Project

https://github.com/springframeworkguru/tb2g-testing-spring

Copy this code

To start up the application run from plugin jetty:run-war

Or mvn Jetty:run-war

Or ./mvnw jetty:run-war

Write test for vet controller

```
package org.springframework.samples.petclinic.web;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import org.springframework.samples.petclinic.model.Vet;
import org.springframework.samples.petclinic.model.Vets;
import org.springframework.samples.petclinic.service.ClinicService;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyString;
import static org.mockito.BDDMockito.given;
import static org.mockito.BDDMockito.then;

@ExtendWith(MockitoExtension.class)
class VetControllerTest {

    @InjectMocks
    private VetController vetController;

    @Mock
    private ClinicService clinicService;
    @Mock
    private Map<String, Object> model;

    private List<Vet> vetList = new ArrayList<>();
    @BeforeEach
    void setUp() {
        vetList.add(new Vet());
        given(clinicService.findVets()).willReturn(vetList);
    }

    @Test
    void showVetList() {
        String view = vetController.showVetList(model);
        then(clinicService).should().findVets();
        then(model).should().put(anyString(), any());
        assertThat("vets/vetList").isEqualToIgnoringCase(view);
    }

    @Test
    void showResourcesVetList() {
        Vets vets = vetController.showResourcesVetList();
        then(clinicService).should().findVets();
        assertThat(vets.getVetList()).hasSize(1);
    }
}
```

Write test for clinical service implementation class.

```
@ExtendWith(MockitoExtension.class)
class ClinicServiceImplTest {

    @Mock
    private PetRepository petRepository;
    @Mock
    private VetRepository vetRepository;

    @Mock
    private OwnerRepository ownerRepository;

    @Mock
    private VisitRepository visitRepository;

    @InjectMocks
    private ClinicServiceImpl clinicService;

    @Test
    void findPetTypes() {
        List<PetType> list = new ArrayList<>();
        given(petRepository.findPetTypes()).willReturn(list);
        Collection<PetType> result = clinicService.findPetTypes();
        then(petRepository).should().findPetTypes();
        assertThat(result).isNotNull();
    }

    @Test
    public void testFindVetsCaching() {
        // Mock the vetRepository
        Vet vet1 = new Vet();
        vet1.setId(1);
        Vet vet2 = new Vet();
        vet2.setId(2);
        Collection<Vet> vets = Arrays.asList(vet1, vet2);
        when(vetRepository.findAll()).thenReturn(vets);

        // First call should execute the method and cache the result
        Collection<Vet> result1 = clinicService.findVets();
        assertEquals(vets.size(), result1.size());

        // Second call should return the cached result without executing the
method again
        Collection<Vet> result2 = clinicService.findVets();
        assertEquals(vets.size(), result2.size());

        // Verify that the method was called only once
        verify(vetRepository, times(1)).findAll();

    }
}
```