****

**Contenido**

[**Introducción**](#_xi99jfz9m49s) **3**

[Resumen de las actividades de prueba](#_936vdgwti2zs) 3

[**Alcance**](#_bib9o2z0f5h2) **3**

[Dentro del Alcance](#_d2gi7eibz1ld) 3

[Fuera de Alcance](#_iglz7ppc8kxg) 3

[Tipos de Pruebas Ejecutadas](#_u7scyhbq3kq) 3

[Enfoque de la Prueba](#_r235vqm2w1ua) 4

[**Exit Criteria**](#_2q0hncwdrrtr) **4**

[**Resumen de Resultados**](#_njtem23xxxoe) **5**

[Diseño de Pruebas](#_7f8rtzklrzfi) 5

[Ejecución de Pruebas](#_24r2icnnfb7w) 6

[Ejecución Manual](#_tt608je7becg) 6

[Ejecución Automática](#_natknhjepgt) 6

[Reporte de Defectos](#_sii7y4y0hsw8) 6

[Todos los defectos](#_5384b1vu9cc5) 6

[Defectos por prioridad](#_q5uq6lnchdig) 7

[Defectos por Severidad](#_kw38y6o4hfld) 7

[Defecto por Estado](#_p77l9wifit) 7

[Defectos Creados vs Resueltos](#_u0mx51xzw54u) 8

[Defectos Abiertos](#_3u007c6wxpl0) 9

[**Lecciones Aprendidas / Conclusión**](#_q7ef8uwmmty2) **9**

# Introducción

Este documento es el Informe Final de Pruebas del sistema Rent Car Company. El propósito de este documento es proveer evidencia de que el Exit Criteria para el proceso de Testing se cumplió y por lo tanto, se concluye la fase de pruebas y puede cerrarse. Se demuestra que los Issues de GitLab relacionados con testing fueron implementados desde los Sprint 1 a 4. Este documento va a ser utilizado como entrada para la revisión general de las actividades de prueba y para tomar la decisión si el sistema cumple con las expectativas.

## Resumen de las actividades de prueba

Rent Car Company es un sitio de reservas de autos. En esta web podrá registrar usuarios, con diferentes roles. Un rol define el conjunto de privilegios asignados a un miembro. Los privilegios se asignan a los miembros mediante un rol predeterminado o un rol personalizado.

La implementación del rol administrador en la página web, es aquel que asegura que la información del sitio web es correcta, segura y está actualizada. Tiene funcionalidades sobre la de los usuarios comunes.

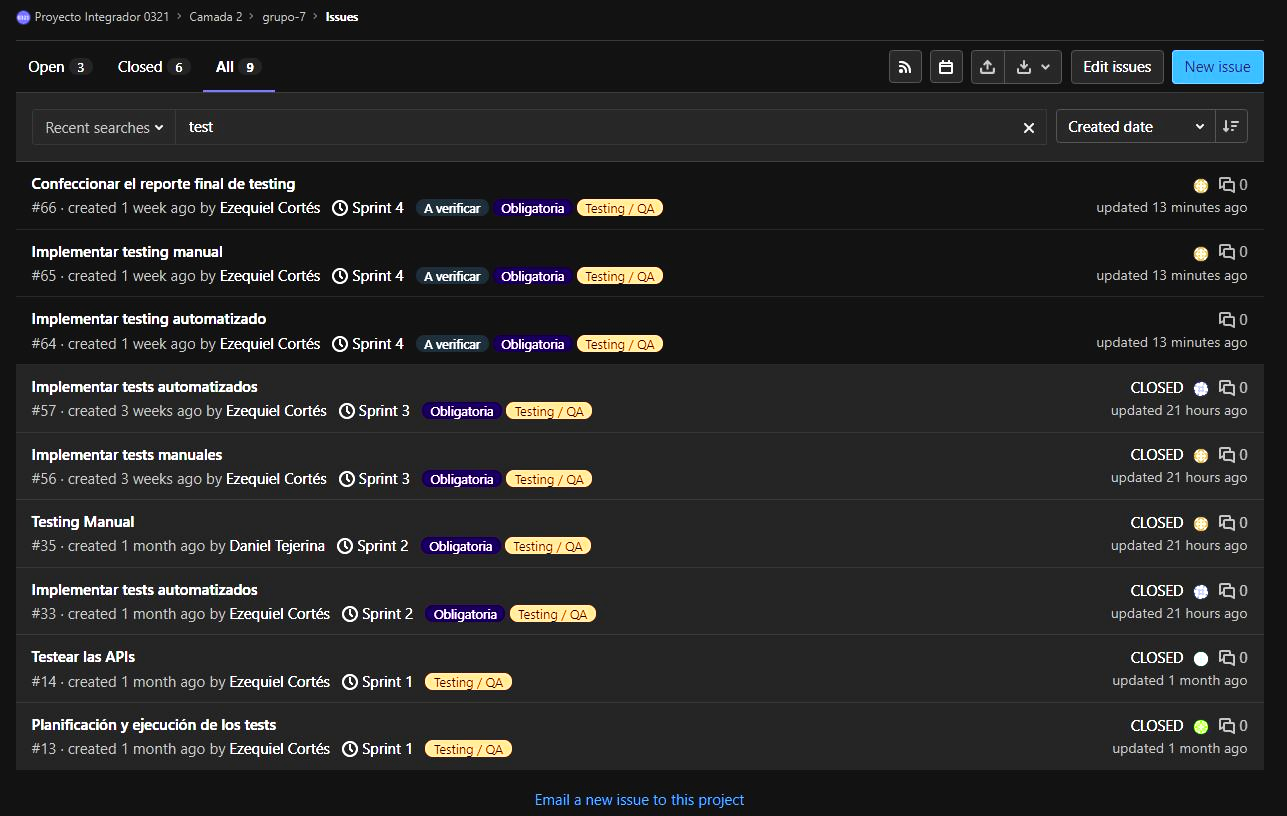
Al ingresar en Rent Car Company, podrá visualizar una página estéticamente armoniosa,

donde los productos disponibles se encuentran divididos en categorías interactivas.

Tendrá la opción de elegir sus reservas por fechas, y visualizar las que están ocupadas.

La página consta de una primera visualización donde mostramos las categoría de autos más solicitados, y una galería de productos, los cuales tienen sus especificaciones y utilidades.Estas especificaciones constan de una galería lateral, de fotos relacionadas al producto, así como también la posibilidad de imaginar y visualizar el auto elegido en el color deseado.

*URL para acceder al mismo (IP de AWS): http://3.145.62.164/#car-options*

**

# 

# Alcance

# Dentro del Alcance

Las funcionalidades que fueron probadas fueron tales como , implementación de componentes responsive, bloque de categorías, login y registro, bloque de listado , de un buscador, evento del login usuario y header, y otras muchas funcionalidades que están dispuestas es historias de usuarios en la tabla siguiente:

**

## Fuera de Alcance

Todas las funcionalidades fueron probadas, si bien no fueron documentadas algunas, como los responsive de varios tipos de visualizaciones.

Los mapas fueron probados y documentados, si bien no estaban en las historias de usuario.

Algunos botones no fueron probados porque había historias de usuario probando funcionalidades de la redirección del botón, si que no la incluimos porque era parte del camino.

## Tipos de Pruebas Ejecutadas

|  | *Sprint 1* | *Sprint 2* | *Sprint 3* | *Sprint 4* |
| --- | --- | --- | --- | --- |
| *Prueba Estática* | *NO* | *NO* | *SI* | *NO* |
| *Prueba Exploratoria* | *SI* | *SI* | *SI* | *SI* |
| *Prueba de Humo* | *SI* | *SI* | *NO* | *NO* |
| *Prueba de Regresión* | *SI* | *SI* | *SI* | *SI* |
| *Prueba de Componente / Unidad* | *SI* | *SI* | *SI* | *SI* |
| *Prueba de Integración (Postman)* | *SI* | *SI* | *NO* | *SI* |

## Enfoque de la Prueba

Se crearon test funcionales mayormente positivos, pero se tuvieron en cuenta varios caminos negativos, como por ejemplo para los mensajes de error.

Cuando se subían las nuevas issues, se leían en detalles todas las historias y sus descripciones, de ahí se sacaban las funcionalidades y componentes a probar.

Luego de eso se ejecutaban las pruebas de humo y las nuevas funcionalidades.

Si estas no pasaban, o no llegaban al camino del éxito, luego de seguir los pasos especificados, se reportaba su fallo y pasan a la planilla de defecto.

Llegando al final del sprint se ejecutan las pruebas exploratorias, en base a las issues y todos los defectos encontrados se cargaran sobre la misma planilla de defectos.

Las pruebas de unidad se ejecutaban cada vez que se generaba un nuevo build del sistema.

*En el último sprint se ejecutó la suite de pruebas de regresión completa y se reportaron los defectos encontrados en la planilla de defectos.*

**Link Planilla de Casos de Prueba: https://docs.google.com/spreadsheets/d/1IkXB-uvu5hYAIRC\_fSGfeiKRRP2SYVKm/edit#gid=11517761**

**Link Planilla de Defectos:https://docs.google.com/spreadsheets/d/1IkXB-uvu5hYAIRC\_fSGfeiKRRP2SYVKm/edit#gid=2144479876**

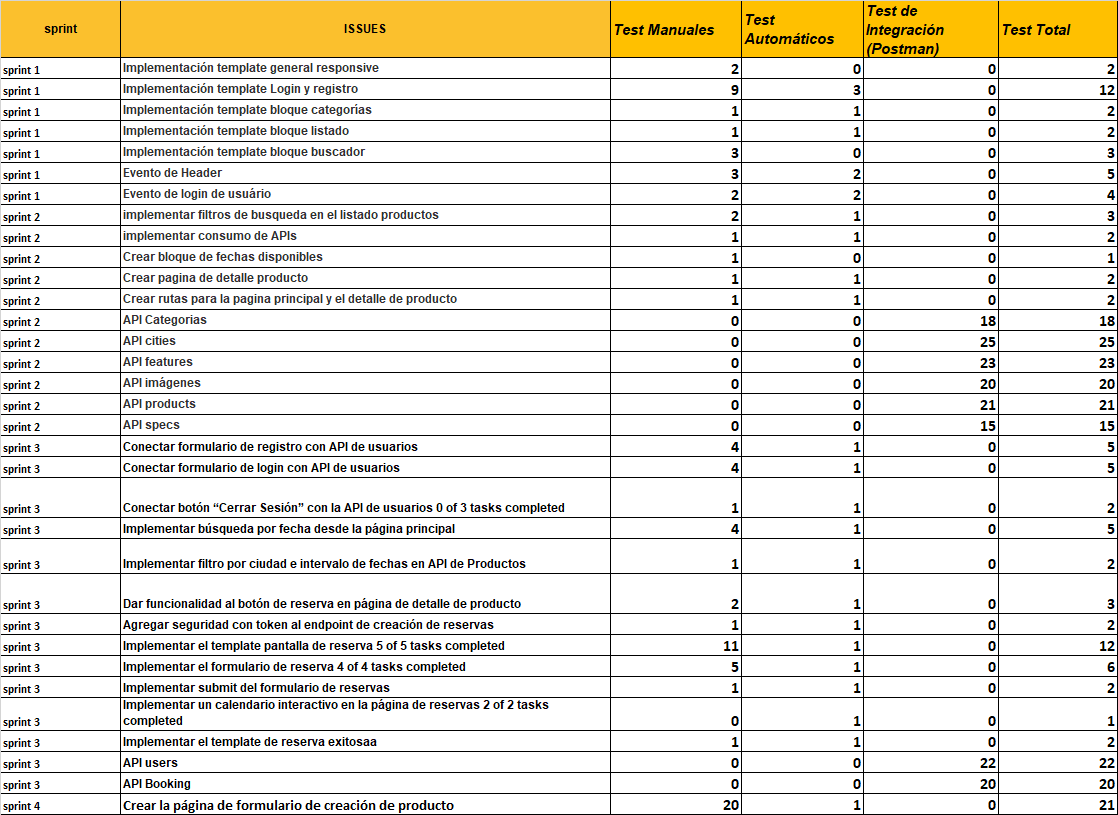
# Exit Criteria

Se definió los siguientes criterios de aceptación para finalizar las pruebas:

* No se debe tener defectos en estado abierto de severidad crítica y/o bloqueante.
* La ejecución de las pruebas de unidad deben tener un 90% de tasa de estados pasados.
* La ejecución de test Postman deben tener un 90% de past rate,
* *La ejecución de los test de regresión debe tener un 90% de pass rate*
* *La ejecución de los test Jest deben tener un 80% de pas rate,*

# Resumen de Resultados

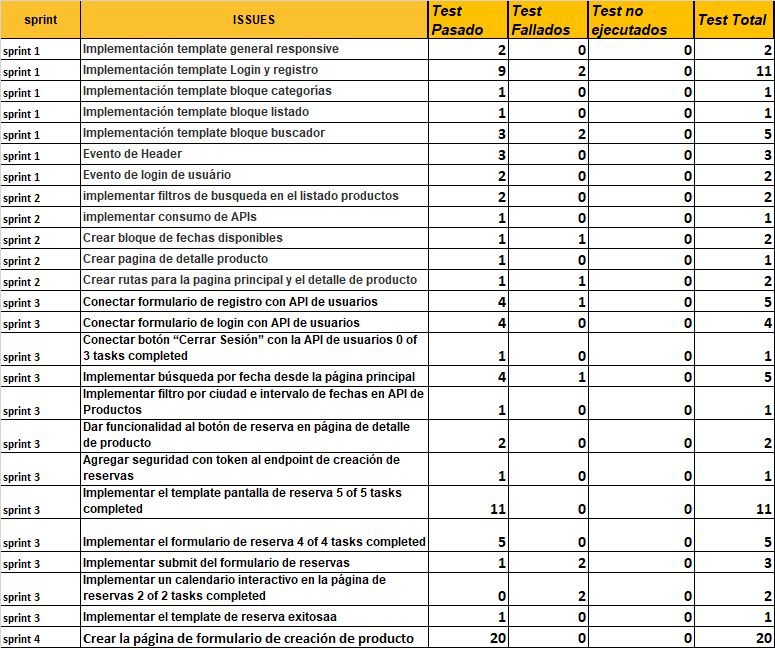
## Diseño de Pruebas

**

***link:****https://docs.google.com/spreadsheets/d/193oIG1N\_6sAoUyrAUO\_ar83dHFMmR\_pdafiUqGiDkWA/edit?usp=sharing*

## Ejecución de Pruebas

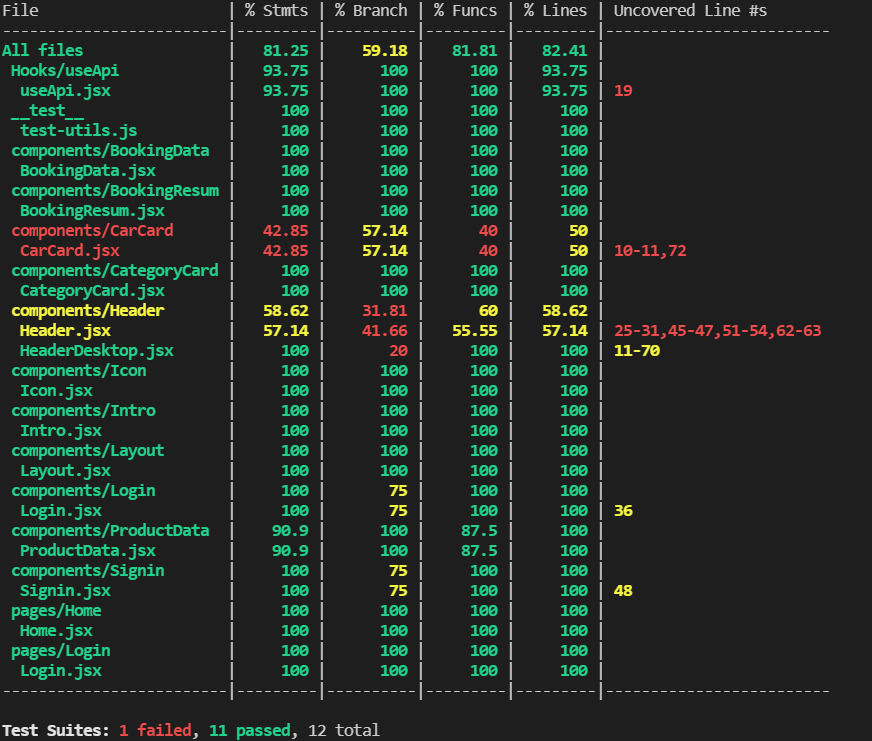
### Ejecución Manual



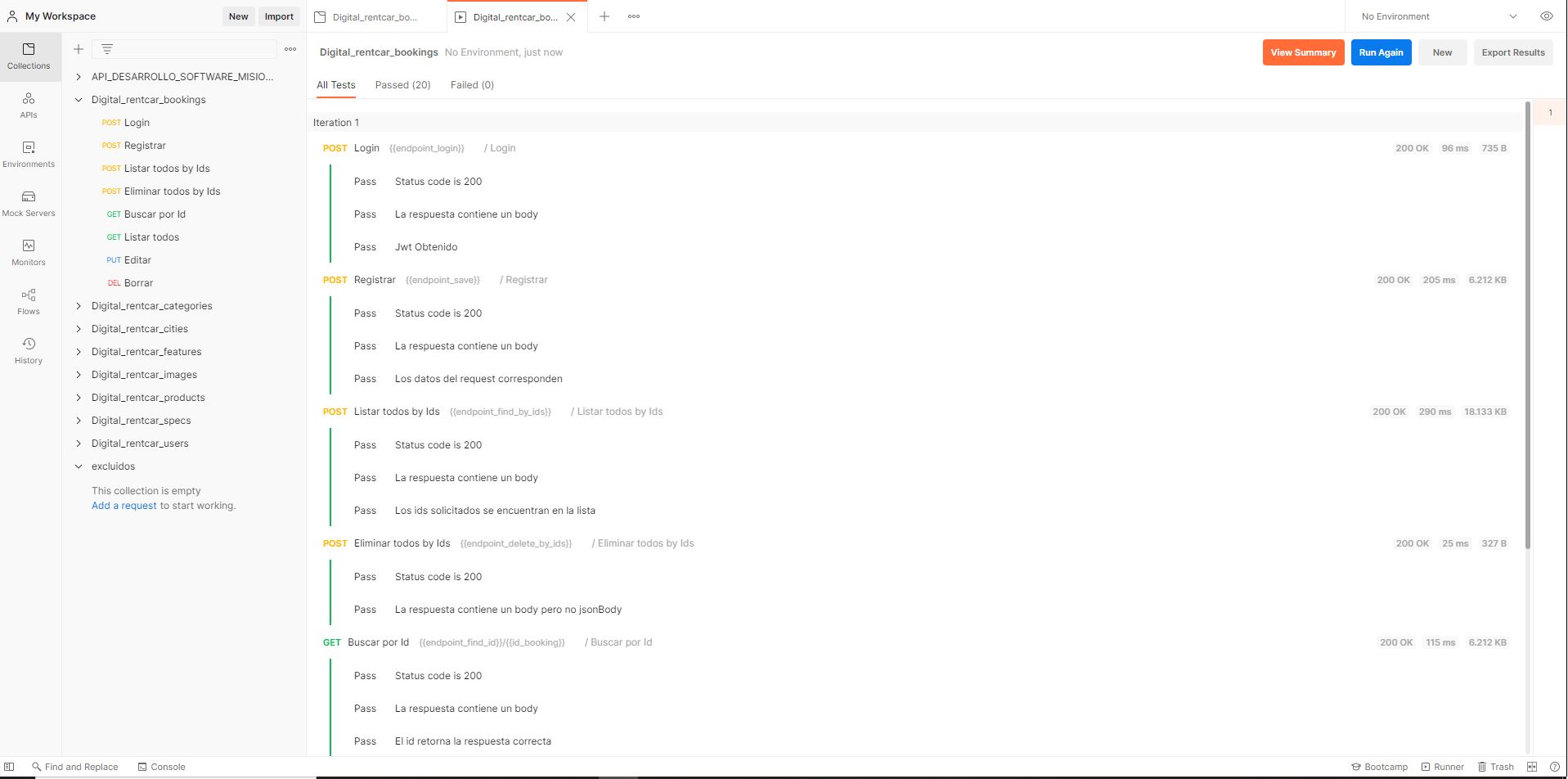
### link:https://docs.google.com/spreadsheets/d/1o0JeRAFs1GGuOpkLdioEGBlkHCJovN8yErVVBLVi7CU/edit?usp=sharing

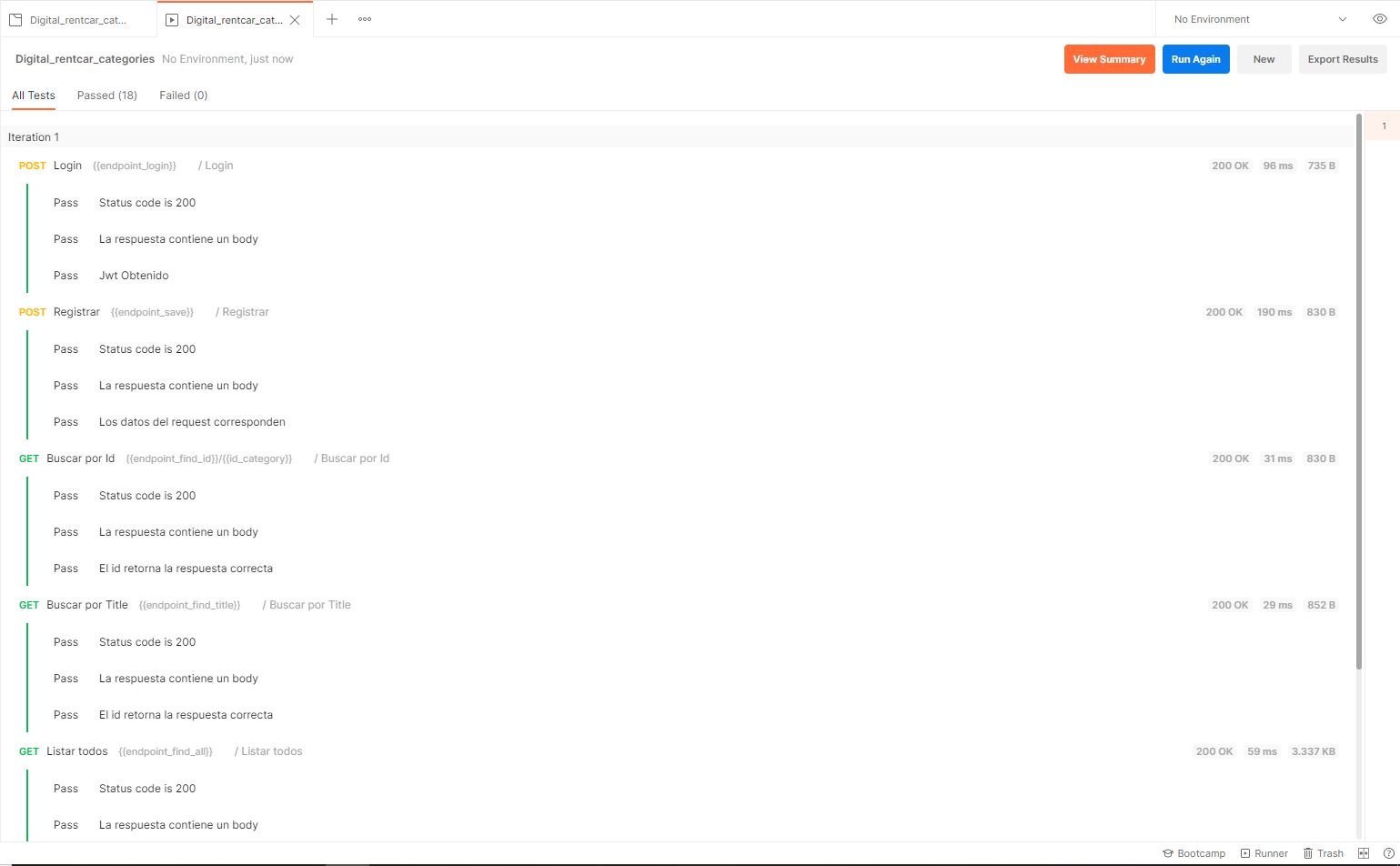
### 

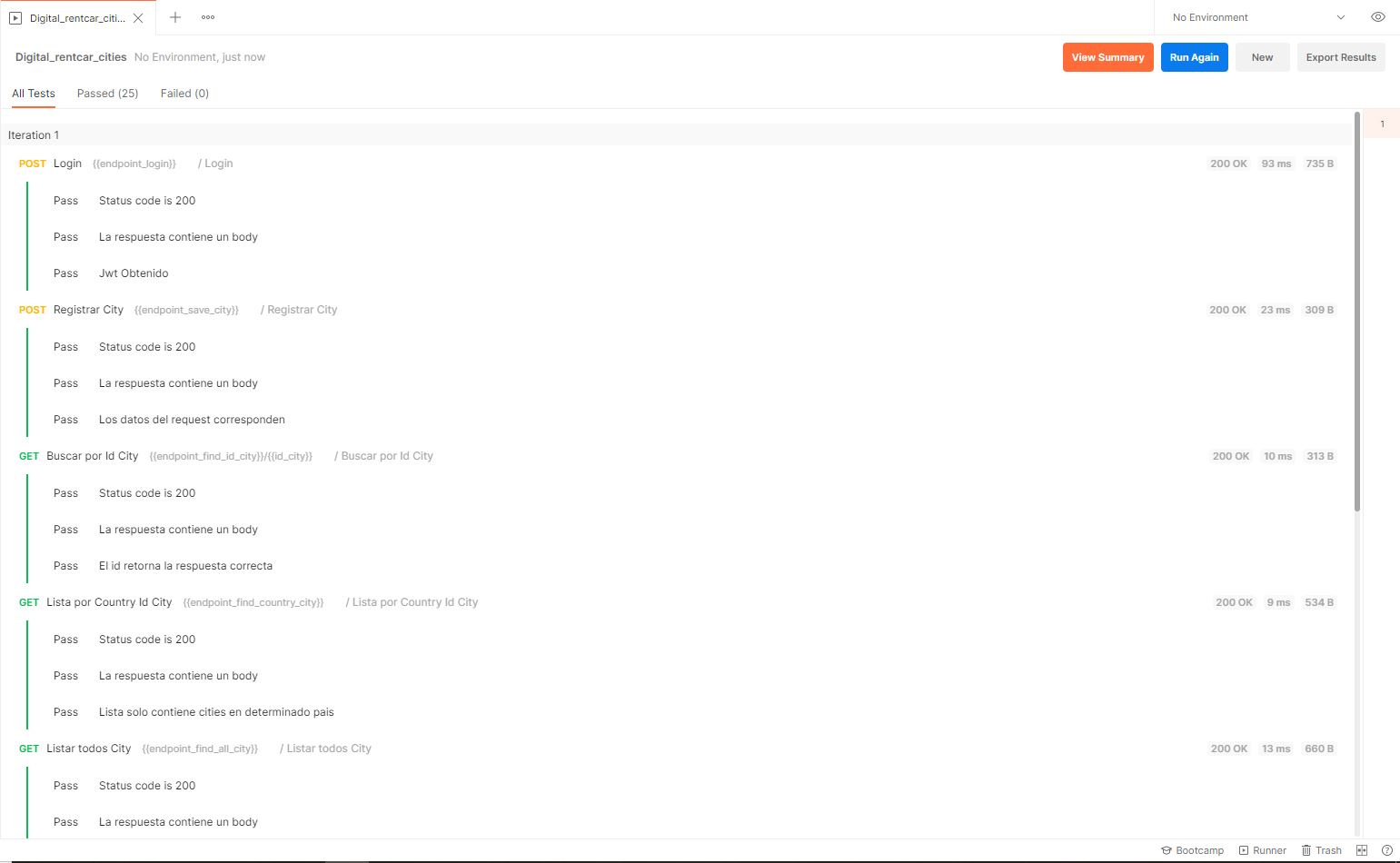
### Ejecución Automática

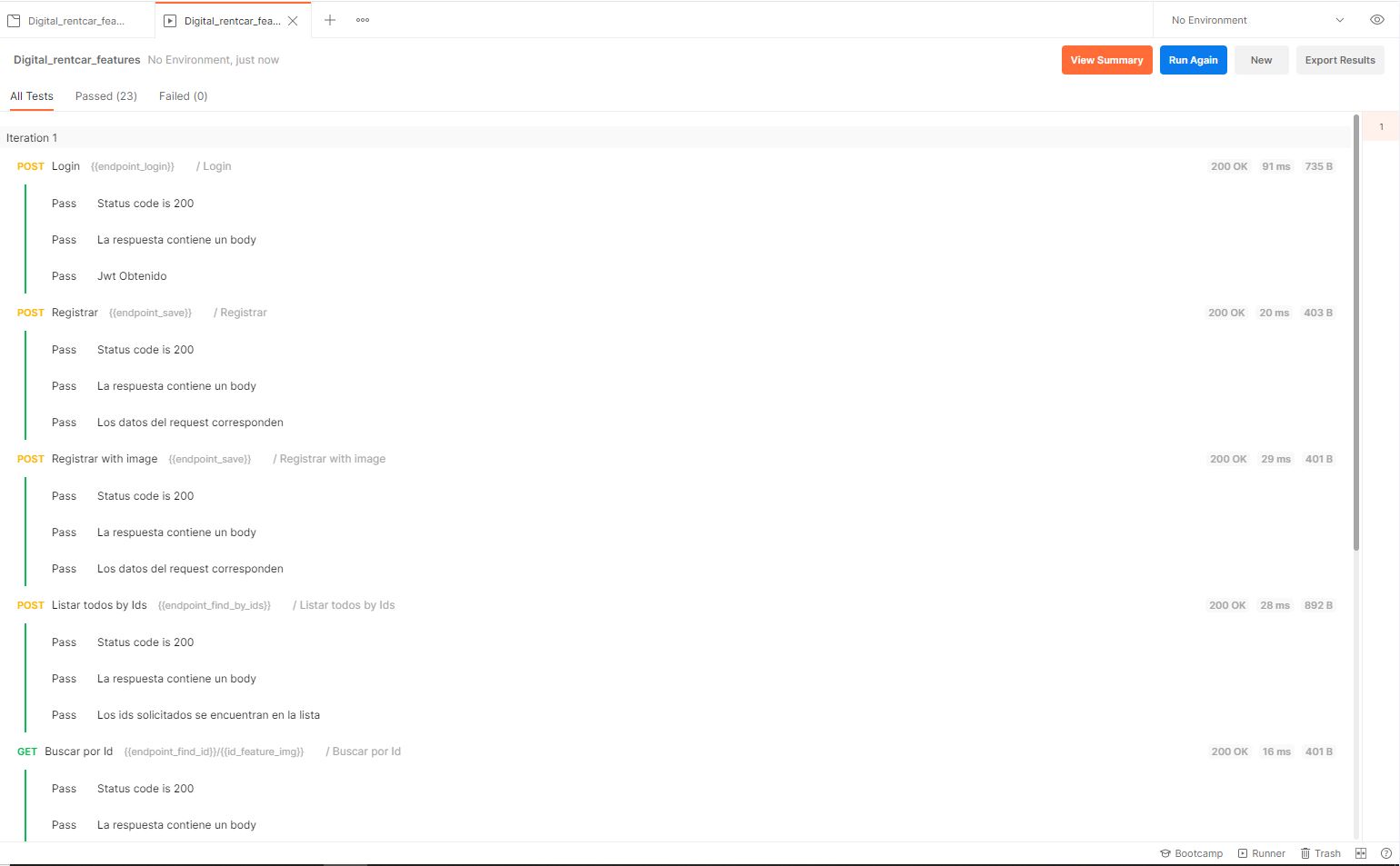


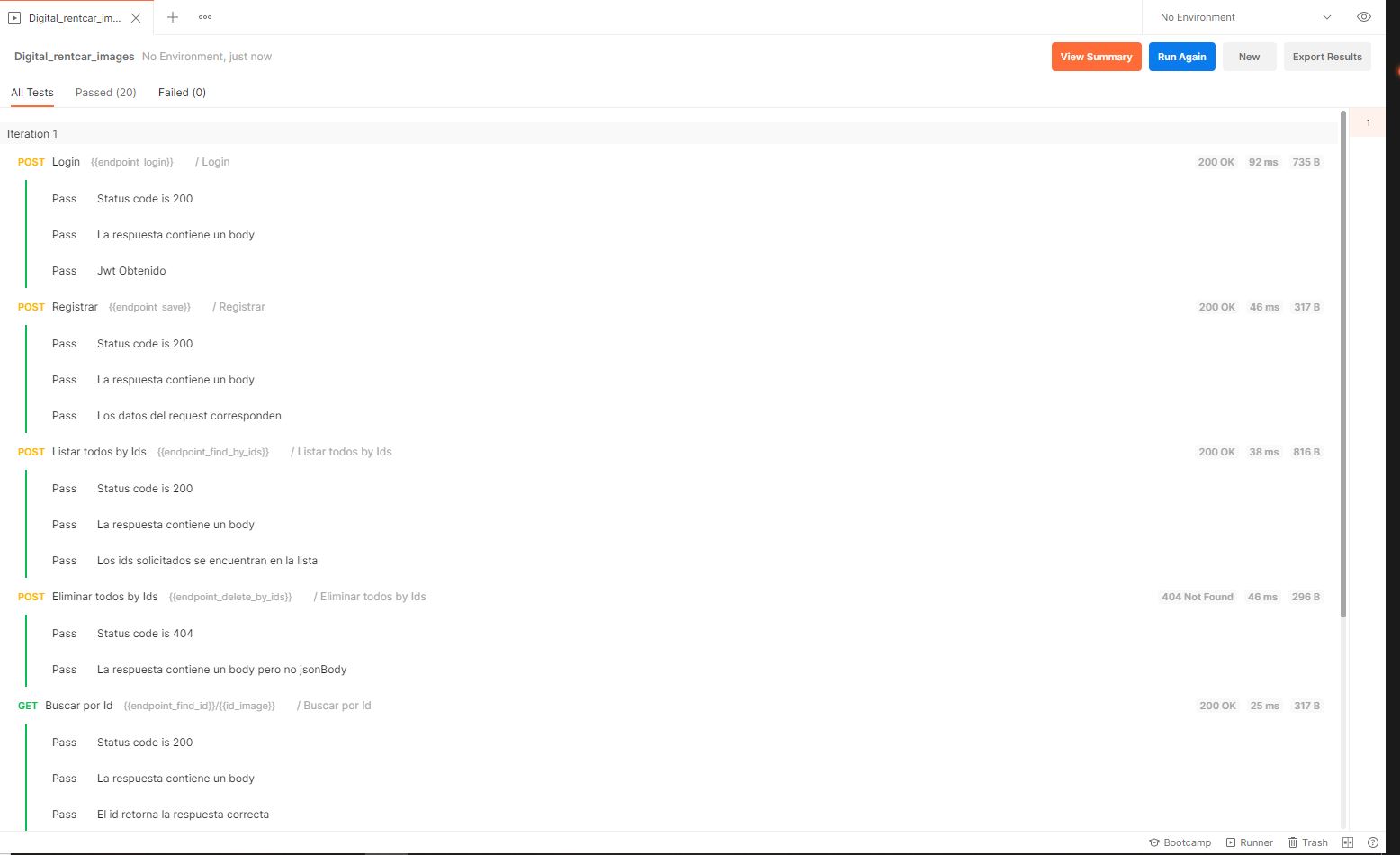
api\_booking

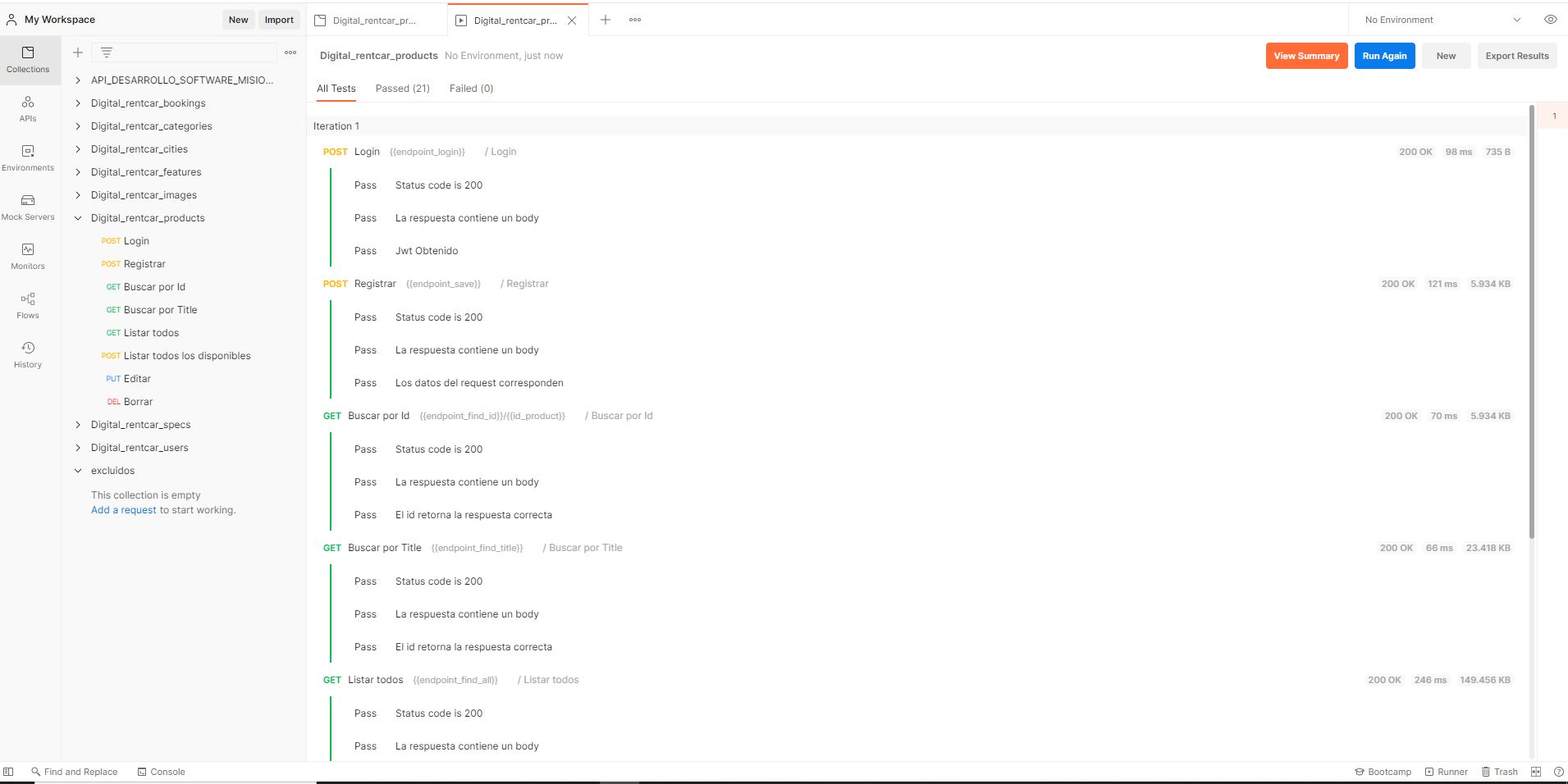


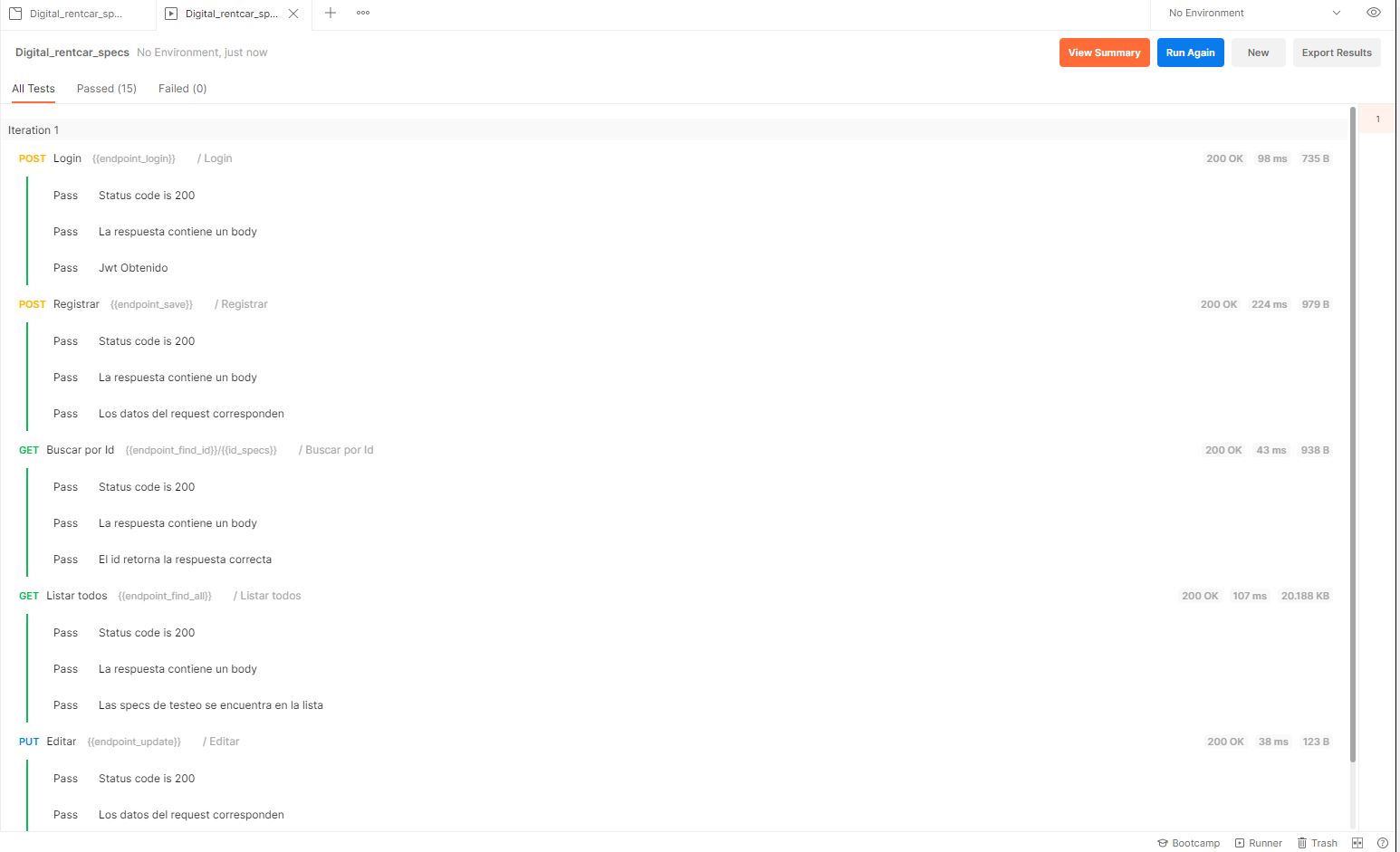
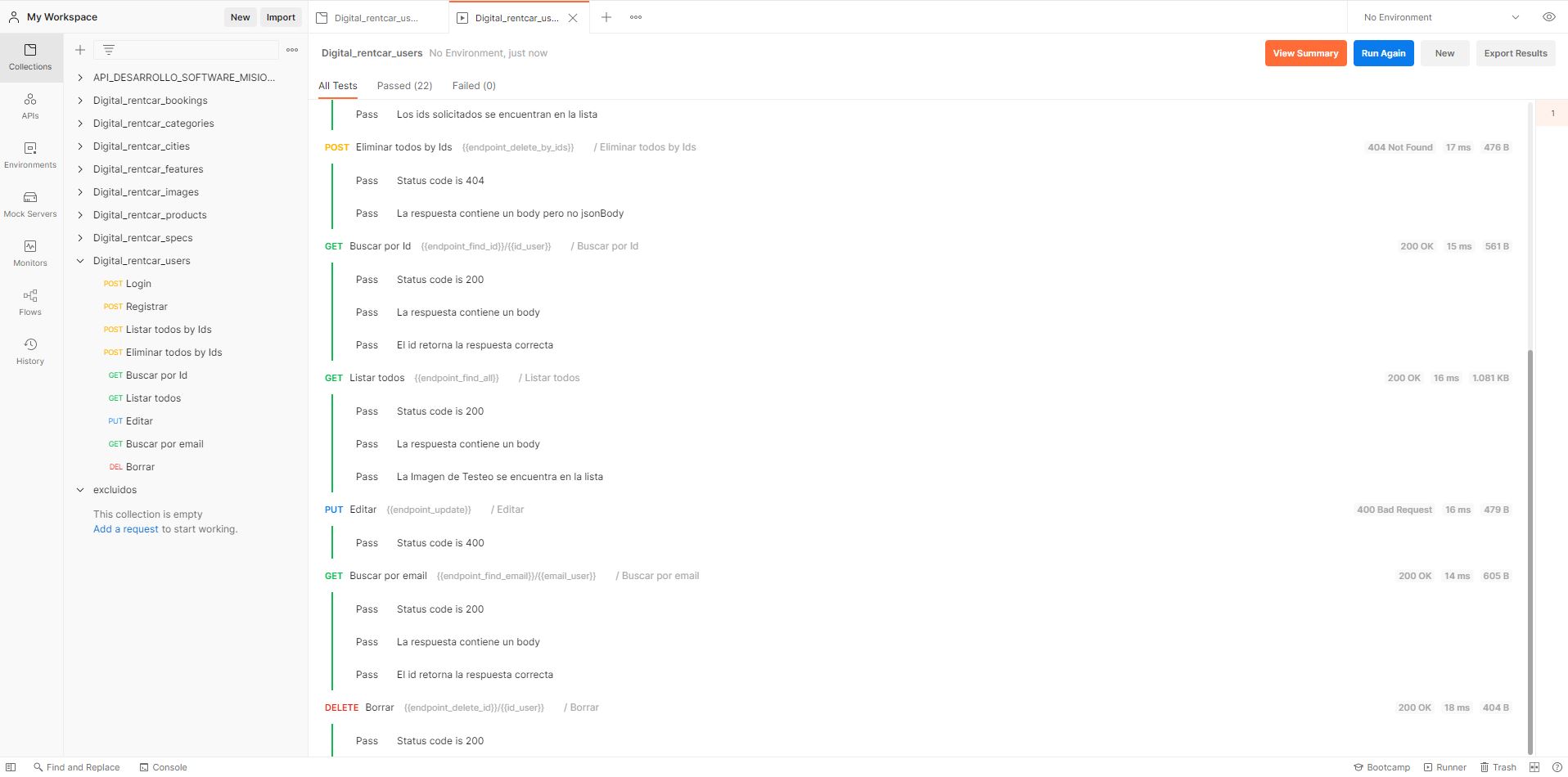
api\_categorias

api\_cities

api\_features

api\_imagenes

api\_products

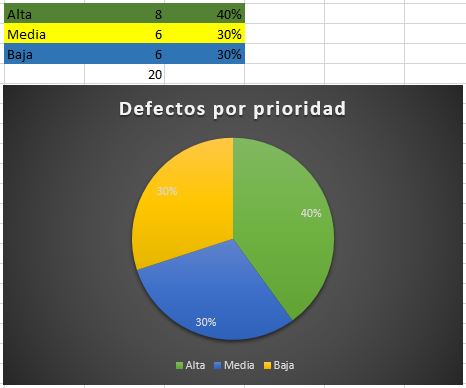
api\_specsapi\_users

## Reporte de Defectos

## Todos los defectos

La siguiente sección muestra información con respecto al número total de defectos que se han presentado durante la duración de la fase de prueba.

### Defectos por prioridad

**

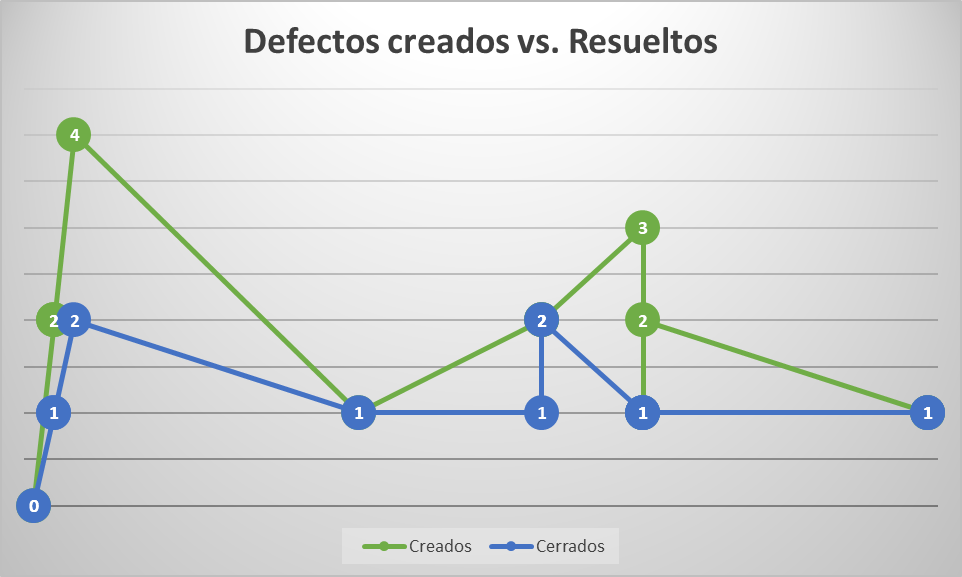
### Defectos por Severidad

**

### Defecto por Estado



### Defectos Creados vs Resueltos



# Lecciones Aprendidas

***QA*** *(Quality Assurance)*

*Pruebas Funcionales: pruebas que evalúan las funciones que el sistema debe realizar. Requisitos, user stories, etc.*

*Pruebas de Casos de Uso: emulan el uso del sistema.*

*Pruebas de Exactitud: abarcan la exactitud de cálculo.*

*Pruebas de Adecuación: implica validar la eficiencia de un conjunto de funciones para la consecución de tareas previstas.*

*Pruebas de Sistema: se centra en el comportamiento del sistema en su totalidad, teniendo en cuenta tareas extremo a extremo.*

*Pruebas de Regresión: detectan efectos secundarios no deseados en el software, luego de haber realizado cambios o agregado funcionalidades nuevas al código.*

*Pruebas de Confirmación: consiste en volver a ejecutar los pasos para volver a reproducir el fallo causado por un defecto una vez corregido dicho defecto, para verificar si el defecto era la causa del fallo o no.*

**CASOS DE PRUEBA**

es un documento escrito que proporciona información escrita sobre qué y cómo probar. Debe tener un **identificador**, un **nombre conciso** (conviene establecer una nomenclatura), una **descripción** (qué se va a probar, cual es el ambiente de prueba y los datos), una **precondición** (asunción que debe cumplirse antes de ejecutar), unos **pasos** y unos **resultados esperados**. Conviene que sea simple y con un título fuerte.

* No asumir funcionalidades y características.
* Asegurar la mayor cobertura de requisitos posible.
* Autonomía (mismos resultados siempre).
* Evitar la repetición de casos de prueba.
* Tener en cuenta al usuario final.

Testing Positivo: son aquellos casos de prueba que validan el flujo normal de un sistema bajo prueba. Es decir, flujos que están relacionados a los requisitos funcionales.

Testing Negativo: validan flujos no contemplados dentro de los requisitos del sistema. Es intentar romper todo (poner letras en un DNI y tendría que dar un error. El sistema tendría que avisar que el caracter no es válido (entonces pasa la prueba, porque es lo que debería hacer)).

**CASO DE USO**

Un caso de uso cuenta la historia de cómo un usuario interactúa con un software para lograr un objetivo (las rutas que sigue el usuario). Se redactan en infinitivo.

Happy Path Testing: es el "camino único" con el que se prueba una aplicación, sigue el mismo flujo que realiza un usuario final de manera regular. Está destinado a ver que un producto funcione como ha sido diseñado, no para encontrar defectos. Es el que se quiere usar para llegar a la felicidad del cliente. Si algo tiene que funcionar, es el happy path.

**Ejecución de los Casos de Prueba**

PASOS para ejecutar un caso de prueba. Pueden modificarse pero se recomienda el siguiente orden::

1. Registrar los identificadores y las versiones de los elementos de prueba, las herramientas de prueba y los productos de prueba.
2. Ejecutar pruebas (manual o automation).
3. Comparar resultados reales con resultados esperados.
4. Analizar anomalías para establecer sus causas probables.
5. Informar sobre los defectos en función de lo observado.
6. Registrar el resultado de la ejecución de la prueba.
7. Repetir las actividades de prueba, ya sea como resultado de una acción tomada para una anomalía o como parte de la prueba planificada.

***CICLO DE VIDA*** *de un Defecto:*

1. *Reportado/Inicial: se registra el defecto.*
2. *Asignado: se asigna al equipo de desarrollo o:* 
   * *Duplicado: si es repetido o misma causa raíz.*
   * *Rechazado: se solicita más información.*
   * *Diferido: no es prioritario y se solucionará luego.*
3. *En progreso: se analiza y trabaja en la solución.*
4. *Corregido: se realizan los cambios de código.*
5. *En espera de verificación: se le debe asignar un probador.*
6. *En Verificación: se hace una prueba de confirmación.*
7. *Verificado: se obtiene el resultado esperado en la prueba.*
8. *Cerrado: el defecto fue corregido.*

*Los Defectos son reportados (por el usuario/cliente)*

*Escribir un buen INFORME:*

* *Los bugs deben tener identificadores únicos.*
* *Una falla debe ser reproducible para reportarla.*
* *Ser específico.*
* *Reportar cada paso realizado para reproducirlo.*

*Debe ser suficiente información para la:*

1. *Gestión del informe durante el ciclo de vida del defecto.*
2. *Evaluación del estado del proyecto (progreso y calidad).*
3. *Evaluación de la capacidad del proceso.*

*La prioridad la selecciona el cliente, y la severidad depende del negocio (si en un ecommerce no puedo usar el carrito de compras, es de severidad crítica).*

***Prueba de Componente***

*El objetivo principal es aislar cada unidad del sistema para identificar, analizar y corregir los defectos en una fase temprana. Cubre tanto la funcionalidad como las características no funcionales.*

*Permite una mejor refactorización del código. Ahorran tiempo, esfuerzo y dinero.*

*Pueden ser:*

* *Manuales: se emplea un documento instructivo paso a paso.*
* *Automatizadas: se necesita de un framework para escribir los scripts de prueba. Se necesita un test runner que ejecute los test (Mocha) y una assertion library que valide la lógica de prueba (Chai).* 
  + *Los frameworks más utilizados son: Junit, Nunit, Jmockit, EMMA, PHPUnit, JEST.*

*Técnicas de prueba de caja blanca:*

*Se basan en la estructura interna (código fuente) del objeto de prueba.*

* *Prueba y cobertura de sentencia: ejercita las sentencias ejecutables, exponiendo código que nunca se ejecuta. Todas las sentencias fueron probadas al menos una vez, pero no asegura la prueba de toda la lógica de decisión.*
* *Prueba y cobertura de decisión: ejercita las decisiones en el código y los casos de prueba siguen los flujos de control que se producen desde un punto de decisión.*

*Mock: objetos preprogramados con expectativas que conforman la especificación de cómo se espera que se reciban las llamadas. Podemos definir expectativas detalladas, y validar el comportamiento en la colaboración.*

*Stub: proporcionan respuestas predefinidas a llamadas hechas durante los tests, sin responder en absoluto a cualquier otra cosa fuera de aquello para lo que ha sido programado. Tan solo devuelve respuestas preprogramadas a posibles llamadas, y simula respuestas a consultas.*

***Postman***

*Postman: es un software para desarrollar APIs y manejar sus consultas y respuestas. Permite automatizar pruebas y es útil para API testing.*

*Lo vamos a usar en los ejemplos.*

*Postman Tests: permite asegurarnos de que un API funciona como se esperaba. Nos permite establecer que las integraciones entre los servicios funcionen de manera confiable y verificar que los nuevos desarrollos no hayan roto ninguna funcionalidad existente. Nos ayuda a verificar resultados, como el estado exitoso o fallido, la comparación de los resultados esperados, etc. Usa JavaScript.*

# Conclusión

En fin, llegamos a la conclusión que en todo proyecto de desarrollo, es sumamente importante testear, diseñar los casos y ejecutar cada una de las funcionalidades y componentes del proyecto.

A partir de esto es posible que nuestro proyecto vaya subiendo su calidad y eficiencia, si bien como el principio de testing lo afirma, no está exento de errores, es posible minimizar lo más, que sea posible.

Es sumamente importante llevar un registro detallado de todas las pruebas realizadas, para que en cada sprint resulte más fácil y llevadero. Así como hacer las ejecuciones de las pruebas lo antes posible para evitar daños y modificaciones más costosas, sobre todo por el tiempo.