DA\_RESTASSURED\_Appointment\_Management\_Use\_Case-L1-D02-03

# Introduction to RestAssured

RestAssured is a Java DSL for easy testing of REST services. It supports GET, POST, PUT, DELETE, OPTIONS, PATCH and HEAD requests and can be used to validate and verify the response of these requests.

# Introduction

# Conduct Guidelines

# Use Case Template

# Use Case ID: UC-Appointment01

# RestAssured Test Case: Appointment Management API Test

Objective: To ensure the Appointment Management API operates correctly, securely, and efficiently, covering all functionalities from booking to cancellation.

# Introduction

This document details the specifications for use cases and guidelines for proper conduct during assessments. It is aimed at ensuring a thorough understanding and compliance with the best practices in API testing, focusing on RestAssured tool generate the API testing Metrics.

# Conduct Guidelines

# Important Notice to All Assessors:

Prior to commencing the assessment, please ensure that you have read and understood the following conduct guidelines. Failure to adhere to these guidelines may result in disqualification from the assessment process.

# Resource Restricted Usage:

Access to resources like GMAIL, Google ChatGPT, Quora, Stack Overflow, official emails, search engines, and AI tools will be restricted or monitored.

# Monitoring and Compliance:

Your activities will be monitored. Accessing blocked sites or using unauthorized tools will be recorded and may lead to disqualification.

# RestAssured Standards:

In RestAssured API testing, it's essential to use descriptive naming, document thoroughly, and logically structure tests for clarity and maintainability. Emphasize data-driven testing through parameterization, leverage scripting for dynamic adjustments, and ensure validations with assertions. Focus on efficient error handling, logging, and resource-conscious configurations to optimize performance testing, ultimately enhancing API reliability and efficiency.

# Use Case Details

### Use Case ID: UC-HealthEMR01 Title: Appointment Management Actors: User, SASA Health EMR System, Healthcare Provider, Appointment System

### Description: This RestAssured test case aims to validate the Appointment Booking feature of the SASA Health EMR System. The test will emulate user actions for booking a new patient appointment, ensuring the system handles the process accurately under various conditions. The test will cover patient registration, appointment scheduling, and validation of the booking process.

# Pre-Test Setup and Prerequisites:

Verify RestAssured Installation and Environment Setup (Test Case 1): Ensure that RestAssured is installed correctly and is operational.

Verify that all necessary plugins and dependencies for the test are in place.

RestAssured is set up.  
The healthcare appointment system API is accessible and functional.  
Test data for appointments is prepared.

Results to be logged for reporting and into summary of execution report.

Use the attached XLS to write the output of RestAssured test cases.



details for Extract single node Depth First Search

use the token for login to the app - Authorization Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..SuWA5HdKhjvsv50WEcjaH7Y8r7H-VC7bq5yb6P2i-C8

**Preconditions:**

The user (patient or healthcare provider) is registered and logged into the system.  
The user has access to the appointment management functionalities.

The patient registration system is fully operational.  
Hospital staff have secure access and are authorized to operate the registration module.

Use the URL – http://healthapp.yaksha.com/

Username: admin / Password: pass123

# Description:

This use case involves managing patient appointments in a healthcare system, including listing doctors, searching for patients, creating, retrieving, and updating appointments.

# Use Case: Comprehensive Appointment Management System

# Business Requirements:

# Appointment Scheduling and Confirmation:

Patients can schedule appointments with available doctors for specific dates and times. The system confirms appointments back to patients and updates doctors' schedules accordingly.

# Appointment Updates:

Patients or administrative staff can update appointment details, including rescheduling or canceling. The system reflects these changes in real-time, ensuring up-to-date scheduling.

# Clash Detection and Management:

The system checks for scheduling conflicts for both patients and doctors, preventing double-booking and optimizing doctor availability.

# Department and Membership Services:

Patients can view applicable departments for appointments, enhancing navigation and scheduling efficiency. Additionally, membership details, such as discounts and benefits, are accessible, improving patient engagement and loyalty.

# Patient and Doctor Data Integrity:

Ensures accurate and up-to-date patient and doctor information within appointments, including automated EMPI (Enterprise Master Patient Index) generation and visit code updates for patient visits, reinforcing data integrity and system reliability.

# Main Success Scenario & Basic Flow :

1. Login Workflow

1. The hospital staff member navigates to the HMS login page.
2. They enter their credentials and are granted access to the system.

2. Appointment Booking Workflow:

a) Upon successful login, the staff member selects the 'Appointment' section and chooses 'Book Appointment' to initiate a new appointment entry.

b) They input the patient's personal and contact details into the corresponding fields: First Name, Last Name, Gender, Age, Contact Number, along with the reason for the visit.

c) The staff member selects the desired appointment date and time from the available slots, and assigns the patient to the appropriate department and doctor.

d) After reviewing the details for accuracy, they submit the booking request by clicking on 'Add Appointment.'

e) The system processes the request and, upon successful booking, confirms the appointment, which is then visible in the 'Appointment Booking List' for future reference and management.

# Basic Flow:

# List of API Test Cases for Appointment Management

1. Get All Applicable Doctors

Description: Retrieve a list of doctors available for appointments.

Endpoint: GET http://healthapp.yaksha.com/api/Visit/AppointmentApplicableDoctors

Expected Output: Array of doctors' details.

Test: Verify the status code is 200 and the response body contains an array of doctors' details.

2. Search Patients

Description: Search for patients by name or hospital number.

Endpoint: GET http://healthapp.yaksha.com/api/Patient/SearchPatientForNewVisit

Expected Output: Array of matching patient details.

Test: Confirm the status code is 200 and the response body includes matching patient details.

3. Create Appointment

Description: Submit details to schedule a new appointment.

Endpoint: POST http://healthapp.yaksha.com/api/Appointment/AddAppointment

Expected Output: Confirmation with appointment details.

Test: Ensure the status code is 201 and the response body includes the newly created appointment details.

4. Get Appointments

Description: Fetch appointments based on criteria like date, performerId, and status.

Endpoint: GET http://healthapp.yaksha.com/api/Appointment/Appointments

Expected Output: List of appointments matching the criteria.

Test: Validate the status code is 200 and the response body contains the list of appointments.

5. Update Appointment

Description: Update details for an existing appointment.

Endpoint: PUT http://healthapp.yaksha.com/api/Appointment/UpdateAppointment

Expected Output: Confirmation of updated appointment details.

Test: Check the status code is 200 or 204 and the response, if applicable, shows updated appointment details.

6. List Appointments within Date Range and by Performer

Description: Retrieve appointments within a specified date range and by a specific performer.

Endpoint: GET http://healthapp.yaksha.com/api/Appointment/Appointments?FromDate=<date>&ToDate=<date>&performerId=<id>

Expected Output: Array of appointment objects within the date range and by the performer.

Test: Confirm the status code is 200 and the response body contains an array of appointment objects.

7. Check for Clashing Appointments

Description: Check if there are any clashing appointments for a given date and performer.

Endpoint: GET http://healthapp.yaksha.com/api/Appointment/CheckClashingAppointment?patientId=<id>&requestDate=<date>&performerId=<id>

Expected Output: Boolean indicating if there is a clash.

Test: Verify the status code is 200 and the response body indicates if a clash exists.

8. List Appointment Applicable Departments

Description: Fetch a list of departments where appointments can be made.

Endpoint: GET http://healthapp.yaksha.com/api/Appointment/AppointmentApplicableDepartments

Expected Output: List of departments.

Test: Ensure the status code is 200 and the response body includes a list of departments.

9. Get Membership Details

Description: Obtain membership details using a membershipTypeId.

Endpoint: GET http://healthapp.yaksha.com/api/Appointment/MembershipDetail?membershipTypeId=<id>

Expected Output: Membership details.

Test: Check the status code is 200 and the response body contains membership details.

10. List Patients with Appointments

Description: Retrieve a list of patients with appointments for a given date and performer.

Endpoint: GET http://healthapp.yaksha.com/api/Appointment/PatientsWithAppointments?performerId=<id>&requestDate=<date>

Expected Output: List of patients with appointments.

Test: Confirm the status code is 200 and the response body includes a list of patients with appointments.

11. Update Patient in Appointment

Description: Update the patient information for a specific appointment.

Endpoint: PUT http://healthapp.yaksha.com/api/Appointment/UpdatePatientInAppointment?appointmentId=<id>&patientId=<id>

Expected Output: Success message indicating the update.

Test: Verify the status code is 200 and the response body contains a success message.

12. Update Appointment Status

Description: Change the status of an appointment to a new value.

Endpoint: PUT http://healthapp.yaksha.com/api/Appointment/AppointmentStatus?appointmentId=<id>&status=<status>&PerformerId=<id>&PerformerName=<name>

Expected Output: Confirmation of the appointment status update.

Test: Ensure the status code is 200 and the response body confirms the status update.

Note: Replace placeholders like <date>, <id>, and <status> with actual values as per your test scenarios.

### General Testing Guidelines:

### Endpoint Testing: Validate both successful responses and error handling for each endpoint. This includes testing for expected success responses and various error conditions like invalid inputs or unauthorized access attempts.

### Security Compliance: Ensure that tests verify the security measures in place, particularly for endpoints handling sensitive data. Utilize SSL verification and header checks to ensure secure data transmission.

### Dynamic Data Use: Implement dynamic variables in tests to simulate real-world scenarios, such as creating a patient record followed by booking an appointment for the same patient.

### Data-Driven Testing: Leverage RestAssured's support for data-driven testing to automate tests across various input combinations, enhancing coverage.

### Response Validation: Beyond HTTP status codes, verify the response body for accuracy, ensuring that the returned data matches the expected results based on the request payload.

### Post-Conditions:

Data Integrity: Ensure that any modifications made through the API are accurately reflected in the system's data, with appointments and patient information updated as expected.

User Feedback: Confirm that the API provides clear feedback on the operation's outcome, including success messages and detailed error information for troubleshooting.

### Delivery

**Working Code:** Implement Java classes for testing each API endpoint, including GET, POST (with JSON and XML), PUT, and DELETE requests. Use serialization for request bodies, validate responses against schemas, and extract data using JSON and XML paths.

**Folder Structure:** Organize your project into directories for source code, tests, models, configurations, and resources. Include a dedicated folder for logs and reports.

**Use Case Coverage:** Ensure your tests cover all functional scenarios described in the use case, including dynamic data handling, error and edge case handling, and security checks.

**Reports and Logs**: Configure RestAssured to generate detailed logs for each test execution and use a reporting tool or framework integration (like Maven Surefire or TestNG with ExtentReports) to produce readable reports highlighting pass/fail status, response times, and test coverage.

**Documentation**: Include a README file explaining the project setup, how to run tests, and how to interpret the logs and reports.