**Lean Platform Technologies**

Backend Developer Intern - Task Assignment

Test cases:

This file would typically be named something like ConsultantControllerTest.java:

@RunWith(SpringRunner.class)

@WebMvcTest(ConsultantController.class)

public class ConsultantControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private ConsultantService consultantService;

@Test

public void testGetAllConsultantRequests() throws Exception {

ConsultantRequest request1 = new ConsultantRequest("1", "John Doe", "john@example.com");

ConsultantRequest request2 = new ConsultantRequest("2", "Jane Smith", "jane@example.com");

List<ConsultantRequest> requests = Arrays.asList(request1, request2);

given(consultantService.getAllConsultantRequests()).willReturn(requests);

mockMvc.perform(get("/api/consultant-requests")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$", hasSize(2)))

.andExpect(jsonPath("$[0].name", is("John Doe")))

.andExpect(jsonPath("$[1].name", is("Jane Smith")));

}

// Write similar test methods for other API endpoints

}

Below is an example of a test case file for the provided backend code. This file would typically be named something like ConsultantControllerTest.java:

java

Copy code

@RunWith(SpringRunner.class)

@WebMvcTest(ConsultantController.class)

public class ConsultantControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private ConsultantService consultantService;

@Test

public void testGetAllConsultantRequests() throws Exception {

ConsultantRequest request1 = new ConsultantRequest("1", "John Doe", "john@example.com");

ConsultantRequest request2 = new ConsultantRequest("2", "Jane Smith", "jane@example.com");

List<ConsultantRequest> requests = Arrays.asList(request1, request2);

given(consultantService.getAllConsultantRequests()).willReturn(requests);

mockMvc.perform(get("/api/consultant-requests")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$", hasSize(2)))

.andExpect(jsonPath("$[0].name", is("John Doe")))

.andExpect(jsonPath("$[1].name", is("Jane Smith")));

}

// Write similar test methods for other API endpoints

}

In this test file:

@RunWith(SpringRunner.class) and @WebMvcTest(ConsultantController.class) are annotations to configure Spring Boot for testing the MVC controller.

@Autowired MockMvc mockMvc is used to simulate HTTP requests to the controller.

@MockBean private ConsultantService consultantService is used to mock the service dependency.

The testGetAllConsultantRequests() method tests the getAllConsultantRequests() API endpoint by mocking the service to return a list of consultant requests and then asserting the response status, JSON content, etc.

Similar test methods can be written for other API endpoints such as approving/rejecting consultant requests, adding consultant details, fetching consultants, and retrieving web app statistics.

Remember to import the necessary static methods (Mockito.\*, MockMvc.\*, HttpStatus.\*, etc.) and classes (Test, Before, RunWith, etc.) as needed.

These test cases ensure that the controller behaves as expected under various scenarios, helping to maintain the integrity of the backend functionality. You can expand upon these tests to cover additional edge cases and scenarios as required.