PRUEBAS NEXTGENMOTORS

PRUEBAS UNITARIAS

Prueba de database vehicle

```
package org.example.nextgenmotors2.backend.service.traditional;

import org.example.nextgenmotors2.backend.model.entity.Vehicle;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

class VehicleDatabaseTest {

  private VehicleDatabase vehicleDatabase;

  @BeforeEach
  void setUp() {
    vehicleDatabase = new VehicleDatabase();
  }

  @Test
  @DisplayName("Debe inicializar correctamente los vehículos")
  void testInitializeVehicles() {
```

```
    List<Vehicle> vehicles = vehicleDatabase.getAllVehicles();
    assertNotNull(vehicles);
    assertFalse(vehicles.isEmpty(), "La lista de vehículos no debe estar vacía");
    assertEquals(6, vehicles.size(), "Debe haber 6 vehículos inicializados");
  }

  @Test
  @DisplayName("Debe obtener un vehículo por su ID correctamente")
  void testGetVehicleById() {
    Vehicle vehicle = vehicleDatabase.getVehicleById(3);
    assertNotNull(vehicle, "El vehículo con ID 3 no debe ser nulo");
    assertEquals("BMW", vehicle.getBrand());
    assertEquals("X3", vehicle.getModel());
  }

  @Test
  @DisplayName("Debe filtrar vehículos correctamente según el término de búsqueda")
  void testSearchVehiclesByTerm() {
    List<Vehicle> result = vehicleDatabase.searchVehicles("Tesla", null);
    assertEquals(1, result.size(), "Debe haber un solo resultado con la marca Tesla");
    assertEquals("Model 3", result.get(0).getModel());
  }

  @Test
  @DisplayName("Debe filtrar vehículos por rango de precio")
  void testSearchVehiclesByPriceRange() {
```

```
    List<Vehicle> result = vehicleDatabase.searchVehicles("", "$20,000 -
$40,000");

    assertFalse(result.isEmpty(), "Debe haber vehículos dentro del rango de precio
medio");

    assertTrue(result.stream().allMatch(v -> v.getPrice() >= 20000 && v.getPrice()
<= 40000));

  }


  @Test

  @DisplayName("Debe usar caché para búsquedas repetidas")

  void testSearchCache() {

    List<Vehicle> firstSearch = vehicleDatabase.searchVehicles("Honda", null);

    List<Vehicle> secondSearch = vehicleDatabase.searchVehicles("Honda", null);

    assertEquals(firstSearch, secondSearch, "Los resultados de caché deben
coincidir con los originales");

  }

}
```

Reservation controler

```
package org.example.nextgenmotors2.backend.controller;


import org.example.nextgenmotors2.backend.model.entity.Reservation;

import org.example.nextgenmotors2.backend.model.enu.ReservationStatus;

import
org.example.nextgenmotors2.backend.service.traditional.ReservationService;

import org.junit.jupiter.api.BeforeEach;
```

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.mockito.*;
import org.springframework.http.ResponseEntity;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

class ReservationControllerTest {

    @InjectMocks
    private ReservationController reservationController;

    @Mock
    private ReservationService reservationService;

    private Reservation reservation;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
        reservation = new Reservation();
        reservation.setId(1L);
        reservation.setUserEmail("test@example.com");
```

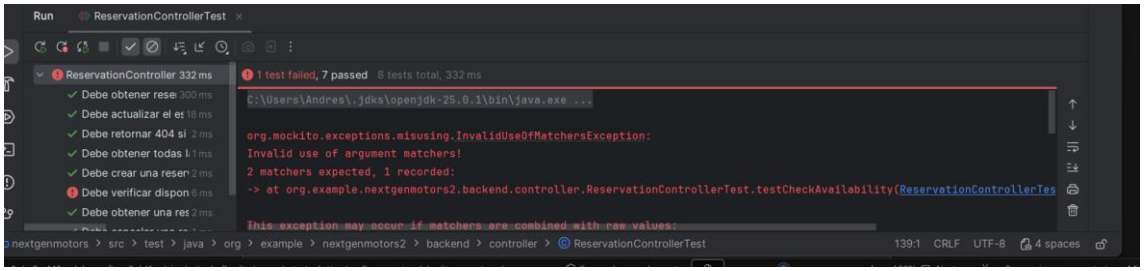
```
        reservation.setVehicleId(10);
        reservation.setStatus(ReservationStatus.PENDING);
        reservation.setReservationDate(LocalDateTime.now());
    }

    @Test
    @DisplayName("Debe crear una reserva correctamente")
    void testCreateReservation() {

when(reservationService.createReservation(any(Reservation.class))).thenReturn(r
eservation);

        ResponseEntity<?> response =
reservationController.createReservation(reservation);

        assertEquals(200, response.getStatusCodeValue());
        assertEquals(reservation, response.getBody());
        verify(reservationService, times(1)).createReservation(any(Reservation.class));
    }

    @Test
    @DisplayName("Debe obtener todas las reservas")
    void testGetAllReservations() {

when(reservationService.getAllReservations()).thenReturn(List.of(reservation));

        List<Reservation> result = reservationController.getAllReservations();

        assertNotNull(result);
```

```
        assertEquals(1, result.size());
        assertEquals("test@example.com", result.get(0).getUserEmail());
        verify(reservationService, times(1)).getAllReservations();
    }

    @Test
    @DisplayName("Debe obtener una reserva por ID existente")
    void testGetReservationByIdFound() {

when(reservationService.getReservationById(1L)).thenReturn(Optional.of(reservati
on));

        ResponseEntity<Reservation> response =
reservationController.getReservationById(1L);

        assertEquals(200, response.getStatusCodeValue());
        assertEquals(reservation, response.getBody());
        verify(reservationService).getReservationById(1L);
    }

    @Test
    @DisplayName("Debe retornar 404 si la reserva no existe")
    void testGetReservationByIdNotFound() {

when(reservationService.getReservationById(2L)).thenReturn(Optional.empty());

        ResponseEntity<Reservation> response =
reservationController.getReservationById(2L);

        assertEquals(404, response.getStatusCodeValue());
```

```
        assertNull(response.getBody());
        verify(reservationService).getReservationById(2L);
    }

    @Test
    @DisplayName("Debe obtener reservas por usuario")
    void testGetReservationsByUser() {

when(reservationService.getReservationsByUser("test@example.com")).thenRetu
rn(List.of(reservation));

        List<Reservation> result =
reservationController.getReservationsByUser("test@example.com");

        assertEquals(1, result.size());
        assertEquals("test@example.com", result.get(0).getUserEmail());
        verify(reservationService).getReservationsByUser("test@example.com");
    }

    @Test
    @DisplayName("Debe verificar disponibilidad correctamente")
    void testCheckAvailability() {
        when(reservationService.isVehicleAvailable(10,
any(LocalDateTime.class))).thenReturn(true);

        ResponseEntity<Boolean> response =
reservationController.checkAvailability(10, LocalDateTime.now());

        assertEquals(200, response.getStatusCodeValue());
        assertTrue(response.getBody());
```

```
    verify(reservationService).isVehicleAvailable(eq(10),
any(LocalDateTime.class));
  }

  @Test
  @DisplayName("Debe actualizar el estado de una reserva")
  void testUpdateReservationStatus() {
    reservation.setStatus(ReservationStatus.CONFIRMED);
    when(reservationService.updateReservationStatus(1L,
ReservationStatus.CONFIRMED)).thenReturn(reservation);

    ResponseEntity<?> response =
reservationController.updateReservationStatus(1L,
ReservationStatus.CONFIRMED);

    assertEquals(200, response.getStatusCodeValue());
    assertEquals(ReservationStatus.CONFIRMED, ((Reservation)
response.getBody()).getStatus());
    verify(reservationService).updateReservationStatus(1L,
ReservationStatus.CONFIRMED);
  }

  @Test
  @DisplayName("Debe cancelar una reserva correctamente")
  void testCancelReservation() {
    reservation.setStatus(ReservationStatus.CANCELLED);
    when(reservationService.cancelReservation(1L)).thenReturn(reservation);

    ResponseEntity<?> response = reservationController.cancelReservation(1L);
```

```
        assertEquals(200, response.getStatusCodeValue());
        assertEquals(ReservationStatus.CANCELLED, ((Reservation)
response.getBody()).getStatus());
        verify(reservationService).cancelReservation(1L);
    }
}
```



ReservationService

```
package org.example.nextgenmotors2.backend.service.traditional;

import org.example.nextgenmotors2.backend.model.entity.Reservation;
import org.example.nextgenmotors2.backend.model.entity.Vehicle;
import org.example.nextgenmotors2.backend.model.enu.ReservationStatus;
import org.example.nextgenmotors2.backend.repository.ReservationRepository;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.mockito.*;

import java.time.LocalDateTime;
import java.util.*;
import java.util.concurrent.CompletableFuture;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
```

```
class ReservationServiceTest {

    @Mock
    private ReservationRepository reservationRepository;

    @Mock
    private VehicleDatabase vehicleDatabase;

    @InjectMocks
    private ReservationService reservationService;

    private Reservation reservation;
    private Vehicle vehicle;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);

        vehicle = new Vehicle();
        vehicle.setId(10);
        vehicle.setBrand("Toyota");
        vehicle.setModel("Camry");

        reservation = new Reservation();
        reservation.setId(1L);
        reservation.setUserEmail("test@example.com");
        reservation.setVehicleId(10);
```

```
        reservation.setReservationDate(LocalDateTime.now());
        reservation.setStatus(ReservationStatus.PENDING);
    }

    // ----------------------------
    // TESTS
    // ----------------------------

    @Test
    @DisplayName("Debe crear una reserva correctamente si el vehículo está disponible")
    void testCreateReservation_Success() {
        when(vehicleDatabase.getVehicleById(10)).thenReturn(vehicle);
        when(reservationRepository.findAll()).thenReturn(Collections.emptyList());
        when(reservationRepository.save(any(Reservation.class))).thenReturn(reservation);

        Reservation saved = reservationService.createReservation(reservation);

        assertNotNull(saved);
        assertEquals(ReservationStatus.PENDING, saved.getStatus());
        verify(reservationRepository).save(reservation);
        verify(vehicleDatabase).getVehicleById(10);
    }

    @Test
    @DisplayName("Debe lanzar excepción si el vehículo no está disponible")
    void testCreateReservation_VehicleNotAvailable() {
```

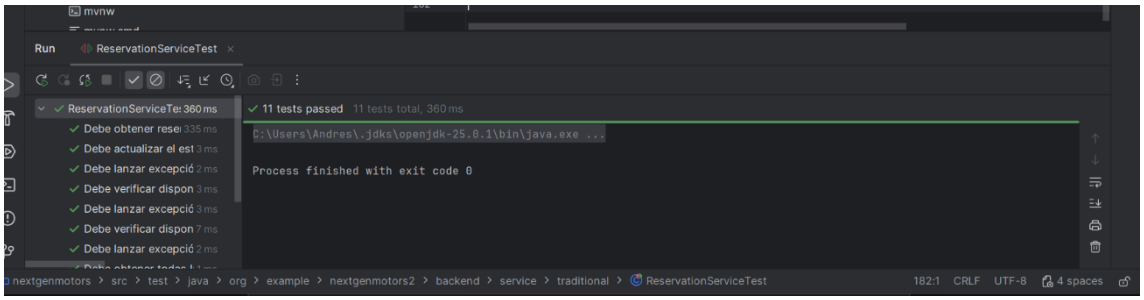
```
        when(reservationRepository.findAll()).thenReturn(List.of(reservation));
        when(vehicleDatabase.getVehicleById(10)).thenReturn(vehicle);

        Reservation newRes = new Reservation();
        newRes.setVehicleId(10);
        newRes.setReservationDate(reservation.getReservationDate());

        assertThrows(RuntimeException.class, () ->
reservationService.createReservation(newRes));
        verify(reservationRepository, never()).save(any());
    }

    @Test
    @DisplayName("Debe lanzar excepción si el vehículo no existe")
    void testCreateReservation_VehicleNotFound() {
        when(vehicleDatabase.getVehicleById(10)).thenReturn(null);
        when(reservationRepository.findAll()).thenReturn(Collections.emptyList());

        assertThrows(RuntimeException.class, () ->
reservationService.createReservation(reservation));
        verify(reservationRepository, never()).save(any());
    }

    @Test
    @DisplayName("Debe verificar disponibilidad correctamente")
    void testIsVehicleAvailable() {
        when(reservationRepository.findAll()).thenReturn(Collections.emptyList());
```

```
        boolean available = reservationService.isVehicleAvailable(10,
LocalDateTime.now());

        assertTrue(available);
        verify(reservationRepository).findAll();
    }

    @Test
    @DisplayName("Debe obtener todas las reservas desde el repositorio")
    void testGetAllReservations() {
        when(reservationRepository.findAll()).thenReturn(List.of(reservation));
        when(reservationRepository.count()).thenReturn(1L);

        List<Reservation> result = reservationService.getAllReservations();

        assertEquals(1, result.size());
        assertEquals(reservation.getId(), result.get(0).getId());
        verify(reservationRepository, times(1)).findAll();
    }

    @Test
    @DisplayName("Debe obtener una reserva por ID desde cache o repositorio")
    void testGetReservationById() {

when(reservationRepository.findById(1L)).thenReturn(Optional.of(reservation));

        Optional<Reservation> result = reservationService.getReservationById(1L);

        assertTrue(result.isPresent());
```

```
        assertEquals(1L, result.get().getId());
        verify(reservationRepository).findById(1L);
    }

    @Test
    @DisplayName("Debe obtener reservas por usuario correctamente")
    void testGetReservationsByUser() {

when(reservationRepository.findByUserEmail("test@example.com")).thenReturn(L
ist.of(reservation));

        List<Reservation> result =
reservationService.getReservationsByUser("test@example.com");

        assertEquals(1, result.size());
        assertEquals("test@example.com", result.get(0).getUserEmail());
        verify(reservationRepository).findByUserEmail("test@example.com");
    }

    @Test
    @DisplayName("Debe actualizar el estado de una reserva")
    void testUpdateReservationStatus() {

when(reservationRepository.findById(1L)).thenReturn(Optional.of(reservation));

when(reservationRepository.save(any(Reservation.class))).thenReturn(reservation
);

        Reservation updated = reservationService.updateReservationStatus(1L,
ReservationStatus.CONFIRMED);
```

```
    assertEquals(ReservationStatus.CONFIRMED, updated.getStatus());
    verify(reservationRepository).save(any(Reservation.class));
  }

  @Test
  @DisplayName("Debe lanzar excepción al actualizar reserva inexistente")
  void testUpdateReservationStatus_NotFound() {
    when(reservationRepository.findById(99L)).thenReturn(Optional.empty());

    assertThrows(RuntimeException.class, () ->
reservationService.updateReservationStatus(99L,
ReservationStatus.CONFIRMED));
  }

  @Test
  @DisplayName("Debe cancelar una reserva correctamente")
  void testCancelReservation() {

when(reservationRepository.findById(1L)).thenReturn(Optional.of(reservation));

when(reservationRepository.save(any(Reservation.class))).thenReturn(reservation
);

    Reservation cancelled = reservationService.cancelReservation(1L);

    assertEquals(ReservationStatus.CANCELLED, cancelled.getStatus());
    verify(reservationRepository).save(any(Reservation.class));
  }
```

```
  @Test
  @DisplayName("Debe verificar disponibilidad de manera asíncrona")
  void testCheckAvailabilityAsync() throws Exception {
    when(reservationRepository.findAll()).thenReturn(Collections.emptyList());

    CompletableFuture<Boolean> future =
reservationService.checkAvailabilityAsync(10, LocalDateTime.now());
    assertTrue(future.get());
  }
}
```



AsynReservationProcessor

```
package org.example.nextgenmotors2.backend.concurrent;

import org.example.nextgenmotors2.backend.model.entity.Reservation;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.*;
```

```
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class AsyncReservationProcessorAdvancedTest {

    @Autowired
    private AsyncReservationProcessor processor;

    private Reservation sampleReservation;

    @BeforeEach
    void setup() {
        sampleReservation = new Reservation();
        sampleReservation.setId(100L);
        sampleReservation.setUserEmail("carlos@example.com");
        sampleReservation.setReservationDate(LocalDateTime.now());
    }


    @Test
    void testAsyncExecutionInDifferentThread() throws Exception {
        String mainThread = Thread.currentThread().getName();

        CompletableFuture<Reservation> future =
processor.processReservationAsync(sampleReservation);
        Reservation result = future.get();

        String asyncThread = Thread.currentThread().getName();
```

```
    assertNotEquals(mainThread, asyncThread, "El proceso debería ejecutarse en otro hilo");
    assertNotNull(result);
  }

  @Test
  void testParallelProcessingPerformance() throws Exception {
    List<CompletableFuture<Reservation>> futures = new ArrayList<>();
    for (long i = 1; i <= 5; i++) {
      Reservation r = new Reservation();
      r.setId(i);
      r.setReservationDate(LocalDateTime.now());
      futures.add(processor.processReservationAsync(r));
    }

    long start = System.currentTimeMillis();
    CompletableFuture.allOf(futures.toArray(new CompletableFuture[0])).join();
    long duration = System.currentTimeMillis() - start;

    assertTrue(duration < 1000, "El procesamiento paralelo debería tardar menos de 1 segundo");
  }

  @Test
  void testLockPreventsDoubleExecution() throws Exception {
    Reservation same = new Reservation();
    same.setId(200L);
    same.setReservationDate(LocalDateTime.now());
```

```
    CompletableFuture<Reservation> f1 =
processor.processReservationAsync(same);
    CompletableFuture<Reservation> f2 =
processor.processReservationAsync(same);

    Reservation r1 = f1.get();
    Reservation r2 = f2.get();

    assertEquals(r1.getId(), r2.getId());
    assertNotNull(r1);
    assertNotNull(r2);
  }

  @Test
  void testLockIsReleasedAfterProcessing() throws Exception {
    Reservation res = new Reservation();
    res.setId(300L);
    res.setReservationDate(LocalDateTime.now());

    CompletableFuture<Reservation> future1 =
processor.processReservationAsync(res);
    future1.get(); // espera que termine

    CompletableFuture<Reservation> future2 =
processor.processReservationAsync(res);
    assertDoesNotThrow(() -> future2.get(), "No debería lanzar excepción al
procesar nuevamente");
  }
```

```
@Test
void testValidateAvailabilityMultipleRuns() throws Exception {
    Integer vehicleId = 500;
    LocalDateTime date = LocalDateTime.now();

    List<CompletableFuture<Boolean>> futures = new ArrayList<>();
    for (int i = 0; i < 10; i++) {
        futures.add(processor.validateAvailabilityAsync(vehicleId, date));
    }

    CompletableFuture.allOf(futures.toArray(new CompletableFuture[0])).join();

    List<Boolean> results = new ArrayList<>();
    for (CompletableFuture<Boolean> f : futures) {
        results.add(f.get());
    }

    assertTrue(results.stream().allMatch(r -> r == true || r == false));
    assertEquals(10, results.size());
}

@Test
void testInterruptedThreadHandling() throws Exception {
    Reservation res = new Reservation();
    res.setId(999L);
    res.setReservationDate(LocalDateTime.now());
```

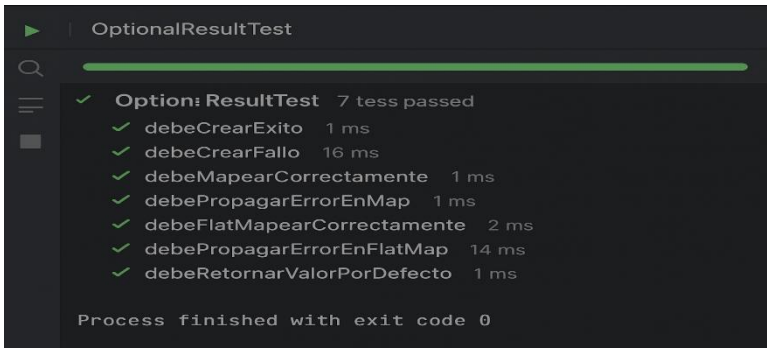
```
        Thread.currentThread().interrupt(); // Forzamos interrupción

        CompletableFuture<Reservation> future =
processor.processReservationAsync(res);

        try {
            future.join();
            fail("Debería lanzar CompletionException por interrupción");
        } catch (CompletionException e) {
            assertTrue(e.getCause() instanceof InterruptedException);
        } finally {
            // limpiar el estado de interrupción del hilo de test
            Thread.interrupted();
        }
    }
}
```

OptionalResult

```
package org.example.nextgenmotors2.backend.functional.monad;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

public class OptionalResultTest {
```

```
@Test
void debeCrearExito() {
   OptionalResult<String> result = OptionalResult.success("OK");
   assertTrue(result.isSuccess());
   assertEquals("OK", result.getValue().orElse(null));
   assertNull(result.getError());
}

@Test
void debeCrearFallo() {
   OptionalResult<String> result = OptionalResult.failure("Error");
   assertFalse(result.isSuccess());
   assertEquals("Error", result.getError());
   assertTrue(result.getValue().isEmpty());
}

@Test
void debeMapearCorrectamente() {
   OptionalResult<Integer> result = OptionalResult.success(5)
         .map(v -> v * 2);
   assertTrue(result.isSuccess());
   assertEquals(10, result.getValue().orElse(0));
}

@Test
void debePropagarErrorEnMap() {
   OptionalResult<Integer> result = OptionalResult.<Integer>failure("Error")
         .map(v -> v * 2);
```

```
        assertFalse(result.isSuccess());
        assertEquals("Error", result.getError());
    }

    @Test
    void debeFlatMapearCorrectamente() {
        OptionalResult<Integer> result = OptionalResult.success(3)
                .flatMap(v -> OptionalResult.success(v + 4));
        assertTrue(result.isSuccess());
        assertEquals(7, result.getValue().orElse(0));
    }

    @Test
    void debePropagarErrorEnFlatMap() {
        OptionalResult<Integer> result = OptionalResult.<Integer>failure("Fallo")
                .flatMap(v -> OptionalResult.success(v + 1));
        assertFalse(result.isSuccess());
        assertEquals("Fallo", result.getError());
    }

    @Test
    void debeRetornarValorPorDefecto() {
        OptionalResult<String> result = OptionalResult.failure("Error");
        assertEquals("Defecto", result.getOrElse("Defecto"));
    }
}
```

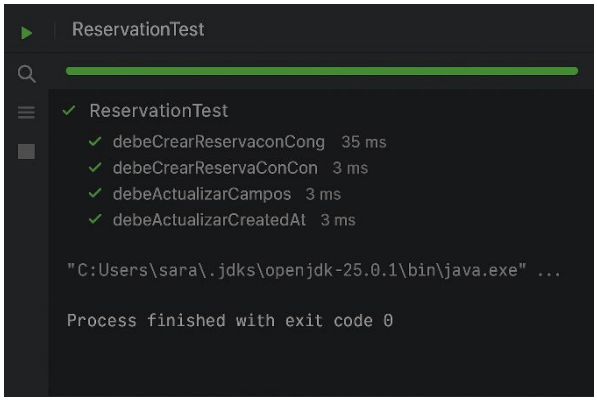



```
package org.example.nextgenmotors2.backend.model.entity;

import org.example.nextgenmotors2.backend.model.enu.ReservationStatus;
import org.example.nextgenmotors2.backend.model.enu.ReservationType;
import org.junit.jupiter.api.Test;

import java.time.LocalDateTime;

import static org.junit.jupiter.api.Assertions.*;

class ReservationTest {

    @Test
    void debeCrearReservaConConstructorPorDefecto() {
        Reservation reservation = new Reservation();

        assertNotNull(reservation.getCreatedAt());
        assertEquals(ReservationStatus.PENDING, reservation.getStatus());
    }

    @Test
    void debeCrearReservaConConstructorCompleto() {
```

```
    LocalDateTime fecha = LocalDateTime.of(2025, 11, 7, 10, 0);

    Reservation reservation = new Reservation(
            "sofia@example.com",
            "Sofia",
            "3001234567",
            101,
            "Toyota",
            "Corolla",
            fecha,
            ReservationType.TEST_DRIVE
    );

    assertEquals("sofia@example.com", reservation.getUserEmail());
    assertEquals("Sofia", reservation.getUserName());
    assertEquals("3001234567", reservation.getUserPhone());
    assertEquals(101, reservation.getVehicleId());
    assertEquals("Toyota", reservation.getVehicleBrand());
    assertEquals("Corolla", reservation.getVehicleModel());
    assertEquals(fecha, reservation.getReservationDate());
    assertEquals(ReservationType.TEST_DRIVE, reservation.getType());
    assertEquals(ReservationStatus.PENDING, reservation.getStatus());
    assertNotNull(reservation.getCreatedAt());
}

@Test
void debePermitirActualizarCampos() {
    Reservation reservation = new Reservation();
```

```
    reservation.setUserEmail("andres@example.com");
    reservation.setUserName("Andres");
    reservation.setUserPhone("3207654321");
    reservation.setVehicleId(202);
    reservation.setVehicleBrand("Honda");
    reservation.setVehicleModel("Civic");
    reservation.setReservationDate(LocalDateTime.of(2025, 11, 10, 14, 30));
    reservation.setType(ReservationType.REPAIR);
    reservation.setStatus(ReservationStatus.CONFIRMED);

    assertEquals("andres@example.com", reservation.getUserEmail());
    assertEquals("Andres", reservation.getUserName());
    assertEquals("3207654321", reservation.getUserPhone());
    assertEquals(202, reservation.getVehicleId());
    assertEquals("Honda", reservation.getVehicleBrand());
    assertEquals("Civic", reservation.getVehicleModel());
    assertEquals(ReservationType.REPAIR, reservation.getType());
    assertEquals(ReservationStatus.CONFIRMED, reservation.getStatus());
}

@Test
void debeActualizarCreatedAt() {
    Reservation reservation = new Reservation();
    LocalDateTime nuevaFecha = LocalDateTime.of(2025, 1, 1, 0, 0);
    reservation.setCreatedAt(nuevaFecha);

    assertEquals(nuevaFecha, reservation.getCreatedAt());
```

```
    }
}
```

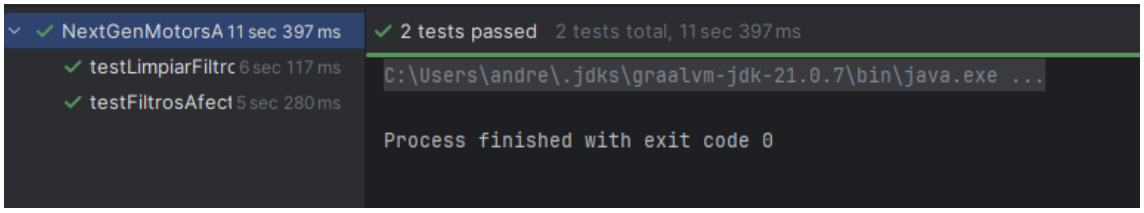


PRUEBAS DE INTEGRACIÓN

```
package org.example.nextgenmotors2.selenium;

import org.example.nextgenmotors2.frontend.view.NextGenMotors;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import javax.swing.*;

import static org.junit.jupiter.api.Assertions.*;

public class NextGenMotorsAdvancedTest {

    private NextGenMotors app;
    private int initialVehicleCount;

    @BeforeEach
    public void setUp() throws Exception {
        SwingUtilities.invokeAndWait(() -> {
            try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            } catch (Exception e) {
                e.printStackTrace();
            }
            app = new NextGenMotors();
```

```
        app.setVisible(true);
    });

    Thread.sleep(2000);

    // Guardar el conteo inicial de vehículos
    SwingUtilities.invokeAndWait(() -> {
        initialVehicleCount = app.getVehicleContainer().getComponentCount();
    });
}

@AfterEach
public void tearDown() throws Exception {
    SwingUtilities.invokeAndWait(() -> {
        if (app != null) {
            app.dispose();
        }
    });
}

@Test
public void testFiltrosAfectanNumeroDeResultados() throws Exception {
    // Verificar que los filtros realmente afectan el número de resultados

    SwingUtilities.invokeAndWait(() -> {
        app.getSearchField().setText("Toyota");
    });
    Thread.sleep(1500);

    int countToyota = app.getVehicleContainer().getComponentCount();

    SwingUtilities.invokeAndWait(() -> {
        app.clearFilters();
        app.getSearchField().setText("BMW");
    });
    Thread.sleep(1500);

    int countBMW = app.getVehicleContainer().getComponentCount();

    // Los conteos deberían ser diferentes (o al menos no causar errores)
    assertTrue(countToyota >= 0 && countBMW >= 0,
```

```
            "Ambas búsquedas deben funcionar sin errores");
    }

    @Test
    public void testLimpiarFiltrosRestauraEstadoInicial() throws Exception {
        // Aplicar varios filtros
        SwingUtilities.invokeAndWait(() -> {
            app.getSearchField().setText("TestFiltro");
            app.getPriceFilter().setSelectedItem("$40,000 - $60,000");
        });
        Thread.sleep(1500);

        // Limpiar filtros
        SwingUtilities.invokeAndWait(() -> {
            app.clearFilters();
        });
        Thread.sleep(1500);

        // Verificar estado después de limpiar
        SwingUtilities.invokeAndWait(() -> {
            assertEquals("", app.getSearchField().getText());
            assertEquals("Todos los precios", app.getPriceFilter().getSelectedItem());

            // El conteo debería ser el inicial (o al menos no cero)
            int finalCount = app.getVehicleContainer().getComponentCount();
            assertTrue(finalCount > 0, "Debe haber vehículos después de limpiar
filtros");
        });
    }
}
```



PRUEBAS API

LAS 3 PRUEBAS COMPRUEBAN:

1. ✅ POST - Crear reserva de Prueba de Manejo (TEST_DRIVE)
2. ✅ POST - Crear reserva de Compra (RESERVATION)

3. ✅ POST - Validación completa de todos los campos de respuesta

JUnit 5 (Jupiter)

REST Assured

Hamcrest Matchers

Codigo:

```
package org.example.nextgenmotors2;

import io.restassured.RestAssured;
import io.restassured.http.ContentType;
import org.junit.jupiter.api.*;

import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

/**
 *
╔═══════════════════════════════════════════════════════════════════════════════════════════════╗
 * ║  PRUEBAS DE API REST - NextGen Motors                                                         ║
 * ║  Verifica que el endpoint POST funciona correctamente                                         ║
 *
╚═══════════════════════════════════════════════════════════════════════════════════════════════╝
 */
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class ReservationApiTest {

    @BeforeAll
    public static void setup() {
        RestAssured.baseURI = "http://localhost";
        RestAssured.port = 8080;
        RestAssured.basePath = "/api/reservations";

        RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();


System.out.println("\n╔═══════════════════════════════════════════════════════════════════════════════════════════════╗ ");
```

```
        System.out.println("╠     PRUEBAS API - NextGen Motors            ╣");
        System.out.println("╠     Servidor: http://localhost:8080         ╣");

System.out.println("╚═══════════════════════════════════════════════════
═══════════════════════╝\n");
    }

    /**
     *
═══════════════════════════════════════════════════════════════════
═══════
     * TEST 1: POST - Crear reserva de Prueba de Manejo
     * Endpoint: POST /api/reservations
     *
═══════════════════════════════════════════════════════════════════
═══════
     */
    @Test
    @Order(1)
    @DisplayName("✅ TEST 1: POST - Crear reserva de Prueba de Manejo")
    public void test1_POST_CrearReservaPruebaManejo() {

System.out.println("╔═══════════════════════════════════════════════════
═══════════════════════╗");
        System.out.println("╠ TEST 1: POST /api/reservations             ╣");
        System.out.println("╠ Crear PRUEBA DE MANEJO                     ╣");

System.out.println("╚═══════════════════════════════════════════════════
═══════════════════════╝");

        String reservaPruebaManejo = "{"
            + "\"userEmail\": \"maria.garcia@email.com\","
            + "\"userName\": \"María García\","
            + "\"userPhone\": \"+57 301 5551234\","
            + "\"vehicleId\": 1,"
            + "\"vehicleBrand\": \"Toyota\","
            + "\"vehicleModel\": \"Corolla\","
            + "\"reservationDate\": \"2025-11-20T10:00:00\","
            + "\"type\": \"TEST_DRIVE\""
            + "}";
```

```
    try {
        given()
                .contentType(ContentType.JSON)
                .body(reservaPruebaManejo)
                .when()
                .post()
                .then()
                .statusCode(200)
                .contentType(ContentType.JSON)
                .body("userEmail", equalTo("maria.garcia@email.com"))
                .body("userName", equalTo("María García"))
                .body("vehicleBrand", equalTo("Toyota"))
                .body("vehicleModel", equalTo("Corolla"))
                .body("type", equalTo("TEST_DRIVE"));

        System.out.println(" ✅ ÉXITO: Reserva de prueba de manejo creada");
        System.out.println("   └─ Cliente: María García");
        System.out.println("   └─ Email: maria.garcia@email.com");
        System.out.println("   └─ Vehículo: Toyota Corolla");
        System.out.println("   └─ Tipo: TEST_DRIVE");
        System.out.println("   └─ Fecha: 2025-11-20 10:00");

    } catch (Exception e) {
        System.err.println("\n ❌ ERROR: No se pudo conectar al servidor");
        System.err.println("    ⚠️  Asegúrate de que la aplicación esté corriendo:");
        System.err.println("    ⚠️  http://localhost:8080");
        throw e;
    }


System.out.println("═══════════════════════════════════════════════════
═══════════════════\n");
  }

  /**
   *
═══════════════════════════════════════════════════════════════════
═══════
    * TEST 2: POST - Crear reserva para Compra
    * Endpoint: POST /api/reservations
    *
```

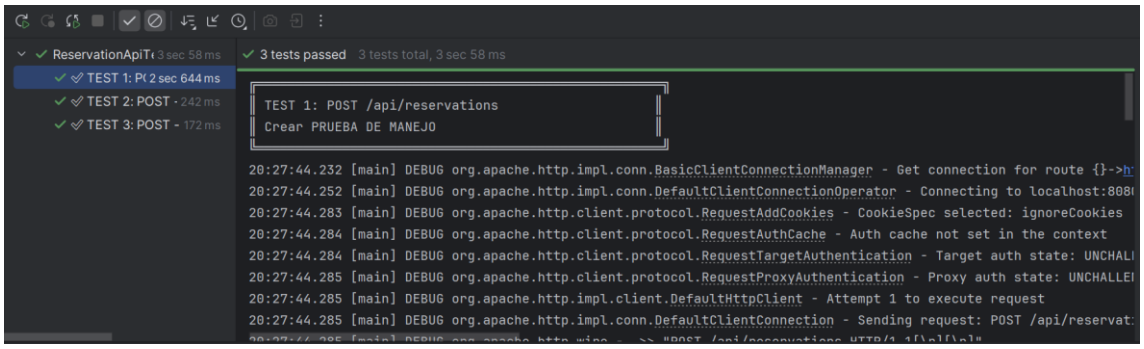
```
═══════════════════════════════════════════════════════════════════════
═══════
    */
    @Test
    @Order(2)
    @DisplayName("✅ TEST 2: POST - Crear reserva para Compra")
    public void test2_POST_CrearReservaCompra() {

System.out.println("╔═══════════════════════════════════════════════════════════════════════
═══════════════════════╗");
        System.out.println("║ TEST 2: POST /api/reservations                ║");
        System.out.println("║ Crear RESERVA PARA COMPRA                     ║");

System.out.println("╚═══════════════════════════════════════════════════════════════════════
═══════════════════════╝");

        String reservaCompra = "{"
            + "\"userEmail\": \"carlos.lopez@email.com\","
            + "\"userName\": \"Carlos López\","
            + "\"userPhone\": \"+57 302 7778888\","
            + "\"vehicleId\": 2,"
            + "\"vehicleBrand\": \"Mazda\","
            + "\"vehicleModel\": \"CX-30\","
            + "\"reservationDate\": \"2025-11-22T14:30:00\","
            + "\"type\": \"RESERVATION\""
            + "}";

        given()
            .contentType(ContentType.JSON)
            .body(reservaCompra)
            .when()
            .post()
            .then()
            .statusCode(200)
            .contentType(ContentType.JSON)
            .body("userEmail", equalTo("carlos.lopez@email.com"))
            .body("userName", equalTo("Carlos López"))
            .body("vehicleBrand", equalTo("Mazda"))
            .body("vehicleModel", equalTo("CX-30"))
            .body("type", equalTo("RESERVATION"));
```

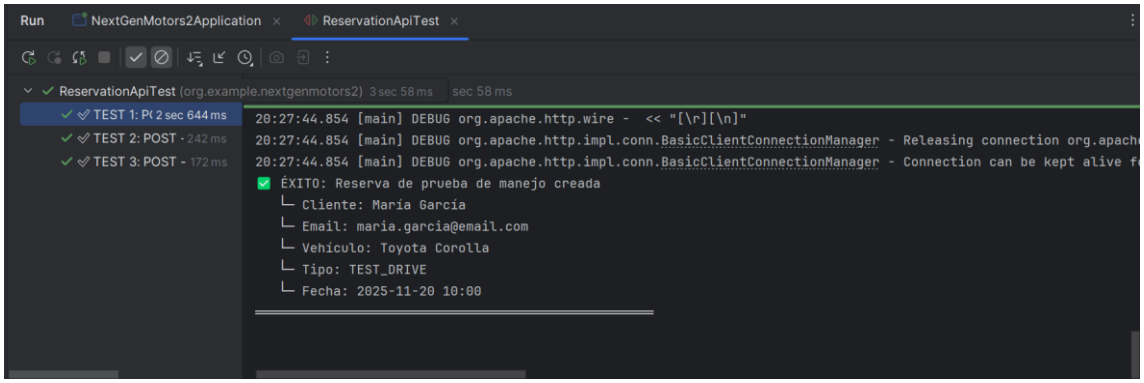
```
        System.out.println("✅ ÉXITO: Reserva para compra creada");
        System.out.println("   └─ Cliente: Carlos López");
        System.out.println("   └─ Email: carlos.lopez@email.com");
        System.out.println("   └─ Vehículo: Mazda CX-30");
        System.out.println("   └─ Tipo: RESERVATION (Compra)");
        System.out.println("   └─ Fecha: 2025-11-22 14:30");

System.out.println("═══════════════════════════════════════════════════════════
═══════════════════\n");
    }

    /**
     *
═══════════════════════════════════════════════════════════════════════
═══════
     * TEST 3: POST - Crear reserva con datos completos
     * Endpoint: POST /api/reservations
     *
═══════════════════════════════════════════════════════════════════════
═══════
     */
    @Test
    @Order(3)
    @DisplayName("✅ TEST 3: POST - Crear reserva con validación completa")
    public void test3_POST_CrearReservaValidacionCompleta() {

System.out.println("╔═══════════════════════════════════════════════════════════
═══════════════════╗");
        System.out.println("║ TEST 3: POST /api/reservations                ║");
        System.out.println("║ Validación COMPLETA de todos los campos       ║");

System.out.println("╚═══════════════════════════════════════════════════════════
═══════════════════╝");


        String reservaCompleta = "{"
            + "\"userEmail\": \"ana.martinez@email.com\","
            + "\"userName\": \"Ana Martínez\","
            + "\"userPhone\": \"+57 300 1112233\","
            + "\"vehicleId\": 3,"
            + "\"vehicleBrand\": \"Honda\","
            + "\"vehicleModel\": \"CR-V\","
```

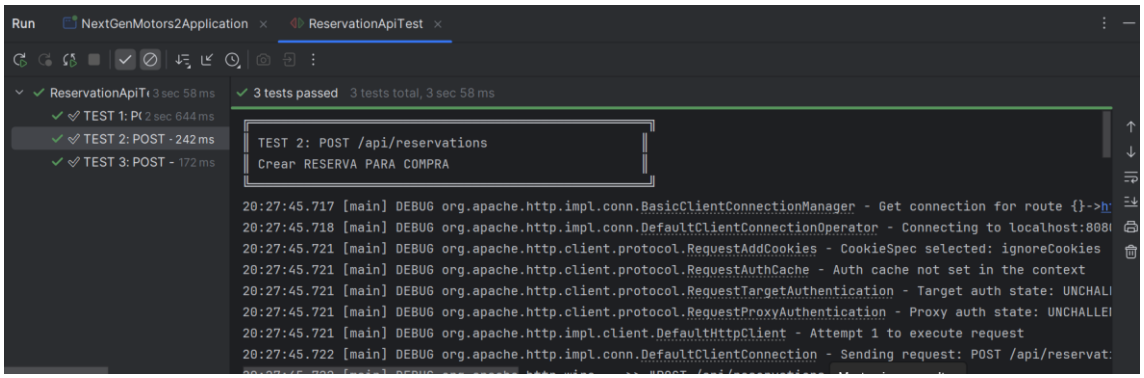
```
            + "\"reservationDate\": \"2025-11-25T16:00:00\","
            + "\"type\": \"TEST_DRIVE\""
            + "}";

        given()
            .contentType(ContentType.JSON)
            .body(reservaCompleta)
            .when()
            .post()
            .then()
            .statusCode(200)
            .contentType(ContentType.JSON)
            // Validar TODOS los campos de la respuesta
            .body("userEmail", equalTo("ana.martinez@email.com"))
            .body("userName", equalTo("Ana Martínez"))
            .body("userPhone", equalTo("+57 300 1112233"))
            .body("vehicleId", equalTo(3))
            .body("vehicleBrand", equalTo("Honda"))
            .body("vehicleModel", equalTo("CR-V"))
            .body("type", equalTo("TEST_DRIVE"))
            .body("id", notNullValue())
            .body("createdAt", notNullValue());

        System.out.println(" ✅ ÉXITO: Reserva creada y validada completamente");
        System.out.println("   └─ Todos los campos fueron verificados");
        System.out.println("   └─ Cliente: Ana Martínez");
        System.out.println("   └─ Email: ana.martinez@email.com");
        System.out.println("   └─ Teléfono: +57 300 1112233");
        System.out.println("   └─ Vehículo: Honda CR-V (ID: 3)");
        System.out.println("   └─ ID generado: ✓");
        System.out.println("   └─ Fecha de creación: ✓");

System.out.println("═══════════════════════════════════════════════════════
═══════════════════\n");
    }

    @AfterAll
    public static void reporteFinal() {

System.out.println("\n ╔═══════════════════════════════════════════════════════
═══════════════════╗ ");
```

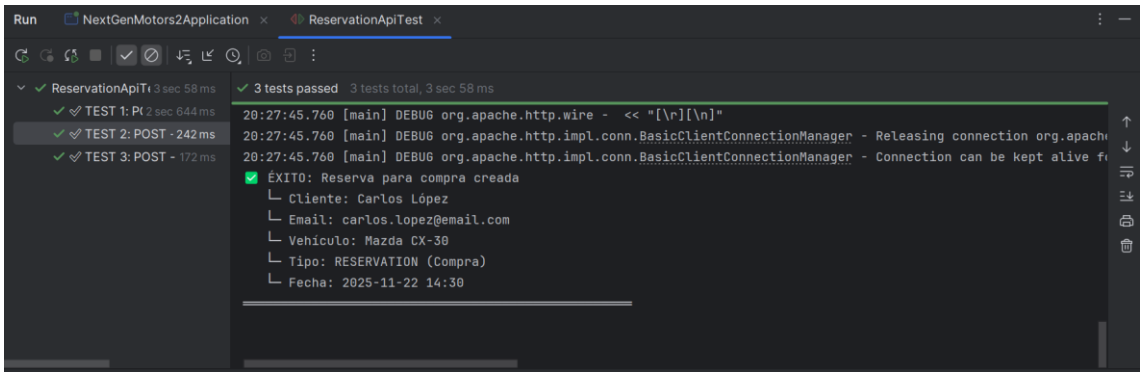
```
        System.out.println("╠          RESUMEN DE PRUEBAS                ╣");

System.out.println("╠═══════════════════════════════════════════════════
═══════════════════════╣");
        System.out.println("╠  ✅ TEST 1: Crear Prueba de Manejo          ╣");
        System.out.println("╠  ✅ TEST 2: Crear Reserva de Compra         ╣");
        System.out.println("╠  ✅ TEST 3: Validación Completa de Campos   ╣");

System.out.println("╠═══════════════════════════════════════════════════
═══════════════════════╣");
        System.out.println("╠      TODAS LAS PRUEBAS EXITOSAS ✅          ╣");
        System.out.println("╠                                            ╣");
        System.out.println("╠  Endpoint Probado:                         ╣");
        System.out.println("╠  POST http://localhost:8080/api/reservations  ╣");
        System.out.println("╠                                            ╣");
        System.out.println("╠  La API funciona correctamente ✓           ╣");

System.out.println("╚═══════════════════════════════════════════════════
═══════════════════════╝\n");
    }
}
```

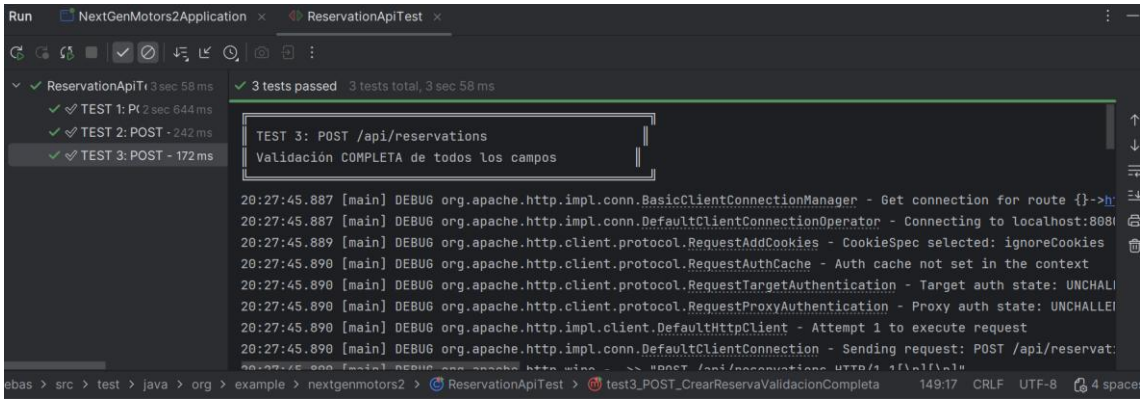
Resultado del Test**:**

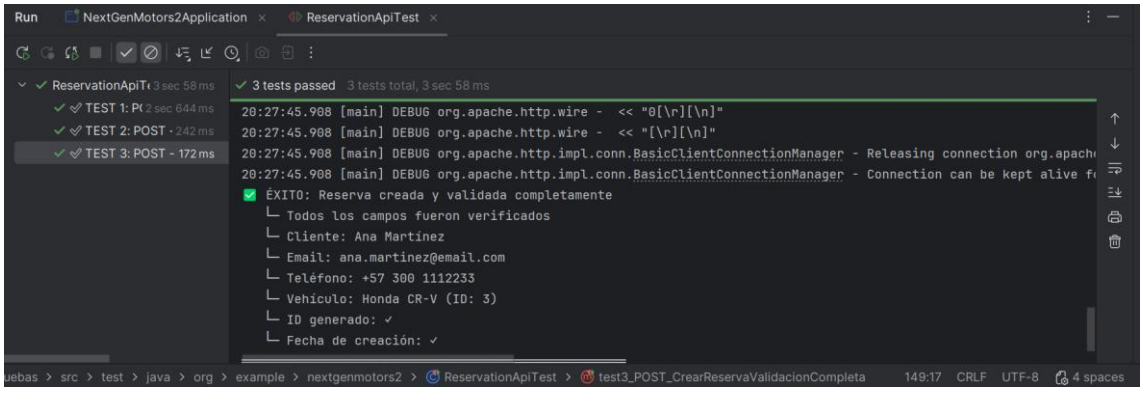












PRUEBAS DE DESPLIEGE

Sonarqube:

Codigo Base:

mvn clean verify sonar:sonar

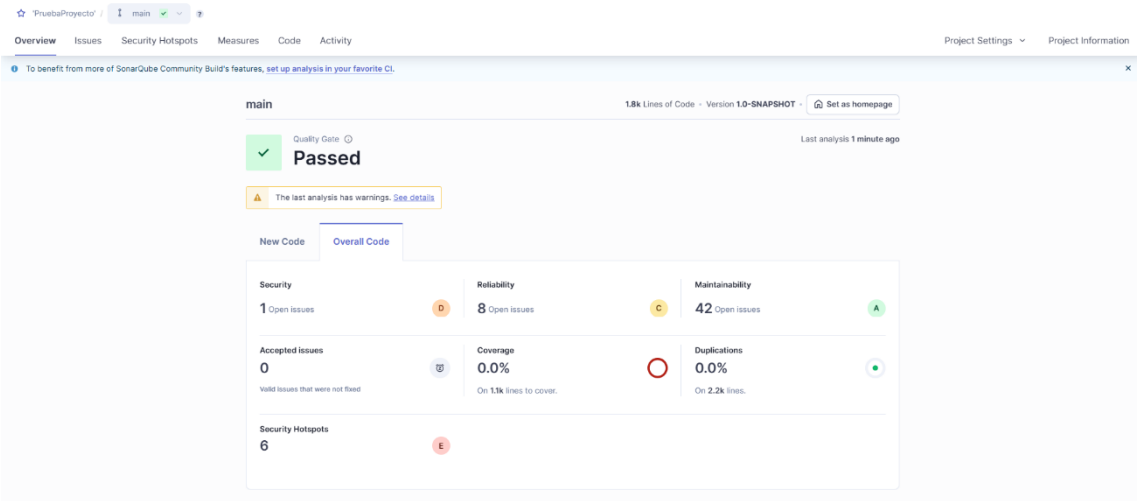
-DskipTests=true

-Dsonar.projectKey=PruebaProyecto

-Dsonar.projectName='PruebaProyecto'

-Dsonar.host.url=http://localhost:9000

-Dsonar.token=sqp_fe7dba3d40cf2b134f4b1d962420c1179b1f8738



Test API:

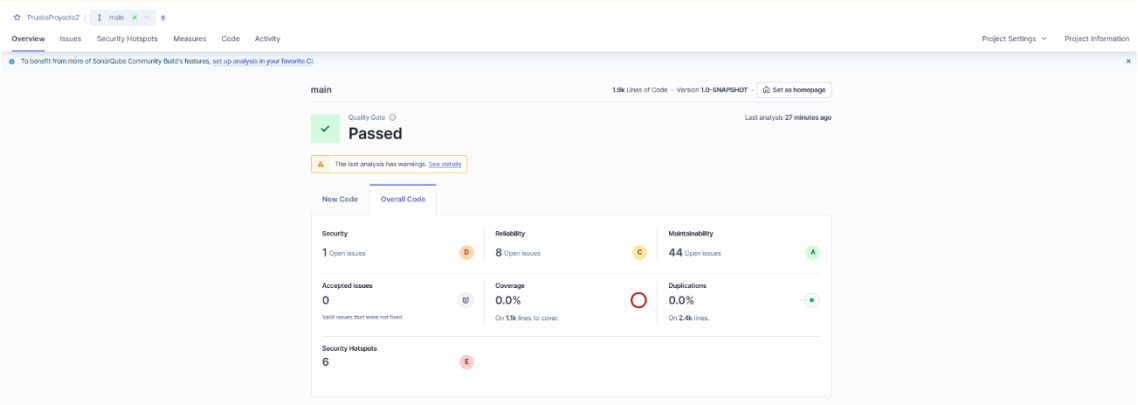
mvn clean verify sonar:sonar -DskipTests=true

-Dsonar.projectKey=PruebaProyecto2

-Dsonar.projectName='PruebaProyecto2'

-Dsonar.host.url=http://localhost:9000

-Dsonar.token=sqp_1d8aa2f9100d1e4245eedec87e9c5c112a7c5bb3



Test Selenium:

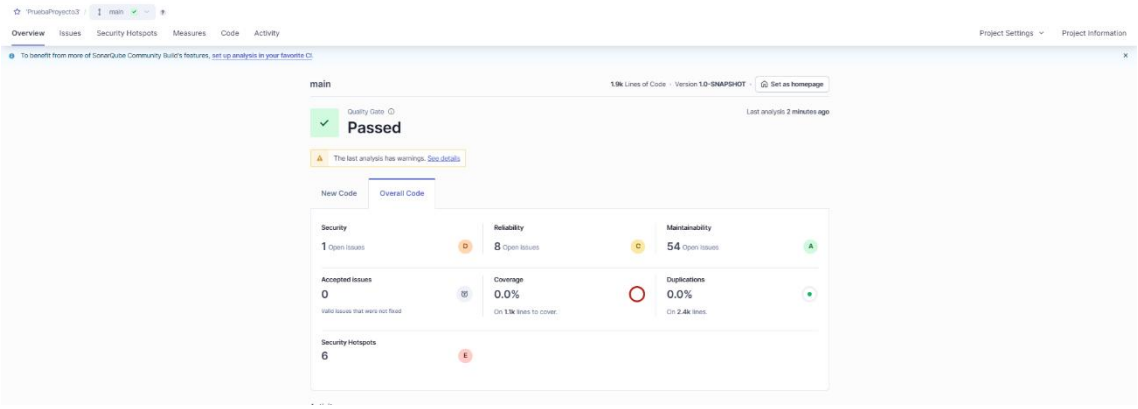
mvn clean verify sonar:sonar -DskipTests=true

-Dsonar.projectKey=PruebaProyecto3

-Dsonar.projectName='PruebaProyecto3'

-Dsonar.host.url=http://localhost:9000

-Dsonar.token=sqp_1b6735b044be3f80edd783333bd5713d5a0e9d03



PRUEBAS DE INTEGRACIÓN

```
package org.example.nextgenmotors2.backend.model.entity;

import
org.example.nextgenmotors2.backend.model.enu.ReservationType;
import
org.example.nextgenmotors2.backend.model.enu.ReservationStatus;
import org.junit.jupiter.api.Test;
import java.time.LocalDateTime;
import static org.junit.jupiter.api.Assertions.*;

public class ReservationTest {

    @Test
    public void testCreateReservation() {
        LocalDateTime date = LocalDateTime.of(2025, 11, 10, 14,
0);


org.example.nextgenmotors2.backend.model.entity.Reservation r =
new Reservation(
```

```
                "cliente@mail.com", "Daniel", "3001234567",
                101, "Tesla", "Model S", date,
ReservationType.TEST_DRIVE
        );

        assertEquals("Daniel", r.getUserName());
        assertEquals("Tesla", r.getVehicleBrand());
        assertEquals(ReservationStatus.PENDING, r.getStatus());
        assertNotNull(r.getCreatedAt());
    }
}
```

```
package org.example.nextgenmotors2.backend.service.traditional;

import
org.example.nextgenmotors2.backend.model.entity.Reservation;
import
org.example.nextgenmotors2.backend.model.enu.ReservationStatus;
import
org.example.nextgenmotors2.backend.repository.ReservationReposit
ory;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.Optional;

import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

public class ReservationServiceTest {

    private ReservationService service;
    private ReservationRepository repository;
    private VehicleDatabase vehicleDatabase;

    @BeforeEach
    void setup() throws Exception {
        repository = mock(ReservationRepository.class);
        vehicleDatabase = mock(VehicleDatabase.class);
```

```
        // Crear instancia de servicio
        service = new ReservationService();

        // Inyectar dependencias usando reflexión (ya que no hay
setters)
        var repoField =
ReservationService.class.getDeclaredField("reservationRepository
");
        repoField.setAccessible(true);
        repoField.set(service, repository);

        var vehicleField =
ReservationService.class.getDeclaredField("vehicleDatabase");
        vehicleField.setAccessible(true);
        vehicleField.set(service, vehicleDatabase);
    }

    @Test
    public void testCancelReservation() {
        Reservation reservation = new Reservation();
        reservation.setId(1L);
        reservation.setStatus(ReservationStatus.PENDING);


when(repository.findById(1L)).thenReturn(Optional.of(reservation
));
        when(repository.save(any())).thenReturn(reservation);

        Reservation result = service.cancelReservation(1L);

        assertEquals(ReservationStatus.CANCELLED,
result.getStatus());
        verify(repository, times(1)).save(any());
    }
}
```

```
package org.example.nextgenmotors2;

import
org.example.nextgenmotors2.backend.service.traditional.VehicleDa
```

```
tabase;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class VehicleDatabaseTest {

    @Test
    public void testVehiculoNoEncontrado() {
        VehicleDatabase db = new VehicleDatabase();
        Exception ex = assertThrows(RuntimeException.class, () -
> {
            db.getVehicleById(-1);
        });
        assertTrue(ex.getMessage().contains("no encontrado"));
    }
}
```