Airline Reservation System - System Integration Test Plan

Version 1.0

Prepared for: FIT5171 Assessment 2

Date: April 30, 2025

---

1. Introduction

1.1 Test Objectives

The objective of this testing activity is to ensure the Airline Ticket Booking System operates reliably and conforms to functional and non-functional requirements. The primary goal is to validate that passengers can seamlessly browse flights, choose tickets, and complete bookings with accurate pricing, proper input validation, and secure data handling.

- Verify that each class behaves correctly in isolation (unit testing) and in interaction with other components (integration testing).
- Confirm input validation for email, mobile numbers, passport formats, seat assignment, and ticket status.
- Ensure discount policies and service tax calculations are applied correctly under varying age conditions.
- Validate system behavior under both successful and failed booking scenarios.
- Employ mocking (using Mockito) where applicable, particularly for interactions between Passenger and Person, or Ticket and Flight/Passenger.
- Guarantee robustness by testing edge cases and handling invalid or missing data gracefully.
- Provide traceable evidence of test coverage through assertions, mock verifications, and scenario-driven integration tests.
- This structured testing process ensures that all business rules are respected and the system provides a trustworthy user experience before final deployment.

1.2 Scope of Testing

The system integration test will cover:

- This test plan encompasses all core functionalities and integrations of the enhanced Airline Reservation System, including:
- Airplane and Flight Validation: Verifying correct creation and retrieval of Airplane and Flight objects, enforcing mandatory fields, date/time formats, duplicate prevention, and seat-map integrity (rows A–J, seven seats each).
- Person & Passenger Validation: Ensuring Person and Passenger entities enforce required fields, name and gender restrictions, email patterns, phone number formats (Australia plus two additional countries), and passport validations.
- Ticket Management: Testing creation, status transitions (available ↔ booked), age-based discounts, non-negative pricing, and 12% service tax application in Ticket.
- Booking Workflow: End-to-end scenarios for direct and transfer flight bookings via TicketSystem.chooseTicket and buyTicket, including handling of invalid IDs, empty inputs, and fallback routing.
- Collections Operations: Confirming FlightCollection and TicketCollection correctly add, retrieve, list, and validate entries.
- Continuous Integration: Validating that the GitLab CI pipeline (.gitlab-ci.yml with build and test stages) reliably compiles and runs the full suite on each push, with visible pass/fail statuses.
- Both unit tests (with Mockito mocks for external dependencies) and integration tests (real interactions among objects) will be executed to guarantee comprehensive coverage of business rules and error handling across the system.

1.3 System Overview

The Airline Reservation System streamlines online flight bookings by managing flights, aircraft, passengers, and tickets in a cohesive application. Key components include:

Flight Management: Creation and maintenance of flight records, enforcing unique IDs, valid dates/times, and preventing duplicates.

Airplane Catalog: Each aircraft carries an ID, model name, passenger and crew seat counts, plus a seat-map (rows A–J, 7 seats per row) for occupancy tracking.

User Entities: Person and Passenger classes store demographic and contact details, with strict validation of names, genders, emails, phone formats (Australia + two additional countries), and passports.

Ticketing Engine: Ticket objects hold flight, passenger, pricing (including age-based discounts and 12% service tax), and booking status, with TicketSystem orchestrating booking flows for direct and multi-leg itineraries.

Collections: FlightCollection and TicketCollection manage in-memory lists, offering add, retrieve, list, and validation operations.

Testing Infrastructure: A JUnit 5 & Mockito–backed suite covers all validation, logic, and integration paths, while a GitLab CI pipeline compiles, tests, and reports build health on every commit.

1.4 Definitions/Acronyms

- ARS: Airline Reservation System
- Test Case (TC): A specific scenario with defined inputs and expected outcomes.
- JUnit: Java Unit Testing Framework
- Maven: Project management and build automation tool
- CI (Continuous Integration) automatically building and testing code on each commit
- Mockito Java library for creating mock objects in unit tests

2. Approach

2.1 Assumptions/Constraints

- The ARS system is deployed in a local environment.
- The ARS codebase and all dependencies (Java, Maven, JUnit 5, Mockito) are installed and configured locally.
-
- A GitLab CI runner is available and correctly registered to execute our .gitlab-ci.yml pipeline.
- Team members have already extended the code under TDD following Phase 1 deliverables.
- Mockito will be used to stub/mimic dependencies (e.g. mocking Airplane.getAirPlaneInfo in Flight tests).
-
- Black-box test design (Boundary Value, Equivalence, Decision Table) suffices to cover required validation rules.
-
- The GitLab CI pipeline (.gitlab-ci.yml, config.toml) is already registered and can build + run tests on every push.
- CI jobs are limited to the one configured runner (per config.toml), so pipelines must complete under typical runner resource limits.
-
- All tests must execute automatically via mvn test in the CI environment; no manual steps permitted.

2.2 Coverage

The test coverage will include:

- Core booking and ticketing functionalities.
- Airplane – attribute validation, seat-map generation, passenger/crew counts.
- Flight – field non-null checks, date/time formats, duplicate-flight prevention.
- Person/Passenger – mandatory fields, name-format rules, gender options, phone/email/passport patterns.
- FlightCollection & TicketCollection – addition, retrieval, and lookup logic.

- Pipeline build and test stages under GitLab CI, ensuring every commit triggers compilation and full test suit.

2.3 Test Tools

- JUnit 5: For writing and executing unit and integration tests.
- Maven: To manage dependencies and run test cases.
- IntelliJ IDEA: IDE for development and testing.

2.4 Test Types

- Unit Testing : Verify each class in isolation (e.g., AirplaneTest, FlightTest, PassengerTest, TicketTest).
- Integration Testing : Check interactions between components (e.g., booking flows in TicketSystemIntegrationTest, end-to-end scenarios).
- Mock-Based Testing : Use Mockito to stub out collaborators, exercising only the class under test in unit tests.
- Continuous Integration Testing : Pipeline jobs that run mvn compile and mvn test to ensure regressions are caught immediately.
- Black-Box Techniques : Boundary Value Testing on ID, age, seat counts. Equivalence Partitioning for formats (emails, phone, passport). Decision Table Testing for combinations of purchase decision and routing logic

3. Test Plan

Name	Title	Responsibilities
Gaurav	Independent Tester	Design and executed test case 6, 7, 8, 9
Amita	Independent Tester	Design and executed test case 5, 10, 11, 12
Varun	Independent Tester	Design and executed test cases 1, 2, 3, 4

3.1 Major Tasks and Deliverables

- Task 1: Test Case Design
- Task 2: Environment Setup
- Task 3: Test Execution

3.2 Testing Environmental

- Operating System: Windows 10: Windows 10 is a widely used operating system developed by Microsoft. It provides a robust and user-friendly environment for development and testing.
- IDE: IntelliJ IDEA: IntelliJ IDEA is a powerful and versatile integrated development environment (IDE) developed by JetBrains.
- Build Tool: Maven: Apache Maven is a build automation tool used primarily for Java projects. It simplifies project management by automating the compilation, packaging, and testing processes.
- Testing Framework: JUnit 5. JUnit 5 is a widely used testing framework for Java applications. It is designed to write and run repeatable automated tests.
- Dependency/Mocking Framework: Mockito (e.g. 4.4.0) for your unit-test mocks
- CI/CD & Runner:
  GitLab-CI (or GitHub Actions) version / runner tags you use
  Shell environment (bash, pwsh) or Docker image tag for your jobs

4. Features to be Tested

- End-to-end ticket booking (selection, purchase confirmation, seat assignment)

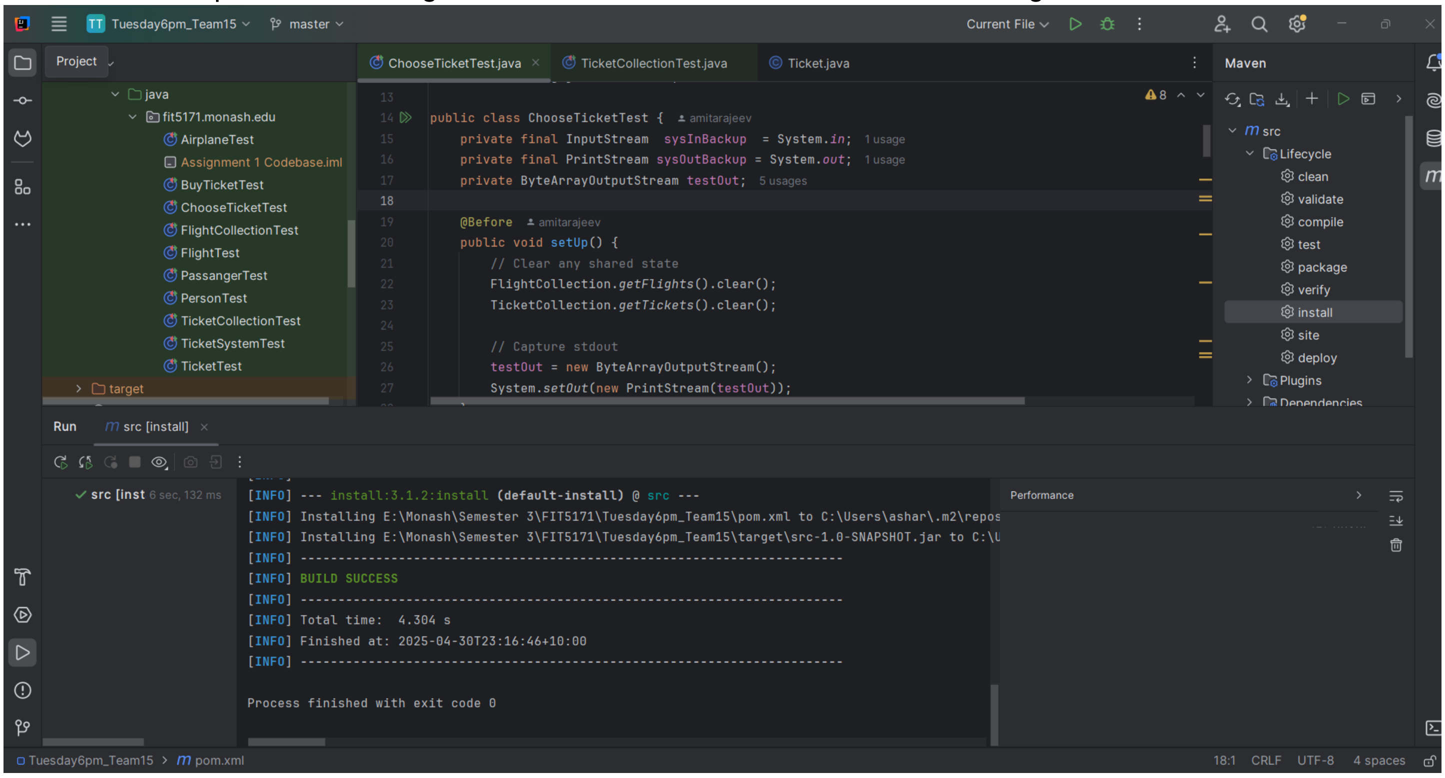
- Direct and transfer flight routing and fallback logic
- Validation of passenger data (names, contact, passport, email formats)
- Pricing and discount calculations (age-based fares, service tax application)
- Seat inventory updates and availability reporting (business vs. economy)

5. Testing Procedures

5.1 Test Execution

- Ran the full test suite locally, ensuring both unit and integration tests completed with zero failures.
- Execute all tests using Maven
- Upon committing to GitLab, the CI pipeline automatically executed the build and test stages; we confirmed both jobs succeeded before finalizing.
- Record the results in the test report.

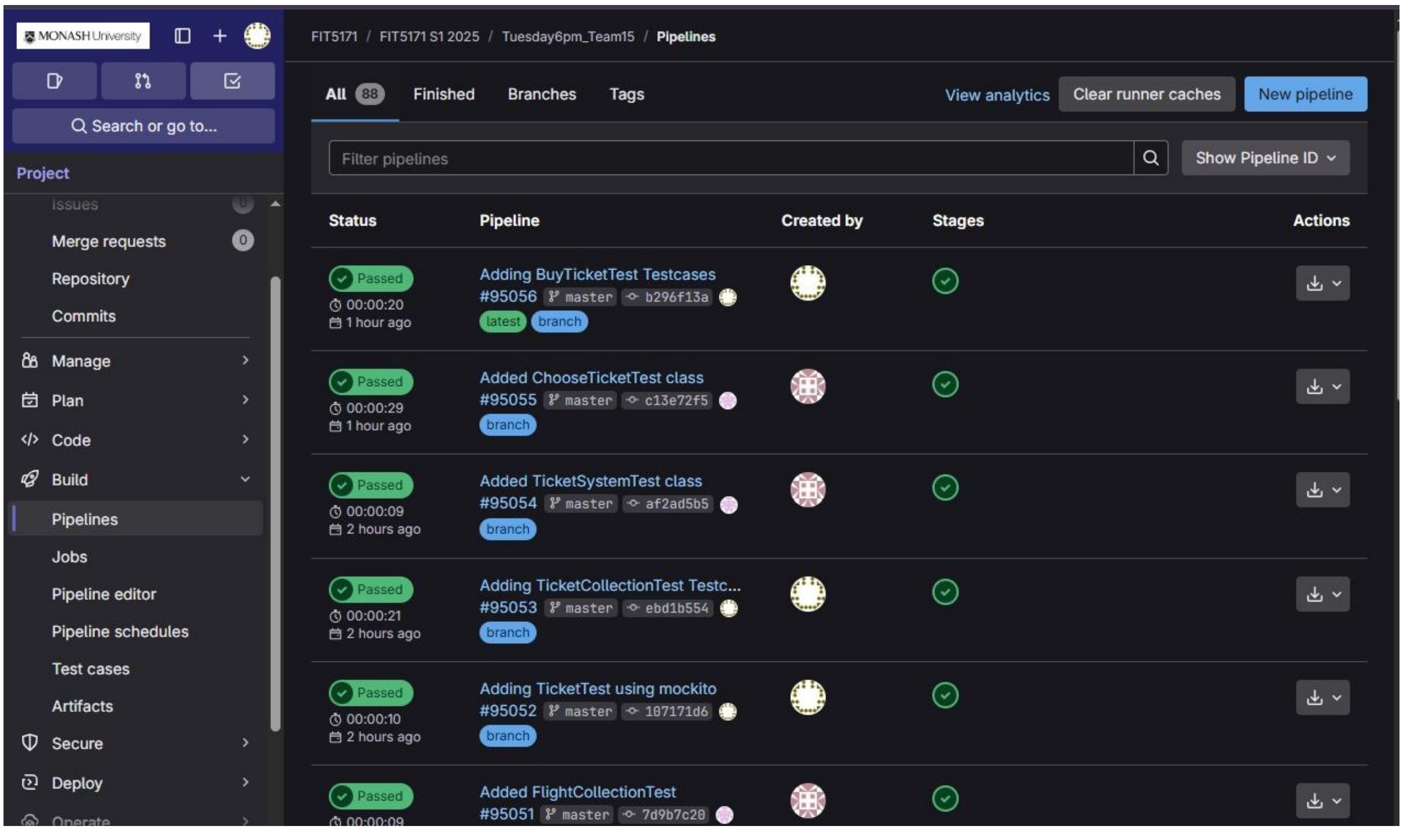
Successful Setup of local working environment and execution of test cases using Maven.



```java
public class ChooseTicketTest {  amitarajeev
    private final InputStream  sysInBackup  = System.in;  1 usage
    private final PrintStream sysOutBackup = System.out;  1 usage
    private ByteArrayOutputStream testOut;  5 usages

    @Before  amitarajeev
    public void setUp() {
        // Clear any shared state
        FlightCollection.getFlights().clear();
        TicketCollection.getTickets().clear();

        // Capture stdout
        testOut = new ByteArrayOutputStream();
        System.setOut(new PrintStream(testOut));
```

```
[INFO] --- install:3.1.2:install (default-install) @ src ---
[INFO] Installing E:\Monash\Semester 3\FIT5171\Tuesday6pm_Team15\pom.xml to C:\Users\ashar\.m2\repos
[INFO] Installing E:\Monash\Semester 3\FIT5171\Tuesday6pm_Team15\target\src-1.0-SNAPSHOT.jar to C:\U
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.304 s
[INFO] Finished at: 2025-04-30T23:16:46+10:00
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

Successful working of GitLab CI/CD Pipeline.



FIT5171 / FIT5171 S1 2025 / Tuesday6pm_Team15 / Pipelines

Status	Pipeline	Created by	Stages	Actions
Passed 00:00:20 1 hour ago	Adding BuyTicketTest Testcases #95056 master b296f13a latest branch		✓	
Passed 00:00:29 1 hour ago	Added ChooseTicketTest class #95055 master c13e72f5 branch		✓	
Passed 00:00:09 2 hours ago	Added TicketSystemTest class #95054 master af2ad5b5 branch		✓	
Passed 00:00:21 2 hours ago	Adding TicketCollectionTest Testc... #95053 master ebd1b554 branch		✓	
Passed 00:00:10 2 hours ago	Adding TicketTest using mockito #95052 master 107171d6 branch		✓	
Passed 00:00:09	Added FlightCollectionTest #95051 master 7d9b7c20		✓	

5.2 Unit test cases and Order of execution

Test Case ID	Test Class	Method	Description	Input	Expected Output	Actual Output	Status
TC_01	AirplaneTest	testValidAirplaneCreation	Validate constructor and seat map initialization	Airplane(1,"B737",10,50,5)	Object created; seatMap size=70; all seats unoccupied	No exceptions	Pass
TC_02	AirplaneTest	testAirplaneIDValidation	Reject non-positive airplane IDs	new Airplane(0,"M",1,1,1)	ILlegalArgumentException	Error message logged	Pass
TC_03	FlightTest	testNullAndEmptyValidations	Reject null/empty flight fields	new Flight(1,null,"A","C1","Co",ts1,ts2,plane)	ILlegalArgumentException	Error message logged	Pass
TC_04	FlightTest	testDuplicateFlightThrows	Prevent duplicate flight ID+code	new Flight (100,"B","A","X","Co	ILlegalArgumentException on 2nd	Error message logged	Pass

				",ts1, ts2,plane) twice			
TC_05	PersonTest	testMandatoryFields	Person requires all fields; names must start with letter	new Person("1John","Doe",30,"Man")	ILlegalArgumentException	Error message logged	Pass
TC_06	PassengerTest	testMobileNumberFormats	Accept AU and two other country formats	"04xx xxx xxx", "+61 4xx xxx xxx", "+1-555-1234"	Set phoneNumber with country code	No exceptions	Pass
TC_07	PassengerTest	testEmailAndPassportValidation	Reject invalid email/passport	"abc@","123"	ILlegalArgumentException	Error message logged	Pass
TC_08	TicketTest	testSaleByAgeChild	Apply 50% discount for age<15	ticket.setPrice(200) with passenger.age=10	price==100×1.12 tax applied	No exceptions	Pass
TC_09	TicketTest	testServiceTaxAlwaysApplied	Always add 12% tax	setPrice(100)	price==112	No exceptions	Pass
TC_10	FlightCollectionTest	testAddAndRetrieveFlight	Add flight and get by cities	FlightCollection.add(f); getFlightInfo("A","B")	returns f	No exceptions	Pass
TC_11	TicketCollectionTest	testAddAndRetrieveTicket	Add ticket and get by ID	TicketCollection.add(t); getTicketInfo(t.id)	returns t	No exceptions	Pass
TC_12	TicketSystemTest	testChooseTicketFallbackTransfer	Fallback to transfer route when no direct flight	flights A→C, C→B; chooseTicket("A","B")	returns true	No exceptions	Pass