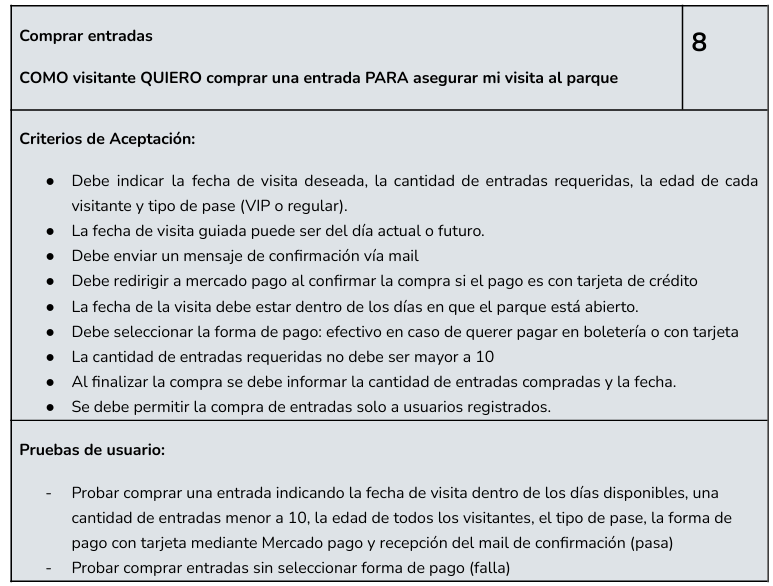
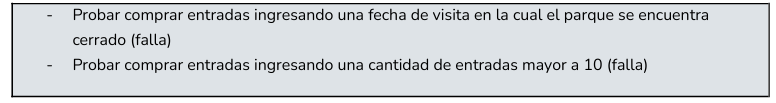


User Story: **Comprar entradas**





Test: Azul | Función a testear: Verde

PseudoCódigos para TDD:

fecha mayor o igual a la actual:

(consideramos que la fecha actual es "2025-06-17")

function test\_fecha\_futura\_exito() {

fecha\_visita = "2025-06-17";

assert.greaterEqual(obtenerFechaActual(), fecha\_visita) //compara que la fecha de visita sea mayor o igual a la actual.

}

function obtenerFechaActual() {

return getDate(); //retorna la fecha actual - “Formato: YYYY-MM-DD”

}

// -----------------------------------------------------------------------------------------------------------------------

// caso de prueba de falla (fecha de visita menor a la actual) - en este caso validamo que lo que reciba sea un mensaje de “Fecha pasada”

function test\_fecha\_pasada\_pasa() {

fechaVisita = "2025-06-16";

fechaActual = obtenerFechaActual();

try {

compararFechas(fechaVisita, fechaActual);

} catch (ex) {

assert.equals(ex.message, "Fecha pasada");

}

}

function compararFechas(fecha\_visita, fecha\_actual) {

if (assert.smallerThan(fecha\_visita, fecha\_actual)) {

throw new FechaPasadaException("Fecha pasada");

} else {

return "Fecha válida";

}

}

// Validación de día permitido (excepto martes o jueves)

function esDiaPermitido(fechaVisitaStr) {

fecha = new Date(fechaVisitaStr); // Convertimos los string a Date

diaSemana = fecha.getDay(); // 0 = domingo, 1 = lunes, ..., 6 = sábado

if (diaSemana === 2 || diaSemana === 4) {

throw new DiaNoPermitidoException("El día de visita no está permitido (martes o jueves)");

}

return true; // Día permitido, No jueves ni martes.

}

function test\_fecha\_martes\_falla() {

fechaVisita = "2025-06-17"; // Martes

try {

esDiaPermitido(fechaVisita);

} catch (ex) {

assert.equals(ex.message, "El día de visita no está permitido (martes o jueves)");

}

}

function test\_fecha\_miercoles\_exito() {

const fechaVisita = "2025-06-18"; // Miércoles

const permitido = esDiaPermitido(fechaVisita);

assert.equals(permitido, true);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

cantidad entradas

function test\_cantidadEntradas\_exito() {

a = 10

assert.lessEqual(cantidadEntradas(a), 10)

}

function cantidadEntradas(a) {

if (a>= 1 and a <=10)

return a

else

throw entradaException ("Debe ingresar al menos una entrada y hasta 10 entradas")

function test\_cantidadEntradas\_ex0 ()

b ‎ = 0 (acá es a pero no me deja)

try { cantidaEntradas(a)}

catch (ExceptionType ex)

finally {assert.equals(ex, ""Debe ingresar al menos una entrada y hasta 10 entradas")}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Cupos mayor a 0

// ---------- Casos de ÉXITO ----------

function test\_cupos\_success\_multiples() {

entradasDisponibles = 5

assert.equals(validarCupos(entradasDisponibles), true)

}

// ---------- Casos de FALLO ----------

function test\_cupos\_failure\_cero() {

entradasDisponibles = 0

try { validarCupos(entradasDisponibles) }

catch (invalidCuposException ex)

finally {

assert.equals(ex, "No hay entradas suficientes (debe ser > 0)")

}

}

function test\_cupos\_failure\_negativo() {

entradasDisponibles = -3

try { validarCupos(entradasDisponibles) }

catch (invalidCuposException ex)

finally {

assert.equals(ex, "No hay entradas suficientes (debe ser > 0)")

}

}

function validarCupos(entradasDisponibles) {

if (entradasDisponibles > 0) {

return true;

} else {

throw new invalidCuposException("No hay entradas suficientes (debe ser > 0)");

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

metodo de pago

function formaPago(metodo) {

if (metodo == "efectivo" || metodo == "tarjeta") {

return metodo

} else {

throw formaPagoException("Debe seleccionar 'efectivo' o 'tarjeta'")

}

}

function test\_formaPago\_efectivo\_success() {

metodo = "efectivo"

assert.equal(formaPago(metodo), "efectivo")

}

function test\_formaPago\_tarjeta\_success() {

metodo = "tarjeta"

assert.equal(formaPago(metodo), "tarjeta")

}

function test\_formaPago\_invalido\_error() {

metodo = "transferencia"

try {

formaPago(metodo)

} catch (ExceptionType ex) {

assert.equal(ex, "Debe seleccionar 'efectivo' o 'tarjeta'")

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Edad:

function validarEdad(edad) {

if (edad > 0) {

return true;

} else {

throw new EdadInvalidaException("La edad debe ser mayor a 0");

}

}

function test\_validarEdad\_pasa() {

edad = 25;

assert.equal(validarEdad(edad), true);

}

function test\_validarEdad\_falla() {

edad = 0;

try {

validarEdad(edad);

} catch (ex) {

assert.equal(ex.message, "La edad debe ser mayor a 0");

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tipo de pase:   
  
function validarTipoPase(pase) {

if (pase === "VIP" || pase === "regular") {

return true;

} else {

throw new TipoPaseInvalidoException("El tipo de pase debe ser 'VIP' o 'regular'");

}

}

function test\_validarTipoPase\_pasa() {

pase = "VIP";

assert.equal(validarTipoPase(pase), true);

}

function test\_validarTipoPase\_falla() {

const pase = "gold";

try {

validarTipoPase(pase);

} catch (ex) {

assert.equal(ex.message, "El tipo de pase debe ser 'VIP' o 'regular'");

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

usuario registrado

function verificarUsuarioLogueado(usuarioLogueado) {

if (!usuarioLogueado) {

throw new UsuarioNoLogueadoException("Debe estar logueado para comprar entradas");

}

return true;

}

function test\_verificarUsuarioLogueado\_pasa() {

logueado = true;

assert.equal(verificarUsuarioLogueado(logueado), true);

}

function test\_verificarUsuarioLogueado\_falla() {

logueado = false;

try {

verificarUsuarioLogueado(logueado);

} catch (ex) {

assert.equal(ex.message, "Debe estar logueado para comprar entradas");

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Envio email

function enviarConfirmacionMail(email) {

if (!email || !email.includes("@")) {

return false; // Email inválido, simulamos que no se pudo enviar

}

console.log(` Enviando mail de confirmación a ${email}...`);

return true;

}

function test\_enviarConfirmacionMail\_pasa() {

resultado = enviarConfirmacionMail("lucas@correo.com");

assert.equal(resultado, true);

}

function test\_enviarConfirmacionMail\_falla() {

resultado = enviarConfirmacionMail(""); // sin email válido

assert.equal(resultado, false);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Redirección mercadoPago

function redirigirMercadoPago(metodo\_pago) {

if (metodo\_pago == "tarjeta") {

return "Redirigiendo a MercadoPago..."

} else if (metodo\_pago == "efectivo") {

return "No se requiere redirección"

} else {

throw metodoPagoException("Método de pago inválido")

}

}

function test\_redirigirMercadoPago\_tarjeta\_pasa() {

metodo = "tarjeta"

resultado = redirigirMercadoPago(metodo)

assert.equals(resultado, "Redirigiendo a MercadoPago...")

}

function test\_redirigirMercadoPago\_efectivo\_pasa() {

metodo = "efectivo"

resultado = redirigirMercadoPago(metodo)

assert.equals(resultado, "No se requiere redirección")

}

function test\_redirigirMercadoPago\_invalido\_falla() {

metodo = "cheque"

try {

redirigirMercadoPago(metodo)

} catch (ex) {

assert.equals(ex, "Método de pago inválido")

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Comprar entradas

function comprarEntrada({ fecha, edad, cantidad, tipoPase,formaPago, usuarioLogueado,cuposDisponibles,parqueAbierto

}) {

verificarUsuarioLogueado(usuarioLogueado);

if (!parqueAbierto) throw new ParqueCerradoException("El parque no está disponible ese día");

validarEdad(edad);

validarTipoPase(tipoPase);

formaPagoValidado = formaPagoMetodo(formaPago);

cantidadEntradas(cantidad);

validarCupos(cuposDisponibles);

redirigirPago(formaPagoValidado);

enviarConfirmacionMail("lucas@gmail.com"); // simplificado

return { fecha, cantidad };

}

function test\_comprarEntrada\_pasa() {

mailEnviado = false;

redirigidoAMercadoPago = false;

data = {

fecha: "2025-06-18", // Miércoles válido

edad: 25,

cantidad: 3,

tipoPase: "VIP",

formaPago: "tarjeta",

usuarioLogueado: true,

cuposDisponibles: 5,

parqueAbierto: true

};

resultado = comprarEntrada(data);

assert.equals(resultado.fecha, "2025-06-18");

assert.equals(resultado.cantidad, 3);

assert.equals(mailEnviado, true);

assert.equals(redirigidoAMercadoPago, true);

}

function test\_comprarEntrada\_sinFormaPago\_falla() {

try {

comprarEntrada({

fecha: "2025-06-18",

edad: 30,

cantidad: 2,

tipoPase: "VIP",

formaPago: "", // inválido

usuarioLogueado: true,

cuposDisponibles: 5,

parqueAbierto: true

});

} catch (ex) {

assert.equal(ex.message, "Debe seleccionar 'efectivo' o 'tarjeta'");

}

}

function test\_comprarEntrada\_parqueCerrado\_falla() {

try {

comprarEntrada({

fecha: "2025-06-20",

edad: 30,

cantidad: 2,

tipoPase: "regular",

formaPago: "efectivo",

usuarioLogueado: true,

cuposDisponibles: 5,

parqueAbierto: false // parque cerrado ese día

});

} catch (ex) {

assert.equal(ex.message, "El parque no está disponible ese día");

}

}

function test\_comprarEntrada\_masDeDiez\_falla() {

try {

comprarEntrada({

fecha: "2025-06-18",

edad: 30,

cantidad: 12, // inválido

tipoPase: "regular",

formaPago: "efectivo",

usuarioLogueado: true,

cuposDisponibles: 12,

parqueAbierto: true

});

} catch (ex) {

assert.equal(ex.message, "Debe ingresar al menos una entrada y hasta 10 entradas");

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

function calcularNuevoNumeroCompra(ultimoNumeroCompra){

nuevoNumero = ultimoNumeroCompra + 1

return nuevoNumero

}

function test\_calcularNuevoNumeroCompra\_exito(){

ultimoNumeroCompra = 105

numeroEsperado = 106

assert.equals(calcularNuevoNumeroCompra(105), numeroEsperado)

}