FUNCTIONAL AND PERFORMANCE TESTING

1. Introduction

1.1 Purpose

This document outlines the performance testing strategy for the Airline Management System built on the Salesforce platform. The purpose is to ensure the system performs optimally under expected and peak load conditions, delivering a seamless experience to users including airline staff, travel agents, and customers.

1.2 Scope

Performance testing will cover major modules:

- Flight Booking
- Ticket Cancellation
- Flight Search
- Loyalty Program Management
- Check-in and Boarding Pass Generation
- Admin Reporting Dashboards

Salesforce components under test:

- Lightning Web Components (LWC)
- Apex Controllers
- Salesforce APIs
- Third-party Integrations

2. Performance Testing Objectives

- Validate system response times for critical transactions.
- Identify bottlenecks under load.
- Ensure system stability during high traffic.
- Confirm scalability across regions and user roles.

3. Test Environment

Component	Description
Platform	Salesforce Lightning Experience
Edition	Salesforce Enterprise
Tools Used	JMeter, BlazeMeter, Salesforce Debug Logs, Developer Console
Test Data	100k+ mock users, flight records, and booking scenarios
Integrations	Payment Gateway, Airline Scheduling APIs, Email Services

4. Types of Performance Testing

Type	Purpose
Load Testing	Check performance under expected user load
Stress Testing	Evaluate system behavior under extreme load
Soak Testing	Test system stability over extended periods
Spike Testing	Analyze reaction to sudden traffic spikes
Scalability Testing	Verify response with increasing user base

5. Key Performance Metrics

Metric	Target
Response Time	< 3 seconds for 95% of transactions
Throughput	1000 transactions/min under peak load

Error Rate	< 1%
CPU/Memory Usage	Within Salesforce org limits
API Latency	< 2 seconds per call (on average)
Transaction Success Rate	≥ 99.5%

6. Test Scenarios

Scenario	Description	Load
Flight Search	500 concurrent users search flights	Load
Booking Transaction	300 users book flights with payment	Load
Cancellation Process	100 users cancel bookings	Stress
Loyalty Redemption	200 users redeem points	Spike
Dashboard Load	50 admins access reports	Soak

7. Test Data Requirements

- Users: Regular passengers, Frequent Flyers, Admins
- Flights: 1,000+ records across various time zones
- Bookings: Historical and upcoming booking data
- Rewards: Loyalty balances and tier statuses

8. Test Execution Plan

8.1 Load Testing

- Simulate 1,000 users using JMeter or BlazeMeter
- Monitor page load and API response time

8.2 Stress Testing

- Gradually increase user load until system failure
- Identify breaking point and memory usage

8.3 Soak Testing

- Run normal load for 8+ hours
- Check for performance degradation

9. Monitoring and Reporting Tools

Tool	Purpose
Salesforce Debug Logs	Backend performance monitoring
JMeter/BlazeMeter	Simulate load, analyze response
Developer Console	Real-time performance monitoring
Salesforce Optimizer	Org performance recommendations

10. Risks and Mitigation

Risk	Mitigation
Governor Limits Breach	Optimize Apex and SOQL queries
API Rate Limits	Implement caching and retry logic
Integration Timeout	Use asynchronous calls or queues
Data Skew	Test with realistic, distributed data sets

11. Exit Criteria

- ≥ 95% test cases passed under expected load
- Response time under 3 seconds for key operations
- No critical or high defects remain open
- Performance reports signed off by QA Lead

12. Conclusion

This performance testing document aims to ensure the Airline Management System on Salesforce can handle large volumes of users and data with reliability and speed, maintaining high user satisfaction and operational efficiency.

To demonstrate testing for an **Airlines Management System** built on **Salesforce**, you should follow a structured approach that covers the **functional, integration, UI, security, and performance aspects** of the application. Here's a step-by-step guide tailored for **Salesforce-based testing**, with examples relevant to airline operations (like bookings, check-ins, loyalty, etc.).

Define the Testing Scope

Focus on key modules of the airline system:

- **Booking & Ticketing**
- **Flight Scheduling**
- **Customer Profiles / Loyalty Program**
- **Check-in Management**
- **Payment Integration**
- **CRM / Customer Support**

Prepare the Test Environment

- Use **Salesforce Sandbox** for testing.
- Set up **test data**: dummy passengers, flight schedules, fare rules.
- Ensure integration with external systems (payment gateway, airport systems, etc.) is mocked or sandboxed.

Write and Execute Test Cases

📌 Functional Test Cases

Booking Module

- TC001: Verify that a customer can book a flight from city A to city B.
- TC002: Verify error message when booking with invalid credit card.
- TC003: Verify loyalty points are credited after successful booking.

Check-in Module

- TC004: Verify online check-in is available 24 hrs before flight.
- TC005: Ensure seat selection is saved and updated in customer profile.

📌 UI Testing (Lightning Experience)

Use **Salesforce Lightning UI** testing tools like:

- **Salesforce Inspector**
- **Selenium with WebDriver**
- **Provar (Salesforce testing tool)**

Test Case:

- TC006: Ensure the "Book Now" button is visible and responsive.
- TC007: On mobile, verify responsive design for the Check-in screen.

📌 Apex Unit Testing

If there's custom logic written in Apex, write **@isTest** annotated methods to ensure coverage:

```
@isTest
private class BookingTest {
   static testMethod void testCreateBooking() {
      Booking__c booking = new Booking__c(
         Passenger__c = 'Test Passenger',
         Flight__c = 'Test Flight'
      );
      insert booking;
```

```
        System.assertNotEquals(null, booking.Id);
    }
}
```

Salesforce requires **75%+ code coverage** to deploy Apex code.

- Verify users with “Ground Staff” role cannot cancel flights.
- Ensure PII like passport numbers are encrypted at rest.

Performance Testing

Test performance using:

- Salesforce Performance Assistant
- Lightning Page Load Times

Check:

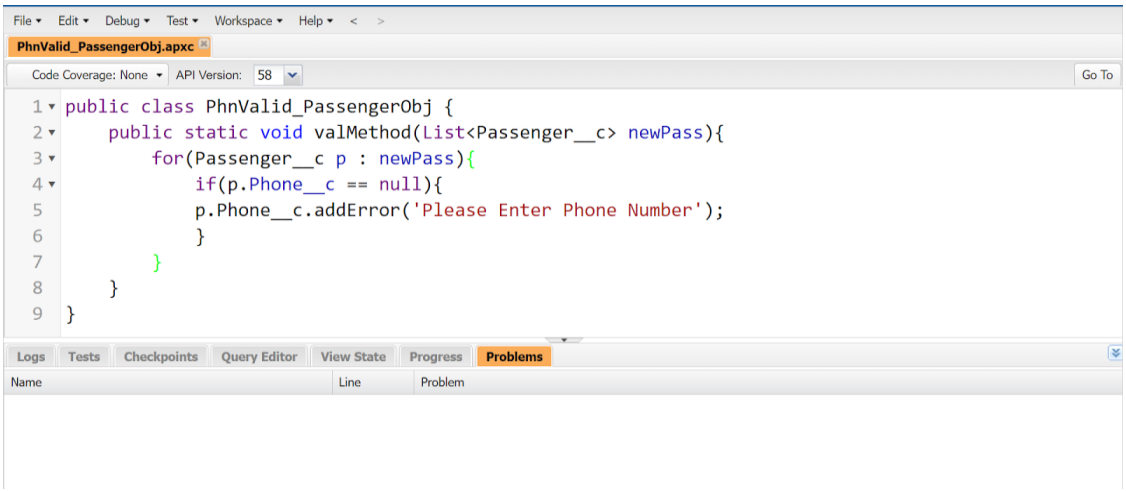
- TC010: Page load time for booking screen is < 2 seconds.
- TC011: System handles 1000 bookings/hour without timeout.

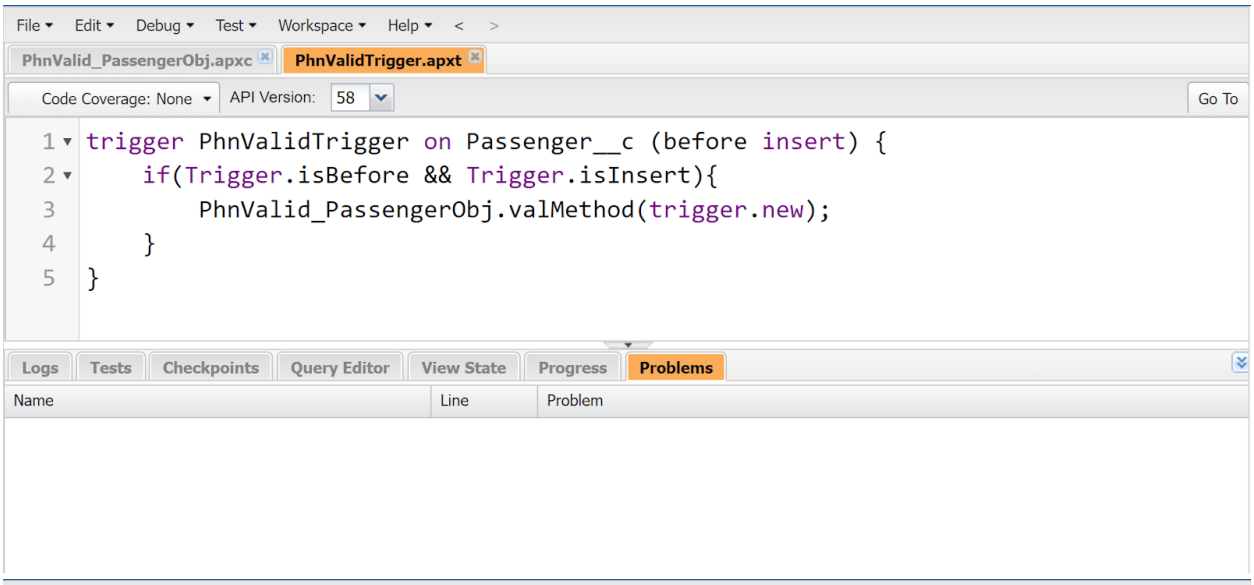
- **orce)**

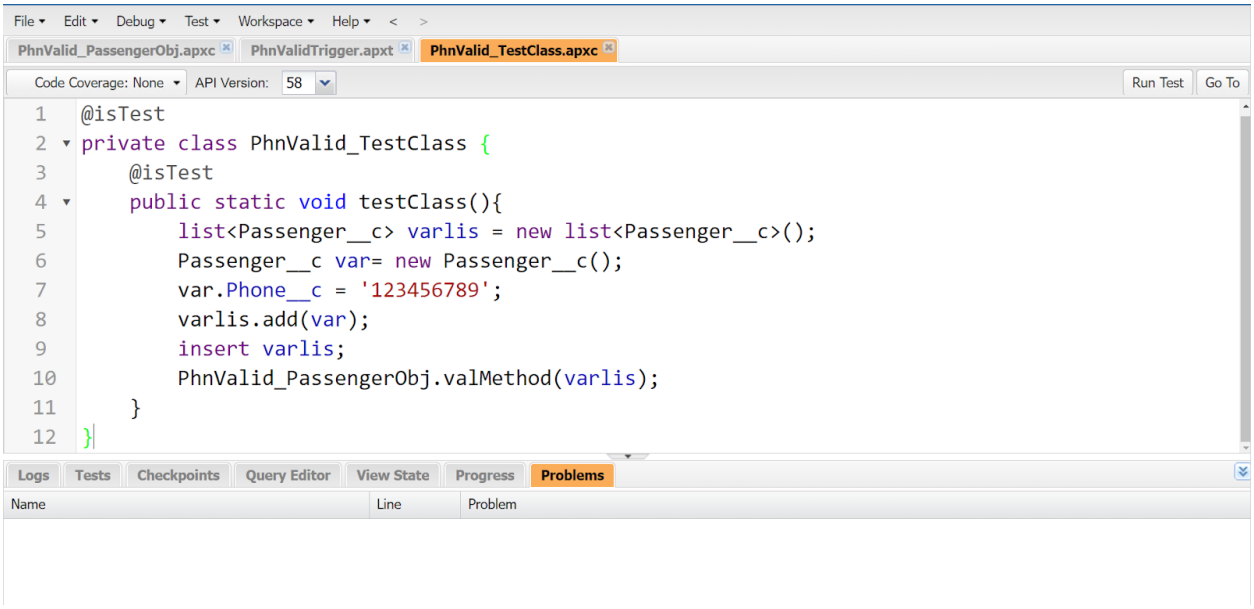
Reporting & Validation

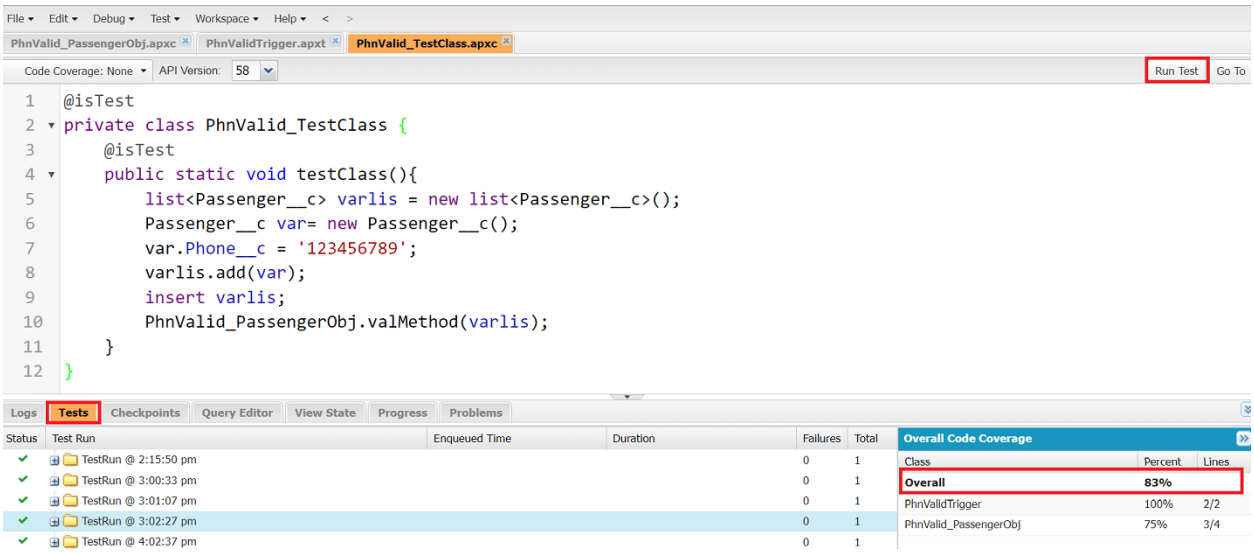
- Use **Salesforce Test Execution Reports**
- Track bugs via **JIRA** or built-in Salesforce Case Management
- Conduct **UAT (User Acceptance Testing)** with airline staff

OUTPUT:


File Edit Debug Test Workspace Help < >
PhnValid_PassengerObj.apxc
Code Coverage: None API Version: 58 Go To
1 public class PhnValid_PassengerObj {
2 public static void valMethod(List<Passenger__c> newPass){
3 for(Passenger__c p : newPass){
4 if(p.Phone__c == null){
5 p.Phone__c.addError('Please Enter Phone Number');
6 }
7 }
8 }
9 }
Logs Tests Checkpoints Query Editor View State Progress Problems
Name Line Problem


File Edit Debug Test Workspace Help
PhnValid_PassengerObj.apxc
PhnValidTrigger.apxt
Code Coverage: None
API Version: 58
Go To
trigger PhnValidTrigger on Passenger__c (before insert) {
if(Trigger.isBefore && Trigger.isInsert){
PhnValid_PassengerObj.valMethod(trigger.new);
}
}
Logs
Tests
Checkpoints
Query Editor
View State
Progress
Problems
Name
Line
Problem


File Edit Debug Test Workspace Help
PhnValid_PassengerObj.apxc
PhnValidTrigger.apxt
PhnValid_TestClass.apxc
Code Coverage: None
API Version: 58
Run Test
Go To
@isTest
private class PhnValid_TestClass {
@isTest
public static void testClass(){
list<Passenger__c> varlis = new list<Passenger__c>();
Passenger__c var= new Passenger__c();
var.Phone__c = '123456789';
varlis.add(var);
insert varlis;
PhnValid_PassengerObj.valMethod(varlis);
}
}
Logs
Tests
Checkpoints
Query Editor
View State
Progress
Problems
Name
Line
Problem


File
Edit
Debug
Test
Workspace
Help
PhnValid_PassengerObj.apxc
PhnValidTrigger.apxt
PhnValid_TestClass.apxc
Code Coverage: None
API Version:
58
Run Test
Go To
@isTest
private class PhnValid_TestClass {
@isTest
public static void testClass(){
list<Passenger__c> varlis = new list<Passenger__c>();
Passenger__c var= new Passenger__c();
var.Phone__c = '123456789';
varlis.add(var);
insert varlis;
PhnValid_PassengerObj.valMethod(varlis);
}
}
Logs
Tests
Checkpoints
Query Editor
View State
Progress
Problems
Status
Test Run
Enqueued Time
Duration
Failures
Total
TestRun @ 2:15:50 pm
TestRun @ 3:00:33 pm
TestRun @ 3:01:07 pm
TestRun @ 3:02:27 pm
TestRun @ 4:02:37 pm
Overall Code Coverage
Class
Percent
Lines
Overall
83%
PhnValidTrigger
100%
2/2
PhnValid_PassengerObj
75%
3/4