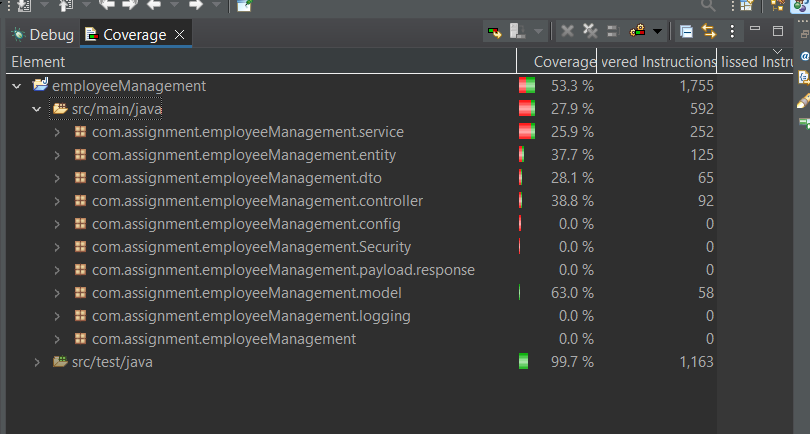
**End point creation:**

* **Api/admin/request – for list of all request.**
* **Api/manager/info – for manager info**
* **List of employee in any project**
* **Adding view all manager in admin**
* **Request end point in manager**
* **Home page of manager dashboard**
* **Linking everything. Login page to dashboard and fetch data according to the signin user rather than hardcode username word given in script(now)**
* **Updating skills**
* **Password changing**
* **Request entity and proper response for all api**
* **Hiding password from response.**
* **logger**

****

**Admin unassigned employee – manager**

**Admin – edit employee**

**Section for pending and action taken.**

**Automatically refresh after any action**

**Employee id, project id valid or not while requesting.**

**Edge cases ui reaction.**

**Fidler.**

**To do-**

Admin – approveRequest

Admin – update employee

**package com.assignment.employeeManagement.controller;**

**import static org.junit.Assert.assertEquals;**

**import static org.mockito.ArgumentMatchers.any;**

**import static org.mockito.ArgumentMatchers.eq;**

**import static org.mockito.Mockito.doNothing;**

**import static org.mockito.Mockito.when;**

**import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.delete;**

**import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;**

**import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;**

**import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.put;**

**import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;**

**import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;**

**import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.HashSet;**

**import java.util.List;**

**import java.util.Optional;**

**import java.util.Set;**

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

**import org.junit.jupiter.api.Test;**

**import org.junit.runner.RunWith;**

**import org.mockito.InjectMocks;**

**import org.mockito.Mock;**

**import org.mockito.MockitoAnnotations;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;**

**import org.springframework.boot.test.mock.mockito.MockBean;**

**import org.springframework.http.MediaType;**

**import org.springframework.http.ResponseEntity;**

**import org.springframework.test.web.servlet.MockMvc;**

**import org.springframework.test.web.servlet.setup.MockMvcBuilders;**

**import com.assignment.employeeManagement.dto.EmployeeDTO;**

**import com.assignment.employeeManagement.dto.ManagerDTO;**

**import com.assignment.employeeManagement.dto.ProjectDTO;**

**import com.assignment.employeeManagement.dto.UserAddDTO;**

**import com.assignment.employeeManagement.entity.Employee;**

**import com.assignment.employeeManagement.entity.Manager;**

**import com.assignment.employeeManagement.entity.Project;**

**import com.assignment.employeeManagement.entity.Request;**

**import com.assignment.employeeManagement.entity.User;**

**import com.assignment.employeeManagement.service.AdminService;**

**import com.assignment.employeeManagement.service.EmployeeService;**

**import com.assignment.employeeManagement.service.UserService;**

**import com.fasterxml.jackson.databind.ObjectMapper;**

**@WebMvcTest(AdminController.class)**

**public class TestAdminController {**

**@Autowired**

**private MockMvc mockMvc;**

**@MockBean**

**private EmployeeService employeeService;**

**@MockBean**

**private AdminService adminService;**

**@MockBean**

**private UserService userService;**

**@Autowired**

**private ObjectMapper objectMapper;**

**private Employee employee;**

**private EmployeeDTO employeeDTO;**

**private Manager manager;**

**private ManagerDTO managerDTO;**

**@BeforeEach**

**public void setup() {**

**employee = new Employee();**

**employee.setEmployeeId(1L);**

**employeeDTO = new EmployeeDTO();**

**employeeDTO.setUserId(1);**

**manager = new Manager();**

**manager.setManagerId(1L);**

**managerDTO = new ManagerDTO();**

**managerDTO.setUserId(1);**

**}**

**@Test**

**public void testAddEmployee() throws Exception {**

**when(employeeService.addEmployee(any(EmployeeDTO.class))).thenReturn(employee);**

**mockMvc.perform(post("/api/admin/employee")**

**.contentType(MediaType.APPLICATION\_JSON)**

**.content(objectMapper.writeValueAsString(employeeDTO)))**

**.andExpect(status().isCreated())**

**.andExpect(jsonPath("$.employeeId").value(employee.getEmployeeId()));**

**}**

**@Test**

**public void testFetchAllEmployee() throws Exception {**

**when(employeeService.getAllEmployees()).thenReturn(Arrays.asList(employee));**

**mockMvc.perform(get("/api/admin/employee")**

**.contentType(MediaType.APPLICATION\_JSON))**

**.andExpect(status().isOk())**

**.andExpect(jsonPath("$[0].employeeId").value(employee.getEmployeeId()));**

**}**

**@Test**

**public void testFetchEmployeeById() throws Exception {**

**when(employeeService.getEmployeeById(employee.getEmployeeId())).thenReturn(Optional.of(employee));**

**mockMvc.perform(get("/api/admin/employee/{employeeId}", employee.getEmployeeId())**

**.contentType(MediaType.APPLICATION\_JSON))**

**.andExpect(status().isOk())**

**.andExpect(jsonPath("$.employeeId").value(employee.getEmployeeId()));**

**}**

**@Test**

**public void testDeleteEmployeeById() throws Exception {**

**doNothing().when(adminService).deleteEmployee(employee.getEmployeeId());**

**mockMvc.perform(delete("/api/admin/employee/{employeeId}", employee.getEmployeeId())**

**.contentType(MediaType.APPLICATION\_JSON))**

**.andExpect(status().isOk())**

**.andExpect(content().string("Successfully Deleted"));**

**}**

**@Test**

**public void testAddManager() throws Exception {**

**when(adminService.addmanager(any(ManagerDTO.class))).thenReturn(manager);**

**mockMvc.perform(post("/api/admin/manager")**

**.contentType(MediaType.APPLICATION\_JSON)**

**.content(objectMapper.writeValueAsString(managerDTO)))**

**.andExpect(status().isCreated())**

**.andExpect(jsonPath("$.managerId").value(manager.getManagerId()));**

**}**

**@Test**

**public void testGetAllManager() throws Exception {**

**when(adminService.getAllManager()).thenReturn(Arrays.asList(manager));**

**mockMvc.perform(get("/api/admin/manager")**

**.contentType(MediaType.APPLICATION\_JSON))**

**.andExpect(status().isOk())**

**.andExpect(jsonPath("$[0].managerId").value(manager.getManagerId()));**

**}**

**// private MockMvc mockMvc;**

**//**

**// @Mock**

**// private EmployeeService employeeService;**

**//**

**// @Mock**

**// private UserService userService;**

**//**

**// @Mock**

**// private AdminService adminService;**

**//**

**// @InjectMocks**

**// private AdminController adminController;**

**//**

**// private ObjectMapper objectMapper = new ObjectMapper();**

**// private List<Manager> managerList;**

**// private ManagerDTO managerDTO;**

**// private Manager manager;**

**//**

**// @BeforeEach**

**// public void setup() {**

**// MockitoAnnotations.openMocks(this);**

**// mockMvc = MockMvcBuilders.standaloneSetup(adminController).build();**

**// manager = new Manager();**

**// manager.setManagerId(1L);**

**// User user = new User();**

**// manager.setUser(user);**

**//**

**//**

**// managerList = Arrays.asList(manager);**

**//**

**// managerDTO = new ManagerDTO();**

**// managerDTO.setUserId(1);**

**// }**

**//**

**// @Test**

**// public void testAddEmployee() {**

**// EmployeeDTO employeeDTO = getEmployeeDTO();**

**//**

**// Employee employee = createEmployee();**

**// when(employeeService.addEmployee(employeeDTO)).thenReturn(employee);**

**//**

**// Employee result = adminController.addEmployee(employeeDTO);**

**//**

**// assertEquals(employee, result);**

**// }**

**//**

**// @Test**

**// public void testFetchAllEmployee() {**

**// List<Employee> employeeList = createEmployeeList();**

**// when(employeeService.getAllEmployees()).thenReturn(employeeList);**

**//**

**// List<Employee> result = adminController.fetchAllEmployee();**

**//**

**// assertEquals(employeeList, result);**

**// }**

**//**

**// @Test**

**// public void testFetchEmployeeById() {**

**// Optional<Employee> employee = Optional.ofNullable(getEmployee());**

**// when(employeeService.getEmployeeById(1L)).thenReturn(employee);**

**//**

**// Optional<Employee> result = adminController.fetchEmployeeById(1L);**

**//**

**// assertEquals(employee, result);**

**// }**

**////**

**// @Test**

**// public void testDeleteEmployeeById() {**

**// ResponseEntity<String> responseEntity = ResponseEntity.ok("Successfully Deleted");**

**//**

**// ResponseEntity<String> result = adminController.deleteEmployeeById(1L);**

**//**

**// assertEquals(responseEntity, result);**

**// }**

**////**

**// @Test**

**// public void testUpdateEmployee() {**

**// EmployeeDTO employeeDTO = getEmployeeDTO();**

**// Employee employee = getEmployee();**

**// when(employeeService.updateEmployee(1L, employeeDTO)).thenReturn(employee);**

**//**

**// ResponseEntity<Employee> result = adminController.updateEmployee(1L, employeeDTO);**

**//**

**// assertEquals(ResponseEntity.ok(employee), result);**

**// }**

**////**

**//// @Test**

**//// public void testAddProject() {**

**//// ProjectDTO projectDTO = new ProjectDTO();**

**//// Project project = new Project();**

**//// when(adminService.addProject(any(ProjectDTO.class))).thenReturn(project);**

**////**

**//// Project result = adminController.addProject(projectDTO);**

**////**

**//// assertEquals(project, result);**

**//// }**

**////**

**//// @Test**

**//// public void testGetAllProject() {**

**//// List<Project> projectList = Arrays.asList(new Project(), new Project());**

**//// when(adminService.getAllProjects()).thenReturn(projectList);**

**////**

**//// List<Project> result = adminController.getAllProject();**

**////**

**//// assertEquals(projectList, result);**

**//// }**

**////**

**//// @Test**

**//// public void testAssignProject() {**

**//// Employee employee = new Employee();**

**//// when(adminService.assignProjectToEmployee(anyLong(), anyLong())).thenReturn(employee);**

**////**

**//// Employee result = adminController.assignProject(1L, 1L);**

**////**

**//// assertEquals(employee, result);**

**//// }**

**////**

**//// @Test**

**//// public void testUnassignProject() {**

**//// Employee employee = new Employee();**

**//// when(adminService.unassignProjectFromEmployee(anyLong())).thenReturn(employee);**

**////**

**//// Employee result = adminController.unassignProject(1L);**

**////**

**//// assertEquals(employee, result);**

**//// }**

**////**

**//// @Test**

**//// public void testApproveProject() {**

**//// Request request = new Request();**

**//// when(adminService.approveRequest(anyLong())).thenReturn(request);**

**////**

**//// Request result = adminController.approveProject(1L);**

**////**

**//// assertEquals(request, result);**

**//// }**

**////**

**//// @Test**

**//// public void testRejectRequest() {**

**//// Request request = new Request();**

**//// when(adminService.rejectRequest(anyLong())).thenReturn(request);**

**////**

**//// Request result = adminController.rejectRequest(1L);**

**////**

**//// assertEquals(request, result);**

**//// }**

**// @Test**

**// public void testAddProject() throws Exception {**

**// ProjectDTO projectDTO = new ProjectDTO();**

**// Project project = new Project();**

**//**

**// when(adminService.addProject(any(ProjectDTO.class))).thenReturn(project);**

**//**

**// mockMvc.perform(post("/api/admin/project")**

**// .contentType(MediaType.APPLICATION\_JSON)**

**// .content(objectMapper.writeValueAsString(projectDTO)))**

**// .andExpect(status().isOk());**

**// }**

**//**

**// @Test**

**// public void testGetAllProjects() throws Exception {**

**// List<Project> projects = Arrays.asList(new Project(), new Project());**

**//**

**// when(adminService.getAllProjects()).thenReturn(projects);**

**//**

**// mockMvc.perform(get("/api/admin/projects"))**

**// .andExpect(status().isOk());**

**// }**

**//**

**// @Test**

**// public void testAssignProject() throws Exception {**

**// Employee employee = new Employee();**

**//**

**// when(adminService.assignProjectToEmployee(eq(1L), eq(1L))).thenReturn(employee);**

**//**

**// mockMvc.perform(put("/api/admin/employee/1/assignProject/1"))**

**// .andExpect(status().isOk());**

**// }**

**//**

**// @Test**

**// public void testUnassignProject() throws Exception {**

**// Employee employee = new Employee();**

**//**

**// when(adminService.unassignProjectFromEmployee(eq(1L))).thenReturn(employee);**

**//**

**// mockMvc.perform(put("/api/admin/employee/1/unassignProject"))**

**// .andExpect(status().isOk());**

**// }**

**//**

**// @Test**

**// public void testApproveProject() throws Exception {**

**// Request request = new Request();**

**//**

**// when(adminService.approveRequest(eq(1L))).thenReturn(request);**

**//**

**// mockMvc.perform(put("/api/admin/request/1/approve"))**

**// .andExpect(status().isOk());**

**// }**

**//**

**// @Test**

**// public void testRejectRequest() throws Exception {**

**// Request request = new Request();**

**//**

**// when(adminService.rejectRequest(eq(1L))).thenReturn(request);**

**//**

**// mockMvc.perform(put("/api/admin/request/1/reject"))**

**// .andExpect(status().isOk());**

**// }**

**//**

**// @Test**

**// public void testSaveUser() throws Exception {**

**// UserAddDTO userAddDTO = new UserAddDTO();**

**// User user = new User();**

**//**

**// when(userService.addUser(any(UserAddDTO.class))).thenReturn(user);**

**//**

**// mockMvc.perform(post("/api/admin/user")**

**// .contentType(MediaType.APPLICATION\_JSON)**

**// .content(objectMapper.writeValueAsString(userAddDTO)))**

**// .andExpect(status().isOk());**

**// }**

**//**

**// @Test**

**// public void testGetAllRequests() throws Exception {**

**// List<Request> requests = Arrays.asList(new Request(), new Request());**

**//**

**// when(adminService.getAllRequest()).thenReturn(requests);**

**//**

**// mockMvc.perform(get("/api/admin/request"))**

**// .andExpect(status().isOk());**

**// }**

**// public EmployeeDTO getEmployeeDTO() {**

**// EmployeeDTO employeeDTO = new EmployeeDTO();**

**// employeeDTO.setEmployeeId(1L);**

**// employeeDTO.setFullName("Rahul");**

**// Set<String> skills = new HashSet<>();**

**// skills.add("Java");**

**// skills.add("Spring");**

**// employeeDTO.setSkills(skills);**

**// return employeeDTO;**

**// }**

**//**

**// public Employee createEmployee() {**

**// Employee employee1 = new Employee();**

**// employee1.setEmployeeId(1L);**

**// employee1.setFullName("Rahul");**

**// Set<String> skillsSet = new HashSet<>();**

**// skillsSet.add("Java");**

**// skillsSet.add("Spring");**

**// employee1.setSkills(skillsSet);**

**// return employee1;**

**// }**

**//**

**// public List<Employee> createEmployeeList(){**

**// Employee employee1 = new Employee();**

**// employee1.setEmployeeId(1L);**

**// employee1.setFullName("Rahul");**

**// Set<String> skillsSet = new HashSet<>();**

**// skillsSet.add("Java");**

**// skillsSet.add("Spring");**

**// employee1.setSkills(skillsSet);**

**//**

**//**

**// Employee employee2 = new Employee();**

**// employee1.setEmployeeId(2L);**

**// employee1.setFullName("Kishan");**

**// Set<String> skillsSet2 = new HashSet<>();**

**// skillsSet2.add("React");**

**// skillsSet2.add("MongoDB");**

**// employee1.setSkills(skillsSet2);**

**// List<Employee> employeeList = new ArrayList<>();**

**// employeeList.add(employee1);**

**// employeeList.add(employee2);**

**// return employeeList;**

**// }**

**//**

**// public Employee getEmployee() {**

**// Employee employee1 = new Employee();**

**// employee1.setEmployeeId(1L);**

**// employee1.setFullName("Rahul");**

**// Set<String> skillsSet = new HashSet<>();**

**// skillsSet.add("Java");**

**// skillsSet.add("Spring");**

**// employee1.setSkills(skillsSet);**

**// return employee1;**

**// }**

**//**

**// @Test**

**// public void testGetAllManager() throws Exception {**

**// when(adminService.getAllManager()).thenReturn(managerList);**

**//**

**// mockMvc.perform(get("/api/admin/manager"))**

**// .andExpect(status().isOk())**

**// .andExpect(content().contentType(MediaType.APPLICATION\_JSON));**

**//**

**// }**

**//**

**// @Test**

**// public void testAddManager() throws Exception {**

**// when(adminService.addmanager(any(ManagerDTO.class))).thenReturn(manager);**

**//**

**// mockMvc.perform(post("/api/admin/manager")**

**// .contentType(MediaType.APPLICATION\_JSON)**

**// .content(new ObjectMapper().writeValueAsString(managerDTO)))**

**// .andExpect(status().isOk())**

**// .andExpect(content().contentType(MediaType.APPLICATION\_JSON));**

**//**

**// }**

**}**

**Manager controller:**

**package com.assignment.employeeManagement.controller;**

**import static org.mockito.ArgumentMatchers.any;**

**import static org.mockito.ArgumentMatchers.eq;**

**import static org.junit.Assert.assertEquals;**

**import static org.mockito.Mockito.when;**

**import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;**

**import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;**

**import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;**

**import java.security.Principal;**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.HashSet;**

**import java.util.List;**

**import java.util.Set;**

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

**import org.junit.jupiter.api.Test;**

**import org.junit.runner.RunWith;**

**import org.mockito.InjectMocks;**

**import org.mockito.Mock;**

**import org.mockito.MockitoAnnotations;**

**import org.mockito.junit.MockitoJUnitRunner;**

**import org.springframework.http.MediaType;**

**import org.springframework.test.web.servlet.MockMvc;**

**import org.springframework.test.web.servlet.setup.MockMvcBuilders;**

**import com.assignment.employeeManagement.controller.ManagerController;**

**import com.assignment.employeeManagement.dto.ManagerInfoDTO;**

**import com.assignment.employeeManagement.entity.Employee;**

**import com.assignment.employeeManagement.entity.Manager;**

**import com.assignment.employeeManagement.entity.Project;**

**import com.assignment.employeeManagement.entity.Request;**

**import com.assignment.employeeManagement.service.ManagerService;**

**import com.fasterxml.jackson.databind.ObjectMapper;**

**@RunWith(MockitoJUnitRunner.class)**

**public class ManagerControllerTest {**

**private MockMvc mockMvc;**

**@Mock**

**private ManagerService managerService;**

**@InjectMocks**

**private ManagerController managerController;**

**private ObjectMapper objectMapper = new ObjectMapper();**

**@BeforeEach**

**public void setUp() {**

**MockitoAnnotations.openMocks(this);**

**mockMvc = MockMvcBuilders.standaloneSetup(managerController).build();**

**}**

**@Test**

**public void testGetAllEmployee() {**

**List<Employee> employeeList = createEmployeeList();**

**when(managerService.getAllEmployees()).thenReturn(employeeList);**

**List<Employee> result = managerController.getAllEmployee();**

**assertEquals(employeeList, result);**

**}**

**@Test**

**public void testGetUnassignedEmployees() {**

**List<Employee> employeeList = createEmployeeList();**

**when(managerService.getUnassignedEmployees()).thenReturn(employeeList);**

**List<Employee> result = managerController.getUnassignedEmployees();**

**assertEquals(employeeList, result);**

**}**

**@Test**

**public void testFilterAllEmployeesBySkills() {**

**List<Employee> employeeList = createEmployeeList();**

**List<String> skills = Arrays.asList("Java", "Spring");**

**when(managerService.filterEmployeesBySkills(skills)).thenReturn(employeeList);**

**List<Employee> result = managerController.filterAllEmployeesBySkills(skills);**

**assertEquals(employeeList, result);**

**}**

**@Test**

**public void testGetAllManager() throws Exception {**

**List<Manager> managers = Arrays.asList(new Manager(), new Manager());**

**when(managerService.getAllManagers()).thenReturn(managers);**

**mockMvc.perform(get("/api/manager/manager"))**

**.andExpect(status().isOk());**

**}**

**@Test**

**public void testGetAllProject() throws Exception {**

**List<Project> projects = Arrays.asList(new Project(), new Project());**

**when(managerService.getAllProjects()).thenReturn(projects);**

**mockMvc.perform(get("/api/manager/project"))**

**.andExpect(status().isOk());**

**}**

**// @Test**

**// public void testRequestEmployeesForProject() throws Exception {**

**// Request request = new Request();**

**//**

**// when(managerService.requestEmployeesForProject(any(String.class), any(Long.class), any(List.class))).thenReturn(request);**

**//**

**// mockMvc.perform(post("/api/manager/request/employees")**

**// .param("email", "test@example.com")**

**// .param("projectId", "1")**

**// .contentType(MediaType.APPLICATION\_JSON)**

**// .content(objectMapper.writeValueAsString(Arrays.asList(1L, 2L))))**

**// .andExpect(status().isOk());**

**// }**

**@Test**

**public void testGetManagerInfo() throws Exception {**

**ManagerInfoDTO manager = new ManagerInfoDTO();**

**Principal principal = () -> "test@example.com";**

**when(managerService.getManagerInfo(any(Principal.class))).thenReturn(manager);**

**mockMvc.perform(get("/api/manager/info")**

**.principal(principal))**

**.andExpect(status().isOk());**

**}**

**@Test**

**public void testGetEmployeeListByProjectId() throws Exception {**

**List<Employee> employees = Arrays.asList(new Employee(), new Employee());**

**when(managerService.getAllEmployeeByProject(eq(1L))).thenReturn(employees);**

**mockMvc.perform(get("/api/manager/employee/project/1"))**

**.andExpect(status().isOk());**

**}**

**public List<Employee> createEmployeeList(){**

**Employee employee1 = new Employee();**

**employee1.setEmployeeId(1L);**

**employee1.setFullName("Rahul");**

**Set<String> skillsSet = new HashSet<>();**

**skillsSet.add("Java");**

**skillsSet.add("Spring");**

**employee1.setSkills(skillsSet);**

**Employee employee2 = new Employee();**

**employee1.setEmployeeId(2L);**

**employee1.setFullName("Kishan");**

**Set<String> skillsSet2 = new HashSet<>();**

**skillsSet2.add("React");**

**skillsSet2.add("MongoDB");**

**employee1.setSkills(skillsSet2);**

**List<Employee> employeeList = new ArrayList<>();**

**employeeList.add(employee1);**

**employeeList.add(employee2);**

**return employeeList;**

**}**

**}**

**Employee controller:**

**package com.assignment.employeeManagement.controller;**

**import static org.hamcrest.CoreMatchers.any;**

**import static org.junit.Assert.assertEquals;**

**import static org.mockito.Mockito.when;**

**import org.junit.jupiter.api.Test;**

**import java.security.Principal;**

**import java.util.ArrayList;**

**import java.util.HashSet;**

**import java.util.List;**

**import java.util.Set;**

**import org.assertj.core.util.Arrays;**

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

**import org.junit.runner.RunWith;**

**import org.mockito.InjectMocks;**

**import org.mockito.Mock;**

**import org.mockito.MockitoAnnotations;**

**import org.mockito.junit.MockitoJUnitRunner;**

**import com.assignment.employeeManagement.controller.EmployeeController;**

**import com.assignment.employeeManagement.entity.Employee;**

**import com.assignment.employeeManagement.service.EmployeeService;**

**@RunWith(MockitoJUnitRunner.class)**

**public class EmployeeControllerTests {**

**@Mock**

**private EmployeeService employeeService;**

**@InjectMocks**

**private EmployeeController employeeController;**

**@BeforeEach**

**public void setUp() {**

**MockitoAnnotations.openMocks(this);**

**}**

**@Test**

**public void testFetchAllEmployee() {**

**Employee employee1 = new Employee();**

**employee1.setEmployeeId(1L);**

**employee1.setFullName("Rahul");**

**Set<String> skillsSet = new HashSet<>();**

**skillsSet.add("Java");**

**skillsSet.add("Spring");**

**employee1.setSkills(skillsSet);**

**Employee employee2 = new Employee();**

**employee1.setEmployeeId(2L);**

**employee1.setFullName("Kishan");**

**Set<String> skillsSet2 = new HashSet<>();**

**skillsSet2.add("React");**

**skillsSet2.add("MongoDB");**

**employee1.setSkills(skillsSet2);**

**List<Employee> employeeList = new ArrayList<>();**

**employeeList.add(employee1);**

**employeeList.add(employee2);**

**when(employeeService.getAllEmployees()).thenReturn(employeeList);**

**List<Employee> result = employeeController.fetchAllEmployee();**

**assertEquals(employeeList, result);**

**}**

**@Test**

**public void testGetInfo() {**

**Principal principal = () -> "user";**

**Employee employee = createEmployee();**

**when(employeeService.getEmployeeInfo(principal)).thenReturn(employee);**

**Employee result = employeeController.getInfo(principal);**

**assertEquals(employee, result);**

**}**

**@Test**

**void testUpdateSkills() {**

**Principal principal = () -> "user";**

**List<String> skills = new ArrayList<>();**

**skills.add("Flutter");**

**skills.add("Android");**

**Employee employee = createEmployee();**

**when(employeeService.updateSkills(principal, skills)).thenReturn(employee);**

**Employee result = employeeController.updateSkills(principal, skills);**

**assertEquals(employee, result);**

**}**

**public Employee createEmployee() {**

**Employee employee1 = new Employee();**

**employee1.setEmployeeId(1L);**

**employee1.setFullName("Rahul");**

**Set<String> skillsSet = new HashSet<>();**

**skillsSet.add("Java");**

**skillsSet.add("Spring");**

**employee1.setSkills(skillsSet);**

**return employee1;**

**}**

**}**