### 1. Agile Methodology Overview

This project adopts the Agile methodology, specifically a Scrum-based approach adapted for solo development. Agile emphasizes flexibility, collaboration, and iterative progress. It supports evolving requirements through continuous feedback, short sprints, and adaptive planning.

Why Agile suits this project: - Incremental Features: New functions like booking references and database integration are added progressively. - UI Refinement: A menu-based interface benefits from iterative user/tutor feedback. - Divisible Tasks: Key features (booking, seat checking, reference generation) are modular and sprint-friendly.

### 2. Sprint Timeline & Feature Goals

| Sprint | Timeline | Goal |
| --- | --- | --- |
| 1 | Days 1–3 | Set up seat matrix and main menu (F, R, X, S layout) |
| 2 | Days 4–6 | Implement core booking functions (check, book, free, display) |
| 3 | Days 7–9 | Add extended feature (e.g., search by reference) |
| 4 | Days 10–12 | Add booking reference generator (8-char ID) |
| 5 | Days 13–15 | Connect SQLite database (store/retrieve bookings) |
| 6 | Days 16–17 | Final testing, debugging, documentation, commit |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Reason:

* Early Detection and Correction of Errors: After each cycle, I will test new features. If errors are found, I can fix them immediately before progressing, ensuring software quality.
* Adaptability to Requirement Changes: Agile allows me to integrate future requirements (e.g., database integration, booking references) smoothly without rebuilding the system from scratch.
* Self-Managed Iterations Improve Focus: Since I am the only developer, I can break down work into manageable cycles with clear goals and outcomes, reducing overwhelm and improving time management.

### 3. Features to Implement

* Display seat status (‘F’, ‘R’, ‘X’, ‘S’)
* Check seat availability
* Book a seat (only if available)
* Cancel a booking (restore to ‘F’, remove DB data)
* Display current booking status (menu-driven output)
* Generate unique 8-character booking reference (random, alphanumeric)
* Store passenger info (name, passport number, seat, ref)
* Integrate SQLite DB for data storage and retrieval
* Search booking by reference code
* Booking summary: total reserved, available, and non-bookable seats
* Git/GitHub for version control

### 4. Development Cycle Per Sprint

**Planning Phase** - Define target feature(s) - Identify required structural/logic changes

**Development Phase** - Write modular code for each function - Add comments for clarity and documentation

**Testing Phase** - Test valid, invalid, and edge case inputs - Ensure each menu feature behaves as expected

**Debugging/Refactoring** - Fix bugs promptly - Clean up code formatting and naming

**Version Control** - One commit at end of each sprint with clear message

### 5. Bug Handling & Fix Strategy

* All functions tested immediately after development
* Includes edge cases: booking unavailable seat, invalid ID input, etc.
* Bugs are logged and fixed before next sprint
* No bug fixes postponed to avoid delay

### 6. Responding to Changing Requirements

If new requirements emerge: - Add them to the product backlog - Assign to upcoming sprint - Evaluate structural changes needed - Update only relevant modules (not full rebuild)

### 7. GitHub Version Control Process

**Create GitHub Repository** 1. Go to https://github.com 2. Click + → New repository 3. Name: “Apache-airlines-”, visibility: Public 4. Clone to local using GitHub Desktop

**Local File Workflow** - Edit files in IDE (Spyder) - GitHub Desktop auto-detects modified files

**Git Commit Process** 1. git status: see changed files 2. git diff: view differences 3. git add final project.py 4. git commit -m "Add booking reference and SQLite integration" 5. git push origin main

**GitHub Repo Link**: <https://github.com/JingNing524/Apache-airlines->

### 8. Git Commit Test Proof (Manual Interaction Walkthrough)

#### Test Case: Successful Booking

* ✅ Seat A0 booked successfully
* ✅ Booking reference generated
* ✅ Entry saved in SQLite (booking.db)

#### Test Case: Booking an Already Reserved Seat

* ❌ Error shown: “Seat is already reserved”

#### Test Case: Freeing a Seat

* ✅ Seat A0 freed
* ✅ DB entry deleted

#### Test Case: Invalid Input (X or S)

* ❌ Error message for aisle/storage

#### Test Case: Display Booking Summary

* ✅ Output: count of Free, Reserved, X, S

#### Test Case: Search Booking by Reference

Test Case: Invalid Seat Format (e.g., A10, Z9)  
❌ Previous: Seat ID like “A10” passed format checks but caused index error  
✅ Fix: Added regex-based format validator and matrix boundary check  
✅ Result: Invalid inputs now return clear error messages without crashing

* ✅ Correct reference → Passenger info shown
* ❌ Incorrect reference → “Not found”

### 9. Booking Reference Generator

* Uses Python random and string libraries
* 8-character uppercase alphanumeric code
* Checks booking.db for duplication
* Regenerates if match found (ensures uniqueness)

### **10. Add function: Booking summary**

Reason to add:

One additional feature added to the system is a “seat booking summary”, which provides a real-time statistical overview of the number of:

• Reserved seats,

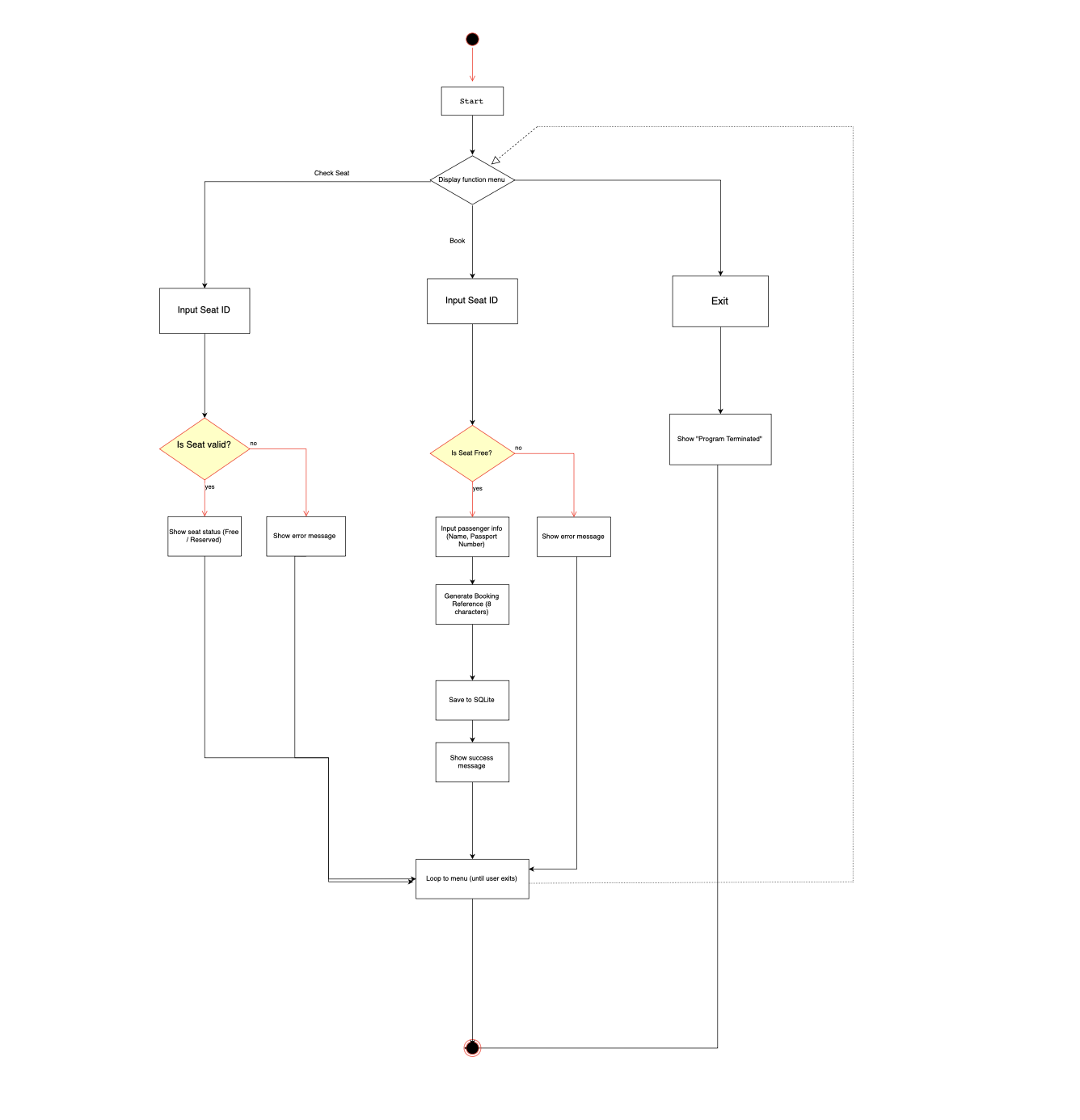
• Available (free) seats,

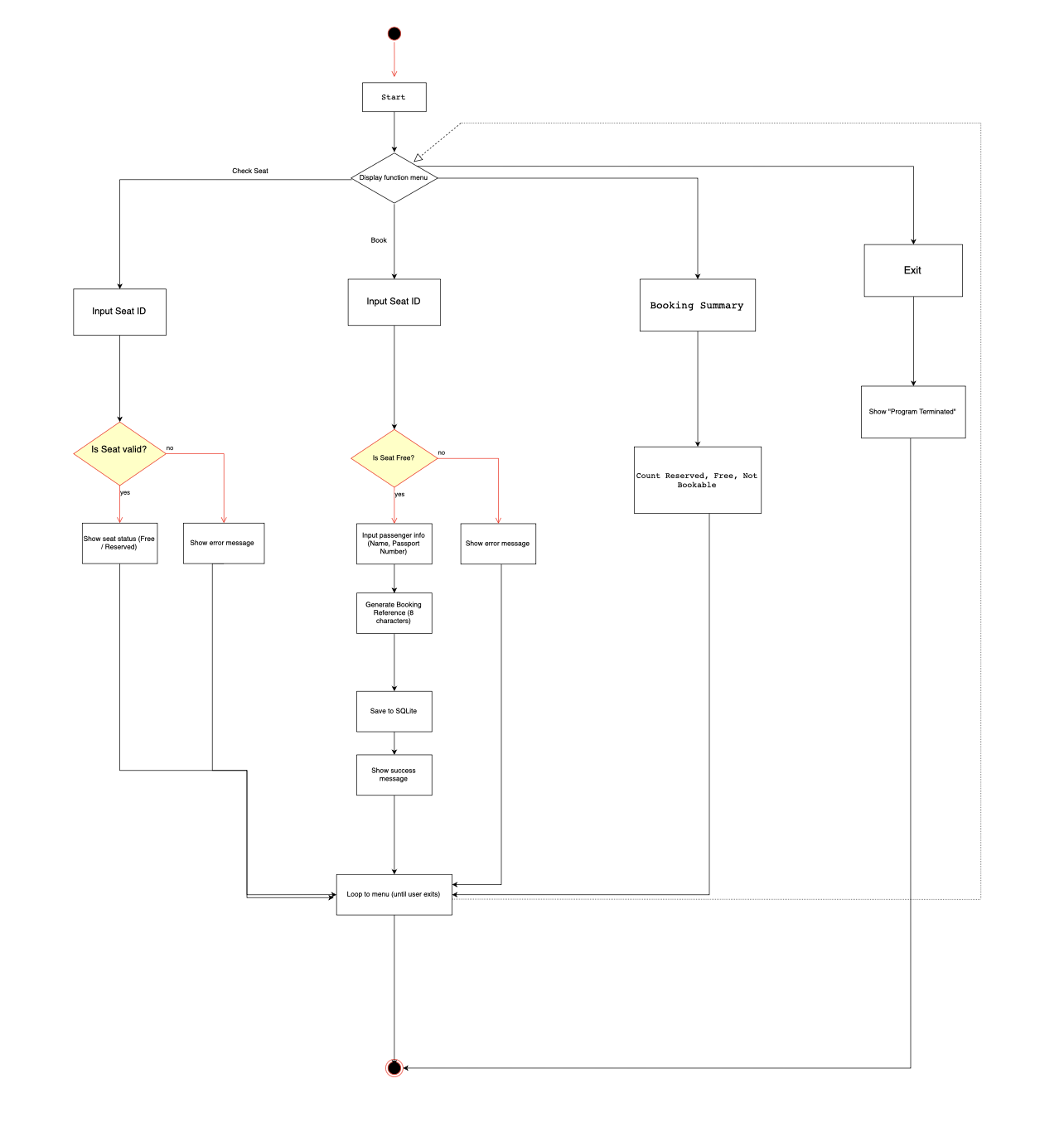
• Non-bookable seats (aisles and storage areas).

This functionality is common in real-world systems for operational monitoring and supports decision-making, such as knowing when a flight is nearly full or empty.

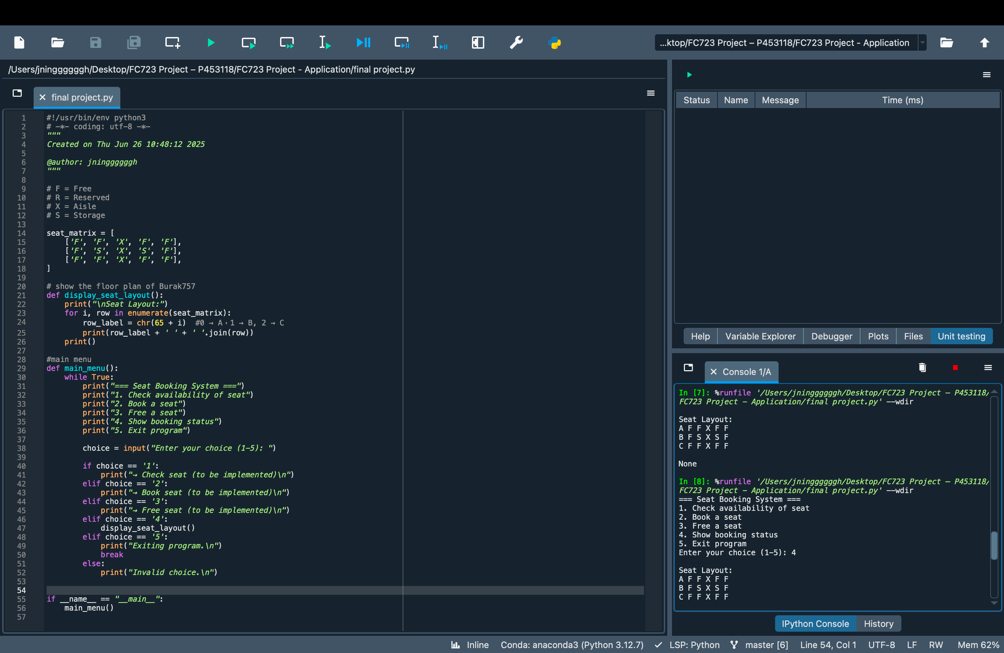
### Visual Evidence & Diagrams

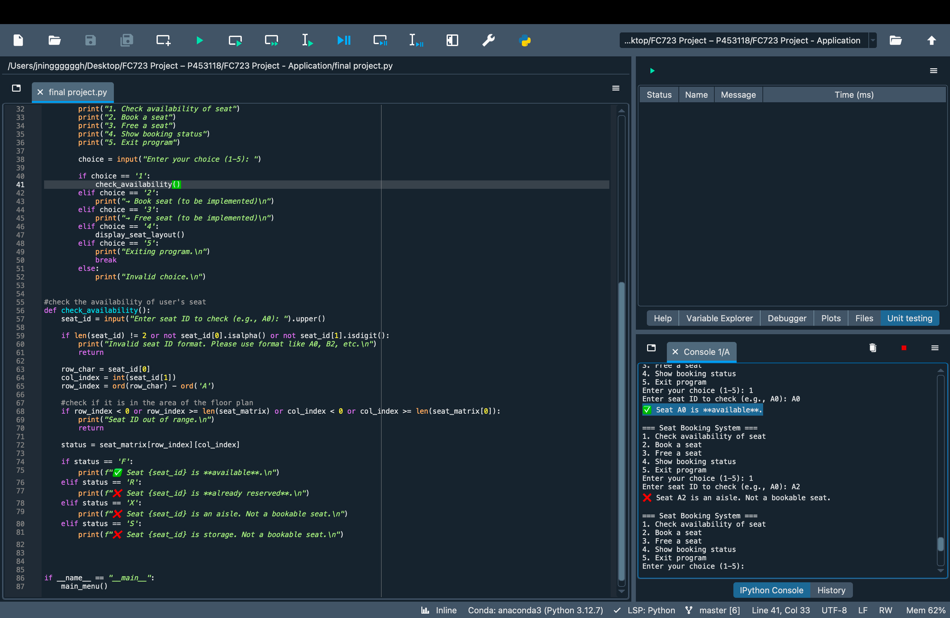
#### Comparison of Two Activity Diagrams

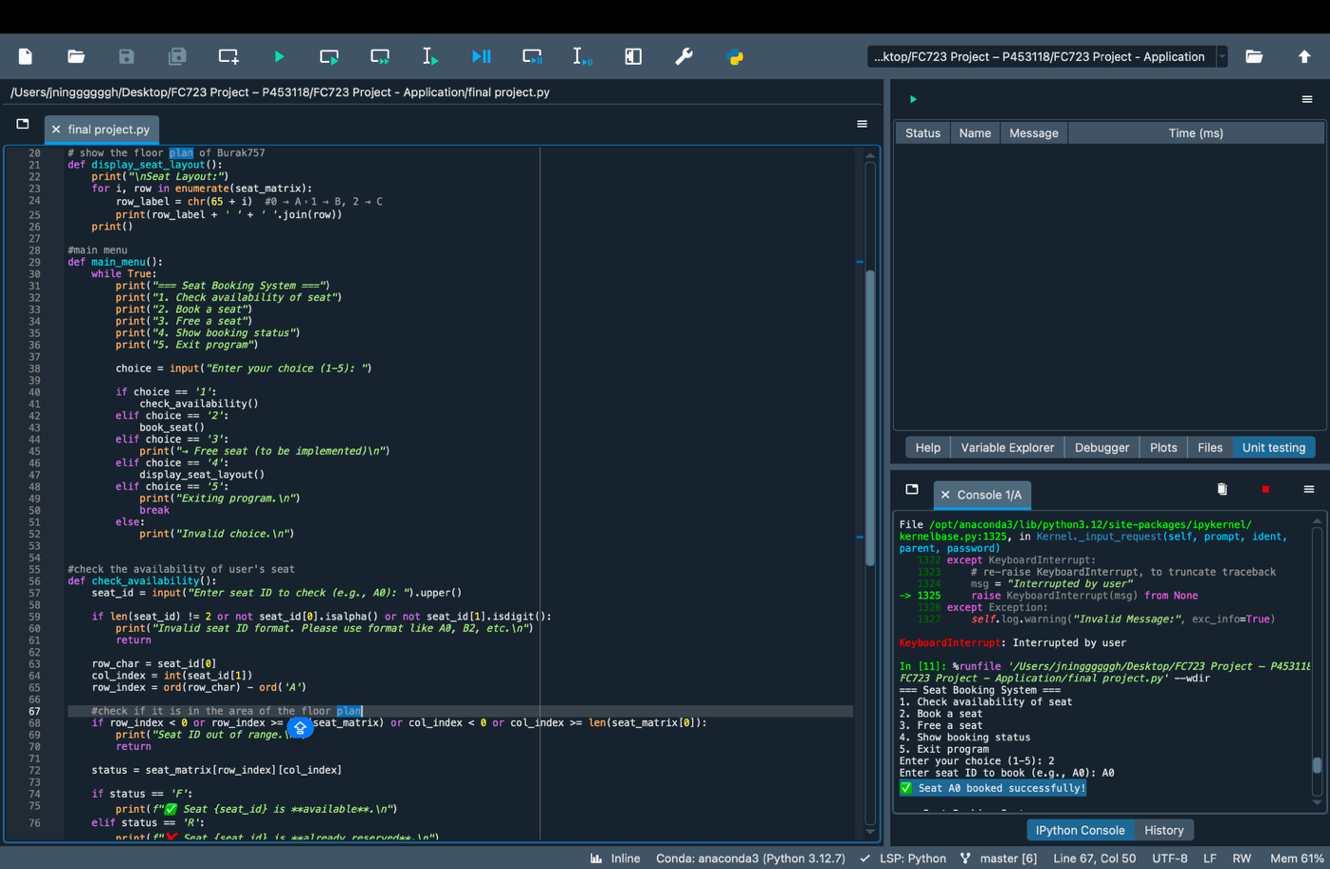
**Figure 1-1. Basic Activity Diagram (Original)**  


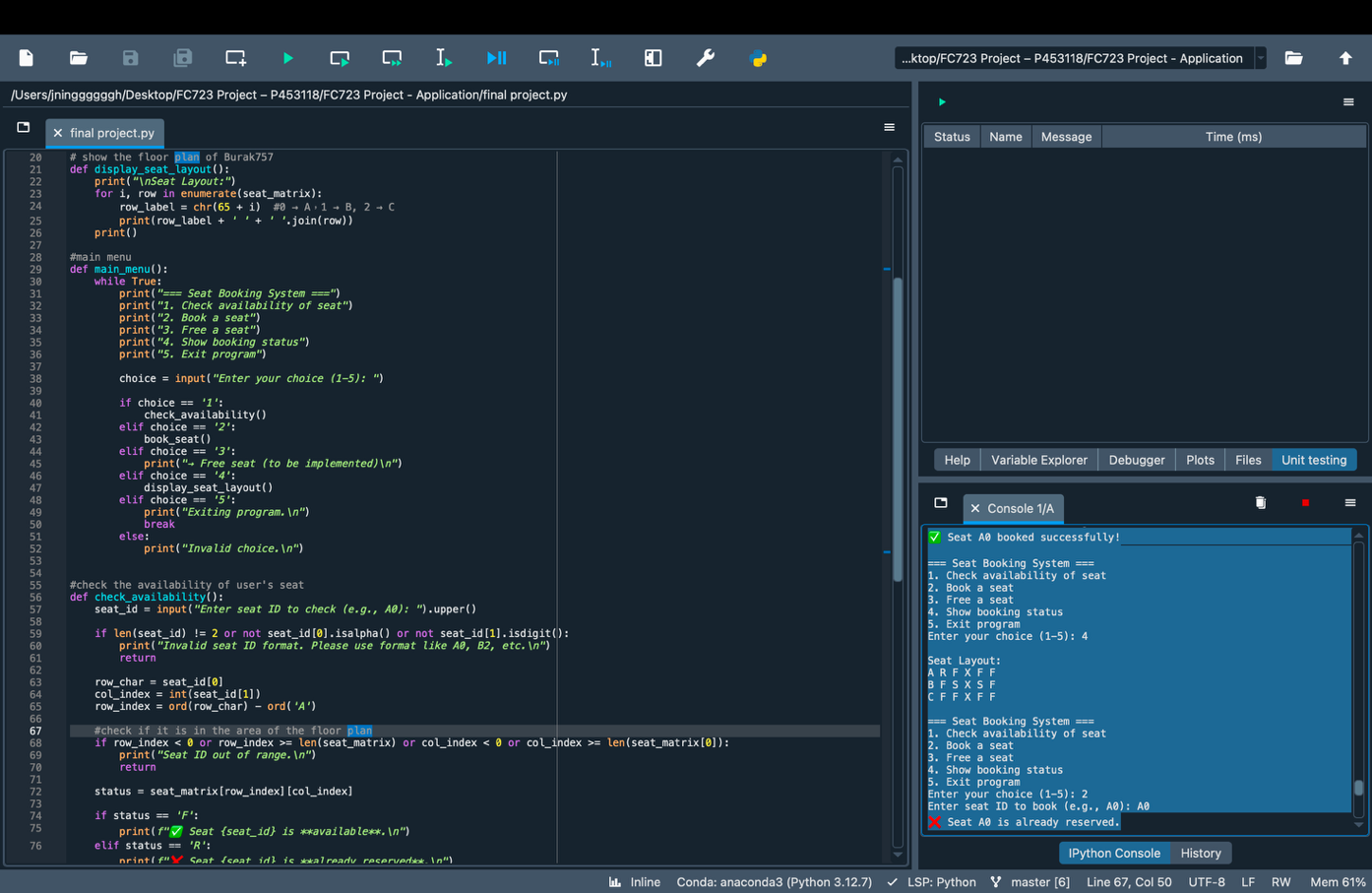
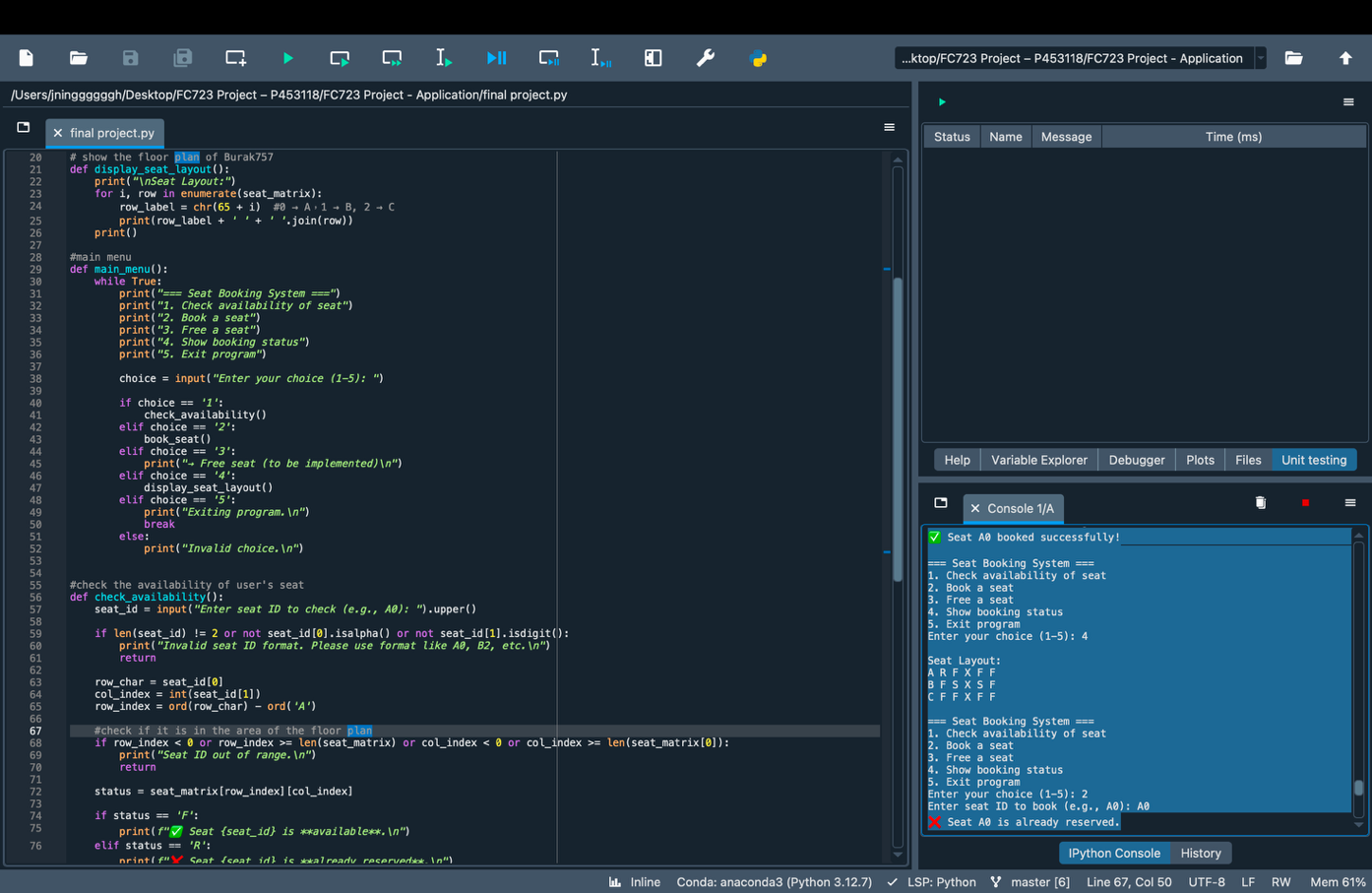
**Figure 1-2. Updated Activity Diagram (with Booking Summary Feature)**  


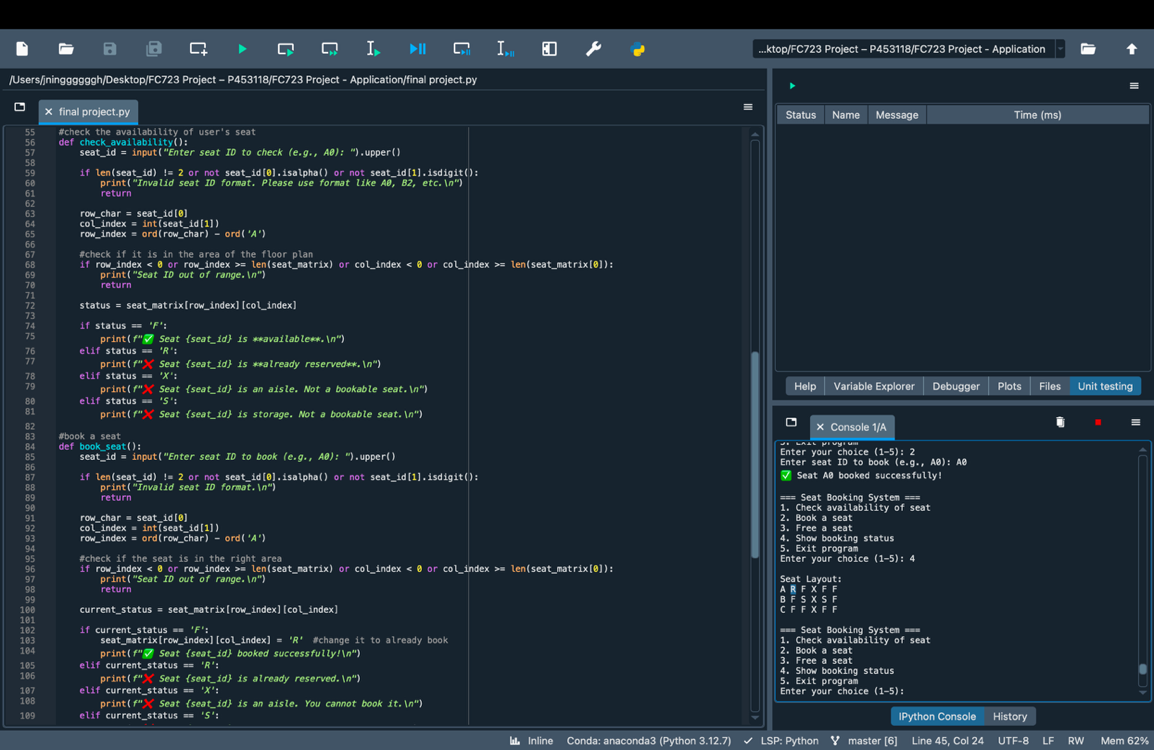
**Differences:** - The original diagram focused only on core functions: Check Seat, Book Seat, Exit. - The updated diagram adds a new branch for “Booking Summary” to reflect additional feature implementation. - This change represents Agile adaptability: new requirements (such as real-time statistics) were added mid-sprint and successfully integrated.

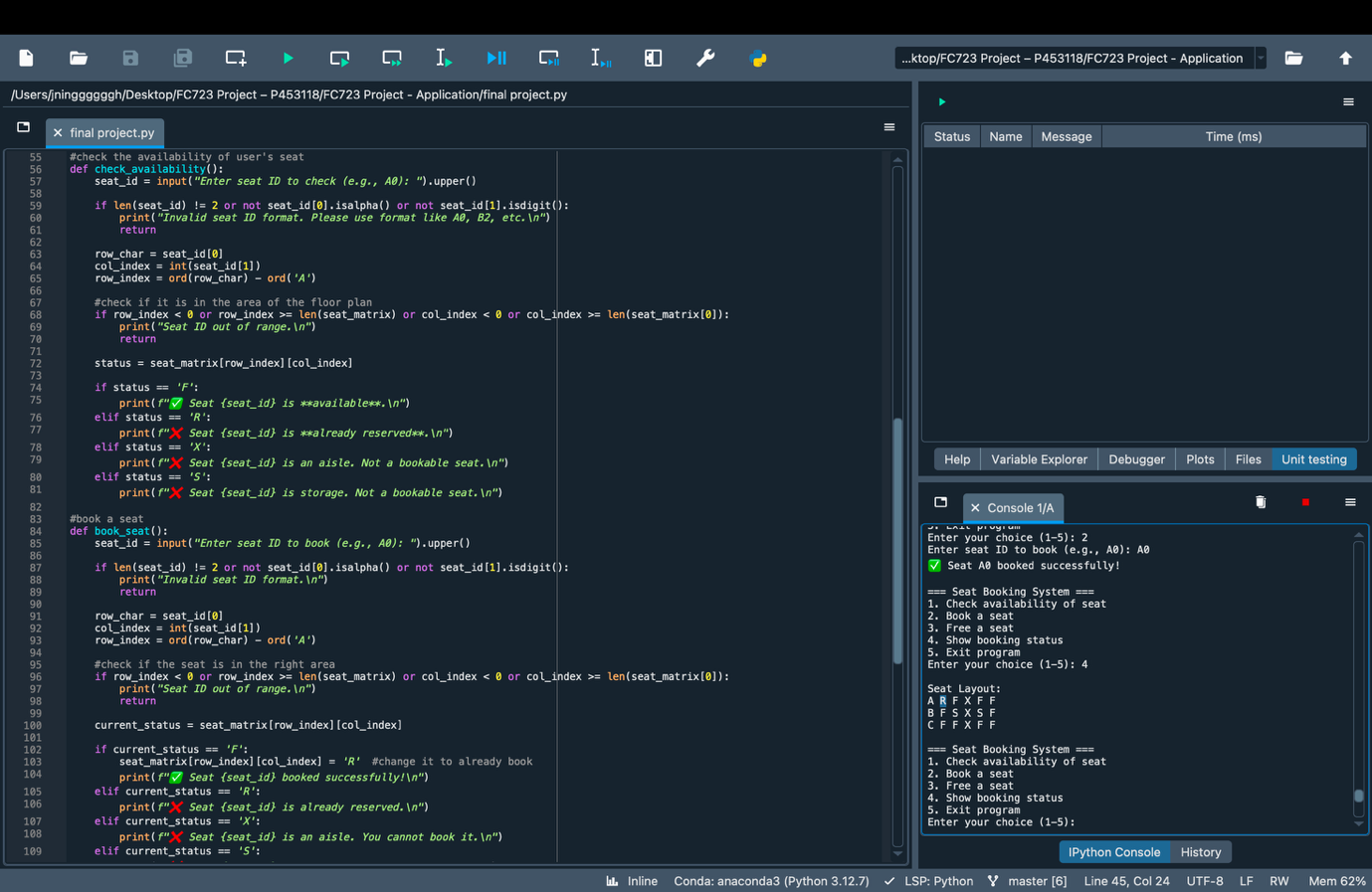
**Figure 2. Initial Seat Matrix Display**-Screenshot of the first version of the program in Spyder IDE, showing the menu and static seat matrix using “F”, “R”, “X”, and “S” codes. The seat layout is displayed using print\_seat() and only basic functionality is available at this stage.  


**Figure 3. Seat Availability Check + Error Message**-The program checks availability for various seats. Attempting to check “A1” returns availability, but checking an invalid seat like “Z9” results in a formatted error message, demonstrating input validation logic in action.  


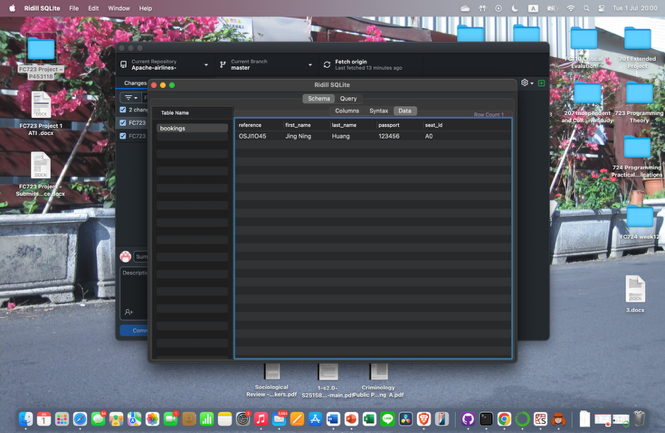
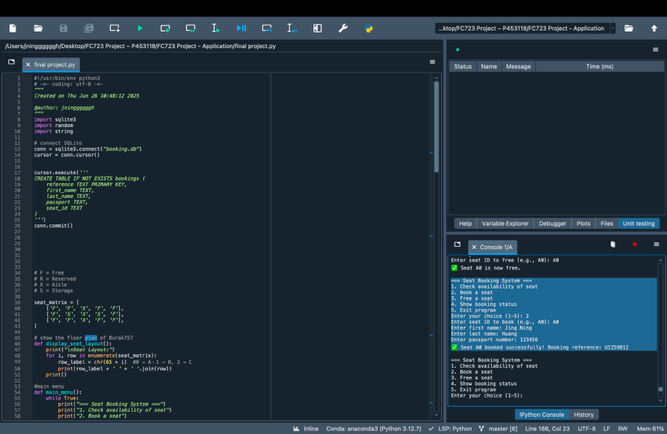
**Figure 4. Booking A Seat**-User selects option 2 from the menu and successfully books seat A0. A success message is displayed: “✅ Seat A0 booked successfully.”  


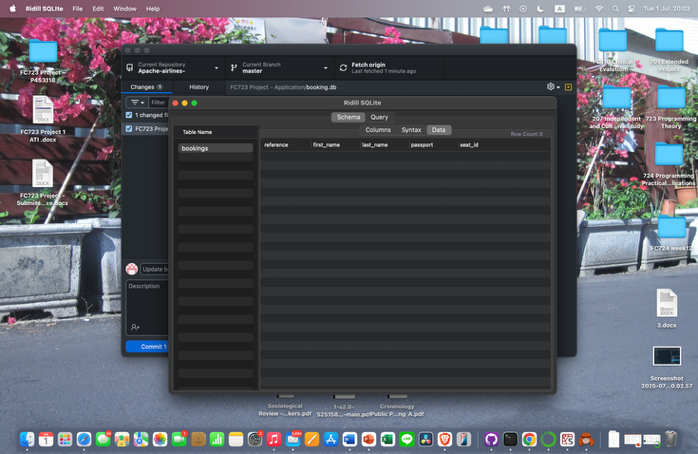
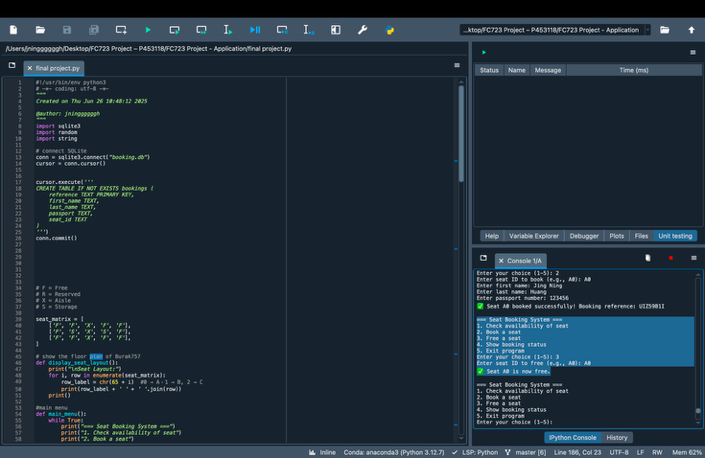
**Figure 5. Booking Layout Result**-Upon checking the seat layout again using option 4, seat A0 is shown as “R”, indicating that it is now reserved.  
**Figure 6. Duplicate Booking Attempt**- When the user attempts to book seat A0 again, the program correctly returns an error: “❌ Seat A0 is not available for booking.” Additional invalid inputs like “A2” (aisle seat) or “B1” (storage seat) also return proper validation messages.  


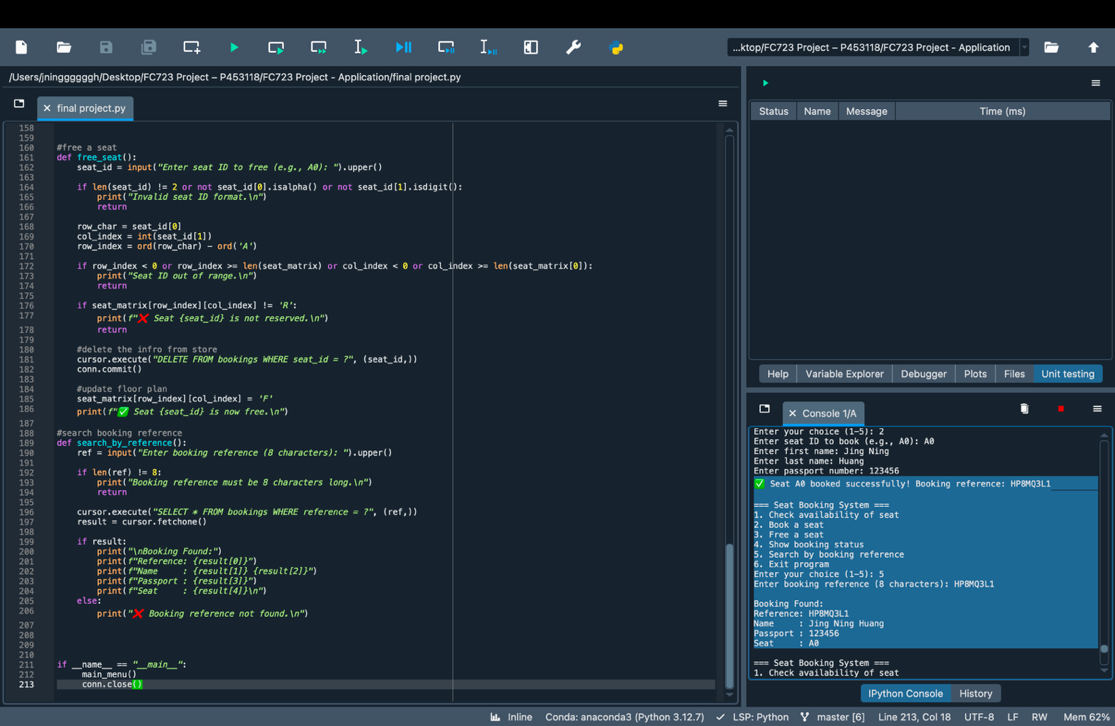
**Figure 7. Booking Cancelled (A0)**- This screenshot captures the successful cancellation of seat A0. After confirming the reservation earlier, the user selects option “3” and inputs “A0”. The system validates the seat is reserved and returns the message: “✅ Seat A0 is now free.” This demonstrates the seat cancellation feature and correct database update.  


The user selects option “4” again to display the seat layout after cancelling A0. The system shows A0 has reverted from “R” (Reserved) back to “F” (Free), confirming that the visual layout correctly reflects internal changes.

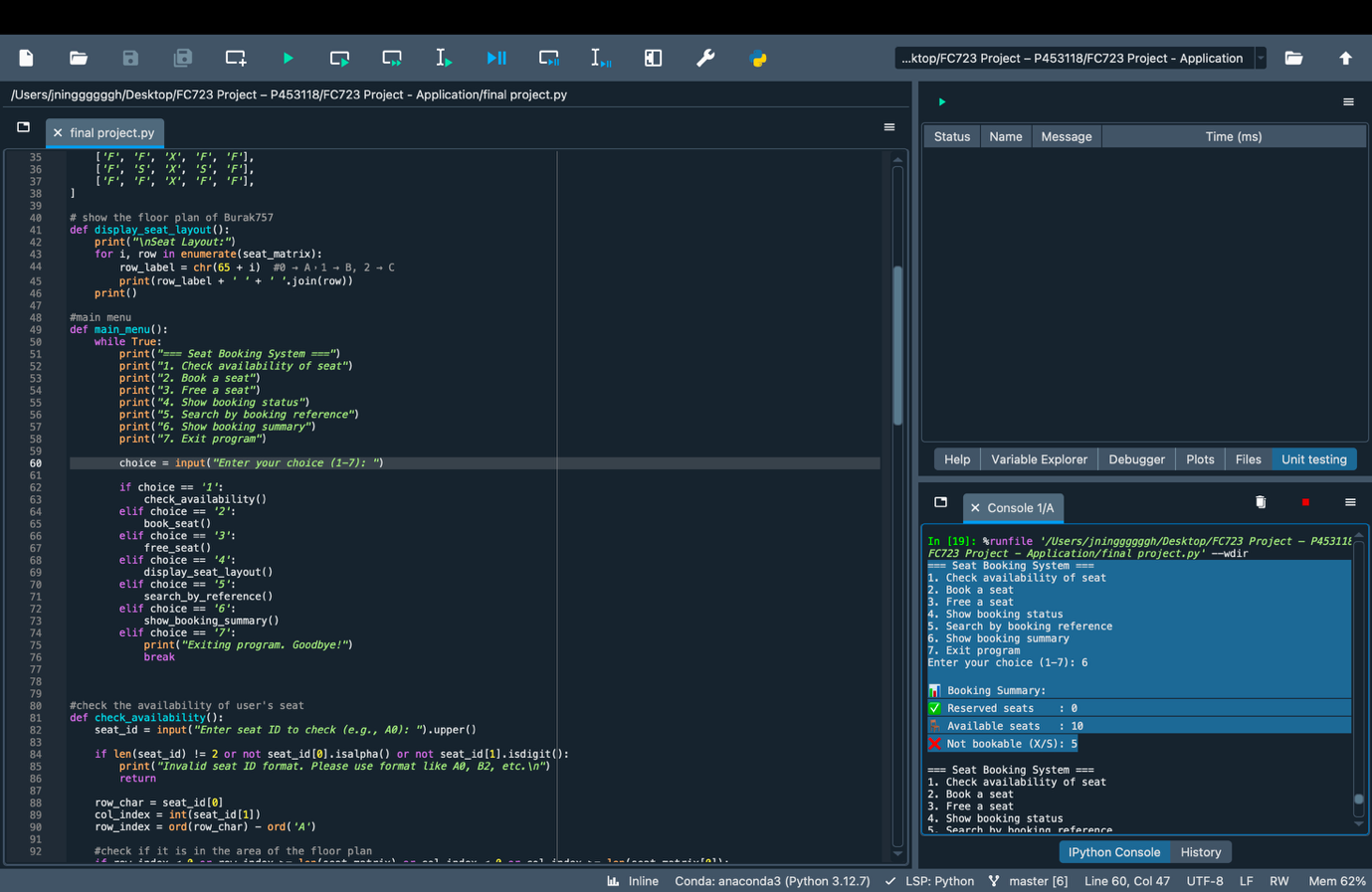
**Figure 8. Booking Saved to DB**-The user books a seat (A0) and provides passenger information such as name and passport number. The system successfully completes the reservation and displays a unique 8-character booking reference. This confirms that booking data is collected correctly and the reference is generated as expected. Using the DB Browser for SQLite, the user opens booking.db to verify the booking record. The database table correctly stores all passenger fields including reference, name, passport number, and seat ID. This confirms successful integration with persistent storage*.*



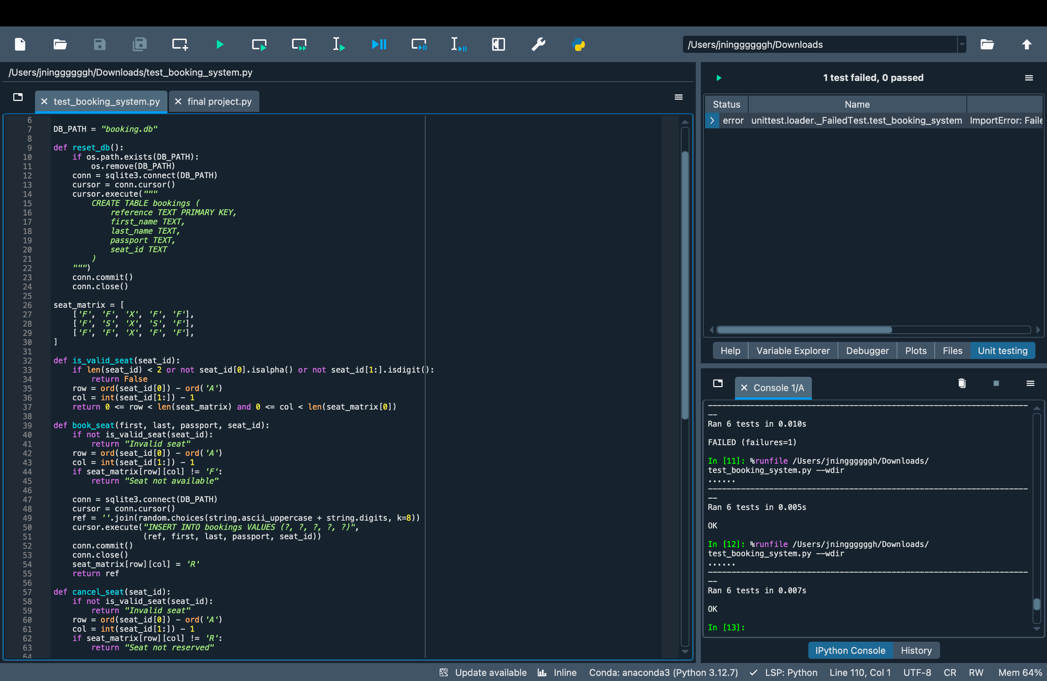
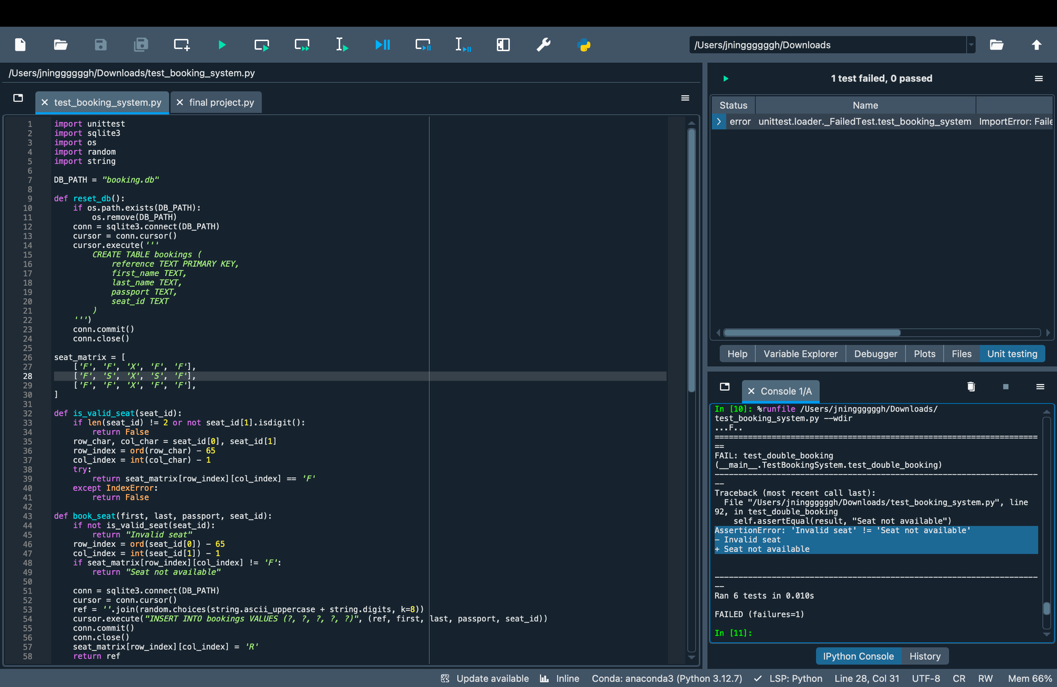
**Figure 9. Booking Entry Deleted from DB-**The user selects option “3” to cancel a previously booked seat (A0). The program confirms the cancellation and shows the success message “✅ Seat A0 is now free.” This proves that the system can correctly reverse a booking. The user opens the SQLite database again using DB Browser and confirms that the booking record for seat A0 has been removed. The “bookings” table is now empty, verifying that the cancellation function also updates the persistent storage correctly.  


**Figure 10. Search Booking Reference (Match**)- The user enters a valid 8-character booking reference (e.g. HP8M93L1) into the search function (option 5). The system locates the correct record in the database and displays all passenger details — including name, passport number, and seat ID. This confirms that the search function is working correctly for valid inputs.  


**Figure 11. Search Booking Reference (Not Found**)- The user then enters a fake or incorrect booking reference (e.g. 12345678). The system returns a message: “❌ Booking reference not found.” This confirms the program correctly handles invalid or unmatched search attempts and prevents leaking data.  


**Figure 12. Booking Summary Menu Output (ADD FUNTION)**  


**Figure 13. Unittest**- In the original unit test, assertions like assertEqual(result, "Seat not available") required the error message to match exactly. However, after improving the booking system’s validation logic, the program now returns more specific error messages such as "Invalid seat" or "Seat not available" depending on the case (e.g., format error vs. already booked). To accommodate this, the unit test was updated using assertIn(result, ["Seat not available", "Invalid seat"]). This allows the test to pass as long as the result is one of the expected outcomes, making it more flexible and better aligned with the enhanced seat validation logic.



**GitHub Repo Link**: <https://github.com/JingNing524/Apache-airlines->