durante todo su alcance.
3. Concurrente: Está inspirado en CSP - Communicating sequential processes[?]
4. Uso de paquetes: Usa paquetes para organizar el código.

El generador propuesto es capaz de generar servicios web en el lenguaje de programación GO y también en el lenguaje de programación Javascript[?] haciendo uso del framework NODE. JavaScript es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Node[?] ( node.js) es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome.

De manera adicional y con el objetivo de validar que efectivamente el código generado para el servicio es correcto, el generador también produce un caso de uso básico en el que probar las capacidades CRUD(Create,

Read, Update and Delete) del servicio. Para ello, el generador producirá un caso de uso en lenguaje de scripting BASH[?] donde usando la instrucción cURL[?] se automatizarán distintas peticiones HTTP al servicio generado.

Finalmente para la confección de esta memoria se ha utilizado LaTeX con el paquete de librerías texlive-full y el editor texmaker. Los siguientes enlaces muestran como instalar y utilizar correctamente LaTeX, han sido utilizados como referencia para el presente documento.

1. Instalar LaTeX
2. Usar LaTeX

## 4.2. Especificación de la solución

El generador puede ser encontrado en el siguiente repositorio de GitHub [hugobarzano/restandtestgenerator](https://github.com/hugobarzano/restandtestgenerator) el código relativo al generador será entregado como parte del trabajo de investigación, pero para facilitar la comprensión de las partes que forman el generador, se hará referencia a dicho repositorio de GitHub.

A grandes rasgos, en una ejecución nominal del generador, se leerán todos los ficheros que se encuentren en el directorio input/. Por cada uno de los ficheros de este directorio, el generador producirá el código funcional correspondiente a un servicio web en GO y en NODE. De manera adicional, se generará un caso de prueba donde se realizaran peticiones a la API generada mediante Shell scripting. El siguiente bloque muestra un ejemplo de configuración, correspondiente al fichero de ejemplo `restandtestgenerator/input/cats_cloud.service.json`.

Listing 4.1: Ejemplo Entrada Generador

```
1 [{  
2   "name": "Miauuu as a service",  
3   "service": "/cats",  
4   "url": "http://localhost",  
5   "port": ":8080",  
6   "body": {  
7     "name": "Wurst",  
8     "alias": "Mi bolita gatita bonita",  
9     "reina": true,  
10    "age": 1.5,  
11    "color": "Negro como mi alma"  
12  }  
13 }]
```

Los campos correspondientes al ejemplo de configuración representan:

1. "name": Nombre común del servicio.
2. "service": Api generada donde se expondrá el servicio.

3. "url": Host donde se probará el servicio.
4. "port": Puerto donde el servicio será expuesto.
5. "body": Representa el modelo de negocio o modelo de datos. Es la entidad que el servicio será capaz de manejar. Los datos de ejemplo dados serán usados en la generación de las pruebas.

que los códigos de estado devueltos por el protocolo son los que marcan el comportamiento de las peticiones atendidas por el servicio web.


| HTTP Status Codes   |                       |
|---|-----------------------|
|  | 1xx - Informational > |
|  | 2xx - Success >       |
|  | 3xx - Redirection >   |
|  | 4xx - Client Error >  |
|  | 5xx - Server Error >  |

Figura 4.1: HTTP Status Codes

Listing 4.2: Entrada Unitaria Configuración Generador

```
1 [{
2   "name": "basic web app",
3   "route": "/",
4   "http_verb": "GET",
5   "url": "http://localhost",
6   "port": ":8080",
7   "body": "",
8   "expected_code": "200"
9 }]
```

### 4.3. Servicio Generado: GO

### 4.4. Servicio Generado: NODE

### 4.5. Test Generado: Caso de uso CRUD

Listing 4.3: Entrada Unitaria Configuración Generador

```
1 [{
2   "name": "basic web app",
3   "route": "/",
4   "http_verb": "GET",
5   "url": "http://localhost",
6   "port": ":8080",
```

```

7   "body": "",
8   "expected_code": "200"
9 }]
```

Para cada uno de estas entradas, el generador producirá el siguiente TestStep:

Listing 4.4: Test Step

```

1 TestStep_0() {
2   echo "----- Test Step - 0 -----"
3   echo "TEST STEP - 0 "
4   echo "API NAME: basic web app"
5   echo "URL: http://localhost:8080/"
6
7   response_code=$(curl -XGET -i -k --write-out %{http_code} --output /
8     dev/null http://localhost:8080/)
9
10  if [ $response_code = "200" ]; then
11    echo "STEP - 0: PASS"
12    echo "----- --- -----"
13    return 0
14  else
15    echo "STEP - 0: FAIL"
16    return 1
17  fi
18 }
```

Con el conjunto de TestSteps que el generador produzca, se creará un TestCase como el que se muestra a continuación:

Listing 4.5: Test Case

```

1  #!/usr/bin/env bash
2
3  echo
4  TestStep_0() {
5    echo "----- Test Step - 0 -----"
6    echo "TEST STEP - 0 "
7    echo "API NAME: basic web app"
8    echo "URL: http://localhost:8080/"
9
10    response_code=$(curl -XGET -i -k --write-out %{http_code} --output /
11      dev/null http://localhost:8080/)
12
13    if [ $response_code = "200" ]; then
14      echo "STEP - 0: PASS"
15      echo "----- --- -----"
16      return 0
17    else
18      echo "STEP - 0: FAIL"
19      return 1
20    fi
21  }
22  echo "----- --- -----"
23  echo
24  echo
25  TestStep_1() {
```

```
26 echo "----- Test Step - 1 -----"
27 echo "TEST STEP - 1 "
28 echo "API NAME: basic web app"
29 echo "URL: localhost:8080/users"
30
31 response_code=$(curl -XPOST -i -k --write-out %{http_code} --output /
    dev/null localhost:8080/users)
32
33 if [ $response_code = "" ]; then
34     echo "STEP - 1: PASS"
35     echo "----- --- -----"
36     return 0
37 else
38     echo "STEP - 1: FAIL"
39     return 1
40 fi
41 }
42 echo "----- --- -----"
43 echo
44
45 TEST_PASS=0
46 TEST_FAIL=0
47 TOTAL_TEST=0
48 declare -a arr=("TestStep_0" "TestStep_1" )
49
50 for i in "${arr[@]}"
51 do
52     if $i; then
53         TEST_PASS=$((TEST_PASS+1));
54     else
55         TEST_FAIL=$((TEST_FAIL+1));
56     fi
57 done
58
59 echo
60 echo
61 echo "--- TEST CASE REPORT ---"
62 echo "TEST PASS: " $TEST_PASS
63 echo "TEST FAIL: " $TEST_FAIL
64 echo "TOTAL EXECUTED: " ${#arr[@]}
65 echo "----- --- -----"
```

Este TestCase será el encargado de ejecutar todos los Test Steps, dando un informe del estado de la API, o APIS que se esten probando con la configuración de entrada.

## 4.6. Pruebas

Con el Objetivo de mostrar la potencia del generador, se van a realizar una serie de ejecuciones de prueba para mostrar los posibles escenarios del generador. Para ello es necesario disponer de un servicio web sobre el que generar y ejecutar los casos de prueba generados. En este caso, disponemos de un servicio de web basado contenedores ejecutalo localmente, los fuentes del servicio web usado para testear pueden encontrarse aquí <https://github.com/hugobarzano/DataRestful>

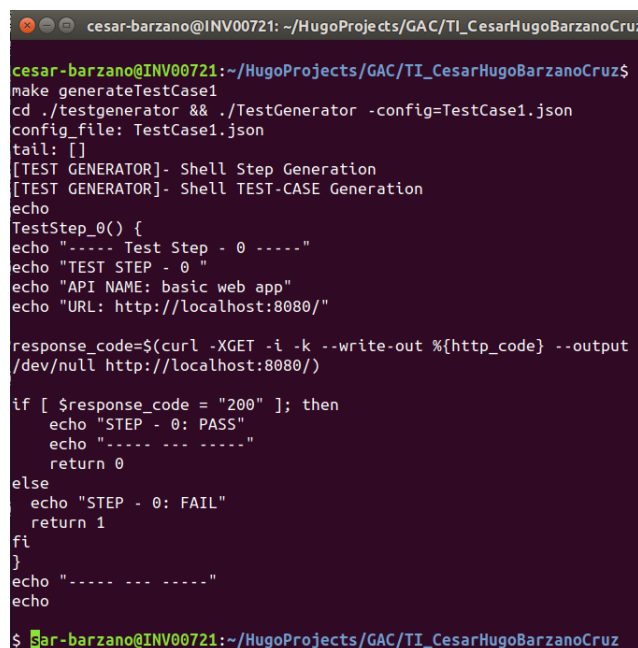
## TestCase1

El primer caso de prueba, utilizará como configuracion un solo end-point de api y puede ser generado mediante **make generateTestCase1**

Listing 4.6: TestCase1.json

```
1 [{  
2   "name": "basic web app",  
3   "route": "/",  
4   "http_verb": "GET",  
5   "url": "http://localhost",  
6   "port": ":8080",  
7   "body": "",  
8   "expected_code": "200"  
9 }  
10 ]
```

El generador producirá la siguiente salida:



```
cesar-barzano@INV00721: ~/HugoProjects/GAC/TI_CesarHugoBarzanoCruz$  
make generateTestCase1  
cd ./testgenerator && ./TestGenerator -config=TestCase1.json  
config_file: TestCase1.json  
tail: []  
[TEST GENERATOR]- Shell Step Generation  
[TEST GENERATOR]- Shell TEST-CASE Generation  
echo  
TestStep_0() {  
echo "----- Test Step - 0 -----"  
echo "TEST STEP - 0 "  
echo "API NAME: basic web app"  
echo "URL: http://localhost:8080/"  
  
response_code=$(curl -XGET -i -k --write-out %{http_code} --output  
/dev/null http://localhost:8080/)  
  
if [ $response_code = "200" ]; then  
echo "STEP - 0: PASS"  
echo "-----"   
return 0  
else  
echo "STEP - 0: FAIL"  
return 1  
fi  
}  
echo "-----"   
echo  
$
```

Figura 4.2: TestCase1

Una vez generado el primer caso de prueba y con el servicio web corriendo, podemos ejecutarlo mediante **make runTestCase1** produciendo el siguiente resultado:

El generador producirá la siguiente salida:



```

cesar-barzano@INV00721: ~/HugoProjects/GAC/TI_CesarHugoBarzanoCruz
cesar-barzano@INV00721:~/HugoProjects/GAC/TI_CesarHugoBarzanoCruz$
make runTestCase1
cd ./testgenerator/src/testgenerator/output/ && /bin/bash TestCase1
.sh

-----
----- Test Step - 0 -----
TEST STEP - 0
API NAME: basic web app
URL: http://localhost:8080/
% Total % Received % Xferd Average Speed Time Time T
ime Current
left Speed
Dload Upload Total Spent L
0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:
100 458 100 458 0 0 223k 0 --:--:-- --:--:-- --:
--:-- 223k
STEP - 0: PASS
-----

--- TEST CASE REPORT ---
TEST PASS: 1
TEST FAIL: 0
TOTAL EXECUTED: 1
-----
cesar-barzano@INV00721:~/HugoProjects/GAC/TI_CesarHugoBarzanoCruz$

```

Figura 4.3: RUN TestCase1

## TestCase2

El segundo caso de prueba, utilizará como configuracion un 2 end-point de api para disintos recursos y puede ser generado mediante **make generateTestCase2**

Listing 4.7: TestCase2.json

```

1 [{
2   "name": "Datarestful - Index",
3   "route": "/",
4   "http_verb": "GET",
5   "url": "http://localhost",
6   "port": ":8080",
7   "body": "",
8   "expected_code": "200"
9 },
10 {
11   "name": "Datarestful",
12   "route": "/Services/",
13   "http_verb": "GET",
14   "url": "http://localhost",
15   "port": ":8080",
16   "body": "",
17   "expected_code": "200"
18 }
19 ]

```

Una vez generado el segundo caso de prueba y con el servicio web corriendo, podemos ejecutarlo mediante **make runTestCase2** produciendo el siguiente resultado:

```

cesar-barzano@INV00721: ~/HugoProjects/GAC/TI_CesarHugoBarzanoCruz
API NAME: Datarestful - Index
URL: http://localhost:8080/
% Total    % Received % Xferd  Average Speed   Time    Time     T
ime Current                        Dload  Upload  Total  Spent    L
eft Speed
  0     0    0     0    0    0     0     0  --:--:--  --:--:--  --:
100  458  100  458    0    0  223k     0  --:--:--  --:--:--  --:
--- -- 223k
STEP - 0: PASS
-----
----- Test Step - 1 -----
TEST STEP - 1
API NAME: Datarestful
URL: http://localhost:8080/Services/
% Total    % Received % Xferd  Average Speed   Time    Time     T
ime Current                        Dload  Upload  Total  Spent    L
eft Speed
  0     0    0     0    0    0     0     0  --:--:--  --:--:--  --:
100  361  100  361    0    0  176k     0  --:--:--  --:--:--  --:
--- -- 176k
STEP - 1: PASS
-----

--- TEST CASE REPORT ---
TEST PASS: 2
TEST FAIL: 0
TOTAL EXECUTED: 2
-----
cesar-barzano@INV00721:~/HugoProjects/GAC/TI_CesarHugoBarzanoCruz$

```

Figura 4.4: RUN TestCase2

### TestCase3

El tercer caso de prueba, utilizará como configuracion un muchos end-point de api para disintos recursos y puede ser generado mediante **make generateTestCase3**

Listing 4.8: TestCase3.json

```

1 [{
2   "name": "Datarestful - Index",
3   "route": "/",
4   "http_verb": "GET",
5   "url": "http://localhost",
6   "port": ":8080",
7   "body": "",
8   "expected_code": "200"
9 },
10 {
11   "name": "Datarestful",
12   "route": "/Services/",
13   "http_verb": "GET",
14   "url": "http://localhost",
15   "port": ":8080",
16   "body": "",
17   "expected_code": "200"
18 },
19 {
20   "name": "Datarestful - Index",
21   "route": "/",
22   "http_verb": "GET",

```

```
23     "url": "http://localhost",
24     "port": ":8080",
25     "body": "",
26     "expected_code": "200"
27 },
28 {
29     "name": "Datarestful",
30     "route": "/Services/",
31     "http_verb": "GET",
32     "url": "http://localhost",
33     "port": ":8080",
34     "body": "",
35     "expected_code": "200"
36 },{
37     "name": "Datarestful - Index",
38     "route": "/",
39     "http_verb": "GET",
40     "url": "http://localhost",
41     "port": ":8080",
42     "body": "",
43     "expected_code": "200"
44 },
45 {
46     "name": "Datarestful",
47     "route": "/Services/",
48     "http_verb": "GET",
49     "url": "http://localhost",
50     "port": ":8080",
51     "body": "",
52     "expected_code": "200"
53 },{
54     "name": "Datarestful - Index",
55     "route": "/",
56     "http_verb": "GET",
57     "url": "http://localhost",
58     "port": ":8080",
59     "body": "",
60     "expected_code": "200"
61 },
62 {
63     "name": "Datarestful",
64     "route": "/Services/",
65     "http_verb": "GET",
66     "url": "http://localhost",
67     "port": ":8080",
68     "body": "",
69     "expected_code": "200"
70 }
71 ]
```

Una vez generado el tercer caso de prueba usando **make generateTest-Case3** y con el servicio web corriendo, podemos ejecutarlo mediante **make runTestCase3**. Al tratarse de 7 end-points de api, lo generado y ejecutado mediante make es:

Listing 4.9: TestCase3.sh

```
1 #!/usr/bin/env bash
2
3 echo
```

```

4 TestStep_0() {
5 echo "----- Test Step - 0 -----"
6 echo "TEST STEP - 0 "
7 echo "API NAME: Datarestful - Index"
8 echo "URL: http://localhost:8080/"
9
10 response_code=$(curl -XGET -i -k --write-out %{http_code} --output /
    dev/null http://localhost:8080/)
11
12 if [ $response_code = "200" ]; then
13     echo "STEP - 0: PASS"
14     echo "----- --- ----"
15     return 0
16 else
17     echo "STEP - 0: FAIL"
18     return 1
19 fi
20 }
21 echo "----- --- ----"
22 echo
23
24 echo
25 TestStep_1() {
26 echo "----- Test Step - 1 -----"
27 echo "TEST STEP - 1 "
28 echo "API NAME: Datarestful"
29 echo "URL: http://localhost:8080/Services/"
30
31 response_code=$(curl -XGET -i -k --write-out %{http_code} --output /
    dev/null http://localhost:8080/Services/)
32
33 if [ $response_code = "200" ]; then
34     echo "STEP - 1: PASS"
35     echo "----- --- ----"
36     return 0
37 else
38     echo "STEP - 1: FAIL"
39     return 1
40 fi
41 }
42 echo "----- --- ----"
43 echo
44
45 echo
46 TestStep_2() {
47 echo "----- Test Step - 2 -----"
48 echo "TEST STEP - 2 "
49 echo "API NAME: Datarestful - Index"
50 echo "URL: http://localhost:8080/"
51
52 response_code=$(curl -XGET -i -k --write-out %{http_code} --output /
    dev/null http://localhost:8080/)
53
54 if [ $response_code = "200" ]; then
55     echo "STEP - 2: PASS"
56     echo "----- --- ----"
57     return 0
58 else
59     echo "STEP - 2: FAIL"
60     return 1
61 fi
62 }

```

```
63 echo "----- --- -----"
64 echo
65
66 echo
67 TestStep_3() {
68 echo "----- Test Step - 3 -----"
69 echo "TEST STEP - 3 "
70 echo "API NAME: Datarestful"
71 echo "URL: http://localhost:8080/Services/"
72
73 response_code=$(curl -XGET -i -k --write-out %{http_code} --output /
74 dev/null http://localhost:8080/Services/)
75
76 if [ $response_code = "200" ]; then
77     echo "STEP - 3: PASS"
78     echo "----- --- -----"
79     return 0
80 else
81     echo "STEP - 3: FAIL"
82     return 1
83 fi
84 echo "----- --- -----"
85 echo
86
87 echo
88 TestStep_4() {
89 echo "----- Test Step - 4 -----"
90 echo "TEST STEP - 4 "
91 echo "API NAME: Datarestful - Index"
92 echo "URL: http://localhost:8080/"
93
94 response_code=$(curl -XGET -i -k --write-out %{http_code} --output /
95 dev/null http://localhost:8080/)
96
97 if [ $response_code = "200" ]; then
98     echo "STEP - 4: PASS"
99     echo "----- --- -----"
100     return 0
101 else
102     echo "STEP - 4: FAIL"
103     return 1
104 fi
105 echo "----- --- -----"
106 echo
107
108 echo
109 TestStep_5() {
110 echo "----- Test Step - 5 -----"
111 echo "TEST STEP - 5 "
112 echo "API NAME: Datarestful"
113 echo "URL: http://localhost:8080/Services/"
114
115 response_code=$(curl -XGET -i -k --write-out %{http_code} --output /
116 dev/null http://localhost:8080/Services/)
117
118 if [ $response_code = "200" ]; then
119     echo "STEP - 5: PASS"
120     echo "----- --- -----"
121     return 0
122 else
```

```

122     echo "STEP - 5: FAIL"
123     return 1
124 fi
125 }
126 echo "----- --- -----"
127 echo
128
129 echo
130 TestStep_6() {
131     echo "----- Test Step - 6 -----"
132     echo "TEST STEP - 6 "
133     echo "API NAME: Datarestful - Index"
134     echo "URL: http://localhost:8080/"
135
136     response_code=$(curl -XGET -i -k --write-out %{http_code} --output /
137         dev/null http://localhost:8080/)
138
139     if [ $response_code = "200" ]; then
140         echo "STEP - 6: PASS"
141         echo "----- --- -----"
142         return 0
143     else
144         echo "STEP - 6: FAIL"
145         return 1
146     fi
147 }
148 echo "----- --- -----"
149 echo
150
151 echo
152 TestStep_7() {
153     echo "----- Test Step - 7 -----"
154     echo "TEST STEP - 7 "
155     echo "API NAME: Datarestful"
156     echo "URL: http://localhost:8080/Services/"
157
158     response_code=$(curl -XGET -i -k --write-out %{http_code} --output /
159         dev/null http://localhost:8080/Services/)
160
161     if [ $response_code = "200" ]; then
162         echo "STEP - 7: PASS"
163         echo "----- --- -----"
164         return 0
165     else
166         echo "STEP - 7: FAIL"
167         return 1
168     fi
169 }
170 echo "----- --- -----"
171 echo
172
173 TEST_PASS=0
174 TEST_FAIL=0
175 TOTAL_TEST=0
176
177 declare -a arr=("TestStep_0" "TestStep_1" "TestStep_2" "TestStep_3" "
178     TestStep_4" "TestStep_5" "TestStep_6" "TestStep_7" )
179
180 for i in "${arr[@]}"
181 do
182     if $i; then
183         TEST_PASS=$((TEST_PASS+1));

```

```
181     else
182         TEST_FAIL=$((TEST_FAIL+1));
183     fi
184 done
185
186 echo
187 echo
188
189 echo "--- TEST CASE REPORT ---"
190 echo "TEST PASS: " $TEST_PASS
191 echo "TEST FAIL: " $TEST_FAIL
192 echo "TOTAL EXECUTED: " ${#arr[@]}
193 echo "--- --- --- --- --- ---"
```





## Capítulo 5

# Entrega

En este capítulo se detallan cada uno de los ficheros/directorios que forman parte de la entrega.

### 5.1. Generador

El generador propuesto por el enunciado se compone de una serie de paquetes GO alojados dentro de testgenerator/src. El generador ha sido desarrollado sobre Ubuntu.18 y hace uso de la versión Go 1.10 pero al ser un lenguaje compilado, podemos elegir entre ejecutar el código fuente (necesario tener GO instalado en el sistema o utilizar el ejecutable generado con la compilación).

### 5.2. Makefile

El fichero "makefile" establece las distintas ejecuciones de pruebas para el generador.

### 5.3. TI\_CesarHugoBarzanoCruz.pdf

Memoria de la práctica, referencia a este documento en si mismo, alojado en el directorio raíz de la entrega.

### 5.4. Directorio DOC

Directorio donde se almacenan todos los fuentes usados para generar esta documentación utilizando LaTeX. Incluye tambien las imagenes usadas en la memoria.

## 5.5. Directorio src/testgenerator/input

Directorio donde se almacenan todos los ficheros usados como configuración para el generador.

## 5.6. Directorio src/testgenerator/output

Directorio donde se almacenan todos los ficheros resultados de la ejecución de la pruebas.

# Bibliografía

- [1] GO, *The GO Programming Language* <https://golang.org/>
- [2] POSTMAN, *POSTMAN API DEVELOPMENT* <https://www.getpostman.com/>
- [3] SELENIUM, *Selenium Browser Automation* <https://www.seleniumhq.org/>
- [4] SCRUM, *Scrum Process* <https://winred.es/management/metodologia-scrum-que-es/gmx-niv116-con24594.htm>
- [5] KANBAN, *Kamban Table* <http://www.itmplatform.com/es/blog/kanban-por-que-es-agil-y-en-que-aventaja-a-scrum>
- [6] LEAN, *Lean Process* <https://centricconsulting.com/business-consulting/improve-operational-performance/business-process-improvement-lean-six-sigma>
- [7] KENT BECK , *Kent Beck - Extreme programming explained: embrace change -Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©2000 ISBN:0-201-61641-6* <https://dl.acm.org/citation.cfm?id=318762>
- [8] XP, *Basic Extreme Programming Practices* <https://ronjeffries.com/xprog/what-is-extreme-programming>
- [9] MICRO-SERVICES ARC, *Arquitectura basada en microservicios* <https://openwebinars.net/blog/diferencia-entre-arquitectura-monolitica-y-microservicios>
- [10] CHARLES ANTONY HOARE, *Communicating sequential processes* <https://www.cs.cmu.edu/~crary/819-f09/Hoare78.pdf>
- [11] JAVASCRIPT, <https://www.javascript.com>
- [12] NODE JS, *JavaScript runtime built on Chrome's V8 JavaScript engine.* <https://nodejs.org/es>

- [13] SHELL BASH, *Bash Reference Manual* <https://www.gnu.org/software/bash/manual/bash.html>
- [14] CURL, *command line tool and library for transferring data with URLs* <https://curl.haxx.se/>

## Capítulo 6

## Anexo