Software Engineering Audit Report

Relazione.pdf

CAPRA

February 25, 2026

Contents

1 Document Context

Project Objective

The ITA Airways flight management system is designed to automate the daily allocation of aircraft and personnel to flight routes. The system assigns appropriate aircraft based on distance, passenger capacity, and airport compatibility, while ensuring crew assignments comply with aviation regulations regarding pilot certifications, minimum flight assistant requirements, and dual-crew mandates for long-haul flights.

Main Use Cases

- UC-1 – Consultazione voli: Allows consultants to view all available flights after authentication and authorization.
- UC-2 – Consultazione voli personali: Enables pilots to view their assigned flights and personal schedule.
- UC-3 – Autenticazione: Verifies user credentials (username and password) for system access.
- UC-4 – Autorizzazione: Grants appropriate access privileges based on user role (admin, pilot, or consultant).
- UC-5 – Visualizza dati velivolo: Displays detailed aircraft specifications and status information.
- UC-6 – Segnala partenza: Allows pilots to report aircraft departure from gate.
- UC-7 – Segnala arrivo: Allows pilots to report aircraft landing at destination airport.
- UC-8 – Gestione personale di volo: Enables personnel administrators to manage flight crew assignments.
- UC-9 – Gestione equipaggio di cabina: Manages flight assistant assignments and availability.
- UC-10 – Gestione piloti: Manages commander and first officer assignments and certifications.
- UC-11 – Gestione abilitazioni piloti: Handles pilot certifications for specific aircraft models.
- UC-12 – Gestione voli: Allows flight dispatchers to create, modify, and manage flight schedules.
- UC-13 – Gestione rotte: Enables flight dispatchers to manage available flight routes.
- UC-14 – Gestione velivoli: Allows flight dispatchers to manage aircraft inventory and instances.

Functional Requirements

- Aircraft must be assigned to flights based on seat capacity matching or exceeding booked passengers.
- Aircraft range must be sufficient to complete the route without refueling, with a 15% safety tolerance margin.
- Aircraft must be compatible with departure, stopover, and arrival airport dimension classes.
- Pilots must hold valid certifications for the assigned aircraft model; Airbus A318/A319/A320/A321 share a single certification.
- Flights exceeding nine hours require two pilots in the cockpit.
- Minimum flight assistants must be assigned based on aircraft type and ENAC regulations.
- System must track real-time aircraft location and status (in-flight or parked).
- Personnel must be assigned to flights respecting their role (commander, first officer, flight assistant) and certifications.
- System must support CRUD operations for aircraft, employees, routes, and flights.
- User authentication must support both admin and individual employee credentials.
- System must display employee data, aircraft details, route information, and flight schedules based on user role.
- Personal area must allow authenticated users to view their assigned flights and personal data.

Non-Functional Requirements

- System must maintain database consistency with referential integrity constraints on all foreign keys.
- Aircraft assignment algorithm must complete scheduling within acceptable time limits for operational use.
- Simulated clock must accurately track elapsed time in hours and minutes for flight monitoring.
- Password storage must use SHA-512 hashing for security.
- System must handle concurrent access to shared resources (aircraft location, personnel availability) using semaphore synchronization.
- Database connection failures must be handled gracefully with appropriate error notifications.
- System must support scalability for future integration with reservation management systems.
- Command-line interface must provide clear navigation and error handling for invalid user inputs.

Architecture

The system follows a layered architecture with clear separation of concerns. The **Data Access Layer** uses the DAO (Data Access Object) pattern with `PgDB` for PostgreSQL connectivity and specialized DAO classes (`FlightRouteDaoPg`, `EmployeeDaoPg`, etc.) implementing factory pattern for object instantiation. The **Domain Model Layer** includes core entities: `Aircraft`, `Airport`, `Employee`, `Flight`, and `FlightRoute`. The **Business Logic Layer** implements the Strategy pattern via `SchedulingStrategy` interface with `SimpleSchedule` concrete implementation using BFS graph traversal for aircraft allocation. The `SimulatedClock` class extends `Thread` with semaphore-based synchronization for time management. The **Presentation Layer** uses `CLI` (Command Line Interface) for user interaction. `ManagementSystem` serves as the central orchestrator coordinating all components. The system uses an `AirportGraph` data structure to model flight routes and optimize scheduling decisions.

Testing Strategy

The document does not provide explicit details on testing methodology. However, the architecture supports unit testing through the use of DAO interfaces in a separate sub-package, enabling mock implementations for isolated testing of business logic components. The `CredentialsManager` can be tested independently for authentication logic. The `SimpleSchedule` algorithm can be validated by verifying that all scheduling constraints are satisfied: aircraft capacity, range, airport compatibility, pilot certifications, and crew requirements. Integration testing would verify correct data flow between `CLI`, `ManagementSystem`, and database layers. The simulated clock can be tested for accurate time progression and synchronization across concurrent operations.

2 Executive Summary

Quick Overview

Total issues: **10** — HIGH: **3** MEDIUM: **6** LOW: **1**

Average confidence: **97%**

Executive Summary: Software Engineering Document Audit

This audit evaluates a university student's Software Engineering document describing a flight scheduling and crew management system. The document includes requirements analysis, domain modeling, design specifications, and a CLI-based implementation. The purpose is to assess completeness, consistency, and traceability between stated business rules, functional requirements, and implemented code.

The audit identified **10 issues across two categories**: 3 HIGH severity, 6 MEDIUM severity, and 1 LOW severity. The predominant pattern is a **critical gap between business rules and formal requirements**. Key business constraints—such as minimum crew sizes, double-crew requirements for long flights, passenger capacity validation, and pilot habilitation rules—are implemented in code but lack explicit, testable functional requirements with clear preconditions, postconditions, and error handling flows. Additionally, use case descriptions are limited to high-level diagrams without textual templates (main flow, alternative flows, error scenarios), making it impossible to verify how safety-critical rules are enforced from the user perspective.

The **three HIGH-severity issues** pose the greatest risk to document quality and system correctness. First, crew and flight duration constraints are coded but not formally specified as requirements (ISS-001). Second, a core business rule—aircraft seat capacity must exceed booked passengers—is violated in the implementation, where passenger numbers are ignored entirely (ISS-002). Third, the requirements section lacks detailed use case templates, preventing verification of how business rules are enforced (ISS-003). These gaps undermine traceability and create ambiguity about system behavior under edge cases.

Secondary concerns include incomplete mapping between use case diagrams and CLI implementation (ISS-006), missing business rules for personnel management operations (ISS-007), and absence of test coverage and traceability matrices (TST-001, TST-002). While the domain model and code structure are reasonably sound, the document fails to establish a clear chain of evidence from business needs to verified implementation.

Overall Assessment: The document demonstrates technical competence in design and coding but falls short of professional Software Engineering standards due to weak requirements specification and missing verification artifacts. Quality is **below acceptable** for a formal SWE deliverable.

Priority Actions:

1. **Formalize all business rules as explicit functional requirements** with preconditions, postconditions, and error handling. Specifically: (a) define crew minimums and double-crew rules with failure modes; (b) add passenger capacity as a hard constraint in scheduling; (c) document pilot habilitation decay and conflict resolution. Link each requirement to corresponding code and test cases.

2. **Expand use case descriptions** from diagrams to detailed textual templates (main flow, preconditions, postconditions, alternative/error flows) for all actors and use cases. Ensure each template explicitly references the business rules it enforces and maps to CLI menu items or API methods.

3. **Create a traceability matrix and test plan** linking each functional requirement and use case to specific test cases (manual or automated). Demonstrate test coverage for all business rules, edge cases (e.g., crew unavailability, incompatible airports), and database integrity constraints. Include evidence of test execution.

3 Strengths

- **Comprehensive Database Schema Design** — The document presents a well-structured Entity-Relationship model with clearly defined entities (Aircraft, Flight, Route, Airport, Personal), relationships with proper cardinality constraints, and explicit integrity rules. The logical schema is properly normalized with appropriate use of foreign keys and referential integrity constraints.

- **Detailed Use Case Analysis** — Three distinct use case diagrams are provided covering different user roles (Consultatore, Pilota, AmministratorePersonale, FlightDispatcher) with clear relationships using include/extend stereotypes, demonstrating comprehensive coverage of system functionality including authentication, flight management, and personnel administration.

- **Well-Organized Class Architecture** — The design implements established patterns including DAO (Data Access Object) for database abstraction, Factory pattern for object instantiation, and clear separation of concerns across CLI, database connectivity (PgDB), and domain model layers. Interface-based design in the DAO package facilitates maintainability and unit testing.

- **Robust Database Access Implementation** — The PgDB class demonstrates proper JDBC connection management with error handling, prepared statements to prevent SQL injection, and a comprehensive set of predefined queries. The runAndFetch() method properly handles ResultSet processing with metadata extraction.

- **Complete CRUD Operations** — The system implements full Create, Read, Update, Delete functionality for all major entities (aircraft, employees, routes, flights) through the systemCrud() method, enabling dynamic management of company data throughout the system's lifecycle.
- **Detailed Domain Model Documentation** — The Aircraft, Flight, and Route classes are thoroughly documented with clear explanations of attributes (plate, manufacturer, model, range, assistantsNumber) and their real-world significance in aviation operations.

4 Expected Feature Coverage

Of 7 expected features: 2 present, 4 partial, 1 absent. Average coverage: **55%**.

Feature	Status	Coverage	Evidence
Unit testing framework implementation	Absent	0% (0/5)	The document does not mention any unit tests, assertion-based verification, test classes, mocks, or coverage of unit/integration testing; it focuses on design, database, and business logic implementation.
Use of UML Diagrams for system modeling	Present	88% (7/8)	Multiple UML use case diagrams are described (actors like Consultatore, Pilota, FlightDispatcher with use cases such as Gestione voli, Consultazione voli), and a UML class diagram is detailed for entities like Flight, Route, Airport, Aircraft. Relationships between actors and use cases are explicitly listed, and diagrams clarify functional requirements. UI mockups are present but are not explicitly labeled as UML structure diagrams; navigation flows are shown via CLI screenshots and menu descriptions.
Identification and definition of system actors	Present	88% (7/8)	Actors such as Consultatore, Pilota, Comandante primo ufficiale, AmministratorePersonale, and FlightDispatcher are clearly identified in the use case diagrams. Their interactions with system functionalities (e.g., Gestione personale di volo, Gestione voli, Consultazione voli personali, Segnala arrivo/partenza) are documented, and the CLI section further details actions available to admins and users, including different access rights and menus. Explicit textual responsibility descriptions per actor role are limited but implied through associated use cases and menus.

Feature	Status	Coverage	Evidence
Definition and Documentation of Use Cases	Partial	40% (2/5)	Use case diagrams define specific user interactions (e.g., Consultazione voli includes Autenticazione and Autorizzazione; Gestione personale di volo includes Gestione equipaggio di cabina and Gestione piloti), and relationships between use cases and subcases are shown via «include» and «extend». However, there are no detailed textual templates with user goals, preconditions, post-conditions, or alternative flows for each use case.
User interface and interaction design principles	Partial	43% (3/7)	The document includes CLI UI mockups and screenshots (main menu, employees data menu, aircraft/route/flight detail views) and describes clear navigation via numbered menus and 'back' options. It also documents some validation mechanisms (e.g., login checks, InputMismatchException handling, ID validity checks). There is no step-by-step reservation creation process, no explicit structured input forms beyond CLI prompts, and no description of dynamic UI updates of user selections.
Separation of concerns in software architecture	Partial	60% (3/5)	The architecture separates concerns into CLI (view/controller-like), Domain Model (Aircraft, Airport, Employee, FlightRoute, Flight), Business logic (FlightManager, SimpleSchedule, FlightSchedule, ManagementSystem), and DAO (PgDB, *DaoPg, interfaces). Interactions between business logic and DAOs are documented (e.g., FlightManager uses EmployeeDaoPg; SimpleSchedule uses FlightRouteDaoI, ParkingDaoI, AirportDao). However, there is no explicit package named 'Controller' or 'Service', and interactions between a formal Controller layer and Service/DAO layers are not described.

Feature	Status	Coverage	Evidence
Data Access Object (DAO) pattern	Partial	71% (5/7)	The document explicitly describes the DAO pattern, with a PgDB class encapsulating DB access and *DaoPg classes implementing interfaces like $Type$DaoI. SQL queries are encapsulated in ConstantQueries and used inside DAOs (e.g., getCompanyEmployees, getFlights, deleteAircraftModel). Separate DAOs exist for different entities (e.g., EmployeeDaoPg, FlightRouteDaoPg, ParkingDaoI). DAOs abstract database access for business logic (e.g., FlightManager, SimpleSchedule). CRUD operations are mentioned conceptually in the CLI systemCrud section but not clearly defined as methods in DAO interfaces, and no testing strategies for DAO methods are described.

5 Summary Table

Category	HIGH	MEDIUM	LOW	Total
Requirements	3	4	1	8
Testing	0	2	0	2
Total	**3**	**6**	**1**	**10**

6 Issue Details

6.1 Requirements (8 issues)

ISS-001 — HIGH [100%] — Page 1 The business rules about minimum number of assistants per aircraft and double crew for flights longer than nine hours are implemented in code (assistantsNumber, isLongFlight, getCommanders/getFirstOfficers) but are not captured as explicit, testable functional requirements or use case conditions. There is no requirement stating what should happen if these constraints cannot be satisfied (e.g., flight not scheduled, error to dispatcher).

> *A ciascun volo viene assegnato un aeromobile, in base al tipo del quale l'ENAC (Ente Nazionale Aviazione Civile), oltre a comandante e primo ufficiale, prevede un numero minimo di assistenti di volo per poter autorizzare il decollo. Altra norma dell' aviazione è che, per un volo superiore alle nove ore, ci siano due equipaggi presenti in cabina di pilotaggio.*

Recommendation: In the requirements section, add explicit functional requirements for crew constraints, for example: (1) "For each flight, the number of assigned flight assistants must be >= aircraft.assistantsNumber; if not, the flight must not be scheduled"; (2) "For flights with duration > 9 hours, at least 2 commanders and 2 first officers must be assigned; otherwise the flight is rejected". Then, in the use case "Gestione voli" (FlightDispatcher) and in any scheduling-related use case, add alternative flows describing: (a) detection of insufficient assistants or pilots; (b)

system response (e.g., do not create the flight, show an error, log the issue). Finally, ensure that isFlightValid(...) behavior is documented in the business logic section as enforcing these requirements, and that the DB schema (e.g., assistants_number) is referenced as the source of the minimums.

ISS-002 — HIGH [100%] — Page 1 The statement defines a key business rule that aircraft seats must be greater than or equal to the number of booked tickets, but the implemented scheduling logic and described algorithms ignore passenger numbers entirely (passengerNumber is always set to 0 and never used in constraints). This is a direct violation of a core requirement.

> *I velivoli, di varia tipologia, devono essere adatti alla tratta, con criteri quali: numero di biglietti prenotati (numero di posti maggiore o uguale alle prenotazioni), range dell'aereo (autonomia dichiarata dalla casa produttrice, che deve consentire l'esecuzione senza scali della tratta), classe degli aeroporti di arrivo, scalo e partenza che devono essere in grado di accogliere quella specifica categoria di aeromobili.*

Recommendation: Update the scheduling requirements and logic so that passenger numbers are actually considered when assigning aircraft. Concretely: (1) In the requirements section, add an explicit rule like "For each scheduled flight, the assigned aircraft.seats must be >= flight.passengerNumber"; (2) In the scheduling algorithm (e.g., in canFly(...) or in a dedicated validation method), add a check that rejects aircraft whose seats are insufficient for the passengers of that route/flight; (3) When creating Flight objects in the BFS scheduling, set passengerNumber to a realistic value (from DB or from a configurable parameter) instead of hard-coding 0, and document this behavior; (4) Reflect this rule in at least one use case (e.g., "Gestione voli") with an alternative flow describing what happens when no aircraft can satisfy the passenger capacity constraint.

ISS-003 — HIGH [100%] — Page 5 The requirements analysis only contains high-level use case diagrams without any textual templates (no main flow, preconditions, postconditions, or alternative/error flows). This makes it impossible to verify how the business rules (aircraft suitability, crew minimums, long-haul double crew, pilot habilitation decay, airport class compatibility, etc.) are supposed to be enforced from the user perspective.

> *Analisi dei requisiti Use cases e templates Use case e template 1*

Recommendation: For each use case in the three diagrams ("Consultazione voli", "Consultazione voli personali", "Autenticazione", "Autorizzazione", "Visualizza dati velivolo", "Segnala arrivo", "Segnala partenza", "Gestione personale di volo", "Gestione equipaggio di cabina", "Gestione piloti", "Gestione abilitazioni piloti", "Gestione voli", "Gestione rotte", "Gestione velivoli") add a textual template with at least: (1) Preconditions (e.g., user authenticated as admin/pilot, system connected to DB, clock running); (2) Main success scenario (step-by-step interaction between actor and system); (3) Postconditions (what data is created/updated/deleted, what state changes); (4) Alternative and error flows (invalid credentials, missing data, constraint violations such as aircraft not compatible with route, insufficient crew, invalid habilitation). This will make the link between business rules and user interactions explicit and checkable.

ISS-004 — MEDIUM [100%] — Page 1 The special rule for Airbus A320 family and the decay of previous habilitation when a pilot gets a new one are implemented in helper methods (isAbilitationValid) and in the data model, but there is no corresponding use case or requirement describing how "Gestione abilitazioni piloti" should enforce these rules (e.g., what happens to existing flights when a pilot's habilitation changes, or how conflicts are prevented).

> *L'abilitazione dei piloti è strettamente legata al modello di aereo. Eccezione risiede nella famiglia Airbus A320: un pilota abilitato per tale modello, può infatti mettersi ai comandi anche di A318, A319, A320 e A321, dovuto al fatto che sono aerei molto simili tra loro. Nel caso in cui un pilota si abiliti per un nuovo velivolo,è previsto il decadimento del brevetto precedente.*

Recommendation: Extend the "Gestione abilitazioni piloti" use case with a textual description that includes: (1) Preconditions (admin authenticated, pilot exists); (2) Main flow for assigning a new habilitation; (3) A postcondition stating that when a new habilitation is assigned, any previous habilitation is automatically invalidated; (4) An explicit rule for the Airbus A318/319/320/321 family (e.g., "If abilitation is any of A318, A319, A320, A321, the system considers the pilot valid for all four models"); (5) Alternative flows for conflicts, such as existing scheduled flights that would become invalid after habilitation change (decide whether to block the change, warn the admin, or reschedule). Align the implementation of isAbilitationValid(...) with these documented behaviors and reference it in the business logic section as the enforcement mechanism.

ISS-005 — MEDIUM [100%] — Page 2 Airport/aircraft compatibility via DimensionClass and IsCompatible() is described at domain level, but there is no explicit functional requirement or use case step stating that the system must reject scheduling a flight when the aircraft dimension class is incompatible with departure/arrival/stopover airports. This weakens traceability between the safety-critical rule in the statement and the implemented logic (aircraft.canGo, canFly).

> *DimensionClass Modella la dimensione di un aeroporto, rappresentata da un numero da 1 a 4 che indica la lunghezza maggiore tra tutte le piste di decollo/atterraggio (da 1, la minore possibile, inferiore a 800m, a 4, la maggiore, superiore a 1800m) e una lettera che si riferisce alle dimensioni degli aeromobile che l'aeroporto può ospitare (da A a F, in ordine crescente) boolean IsCompatible() Questo metodo esegue il confronto tra due oggetti di dimensionClass e stabilisce se quella corrente (tipicamente dell'aeromobile) sia compatibile con other (tipicamente quella dell'aeroporto). Se sia il numero che la lettera di this sono inferiori a quelli di other restituisce true, altrimenti false.*

Recommendation: Add a clear functional requirement such as: "The system must not assign an aircraft to a route if its DimensionClass is incompatible with any of the route's airports (departure, stopover, arrival)." Then, in the "Gestione rotte" and "Gestione voli" use cases, add steps and alternative flows describing: (1) When a dispatcher selects a route and aircraft, the system checks DimensionClass compatibility for all involved airports; (2) If any airport is incompatible, the system refuses the assignment and shows a meaningful error. In the business logic section, explicitly link aircraft.canGo(...) and canFly(...) to this requirement, and mention that they are used during BFS scheduling and any manual flight creation.

ISS-006 — MEDIUM [100%] — Page 3 The use case diagram for consultation and pilot actions introduces actors and use cases (Consultatore, Pilota, Comandante primo ufficiale, Segnala arrivo/partenza, etc.), but the rest of the document only describes CLI menus and methods (accessFlightSchedule, accessPersonalArea, Segnala arrivo/partenza not described) without mapping them back to these use cases. There is no explanation of how the simulated clock and real-time status updates (aircraft busy/position, employee position) are exposed to these actors.

> *Use case e template 1 [IMAGE 1]: - **Diagram Type:** Use Case Diagram - **Elements:** - Actors: "Consultatore," "Pilota," "Comandante primo ufficiale" - Use Cases: "Consultazione voli," "Consultazione voli personali," "Autenticazione," "Autorizzazione," "Visualizza dati velivolo," "Segnala arrivo," "Segnala partenza"*

Recommendation: Create a mapping table that links each use case in "Use case e template 1" to the corresponding CLI functions and business logic methods. For example:

"Consultazione voli" → CLI.accessFlightSchedule() + ManagementSystem.runScheduling() + FlightSchedule.makeSchedule(); "Segnala arrivo"/"Segnala partenza" → specific CLI commands and updates to Aircraft.busy/position and Employee.position. Then, for each of these use cases, add textual steps describing how the simulated time (SimulatedClock) and state changes are visible to the actor (e.g., how a pilot sees their current flight status, how a consultatore sees which flights are in progress). This will make the real-time monitoring requirement explicit and consistent with the domain model.

ISS-007 — MEDIUM [100%] — Page 4 The personnel management use case diagram defines high-level operations (Gestione personale di volo, Gestione equipaggio di cabina, Gestione piloti, Gestione abilitazioni piloti), but the only detailed behavior described later is the generic CRUD menu in CLI.systemCrud() and some DAO operations. There is no requirement-level description of business rules for creating/removing employees (e.g., preventing deletion of crew assigned to future flights, validating roles vs habilitation, constraints on airport assignments).

> *Use case e templates 2 [IMAGE 2]: 1. **Image Type:** Diagram (Use Case Diagram) 2. **Analysis:** - **Diagram Type:** Use Case Diagram - **Elements:** - Actor: AmministratorePersonale - Use Cases: - Gestione personale di volo - Gestione equipaggio di cabina - Gestione piloti - Gestione abilitazioni piloti*

Recommendation: For the "AmministratorePersonale" use cases, add textual templates that specify: (1) What data can be created/updated/deleted for each role (Commander, FirstOfficer, FlightAssistant); (2) Business constraints, such as "It is not allowed to delete an employee who is assigned to a future scheduled flight" or "A Commander/FirstOfficer must always have a valid habilitation"; (3) How habilitation changes interact with existing schedules; (4) Error flows when these constraints are violated. Then, align the CLI.systemCrud() description with these use cases, explicitly stating which menu options implement which use case and how they enforce the constraints.

ISS-008 — LOW [100%] — Page 6 The presence of a FIXME comment in the description of SchedulingStrategy indicates that the design and/or documentation of the scheduling strategy interface has changed but the requirements and design sections have not been updated accordingly. This can cause confusion about which scheduling algorithms are actually available and how they should be selected.

> *SchedulingStrategy //FIXME refactoring has been made*

Recommendation: Resolve the "//FIXME refactoring has been made" by updating both the design and requirements: (1) Clearly describe the current set of concrete strategies (e.g., only SimpleSchedule at the moment) and how they are chosen (hard-coded, configuration, future extension); (2) If multiple strategies are planned, add a non-functional or functional requirement explaining under which conditions a different strategy would be used (e.g., optimization for fuel vs crew usage); (3) Remove or replace the FIXME with a stable description so that the document reflects the actual implementation and intended extensibility.

6.2 Testing (2 issues)

TST-001 — MEDIUM [87%] — Page 1 The CLI implements many user-visible flows corresponding to use cases (consultation of employees, aircraft, routes, flight schedule, personal area, CRUD operations), including input validation and error messages. However, the document does not describe any tests (automated or manual) that verify these flows, nor is there a traceability matrix linking use cases to specific test cases. This makes it hard to demonstrate that all functional requirements are actually verified.

> *La classe CLI (Command Line Interface) è resposabile dell'interazione con l'utente, fornendo le possibilità di navigazione all'interno del sistema, per poter consultare le informazioni relative a voli e personale. ... void run() Predispone un menù iniziale che permette di accedere a diverse categorie di informazioni in base al numero da 1 a 7 che viene inserito: 1. Dati degli impiegati 2. Dati degli aeromobili 3. Dati delle rotte disponibili 4. Dati dei voli disponibili 5. Area del personale 6. Aggiungi/Rimuovi aerei/impiegati/rotte 7. Esci Il caso in cui non sia inserito uno di questi caratteri prevede il lancio dell'eccezione di tipo InputMismatchException , che sarà gestita chiedendo di digitare nuovamente un comando ricordando che deve essere un numero da 1 a 7. ... (1) void accessEmployeesData() Stampa le informazioni riguardanti tutti gli impiegati che lavorano in ITA Airways. ... Visualizza uno specifico impiegato, "View a specific employee" Dopo aver inserito il suo id, verifica che sia valido (nel caso in cui non lo fosse notifica l'errore e chiede all'utente di riprovare) e stampa poi i suoi dettagli, sempre contenuti in managementSystem , mediante il metodo getEmployeeById(requestedId) ... (6) void systemCrud() Questa sezione della CLI risulta essere fondamentale per garantire la longevità del nostro sistema; riguarda infatti l'implementazione dei CRUD (acronimo di "Create, Read, Update, Delete", le quali altro non sono che le quattro operazioni basilari per la gestione persistente dei dati) per aerei, personale e rotte.*

Recommendation: Introduce a lightweight traceability and test plan for the main CLI use cases: - Create a table mapping each use case (e.g., "Consultazione voli", "Consultazione voli personali", "Gestione personale di volo", "Gestione voli", "Gestione rotte", "Gestione velivoli") to the corresponding CLI methods (run, accessEmployeesData, accessAircraftDetails, accessRoutesDetails, accessFlightSchedule, accessPersonalArea, systemCrud, etc.). - For each mapping, define at least one positive test (valid input, expected output/menu) and one negative test (invalid menu option, invalid ID, missing permissions) and describe the expected behavior (e.g., re-prompt, error message, or denial of access). - If full automation is too heavy, document these as manual test cases with clear steps and expected results; this will still improve your grade by showing systematic verification of the implemented flows.

TST-002 — MEDIUM [84%] — Page 1 There is no explicit test coverage or traceability for database integrity constraints and DAO behavior. The logical schema defines referential integrity and business rules (e.g., flights must reference existing routes, aircraft instances must reference existing aircraft, crew must exist in Personal), and DAOs/queries implement these, but the document does not show tests that verify that invalid data are rejected or that DAOs correctly map rows to domain objects.

> *Regole di vincolo Aircraft_instance: L'istanza deve far riferimento ad un modello di aereo esistente. Route: L'aeroporto di partenza di una rotta deve esistere tra gli aeroporti del database L'aeroporto di scalo (se diverso da null) deve esistere tra gli aeroporti del database L'aeroporto di partenza di una rotta deve esistere tra gli aeroporti del database Flight: Il comandante/i di un volo deve esistere tra il personale della Compagnia Il/I primo ufficiale/i di un volo deve esistere tra il personale della Compagnia Gli assistenti di volo devono esistere tra il personale della compagnia Un volo deve riguardare una delle rotte offerte dalla compagnia L'aeromobile che opera il volo deve comparire tra gli aerei in possesso. ... Metodo Vector<Type> getAll() Questo metodo stabilisce, usando l'oggetto db della classe PgDB, una connessione con il database. Facendo uso di una query costante estrae i dati richiesti e costruisce le necessarie strutture dati (un vettore di <Type>).*

Recommendation: Introduce a small set of integration tests (or clearly described manual tests) that exercise the DAOs against a test database and verify the integrity rules: - For each main table (Aircraft, Aircraft_instance, Airport, Route, Flight, Personal), prepare a minimal test dataset and write tests that call the corresponding *DaoPg.getAll() and assert that the number of returned objects and key fields match the database contents. - For Flight.commitFlight: attempt to insert a Flight that violates a constraint (e.g., non-existent route ID, non-existent aircraft_instance, or crew IDs not in Personal) and verify that the database rejects it (e.g., SQL

exception) and that your application either reports the error or does not leave inconsistent state. - Add a simple traceability table mapping each “Regole di vincolo” item to at least one DAO or commit method and to the test that verifies it. This will clearly show that your implementation respects the logical schema and constraints.

7 Priority Recommendations

The following actions are considered priority:

1. **ISS-001** (p. 1): In the requirements section, add explicit functional requirements for crew constraints, for example: (1) "For each flight, the number of assigned flig...
2. **ISS-002** (p. 1): Update the scheduling requirements and logic so that passenger numbers are actually considered when assigning aircraft.
3. **ISS-003** (p. 5): For each use case in the three diagrams ("Consultazione voli", "Consultazione voli personali", "Autenticazione", "Autorizzazione", "Visualizza dati ve...

8 Traceability Matrix

Of 21 traced use cases: 0 fully covered, 0 without design, 21 without test.

ID	Use Case	Design	Test	Gap
UC-1	Consultazione voli	✓	×	No test cases are documented for verifying flight consultation (listing, filtering, access control).
UC-2	Consultazione voli personali	✓	×	No tests are specified to ensure that a logged-in pilot/commander can only see their own flights.
UC-3	Autenticazione	✓	×	There are no documented tests for successful and failed authentication, password hashing, or credential validation.
UC-4	Autorizzazione	✓	×	No tests are described to verify authorization rules for admin vs normal users on each protected function.
UC-5	Visualizza dati velivolo	✓	×	There are no tests ensuring correct retrieval and display of aircraft data or enforcement of admin-only access.
UC-6	Segnala arrivo	✓	×	The design lacks an explicit UI/API operation for pilots to signal arrival and there are no tests for this behavior.
UC-7	Segnala partenza	✓	×	There is no clearly defined interface or tests for pilots to signal flight departure and update state accordingly.

ID	Use Case	Design	Test	Gap
UC-8	Gestione personale di volo	✓	×	No tests are provided for creating, updating, deleting, or listing flight personnel and their impact on scheduling.
UC-9	Gestione equipaggio di cabina	✓	×	There are no tests for managing cabin crew data or verifying correct assignment and relocation of flight assistants.
UC-10	Gestione piloti	✓	×	No tests are documented for pilot CRUD operations or for correct selection and assignment of pilots to flights.
UC-11	Gestione abilitazioni piloti	✓	×	There are no tests verifying that pilot abilitations are correctly enforced, including the Airbus family special rule.
UC-12	Gestione voli	✓	×	No tests are described for creating, scheduling, persisting, or listing flights, nor for validating crew and aircraft co...
UC-13	Gestione rotte	✓	×	There are no tests for CRUD operations on routes or for verifying that route changes are reflected in the scheduling gra...
UC-14	Gestione velivoli	✓	×	No tests are specified for aircraft CRUD, parking management, or enforcement of range and airport compatibility rules.
UC-15	Access employees data	✓	×	There are no tests for listing/filtering employees, viewing specific employee details, or enforcing login/role restricti...
UC-16	Access aircrafts details	✓	×	No tests are documented for accessing aircraft details or verifying that only admins can access this information.
UC-17	Access routes details	✓	×	Tests are missing for verifying correct retrieval and display of route details.
UC-18	Access flight schedule	✓	×	No tests verify that the flight schedule is correctly generated, persisted, and displayed for different user roles.
UC-19	Access personal area	✓	×	There are no tests for personal area access, including correct display of personal data and personal scheduling.

ID	Use Case	Design	Test	Gap
UC-20	System CRUD (Insert/Remove aerei/impiegati/rotte/aeroporti)	✓	×	No tests are defined for any CRUD operations or for enforcing admin-only access to the CRUD menu.
UC-21	Quit	✓	×	There are no tests for graceful shutdown, including stopping the simulated clock and closing database connections.

9 Terminological Consistency

Found **10** terminological inconsistencies (8 major, 2 minor).

Group	Variants found	Severity	Suggestion
Company staff / personnel	"Personal" (ER entity/table, SQL table "personal", association names, queries), "personale" (Italian text), "Employees" / "employees" (UI/menu, methods like getAllEmployees, getEmployeeById, class Employee), "impiegati" (Italian text), "Operatori della compagnia" (entity description), "personale di volo"	MAJOR	Use a single term for staff: prefer "Employee" / "employees" for the general concept and reserve "Employee" for the class name and "Personal" only for the DB table if strictly necessary, documenting the mapping.
Crew roles	"Commander" (UI, EmployeeRole enum, text), "First Officer" (UI, EmployeeRole enum, text), "Flight Assistant" / "Flight Assistant(s)" (UI, text), "comandante" / "comandanti" (Italian text), "primo ufficiale" / "primi ufficiali" (Italian text), "assistenti di volo" (Italian text), "hostess/steward" (DB value mapped in buildFromRow), "equipaggio di cabina"	MAJOR	Use one consistent role naming scheme; e.g. use "Commander", "First Officer", "Flight Assistant" everywhere in code, DB, UI, and documentation, and document any legacy DB values if they must differ.

Group	Variants found	Severity	Suggestion
Aircraft entities and naming	"Aircraft" (ER entity, class), "Aircraft_instance" (ER entity, table, sometimes written "Aircraft-instance"), "aeromobile" / "aeromobili" (Italian text), "velivolo" / "velivoli" (Italian text), "aerei" (Italian text), "aircrafts" (UI text: "Access aircrafts details")	MAJOR	Use a single term for the aircraft model vs physical aircraft distinction; e.g. keep "Aircraft" for the model and "Aircraft_instance" (or better "AircraftInstance") for the physical aircraft, and avoid using generic "aerei" / "velivoli" when the distinction matters.
Route vs flight route vs flight	"Route" (ER entity/table, SQL, attributes), "FlightRoute" (domain class, DAO, scheduling), "rotte" / "rotta" / "tratta" / "tratte" (Italian text), "Gestione rotte" (use case), "flight route" (menu item: "Insert/Remove flight route"), "operativo voli" (for schedule of flights)	MAJOR	Use one consistent term for routes; e.g. use "Route" / "FlightRoute" for the generic path and reserve "Flight" for the specific instance, documenting clearly that "Route" (DB) maps to "FlightRoute" (domain).
Airport and dimension class	"Airport" (ER entity, class, table), "aeroporto" / "aeroporti" (Italian text), "class" (airport attribute), "DimensionClass" (class name, sometimes written "dimensionClass" in text), "classe" (Italian text for airport class), "AirportWeighted" (wrapper using same concept)	MAJOR	Use a single term for airports and their classification; e.g. keep "Airport" and "DimensionClass" (or a consistently cased variant) and avoid mixing with generic Italian descriptions when referring to the same formal concept.
Scheduling vs schedule vs operativo voli	"scheduling" (class names SchedulingStrategy, SimpleSchedule, FlightSchedule, text), "schedule" (method makeSchedule, variable names), "schedulare" (Italian verb in text), "operativo voli" (Italian phrase for flight schedule), "flight schedule" (UI: "Access flight schedule")	MAJOR	Use one consistent term for the scheduling algorithm and process; e.g. use "scheduling" and "schedule" in English everywhere, or consistently use the Italian equivalent, but avoid mixing within the same context.

Group	Variants found	Severity	Suggestion
CRUD operations	"CRUD" (acronym), "CRUD (acronimo di \"Create, Read, Update, Delete\")" (Italian explanation), "CRUD (acronimo di \"Create, Read, Update, Delete\", le quali altro non sono che le quattro operazioni basilari...)" (long form), menu items "Insert/Remove ..." (English verbs for same operations), "Insert/Remove aircraft" / "Insert/Remove airport" / "Insert/Remove employee" / "Insert/Remove flight route"	MINOR	Standardize the acronym and its expansion; e.g. always use "CRUD" and expand it once as "Create, Read, Update, Delete" without translating or re-explaining differently.
Pilot qualification / abilitazione	"abilitation" (attribute in Personal, Employee, UI output), "Abilitation" (capitalized in UI), "abilitazione" (Italian text), "brevetto" (Italian text for license), "passaggio macchina" (Italian text for type rating change)	MAJOR	Use a single, consistently spelled term for pilot qualification; e.g. use "abilitation" or better the correct English "qualification" everywhere in code, DB, and UI, and align the Italian narrative to that term.
Simulated time / clock	"SimulatedClock" (class name), "orologio simulato" (Italian text), "orologio" (generic Italian), "tempo di simulazione" (Italian text), "clock" (field name in ManagementSystem)	MINOR	Use one consistent term for the simulated time component; e.g. always refer to it as "SimulatedClock" and "tempo di simulazione" or the English equivalent, and avoid mixing with generic "orologio" / "orologio simulato" when referring to the class.
Authentication / credentials / login	"Credentials" (domain class), "CredentialsManager" (class), "login" (Italian text), "Autenticazione" (use case), "Autorizzazione" (use case), "username" / "password" (UI and DB attributes)	MAJOR	Use a single term for user authentication and authorization concepts; e.g. keep "Credentials" / "CredentialsManager" and consistently refer to "login" and "autenticazione" (or their English equivalents) without mixing multiple labels for the same step.