VNIVERSITAT DE VALÈNCIA

Máster Universitario en **Tecnologías Web, Computación en la Nube y Aplicaciones Móviles**

Front-end Angular + Back-end J2EE

Ejemplo de aplicación web de página única

J. Gutiérrez, R. Olanda, I. Panach, R. Peña-Ortiz

Curso 2022-23

Índice

1. Introducción

El presente ejemplo de aplicación web de página única [1], basada en la integración de `Angular` [2] con una `API REST` [3] desarrollada en `J2EE` [4], se enmarca en el Máster oficial en Tecnologías Web, Computación en la Nube y Aplicaciones Móviles [5] de la ETSE-UV y tiene un triple objetivo. En primer lugar, proporcionar un sencillo ejemplo de `API REST` desarrollada con un `Servlet` [6], donde la lógica de negocio ha sido desarrollada sin persistencia en base de datos. En segundo lugar, mostrar un ejemplo de consumo de esa `API REST` desde `Angular`. Por último, proporciona ejemplos de pruebas unitarias con `JUnit 5` [7] sobre el `API REST` y sobre la lógica de negocio implementada.

De esta manera, este ejemplo integra conceptos vistos en las asignaturas *"Métodos de producción de software (MPDS)"*, *"Programación del lado del cliente y visualización (PLCV)"* y *"Programación del lado del servidor (PLS)"* del mencionado máster y ofrece un ejemplo para desarrollar proyectos finales y/o tareas de las mismas.

El resto del documento se estructura como sigue. La sección 2 describe la arquitectura de componentes adoptada por la aplicación web y como poder ejecutar la misma. La sección 3 describe la interface web desarrollada en `Angular`, mientras que las secciones 5 y 4 detallan el desarrollo `Java` de la `API REST` y la lógica de negocio, respectivamente, y su testing con `JUnit 5`. Finalmente, la sección 6 incorpora un anexo sobre como se ha confeccionado y generado este documento.

2. Arquitectura de componentes

La Figura 1 muestra el diagrama de componentes asociado al ejemplo de aplicación web de página única formada por un `Front-end`, desarrollado en `Angular`, y un `Back-end`, implementado en `J2EE`. El `Front-end` interactúa con una `API REST` del `Back-end`, implementada por un `Servlet`, que hace accesible toda lógica de negocio del `Backend` bajo protocolo `HTTP`.

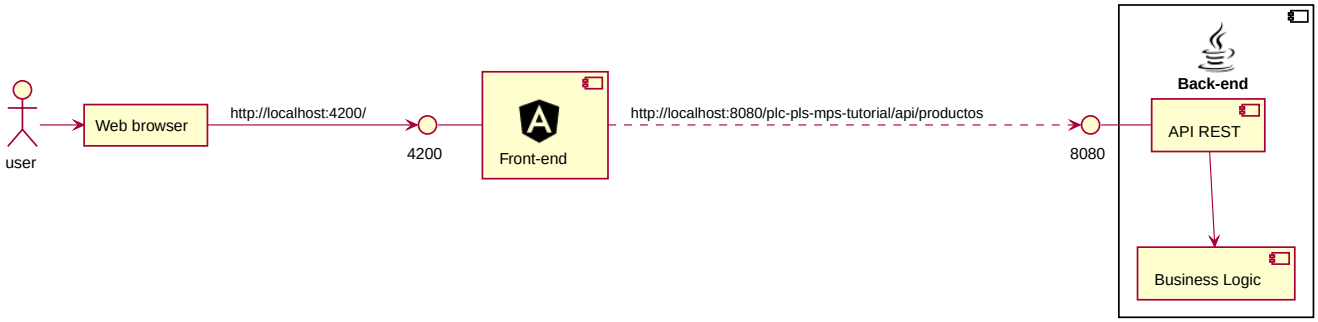

Figura 1. Diagrama de componentes

La estructura de directorios del código suministrado y su alineación con las componentes software se introduce en el Listado 1.

Listado 1. Estructura de directorios del ejemplo

```
.
├── README.adoc ①
├── README.bib ②
├── README.pdf ③
├── frontend ④
├── backend ⑤
└── style ⑥
```

donde:

① es el código fuente en `AsciiDoc` [8] de este documento.

② es el fichero de bibliografía en `BibTex` [9] de este documento.

③ es este documento.

④ es el directorio del `Front-end`.

⑤ es el directorio del `Back-end`.

⑥ es el directorio de estilos para este documento.

El `Back-end` ha sido desarrollado como `Maven Web Application` [10] de tal manera que puede ser importada en cualquier IDE y ejecutada en cualquier servidor de aplicaciones J2EE.

Previamente a arrancar el servidor de aplicaciones, habrá que indicar en la propiedad `application-server.deployments-dir` del fichero `backend/pom.xml` la ruta del directorio de `deployments` del servidor de aplicaciones.

Posteriormente, y tras arrancar el servidor de aplicaciones, se procederá a ejecutar el `Back-end` siguiendo los pasos del Listado 2.

Listado 2. Comandos para ejecutar el Back-end

```
$ cd backend/ ①
$ mvn package ②
```

donde:

① Cambia al directorio de la aplicación web del Back-end.

② Instala el Back-end en el servidor de aplicaciones indicado, previa compilación del código, ejecución de las pruebas unitarias y empaquetado de la aplicación en un Web Archive Release que copia al directorio de deployments.

El Listado 3 proporciona algunos comandos curl [11] para probar el API REST del Back-end, requiriendo que éste esté en ejecución.

Listado 3. Comandos curl para probar el API REST

```
$ curl -X GET http://localhost:8080/plc-pls-mps-tutorial/api/productos/ ①
$ curl -X GET http://localhost:8080/plc-pls-mps-tutorial/api/productos?oferta=true ②
$ curl -X GET http://localhost:8080/plc-pls-mps-tutorial/api/productos/1 ③
$ curl -X PUT  http://localhost:8080/plc-pls-mps-tutorial/api/productos/1 \ ④
       -H 'Content-Type: application/json' \
       -d '{"id":1,"nombre":"Producto 1", \
       "precio": 300,"imagen":"/assets/images/movil1.jpg","oferta":"true", \
       "comentarios":[{"estrellas":5, "comentario":"Producto funciona perfectamente, envío perfecto", "autor":"Juan
García","fecha":"2017-10-16T12:32:23.126094Z"}]}'
```

donde:

① Obtiene el listado de objetos Producto.

② Obtiene el listado de objetos Producto que están en oferta.

③ Obtiene el objeto Producto de identificador 1.

④ Modifica el objeto Producto de identificador 1 con la información suministrada.

Para ejecutar el Front-end hay que seguir los pasos del Listado 4, de manera que la aplicación Angular se podrá acceder en http://localhost:4200.

Listado 4. Comandos para ejecutar el Cliente

```
$ cd frontend/ ①
$ npm install ②
$ ng serve ③
```

donde:

① Cambia al directorio del Front-end.

② Instala la aplicación Angular.

③ Ejecuta la aplicación Angular.

3. Front-end

El código que se incorpora en la aplicación del Front-end es una simplificación del ejercicio que se desarrolla a lo largo del curso en la asignatura *PLCV*, realizando únicamente un par de modificaciones.

En primer lugar se ha añadido al fichero baseurl.ts una nueva constante, baseAPIURL, que contiene la URL a utilizar para conectar con el API-REST, tal y como muestra el Listado 5. Esto ha supuesto la modificación del servicio producto.service.ts para que haga uso de baseAPIURL a la hora de invocar el API-REST del Back-end, tal y como muestra el Listado 6.

Listado 5. Fichero baseurl.ts

```
1 export const baseURL = 'http://localhost:3000/';
2 export const baseAPIURL = 'http://localhost:8080/plc-pls-mps-tutorial/api/'; ①
```

① URL a utilizar para conectar con el API-REST.

En segundo lugar, los datos de las imágenes de la aplicación se han alojado en el directorio assets del proyecto (frontend/src/assets/images), en lugar de utilizar un servidor web para su descarga, y las diferentes páginas .html se han adaptado para que hagan uso de esas imágenes. El fichero db.json también se ha modificado para incluir la ruta correcta en el campo imagen del producto. También destacar que el fichero db.json con los datos de los productos ahora se encuentra alojado en el Back-end en la ruta backend/src/main/webapp/WEB-INF para al ejecución de la aplicación ejemplo, y en la ruta backend/src/test/resources/ para la ejecución de los test del Back-End.

Listado 6. Fichero producto.service.ts

```
import { Injectable } from '@angular/core';
import { Producto } from '../compartido/producto';
import { delay } from 'rxjs/operators';
import { HttpClient } from '@angular/common/http';
import { baseURL, baseAPIURL } from '../compartido/baseurl'; ①
import { Observable } from 'rxjs';
import { map, catchError } from 'rxjs/operators';
import { ProcesaHTTPMsjService } from './procesa-httpmsj.service';
import { HttpHeaders } from '@angular/common/http';

    const httpOptions = {
    headers: new HttpHeaders({
    'Content-Type':  'application/json',
    'Authorization': 'my-auth-token'
    })
    };

@Injectable({
  providedIn: 'root'
})
export class ProductoService {

  constructor(private http: HttpClient,
    private procesaHttpmsjService: ProcesaHTTPMsjService) { }

getProductos(): Observable<Producto[]> {
  return this.http.get<Producto[]>(baseAPIURL + 'productos')
         .pipe(catchError(this.procesaHttpmsjService.gestionError));
}

getProducto(id: number): Observable<Producto> {
  return this.http.get<Producto>(baseAPIURL + 'productos/'+ id)
  .pipe(catchError(this.procesaHttpmsjService.gestionError));
    }
getProductosOferta(): Observable<Producto[]> {
  return this.http.get<Producto[]>(baseAPIURL + 'productos?oferta=true')
  .pipe(catchError(this.procesaHttpmsjService.gestionError));
 }

getProductosIds(): Observable<number[] | any>{
return this.getProductos()
.pipe(map(productos => productos.map(producto => producto.id))) ;
}

setProducto(producto:Producto): Observable<Producto> {
  return this.http.put<Producto>(baseAPIURL + 'productos/'+ producto.id, producto, httpOptions)
  .pipe(catchError(this.procesaHttpmsjService.gestionError));
  }
}
```

① baseAPIURL se usa para invocar el API-REST que devuelve los productos.

4. Lógica de negocio

La Figura 2 muestra el diagrama de clases de la `lógica de negocio` implementada en el paquete `es.uv.etse.twcam.backend.business`. Dicho paquete contiene un servicio, implementado bajo el patrón `singleton` [12], que gestiona los productos y varios `JavaBean` [13] asociados a la información básica de los mismos. Adicionalmente se define una jerarquía de excepciones que pretenden cubrir los posibles errores de invocación al servicio cuando se busca o actualiza un producto. Nótese que para disponer de un servicio completo habría que implementar el resto de métodos de éste.

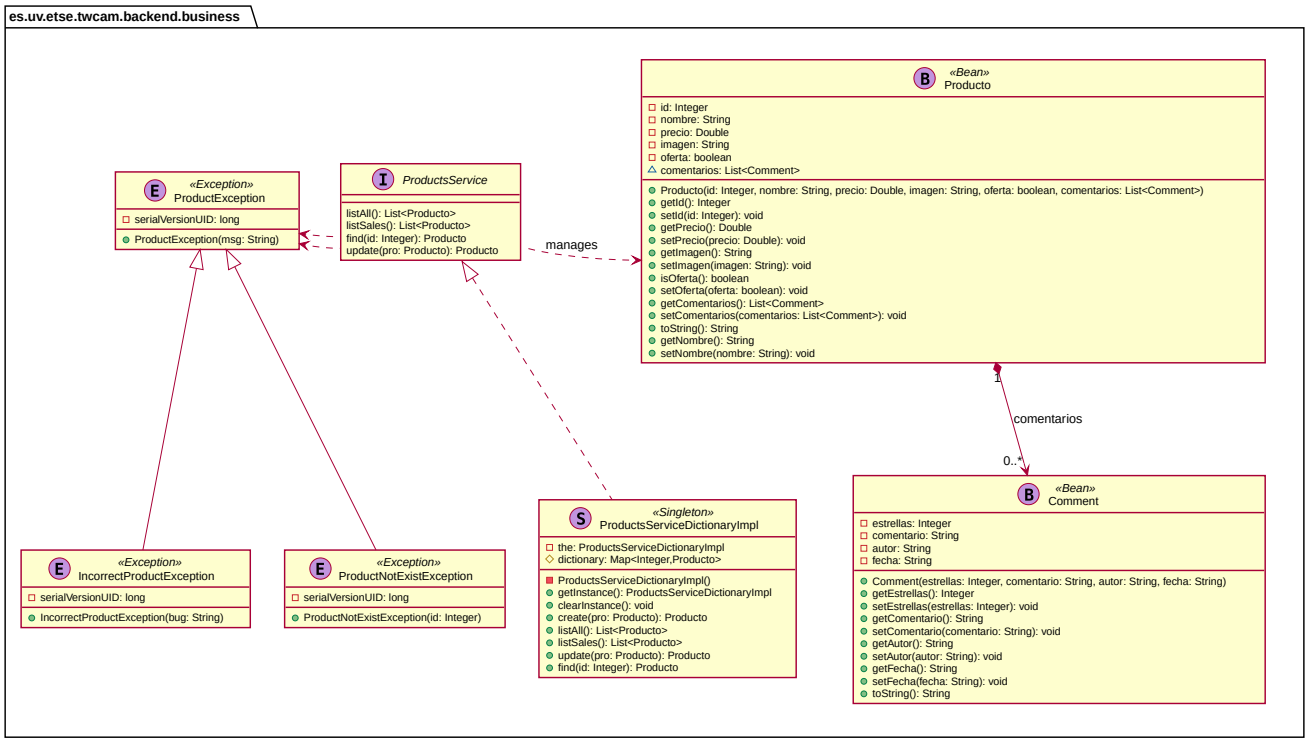

Figura 2. "Jerarquía de clases de la lógica de negocio `

Los test unitarios `JUnit 5` que validan el 100% del código de la lógica de negocio se introducen en el diagrama de clases de la Figura 3.

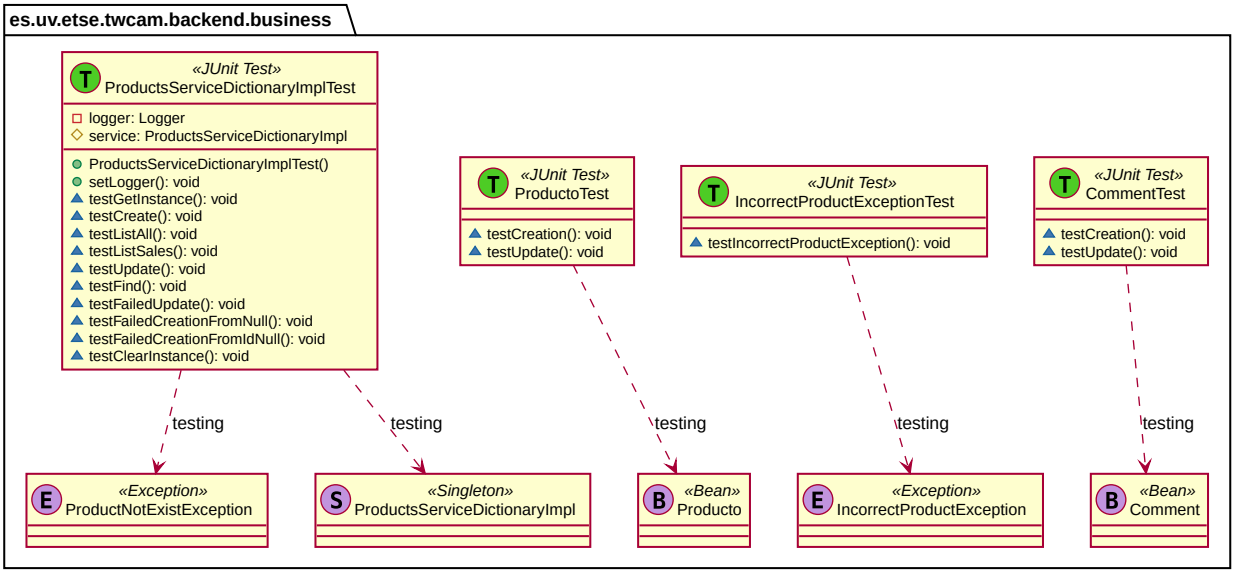

Figura 3. "Jerarquía de clases de las pruebas unitarias de la lógica de negocio `

5. API REST

La Figura 4 muestra el diagrama de clases de la aplicación web que implementa el API REST en el paquete es.uv.etse.twcam.backend.apirest. La clase InitServlet inicializará aquellos parámetros necesarios para la aplicación, mientras que ProductosEndpoint es el Servlet que implementa los comandos GET y PUT del API REST.

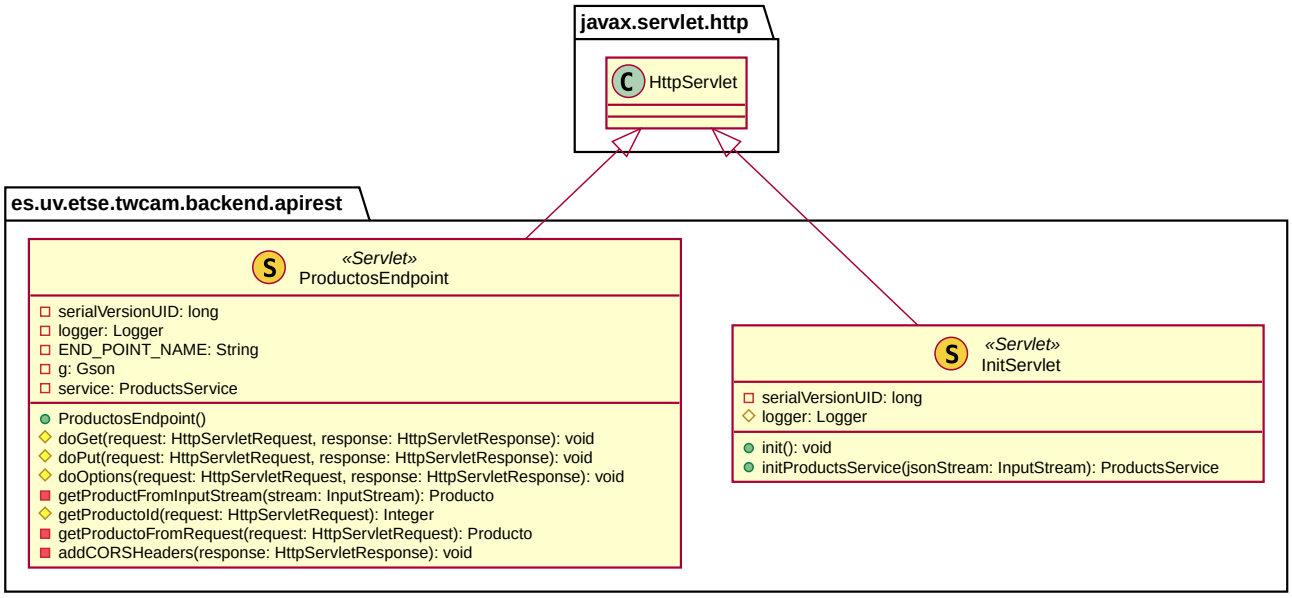

Figura 4. Jerarquía de clases del API REST

El Listado 7 muestra el código Java del Servlet encargado de inicializar la aplicación J2EE. Nótese que InitSerlvet no tiene mapeada ninguna URL y fuerza su carga al desplegar la aplicación asignando valor 1 a la propiedad load-on-startup del fichero web.xml. En este caso la única actividad de inicialización es la carga de datos asociados a los productos desde un fichero JSON.

Listado 7. Fichero InitServlet.java

```
1  package es.uv.etse.twcam.backend.apirest;
2
3  import java.io.InputStream;
4  import java.io.InputStreamReader;
5  import java.io.Reader;
6
7  import javax.servlet.ServletException;
8  import javax.servlet.http.HttpServlet;
9
10 import org.apache.logging.log4j.*;
11
12 import com.google.gson.Gson;
13 import com.google.gson.GsonBuilder;
14
15 import es.uv.etse.twcam.backend.business.Producto;
16 import es.uv.etse.twcam.backend.business.ProductException;
17 import es.uv.etse.twcam.backend.business.ProductsService;
18 import es.uv.etse.twcam.backend.business.ProductsServiceDictionaryImpl;
19
20 /**
21  * Servlet de inicializaci&oacute;n
22  *
23  * @author <a href="mailto:raul.penya@uv.es">Ra&uacute;l Pe&ntilde;a-Ortiz</a>
24  */
25 public class InitServlet extends HttpServlet {
26
```

```
27
28      /**
29       * Identificador de versi&oacute;n
30       */
31      private static final long serialVersionUID = 1L;
32
33      /**
34       * Logger
35       */
36      protected static Logger logger = LogManager.getLogger(InitServlet.class.getName());
37
38      @Override
39      public void init() throws ServletException {
40          try {
41
42              logger.info("Starting angular-j2e-example apirest ...");
43
44              String jsonFile = getServletConfig().getInitParameter("json-database"); ①
45
46              InputStream jsonStream = getServletContext().getResourceAsStream(jsonFile); ②
47
48              initProductsService(jsonStream); ③
49
50              logger.info("plc-pls-mps-tutorial apirest is started");
51
52          } catch (Exception e) {
53              logger.error("plc-pls-mps-tutorial apirest is not able to be started: ", e);
54              throw new ServletException(e);
55          }
56      }
57
58      /**
59       * Crea el servicio de productos y lo inicializa a partir de un stream JSON.
60       * @param jsonStream Stream JSON
61       * @throws Exception Indicador de errores
62       */
63      public static ProductsService initProductsService(InputStream jsonStream)
64      throws ProductException { ③
65
66          ProductsServiceDictionaryImpl service = ProductsServiceDictionaryImpl.getInstance();
67
68          Reader jsonReader = new InputStreamReader(jsonStream);
69
70          Gson gson = new GsonBuilder().create();
71
72          Producto[] productos = gson.fromJson(jsonReader, Producto[].class);
73
74          for (Producto producto : productos) {
75              service.create(producto);
76          }
77
78          logger.info("Cargados {} productos", productos.length);
79
80          return service;
81      }
82  }
```

destacando:

① Nombre del fichero `JSON` de productos definido en el fichero `web.xml` mediante el parámetro `json-database`.

② Creación de un `stream` sobre el fichero `JSON`.

③ Método encargado de la creación del servicio y adición de los productos leídos del fichero `JSON`.

El Listado 8 muestra el código Java del Servlet que implementa el endpoint de productos.

Listado 8. Fichero ProductosEndpoint.java

```java
package es.uv.etse.twcam.backend.apirest;

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

import es.uv.etse.twcam.backend.business.Producto;
import es.uv.etse.twcam.backend.business.ProductsService;
import es.uv.etse.twcam.backend.business.ProductsServiceDictionaryImpl;

import org.apache.logging.log4j.*;

/**
 * Implementaci&oacute;n b&aacute;sica del Endpoint <b>Productos</b>.
 *
 * @author <a href="mailto:raul.penya@uv.es">Ra&uacute;l Pe&ntilde;a-Ortiz</a>
 */
@WebServlet("/api/productos/*") ①
public class ProductosEndpoint extends HttpServlet {

    private static final long serialVersionUID = 1L;

    /**
     * Logger
     */
    private static final Logger logger = LogManager.getLogger(ProductosEndpoint.class.getName());  ⑦

    /**
     * Nombre del endpoint
     */
    private static final String END_POINT_NAME = "productos";

    /**
     * Gson parser
     */
    private final Gson g = new GsonBuilder().create();

    /**
     * Servicio sobre productos.
     */
    private static ProductsService service = ProductsServiceDictionaryImpl.getInstance();

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ProductosEndpoint() {
        super();
        logger.info("Product EndPoint creado"); ⑦
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException {

        String result = null;
        Integer id = null;

```

```
66          try {
67              id = getProductoId(request);
68          } catch (Exception e) {
69              logger.info("No se ha podido obtener el identificador del request"); ⑦
70          }
71
72          logger.info("GET at {} with ID: {}", request.getContextPath(),id); ⑦
73
74          if (id == null) {
75              List<Producto> productos = null;
76              String oferta = request.getParameter("oferta");
77
78              if (oferta!= null && oferta.equals("true")) {
79                  logger.info("GET sales"); ⑦
80                  productos = service.listSales();
81              } else
82                  productos = service.listAll();
83
84              result = g.toJson(productos);
85          } else {
86              try {
87                  Producto pro = service.find(id);
88                  result = g.toJson(pro);
89              } catch (Exception e) {
90                  logger.error("Producto no encontrado"); ⑦
91              }
92          }
93
94          addCORSHeaders(response); ②
95
96          if (result != null) {
97              response.setStatus(HttpServletResponse.SC_ACCEPTED);③
98          } else {
99              response.setStatus(HttpServletResponse.SC_NOT_FOUND);③
100             result="{}";
101         }
102
103         try {
104             PrintWriter pw = response.getWriter();
105             pw.println(result);
106             pw.flush();
107             pw.close();
108         } catch (IOException ex) {
109             logger.error("Imposible enviar respuesta",ex); ⑦
110         }
111     }
112
113     @Override
114     protected void doPut(HttpServletRequest request, HttpServletResponse response)
115     throws ServletException, IOException {
116
117         Producto pro = null;
118
119         try {
120
121             pro = getProductoFromRequest(request);
122
123             if (pro == null) {
124                 response.setStatus(HttpServletResponse.SC_NOT_FOUND); ③
125                 addCORSHeaders(response); ②
126                 logger.error("Producto no actualizado por no se puede extraer desde JSON"); ⑦
127             } else {
128                 pro = service.update(pro);
129
130                 logger.info("PUT at: {} with {} ", request.getContextPath(), pro); ⑦
131
132                 response.setStatus(HttpServletResponse.SC_ACCEPTED); ③
133                 addCORSHeaders(response); ②
134
135                 PrintWriter pw = response.getWriter();
```

```
136                 pw.println(g.toJson(pro));
137                 pw.flush();
138                 pw.close();
139             }
140 
141         } catch (Exception e) {
142             response.setStatus(HttpServletResponse.SC_NOT_FOUND); ③
143             logger.error("Producto no actualizado", e); ⑦
144         }
145     }
146 
147     @Override
148     protected void doOptions(HttpServletRequest request, HttpServletResponse response) {
149 
150         addCORSHeaders(response); ②
151 
152         try {
153             super.doOptions(request, response);
154         } catch (ServletException se) {
155             logger.error("Error genérico en la clase padre"); ⑦
156         } catch (IOException ioe) {
157             logger.error("Error genérico de salida la clase padre"); ⑦
158         }
159     }
160 
161     /**
162      * Obtiene el Product de un stream JSON
163      * @param stream Stream JSON
164      * @return Product
165      */
166     private Producto getProductFromInputStream(InputStream stream) {
167 
168         Producto pro = null;
169 
170         try {
171 
172             pro = g.fromJson(new InputStreamReader(stream), Producto.class); ④
173 
174         } catch (Exception e) {
175             pro = null;
176             logger.error("Error al obtener producto desde JSON",e); ⑦
177         }
178 
179         return pro;
180     }
181 
182     /**
183      * Obtiene el identificador de un Producto como parte de la URL de la petición HTTP.
184      * @param request Petición HTTP
185      * @return Identificador del Producto
186      */
187     protected static Integer getProductoId(HttpServletRequest request) throws APIRESTException{  ⑤
188 
189         String url = request.getRequestURL().toString();
190 
191         int posIni = url.lastIndexOf("/");
192 
193         int posEnd = url.lastIndexOf("?");
194 
195         if (posEnd < 0) {
196             posEnd = url.length();
197         }
198 
199         String id = url.substring(posIni+1,posEnd);
200 
201         logger.debug("ID: {}", id);⑦
202 
203         if (id.trim().isEmpty()) {
204             id = null;
205         }
```

```
206
207         if (id == null) {
208             throw new APIRESTException("Faltan parámetros en el EndPoint");
209         } else {
210             if (id.equals(END_POINT_NAME)) {
211                 id = null;
212             }
213         }
214
215         Integer valor = null;
216
217         if (id != null) {
218             valor = Integer.valueOf(id);
219         }
220
221         return valor;
222     }
223
224     /**
225      * Obtiene el Product desde la petición HTTP y el identificador como parte de la URL.
226      * @param request Petición HTTP
227      * @return Product
228      */
229     private Producto getProductoFromRequest(HttpServletRequest request) {
230
231         Producto pro = null;
232
233         try {
234
235             Integer id = getProductoId(request);
236
237             if (id != null) {
238                 pro = getProductFromInputStream(request.getInputStream());
239                 if (pro!=null && !pro.getId().equals(id)) ⑥
240                     pro = null;
241             }
242
243         } catch (Exception e) {
244             pro = null;
245         }
246
247         return pro;
248     }
249
250     /**
251      * Añade cabeceras Cross-origin resource sharing (CORS) para poder invocar el API
252      * REST desde Angular
253      *
254      * @param response Repuesta HTTP a la que añadir cabeceras
255      */
256     private void addCORSHeaders(HttpServletResponse response) { ②
257         response.addHeader("Content-Type", "application/json");
258         response.addHeader("Access-Control-Allow-Credentials", "true");
259         response.addHeader("Access-Control-Allow-Methods", "GET, OPTIONS, HEAD, PUT, POST");
260         response.addHeader("Access-Control-Allow-Headers", "authorization,content-type");
261         response.addHeader("Access-Control-Allow-Origin", "*");
262     }
263 }
```

destacando:

① El mapeo del `Servlet` del `API REST` se hace a `/api/productos/*` para permitir que se pase el identificador de un producto como parte de la URL (e.g., `/api/productos/1`).

② En todo comando añadimos las cabeceras de seguridad que nos permiten invocar el `API REST` desde un navegador con `Javascript`.

③ Debemos hacer un buen uso de los código `HTTP` en cada comando.

④ La obtención del `JSON` en los comandos `POST` y `PUT` se hace con `GSON` [14].

⑤ El identificador del `Producto` en los comandos `GET` y `PUT` se puede obtener dividiendo la URL de la petición.

⑥ En el caso del comando `PUT` el identificador del `Producto` está en la URL de la petición, pero el resto de parámetros forma parte de un `JSON`.

⑦ Los mensajes de traza y error son escritos por la salida estándar y de error del servidor de aplicaciones.

La Figura 5 muestra el diagrama de clases de las pruebas unitarias realizadas sobre el `API REST`. El test `InitServletTest` valida más del 70% de la clase `InitServlet`, mientras que `ProductosEndpointTest` hace lo propio con más de un 90% del código de `ProductosEndpoint`. Nótese que en este último test se ha utilizado el framework `mockito` [15] para desacoplar las partes del endpoint que dependen del servidor de aplicaciones con el fin de poder ejecutar sus pruebas unitarias sin necesidad de desplegar el `API-REST` en el servidor.

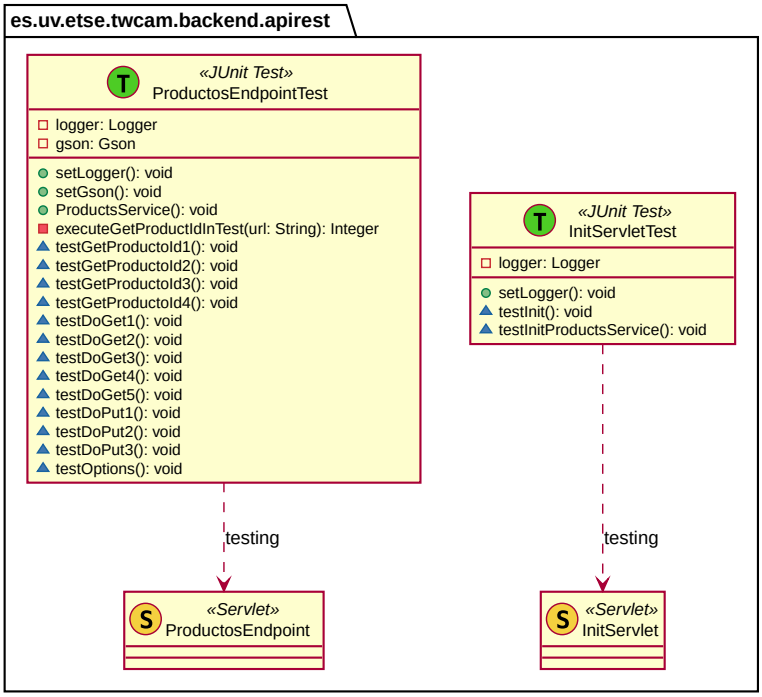

Figura 5. Jerarquía de clases de las pruebas unitarias sobre el `API REST`

6. Anexo generación documento

El presente documento técnico ha sido confeccionado en AsciiDoc [8] haciendo uso de las extensiones para BibTex [9] y PlantUML [16]. El Listado 9 muestra los comandos que permiten generar el PDF del fichero README.adoc usando Asciidoctor Docker Container [17].

Listado 9. Comandos para generar el PDF del fichero README.adoc

```
$ pwd
.../pls-plc-mps-integration-tutorial
$ ls
README.adoc    README.bib     README.pdf     backend/   frontend/   style/

$ docker run -it -v `pwd`:/documents/ asciidoctor/docker-asciidoctor

bash-5.1# asciidoctor-pdf -a pagenums -a source-highlighter=rouge \ ①
                          -a pdf-style=style/uveg-theme.yml \
                          -r asciidoctor-diagram \
                          -r asciidoctor-bibtex  \
                          --trace \
                          README.adoc

bash-5.1# asciidoctor -b xhtml5 -r asciidoctor-diagram -a toc=left \ ②
                      -a source-highlighter=rouge \
                      -r asciidoctor-bibtex --trace \
                      README.adoc
```

donde:

① Genera la versión PDF de README.adoc

② Genera la versión HTML de README.adoc

7. Referencias

1. Gil Fink y Ido Flatow. 2014. Introducing Single Page Applications. En *Pro Single Page Application Development: Using Backbone.js and ASP.NET*. Apress, Berkeley, CA, 3-13. https://doi.org/10.1007/978-1-4302-6674-7_1
2. Google. 2010-2022. ANGULAR: The modern web developer's platform. Recuperado a partir de https://angular.io
3. Roy Thomas Fielding y Richard N. Taylor. 2002. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology* 2, 2: 115-150. https://doi.org/10.1145/514183.514185
4. Oracle. 2022. Java Platform, Enterprise Edition. Recuperado a partir de https://www.oracle.com/es/java/technologies/java-ee-glance.html
5. Universitat de València. 2022. Máster oficial en Tecnologías Web, Computación en la Nube y Aplicaciones Móviles. Recuperado a partir de https://www.uv.es/twcam
6. Oracle. 2019. Java Servlet Technology. Recuperado a partir de https://www.oracle.com/technetwork/java/index-jsp-135475.html
7. Shekhar Gulati y Rahul Sharma. 2017. *Java Unit Testing with JUnit 5*. Apress. https://doi.org/10.1007/978-1-4842-3015-2
8. Eclipse Foundation, Inc. 2022. AsciiDoc: Text based document generation. Recuperado a partir de https://asciidoc.org
9. Overleaf. 2022. Bibliography management with bibtex. Recuperado a partir de https://es.overleaf.com/learn/latex/Bibliography_management_with_bibtex
10. Apache. 2022. Maven. Recuperado a partir de https://maven.apache.org
11. Daniel Stenberg. 1998-2022. command line tool and library for transferring data with URLs. Recuperado a partir de https://curl.se
12. Erich Gamma, Richard Helm, Ralph Johnson, y John Vlissides. 1994. Creational Patterns. En *Design Patterns: Elements of Reusable Object-Oriented Software*, Bryan W. Kernighan (ed.). Addison Wesley, 81-136. Recuperado a partir de https://github.com/TushaarGVS/Design-Patterns-Mentorship
13. Oracle. 2022. JavaBeans Technology. Recuperado a partir de https://www.oracle.com/java/technologies/javase/javabeans-doc.html
14. Google Inc. 2022. Gson. Recuperado a partir de https://github.com/google/gson
15. Szczepan Faber. 2022. Mockito: Tasty mocking framework for unit tests in Java. Recuperado a partir de https://site.mockito.org
16. PlantUML Group. 2021. *Guía de Referencia del Lenguaje PlantUML*. Recuperado a partir de https://plantuml.com/es/guide
17. Asciidoctor Project. 2022. Asciidoctor Docker Container. Recuperado a partir de https://github.com/asciidoctor/docker-asciidoctor