DESARROLLO DE SOFTWARE

EG2. Práctica del Hotel

Curso 2023/2024

Jorge Mejías Donoso 100495807

Javier Moyano San Bruno 100495884

Fernando López de Olmedo Rodríguez-Solano 100495977

Índice de contenidos

1. Introducción

En el contexto del desarrollo de software, la realización de pruebas unitarias desempeña un papel fundamental en la garantía de la calidad del producto final. En este documento, se presenta el análisis y la definición de casos de prueba para un componente específico: el sistema de gestión de reservas y estancias en un hotel. Este componente es crucial para asegurar una experiencia óptima tanto para los clientes como para el personal del hotel.

El objetivo principal de este documento es establecer un conjunto exhaustivo de casos de prueba que abarquen todas las funcionalidades clave del sistema, desde la reserva inicial de habitaciones hasta el registro de la salida de los huéspedes. Para ello, se emplearán técnicas y enfoques de prueba tanto funcionales como estructurales, siguiendo los principios de desarrollo dirigido por pruebas (TDD) y las mejores prácticas de la industria.

En la sección siguiente, se detallarán los requisitos y funcionalidades del sistema de gestión de reservas y estancias en el hotel, proporcionando así un contexto claro para la definición de los casos de prueba. Posteriormente, se abordarán los aspectos clave de la elaboración de casos de prueba, incluyendo la identificación de clases de equivalencia, valores límite, gramática, árboles de derivación, gráficos de control de flujo y rutas básicas.

La calidad y exhaustividad de los casos de prueba son fundamentales para asegurar que el sistema cumpla con los requisitos funcionales y no funcionales establecidos, así como para detectar y corregir posibles defectos de manera temprana en el ciclo de desarrollo. Por tanto, este documento servirá como guía integral para la ejecución de pruebas unitarias efectivas y la validación del sistema de gestión de reservas y estancias en el hotel.

2. Función 1. Solicitar una reserva de hotel

Vamos a ir explicando poco a poco los tests, las clases de equivalencia y los valores límites que hemos indicado en el excel y en el archivo unittest. Para ello vamos a dividir cada uno de los atributos que hay que probar de la función:

Credit Card:

Clase de Equivalencia Valida (CEV1):

Esta clase de equivalencia representa números de tarjeta de crédito válidos, es decir, aquellos que cumplen con el algoritmo de Luhn y tienen 16 dígitos. Se incluye para asegurar que la función maneje correctamente las tarjetas de crédito válidas, lo que es fundamental para su correcto funcionamiento en situaciones normales.

Clase de Equivalencia No Valida (CENV1, CENV2, CENV3, CENV4, CENV5):

Estas clases de equivalencia representan diferentes escenarios de números de tarjeta de crédito no válidos. Cubren casos como argumentos vacíos, números no aceptados por el algoritmo de Luhn, presencia de caracteres no numéricos y longitudes inválidas. Es importante probar estos casos para garantizar que la función maneje adecuadamente las situaciones de entrada incorrecta y proporcione mensajes de error claros y precisos.

Valor Límite Valido (VLV1):

Este valor límite representa números de tarjeta de crédito con la longitud máxima permitida, es decir, 16 dígitos. Se incluye para verificar si la función puede procesar correctamente la entrada con el límite superior de longitud, lo que es crítico para asegurar que no haya errores debido a la longitud incorrecta de la entrada.

Valores Límite No Validos (VLNV1, VLVN2):

Estos valores límite representan situaciones extremas donde la longitud de la tarjeta de crédito excede el límite superior o es menor que el límite inferior. Prueban la capacidad de la función para manejar entradas que se salen de los límites aceptables y verificar si se proporcionan mensajes de error adecuados en tales casos.

TC2 Argumento vacío (CENV1):

Este caso de prueba valida cómo maneja la función la entrada cuando se proporciona un argumento vacío. Esperamos que la función devuelva un error indicando que el número de tarjeta de crédito es inválido.

TC3 Tarjeta de crédito inválida (CENV2):

Aquí evaluamos cómo se comporta la función cuando se proporciona una tarjeta de crédito que no es válida según el algoritmo de Luhn. El resultado esperado es un mensaje de error indicando que el número de tarjeta de crédito es inválido.

TC4 No todos son números (CENV3):

En este caso, estamos probando la capacidad de la función para manejar la entrada que contiene caracteres no numéricos. Esperamos que la función devuelva un mensaje de error indicando que el número de tarjeta de crédito es inválido.

TC5 17 dígitos (CENV4-VLNV1):

Este valor límite evalúa cómo se maneja la entrada cuando se proporciona un número de tarjeta de crédito con más de 16 dígitos. El resultado esperado es un mensaje de error indicando que el número de tarjeta de crédito es inválido.

TC6 15 dígitos (CENV5-VLNV2):

Similar al caso anterior, aquí estamos probando el comportamiento de la función cuando se proporciona un número de tarjeta de crédito con menos de 16 dígitos. Esperamos que

la función devuelva un mensaje de error indicando que el número de tarjeta de crédito es inválido.

idCard:
Clase de Equivalencia Valida (CEV2):

Esta clase de equivalencia representa números de DNI válidos, es decir, aquellos que tienen exactamente 8 dígitos. Se incluye para asegurar que la función maneje correctamente los DNIs válidos, lo que es esencial para su correcto funcionamiento en situaciones normales.

Clase de Equivalencia No Valida (CENV6, CENV7, CENV8, CENV9, CENV10, CENV11, CENV12, CENV13, CENV14):

Estas clases de equivalencia representan diferentes escenarios de DNIs no válidos. Cubren casos como argumentos vacíos, DNIs con formatos incorrectos, DNIs con más o menos de 8 dígitos, y DNIs con caracteres no numéricos. Es importante probar estos casos para garantizar que la función maneje adecuadamente las situaciones de entrada incorrecta y proporcione mensajes de error claros y precisos.

Valor Límite Valido (VLV2, VLV3):

Estos valores límite representan situaciones específicas de DNIs válidos. VLV2 representa DNIs con exactamente 8 dígitos, mientras que VLV3 representa DNIs con exactamente 1 letra. Estos casos de prueba aseguran que la función pueda procesar correctamente los DNIs que cumplen con estas condiciones específicas.

Valores Límite No Validos (VLNV3, VLVN4, VLVN5, VLVN6, VLVN7, VLVN8):

Estos valores límite representan situaciones extremas donde el DNI excede o no cumple con las condiciones normales. VLVN3 representa DNIs con más de 9 dígitos, VLVN4 representa DNIs con 7 dígitos, VLVN5 representa DNIs sin letras, VLVN6 representa DNIs con 2 letras, VLVN7 representa DNIs con 10 caracteres y VLVN8 representa DNIs con 8 caracteres. Estos casos de prueba evalúan la capacidad de la función para manejar entradas que se salen de los límites aceptables y verificar si se proporcionan mensajes de error adecuados en tales casos.

TC7 Argumento vacío (CENV6):

Este caso de prueba valida cómo maneja la función la entrada cuando se proporciona un argumento vacío como DNI. Esperamos que la función devuelva un error indicando que el DNI es inválido.

TC8 DNI incorrecto (CENV7):

Aquí evaluamos cómo se comporta la función cuando se proporciona un DNI que no es válido según los criterios definidos. El resultado esperado es un mensaje de error indicando que el DNI es inválido.

TC9 Ya tiene reserva (CENV8):

Este caso de prueba evalúa cómo se maneja la situación cuando se intenta hacer una reserva con un DNI que ya tiene una reserva asociada. Esperamos que la función devuelva un mensaje de error indicando que el cliente ya tiene una reserva.

TC10 Más caracteres (CENV9, CENV10, CENV13, VLVN3, VLVN6):

En este caso, estamos probando la capacidad de la función para manejar un DNI con más caracteres de los permitidos, incluyendo letras. Esperamos que la función devuelva un mensaje de error indicando que el DNI es inválido debido a la longitud excesiva y la presencia de letras.

TC11 Menos caracteres (CENV11, CENV12, CENV14, VLVN4, VLVN5):

Similar al caso anterior, aquí estamos probando el comportamiento de la función cuando se proporciona un DNI con menos caracteres de los permitidos. Esperamos que la función devuelva un mensaje de error indicando que el DNI es inválido debido a la longitud insuficiente y la falta de letras.

nameSurname:

Clase de Equivalencia Valida (CEV3):

Esta clase de equivalencia representa nombres y apellidos válidos, es decir, cadenas de texto que cumplen con ciertos criterios de longitud y estructura. Se incluye para asegurar que la función maneje correctamente los nombres y apellidos válidos, lo que es esencial para su correcto funcionamiento en situaciones normales.

Clase de Equivalencia No Valida (CENV15, CENV16, CENV17, CENV18):

Estas clases de equivalencia representan diferentes escenarios de nombres y apellidos no válidos. Cubren casos como argumentos vacíos, cadenas con menos de 2 bloques de caracteres, cadenas con menos de 10 caracteres y cadenas con más de 50 caracteres. Es importante probar estos casos para garantizar que la función maneje adecuadamente las situaciones de entrada incorrecta y proporcione mensajes de error claros y precisos.

Valor Límite Valido (VLV4, VLV5, VLV6, VLV7, VLV8):

Estos valores límite representan situaciones específicas de nombres y apellidos válidos. VLV4 representa cadenas de 10 caracteres, VLV5 representa cadenas de 11 caracteres, VLV6 representa cadenas de 49 caracteres, VLV7 representa cadenas de 50 caracteres y VLV8 representa cadenas con exactamente 2 bloques de caracteres. Estos casos de prueba aseguran que la función pueda procesar correctamente las cadenas que cumplen con estas condiciones específicas.

Valores Límite No Validos (VLNV9, VLVN10, VLVN11):

Estos valores límite representan situaciones extremas donde las cadenas de nombres y apellidos no cumplen con los criterios de longitud o estructura esperados. VLNV9 representa una cadena de caracteres única, VLVN10 representa cadenas con 9 caracteres y VLVN11 representa cadenas con 51 caracteres. Estos casos de prueba evalúan la

capacidad de la función para manejar entradas que se salen de los límites aceptables y verificar si se proporcionan mensajes de error adecuados en tales casos.

TC12 Argumento vacío (CENV15):

Este caso de prueba valida cómo maneja la función la entrada cuando se proporciona un argumento vacío como nombre y apellidos. Esperamos que la función devuelva un error indicando que la cadena del nombre y apellidos no es válida.

TC13 1 cadena de texto (CENV16, VLNV9):

Aquí evaluamos cómo se comporta la función cuando se proporciona una cadena de texto que no cumple con la estructura esperada (menos de 2 bloques de caracteres). El resultado esperado es un mensaje de error indicando que la cadena del nombre y apellidos no es válida.

TC14 Cadena demasiado corta (CENV17):

En este caso, estamos probando la capacidad de la función para manejar una cadena de nombre y apellidos que es demasiado corta. Esperamos que la función devuelva un mensaje de error indicando que la cadena del nombre y apellidos no es válida debido a la longitud insuficiente.

TC15 Cadena demasiado larga (CENV18):

Similar al caso anterior, aquí estamos probando el comportamiento de la función cuando se proporciona una cadena de nombre y apellidos que es demasiado larga. Esperamos que la función devuelva un mensaje de error indicando que la cadena del nombre y apellidos no es válida debido a la longitud excesiva.

phoneNumber:

Clase de Equivalencia Valida (CEV4):

Esta clase de equivalencia representa números de teléfono válidos, es decir, cadenas de 9 dígitos. Se incluye para asegurar que la función maneje correctamente los números de teléfono válidos, lo que es esencial para su correcto funcionamiento en situaciones normales.

Clase de Equivalencia No Valida (CENV19, CENV20, CENV21, CENV22):

Estas clases de equivalencia representan diferentes escenarios de números de teléfono no válidos. Cubren casos como argumentos vacíos, cadenas que contienen caracteres no numéricos, cadenas con más de 9 dígitos y cadenas con menos de 9 dígitos. Es importante probar estos casos para garantizar que la función maneje adecuadamente las situaciones de entrada incorrecta y proporcione mensajes de error claros y precisos.

Valor Límite Valido (VLV9):

Este valor límite representa números de teléfono con exactamente 9 dígitos. Se incluye para verificar si la función puede procesar correctamente la entrada con la longitud

adecuada, lo que es crítico para asegurar que no haya errores debido a la longitud incorrecta de la entrada.

Valores Límite No Validos (VLNV12, VLNV13):

Estos valores límite representan situaciones extremas donde la longitud del número de teléfono excede el límite superior o es menor que el límite inferior. VLNV12 representa números de teléfono con 8 dígitos y VLNV13 representa números de teléfono con 10 dígitos. Prueban la capacidad de la función para manejar entradas que se salen de los límites aceptables y verificar si se proporcionan mensajes de error adecuados en tales casos.

TC16 Argumento vacío (CENV19):

Este caso de prueba valida cómo maneja la función la entrada cuando se proporciona un argumento vacío como número de teléfono. Esperamos que la función devuelva un error indicando que el número de teléfono es inválido.

TC17 Demasiado largo y con letras (CENV20, CENV21):

Aquí evaluamos cómo se comporta la función cuando se proporciona un número de teléfono que contiene caracteres no numéricos o tiene más de 9 dígitos. El resultado esperado es un mensaje de error indicando que el número de teléfono es inválido debido a la longitud incorrecta o la presencia de caracteres no numéricos.

TC18 Demasiado corto (CENV22):

En este caso, estamos probando la capacidad de la función para manejar un número de teléfono que tiene menos de 9 dígitos. Esperamos que la función devuelva un mensaje de error indicando que el número de teléfono es inválido debido a la longitud insuficiente.

roomType:
Clase de Equivalencia Valida (CEV5, CEV6, CEV7):

Estas clases de equivalencia representan los diferentes tipos de habitaciones disponibles en el hotel: single, double y suite. Se incluyen para asegurar que la función maneje correctamente los tipos de habitaciones válidos, lo que es esencial para su correcto funcionamiento en situaciones normales.

Clase de Equivalencia No Valida (CENV23, CENV24, CENV25):

Estas clases de equivalencia representan diferentes escenarios de tipos de habitaciones no válidos. Cubren casos como argumentos vacíos, cadenas que no corresponden a las opciones válidas y la presencia de más de una cadena válida a la vez. Es importante probar estos casos para garantizar que la función maneje adecuadamente las situaciones de entrada incorrecta y proporcione mensajes de error claros y precisos.

TC19 Argumento vacío (CENV23):

Este caso de prueba valida cómo maneja la función la entrada cuando se proporciona un argumento vacío como tipo de habitación. Esperamos que la función devuelva un error indicando que el tipo de habitación es inválido.

TC20 Cadena diferente a opciones (CENV24):

Aquí evaluamos cómo se comporta la función cuando se proporciona un tipo de habitación que no corresponde a ninguna de las opciones válidas (single, double o suite). El resultado esperado es un mensaje de error indicando que el tipo de habitación es inválido debido a la cadena diferente a las opciones válidas.

TC21 Repetición de cadena aceptada (CENV25):

En este caso, estamos probando la capacidad de la función para manejar la entrada cuando se proporcionan más de una cadena válida a la vez. Esperamos que la función devuelva un mensaje de error indicando que el tipo de habitación es inválido debido a la presencia de más de una cadena válida a la vez.

Arrival:
Clase de Equivalencia Valida (CEV9, CEV10, CEV11):

Estas clases de equivalencia representan diferentes aspectos de fechas de llegada válidas al hotel. CEV9 abarca fechas que respetan el formato y son posteriores a la fecha actual. CEV10 y CEV11 se refieren al mes y día de la fecha, respectivamente, asegurando que estén dentro de los rangos válidos (mes entre 1 y 12, día entre 1 y 31).

Clase de Equivalencia No Valida (CENV26, CENV27, CENV28, CENV29, CENV30, CENV31, CENV32):

Estas clases de equivalencia representan diferentes escenarios de fechas de llegada no válidas. CENV26 cubre el caso de argumento vacío, mientras que CENV27 y CENV28 se refieren a formatos incorrectos y fechas anteriores a la actual, respectivamente. CENV29 y CENV30 representan meses fuera del rango permitido (superior a 12 o inferior a 1), mientras que CENV31 y CENV32 abordan días fuera del rango permitido (superior a 31 o inferior a 1).

Valor Límite Valido (VLV13, VLV14, VLV15, VLV16, VLV17, VLV18, VLV19, VLV20):

Estos valores límite representan situaciones extremas de fechas de llegada válidas. Se incluyen para verificar si la función puede procesar correctamente la entrada con diferentes valores límite, como el primer día del mes, el último día del mes y fechas fuera de los límites del calendario.

Valor Límite No Valido (VLNV14, VLVN15, VLVN16, VLVN17):

Estos valores límite representan situaciones extremas de fechas de llegada no válidas, como meses fuera del rango permitido (0 y 13) y días fuera del rango permitido (0 y 32).

TC22 Argumento vacío (CENV26):

Este caso de prueba valida cómo maneja la función la entrada cuando se proporciona un argumento vacío como fecha de llegada. Esperamos que la función devuelva un error indicando que el formato de la fecha de llegada es inválido.

TC23 No respeta el formato (CENV27):

Aquí evaluamos cómo se comporta la función cuando se proporciona una fecha de llegada que no respeta el formato dd/mm/yyyy. El resultado esperado es un mensaje de error indicando que el formato de la fecha de llegada es inválido.

TC24 Fecha antigua (CENV28):

Este caso de prueba verifica cómo maneja la función la entrada cuando se proporciona una fecha de llegada que es anterior a la fecha actual. Esperamos que la función devuelva un error indicando que la fecha de llegada debe ser posterior a la fecha actual.

TC25, TC26 Día y mes inválidos (CENV29, CENV30, CENV31, CENV32, VLVN14, VLVN15, VLVN16, VLVN17):

Estos casos de prueba evalúan la capacidad de la función para manejar fechas de llegada con días y meses inválidos, tales como días o meses fuera de los rangos permitidos. Esperamos que la función devuelva un mensaje de error indicando que el formato de la fecha de llegada es inválido.

numDays:

Clase de Equivalencia Valida (CEV8):

Esta clase de equivalencia representa el número de días válidos para la reserva, asegurando que esté dentro del rango permitido de 1 a 10 días. Es fundamental para garantizar que el sistema pueda manejar correctamente la duración de la reserva dentro de los límites aceptables.

Clase de Equivalencia No Valida (CENV33, CENV34, CENV35, CENV36):

Estas clases de equivalencia representan diferentes escenarios de números de días de reserva no válidos. CENV33 abarca el caso de un argumento vacío, mientras que CENV34 se refiere a entradas que no son números. CENV35 y CENV36 abordan situaciones en las que el número de días excede el límite superior o es inferior al límite inferior permitido.

Valor Límite Valido (VLV10, VLV11, VLV12):

Estos valores límite representan situaciones extremas de números de días válidos para la reserva. Se incluyen para verificar si la función puede procesar correctamente la entrada con diferentes valores límite, como 1, 2 y 10 días de reserva.

Valor Límite No Valido (VLNV18, VLVN19):

Estos valores límite representan situaciones extremas de números de días no válidos para la reserva, como 0 días y 11 días. Prueban la capacidad de la función para manejar entradas que se salen de los límites aceptables.

TC27 Argumento vacío (CENV33):

Este caso de prueba valida cómo maneja la función la entrada cuando se proporciona un argumento vacío como el número de días de reserva. Esperamos que la función devuelva un error indicando que el número de días de reserva es inválido.

TC28 No es un número (CENV34):

Aquí evaluamos cómo se comporta la función cuando se proporciona un valor que no es un número como el número de días de reserva. El resultado esperado es un mensaje de error indicando que el número de días de reserva es inválido.

TC29, TC30 Número de días fuera del rango (CENV35, CENV36, VLNV18, VLVN19):

Estos casos de prueba verifican cómo maneja la función la entrada cuando se proporciona un número de días de reserva que está fuera del rango permitido. Esperamos que la función devuelva un mensaje de error indicando que el número de días de reserva es inválido y está fuera del rango aceptable.

3. Función 2. Llegada al Hotel

Para la implementación y prueba del segundo método, adoptaremos la técnica del Análisis Sintáctico como enfoque principal. Esta técnica se centra en evaluar la estructura y el flujo de datos dentro del método, lo que nos permitirá identificar y corregir posibles errores de lógica y sintaxis.

Nuestro objetivo es garantizar que el segundo método funcione según lo esperado en una variedad de situaciones, brindando resultados precisos y coherentes. Al documentar cuidadosamente nuestros casos de prueba en un archivo Excel, aseguraremos una comprensión clara y completa de las pruebas realizadas y los resultados obtenidos.

Esta es la gramática que usaremos para crear el árbol de derivación y posteriormente rellenar el excel con los posibles casos sobre sus nodos:

fichero -> INI_OBJ DATOS FIN_OBJ

ini_obj -> {

fin_obj -> }

datos -> CAMPO1 SEPARADOR CAMPO2

CAMPO1 -> etiqueta_dato1 igualdad valor_dato1

CAMPO2 -> etiqueta_dato2 igualdad valor_dato2

separador -> ,

igualdad -> :

comillas -> ""

etiqueta_dato1 -> comillas Valor_etiqueta1 comillas

valor_etiqueta1 -> localizer

valor_dato1 -> comillas valor_1 comillas

valor1 -> a|b|c|d|e|f|0|1|2|3|4|5|6|7|8|9 {32}

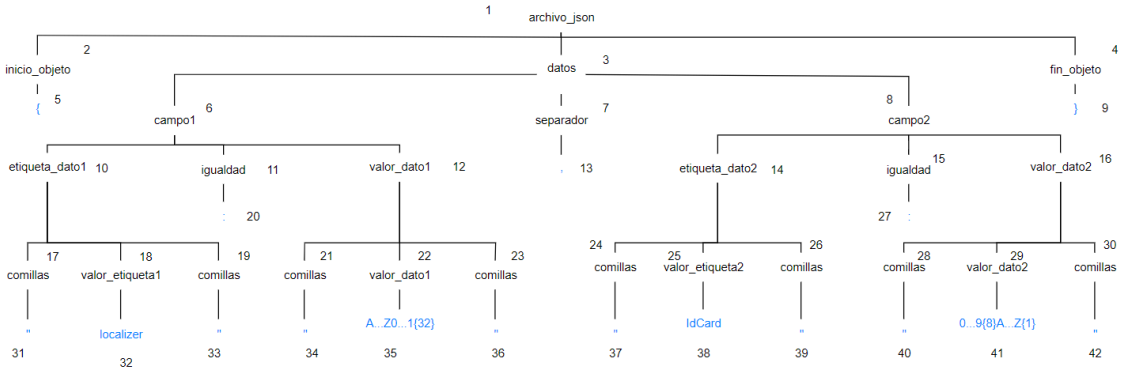
etiqueta_dato2 -> comillas Valor_etiqueta2 comillas

valor_etiqueta2 -> idCard

valor_dato2 -> comillas valor_2 comillas

valor2 -> 0-9{8}a-z{1}

Y este es el diagrama del árbol de derivación:



Esta el la explicación de todos los casos de test creados para comprobar nuestro código:

test1_valido:

Este caso comprueba que pasa cuando todos los datos que se le pasan son válidos y el código hace lo que debería hacer. No se genera ningún error y devuelve verdadero (True).

test2_no_existe_archivo:

Este caso comprueba el comportamiento cuando se intenta acceder a un archivo que no existe. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no existe".

test3_invalido_formato_json:

Este caso evalúa la respuesta del programa cuando se proporciona un archivo con un formato no válido de JSON. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test4_json_vacio_sin_llaves:

Comprueba la reacción del programa ante un archivo JSON vacío sin llaves. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test5_json_vacio_con_llaves:

Examina cómo el programa responde ante un archivo JSON vacío con llaves. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test6_invalido_json_estructure_localizer:

Evalúa la respuesta del programa ante un archivo JSON con una estructura incorrecta, en este caso, sin el campo "IdCard". Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test7_invalido_json_estructure_localizer_nombre:

Similar al caso anterior, pero con un campo diferente ("Invalid_localizer" en lugar de "Localizer"). Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test8_invalido_json_estructure_localizer_num:

Otra variante del caso anterior, pero esta vez con un campo diferente ("Invalid_localizer" en lugar de "Localizer"). Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test9_invalido_json_estructure_idCard:

Examina la respuesta del programa ante un archivo JSON con una estructura incorrecta, esta vez sin el campo "Localizer". Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test10_invalido_json_estructure_idCard_num:

Similar al caso anterior, pero con un campo diferente ("Invalid_IdCard" en lugar de "IdCard"). Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test11_invalido_json_estructure_idCard_num:

Este caso verifica la respuesta del programa cuando se proporciona un archivo JSON con una estructura incorrecta, con un campo de "IdCard" inválido. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test12_invalido_localizer:

Evalúa la respuesta del programa cuando se proporciona un localizador inválido que no figura en el archivo de reservas. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El localizador o el DNI introducido no se corresponde con los datos almacenados".

test13_invalido_localizer_33:

Similar al caso anterior, pero con un localizador más largo. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El localizador o el DNI introducido no se corresponde con los datos almacenados".

test14_invalido_localizer_31:

Similar al caso anterior, pero con un localizador más corto. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El localizador o el DNI introducido no se corresponde con los datos almacenados".

test15_invalido_dni:

Verifica la respuesta del programa cuando se proporciona un DNI inválido en el archivo JSON. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test16_invalido_dni_sin_letra:

Similar al caso anterior, pero con un DNI que carece de la letra final. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test17_invalido_dni_solo_nums:

Similar al caso anterior, pero con un DNI que solo contiene números. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test18_invalido_dni_corto:

Similar al caso anterior, pero con un DNI demasiado corto. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test19_invalido_dni_excede_letra:

Similar al caso anterior, pero con un DNI que excede una letra al final. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test20_invalido_dni_excede_num:

Similar al caso anterior, pero con un DNI que excede un número al final. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test21_localizer_corto:

Este caso evalúa la respuesta del programa cuando se proporciona un localizador que es demasiado corto. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test22_localizer_vacio:

Verifica la respuesta del programa cuando se proporciona un localizador vacío. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test23_dni_vacio:

Evaluación de la respuesta del programa cuando se proporciona un DNI vacío en el archivo JSON. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test24_hora_equivocada:

Este caso comprueba la respuesta del programa cuando la fecha de llegada en el archivo JSON no coincide con la fecha actual. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "La fecha de llegada no coincide con la fecha actual".

test25_invalido_localizer_x2:

Verifica la respuesta del programa cuando se proporciona dos veces el campo "Localizer" en el archivo JSON. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test26_invalido_idCard_x2:

Similar al caso anterior, pero con dos veces el campo "IdCard". Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test27_localizer_caracteres_invalidos_inicio:

Evalúa la respuesta del programa cuando el localizador comienza con caracteres inválidos. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test28_localizer_caracteres_invalidos_mitad:

Similar al caso anterior, pero con caracteres inválidos en la mitad del localizador. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test29_localizer_caracteres_invalidos_fin:

Similar al caso anterior, pero con caracteres inválidos al final del localizador. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test30_dni_caracteres_invalidos_ini:

Evalúa la respuesta del programa cuando el DNI comienza con caracteres inválidos. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test31_dni_caracteres_invalidos_mitad:

Similar al caso anterior, pero con caracteres inválidos en la mitad del DNI. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

test32_dni_caracteres_invalidos_fin:

Similar al caso anterior, pero con caracteres inválidos al final del DNI. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "Los datos del JSON no tienen valores válidos".

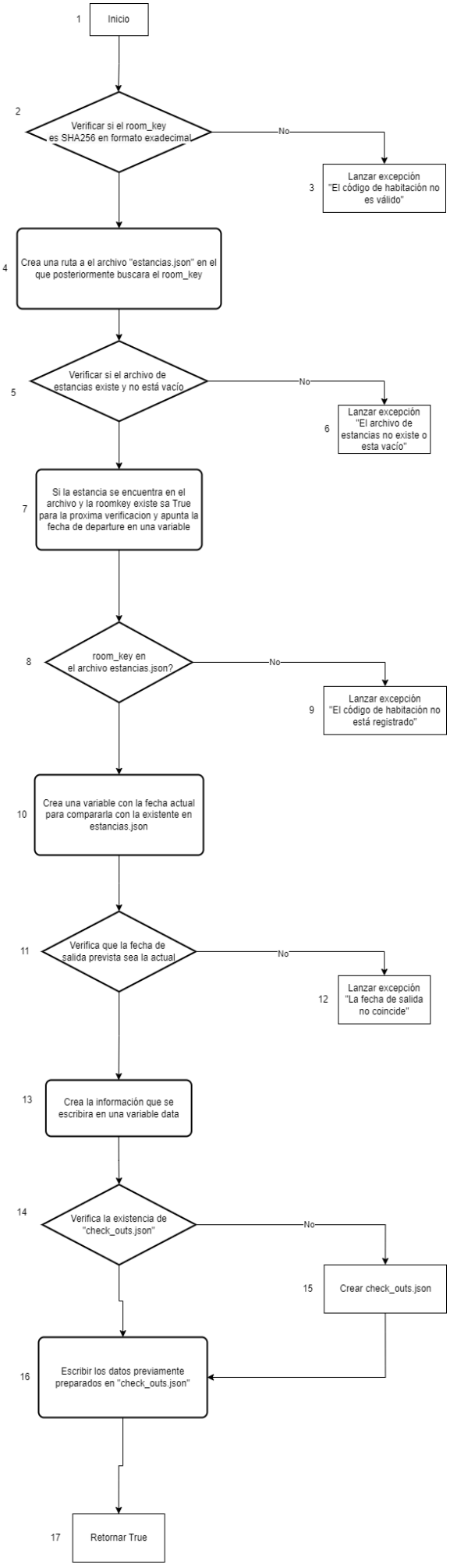
test33_invalido_json_estructure_DNI:

Verifica la respuesta del programa cuando se proporciona un archivo JSON con un campo incorrecto "IdCard". Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "El archivo no tiene formato JSON".

test34_error_tiempo:

Este caso evalúa la respuesta del programa cuando se proporciona un archivo JSON con un campo de "IdCard" repetido. Se espera que se genere una excepción del tipo HotelManagementException con el mensaje "La fecha de llegada no coincide con la fecha actual".

4. Función 3. CheckOut


1
Inicio
2
Verificar si el room_key es SHA256 en formato exadecimal
No
3
Lanzar excepción "El código de habitación no es válido"
4
Crea una ruta a el archivo "estancias.json" en el que posteriormente buscara el room_key
5
Verificar si el archivo de estancias existe y no está vacío
No
6
Lanzar excepción "El archivo de estancias no existe o esta vacío"
7
Si la estancia se encuentra en el archivo y la roomkey existe sa True para la proxima verificacion y apunta la fecha de departure en una variable
8
room_key en el archivo estancias.json?
No
9
Lanzar excepción "El código de habitación no está registrado"
10
Crea una variable con la fecha actual para compararla con la existente en estancias.json
11
Verifica que la fecha de salida prevista sea la actual
No
12
Lanzar excepción "La fecha de salida no coincide"
13
Crea la información que se escribira en una variable data
14
Verifica la existencia de "check_outs.json"
No
15
Crear check_outs.json
16
Escribir los datos previamente preparados en "check_outs.json"
17
Retornar True

Verificaciones

Exactitud del código de habitación: Se verifica que el código de habitación recibido sea una cadena y tenga una longitud de 64 caracteres. Si no cumple con estas condiciones, se levanta una excepción HotelManagementException con el mensaje "El código de habitación no es válido".
Existencia y tamaño del archivo de estancias: Se verifica si el archivo de estancias existe o está vacío. Si no cumple con estas condiciones, se levanta una excepción HotelManagementException con el mensaje "El archivo de estancias no existe o está vacío".
Búsqueda del código de habitación en el archivo de estancias: Se busca el código de habitación en el archivo de estancias JSON. Si no se encuentra el código de habitación, se levanta una excepción HotelManagementException con el mensaje "El código de habitación no está registrado".
Verificación de la fecha prevista de salida: Se verifica que la fecha actual coincida con la fecha de salida prevista. Si no coincide, se levanta una excepción HotelManagementException con el mensaje "La fecha de salida no coincide".

Procesos

Registro de salida del cliente: Se crea un diccionario salida_data con el código de habitación y la fecha de salida actual. Luego, se guarda este diccionario en un archivo de check-outs JSON.

Resultados

- Si todos los procesos se ejecutan correctamente, la función devuelve True.
- Si se produce algún error durante el proceso, se levanta una excepción del tipo HotelManagementException.

Configuración Inicial

Antes de ejecutar las pruebas, se realiza una configuración inicial en el método setUp de la clase TestGuestCheckout. Este método se encarga de preparar el entorno eliminando los archivos de estancias y check-outs si existen, y luego crea un archivo de estancias JSON con una única reserva de ejemplo para utilizar en las pruebas.

Pruebas Realizadas

test_guest_checkout_valid:

- Descripción: Prueba la salida de un cliente del hotel con un código de habitación válido y una fecha de salida correcta.
- Entrada: Código de habitación válido y fecha de salida correcta.
- Salida Esperada: La función guest_checkout devuelve True.

test_guest_checkout_invalid_room_key:

- Descripción: Prueba la salida de un cliente del hotel con un código de habitación inválido.
- Entrada: Código de habitación inválido.
- Salida Esperada: Se espera que se levante una excepción del tipo HotelManagementException con el mensaje "El código de habitación no es válido".

test_guest_checkout_length_room_key:

- Descripción: Prueba la salida de un cliente del hotel con un código de habitación de longitud incorrecta.
- Entrada: Código de habitación con longitud incorrecta.
- Salida Esperada: Se espera que se levante una excepción del tipo HotelManagementException con el mensaje "El código de habitación no es válido".

test_guest_checkout_not_found:

- Descripción: Prueba la salida de un cliente del hotel con un código de habitación que no está registrado en el sistema.
- Entrada: Código de habitación no registrado.
- Salida Esperada: Se espera que se levante una excepción del tipo HotelManagementException con el mensaje "El código de habitación no está registrado".

test_guest_checkout_no_date:

- Descripción: Prueba la salida de un cliente del hotel cuando el archivo de estancias está vacío.
- Entrada: Archivo de estancias vacío.
- Salida Esperada: Se espera que se levante una excepción del tipo HotelManagementException con el mensaje "El archivo de estancias no existe o está vacío".

test_guest_checkout_no_date_2:

- Descripción: Prueba la salida de un cliente del hotel cuando el archivo de estancias no existe.
- Entrada: Archivo de estancias que no existe.
- Salida Esperada: Se espera que se levante una excepción del tipo HotelManagementException con el mensaje "El archivo de estancias no existe o está vacío".

test_guest_checkout_past_date:

- Descripción: Prueba la salida de un cliente del hotel cuando la fecha de salida ya ha pasado.
- Entrada: Código de habitación y fecha de salida con fecha pasada.

- Salida Esperada: Se espera que se levante una excepción del tipo HotelManagementException con el mensaje "La fecha de salida no coincide".

Observaciones

- Se utiliza la librería freezegun para simular fechas específicas durante las pruebas, lo que permite probar escenarios relacionados con el tiempo de manera controlada.
- Se realiza limpieza del entorno de prueba antes y después de ejecutar las pruebas para garantizar la consistencia y evitar interacciones no deseadas entre las pruebas.
- Cada prueba se centra en un caso específico, cubriendo tanto situaciones válidas como situaciones de error que puedan surgir durante la salida de un cliente del hotel.

5. Conclusiones

Durante el proceso de desarrollo de las pruebas unitarias utilizando la metodología TDD (Desarrollo Guiado por Pruebas), hemos enfrentado varios desafíos y aprendido lecciones valiosas. Este enfoque nos ha permitido identificar y corregir errores de manera proactiva, mejorando la calidad y robustez de nuestro código. A través del análisis de clases de equivalencia y valores límite, hemos abordado una amplia gama de escenarios posibles, lo que nos ha llevado a una comprensión más profunda de los requisitos y comportamientos esperados de nuestro sistema.

Lecciones Aprendidas:

Identificación de Casos de Prueba Relevantes: A través del análisis de clases de equivalencia y valores límite, aprendimos a identificar los casos de prueba más relevantes y significativos para validar el comportamiento de nuestra función en una variedad de situaciones.

Resolución de Problemas y Depuración: Enfrentamos desafíos al depurar y resolver errores en nuestros casos de prueba, especialmente cuando surgieron resultados inesperados. Esto nos obligó a profundizar en el código subyacente y comprender mejor su funcionamiento para identificar y corregir los problemas.

Comunicación y Colaboración: Durante el proceso de desarrollo de pruebas unitarias, la comunicación efectiva y la colaboración entre los miembros del equipo fueron fundamentales para abordar los desafíos y encontrar soluciones. Compartir conocimientos y experiencias nos permitió aprender unos de otros y mejorar continuamente nuestro enfoque.

Valor de las Pruebas Unitarias: La práctica de escribir pruebas unitarias nos ha demostrado su valor en la detección temprana de errores y la validación del comportamiento esperado del código. Aunque a veces enfrentamos dificultades para comprender ciertos errores, la persistencia y el análisis cuidadoso nos permitieron superar estos obstáculos y fortalecer nuestra base de pruebas.

En resumen, el proceso de desarrollo guiado por pruebas nos ha brindado una perspectiva única sobre la importancia de las pruebas unitarias en la construcción de software de calidad. Aunque desafiante en ocasiones, esta práctica ha sido invaluable para mejorar nuestra habilidad para escribir código robusto y confiable, y nos ha preparado para enfrentar desafíos futuros con confianza y determinación.