

Universidad Javeriana Cali
Construcción y Pruebas de Software
Proyecto Final del Curso

Presentado por:

William Martín Chávez González

Hernan Danilo Eraso Rosero

Santiago Rojas

Objetivo: Aplicar todos los conocimientos adquiridos durante el curso.

Servicio Backend API: <https://github.com/willy23martin/test-automation-java-service>

Servicio Frontend: <https://github.com/Tiago9617/-consecionario-mat-pruebas/tree/main/consecionario-mat-pruebas>

Descripción:

1. Escoger un producto de software del cual se tenga acceso al código fuente, el cual será el objeto de prueba del proyecto.
 - a. Proyecto personal con una funcionalidad básica, por ejemplo, un CRUD.
2. Realizar las actividades de acuerdo con el proceso de pruebas visto en clase.
3. Documentar los principales entregables del proceso.
4. Preparar una presentación de máximo 15 minutos para compartir con los compañeros de clase.
5. El proyecto debe contener:
 - a. Planeación de pruebas.
 - b. Diseño de casos de prueba (caja blanca y caja negra).
 - c. Resultados de la ejecución de los casos de prueba.
 - d. Diseño de pruebas de desempeño.
 - e. Ejecución y reporte de resultados de la ejecución de las pruebas de desempeño.
 - f. Automatización de pruebas.
6. Los entregables se deben publicar en un repositorio personal y compartirlo con la profesora.

Detalles del entregable (Semana 11):

1. Plan de pruebas:

a. Alcance funcional de las pruebas: descripción de alto nivel de las funcionalidades que serán probadas.

b. Estrategia de pruebas: niveles y tipos de pruebas que se contemplarán.

2. Casos de prueba de caja blanca:

a. Aplicar las técnicas vistas en clase de tal manera que se garanticen las métricas de cobertura de sentencia o decisión.

b. Reporte de resultados de la ejecución de las pruebas.

3. Casos de prueba de caja negra:

a. Diseñar los casos de prueba contemplando las técnicas vistas en clase:

i. Clases de equivalencia.

ii. Valores al límite.

iii. Tablas de decisión.

b. Documentación de los casos de prueba.

c. Reporte de los resultados de la ejecución de los casos de prueba.

4. Detalle del diseño de las pruebas de desempeño.

5. Resultados de la ejecución de las pruebas de desempeño.

6. Automatización de pruebas (es negociable el nivel de pruebas a automatizar).

7. Reporte de resultados de la ejecución de las pruebas automatizadas

1. Plan de Pruebas:

para el ejercicio se tomará un producto de software seleccionado es un catálogo de autos que contiene información sobre diferentes modelos de automóviles, incluyendo imágenes, nombres, precios, descripciones, marcas y campos para establecer rangos de precios o modelo específico acompañado cada imagen con un botón para realizar la compra.

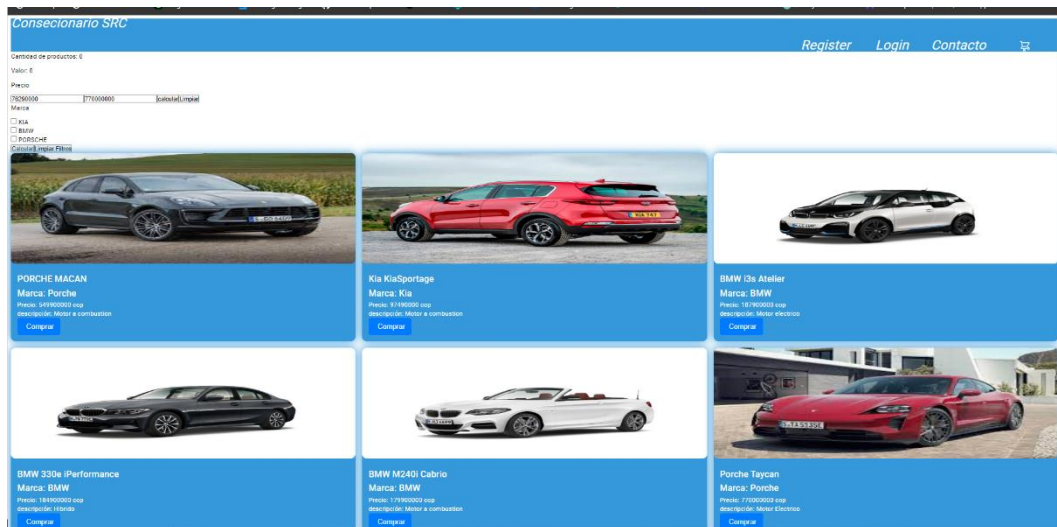
Funcionalidad Básica: La funcionalidad básica de este catálogo de autos incluye la capacidad de buscar autos por marca y precio, obtener los autos mediante un rango de precios y un filtro de marca. Además, el botón donde al escoger un vehículo se procede el paso para comprar

a. Alcance Funcional de las Pruebas

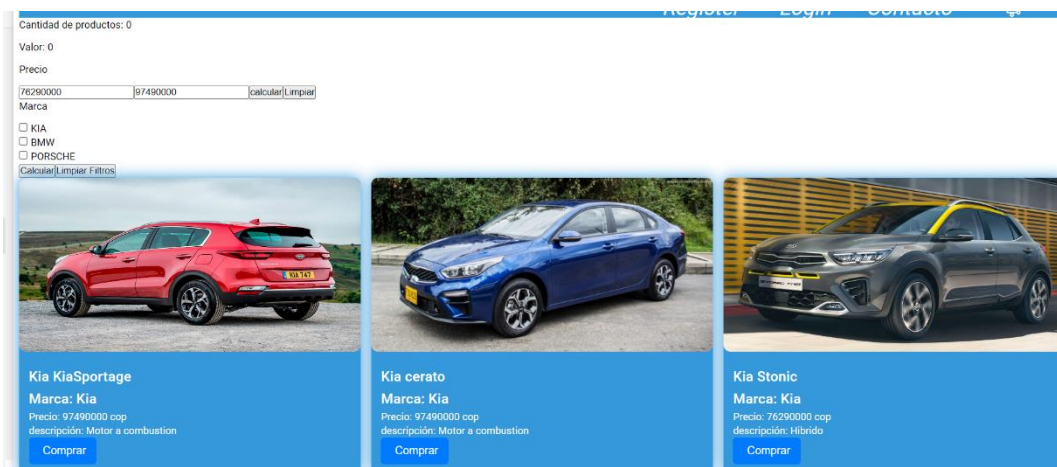
Para el frontend en Angular:

- Asegurarse de que los datos de los autos se muestran correctamente en la interfaz de usuario, búsqueda y filtro de autos por marca y precio.

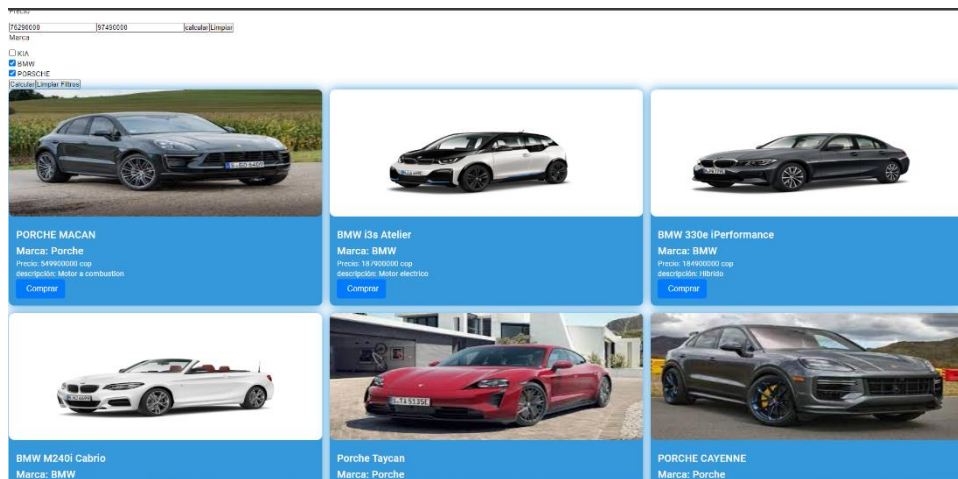
1. Render de todos los productos



2. Filtro por precio retorna la información de Kia



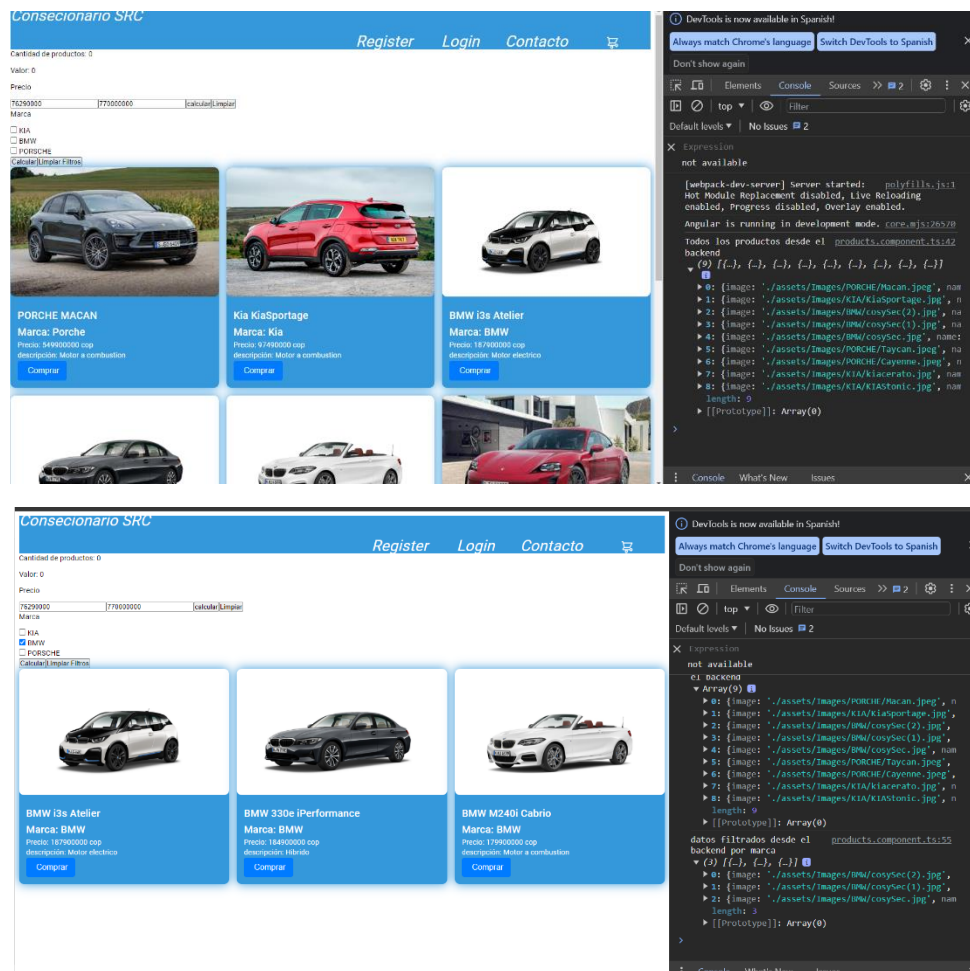
3. Filtro por marca BMW y PORCHE

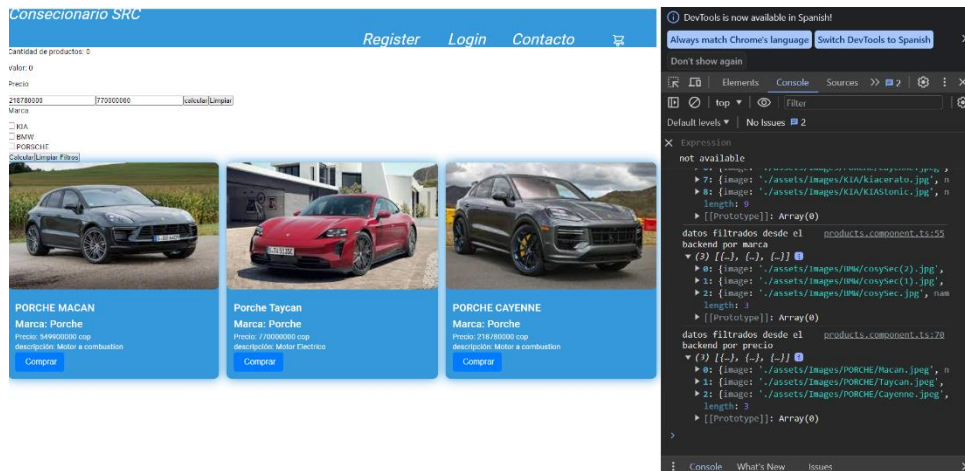


- Comprobar que los filtros de búsqueda funcionen correctamente.
- Validar que la funcionalidad de ordenar autos por precio está funcionando adecuadamente.

Para el backend en Java, comprueba que los servicios de API REST funcionen correctamente al recibir solicitudes del frontend.

- Comunicación del backend por consola de los productos, <http://localhost:8080/api/cars>





b. Estrategia de Pruebas

La estrategia de pruebas podría incluir varios niveles y tipos de pruebas:

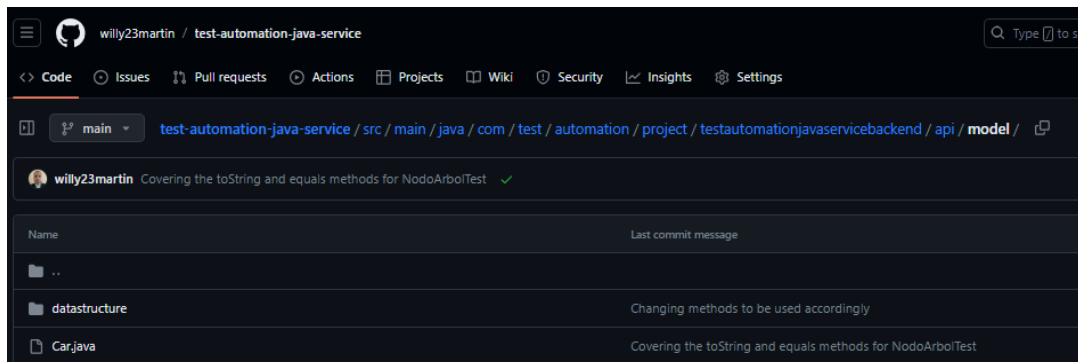
- **Pruebas de Unidad para el Backend - Java:**

De acuerdo con la organización de las capas del proyecto en el repositorio:

<https://github.com/willy23martin/test-automation-java-service>

las pruebas de Unidad se deben realizar para cada componente crítico:

- **Estructura de datos: ArbolBinario**, junto con el **NodoArbol** y el elemento principal **Car** (que representa los carros para guardar en la estructura). Packages: **model** y **model.datastructure**



test-automation-java-service-backend

test-automation-java-service-backend

Element	Missed Instructions	Cov. %	Missed Branches	Cov. %	Missed Cxty	Missed Lines	Missed Methods	Missed Classes	
com.test.automation.project.testautomationjavaservicebackend.api.database	<div><div></div></div>	55 %	<div><div></div></div>	0 %	11 13	1 3	4 6	0	
com.test.automation.project.testautomationjavaservicebackend.api.model	<div><div></div></div>	77 %	<div><div></div></div>	100 %	5 15	1 11	5 14	0	
com.test.automation.project.testautomationjavaservicebackend	<div><div></div></div>	37 %	<div><div></div></div>	n/a	1 2	2 3	1 2	0	
com.test.automation.project.testautomationjavaservicebackend.api.model.datastructure	<div><div></div></div>	100 %	<div><div></div></div>	100 %	0 34	0 56	0 20	0	
com.test.automation.project.testautomationjavaservicebackend.api.services	<div><div></div></div>	100 %	<div><div></div></div>	100 %	0 13	0 22	0 6	0	
com.test.automation.project.testautomationjavaservicebackend.api.config	<div><div></div></div>	100 %	<div><div></div></div>	n/a	0 4	0 7	0 4	0	
com.test.automation.project.testautomationjavaservicebackend.api.controllers	<div><div></div></div>	100 %	<div><div></div></div>	n/a	0 4	0 5	0 4	0	
Total		90 of 621	85 %	14 of 58	75 %	17 85	4 107	10 56	0

Created with JaCoCo 0.8.11.202310141055

Created with JaCoCo 0.8.11.202310140853

- **Pruebas de Integración Karate - Backend:**

Como el sistema se compone de una parte Backend y otra Frontend, con el primero siendo un servicio API REST, se deben realizar pruebas de Integración automatizadas que verifiquen el comportamiento de los diferentes endpoints:

- ✓ **GET /api/cars** - obtener el listado de carros de inventario del Concesionario.
- ✓ **GET api/search?initialPrice=<>&finalPrice=<>** - filtrar los carros disponibles por rango de precios.
- ✓ **GET /api/searchBrands?brands=<>, <>, <>, ...** - filtrar los carros disponibles por marca o por varias marcas.

A continuación, se listan cada uno de los casos de prueba por cada **endpoint del Backend API**:

Endpoint	TestID_Descripción
GET /api/cars	EPCARS001_elInventarioContieneUnListadoDeCarrosPrecargados : al realizar la petición, el sistema deberá retornar el listado de 9 carros precargados de marcas BMW, Kia y Porsche.
GET api/search?initialPrice=<>&finalPrice=<>	EPSEARCHP001_searchCarsByPriceRange_DebeRetornarUnaListaConTres Carros_CuandoElPrecioDelCarroNoSuperaLos100Millones – que son aquellos que corresponden a la marca Kia.
	EPSEARCHP002_searchCarsByPriceRange_DebeRetornarUnaListaConTres Carros_CuandoElPrecioDelCarroEstribaEntre100MillonesY200Millones : aquellos que corresponden a la marca BMW.
	EPSEARCHP003_searchCarsByPriceRange_DebeRetornarUnaListaConTres Carros_CuandoElPrecioDelCarroSupereLos200Millones : que son aquellos que corresponden a la marca Porsche.
	EPSEARCHP004_searchCarsByPriceRange_DebeRetornarUnaListaCon6Carros_CuandoElPrecioDelCarroSupereLos100Millones : que son aquellos que corresponden con las marcas BMW y Porsche.
	EPSEARCHP005_searchCarsByPriceRange_DebeRetornarUnaListaVacía_CuandoElPrecioDelCarroSeaInferiorA50Millones
	EPSEARCHP006_searchCarsByPriceRange_DebeRetornarUnaListaCon9Carros_CuandoElPrecioDelCarroSupereLos50Millones : que corresponde a todos los carros del inventario.
	EPSEARCHP007_searchCarsByPriceRange_DebeLanzarUnIllegalArgument Exception_CuandoElPrecioInicialDelCarroSupereAlPrecioFinalParaFiltrar
GET /api/searchBrands?brands=<>, <>, <>, ...	EPSEARCHB001_searchCarsByBrands_DebeDevolver3Carros_CuandoSeFiltrePorLaMarcaBMW
	EPSEARCHB002_searchCarsByBrands_DebeDevolver3Carros_CuandoSeFiltrePorLaMarcaKia
	EPSEARCHB003_searchCarsByBrands_DebeDevolver3Carros_CuandoSeFiltrePorLaMarcaPorsche
	EPSEARCHB004_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Kia
	EPSEARCHB005_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Porsche
	EPSEARCHB006_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaKia_Y_Porsche
	EPSEARCHB007_searchCarsByBrands_DebeDevolver9Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Kia_Y_Porsche

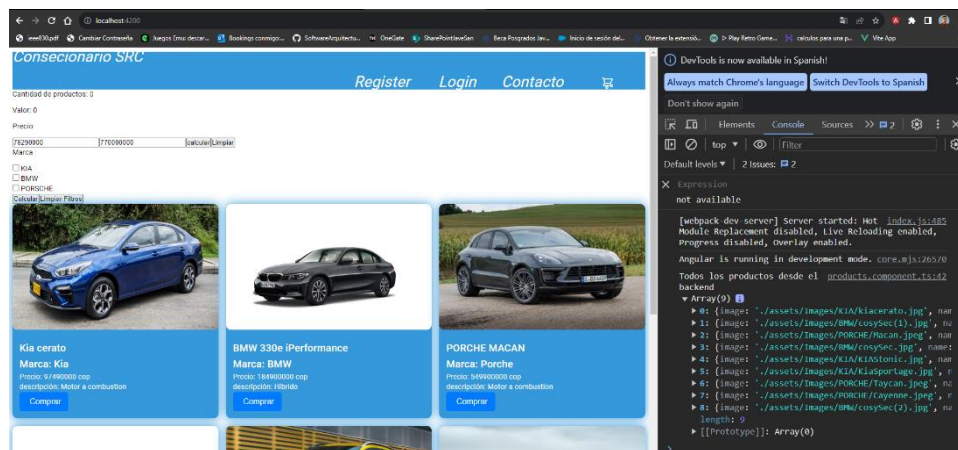
	EPSEARCHB008_searchCarsByBrands_DebeDevolver9Carros_CuandoNoSeEspecifiqueNingunaMarca_DadoQuePorDefectoBuscaraLasTresDisponibles
	EPSEARCHB009_searchCarsByBrands_DebeDevolver9Carros_CuandoNoSeEspecifiqueElParametroMarcas_DadoQuePorDefectoBuscaraLasTresDisponibles

Para probar cómo interactúan los componentes de Angular con el backend Java. Verificar que la comunicación entre el frontend y el backend funcione correctamente, las solicitudes HTTP se envíen y reciban correctamente y que los datos se muestren en el frontend.

- Para la siguiente image, el área de desarrollo backend proporciono el siguiente endpoint:

<http://localhost:8080/api/cars>

para consumir la información de los autos, los datos que traemos



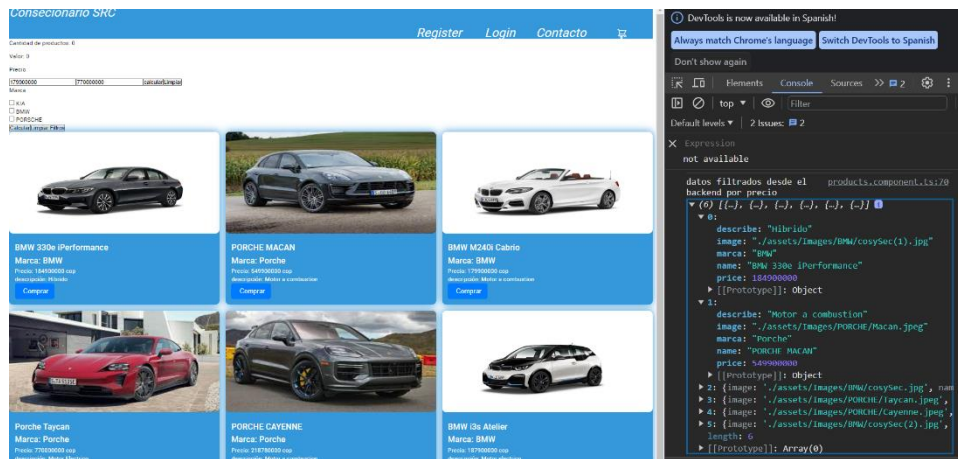
Como se muestra en la imagen anterior mediante el método que nos da TS Console.log(), observamos que la comunicación con el servicio que consume las Apis de todos los productos funciona bien.

- Filtros por precio

Para la siguiente imagen el área de desarrollo backend proporciono el siguiente endpoint:

<http://localhost:8080/api/search?initialPrice=17990000&finalPrice=770000000>

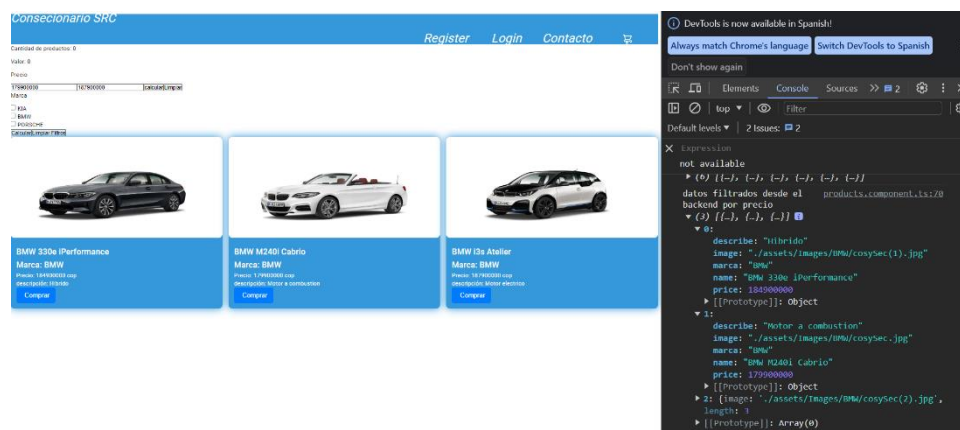
Donde se observa que se envían en los input de texto los límites de precio que estén entre 179900000 y 770000000 nos trae las marcas que se encuentren.



Para la siguiente imagen el área de desarrollo backend proporcione el siguiente endpoint:

<http://localhost:8080/api/search?initialPrice=17990000&finalPrice=187900000>

Donde se observa que se envían en los input de texto los límites de precio que estén entre 179900000 y 187900000 nos trae las marcas que se encuentren.

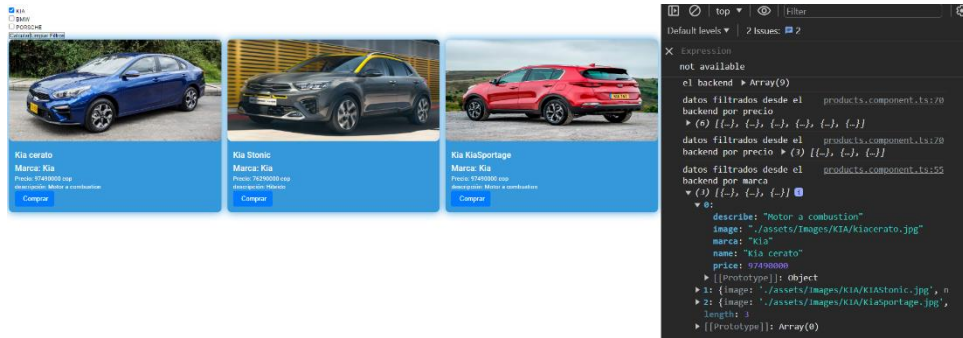


- Filtro por marca

Para la siguiente imagen el área de desarrollo backend proporcione el siguiente endpoint:

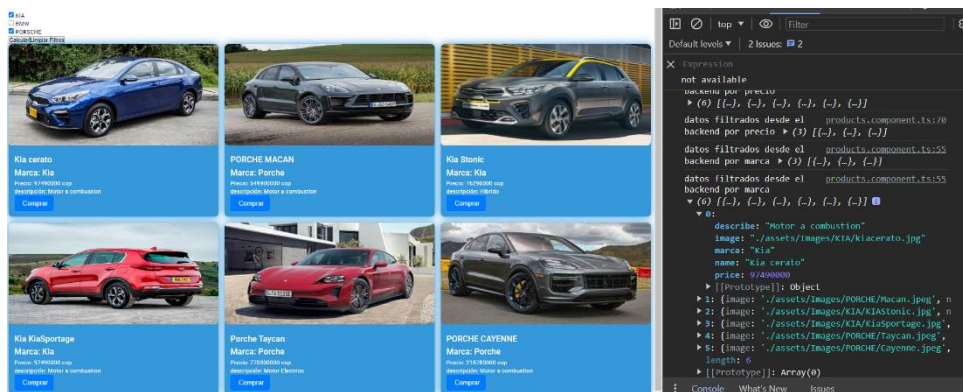
<http://localhost:8080/api/searchBrands?brands=Kia>

Donde se observa que se acciona la marca Kia atreves de un checkbox.



Int: <http://localhost:8080/api/searchBrands?brands=Kia,Porsche>

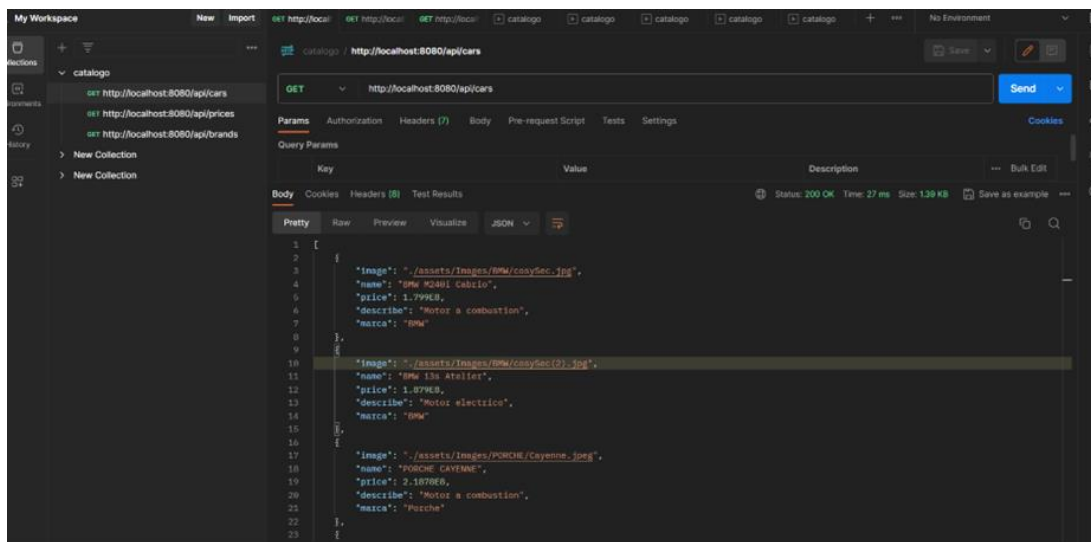
Donde se observa que se acciona la marca Kia y Porsche atreves de un checkbox



- **Pruebas de Sistema:** Probar las funcionalidades principales del catálogo de autos, como buscar, ver detalles de un auto y ordenar autos por precio.
- ✓ Como primer paso se toma el siguiente endpoint:

<http://localhost:8080/api/cars>

Estas pruebas se realizarán mediante la aplicación Postman, donde se obtiene la siguiente respuesta:



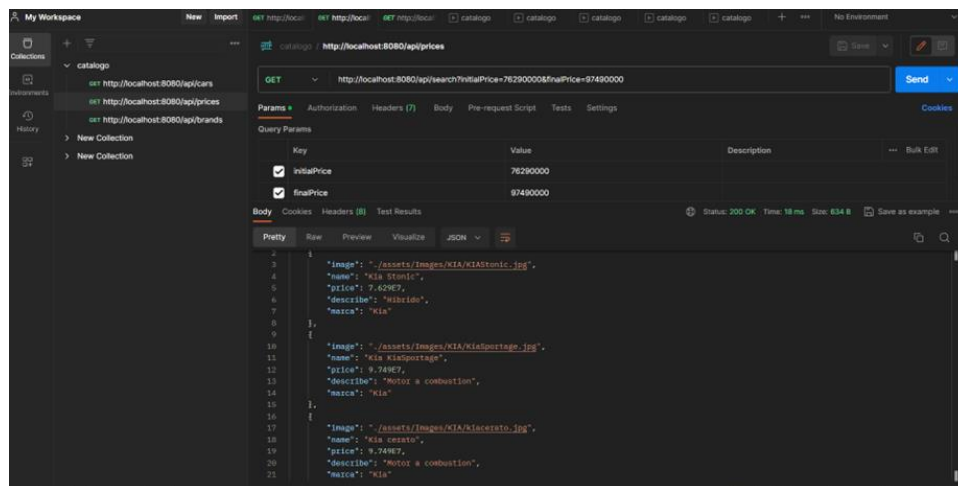
Donde la respuesta como se puede observar de la funcionalidad de obtener la información de los autos que para el ejercicio son 9 con su respectivo catálogo:

```
[
  {
    "image": "./assets/Images/BMW/cosySec.jpg",
    "name": "BMW M240i Cabrio",
    "price": 1.799E8,
    "describe": "Motor a combustion",
    "marca": "BMW"
  },
  {
    "image": "./assets/Images/BMW/cosySec(2).jpg",
    "name": "BMW i3s Atelier",
    "price": 1.879E8,
    "describe": "Motor electrico",
    "marca": "BMW"
  },
  {
    "image": "./assets/Images/PORCHE/Cayenne.jpeg",
    "name": "PORCHE CAYENNE",
    "price": 2.1878E8,
    "describe": "Motor a combustion",
    "marca": "Porche"
  },
  {
    "image": "./assets/Images/KIA/KIAStonic.jpg",
    "name": "Kia Stonic",
    "price": 7.629E7,
    "describe": "Hibrido",
    "marca": "Kia"
  },
  {
    "image": "./assets/Images/KIA/KiaSportage.jpg",
    "name": "Kia KiaSportage",
    "price": 9.749E7,
    "describe": "Motor a combustion",
    "marca": "Kia"
  },
  {
    "image": "./assets/Images/KIA/kiacerato.jpg",
    "name": "Kia cerato",
    "price": 9.749E7,
    "describe": "Motor a combustion",
    "marca": "Kia"
  },
  {
    "image": "./assets/Images/BMW/cosySec(1).jpg",
    "name": "BMW 330e iPerformance",
    "price": 1.849E8,
    "describe": "Hibrido",
    "marca": "BMW"
  },
]
```

```
{
  "image": "./assets/Images/PORCHE/Taycan.jpeg",
  "name": "Porche Taycan",
  "price": 7.7E8,
  "describe": "Motor Electrico",
  "marca": "Porche"
},
{
  "image": "./assets/Images/PORCHE/Macan.jpeg",
  "name": "PORCHE MACAN",
  "price": 5.499E8,
  "describe": "Motor a combustion",
  "marca": "Porche"
}
}
```

✓ Como segunda funcionalidad tenemos:

<http://localhost:8080/api/search?initialPrice=76290000&finalPrice=97490000>

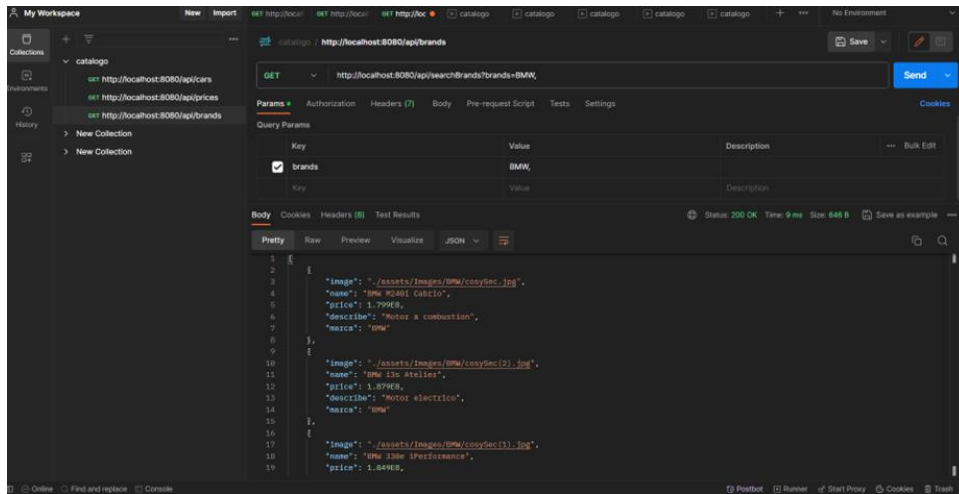


Como se observa en la anterior imagen, el precio mínimo es 78290000 y como valor máximo del rango es 97490000, para la respuesta tenemos 3 autos que están en ese rango de precio.

```
[
  {"image":"./assets/Images/KIA/KIAStonic.jpg","name":"Kia Stonic","price":7.629E7,"describe":"Hibrido","marca":"Kia"},
  {"image":"./assets/Images/KIA/KiaSportage.jpg","name":"Kia KiaSportage","price":9.749E7,"describe":"Motor a combustion","marca":"Kia"},
  {"image":"./assets/Images/KIA/kiacerato.jpg","name":"Kia cerato","price":9.749E7,"describe":"Motor a combustion","marca":"Kia"}
]
```

- ✓ La funcionalidad final es la de la clasificación por marca:

<http://localhost:8080/api/searchBrands?brands=Porche,BMW>,



Para este caso como respuesta de la funcionalidad se tiene solo los autos de marca “BMW” son 3 autos que están en el catálogo.

[

```

{"image": "/assets/Images/BMW/cosySec.jpg", "name": "BMW M240i Cabrio", "price": 1.799E8, "describe": "Motor a combustion", "marca": "BMW"},

```

```

{"image": "/assets/Images/BMW/cosySec(2).jpg", "name": "BMW i3s Atelier", "price": 1.879E8, "describe": "Motor electrico", "marca": "BMW"},

```

```

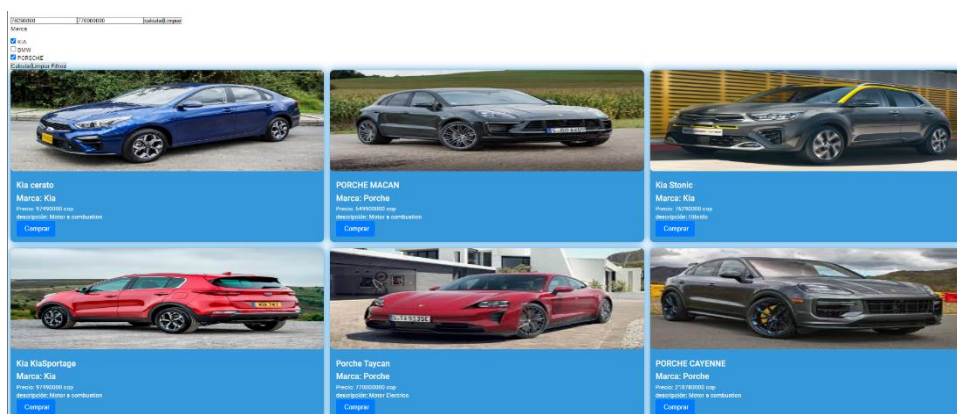
{"image": "/assets/Images/BMW/cosySec(1).jpg", "name": "BMW 330e iPerformance", "price": 1.849E8, "describe": "Hibrido", "marca": "BMW"}

```

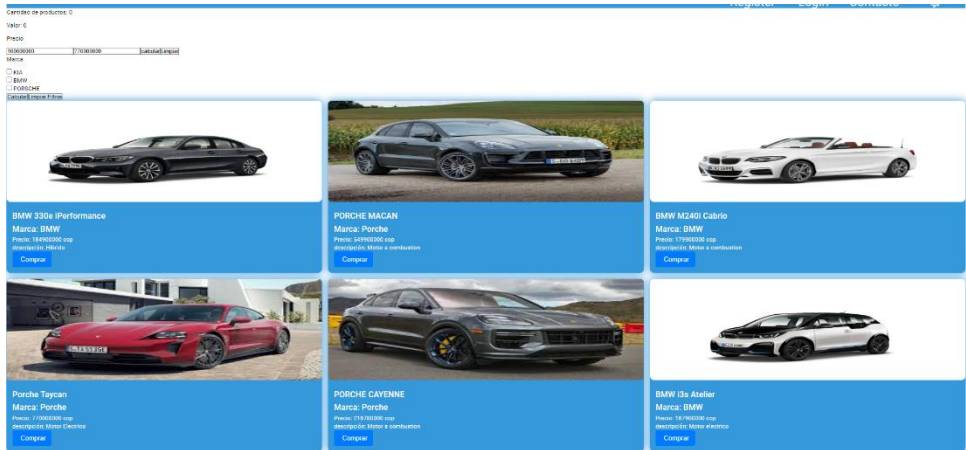
]

- **Pruebas END to END:** Comprueba que las funcionalidades principales de la aplicación funcionen correctamente en conjunto. Realiza pruebas end-to-end para verificar el flujo de usuario, como buscar un auto, ver los detalles y ordenar autos.

Se observa que al utilizar los filtros por marca obtenemos la informacion de los autos de KIA y Porche



En la siguiente imagen se prueba que nos traiga la información filtrada por precio y nos trae la información de BMW y Porche



- **Pruebas de Aceptación:**

Asegurarse de que el catálogo de autos cumple con los requisitos del usuario, incluyendo la funcionalidad y la apariencia.

2. Casos de Prueba de Caja Blanca

a. Aplicar las técnicas para garantizar las métricas de cobertura de sentencia o decisión:

Para el Backend Java, se utiliza JUnit y Mockito para desarrollar las pruebas de unidad que cubran las funciones y métodos del código Java. A continuación, se presentan los casos de pruebas que se contemplan como plan de pruebas para cubrir al menos el 85% del código, y que solo contempla la estructura de datos de Árbol Binario:

Estructura	TestID_Descripción
ArbolBinario	ARB01_estaVacio_DebeRetornarVerdadero_CuandoSeHayaInicializadoSinElementos
	ARB02_estaVacio_DebeRetornarFalso_CuandoElInventarioTengaAlMenosUnCarro
	ARB03_agregarElemento_PorLaPrimeraVez_DebeAsignarloASuNodoRaiz
	ARB04_recorrerInorden_DebeRetornarTodosLosElementosRecorridosDelzqADer_EsDecir_EnOrdenAscendente
NodoArbol	NODAB01_ToString_DebeRetornarLaRepresentacionToStringDelElemento
	NODAB02_Equals_DebeRetornarFalso_CuandoElPrecioDeLosDosElementosDeCadaNodoACompararDifieren
	NODAB03_Equals_DebeRetornarVerdadero_CuandoElPrecioDeLosDosElementosDeCadaNodoACompararSonExactamenteIguales

b. Reporte de Resultados de la Ejecución de las Pruebas

Reporte de JaCoCo – Alcanzando una cobertura esperada del 85% global

← → ↺

localhost:63342/test-automation-java-service/target/site/jacoco/index.html

☆

📄 📄 📄 📄 📄

test-automation-java-service-backend

Sessions

test-automation-java-service-backend

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.test.automation.project.testautomationjavaservicebackend.api.database	<div></div>	55 %	<div></div>	0 %	11	13	1	3	4	6	0	
com.test.automation.project.testautomationjavaservicebackend.api.model	<div></div>	77 %	<div></div>	100 %	5	15	1	11	5	14	0	
com.test.automation.project.testautomationjavaservicebackend	<div></div>	37 %	<div></div>	n/a	1	2	2	3	1	2	0	
com.test.automation.project.testautomationjavaservicebackend.api.model.datastructure	<div></div>	100 %	<div></div>	100 %	0	34	0	56	0	20	0	
com.test.automation.project.testautomationjavaservicebackend.api.services	<div></div>	100 %	<div></div>	100 %	0	13	0	22	0	6	0	
com.test.automation.project.testautomationjavaservicebackend.api.config	<div></div>	100 %	<div></div>	n/a	0	4	0	7	0	4	0	
com.test.automation.project.testautomationjavaservicebackend.api.controllers	<div></div>	100 %	<div></div>	n/a	0	4	0	5	0	4	0	
Total	90 of 621	85 %	14 of 58	75 %	17	85	4	107	10	56	0	

Created with JaCoCo 0.8.11.202310140853

Cobertura de la estructura de datos (datastructure) del Árbol Binario:

← → ↺

localhost:63342/test-automation-java-service/target/site/jacoco/com.test.automation.project.testautomationjavaserviceback

☆

test-automation-java-service-backend > com.test.automation.project.testautomationjavaservicebackend.api.model.datastructure > ArbolBinario

ArbolBinario

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
agregarElemento(Object)	<div></div>	100 %	<div></div>	100 %	0	2	0	4	0	1
recorrerInorden()	<div></div>	100 %	<div></div>	n/a	0	1	0	2	0	1
estaVacio()	<div></div>	100 %	<div></div>	100 %	0	2	0	1	0	1
ArbolBinario()	<div></div>	100 %	<div></div>	n/a	0	1	0	3	0	1
setRaiz(NodoArbol)	<div></div>	100 %	<div></div>	n/a	0	1	0	2	0	1
getRaiz()	<div></div>	100 %	<div></div>	n/a	0	1	0	1	0	1
Total	0 of 44	100 %	0 of 4	100 %	0	8	0	13	0	6

← → ↺

localhost:63342/test-automation-java-service/target/site/jacoco/com.test.automation.project.testautomationjavase

☆

test-automation-java-service-backend > com.test.automation.project.testautomationjavaservicebackend.api.model.datastructure > NodoArbol

NodoArbol

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
agregarElemento(Object)	<div></div>	100 %	<div></div>	100 %	0	6	0	13	0	1
recorrerInorden(ArrayList)	<div></div>	100 %	<div></div>	100 %	0	3	0	6	0	1
compareTo(NodoArbol)	<div></div>	100 %	<div></div>	100 %	0	3	0	6	0	1
NodoArbol(Object)	<div></div>	100 %	<div></div>	n/a	0	1	0	5	0	1
esHojal()	<div></div>	100 %	<div></div>	100 %	0	3	0	1	0	1
esMenor(NodoArbol)	<div></div>	100 %	<div></div>	100 %	0	2	0	1	0	1
equals(Object)	<div></div>	100 %	<div></div>	n/a	0	1	0	1	0	1
setElemento(Object)	<div></div>	100 %	<div></div>	n/a	0	1	0	2	0	1
setIzq(NodoArbol)	<div></div>	100 %	<div></div>	n/a	0	1	0	2	0	1
setDer(NodoArbol)	<div></div>	100 %	<div></div>	n/a	0	1	0	2	0	1
toString()	<div></div>	100 %	<div></div>	n/a	0	1	0	1	0	1
getElemento()	<div></div>	100 %	<div></div>	n/a	0	1	0	1	0	1
getIzq()	<div></div>	100 %	<div></div>	n/a	0	1	0	1	0	1
getDer()	<div></div>	100 %	<div></div>	n/a	0	1	0	1	0	1
Total	0 of 153	100 %	0 of 24	100 %	0	26	0	43	0	14

3. Casos de Prueba de Caja Negra

a. Diseñar los Casos de Prueba

- Clases de Equivalencia: Crea casos de prueba para diferentes categorías de datos de entrada, como autos de diferentes marcas y precios.
- Valores al Límite: Pruebas con datos que están en los límites, autos con precios extremadamente bajos o altos (rangos).

- Tablas de Decisión: Crea tablas que describan combinaciones de entradas y resultados esperados, como la búsqueda de autos de una marca específica. Diseñar casos de prueba para diferentes combinaciones de solicitudes HTTP (por ejemplo, buscar autos de diferentes marcas y precios) y verifica que el backend responda correctamente.

	CLASE DE EQUIVALENCIA o VARIABLE CONDICIÓN	REPRESENT ANTE	TIPO
CONDICIONES	minPrice		
	minPrice < 0	-1	Inválido
	minPrice >= 0	0	Válido
	minPrice < maxprice	1999999999	Inválido
	minprice = maxprice	2000000000	Inválido
	minprice > maxprice	2000000001	Inválido
	MaxPrice MaxPrice <= 2000000000	80000000	Válido
	MaxPrice > 2000000000	2000000001	Inválido
	maxPrice < 0	-1	Inválido
	maxPrice = 0	0	Inválido
ACCIONES	Llama a limpiarFiltro() y "entre" en log Llama a getFilterByPrice() y sin productos Llama a getFilterByPrice()		

b. Documentación de los Casos de Prueba

Documenta todos los casos de prueba de caja negra, las entradas, las condiciones de prueba y los resultados esperados.

ID de la prueba	descripcion	minPrice	MaxPrice	Resultado Esperado
1	Prueba límite inferior de minPrice	-1	2000000000	Llama a limpiarFiltro() y "entre" en log
2	Prueba en el borde inferior de minPrice	0	2000000000	Llama a getFilterByPrice()
3	Prueba límite superior de maxPrice	76290000	97490000	Llama a getFilterByPrice()
4	Prueba en el borde superior de maxPrice	2000	2000000001	Llama a limpiarFiltro() y "entre" en log
5	Prueba con ambos valores en el borde inferior	0	0	Llama a getFilterByPrice() y sin productos
6	Prueba con ambos valores en el borde superior	2000000000	2000000000	Llama a getFilterByPrice() y sin productos
7	Prueba con minPrice mayor que maxPrice	1001	1000	Llama a getFilterByPrice() y sin productos
8	prueba con minPrice 0 y menor negativo maxPrice	0	-1	Llama a limpiarFiltro() y "entre" en log
9	Prueba límite inferior de minPrice, Prueba en el borde superior de maxPrice	-1	2000000001	Llama a limpiarFiltro() y "entre" en log

c. Reporte de los Resultados de la Ejecución de las Pruebas

Registra los resultados de la ejecución de los casos de prueba de caja negra, indicando si las funciones probadas se comportaron bien.

Se observa que los casos de pruebas escritos en la siguiente imagen se muestran por la imagen de Karma inclusive es capaz de mostrar los logs que se programan con la herramienta de Jazmin

```
TS products.component.spec.ts M TS products.service.spec.ts M X TS products.service.ts
consecionario-mat-pruebas > src > app > services > TS products.service.spec.ts > fdescribe('ProductsService') callback
17     providers: [ProductsService]
18   });
19
20   service = TestBed.inject(ProductsService);
21   httpMock = TestBed.inject(HttpTestingController);
22 });
23
24   afterEach(() => {
25     httpMock.verify(); // Verificar que no hay peticiones pendientes
26   });
27   //Todos los productos
28   it('deberia traer todos los productos ', () => {
29
30 >     const products: Product[] = [ ...
94   ];
95   service.getAllProducts().subscribe(data => {
96     expect(data).toEqual(products);
97   });
98
99   const req = httpMock.expectOne('/api/cars');
100   expect(req.request.method).toBe('GET');
101   req.flush(products);
102 });
```

Imagen de Karma

```
START:
ProductsService
  ✓ no deberia traerme ningun producto 1001 y1000
  ✓ no deberia traerme ningun producto 0 y-1
  ✓ deberia traer todos los productos 0y 2000000000
  ✓ deberia traer todos los productos
  ✓ no deberia traerme ningun producto -1 y2000000001
  ✓ no deberia traer datos para el rango de -1y 2000000000
  ✓ no deberia traerme ningun producto 2000 y2000000001
  ✓ deberia traer todos los productos KIA 76290000 y97490000
  ✓ no deberia traerme ningun producto 20000000000 y20000000000
  ✓ no deberia traerme ningun producto 0 y0
ProductsComponent
  ✓ should create
LOG: 'Todos los productos desde el backend', []
  ✓ should load user products on init
LOG: 'datos filtrados desde el backend por marca', [Object{image: './assets/Images/KIA/KiaSportage.jpg', name: 'Kia KiaSportage', price: 97490000, describe: 'Motor a combusti
a combustion', marca: 'Kia'}]
  ✓ should filter by brand
LOG: 'Todos los productos desde el backend', []
  ✓ should load all products from service on init
ProductComponent
  ✓ should create (skipped)
DataService
  ✓ should be created (skipped)
LoginComponent
  ✓ should create (skipped)
NavComponent
  ✓ should create (skipped)
ShoppingCartComponent
  ✓ should create (skipped)
RegisterComponent
  ✓ should create (skipped)
ContactComponent
  ✓ should create (skipped)
ImgComponent
  ✓ should create (skipped)
AppComponent
  ✓ should have as title 'consecionario-mat-pruebas' (skipped)
  ✓ should create the app (skipped)
  ✓ should render title (skipped)

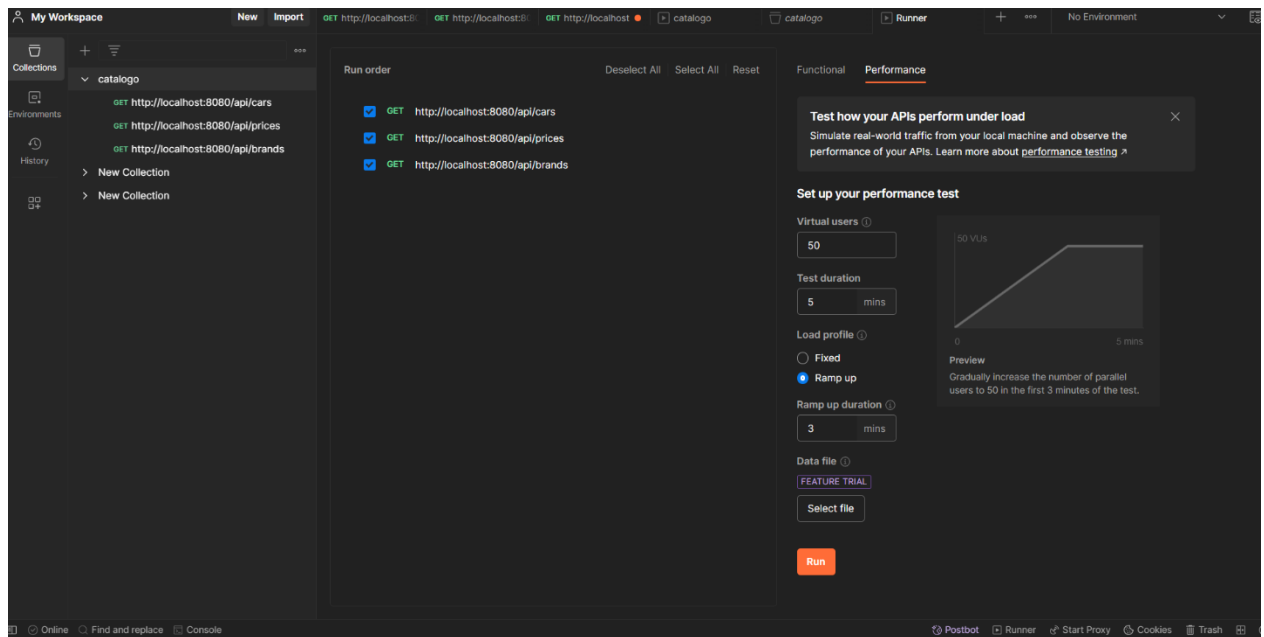
Finished in 0.17 secs / 0.125 secs @ 08:45:47 GMT-0500 (hora estándar de Colombia)

SUMMARY:
✓ 14 tests completed
i 11 tests skipped
```

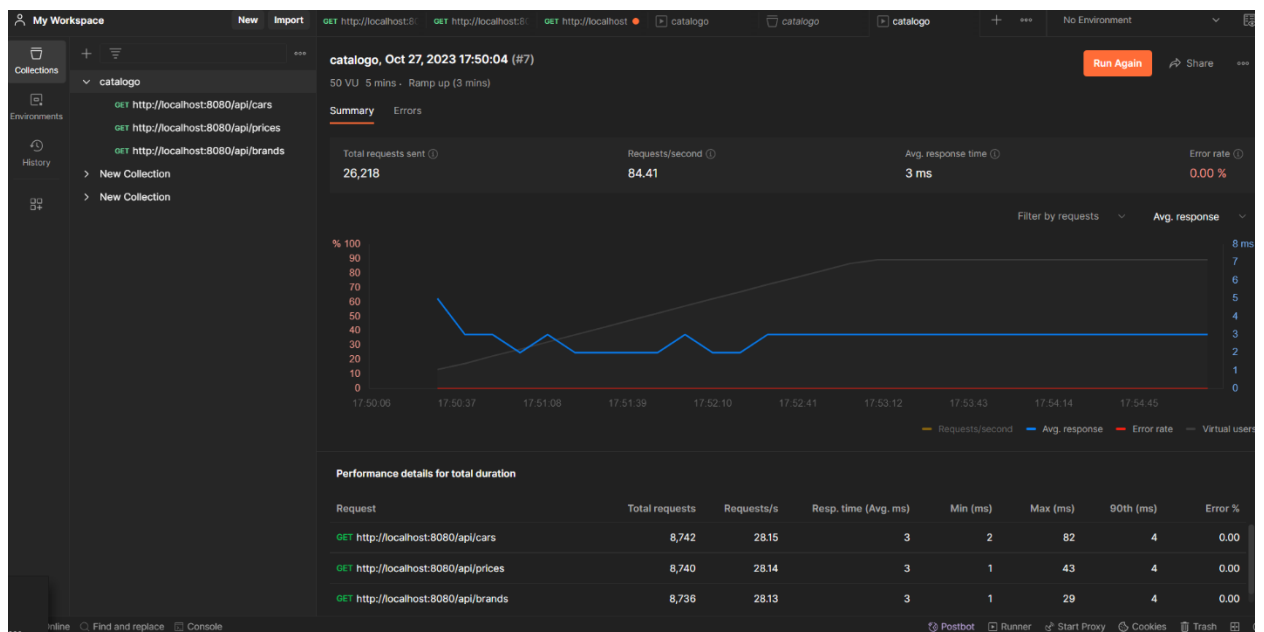
4. Detalle del Diseño de las Pruebas de Desempeño

Para las pruebas de desempeño, debes especificar cómo planeas evaluar el rendimiento de la aplicación web. Se utilizará la herramienta Postman para simular la carga y medir el rendimiento del backend Java en diferentes situaciones, como un alto número de solicitudes de búsqueda de autos.

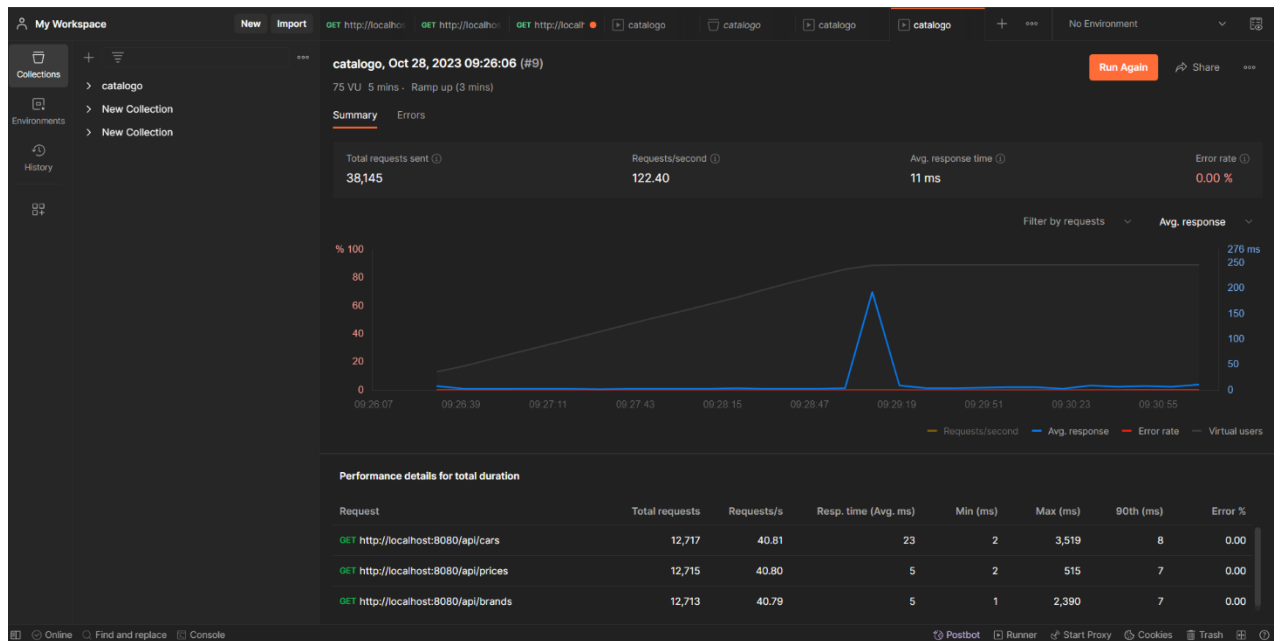
Para las pruebas de desempeño se trabajará con una colección en Postman, donde se encuentran los endpoints de las funcionalidades del proyecto, para esto se realiza lo siguiente:



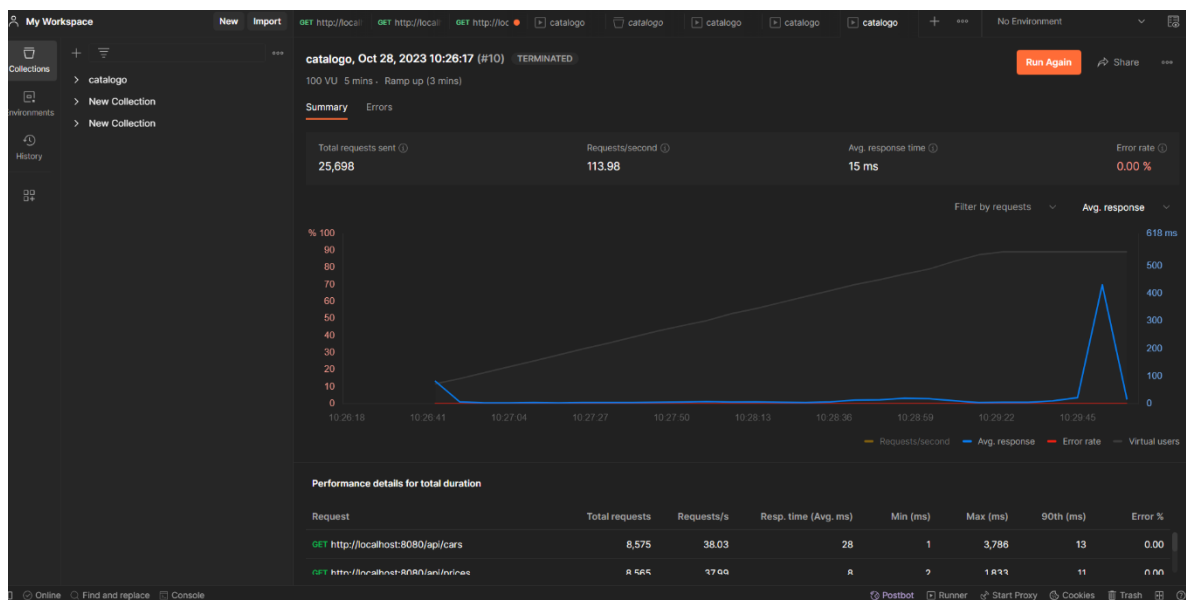
Para este caso se buscará simular una cantidad de usuarios durante 5 minutos (Por complejidad del proyecto que no es tan alto, se irán incrementado los usuarios de 50 hasta 100)



Para el primer caso se tomó 50 usuarios en la opción de rampa con periodo de carga en rampa de 3 min y de carga estable de 2 min, se visualizó que con 50 usuarios responde de manera correcta y no se presentan errores.



Para este segundo caso se tomaron los mismos parámetros exceptuando el valor de los usuarios, para este caso se incrementó a 75 usuarios, para este caso se observa que la prueba en los 3 endpoint se desarrolla de manera correcta sin presentar errores.



Para este tercer y último caso el tiempo de ejecución es el mismo y solo se ve modificado la cantidad de usuarios que serán 100, para este caso se observa también que el funcionamiento es correcto y no se presentan errores en la respuesta de alguna de las 3 funcionalidades.

5. Resultados de la Ejecución de las Pruebas de Desempeño

Documentar los resultados de las pruebas de desempeño, incluyendo métricas de rendimiento, tiempos de respuesta y cualquier problema identificado.

Para los resultados de desempeño se tiene en cuenta tanto los tiempos que se obtuvieron en cada prueba y las métricas de las respuestas de cada solicitud en las funcionalidades.

Peticiones con tiempos de respuesta más lentos

Método de rampa con 50 usuarios.

1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
GET http://localhost:8080/api/cars http://localhost:8080/api/cars	3	4	5	11	2	82
GET http://localhost:8080/api/prices http://localhost:8080/api/search?initialPrice=76290000&finalPrice=97490000	3	4	4	7	1	43
GET http://localhost:8080/api/brands http://localhost:8080/api/searchBrands?brands=Porche,BMW,	3	4	4	6	1	29

Método de rampa con 75 usuarios.

1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
GET http://localhost:8080/api/cars http://localhost:8080/api/cars	23	8	12	73	2	3,519
GET http://localhost:8080/api/prices http://localhost:8080/api/search?initialPrice=76290000&finalPrice=97490000	5	7	10	22	2	515
GET http://localhost:8080/api/brands http://localhost:8080/api/searchBrands?brands=Porche,BMW,	5	7	9	20	1	2,390

Método de rampa con 100 usuarios.

1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
GET http://localhost:8080/api/cars http://localhost:8080/api/cars	28	13	28	216	1	3,786
GET http://localhost:8080/api/prices http://localhost:8080/api/search? initialPrice=76290000&finalPrice=97490000	8	11	19	72	2	1,833
GET http://localhost:8080/api/brands http://localhost:8080/api/searchBrands?brands=Porche,BMW,	8	11	17	65	1	2,292

Métricas de cada solicitud

Valores para 50 usuarios virtuales

2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
GET http://localhost:8080/api/cars http://localhost:8080/api/cars	8,742	28.16	2	3	4	82	0
GET http://localhost:8080/api/prices http://localhost:8080/api/search? initialPrice=76290000&finalPrice=97490000	8,740	28.15	1	3	4	43	0
GET http://localhost:8080/api/brands http://localhost:8080/api/searchBrands?brands=Porche,BMW,	8,736	28.14	1	3	4	29	0

Valores para 75 usuarios virtuales

2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
GET http://localhost:8080/api/cars http://localhost:8080/api/cars	12,717	40.95	2	23	8	3,519	0
GET http://localhost:8080/api/prices http://localhost:8080/api/search? initialPrice=76290000&finalPrice=97490000	12,715	40.94	2	5	7	515	0
GET http://localhost:8080/api/brands http://localhost:8080/api/searchBrands?brands=Porche,BMW,	12,713	40.94	1	5	7	2,390	0

Valores para 100 usuarios virtuales

2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
GET http://localhost:8080/api/cars http://localhost:8080/api/cars	8,575	38.09	1	28	13	3,786	0
GET http://localhost:8080/api/prices http://localhost:8080/api/search? initialPrice=76290000&finalPrice=97490000	8,565	38.05	2	8	11	1,833	0
GET http://localhost:8080/api/brands http://localhost:8080/api/searchBrands?brands=Porche,BMW,	8,558	38.02	1	8	11	2,292	0

6. Automatización de Pruebas

De acuerdo con el plan de pruebas presentado en el punto 1.b para pruebas de componente (unitaria) y pruebas de integración, se presentan a continuación las herramientas que se utilizaron para automatizarlas, en cada build del proyecto (mvn build) y en cada Pull Request generado en el repositorio en GitHub:


- **Pruebas unitarias:** JUnit – Mockito - JaCoCo – GitHub Actions.
- **Pruebas de integración:** JUnit + Karate.

7. Reporte de Resultados de la Ejecución de las Pruebas Automatizadas

- **Pruebas Unitarias o de Componente:**


Comprobación de la cobertura de sentencia generada por un GitHub Actions que cumple la función de la etapa de **Continuous Integration (CI)**, cuyo progreso se muestra a continuación y que fue generado en cada Pull Request generado antes de ser mergeado en la rama principal (main) como se puede comprobar en el historial de Pull Requests: <https://github.com/willy23martin/test-automation-java-service/pulls?q=is%3Apr+is%3Aclosed>

% Cobertura	Evidencias
49.52%	Pull Request / PR's Commit: https://github.com/willy23martin/test-automation-java-service/pull/1/commits/a69ec3c345a0044fb487a6b670ce8628909ac0cc GitHub Action execution: https://github.com/willy23martin/test-automation-java-service/actions/runs/6643289475/job/18050041346


[willy23martin](#) / [test-automation-java-service](#)


[Code](#)
[Issues](#)
[Pull requests](#)
[1](#)
[Actions](#)
[Projects](#)
[Wiki](#)
[Security](#)
[Insights](#)
[Settings](#)


← Quality Code Check


Removing push as an event for GHA #6


[Summary](#)


Jobs

 build (16, ubuntu-latest)

 **Code Coverage**

Run details

 Usage

 Workflow file

GitHub Actions / Code Coverage
 succeeded yesterday in 0s

Code Coverage 49.52 %

This run completed at **10/25/2023 16:24:40**

DETAILS

Coverage Report: Test coverage results

- test-automation-java-service-backend


Outcome	Value
Code Coverage %	49.52%
✓ Number of Lines Covered	52
✗ Number of Lines Missed	53
Total Number of Lines	105

54.29%


Pull Request / PR's Commit: <https://github.com/willy23martin/test-automation-java-service/pull/2/commits/ae837a72af58a03f4c320b8702e6450074e5cee9>

GitHub Action execution: https://github.com/willy23martin/test-automation-java-service/pull/2/checks?check_run_id=18062224277


Adding Mockito and uni tests JUnit to cover test cases for ArbolBinario Data Structure #2


Merged willy23martin merged 6 commits into [main](#) from [test_coverage_branch](#) (14 hours ago)

[Conversation](#)
[Commits](#)
[2](#)
[Checks](#)
[Files changed](#)


 Adding Mockito and uni tests JUnit to cover raiz cases for ArbolBinario... ae837a7

Quality Code Check
 on: pull_request

 build (16, ubuntu-latest)

 **Code Coverage**

GitHub Actions / Code Coverage
 succeeded yesterday in 0s

Code Coverage 54.29 %

This run completed at **10/25/2023 23:18:03**

DETAILS

Coverage Report: Test coverage results

- test-automation-java-service-backend

Outcome	Value
Code Coverage %	54.29%
✓ Number of Lines Covered	57
✗ Number of Lines Missed	48
Total Number of Lines	105

58.10%

Pull Request / PR's Commit: <https://github.com/willy23martin/test-automation-java-service/pull/2/commits/6c6a9ec45adc7877137f37c0308eb4930ef2aec6>

GitHub Action execution: <https://github.com/willy23martin/test-automation-java-service/actions/runs/6647302127/job/18062460823>

willy23martin / test-automation-java-service

<> Code

Issues

Pull requests 1

Actions

Projects

Wiki

Security

Insights

Settings

← Quality Code Check

✓

Adding Mockito and uni tests JUnit to cover test cases for ArbolBinario Data Structure #8

Summary

Jobs

build (16, ubuntu-latest)

Code Coverage

Run details

Usage

Workflow file

GitHub Actions / Code Coverage

succeeded yesterday in 0s

Code Coverage 58.10 %

This run completed at 10/25/2023 23:20:39

DETAILS

Coverage Report: Test coverage results

test-automation-java-service-backend

Outcome	Value
Code Coverage %	58.1%
✓ Number of Lines Covered	61
✗ Number of Lines Missed	44
Total Number of Lines	105

60%

Pull Request / PR's Commit: <https://github.com/willy23martin/test-automation-java-service/pull/2/commits/6c6a9ec45adc7877137f37c0308eb4930ef2aec6>

GitHub Action execution: <https://github.com/willy23martin/test-automation-java-service/actions/runs/6647302127/job/18062460823>

willy23martin / test-automation-java-service

<> Code

Issues

Pull requests 1

Actions

Projects

Wiki

Security

Insights

Settings

← Quality Code Check

✓

Adding Mockito and uni tests JUnit to cover test cases for ArbolBinario Data Structure #9

Summary

Jobs

build (16, ubuntu-latest)

Code Coverage

Run details

Usage

Workflow file

GitHub Actions / Code Coverage

succeeded yesterday in 0s

Code Coverage 60.00 %

This run completed at 10/25/2023 23:25:46

DETAILS

Coverage Report: Test coverage results

test-automation-java-service-backend

Outcome	Value
Code Coverage %	60%
✓ Number of Lines Covered	63
✗ Number of Lines Missed	42
Total Number of Lines	105

70.75%

Pull Request / PR's Commit: <https://github.com/willy23martin/test-automation-java-service/pull/2/commits/255dc66b6abcbdf3c821d5aa965a714c9a3853f6>

GitHub Action execution: <https://github.com/willy23martin/test-automation-java-service/runs/18063186567>

☰

🐙

willy23martin / test-automation-java-service

<> Code

🕒 Issues

🔗 Pull requests

🔄 Actions

📁 Projects

📖 Wiki

🛡 Security

📊 Insights

⚙ Settings

✓

Covering recorner inorden

test_coverage_bran... 255dc66

✓ Quality Code Check

on: pull_request

✓ build (16, ubuntu-latest)

✓ Code Coverage

GitHub Actions / Code Coverage

succeeded yesterday in 0s

Code Coverage 70.75 %

This run completed at 10/26/2023 00:01:12

DETAILS

Coverage Report: Test coverage results

• test-automation-java-service-backend

Outcome	Value
Code Coverage %	70.75%
✓ Number of Lines Covered	75
✗ Number of Lines Missed	31
Total Number of Lines	106

77.57%

Pull Request / PR's Commit: <https://github.com/willy23martin/test-automation-java-service/pull/2/commits/f5430b8ba83542da664441e696bcd790312e90c0>

GitHub Action execution: <https://github.com/willy23martin/test-automation-java-service/runs/18082052023>

☰

🐙

willy23martin / test-automation-java-service

<> Code

🕒 Issues

🔗 Pull requests

🔄 Actions

📁 Projects

📖 Wiki

🛡 Security

📊 Insights

⚙ Settings

✓

Covering the toString and equals methods for NodaAr...

test_coverage_bran... f5430b8

✓ Quality Code Check

on: pull_request

✓ build (16, ubuntu-latest)

✓ Code Coverage

GitHub Actions / Code Coverage

succeeded 15 hours ago in 0s

Code Coverage 77.57 %

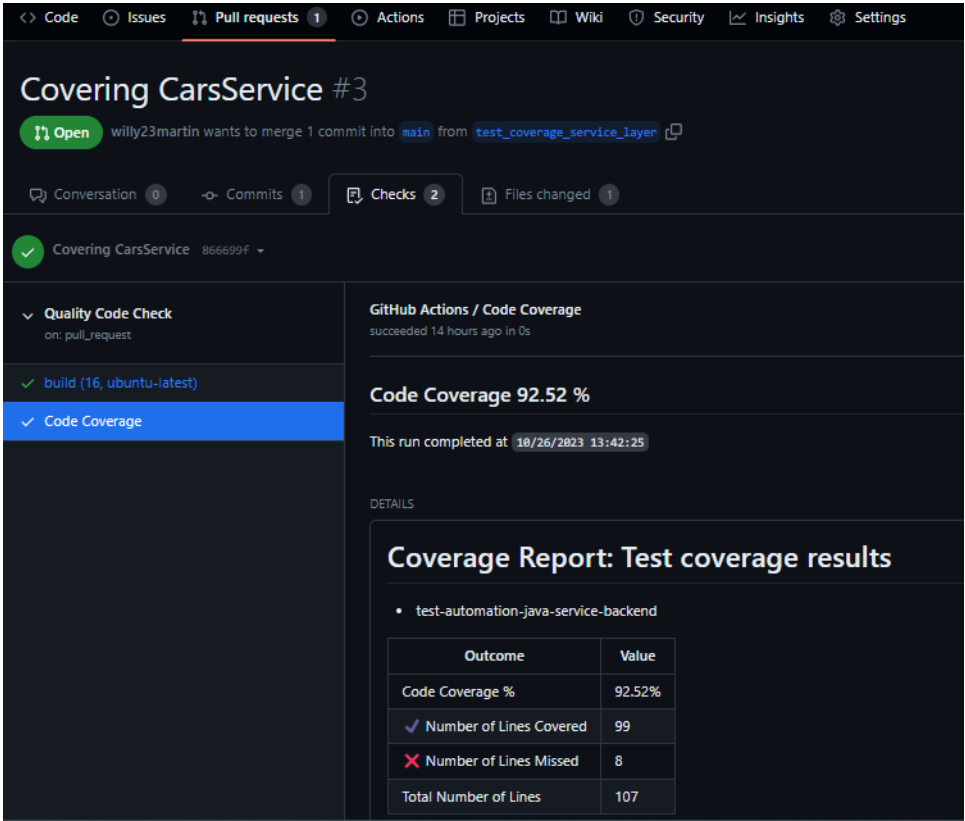
This run completed at 10/26/2023 12:48:05

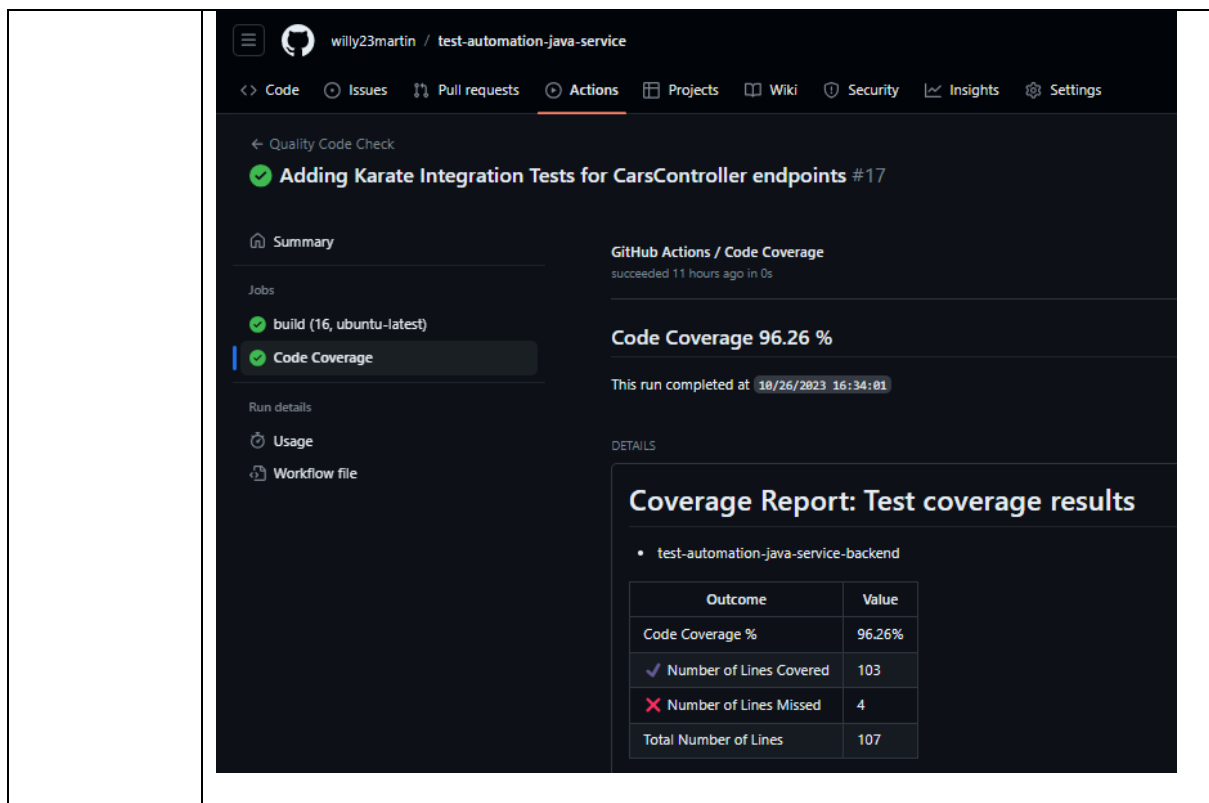
DETAILS

Coverage Report: Test coverage results

• test-automation-java-service-backend

Outcome	Value
Code Coverage %	77.57%
✓ Number of Lines Covered	83
✗ Number of Lines Missed	24
Total Number of Lines	107

<p>92.52%</p>	<p>Pull Request / PR's Commit: https://github.com/willy23martin/test-automation-java-service/pull/3/files</p> <p>GitHub Action execution: https://github.com/willy23martin/test-automation-java-service/pull/3/checks?check_run_id=18084247519</p>  <p>The screenshot shows a GitHub Pull Request interface for 'Covering CarService #3'. The PR is open and shows a merge of 1 commit from 'test_coverage_service_layer' into 'main'. The 'Checks' section shows a 'Code Coverage' check with a status of 'succeeded' and a value of '92.52 %'. The 'Details' section shows a 'Coverage Report: Test coverage results' for 'test-automation-java-service-backend'.</p> <table border="1"> <thead> <tr> <th>Outcome</th><th>Value</th></tr> </thead> <tbody> <tr> <td>Code Coverage %</td><td>92.52%</td></tr> <tr> <td>✓ Number of Lines Covered</td><td>99</td></tr> <tr> <td>✗ Number of Lines Missed</td><td>8</td></tr> <tr> <td>Total Number of Lines</td><td>107</td></tr> </tbody> </table>	Outcome	Value	Code Coverage %	92.52%	✓ Number of Lines Covered	99	✗ Number of Lines Missed	8	Total Number of Lines	107
Outcome	Value										
Code Coverage %	92.52%										
✓ Number of Lines Covered	99										
✗ Number of Lines Missed	8										
Total Number of Lines	107										
<p>96.26%</p>	<p>Pull Request / PR's Commit: https://github.com/willy23martin/test-automation-java-service/pull/4/files</p> <p>GitHub Action execution: https://github.com/willy23martin/test-automation-java-service/actions/runs/6657077957/job/18091073564</p>										




Adicionalmente, con **JaCoCo** se generaban los reportes de cobertura de sentencia y caminos, cuyo último estado (que cubre el 85% de sentencias y 75% de branches) se presenta a continuación:


Se puede comprobar que se alcanzó la cobertura planificada en sentencia y aquellas ramas o sentencias no cubiertas no corresponden a la ruta crítica de las pruebas.

- **Pruebas de Integración: para el Backend API:**


De acuerdo con lo planificado, los resultados de las pruebas de Integración se presentan a continuación en los reportes de Karate.

<div> <div>  <div> <div>3</div> <div>0</div> </div> </div> <div> <div>Karate Labs</div> <div>Features</div> <div>2023-10-27 02:56:58 p. m.</div> </div> </div>					
Tags Timeline					
Feature	Title	Passed	Failed	Scenarios	Time (ms)
com/test/automation/project/testautomationjavaservicebackend/api/karate/GetCars.feature	GET Cars API returns nine cars	1	0	1	26
com/test/automation/project/testautomationjavaservicebackend/api/karate/SearchByPriceRange.feature	GET Search Cars By Price Range	7	0	7	234
com/test/automation/project/testautomationjavaservicebackend/api/karate/SearchByBrand.feature	GET Search Cars By Brand	9	0	9	269

Endpoint: GET /api/cars

<div> <div>  <div> <div>1</div> <div>0</div> </div> </div> <div> <div>Karate Labs</div> <div>Scenarios</div> <div>2023-10-27 02:56:56 p. m.</div> <div>[1:3] EPCARS001 el inventario contiene un listado de nueve carros precargados</div> </div> </div>					
Summary Tags Feature: com/test/automation/project/testautomationjavaservicebackend/api/karate/GetCars.feature GET Cars API returns nine cars					
Scenario: [1:3] EPCARS001 el inventario contiene un listado de nueve carros precargados					ms: 26
4	Given url baseUrl + '/api/cars'				1
5	When method get				19
6	Then status 200				0
7	And match response == '#[9]'				5


Endpoint: GET api/search?initialPrice=<>&finalPrice=<>

<div> <div>  <div> <div>7</div> <div>0</div> </div> </div> <div> <div>Karate Labs</div> <div>Scenarios</div> <div>2023-10-27 02:56:57 p. m.</div> <div>[1:3] EPSEARCHP001_searchCarsB</div> <div>[2:11] EPSEARCHP002_searchCarsB</div> <div>[3:19] EPSEARCHP003_searchCarsB</div> <div>[4:27] EPSEARCHP004_searchCarsB</div> <div>[5:35] EPSEARCHP005_searchCarsB</div> <div>[6:43] EPSEARCHP006_searchCarsB</div> <div>[7:51] EPSEARCHP007_searchCarsB</div> </div> </div>					
Summary Tags Feature: com/test/automation/project/testautomationjavaservicebackend/api/karate/SearchByPriceRange.feature GET Search Cars By Price Range					
Scenario: [1:3] EPSEARCHP001_searchCarsByPriceRange_DebeRetornarUnaListaConTresCarros_CuandoElPrecioDelCarroNoSuperaLos100Millones					ms: 21
4	Given url baseUrl + '/api/search'				1
5	And param initialPrice = 0.0				1
6	And param finalPrice = 100000000.0				1
7	When method get				14
8	Then status 200				0
9	And match response == '#[3]'				4
Scenario: [2:11] EPSEARCHP002_searchCarsByPriceRange_DebeRetornarUnaListaConTresCarros_CuandoElPrecioDelCarroEstribaEntre100MillonesY200Millones					ms: 43
12	Given url baseUrl + '/api/search'				1
13	And param initialPrice = 100000000.0				2
14	And param finalPrice = 200000000.0				1
15	When method get				35
16	Then status 200				0
17	And match response == '#[3]'				2
Scenario: [3:19] EPSEARCHP003_searchCarsByPriceRange_DebeRetornarUnaListaConTresCarros_CuandoElPrecioDelCarroSupereLos200Millones					ms: 34
20	Given url baseUrl + '/api/search'				3
21	And param initialPrice = 200000000.0				1
22	And param finalPrice = 1000000000.0				0
23	When method get				27
24	Then status 200				0
25	And match response == '#[3]'				3
Scenario: [4:27] EPSEARCHP004_searchCarsByPriceRange_DebeRetornarUnaListaCon6Carros_CuandoElPrecioDelCarroSupereLos100Millones					ms: 32
28	Given url baseUrl + '/api/search'				1
29	And param initialPrice = 100000000.0				0
30	And param finalPrice = 1000000000.0				0
31	When method get				20
32	Then status 200				0
33	And match response == '#[6]'				10

<div> <div> <div> <div>7</div> <div>0</div> </div> </div> <div> <div>Scenarios</div> <div>2023-10-27 02:56:57 p. m.</div> <div>[1:3] EPSEARCHP001_searchCarsByPriceRange_DebeRetornarUnaListaCon6Carros_CuandoElPrecioDelCarroSuperaLos100Millones</div> <div>[2:11] EPSEARCHP002_searchCarsByPriceRange_DebeRetornarUnaListaCon6Carros_CuandoElPrecioDelCarroSuperaLos100Millones</div> <div>[3:19] EPSEARCHP003_searchCarsByPriceRange_DebeRetornarUnaListaCon6Carros_CuandoElPrecioDelCarroSuperaLos100Millones</div> <div>[4:27] EPSEARCHP004_searchCarsByPriceRange_DebeRetornarUnaListaCon6Carros_CuandoElPrecioDelCarroSuperaLos100Millones</div> <div>[5:35] EPSEARCHP005_searchCarsByPriceRange_DebeRetornarUnaListaCon6Carros_CuandoElPrecioDelCarroSuperaLos100Millones</div> <div>[6:43] EPSEARCHP006_searchCarsByPriceRange_DebeRetornarUnaListaCon6Carros_CuandoElPrecioDelCarroSuperaLos100Millones</div> <div>[7:51] EPSEARCHP007_searchCarsByPriceRange_DebeRetornarUnaListaCon6Carros_CuandoElPrecioDelCarroSuperaLos100Millones</div> </div> </div>		<div> <div> <div>42 And param initialPrice = 100000000.0</div> <div>23 When method get</div> <div>24 Then status 200</div> <div>25 And match response == '#[3]'</div> </div> <div>Scenario: [4:27] EPSEARCHP004_searchCarsByPriceRange_DebeRetornarUnaListaCon6Carros_CuandoElPrecioDelCarroSuperaLos100Millones ms: 32</div> <div> <div>28 Given url baseUrl + '/api/search'</div> <div>29 And param initialPrice = 100000000.0</div> <div>30 And param finalPrice = 100000000.0</div> <div>31 When method get</div> <div>32 Then status 200</div> <div>33 And match response == '#[6]'</div> </div> <div>Scenario: [5:35] EPSEARCHP005_searchCarsByPriceRange_DebeRetornarUnaListaVacía_CuandoElPrecioDelCarroSeInferiorA50Millones ms: 32</div> <div> <div>36 Given url baseUrl + '/api/search'</div> <div>37 And param initialPrice = 0.0</div> <div>38 And param finalPrice = 50000000.0</div> <div>39 When method get</div> <div>40 Then status 200</div> <div>41 And match response == '#[10]'</div> </div> <div>Scenario: [6:43] EPSEARCHP006_searchCarsByPriceRange_DebeRetornarUnaListaCon9Carros_CuandoElPrecioDelCarroSuperaLos50Millones ms: 41</div> <div> <div>44 Given url baseUrl + '/api/search'</div> <div>45 And param initialPrice = 50000000.0</div> <div>46 And param finalPrice = 100000000.0</div> <div>47 When method get</div> <div>48 Then status 200</div> <div>49 And match response == '#[9]'</div> </div> <div>Scenario: [7:51] EPSEARCHP007_searchCarsByPriceRange_DebeLanzarUnIllegalArgumentException_CuandoElPrecioInicialDelCarroSuperaElPrecioFinalParaFiltrar ms: 32</div> <div> <div>52 Given url baseUrl + '/api/search'</div> <div>53 And param initialPrice = 100000000.0</div> <div>54 And param finalPrice = 50000000.0</div> <div>55 When method get</div> <div>56 Then status 400</div> <div>57 And match karate.toString(response) contains 'Bad Request'</div> </div> </div>
--	--	---

Endpoint: GET /api/searchBrands?brands=<>,<>,<>,...

<div> <div> <div> <div>9</div> <div>0</div> </div> </div> <div> <div>Scenarios</div> <div>2023-10-27 02:56:57 p. m.</div> <div>[1:3] EPSEARCHB001_searchCarsByBrands_DebeDevolver3Carros_CuandoSeFiltrePorLaMarcaBMW</div> <div>[2:10] EPSEARCHB002_searchCarsByBrands_DebeDevolver3Carros_CuandoSeFiltrePorLaMarcaKia</div> <div>[3:17] EPSEARCHB003_searchCarsByBrands_DebeDevolver3Carros_CuandoSeFiltrePorLaMarcaPorsche</div> <div>[4:24] EPSEARCHB004_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Kia</div> <div>[5:31] EPSEARCHB005_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Kia</div> <div>[6:38] EPSEARCHB006_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Kia</div> <div>[7:45] EPSEARCHB007_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Kia</div> <div>[8:52] EPSEARCHB008_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Kia</div> </div> </div>		<div> <div>Summary Tags Feature: com/test/automation/project/testautomationjavaservicebackend/api/karate/SearchByBrand.feature GET Search Cars By Brand</div> <div>Scenario: [1:3] EPSEARCHB001_searchCarsByBrands_DebeDevolver3Carros_CuandoSeFiltrePorLaMarcaBMW ms: 25</div> <div> <div>4 Given url baseUrl + '/api/searchBrands'</div> <div>5 And param brands = 'BMW'</div> <div>6 When method get</div> <div>7 Then status 200</div> <div>8 And match response == '#[3]'</div> </div> <div>Scenario: [2:10] EPSEARCHB002_searchCarsByBrands_DebeDevolver3Carros_CuandoSeFiltrePorLaMarcaKia ms: 27</div> <div> <div>11 Given url baseUrl + '/api/searchBrands'</div> <div>12 And param brands = 'Kia'</div> <div>13 When method get</div> <div>14 Then status 200</div> <div>15 And match response == '#[3]'</div> </div> <div>Scenario: [3:17] EPSEARCHB003_searchCarsByBrands_DebeDevolver3Carros_CuandoSeFiltrePorLaMarcaPorsche ms: 23</div> <div> <div>18 Given url baseUrl + '/api/searchBrands'</div> <div>19 And param brands = 'Porsche'</div> <div>20 When method get</div> <div>21 Then status 200</div> <div>22 And match response == '#[3]'</div> </div> <div>Scenario: [4:24] EPSEARCHB004_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Kia ms: 22</div> <div> <div>25 Given url baseUrl + '/api/searchBrands'</div> <div>26 And param brands = 'BMW,Kia'</div> <div>27 When method get</div> <div>28 Then status 200</div> <div>29 And match response == '#[6]'</div> </div> </div>
--	--	--

<div> <div>  <div> <div>9</div> <div>0</div> </div> </div> <div> <div>Scenarios</div> <div>2023-10-27 02:58:57 p. m.</div> <div> <div>[1-3] EPSEARCHB001_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Porche</div> <div>[2-10] EPSEARCHB002_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaKia_Y_Porche</div> <div>[3-17] EPSEARCHB003_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Kia_Y_Porche</div> <div>[4-24] EPSEARCHB004_searchCarsByBrands_DebeDevolver9Carros_CuandoNoSeEspecifiqueNingunaMarca_DadoQuePorDefectoBuscarLasTresDisponibles</div> <div>[5-31] EPSEARCHB005_searchCarsByBrands_DebeDevolver9Carros_CuandoNoSeEspecifiqueElParametroMarcas_DadoQuePorDefectoBuscarLasTresDisponibles</div> <div>[6-38] EPSEARCHB006_searchCarsByBrands_DebeDevolver9Carros_CuandoNoSeEspecifiqueElParametroMarcas_DadoQuePorDefectoBuscarLasTresDisponibles</div> <div>[7-45] EPSEARCHB007_searchCarsByBrands_DebeDevolver9Carros_CuandoNoSeEspecifiqueElParametroMarcas_DadoQuePorDefectoBuscarLasTresDisponibles</div> <div>[8-52] EPSEARCHB008_searchCarsByBrands_DebeDevolver9Carros_CuandoNoSeEspecifiqueElParametroMarcas_DadoQuePorDefectoBuscarLasTresDisponibles</div> <div>[9-59] EPSEARCHB009_searchCarsByBrands_DebeDevolver9Carros_CuandoNoSeEspecifiqueElParametroMarcas_DadoQuePorDefectoBuscarLasTresDisponibles</div> </div> </div> </div>	
29 And match response == '#[0]'	6
Scenario: [5:31] EPSEARCHB005_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Porche ms: 30	
32 Given url baseUrl = '/api/searchBrands'	2
33 And param brands = 'BMW,Porsche'	1
34 When method get	23
35 Then status 200	0
36 And match response == '#[0]'	5
Scenario: [6:38] EPSEARCHB006_searchCarsByBrands_DebeDevolver6Carros_CuandoSeFiltrePorLaMarcaKia_Y_Porche ms: 42	
39 Given url baseUrl = '/api/searchBrands'	1
40 And param brands = 'Kia,Porsche'	1
41 When method get	33
42 Then status 200	0
43 And match response == '#[0]'	8
Scenario: [7:45] EPSEARCHB007_searchCarsByBrands_DebeDevolver9Carros_CuandoSeFiltrePorLaMarcaBMW_Y_Kia_Y_Porche ms: 48	
46 Given url baseUrl = '/api/searchBrands'	2
47 And param brands = 'BMW,Kia,Porsche'	1
48 When method get	39
49 Then status 200	0
50 And match response == '#[0]'	6
Scenario: [8:52] EPSEARCHB008_searchCarsByBrands_DebeDevolver9Carros_CuandoNoSeEspecifiqueNingunaMarca_DadoQuePorDefectoBuscarLasTresDisponibles ms: 31	
53 Given url baseUrl = '/api/searchBrands'	2
54 And param brands = ''	1
55 When method get	26
56 Then status 200	0
57 And match response == '#[0]'	2
Scenario: [9:59] EPSEARCHB009_searchCarsByBrands_DebeDevolver9Carros_CuandoNoSeEspecifiqueElParametroMarcas_DadoQuePorDefectoBuscarLasTresDisponibles ms: 19	
60 Given url baseUrl = '/api/searchBrands'	1
61 When method get	15
62 Then status 200	0
63 And match response == '#[0]'	4

La última prueba con código **EPSEARCHP007** corresponde a una validación de un usuario que detectó que el backend no enviaba un mensaje de error Bad Request 400 cuando el finalPrice del filtro era menor al initialPrice y por ende se debió incluir la prueba correspondiente, y aplicar TDD para el manejo de esta excepción en el **CarsController**, mediante el manejo de un **IllegalArgumentException** manejada desde la clase **RestResponseStatusExceptionResolver**.