



MÓDULO PROYECTO

CFGS Desarrollo de Aplicaciones
Multiplataforma
Informática y Comunicaciones

Asegurar la calidad

Tutor individual: Juan Antonio Alonso Velasco

Tutor colectivo: Cristina Silvan Pardo

Año: 2024

Fecha de presentación: 28//05/2024

**Nombre y Apellidos: Álvaro Collantes
Hierro**

Email: alvaro.colhie@educa.jcyl.es



Índice

1	Introducción.....	6
2	Metodología	7
3	Requisitos del software	9
3.1	Identificación de requisitos	9
4	Plan de pruebas	10
4.1	Objetivos de las pruebas.....	10
4.2	Estrategia de pruebas.....	11
4.3	Casos de prueba.....	12
4.4	Herramientas de prueba.....	20
5	Desarrollo de las pruebas.....	20
5.1	Implementación de pruebas.....	23
5.2	Automatización de pruebas.....	32
6	Gestión de errores	34
6.1	Reporte de errores	34
7	Corrección de errores y validación final	39
7.1	Corrección de errores	40
7.2	Pruebas de regresión.....	41
7.3	Validación final	44
8	Conclusiones y posibles ampliaciones	45
9	Bibliografía	47

A María Hidalgo y Rosa Guerrero, dos de mis formadoras en Eviden.

Gracias por su constante apoyo y por estar siempre disponibles para ayudarme cuando lo he necesitado durante la realización de este trabajo.

Gracias también a mi familia y mi novia Silvia por su respaldo y por estar siempre
ahí para mí.

1 Introducción

El principal objetivo de este proyecto es documentar y analizar el proceso completo por el que pasa un software. En este caso el ejemplo será a través de una API de Spring boot, desde la identificación de los requisitos iniciales hasta la fase final de puesta en producción.

Con este proyecto se pretende dar una visión integral y detallada del ciclo de vida del desarrollo de software. El trabajo se centrará en las herramientas y metodologías utilizadas en cada proceso con el fin de asegurar un producto final de alta calidad y eficiencia.

Este proyecto no solo busca ilustrar el proceso técnico, sino también destacar la importancia de una buena gestión de proyectos de software, la colaboración entre equipos de desarrollo y QA (Quality Assurance) y la necesidad de una documentación precisa y detallada.

Para lograr aplicaciones de software robustas y eficientes, es prioritario comprender y documentar el proceso de desarrollo. La rápida evolución de la tecnología y la alta demanda de software de calidad requieren que este proceso esté bien estructurado y definido para garantizar el buen desempeño del software desde la primera fase hasta la fase final de la implementación. El siguiente proyecto subrayará la importancia de todas las fases del ciclo de vida del desarrollo de software y cómo la buena práctica en cada fase es vital para entregar un sistema exitoso. El proyecto también se encargará de la colaboración entre diferentes equipos, cómo se deben manejar los errores y cómo se deben realizar pruebas extensas para garantizar la estabilidad y la función del software.

El alcance de este proyecto se centrará en el desarrollo de una API específica utilizando Spring Boot. Se abordarán las siguientes etapas:

- **Identificación y Documentación de Requisitos:** Descripción de cómo se identificaron y documentaron los requisitos funcionales y no funcionales de la API.
- **Planificación y Ejecución de Pruebas:** Elaboración de un plan de pruebas detallado para asegurar que todas las funcionalidades de la API responden como se espera.
- **Desarrollo de Pruebas utilizando Postman:** Implementación de las pruebas usando Postman, incluyendo la configuración de variables y peticiones, así como la validación de respuestas mediante JavaScript.
- **Gestión de Errores:** Proceso de documentación y resolución de errores encontrados durante la fase de pruebas.
- **Pruebas de regresión y validación final:** Ejecución de pruebas después de la corrección de errores y validación final del software antes de su puesta en producción.

El proyecto no incluirá aspectos de implementación de características adicionales postproducción ni la integración con otros sistemas externos. Se centrará únicamente en los procesos internos necesarios para llevar la API a un estado de producción estable y funcional.

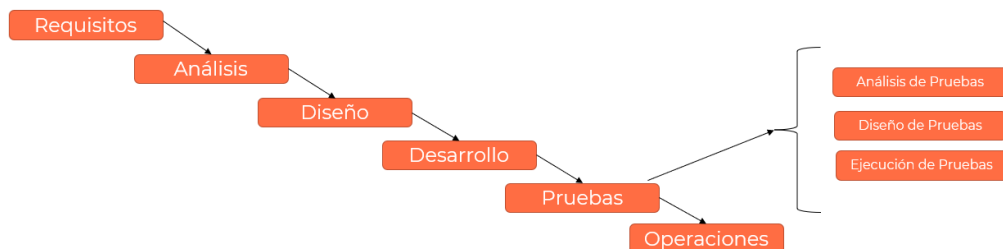
2 Metodología

En el desarrollo de software la elección de una metodología adecuada es crucial para garantizar el éxito del proceso. Las metodologías de gestión nos brindan un marco estructurado que guía el desarrollo de inicio a fin del proyecto.

Para desarrollar este proceso se escoge una metodología waterfall para tener un enfoque estructurado y secuencial que permita una planificación y ejecución que garantice que cada fase del proyecto se complete antes de que se comience con la siguiente. Este enfoque es especialmente aprovechable en proyectos con los

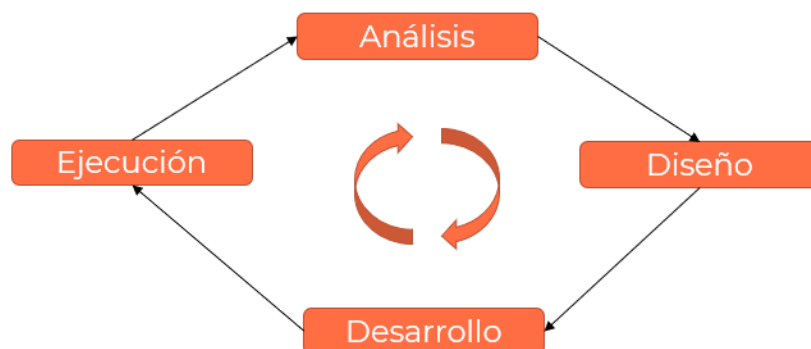
requisitos bien definidos y con baja probabilidad de que haya algún cambio. Primero vamos a ver brevemente qué tipos de metodología manejábamos para llevar a cabo este proyecto.

- **Metodología Waterfall:** es un enfoque secuencial y lineal para el desarrollo de proyectos. Este modelo se caracteriza por la ejecución de fases bien definidas y ordenadas, donde cada etapa debe completarse antes de avanzar a la siguiente.



Fuente: elaboración propia

- **Metodología Agile:** es un enfoque iterativo e incremental para la gestión de proyectos, especialmente popular en el desarrollo de software. Agile se basa en la colaboración continua, la flexibilidad y la entrega frecuente de incrementos funcionales del producto.



Fuente: elaboración propia

- **Metodología mixta (o híbrida):** combina elementos de las metodologías Waterfall y Agile, buscando aprovechar las ventajas de ambas. Este enfoque

es particularmente útil en proyectos que requieren una planificación inicial rigurosa y, al mismo tiempo, necesitan adaptarse a cambios durante su ejecución.

En este caso, la elección fue clara, ya que partíamos de un software previamente desarrollado para un proyecto de clase. La tarea consistía en extraer los requisitos que ese software demandaba y elaborar un plan de pruebas. Este plan tenía como objetivo asegurar que, una vez superadas todas las pruebas, podríamos garantizar la calidad del software. Además, una parte importante del proyecto fue ir pasando por todas las fases del desarrollo y evaluar cómo se iban superando, asegurando así un proceso controlado y exitoso.

3 Requisitos del software

En esta sección se detallan los requisitos necesarios para el correcto funcionamiento del software. Estos requisitos se han identificado y documentado para garantizar que el sistema cumpla con las necesidades y expectativas del proyecto. A continuación, se describen los requisitos específicos del software, seguidos de una documentación exhaustiva de cada uno de ellos.

3.1 Identificación de requisitos

-Búsqueda de hotel por localidad: El sistema debe permitir la búsqueda de hoteles por localidad, devolviendo una lista de hoteles según la localidad indicada.

-Búsqueda de hotel por categoría: El sistema debe permitir la búsqueda de hoteles por categoría, devolviendo una lista de hoteles según la categoría indicada. El atributo categoría es un entero que indica las estrellas (1-5) del hotel.

-Registro de un nuevo hotel: El sistema debe permitir el registro de un nuevo hotel. Los datos del hotel se deben pasar en formato JSON. La clave primaria del hotel será gestionada automáticamente por la base de datos mediante autoincremento, permitiendo que la clave primaria se incluya o no en los datos proporcionados.

- Eliminación de una habitación determinada de un hotel: El sistema debe permitir la eliminación de una habitación específica de un hotel, para lo cual se debe proporcionar el ID de la habitación en la URL.
- Modificación del estado de ocupación de una habitación: El sistema debe permitir modificar el estado de una habitación para indicar que está ocupada. Esto se debe realizar pasando el ID de la habitación y cambiando el valor booleano que representa el estado de ocupación.
- Registro de una nueva habitación en un hotel: El sistema debe permitir el registro de una nueva habitación en un hotel, para lo cual se deben pasar los datos de la habitación en formato JSON.

4 Plan de pruebas

El plan de pruebas del software es esencial para garantizar que todas las funcionalidades operen conforme a los requisitos especificados. A continuación, se describen los objetivos de las pruebas, la estrategia adoptada, los casos de prueba detallados y las herramientas utilizadas para llevar a cabo el proceso de pruebas de manera eficaz.

4.1 Objetivos de las pruebas

Los principales objetivos de las pruebas son:

- Verificación de Funcionalidad: Asegurar que todas las funcionalidades del software responden conforme a los requisitos especificados. Esto incluye verificar que cada componente del sistema cumple su propósito de manera correcta.

- **Detección de Defectos:** Identificar y corregir defectos en el software antes de su implementación en un entorno de producción. Las pruebas exhaustivas ayudan a descubrir errores que podrían afectar la usabilidad y funcionalidad del sistema.
- **Calidad del Software:** Garantizar que el software es de alta calidad, fiable y robusto. Las pruebas rigurosas aseguran que el sistema puede manejar escenarios normales y extremos sin fallos.
- **Validación de Integración:** Comprobar que los diferentes módulos y componentes del sistema se integran correctamente y funcionan juntos sin problemas.
- **Pruebas de Regresión:** Asegurar que las nuevas funcionalidades o cambios en el código no introduzcan nuevos defectos en partes ya probadas del sistema.

4.2 Estrategia de pruebas

La estrategia de pruebas incluye un enfoque meticuloso que abarca varios tipos de pruebas y técnicas, utilizando herramientas especializadas para la automatización:

1. **Pruebas Funcionales:** Estas pruebas se enfocan en verificar que cada función del software opera según las especificaciones. Incluyen pruebas de todas las funcionalidades descritas en los requisitos.
2. **Pruebas de Integración:** Estas pruebas aseguran que todos los módulos y componentes del sistema interactúan correctamente. Se llevan a cabo después de las pruebas funcionales para validar que la integración del sistema no introduce nuevos defectos.

3. Pruebas de Regresión: Cada vez que se realiza un cambio en el código, se ejecutan pruebas de regresión para asegurarse de que el nuevo código no ha afectado negativamente a las funcionalidades existentes.
4. Pruebas Automatizadas: Para facilitar y agilizar el proceso de pruebas, se utiliza Postman, una herramienta para pruebas de API. Postman permite la creación y ejecución de pruebas automatizadas utilizando JavaScript. Esto permite la ejecución eficiente de pruebas repetitivas y complejas, garantizando consistencia y rapidez.

4.3 Casos de prueba

Los casos de prueba para el sistema los vamos a agrupar por funcionalidades para que sea más fácil de interpretar y son los siguientes:

Búsqueda de hoteles por localidad:

URL: `http://localhost:5000/api/buscar_hotel_localidad/{localidad}`

- ID: PP-001
Búsqueda de hoteles por localidad: Enviar solicitud a `http://localhost:5000/api/buscar_hotel_localidad/` con "León" en la URL.
Resultado esperado: Código HTTP 200.
- ID: PP-002
Búsqueda de hoteles por localidad: Enviar solicitud a `http://localhost:5000/api/buscar_hotel_localidad/` sin ningún valor en la URL.
Resultado esperado: Código HTTP 404.

- ID: PP-003

Búsqueda de hoteles por localidad: Enviar solicitud a http://localhost:5000/api/buscar_hotel_localidad/ con "Cáceres" (sin hoteles).
Resultado esperado: Código HTTP 404.

Búsqueda de hoteles por categoría:

URL: http://localhost:5000/api/buscar_hotel_categoria/{categoria}

- ID: PP-004

Búsqueda de hoteles por categoría: Enviar solicitud a http://localhost:5000/api/buscar_hotel_categoria/ con "4" en la URL.
Resultado esperado: Código HTTP 200.

- ID: PP-005

Búsqueda de hoteles por categoría: Enviar solicitud a http://localhost:5000/api/buscar_hotel_categoria/ sin ningún parámetro en la URL.
Resultado esperado: Código HTTP 404.

- ID: PP-006

Búsqueda de hoteles por categoría: Enviar solicitud a http://localhost:5000/api/buscar_hotel_categoria/ con categoría 0.
Resultado esperado: Código HTTP 404.

- ID: PP-007

Búsqueda de hoteles por categoría: Enviar solicitud a http://localhost:5000/api/buscar_hotel_categoria/ con categoría 6.
Resultado esperado: Código HTTP 404.

- ID: PP-008

Búsqueda de hoteles por categoría: Enviar solicitud a http://localhost:5000/api/buscar_hotel_categoria/ con un tipo de dato no entero (e.g., "A").

Resultado esperado: Código HTTP 404.

Registrar un nuevo hotel en la BBDD:

URL: http://localhost:5000/api/insertar_hotel/

- ID: PP-009

Registro de un nuevo hotel: Enviar solicitud a http://localhost:5000/api/insertar_hotel/ con JSON de hotel completo, excepto la clave primaria.

Resultado esperado: Código HTTP 200.

- ID: PP-010

Registro de un nuevo hotel: Enviar solicitud a http://localhost:5000/api/insertar_hotel/ con JSON con todos los atributos.

Resultado esperado: Código HTTP 200.

- ID: PP-011

Registro de un nuevo hotel: Enviar solicitud a http://localhost:5000/api/insertar_hotel/ con JSON sin el nombre del hotel.

Resultado esperado: Código HTTP 400.

- ID: PP-012

Registro de un nuevo hotel: Enviar solicitud a http://localhost:5000/api/insertar_hotel/ con JSON con todos los campos menos la descripción.

Resultado esperado: Código HTTP 200.

- ID: PP-013

Registro de un nuevo hotel: Enviar solicitud a `http://localhost:5000/api/insertar_hotel/` con JSON con todos los campos menos la categoría.

Resultado esperado: Código HTTP 400.

- ID: PP-014

Registro de un nuevo hotel: Enviar solicitud a `http://localhost:5000/api/insertar_hotel/` con JSON con todos los campos menos si tiene piscina.

Resultado esperado: Código HTTP 400.

- ID: PP-015

Registro de un nuevo hotel: Enviar solicitud a `http://localhost:5000/api/insertar_hotel/` con JSON con todos los campos menos la localidad.

Resultado esperado: Código HTTP 200.

- ID: PP-016

Registro de un nuevo hotel: Enviar solicitud a `http://localhost:5000/api/insertar_hotel/` con JSON con todos los campos completos, cambiando el tipo de dato en el campo nombre.

Resultado esperado: Código HTTP 400.

- ID: PP-017

Registro de un nuevo hotel: Enviar solicitud a `http://localhost:5000/api/insertar_hotel/` con JSON con todos los campos completos, cambiando el tipo de dato en el campo categoría.

Resultado esperado: Código HTTP 400.

- ID: PP-018

Registro de un nuevo hotel: Enviar solicitud a `http://localhost:5000/api/insertar_hotel/` con JSON con todos los campos completos, cambiando el tipo de dato en el campo `tiene_piscina`.

Resultado esperado: Código HTTP 400.

- ID: PP-019

Registro de un nuevo hotel: Enviar solicitud a `http://localhost:5000/api/insertar_hotel/` con JSON con todos los campos completos, cambiando el tipo de dato en el campo localidad.

Resultado esperado: Código HTTP 400.

Eliminar una habitación:

URL: `http://localhost:5000/api/eliminar_habitacion/{idHabitacion}`

- ID: PP-020

Enviar solicitud a `http://localhost:5000/api/eliminar_habitacion/{idHabitacion}` con el ID de habitación en la URL.

Resultado esperado: Código HTTP 200.

- ID: PP-021

Eliminar una habitación: Enviar solicitud a `http://localhost:5000/api/eliminar_habitacion/` con el ID de habitación vacío en la URL.

Resultado esperado: Código HTTP 200.

- ID: PP-022

Eliminar una habitación: Enviar solicitud a `http://localhost:5000/api/eliminar_habitacion/{idHabitacion}` con un dato que no sea un número en la URL.

Resultado esperado: Código HTTP 404.

Modificar la ocupación de una habitación:

URL: http://localhost:5000/api/modificar_ocupacion/{idHabitacion}

- ID: PP-023

Modificar la ocupación de una habitación: Enviar solicitud a http://localhost:5000/api/modificar_ocupacion/{idHabitacion} con el ID de una habitación existente en la URL.

Resultado esperado: Código HTTP 200.

- ID: PP-024

Modificar la ocupación de una habitación: Enviar solicitud a http://localhost:5000/api/modificar_ocupacion/ sin ningún ID de habitación en la URL.

Resultado esperado: Código HTTP 404.

- ID: PP-025

Modificar la ocupación de una habitación: Enviar solicitud a http://localhost:5000/api/modificar_ocupacion/{idHabitacion} con un ID de habitación que no sea un número entero en la URL.

Resultado esperado: Código HTTP 400.

Registrar habitación en la BBDD:

URL: http://localhost:5000/api/insertar_habitacion/

- ID: PP-026

Insertar una habitación nueva: Enviar solicitud a http://localhost:5000/api/insertar_habitacion/ con el objeto JSON que contiene todos los campos, excepto el ID de habitación.

Resultado esperado: Código HTTP 200.

- ID: PP-027
Insertar una habitación nueva: Enviar solicitud a http://localhost:5000/api/insertar_habitacion/ con el objeto JSON que contiene todos los campos, excepto el ID de habitación.
Resultado esperado: Código HTTP 400.

- ID: PP-028
Insertar una habitación nueva: Enviar solicitud a http://localhost:5000/api/insertar_habitacion/ con el objeto JSON que contiene todos los campos, excepto el ID de hotel.
Resultado esperado: Código HTTP 400.

- ID: PP-029
Insertar una habitación nueva: Enviar solicitud a http://localhost:5000/api/insertar_habitacion/ con el objeto JSON que contiene todos los campos, excepto el tamaño.
Resultado esperado: Código HTTP 400.

- ID: PP-030
Insertar una habitación nueva: Enviar solicitud a http://localhost:5000/api/insertar_habitacion/ con el objeto JSON que contiene todos los campos, excepto la capacidad.
Resultado esperado: Código HTTP 400.

- ID: PP-031
Insertar una habitación nueva: Enviar solicitud a http://localhost:5000/api/insertar_habitacion/ con el objeto JSON que contiene todos los campos, excepto el precio por noche.
Resultado esperado: Código HTTP 400.

- ID: PP-032
Insertar una habitación nueva: Enviar solicitud a `http://localhost:5000/api/insertar_habitacion/` con el objeto JSON que contiene todos los campos, excepto si incluye desayuno.
Resultado esperado: Código HTTP 400.

- ID: PP-033
Insertar una habitación nueva: Enviar solicitud a `http://localhost:5000/api/insertar_habitacion/` con el objeto JSON que contiene todos los campos, excepto si está ocupada.
Resultado esperado: Código HTTP 400.

- ID: PP-034
Insertar una habitación nueva: Enviar solicitud a `http://localhost:5000/api/insertar_habitacion/` con el objeto JSON que contiene todos los campos, cambiando el tipo de dato esperado en la capacidad.
Resultado esperado: Código HTTP 400.

- ID: PP-035
Insertar una habitación nueva: Enviar solicitud a `http://localhost:5000/api/insertar_habitacion/` con el objeto JSON que contiene todos los campos, cambiando el tipo de dato esperado en el precio por noche.
Resultado esperado: Código HTTP 400.

- ID: PP-036
Insertar una habitación nueva: Enviar solicitud a `http://localhost:5000/api/insertar_habitacion/` con el objeto JSON que contiene todos los campos, cambiando el tipo de dato esperado en si incluye desayuno.
Resultado esperado: Código HTTP 400.

A pesar de los casos de prueba detallados anteriormente, cabe destacar que no se incluye toda la información del plan de pruebas en este documento debido a su extensión. Para obtener información más detallada sobre cada caso de prueba y

aspectos adicionales del plan de pruebas, se recomienda consultar el archivo Excel "PlanDePruebasV1.xlsx" que se entrega con todo el código del software y la automatización de las pruebas.

4.4 Herramientas de prueba

Las herramientas que usadas son las siguientes:

- Postman: Permite realizar solicitudes HTTP a la API, automatizar las pruebas y validar las respuestas recibidas. Su interfaz gráfica de usuario simplifica la creación de pruebas y la organización de colecciones de pruebas.

- JavaScript en Postman: Se utilizará para programar las pruebas y realizar validaciones automáticas de las respuestas de la API. Los scripts en JavaScript permiten definir expectativas y aserciones que comprueban si las respuestas cumplen con los requisitos especificados.

- Node.js: es un entorno de ejecución para JavaScript que permite a los desarrolladores ejecutar código JavaScript en el servidor, fuera del navegador. Se usara para realizar los reportes de pruebas de manera automática a través de Newman.

El uso de estas herramientas garantiza un proceso de pruebas eficiente, repetible y preciso, contribuyendo significativamente a la calidad y fiabilidad del software desarrollado.

5 Desarrollo de las pruebas

La implementación de las pruebas se llevará a cabo utilizando Postman, una herramienta popular para pruebas de APIs. A continuación, se describen los pasos generales para implementar las pruebas:

- Configuración de Variables: Se definirán variables globales y de entorno en Postman para manejar datos dinámicos como URLs de la API, tokens de autenticación y otros parámetros necesarios.

Globals				
Filter variables				
Variables	Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/>	URL	default	http://localhost:5000/api/	http://localhost:5000/api/
<input checked="" type="checkbox"/>	TOKEN	secret
<input checked="" type="checkbox"/>	LOCALIDAD	default	buscar_hotel_localidad/	buscar_hotel_localidad/
<input checked="" type="checkbox"/>	CATEGORIA	default	buscar_hotel_categoria/	buscar_hotel_categoria/
<input checked="" type="checkbox"/>	AUTORIZACION	default	autorizacion	autorizacion
<input checked="" type="checkbox"/>	INSERTAR_HOTEL	default	insertar_hotel/	insertar_hotel/
<input checked="" type="checkbox"/>	ELIMINAR_HABITACION	default	eliminar_habitacion/	eliminar_habitacion/
<input checked="" type="checkbox"/>	MODIFICAR_OCUPACION	default	modificar_ocupacion/	modificar_ocupacion/
<input checked="" type="checkbox"/>	INSERTAR_HABITACION	default	insertar_habitacion/	insertar_habitacion/
<input checked="" type="checkbox"/>	BORRAR_HOTEL	default	borrar_hotel/	borrar_hotel/
<input checked="" type="checkbox"/>	id_hotel	default		
<input checked="" type="checkbox"/>	id_habitacion	default		

Fuente: elaboración propia

Apreciamos en la foto anterior como se ha creado un environment para la gestión de todas nuestras variables, para las distintas peticiones que podemos hacer a la API, para recoger variables de las respuestas a ciertas peticiones, como puede ser el caso de la variable id_hotel e incluso la gestión del token de seguridad necesario para el acceso a la API.

A continuación, vamos a explicar brevemente cómo usar estas variables en el entorno de Postman. Para ello vamos a facilitar una imagen de cómo se realiza una petición en este entorno configurada a través de variables.



Fuente: elaboración propia

Como se puede comprobar observando las 2 imágenes anteriores, la variable {{URL}} es igual a la URL principal de la api: <http://localhost:5000/api/>. Las siguientes variables que estaríamos configurando {{BORRAR_HOTEL}} es igual al método DELETE que tenemos que llamar cuando queremos borrar un hotel, borrar_hotel/ y por último, tenemos la variable {{id_hotel}}, esta variable, si nos fijamos bien en la foto en la que se muestran todas las variables que se van a usar para el proyecto,

vemos que no tiene ningún valor inicial ni actual; esto se debe a que es una variable que se recoge de una de las pruebas cuando insertamos el objeto hotel mediante un método POST y comprobamos que nos devuelve el mensaje correcto. Después, mediante un método GET comprobamos que ese hotel se encuentra en la BBDD, es aquí cuando hacemos la comprobación de que el hotel, efectivamente, está en la BBDD donde vamos a recoger ese id_hotel y lo hacemos recogiendo la variable del json a través de una pm.environment, que se encarga, siempre y cuando el nombre de la variable en el entorno y el nombre de la variable en el test de esa prueba sean igual, de ir actualizando el valor de esa variable a medida que se va usando en los tests. Por lo que la URL que nos dejaría el uso de esas 3 variables sería así: http://localhost:5000/api/borrar_hotel/1

```
pm.test("Check Response Data",function(){  
    pm.environment.set("id_hotel",jsonData[0].idHotel);  
})
```

Fuente: elaboración propia

El mismo método que usa para recoger el id_hotel se utilizará para recoger el token de seguridad de la aplicación

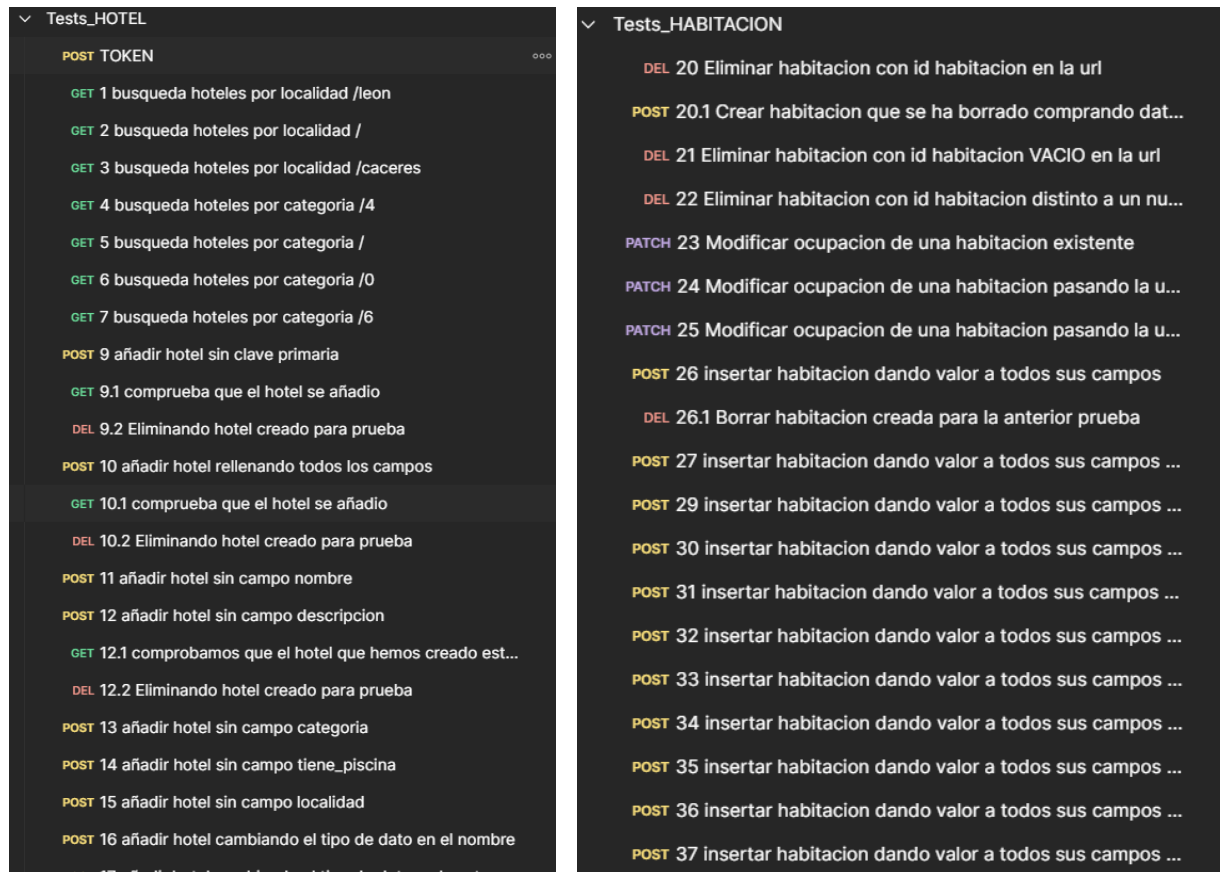
-Creación de Colecciones: Se organizarán las pruebas en colecciones, agrupando las peticiones relacionadas para facilitar su ejecución y mantenimiento.

-Definición de Peticiones: Se crearán peticiones HTTP específicas para cada caso de prueba, definiendo los métodos (GET, POST, PATCH, DELETE), los headers, y el cuerpo de las peticiones según sea necesario.

-Scripts de Validación: Se escribirán scripts de validación en JavaScript para verificar las respuestas de la API. Estos scripts comprobarán el código de estado de las respuestas, la estructura del JSON devuelto y otros criterios definidos en los casos de prueba.

5.1 Implementación de pruebas

Como hemos introducido en el punto anterior, utilizaremos colecciones para organizar la estructura de nuestras pruebas en Postman. Como estamos trabajando con una base de datos relacional de dos tablas: hotel y habitación, se decide crear dos colecciones, una para las operaciones que se van a realizar con hotel y otra para las operaciones que se van a realizar con habitación. De esta manera, al estar utilizando la misma petición en repetidas ocasiones para cubrir todos los planes de prueba que se han previsto, podemos duplicar las peticiones y así solo debemos cambiar el test de la prueba o el body, o los dos si el caso de prueba lo requiere.



Fuente: elaboración propia

Definición de Peticiones HTTP para Casos de Prueba

En el contexto del desarrollo y la prueba de aplicaciones web, es fundamental definir peticiones HTTP precisas y detalladas para cada escenario de prueba. A continuación, se describe cómo se configura cada tipo de petición HTTP, especificando su propósito y los componentes clave que deben incluirse.

Métodos HTTP

-GET: Este método se utiliza para solicitar datos de un servidor. Es una petición de solo lectura y no debe modificar el estado del recurso en el servidor. Ideal para recuperar información sin causar efectos colaterales. Su propósito es obtener datos o recursos específicos del servidor. Componentes:

- URL: La dirección del recurso solicitado.
- Headers: Pueden incluir parámetros como autenticación, tipo de contenido aceptado, etc.

-POST: Este método se emplea para enviar datos al servidor, generalmente para crear nuevos recursos. A diferencia de GET, puede modificar el estado del servidor. Su propósito es enviar datos para crear un nuevo recurso en el servidor. Componentes:

- URL: La dirección del recurso donde se enviarán los datos.
- Headers: Especifican el tipo de contenido enviado, como application/json.
- Cuerpo de la petición: Contiene los datos que se van a enviar, generalmente en formato JSON o XML.

-PATCH: Similar a PUT, pero se usa para aplicar modificaciones parciales a un recurso existente, en lugar de reemplazarlo por completo. Su propósito es actualizar parcialmente un recurso existente en el servidor. Componentes:

- URL: Dirección del recurso que se actualizará.
- Headers: Similar a POST, especifica el tipo de contenido.
- Cuerpo de la petición: Contiene las modificaciones a aplicar en el recurso.

-DELETE: Este método se utiliza para eliminar un recurso existente en el servidor. Al igual que PUT, es idempotente, es decir, hacer la misma petición varias veces no cambiará el resultado. Su propósito es eliminar un recurso específico del servidor. Componentes:

- URL: La dirección del recurso que se desea eliminar.
- Headers: Pueden incluir parámetros de autenticación y otras configuraciones relevantes.

Ahora vamos a explicar un caso de prueba para cada petición HTTP que hemos definido

Método GET:



Fuente: elaboración propia

Esta URL, como hemos explicado anteriormente, sería equivalente a http://localhost:5000/api/buscar_hotel_localidad/leon y nos mostraría todos los hoteles que hay en la localidad de León. Como estamos trabajando en un entorno controlado, sabemos los datos que tenemos en la BBDD y conocemos lo que nos va a devolver esta prueba. De hecho, es primordial conocer la respuesta de las peticiones para poder realizar los test acordes a esos resultados. Esta petición nos debe devolver dos hoteles en León, uno de ellos con el id=3 y el otro con el id=4. Siendo conocedores de estos datos vamos a verificar que, efectivamente, se están devolviendo esos dos hoteles que esperamos recibir y esto lo vamos a hacer en el apartado de test que tiene reservado Postman para este tipo de verificaciones.

```
1  var jsonData = pm.response.json(); // Obtiene los datos de la respuesta en formato JSON
2
3  var body=pm.response.text();
4
5  // Verifica el estado de la respuesta
6  pm.test("Estado de la respuesta", function() {
7      pm.response.to.have.status(200); // Verifica que la respuesta tenga un estado 200
8      // (OK)
9  });
10
11 //Verifica que el body no este vacio
12 pm.test("Comprueba que el body no este vacio", function() {
13     pm.expect(body).to.be.not.empty;
14 });
15
16 // Verifica el numero de registro
17 pm.test("Comprobación id_hotel y localidad", function() {
18     // Comprobación de los 2 ids de los hoteles de leon que son 3 y 4
19     pm.expect(jsonData[0].idHotel).to.eql(3);
20     pm.expect(jsonData[1].idHotel).to.eql(4);
21
22     pm.expect(jsonData[0].localidad).to.eql("Leon");
23     pm.expect(jsonData[1].localidad).to.eql("Leon");
24 });
25
```

Fuente: elaboración propia

Este es el código que vamos a utilizar para comprobar que se está devolviendo la respuesta que nosotros esperábamos.

Vamos a explicar línea a línea el código y así vemos para qué sirve cada parte.

-var jsonData = pm.response.json();: Esta línea convierte la respuesta HTTP en un objeto JSON y lo almacena en la variable jsonData. pm.response.json() es una función de Postman que facilita la conversión de la respuesta a JSON, permitiendo acceder a sus datos de forma estructurada.

-var body = pm.response.text();: Esta línea obtiene el cuerpo de la respuesta HTTP en formato de texto y lo almacena en la variable body. pm.response.text() convierte el cuerpo de la respuesta en una cadena de texto.

-pm.test("Estado de la respuesta", function() { pm.response.to.have.status(200); }): Esta sección define una prueba en Postman que verifica el estado de la respuesta HTTP. La función pm.test toma una descripción de la prueba y una función de callback. Dentro de esta función, pm.response.to.have.status(200) verifica que el estado de la respuesta sea 200, lo cual indica que la petición fue exitosa.

-pm.test("Comprueba que el body no esté vacío", function() {pm.expect(body).to.be.not.empty; });: Esta prueba verifica que el cuerpo de la respuesta no esté vacío. Utiliza pm.expect para hacer la aserción, verificando que body no esté vacío (not.empty).

-pm.test("Comprobación número de registro del medicamento", function() { ... });: Esta prueba verifica valores específicos en los datos JSON de la respuesta.

-pm.expect(jsonData[0].idHotel).to.eql(3);: Verifica que el primer objeto en el array jsonData tenga un idHotel igual a 3.

-pm.expect(jsonData[1].idHotel).to.eql(4);: Verifica que el segundo objeto en el array jsonData tenga un idHotel igual a 4.

-pm.expect(jsonData[0].localidad).to.eql("Leon");: Verifica que el primer objeto en el array jsonData tenga la localidad igual a "Leon".

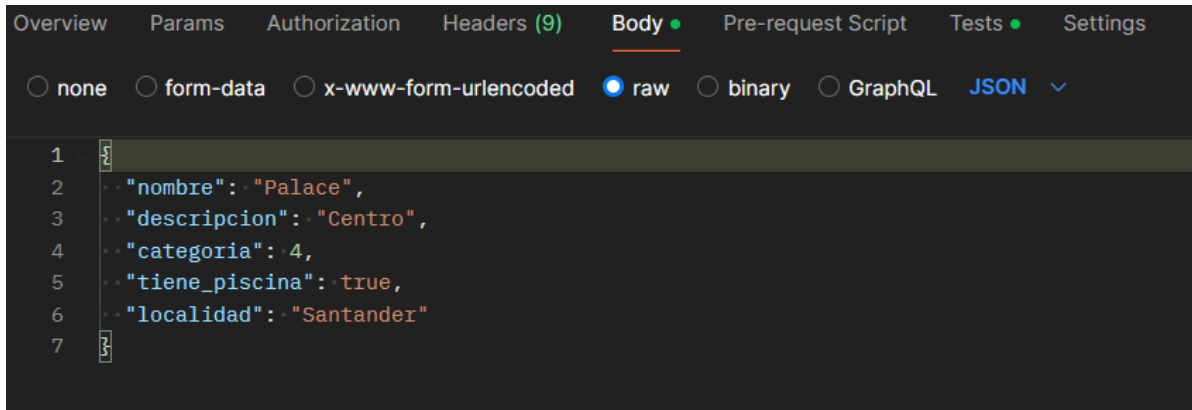
-pm.expect(jsonData[1].localidad).to.eql("Leon");: Verifica que el segundo objeto en el array jsonData tenga la localidad igual a "Leon".

Método POST:



Fuente: elaboración propia

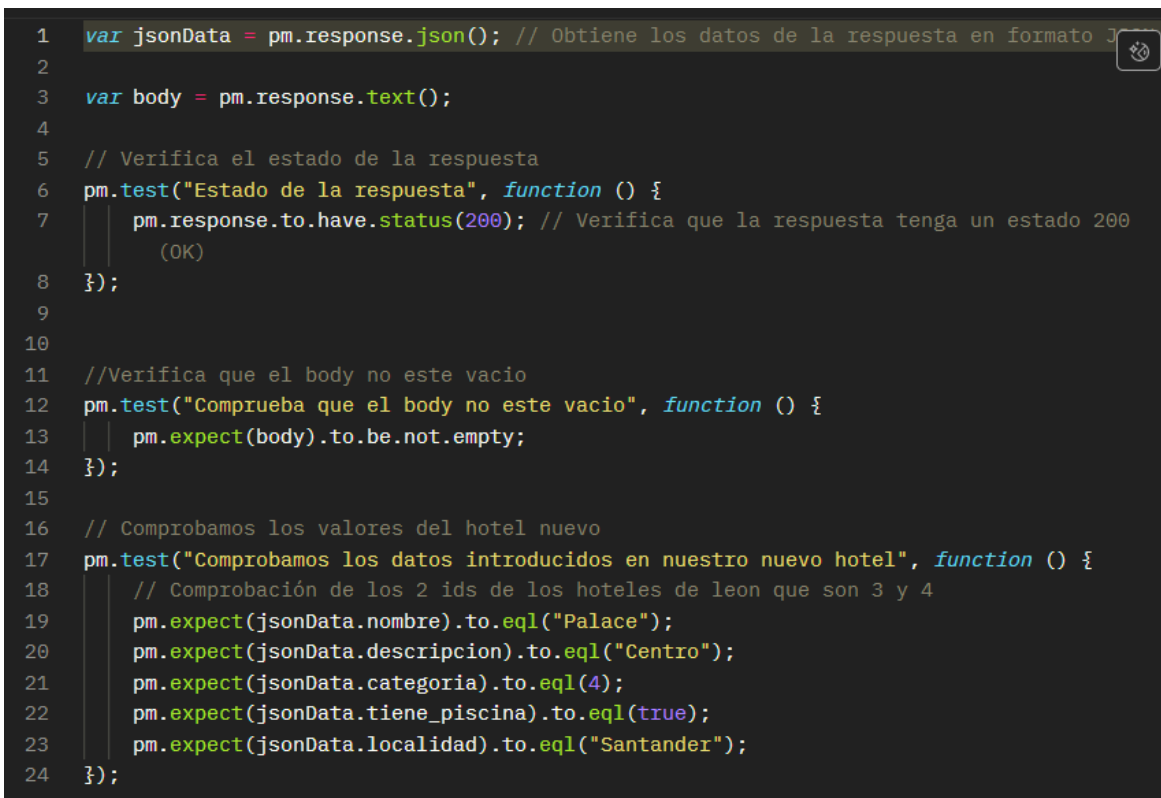
Esta URL sería equivalente a: http://localhost:5000/api/insertar_hotel/ y nos permite insertar un hotel que le pasemos nosotros a la petición en el cuerpo de la misma en formato .json



```
Overview Params Authorization Headers (9) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "nombre": "Palace",
3   "descripcion": "Centro",
4   "categoria": 4,
5   "tiene_piscina": true,
6   "localidad": "Santander"
7 }
```

Fuente: elaboración propia

Estos son los datos que mandamos en formato .json para la inserción del hotel. Para este método esperamos que nos devuelva un código 200 y el objeto hotel que acabamos de insertar. Para ello vamos a implementar las siguientes pruebas.



```
1 var jsonData = pm.response.json(); // Obtiene los datos de la respuesta en formato J
2
3 var body = pm.response.text();
4
5 // Verifica el estado de la respuesta
6 pm.test("Estado de la respuesta", function () {
7   pm.response.to.have.status(200); // Verifica que la respuesta tenga un estado 200
8   (OK)
9 });
10
11 //Verifica que el body no este vacio
12 pm.test("Comprueba que el body no este vacio", function () {
13   pm.expect(body).to.be.not.empty;
14 });
15
16 // Comprobamos los valores del hotel nuevo
17 pm.test("Comprobamos los datos introducidos en nuestro nuevo hotel", function () {
18   // Comprobación de los 2 ids de los hoteles de leon que son 3 y 4
19   pm.expect(jsonData.nombre).to.eql("Palace");
20   pm.expect(jsonData.descripcion).to.eql("Centro");
21   pm.expect(jsonData.categoria).to.eql(4);
22   pm.expect(jsonData.tiene_piscina).to.eql(true);
23   pm.expect(jsonData.localidad).to.eql("Santander");
24 });
```

Fuente: elaboración propia

Vamos a pasar a explicar línea a línea que va a hacer nuestro código:

-var jsonData = pm.response.json();: Esta línea convierte la respuesta HTTP recibida en un objeto JSON y lo almacena en la variable jsonData. Esto permite acceder a los datos de la respuesta de manera estructurada.

-var body = pm.response.text();: Esta línea obtiene el cuerpo de la respuesta HTTP en formato de texto y lo almacena en la variable body. Esto es útil para realizar comprobaciones sobre el contenido de la respuesta en su forma textual.

-pm.test("Estado de la respuesta", function () { pm.response.to.have.status(200); });: Esta sección define una prueba utilizando la función pm.test de Postman. La descripción de la prueba es "Estado de la respuesta". La función de callback dentro de pm.test utiliza pm.response.to.have.status(200) para verificar que el estado de la respuesta HTTP sea 200, indicando que la petición fue exitosa.

-pm.test("Comprueba que el body no esté vacío", function () { pm.expect(body).to.be.not.empty; });: Esta prueba verifica que el cuerpo de la respuesta no esté vacío. Utiliza pm.expect para hacer la aserción, comprobando que body no esté vacío.

-pm.test("Comprobamos los datos introducidos en nuestro nuevo hotel", function () { ... });: Esta prueba verifica que los valores específicos en el objeto JSON de la respuesta sean los esperados para un nuevo hotel.

-pm.expect(jsonData.nombre).to.eql("Palace");: Verifica que la propiedad nombre en jsonData sea igual a "Palace".

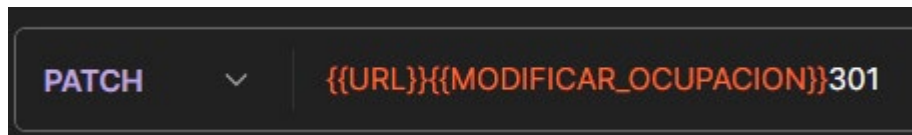
-pm.expect(jsonData.descripcion).to.eql("Centro");: Verifica que la propiedad descripcion en jsonData sea igual a "Centro".

-pm.expect(jsonData.categoria).to.eql(4);: Verifica que la propiedad categoria en jsonData sea igual a 4.

-pm.expect(jsonData.tiene_piscina).to.eql(true);: Verifica que la propiedad tiene_piscina en jsonData sea igual a true.

-pm.expect(jsonData.localidad).to.eql("Santander");: Verifica que la propiedad localidad en jsonData sea igual a "Santander".

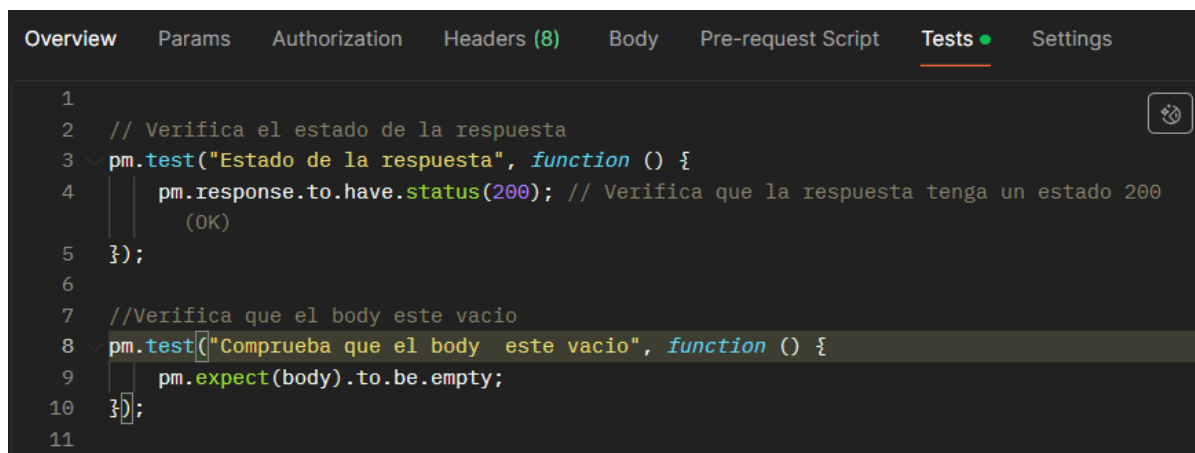
Método PATCH:



Fuente: elaboración propia

Esta URL sería equivalente a: http://localhost:5000/api/modificar_ocupacion/301 y nos permite modificar la ocupación de una habitación insertando la url con el id_Habitacion de la estancia, es decir, si está ocupada esa habitación, cambiaría su estado a vacía y viceversa. De esta petición esperamos un código http 200 y una respuesta vacía el body.

Cabe matizar que no se usa variable para el id_Habitacion porque es una variable que solo entra en juego esta vez.



Fuente: elaboración propia

A continuación, se explicará línea a línea el código:

-pm.test("Estado de la respuesta", function () { pm.response.to.have.status(200); });: Esta sección define una prueba utilizando la función pm.test de Postman. La

descripción de la prueba es "Estado de la respuesta". La función de callback dentro de `pm.test` utiliza `pm.response.to.have.status(200)` para verificar que el estado de la respuesta HTTP sea 200, indicando que la petición fue exitosa.

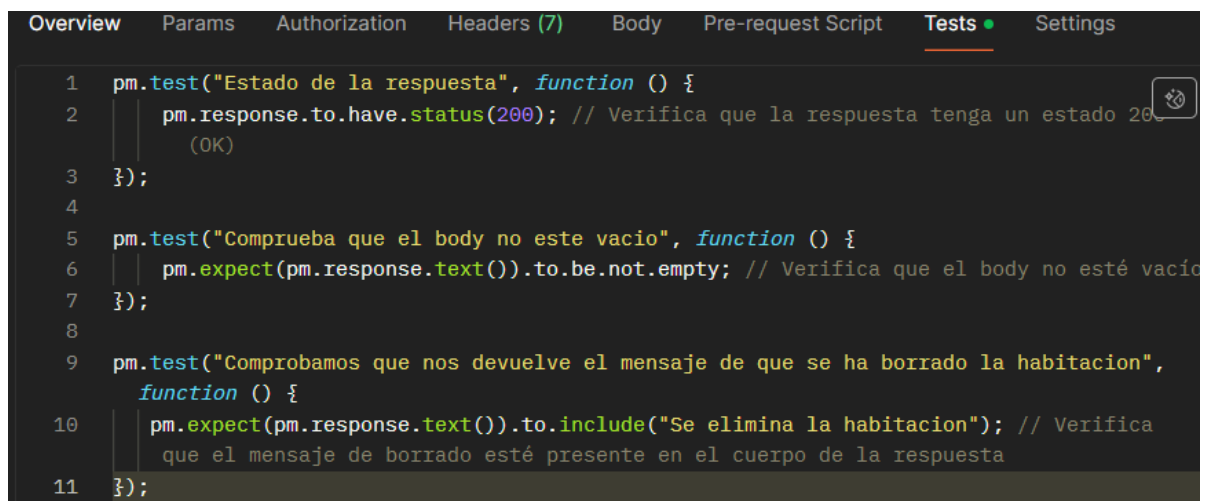
`-pm.test("Comprueba que el body esté vacío", function () {
 pm.expect(body).to.be.empty; });` Esta prueba verifica que el cuerpo de la respuesta esté vacío. Utiliza `pm.expect` para hacer la aserción, comprobando que `body` no esté vacío.

Método DELETE:



Fuente: elaboración propia

La URL sería equivalente a: http://localhost:5000/api/eliminar_habitacion/101. Este método nos permite eliminar una habitación indicando en la petición el `id_Habitacion` de la estancia que deseamos borrar. La respuesta que se espera de esta petición es un código 200 y un mensaje en el body "Se elimina la habitación". Para hacer estas comprobaciones utilizamos este código.



Fuente: elaboración propia

`-pm.test("Estado de la respuesta", function () { pm.response.to.have.status(200); });`:: Esta sección define una prueba utilizando la función `pm.test` de Postman. La descripción de la prueba es "Estado de la respuesta". La función de callback dentro de `pm.test` utiliza `pm.response.to.have.status(200)` para verificar que el estado de la respuesta HTTP sea 200, indicando que la petición fue exitosa.

`-pm.test("Comprueba que el body no esté vacío", function () { pm.expect(body).to.be.not.empty; });`:: Esta prueba verifica que el cuerpo de la respuesta no esté vacío. Utiliza `pm.expect` para hacer la aserción, comprobando que `body` no esté vacío.

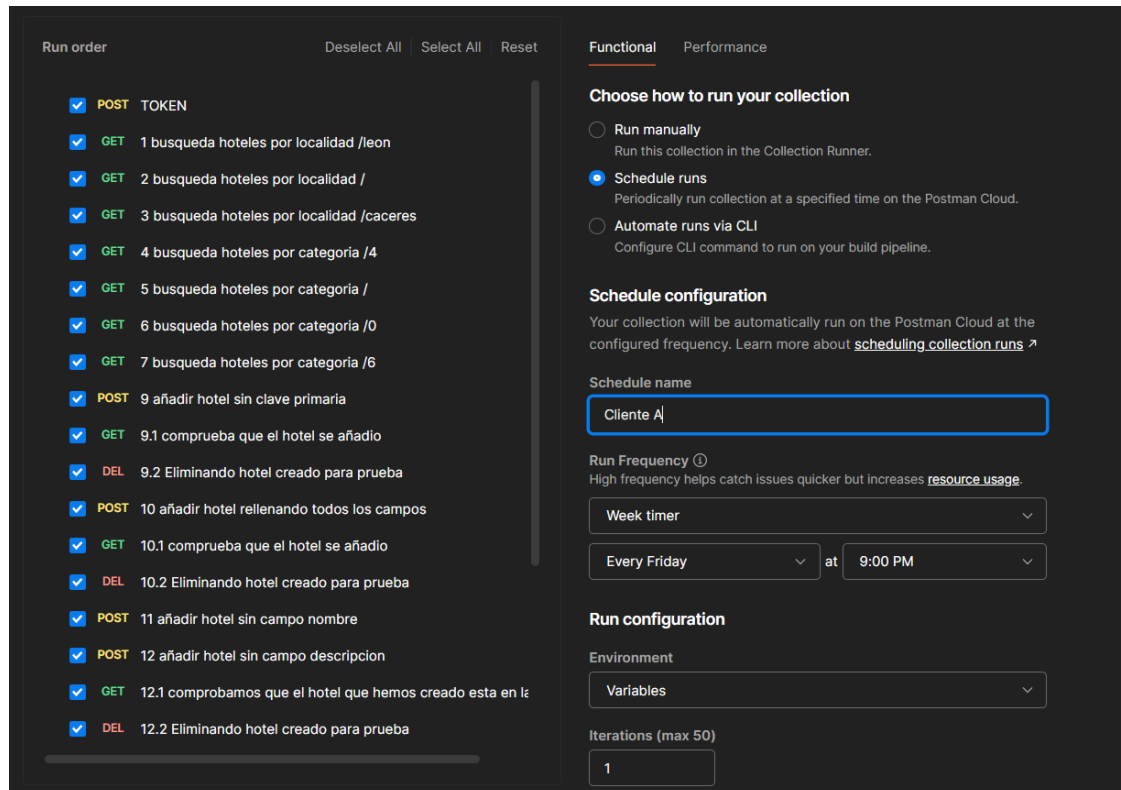
`-pm.test("Comprobamos que nos devuelve el mensaje de que se ha borrado la habitacion", function () { pm.expect(pm.response.text()).to.include("Se elimina la habitacion"); });`:: Esta prueba verifica que el cuerpo de la respuesta sea "Se elimina la habitacion". Utiliza `pm.expect` para hacer la aserción, comprobando que en el cuerpo de la respuesta este incluido el mensaje "Se elimina la habitación".

Así es como hemos realizado el proceso de implementación de las pruebas y las hemos automatizado mediante el uso de test como hemos mostrado en estos cuatro ejemplos, cada uno de ellos de un método distinto. Para profundizar más, se pueden ver todos los casos en la carpeta "Colecciones&EntornoPostman".

5.2 Automatización de pruebas

En la sección previa, se explicó la implementación de pruebas automatizadas empleando la herramienta Postman. Automatizar pruebas implica programar y ejecutar pruebas para validar el funcionamiento de un software. Con Postman, la automatización se logra a través de la elaboración de scripts que interactúan con APIs y verifican automáticamente las respuestas.

Se puede emplear la función de Runner de Postman para mejorar la eficacia del proceso. Esto involucra incluir las series de pruebas en un Runner, que posibilita ejecutarlas de manera regular o en momentos específicos predeterminados. Así, se automatiza más el proceso de prueba, lo que facilita encontrar errores tempranamente y asegura la calidad del software en desarrollo.



Fuente: elaboración propia

En la imagen anterior vemos un ejemplo de configuración del runner para que todos los viernes a las 9 de la noche envíe un informe a la nube del “Cliente A” de cómo su software va evolucionando y ganando calidad gracias a la automatización de pruebas. Gracias a este tipo de proceso nos permite ganar mucho tiempo en reuniones y explicaciones, ya que automatizamos el proceso para mantener al cliente siempre al corriente de la evolución de su producto.

6 Gestión de errores

El proceso de reporte de errores es clave para la resolución de los problemas encontrados durante la fase de pruebas. Los errores se documentarán de manera estructurada para facilitar su identificación y resolución por parte del equipo de desarrollo.

Forma en la que vamos a enviar un reporte de bug al equipo de desarrollo.

Pasos:

- Pasos para reproducir el bug: En este paso se debe especificar claramente y sin lugar a duda, todas las acciones realizadas durante la prueba para hacer más fácil al equipo de desarrollo identificar el bug.
- Comportamiento actual: Se debe explicar, con un lenguaje neutral, el comportamiento exacto que está teniendo la aplicación en el caso en concreto que provoca la apertura del bug.
- Comportamiento esperado: En este apartado se debe especificar el comportamiento esperado que difiere del actual y que provoca la apertura del bug. Siempre que se pueda se hará referencia a la documentación funcional o técnica de lo que estamos probando
- Criticidad del bug: Indica la importancia del bug. Será útil para ir seleccionando de la pila los más prioritarios.

6.1 Reporte de errores

Después de la anterior introducción, vamos a reproducir un reporte de error real, que se ha dado en este proceso de desarrollo.

El caso que se va a utilizar es el siguiente:

- ID: PP-011

Registro de un nuevo hotel: Enviar solicitud a http://localhost:5000/api/insertar_hotel/ con JSON sin el nombre del hotel.

Resultado esperado: Código HTTP 400.

Para ello, vamos a nuestro plan de pruebas y extraemos todos los datos de las columnas:

ID: PP-011

Descripción del caso: añadir un hotel pasando el objeto .json con todos los campos a excepción del nombre del hotel , esperando un código http 400.

Condiciones: que la api esté disponible.

Datos de prueba: vamos a pasar un objeto hotel en formato .json dándole un valor a todos sus atributos menos a nombre, por ejemplo:

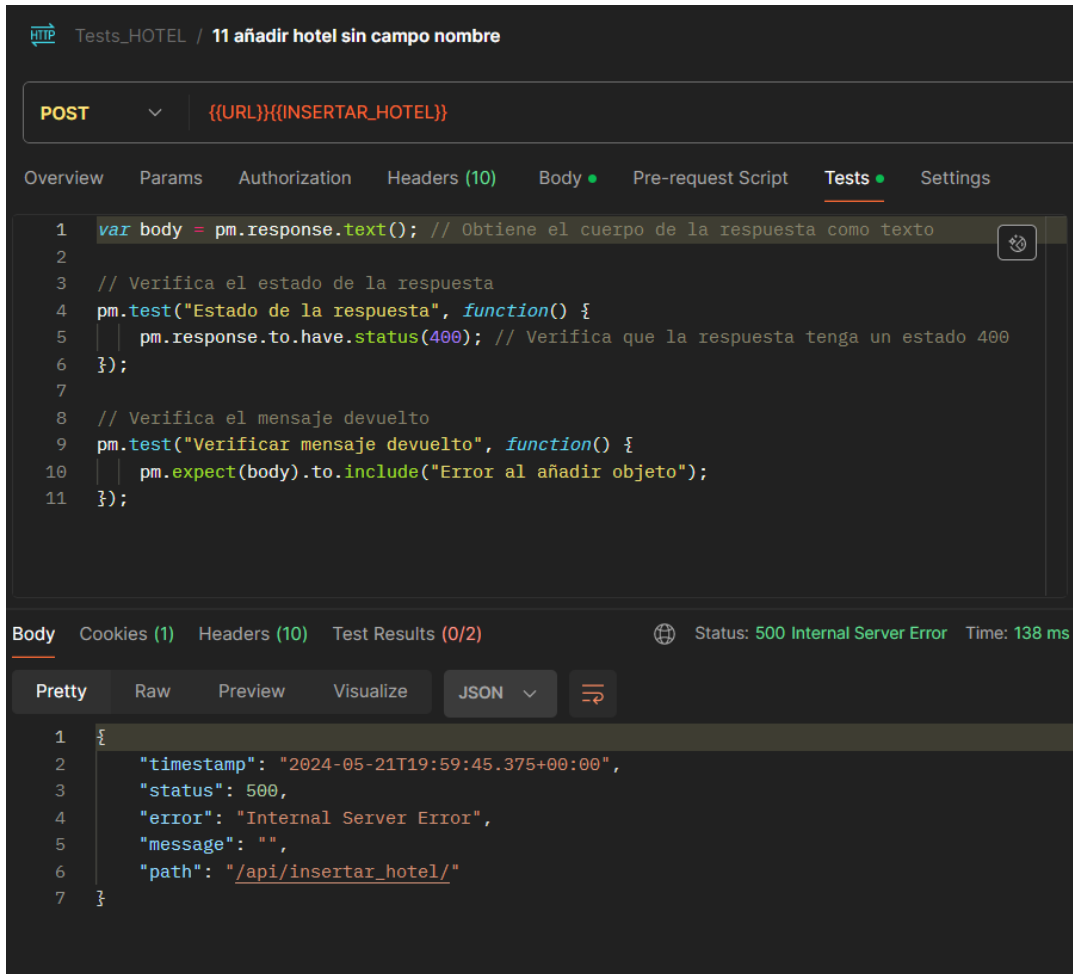
```
{  
  
  "descripcion": "Centro",  
  
  "categoria": 4,  
  
  "tiene_piscina": true,  
  
  "localidad": "Valladolid"  
}
```

Pasos a ejecutar: abrimos el programa para testear la api. Seleccionamos el método post, insertamos la url para llevar a cabo la operación:

http://localhost:5000/api/insertar_hotel/, introducimos los datos de nuestro .json y lanzamos la petición.

Resultado esperado: Emitirá un mensaje "Error al registrar hotel" y mostrará un código HTTP 400 KO.

Una vez tenemos estos datos debemos irnos a postman para realizar la petición y comprobar que resultado no está dando.



```
Tests_HOTEL / 11 añadir hotel sin campo nombre

POST {{URL}}{{INSERTAR_HOTEL}}

Overview Params Authorization Headers (10) Body Pre-request Script Tests Settings

1 var body = pm.response.text(); // Obtiene el cuerpo de la respuesta como texto
2
3 // Verifica el estado de la respuesta
4 pm.test("Estado de la respuesta", function() {
5   pm.response.to.have.status(400); // Verifica que la respuesta tenga un estado 400
6 });
7
8 // Verifica el mensaje devuelto
9 pm.test("Verificar mensaje devuelto", function() {
10   pm.expect(body).to.include("Error al añadir objeto");
11 });

Body Cookies (1) Headers (10) Test Results (0/2) Status: 500 Internal Server Error Time: 138 ms

Pretty Raw Preview Visualize JSON

1 {
2   "timestamp": "2024-05-21T19:59:45.375+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "message": "",
6   "path": "/api/insertar_hotel/"
7 }
```

Fuente: elaboración propia

El resultado que arroja la prueba es este: Emite un mensaje en json:

```
{
  "timestamp": "2024-05-13T16:48:51.864+00:00",
  "status": 500,
  "error": "Internal Server Error",
  "message": "",
  "path": "/api/insertar_hotel/"
}
```

y un código http 500.

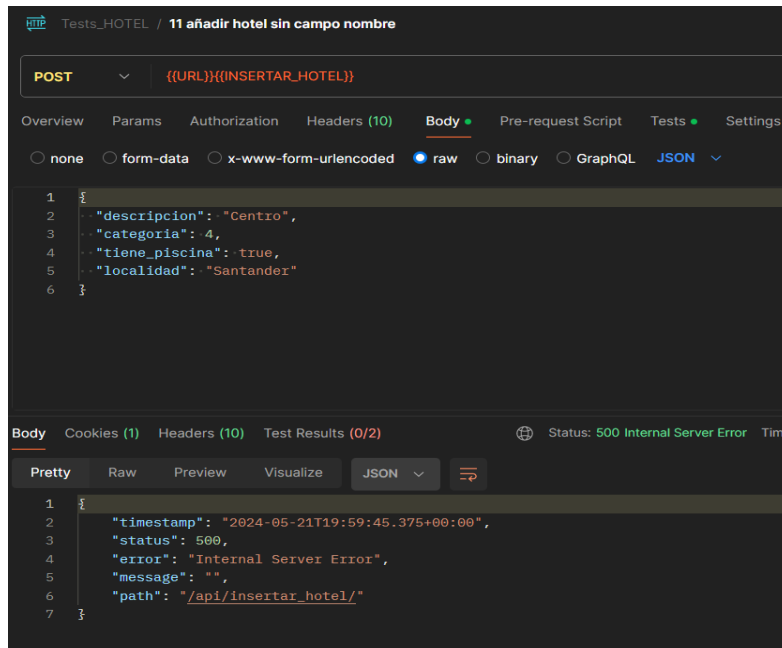
Una vez hemos realizado la prueba y nos damos cuenta de que el resultado esperado no coincide con el resultado real, es cuando comenzamos a realizar el reporte del bug, detallando el proceso por el cual hemos encontrado ese error y buscando evidencias del error para poder pasárselo al equipo de desarrollo y facilitar su tarea a la hora de resolverlo.

INFORME DE BUG

- **Caso:** ID: PP-011
- **Pasos para reproducir el bug:** Abrimos el programa para testear la api, seleccionamos el método post, insertamos la url para llevar a cabo la operación: http://localhost:5000/api/insertar_hotel/, introducimos los datos de nuestro .json y lanzamos la petición.
- **Comportamiento actual:** Devuelve un mensaje de error en formato .json:

```
{  
  
  "timestamp": "2024-05-13T16:48:51.864+00:00",  
  
  "status": 500,  
  
  "error": "Internal Server Error",  
  
  "message": "",  
  
  "path": "/api/insertar_hotel/"  
}
```

y un código http 500.



Fuente: elaboración propia

Se aporta una evidencia del error encontrado en forma de captura de pantalla para que sea más visual para el desarrollador encontrar el problema.

Comportamiento esperado: Que devuelva un mensaje "Error al registrar hotel" y mostrara un código HTTP 400 KO

Criticidad del bug: Menor, al ser un error que no afecta gravemente a la funcionalidad del software, pero que puede llegar a causar molestias al usuario al reportar un error 500.

De esta manera reportaremos los errores a nuestro equipo de desarrollo. En nuestro proyecto hemos detectado alrededor de 30 errores de las 37 pruebas que hemos realizado, entonces el procedimiento a seguir sería el mismo que acabamos de mostrar para el resto de los casos de prueba que han fallado o no han dado el resultado que nosotros esperábamos. Aunque sea una tarea de documentación costosa, es parte fundamental del proceso, ya que si no se hiciesen estas pruebas la calidad de cada sistema sería mucho menor porque sería más susceptible de sufrir algún fallo al no haber sido testeada en profundidad.

7 Corrección de errores y validación final

Después de que el equipo de desarrollo haya recibido los informes de errores del equipo de QA, se procederá a abordar y corregir cada uno de los problemas identificados. A continuación, se describen los pasos a seguir para la corrección de errores y las pruebas de regresión posteriores:

- Corrección de Errores: Los desarrolladores examinarán los informes de errores detallados proporcionados por el equipo de QA y trabajarán en la resolución de cada problema identificado. Esto puede implicar la modificación del código fuente, la corrección de errores lógicos o la actualización de la configuración del sistema.

- Despliegue de una Nueva Versión: Una vez que se hayan realizado las correcciones necesarias, se generará una nueva versión del software que incluya las actualizaciones y soluciones a los errores identificados. Esta nueva versión se desplegará en un entorno de pruebas para su validación antes de su lanzamiento.

- Re-pruebas: El equipo de QA volverá a ejecutar las pruebas que anteriormente resultaron fallidas para verificar si los errores han sido corregidos con éxito. Esto incluirá la repetición de los casos de prueba específicos que inicialmente revelaron problemas, así como pruebas adicionales para confirmar que las modificaciones realizadas no han introducido nuevos errores.

- Validación de Correcciones: Durante las re-pruebas, se prestará especial atención a los aspectos que anteriormente fallaron. Se verificará que los errores identificados se hayan resuelto correctamente y que el comportamiento del sistema ahora cumpla con los requisitos establecidos.

- Documentación de Resultados: Todos los resultados de las re-pruebas se documentarán de manera exhaustiva, incluyendo los casos de prueba que han pasado con éxito y aquellos que todavía presentan problemas. Esta documentación será crucial para evaluar el estado actual del software y determinar si está listo para la fase de producción.

7.1 Corrección de errores

La corrección de errores es un componente crucial en el ciclo de desarrollo de software, porque asegura la calidad y funcionalidad del producto final.

A continuación, se detallan los pasos y estrategias utilizadas para identificar y corregir errores durante el desarrollo de este proyecto.

7.1.1 Identificación de errores

Para detectar errores de manera efectiva, se implementaron las siguientes técnicas:

- Revisión de código: Se llevaron a cabo revisiones de código periódicas en las que el equipo de desarrollo inspeccionaba y discutía el código fuente en busca de errores y posibles mejoras.

- Pruebas automatizadas: Se desarrollaron y ejecutaron pruebas utilizando herramientas como Postman para la ejecución y automatización. Estas pruebas permitieron detectar errores en la primera versión del software.

7.1.2 Análisis de errores

Una vez identificado un error, se procede con el análisis para determinar su causa raíz:

- Depuración: Se emplearon depuradores (debuggers) para ejecutar el software en un entorno controlado, permitiendo examinar el estado del programa y localizar la causa exacta del error.

7.1.3 Resolución de errores

Los errores identificados se solucionaron:

-Priorización: Los errores se clasificaron según su gravedad e impacto en el sistema. Los errores críticos se resuelven los primeros, mientras que los problemas menores se resuelven los últimos.

-Corrección: Se realizaron las modificaciones necesarias en el código fuente para corregir los errores. Este proceso incluyó la reescritura de métodos, la corrección en la gestión de mensajes de error y la mejora de la lógica del programa.

No se puede escribir el código fuente en la memoria, pero se puede revisar las 2 versiones del software en la carpeta “Software” de mi proyecto. Se encuentra dividido en 2 carpetas llamadas “APIProyectoV1” y “APIProyectoV2” una para cada versión.

7.2 Pruebas de regresión

Las pruebas de regresión son vitales en el desarrollo de software. Se llevan a cabo estas pruebas para comprobar que las modificaciones recientes en el código, como la corrección de errores o la incorporación de nuevas funciones, no hayan generado errores adicionales.

El objetivo principal de las pruebas de regresión es asegurar que el sistema siga funcionando correctamente después de cualquier cambio en el código. Estas pruebas ayudan a identificar problemas que podrían surgir debido a la interacción de las nuevas modificaciones con el código existente, garantizando así la estabilidad y la calidad del software.

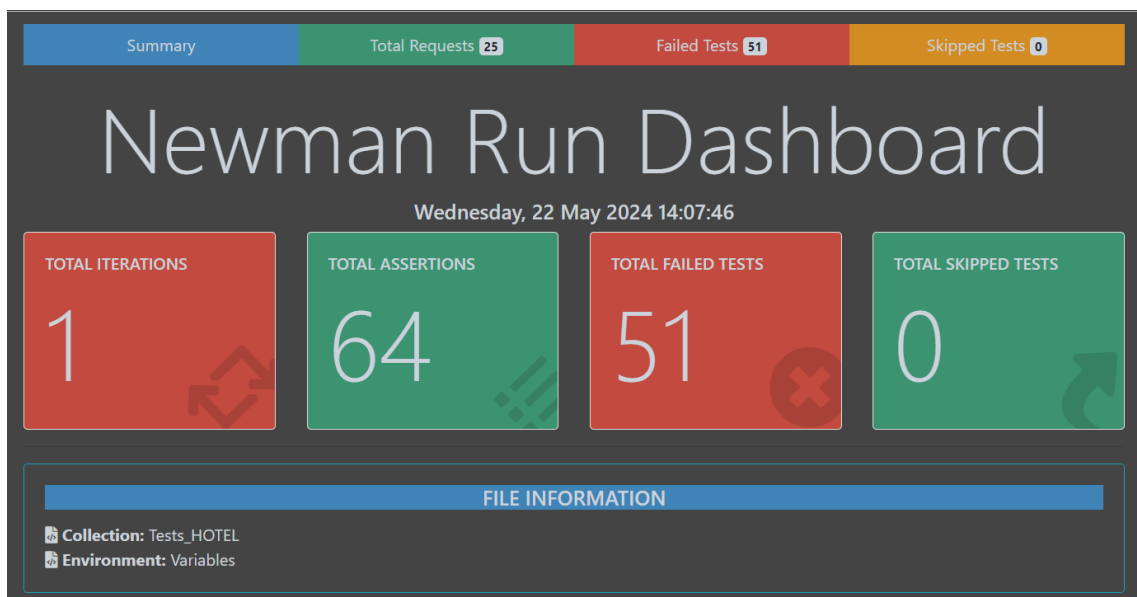
En este proyecto, las pruebas de regresión se realizaron volviendo a ejecutar todas las pruebas iniciales que se llevaron a cabo al comienzo del desarrollo. Para este propósito, se utilizaron herramientas como Postman y Newman para asegurar una cobertura completa y precisa.

-Automatización con Postman y Newman: Las pruebas definidas inicialmente en Postman fueron reutilizadas para las pruebas de regresión. Postman nos permitió estructurar y gestionar las pruebas de manera eficiente. Newman, la herramienta de línea de comandos de Postman, se utilizó para automatizar la ejecución de estas pruebas a través de un script que se adjunta en la carpeta “PruebasNewman”.

Esta automatización facilitó la ejecución repetitiva y consistente de las pruebas, lo que fue crucial para las pruebas de regresión.

-Generación de informes: Newman también se utilizó para generar informes en formato HTML. Estos informes proporcionaron una visión detallada de los resultados de las pruebas, facilitando la identificación de cualquier error o inconsistencia introducida por las modificaciones recientes.

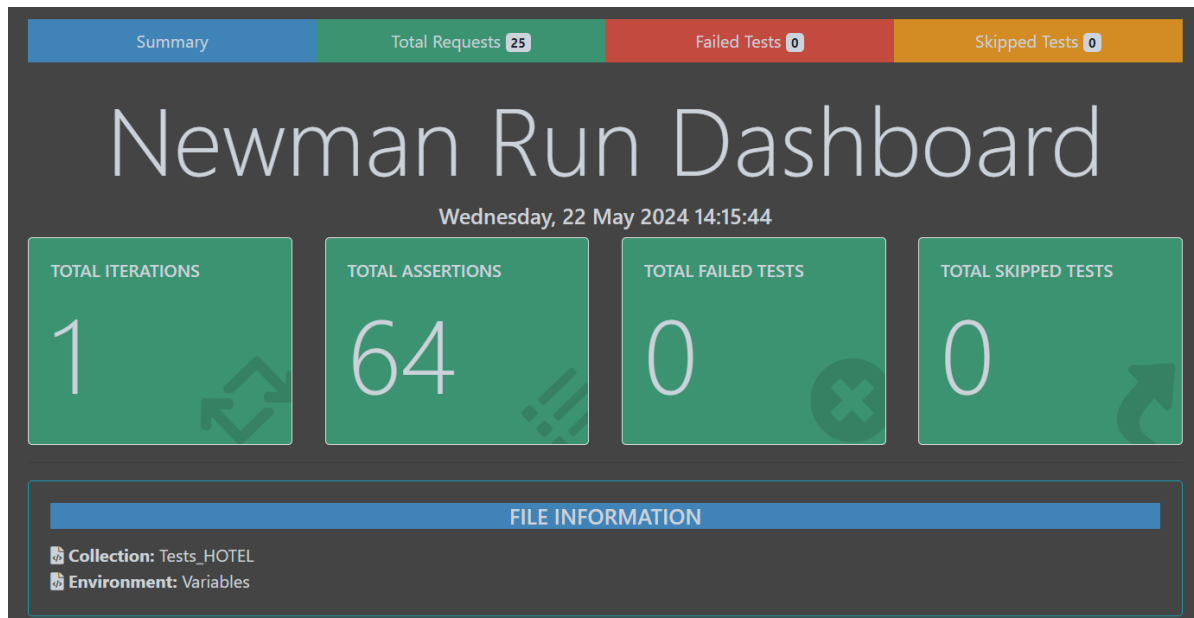
Ahora se mostrará el proceso de evolución que siguieron las pruebas a lo largo de este ciclo de desarrollo.



Fuente: elaboración propia

En esta imagen estamos apreciando el primer lanzamiento de pruebas después de su diseño e implementación. Recibimos un total de 64 aserciones de las cuales 51 de ellas fallan. La gran mayoría de peticiones a nuestra API no funcionaron como comentamos anteriormente (fallaron hasta un total de 30). Entonces, como dijimos hace ya varias páginas, comenzamos a reportar todos los errores y mandárselos al equipo de desarrollo para que vaya solucionándolos siempre de forma ordenada según se criterio de criticidad.

Este informe nos lo reporta Newman, que es la herramienta en línea de comandos de Postman y que en nuestro caso la vamos a utilizar para sacar los informes de pruebas sobre nuestra API.



Fuente: elaboración propia

Esta es la imagen que nos devuelve la prueba cuando se han hecho todas las modificaciones sobre el código fuente del programa, todos los test se superan lo que nos garantiza la calidad de nuestro software.

Las pruebas de regresión son vitales para mantener la calidad y la estabilidad del software. Al re-ejecutar las pruebas iniciales después de cada cambio, se asegura que el sistema, no solo cumple con los nuevos requisitos, sino que también sigue funcionando correctamente en todas las áreas previamente verificadas. En este proyecto, el uso de Postman y Newman permitió una automatización eficaz y una documentación clara de los resultados, asegurando una alta calidad del producto final.

7.3 Validación final

La validación final es un paso crucial en el ciclo de vida del desarrollo de software, cuyo propósito es asegurar que el producto cumple con todos los requisitos especificados y está listo para su lanzamiento. Este apartado describe el proceso de validación final llevado a cabo en este proyecto, destacando que las pruebas de regresión son suficientes para garantizar la calidad del software.

En este proyecto, se determinó que las pruebas de regresión exhaustivas son suficientes para la validación del software. El proceso de validación final incluye las siguientes etapas:

Revisión de requisitos:

Verificación de cumplimiento: Se revisaron todos los requisitos especificados para asegurar que han sido completamente implementados y funcionan según lo esperado.

Ejecución de pruebas de regresión:

-Pruebas automatizadas: Las pruebas de regresión se realizaron utilizando las pruebas automatizadas previamente definidas en Postman, ejecutadas con Newman para asegurar la cobertura completa de todas las funcionalidades clave.

-Validación continua: Se utilizó un sistema de integración continua para ejecutar pruebas de regresión automáticamente con cada cambio en el código, permitiendo una detección temprana de posibles problemas.

Generación y revisión de informes:

Informes de Newman: Los informes en formato HTML generados por Newman proporcionaron una visión detallada de los resultados de las pruebas, facilitando la identificación de cualquier error o inconsistencia introducida por las modificaciones recientes.

Documentación de pruebas:

Se revisaron y archivaron los resultados de todas las pruebas de regresión, asegurando que todos los problemas identificados fueron resueltos.

8 Conclusiones y posibles ampliaciones

Conclusión

El desarrollo de este proyecto ha permitido documentar y analizar de manera exhaustiva el ciclo de vida completo del software, tomando como ejemplo una API desarrollada con Spring boot. A lo largo del proyecto, se han identificado y documentado los requisitos iniciales, se ha elaborado un plan de pruebas detallado y se han implementado pruebas automatizadas utilizando Postman y Newman.

Las pruebas de regresión han sido fundamentales para asegurar que las modificaciones y actualizaciones del software no introdujeran nuevos errores, garantizando así la estabilidad y funcionalidad del sistema. La generación de informes detallados en formato HTML mediante Newman ha facilitado la revisión y validación continua del software.

Con estas pruebas de regresión, se ha logrado una validación completa y exhaustiva del software, demostrando que el sistema cumple con todos los requisitos especificados y funciona correctamente en diversos escenarios y condiciones. Este enfoque ha permitido detectar y corregir errores de manera eficiente, asegurando un producto final de alta calidad y confiabilidad.

Ampliaciones

Para ampliar las capacidades de este proyecto y mejorar aún más el proceso de desarrollo y validación del software, se propone la implementación de Jenkins en combinación con Git. Esta integración permitirá establecer un entorno de integración continua (CI) y entrega continua (CD), proporcionando los siguientes beneficios:

Automatización del Proceso de Pruebas:

Jenkins podrá ejecutar automáticamente las pruebas de regresión cada vez que se realice un cambio en el código fuente almacenado en Git. Esto asegurará una detección temprana de errores y facilitará la corrección inmediata de los mismos.

Generación Automática de Informes:

Los resultados de las pruebas ejecutadas por Jenkins podrán ser automáticamente documentados y almacenados, proporcionando informes detallados de cada ciclo de pruebas. Esto mejorará la visibilidad del estado del proyecto y la trazabilidad de los cambios.

Despliegue Automatizado:

Jenkins puede ser configurado para automatizar el despliegue del software en diferentes entornos (desarrollo, pruebas, producción) tras la validación exitosa de las pruebas. Esto acelerará el ciclo de vida del desarrollo y permitirá entregas más rápidas y frecuentes.

Monitoreo y Notificaciones:

La integración de Jenkins permitirá monitorear continuamente el estado del software y enviar notificaciones automáticas al equipo de desarrollo en caso de fallos o problemas detectados, mejorando la comunicación y coordinación del equipo.

Implementar Jenkins con Git no solo optimizará el proceso de desarrollo y validación, sino que también contribuirá a mantener la calidad y estabilidad del software a lo largo del tiempo, adaptándose a los cambios y nuevas necesidades de manera eficiente y efectiva.

9 Bibliografía

- Universidad Europea. (2022, 23 marzo). *Agile vs. Waterfall: qué diferencias hay entre ambas metodologías*. <https://universidadeuropea.com/blog/agile-vs-waterfall/>
- *Spring boot*. (s. f.). Spring Boot. <https://spring.io/projects/spring-boot>
- Postman API Platform*. (s. f.). <https://www.postman.com/>
- ChatGPT*. (s. f.). ChatGPT. <https://chatgpt.com/>
- Jenkins*. (s. f.). Jenkins. <https://www.jenkins.io/>
- Documentación interna de Eviden

EVIDEN