13 DE DICIEMBRE DE 2021 - [COM 06]

# Testing I

# Examen integrador

Les pedimos que lean atentamente las siguientes consignas y respondan a las preguntas de acuerdo a lo solicitado.

No se aceptarán links de Drive, solo documentos adjuntos. Caso contrario, el examen no será considerado para su corrección.

Nota aclaratoria: al enviar el formulario con el adjunto se debe esperar la confirmación del profesor **antes de salir de la sala de Zoom** para garantizar que se recibió correctamente para posterior corrección. Caso contrario, no se recibirá la evaluación y el alumno deberá recuperar esta instancia de evaluación. **Solo se recibirá 1 (un) documento por alumno**.

**Duración**: 1 hora 30 minutos.

**Nombre y Apellido:** Lara Daniela Converso

Parte teórica

1. ¿Cuál es la diferencia entre una **suite de humo** y una **suite de regresión**?

(Sugerencia: utilizar 2 a 5 renglones/líneas)

Ambas pruebas son muy importantes, pero la suite de humo tiene como objetico verificar la estabilidad del sistema, probando las funcionalidades principales de este y se suelen ejecutar en las etapas iniciales.

En cambio las pruebas de regresión tienen como objetivo testear el buen funcionamiento del sistema cada vez que se terminan de hacer modificaciones, suelen ser mas exhaustivas que las de humo y se ejecutan periódicamente.

1. Explicar la técnica de prueba de caja blanca.

(Sugerencia: utilizar 2 a 5 renglones/líneas)

La técnica de caja blanca consiste en probar el sistema sabiendo como es la estructura interna de este, es decir el código y el funcionamiento de este.

1. Mencionar 2 diferencias entre **debugging** y **testing**.

(Sugerencia: utilizar 2 a 5 renglones/líneas)

Debugging y testing son dos conceptos diferentes, durante el primero se buscan los errores en el código con fin de repararlos; Esta tarea la realiza generalmente los desarrolladores. En cambio, el testing se encarga de buscar los errores y reportarlos para que sean reparados, esta tarea la suele hacer un equipo de QA.

1. ¿En qué ambiente aplicarías **pruebas unitarias**?

(Sugerencia: utilizar 2 a 5 renglones/líneas)

Las pruebas unitarias se aplican en el ambiente de desarrollo (DEV), dado a que es conveniente testear los componentes luego de implementarlos, para asegurar su buen funcionamiento.

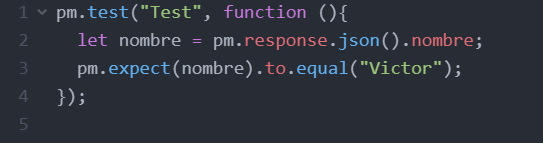
1. Explicar la etapa de planificación dentro del ciclo de vida de las pruebas de software.

(Sugerencia: utilizar 2 a 5 renglones/líneas)

En la etapa de planificación se genera un plan de pruebas, es decir que se definen los enfoques, el alcance, los objetivos a cumplir a parte de evaluar los riesgos y contratiempos que pueden llegar a surgir durante

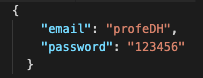
**Parte práctica**

1. ¿Qué nos permite validar el siguiente **test de Postman**?



Este test de Postman nos permite validar que el nombre obtenido de un JSON desde el campo nombre es “Victor”.

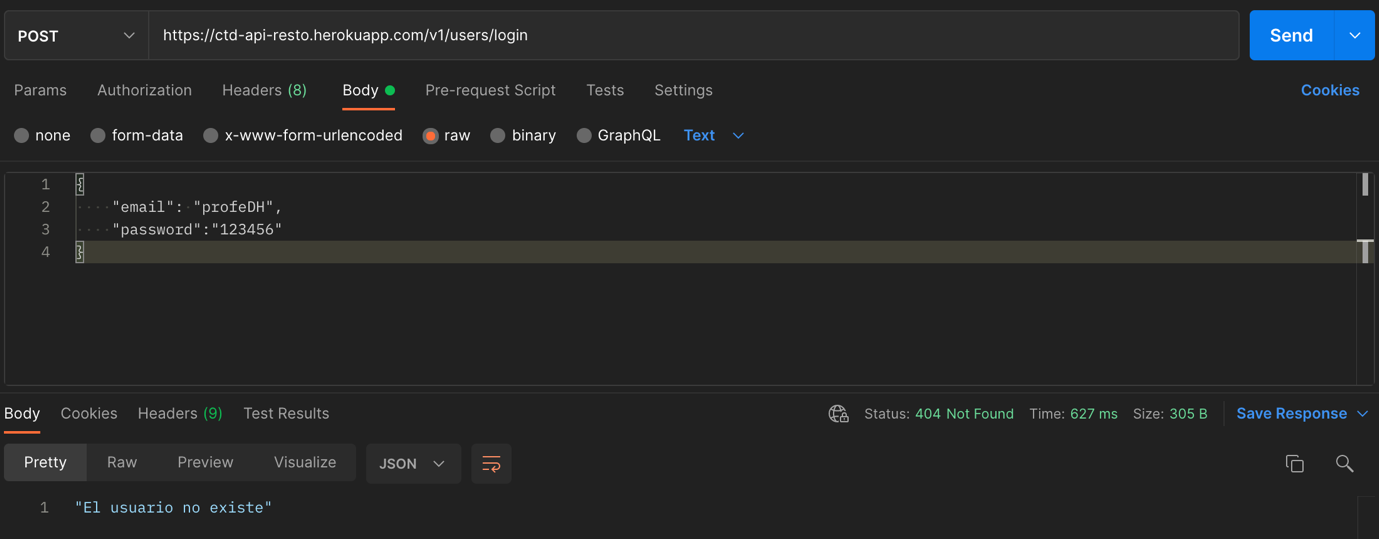
1. Continuamos trabajando con nuestra app **Comida Ya!** Esta se conecta con un servicio back-end. Si realizo una petición **POST en Postman** a la siguiente URL: <https://ctd-api-resto.herokuapp.com/v1/users/login> con los siguientes datos:



¿Qué resultado arroja y por qué? ¿Qué significa el código de respuesta devuelto?

Nos arroja un resultado “404 Not found”, que significa que el servidor web no encuentra el resultado solicitado, junto con el texto “El usuario no existe”.

Adjunto imagen de la prueba:



1. Detallar **1 caso de prueba** que aplicarías en la página de [Comida Ya!](https://ctd-app-resto.herokuapp.com/), solo explicando su descripción (**no utilizar el template**).

[Login] Teniendo un usuario ya registrado en la aplicación, al iniciar sesión con su respectivo mail y contraseña debería poder acceder al sitio sin problemas.

1. Mencionar **1 defecto** que encuentres en la página de [Comida Ya!](https://ctd-app-resto.herokuapp.com/) (**no utilizar el template**).

[Productos] No se visualizan todos los productos con su imagen correspondiente, ni tampoco con precios validos.

1. Si estoy trabajando con Jest y quiero validar que el resultado devuelto sea **false**. ¿Qué matcher puedo utilizar? Dar un ejemplo de un posible test.

Se podria usar el matcher **“.toBeFalsy()”** que nos arroja como resultado

Ejemplo:

const miMascota={

esPerro: true

esGato: false

};

test(“Mi mascota no es un gato”, ()=>{

expect(miMascota.esGato).toBeFalsy();

});

La prueba “Mi mascota no es un gato” seria exitosa.