

Test documentation for CustomerController

testRegisterCustomer

- **Rationale:** Tests the core functionality of adding a new customer to the system.
- **Inputs:** A new customer with complete attributes (customer ID, name, email, phone number).
- **Expected Results:** Customer successfully stored, method call tracked, and correct data returned with all fields verified.

testUpdateCustomer

- **Rationale:** Tests the ability to modify existing customer information.
- **Inputs:** First a new customer, then an updated version with modified email and phone number.
- **Expected Results:** Verification that update was called, multiple fields changed correctly, and retrieving the updated record shows all changes.

testGetCustomer

- **Rationale:** Tests retrieval of an existing customer.
- **Inputs:** Customer ID of a previously registered customer.
- **Expected Results:** Correct customer retrieved with all fields matching and method call tracked.

testGetCustomerNotFound

- **Rationale:** Tests the edge case of requesting a non-existent customer.
- **Inputs:** Invalid customer ID.
- **Expected Results:** Null return value and method call tracked.

Sufficiency of Test Coverage

- **Complete CRUD operations:**
 - Create (register)
 - Read (get)
 - Update
 - (Delete is implemented in the mock but not explicitly tested in the controller)
- **Happy path and edge cases:**
 - Success scenarios (registering, updating, getting valid customers)
 - Error scenarios (getting non-existent customers)
- **Boundary of responsibility:**
 - Tests verify that the controller appropriately delegates to the database layer.
 - Tests confirm controller returns appropriate data structures.
- **Verification of behavior:**

- Each test validates both the functionality and the interaction pattern with dependencies.
- Tests verify all fields of the Customer model are handled correctly.

CustomerControllerTest

```
package hi.verkefni.vidmot;

import hi.verkefni.vinnsla.Customer;
import hi.verkefni.vinnsla.MockCustomerDB;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

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

public class CustomerControllerTest {
    private MockCustomerDB mockDB;
    private CustomerController controller;

    @BeforeEach
    public void setUp() {
        mockDB = new MockCustomerDB();
        controller = new CustomerController(mockDB);
    }

    @Test
    public void testRegisterCustomer() {
        Customer customer = new Customer("2404012073", "John Doe",
"john@example.com", "5551234");

        Customer result = controller.registerCustomer(customer);

        assertEquals(1, mockDB.getInsertCalls(), "Insert method should be called
once");
        assertEquals(1, mockDB.getCustomerCount(), "One customer should be in the
mock database");
        assertEquals("John Doe", result.getName(), "Name should match");
        assertEquals("john@example.com", result.getEmail(), "Email should match");
        assertEquals("5551234", result.getPhoneNumber(), "Phone number should
match");
    }

    @Test
    public void testUpdateCustomer() {
        Customer customer = new Customer("2404012073", "John Doe",
"john@example.com", "5551234");
        controller.registerCustomer(customer);

        customer.setEmail("john.doe@updated.com");
        customer.setPhoneNumber("5559876");
        Customer updatedCustomer = controller.updateCustomer(customer);

        assertEquals(1, mockDB.getUpdateCalls(), "Update method should be called
once");
        assertEquals("john.doe@updated.com", updatedCustomer.getEmail(), "Email
```

```
should be updated");
    assertEquals("5559876", updatedCustomer.getPhoneNumber(), "Phone number
should be updated");

    Customer retrievedCustomer = controller.getCustomer("2404012073");
    assertEquals("john.doe@updated.com", retrievedCustomer.getEmail(),
"Retrieved customer should have updated email");
    assertEquals("5559876", retrievedCustomer.getPhoneNumber(), "Retrieved
customer should have updated phone");
}

@Test
public void testGetCustomer() {
    Customer customer = new Customer("2404012073", "John Doe",
"john@example.com", "5551234");
    controller.registerCustomer(customer);

    mockDB.reset();
    mockDB.insert(customer);

    Customer result = controller.getCustomer("2404012073");

    assertEquals(1, mockDB.getSelectCalls(), "Select method should be called
once");
    assertNotNull(result, "Customer should be found");
    assertEquals("John Doe", result.getName(), "Name should match");
    assertEquals("5551234", result.getPhoneNumber(), "Phone number should
match");
}

@Test
public void testGetCustomerNotFound() {
    assertNull(controller.getCustomer("NONEXISTENT"), "Should return null for
non-existent customer");
    assertEquals(1, mockDB.getSelectCalls(), "Select method should be called
once");
}
}
```

How the Mock Object Simulates Real Behavior

The MockCustomerDB simulates a real database by:

- **Implementing the same interface:**
 - Extends the CustomerDB class.
 - Overrides all the required methods (selectById, insert, update, delete).
- **In-memory data storage:**
 - Uses a `HashMap<String, Customer>` to store and retrieve customer data.
 - Provides similar persistence behavior during a test execution.
- **Operation tracking:**
 - Counts method calls (insertCalls, updateCalls, etc.).
 - Allows tests to verify that controller methods call the expected database operations.
- **Predictable behavior:**
 - Returns consistent results based on stored data.
 - Doesn't depend on external systems that might change or be unavailable.
- **Reset capabilities:**
 - Provides a `reset()` method to clear state between tests.
 - Ensures test isolation by removing interference between test cases.

MockCustomerDB

```
package hi.verkefni.vinnsla;

import java.util.HashMap;
import java.util.Map;

/**
 * Mock implementation of CustomerDB for testing purposes.
 * This class simulates database operations without requiring an actual database.
 */
public class MockCustomerDB extends CustomerDB {
    private Map<String, Customer> customers = new HashMap<>();

    // Track method calls for verification in tests
    private int insertCalls = 0;
    private int updateCalls = 0;
    private int selectCalls = 0;
    private int deleteCalls = 0;

    @Override
    public Customer selectById(String customerId) {
        selectCalls++;
        return customers.get(customerId);
    }

    @Override
    public void insert(Customer customer) {
        insertCalls++;
        customers.put(customer.getCustomerId(), customer);
    }

    @Override
    public void update(Customer customer) {
        updateCalls++;
        customers.put(customer.getCustomerId(), customer);
    }

    @Override
    public void delete(String customerId) {
        deleteCalls++;
        customers.remove(customerId);
    }

    // Methods to help with test verification
    public int getInsertCalls() {
        return insertCalls;
    }

    public int getUpdateCalls() {
        return updateCalls;
    }
}
```

```
    }

    public int getSelectCalls() {
        return selectCalls;
    }

    public int getDeleteCalls() {
        return deleteCalls;
    }

    public void reset() {
        customers.clear();
        insertCalls = 0;
        updateCalls = 0;
        selectCalls = 0;
        deleteCalls = 0;
    }

    public int getCustomerCount() {
        return customers.size();
    }
}
```

Customer Model

```
package hi.verkefni.vinnsla;

public class Customer {
    private String customerId;
    private String name;
    private String email;
    private String phoneNumber;

    public Customer(String customerId, String name, String email, String
phoneNumber) {
        this.customerId = customerId;
        this.name = name;
        this.email = email;
        this.phoneNumber = phoneNumber;
    }

    public String getCustomerId() {
        return customerId;
    }

    public void setCustomerId(String customerId) {
        this.customerId = customerId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }
}
```


Hópur 7F

- **Mikael Sigurður Kristinsson** (msk14@hi.is)
- **Anton Benediktsson** (anb59@hi.is)
- **Valur Ingi Sigurðarson** (vis45@hi.is)
- **Benedikt Arnar Davíðsson** (bad9@hi.is)