

PROGETTAZIONE E SVILUPPO DI UNA ARCHITETTURA JAVA PER LA PRENOTAZIONE DI VOLI AEREI

Software Architectures and Methodologies

Settembre, 2017

Alessio Sortino, Fabio Vittorini



UNIVERSITÀ
DEGLI STUDI
FIRENZE

DINFO

DIPARTIMENTO DI
INGEGNERIA
DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica

Indice

1	Introduzione	5
2	Presentazione e Analisi del Problema	6
2.1	Analisi del Problema	6
2.1.1	Descrizione del Problema	6
2.2	Analisi dei Requisiti	6
2.2.1	Requisiti Funzionali	7
2.2.2	Requisiti non Funzionali	7
2.2.3	Vincoli	7
2.3	Progettazione	8
2.3.1	Modello Concettuale	8
2.3.2	Modello di Dominio	9
2.3.3	Use Case Diagram	11
2.3.4	Page Navigation Diagram	12
2.3.5	Mockups	13
2.3.6	Architettura 3-tier	17
3	Implementazione	18
3.1	Domain Logic	18
3.1.1	Strategy	19
3.1.2	Util	20
3.1.3	Temporary Reservation	21
3.2	Data Access Objects (DAO)	22
3.3	Business Logic	23
3.3.1	Bean	23
3.3.2	@HttpParam	24
3.3.3	Controller	24
3.4	Presentation Pages	25
3.4.1	JSF	26
3.4.2	CSS	26
3.4.3	JavaScript	27
3.4.4	AJAX	27
4	Test	28
4.1	Domain Logic	28
4.1.1	Strategy	28
4.2	DAO	28
4.3	Business Logic	29
4.4	Util	29
4.5	Valutazioni Quantitative	29

5	Funzionalità e Utilizzo	31
5.1	Flussi di Utilizzo	31
5.2	Panoramica dell'Applicazione	31
5.2.1	Ricerca dei Voli e Nuova Prenotazione	31
5.2.2	Gestione della Prenotazione	35
5.2.3	Area di Amministrazione	37
6	Conclusioni e sviluppi futuri	40
A	Controllers	42
A.1	SearchFlightsController	42
A.2	InsertReservationController	42
A.3	ReservationLoginController	43
A.4	ManageReservationController	44
A.5	AdministratorLoginController	44
A.6	InsertFlightController	44
B	Funzioni JavaScript	46
	Bibliografia	48

Elenco delle figure

2.1	Modello concettuale.	9
2.2	Diagramma delle classi riferite al modello di dominio.	10
2.3	Use Case Diagram.	11
2.4	Page Navigation Diagram.	13
2.5	Home page.	14
2.6	Risultati della ricerca dei voli.	14
2.7	Conferma della prenotazione.	14
2.8	Riepilogo della prenotazione.	15
2.9	Accesso all'area riservata per la visualizzazione della prenotazione.	15
2.10	Pagina di login lato amministratore.	15
2.11	Inserimento di un volo (lato amministratore).	16
2.12	Visualizzazione dei voli (lato amministratore).	16
2.13	Relazioni tra Business Logic, Domain Model e DAOs.	17
3.1	Pattern strategy per il calcolo dello sconto da applicare alla prenotazione.	19
3.2	Vista di un utente nella ricerca dei voli con altre prenotazioni temporanee attive.	22
3.3	Organizzazione della cartella <i>\WebContent</i> per la gestione della grafica dell'applicazione.	26
3.4	Esempio di utilizzo di tag JSF.	26
3.5	Esempio di aggiornamento delle date attraverso AJAX.	27
4.1	Analisi con JaCoCo sulla copertura dei test.	30
5.1	Flight Manager - Logo dell'applicazione.	31
5.2	Pagina principale di ricerca voli.	32
5.3	Scelta della data.	32
5.4	Risultati di ricerca dei voli.	33
5.5	Sessione scaduta.	33
5.6	Inserimento indirizzo email.	34
5.7	Inserimento dati dei passeggeri.	34
5.8	Box del prezzo finale con dettaglio dello sconto.	34
5.9	Resoconto della prenotazione.	35
5.10	Pagina di login per la gestione prenotazione.	35
5.11	Profilo prenotazione.	36
5.12	Pagina di resoconto della prenotazione.	36
5.13	Conferma di avvenuta cancellazione.	37
5.14	Pagina di modifica dei passeggeri.	37
5.15	Pagina di login per la gestione amministrazione.	38
5.16	Profilo amministratore.	38
5.17	Pagina di inserimento nuovi voli.	39
5.18	Pagina di ricerca voli.	39

Capitolo 1

Introduzione

Quello della prenotazione di servizi per l'utente è un problema molto comune e importante che include numerosi applicativi per la gestione e l'organizzazione di dati sensibili, e che quindi deve essere strutturato con una logica ed una architettura robusta e sicura. In particolare la prenotazione di voli aerei è una realtà attuale molto importante che riguarda qualsiasi compagnia che metta a disposizione dei voli aerei. Per avere una vendita agevole, comoda e sicura che permetta a qualsiasi persona di prenotare il proprio biglietto ovunque si trovi, è indispensabile che ogni compagnia aerea metta a disposizione una propria applicazione accessibile online che le consenta di svolgere queste e molte altre funzionalità in modo semplice ed intuitivo. Gli utenti, ai quali sarà destinata un'applicazione simile, saranno molteplici: da una parte avremo gli amministratori di tale applicazione che, attraverso un'interfaccia molto intuitiva, si occuperanno di mantenerla aggiornata, ad esempio inserendo periodicamente i voli messi a disposizione dalla compagnia, cancellando dei voli oppure ancora rendendo disponibili le condizioni di viaggio per i passeggeri di un volo; dall'altra invece, abbiamo gli utenti che hanno intenzione di cercare voli ed effettuare una o più prenotazioni valutando i diversi prezzi e le condizioni di viaggio.

In un contesto come questo si è pensato di progettare e realizzare un'architettura software in grado di gestire le prenotazioni di voli aerei. L'applicazione consentirà quindi ad un utente qualsiasi che vi accede di cercare e selezionare i voli aerei di interesse valutando e prenotando uno o più posti a sedere. Una volta effettuata l'operazione, l'utente sarà quindi in grado di consultare la propria prenotazione in qualsiasi momento accedendo all'area riservata; dopo aver effettuato l'accesso sarà anche in grado di cancellarla o di modificare i dati di uno o più passeggeri.

L'applicazione consentirà inoltre agli amministratori di inserire nuovi aeroporti e nuovi voli, oltre che a cancellare voli che non presentano prenotazioni attive.

L'applicazione comprende una parte di *backend* realizzata in Java attraverso lo stack tecnologico J2EE ed una parte di *frontend* realizzata attraverso la tecnologia JSF.

Capitolo 2

Presentazione e Analisi del Problema

2.1 Analisi del Problema

L'obiettivo di questo elaborato è quello di progettare e realizzare un'architettura software in grado di gestire prenotazioni di voli aerei. La piattaforma sarà accessibile da qualsiasi utente per effettuare una ricerca di voli e in seguito per la creazione e la gestione delle prenotazioni; un amministratore invece sarà in grado di creare nuove tratte e nuovi aeroporti nel sistema.

2.1.1 Descrizione del Problema

Un utente che accede alla piattaforma dovrà essere in grado di selezionare gli aeroporti di partenza e destinazione, specificando la data di andata ed eventualmente quella di ritorno, oltre al numero di passeggeri per i quali si intende effettuare la prenotazione. Una volta confermati i dati dovranno essere elencati tutti i voli disponibili per le date selezionate (ordinate per orario di partenza) dalle quali l'utente potrà scegliere le proprie preferenze valutando anche il prezzo per passeggero. Una volta selezionato il volo (o i voli se viene scelto anche il ritorno), sarà visualizzata una pagina di conferma della prenotazione dove saranno specificate le scelte dell'utente e il prezzo totale, valutando uno sconto complessivo in base a diverse scelte (date particolari, numero di passeggeri elevato etc..). Se la prenotazione verrà confermata, l'utente dovrà inserire un indirizzo e-mail, col quale sarà possibile accedere alla gestione della prenotazione, ed i dati di tutti i passeggeri quali nome, cognome, data di nascita e numero della carta d'identità. Al termine verrà visualizzata una pagina di riepilogo contenente i dettagli della stessa.

Accedendo ad un'area riservata attraverso l'indirizzo e-mail inserito ed il numero di prenotazione generato, l'utente dovrà essere in grado di visualizzare i dettagli della prenotazione, modificare i dati dei passeggeri o cancellare l'intera prenotazione.

L'utente amministratore dovrà essere in grado di accedere attraverso le proprie credenziali ad un'area riservata nella quale sarà possibile inserire nuovi voli e nuovi aeroporti. Un volo sarà caratterizzato da un numero di volo, dagli aeroporti di partenza e di destinazione, dall'orario di partenza, dalla durata del volo, dal prezzo per passeggero, dal numero di posti totali e dal numero di posti prenotati. L'amministratore sarà in grado di visualizzare i voli già presenti nel sistema ed inserire più voli per lo stesso giorno che coprono la stessa tratta, ma in orari differenti.

2.2 Analisi dei Requisiti

A partire dall'analisi del problema sono stati estratti i requisiti del sistema esaminando quelli esplicitati nel testo dal committente e ricavando quelli facenti parte dell'applicazione stessa. Sono stati distinti in *requisiti funzionali*, *requisiti non funzionali* e *vincoli*.

2.2.1 Requisiti Funzionali

Sono l'insieme dei requisiti che costituiscono la ragione stessa per il quale il sistema viene costruito.

L'applicazione deve consentire:

- *Ad un utente generico*
 - R1. di cercare i voli presenti nel sistema
 - R2. di selezionare un volo di andata ed eventualmente uno di ritorno dato l'elenco dei voli disponibili su una tratta scelta e riservare i posti inseriti
 - R3. di confermare una prenotazione una volta selezionati i voli desiderati, consultato il prezzo totale e inserito un indirizzo email insieme ai dati dei passeggeri
- *Ad un utente che possiede i dati di una prenotazione*
 - R4. di consultare i dettagli della stessa e stampare la conferma
 - R5. di modificare i dati dei passeggeri
 - R6. di cancellare l'intera prenotazione
- *Ad un amministratore*
 - R7. di creare un nuovo volo
 - R8. di creare un nuovo aeroporto
 - R9. di consultare i dettagli dei voli presenti nel sistema comprensivi del numero di posti prenotati
 - R10. di cancellare i voli aventi nessuna prenotazione attiva

2.2.2 Requisiti non Funzionali

Costituiscono l'insieme delle proprietà che deve possedere il sistema per gestire e organizzare in modo corretto l'applicazione.

L'applicazione deve:

- R11. gestire la concorrenza delle prenotazioni simultanee garantendo la consistenza sul numero di posti disponibili presenti al momento della conferma del volo
- R12. garantire l'applicazione del massimo valore di sconto applicabile ad una prenotazione
- R13. garantire la sicurezza per l'accesso all'area riservata dell'amministratore tramite la cifratura della password

2.2.3 Vincoli

I vincoli sono dei requisiti di carattere generale che si applicano all'intero prodotto e che vengono definiti inizialmente prima dell'effettiva raccolta dei requisiti.

Sono stati specificati solo vincoli di carattere tecnico legati alle tecnologie da utilizzare per la realizzazione dell'applicazione.

È richiesto di implementare un software che deve:

- gestire la persistenza mediante l'uso di *JPA* (*Java Persistence API*), in particolare attraverso l'utilizzo di *Hibernate*;
- interagire con il software *DBMS MySQL*, attraverso una connessione con *JDBC* (*Java Database Connectivity*), per la gestione del database relazionale *SQL* mediante codice *JPQL* (*Java Persistence Query Language*) per formulare le query;
- utilizzare *CDI* (*Context and Dependency Injection*) per l'iniezione di dipendenze e per la gestione del ciclo di vita degli oggetti;
- esporre un'interfaccia realizzata mediante l'utilizzo di *JSF* (*JavaServer Faces*), di *AJAX* (*Asynchronous JavaScript and XML*) per lo scambio di dati asincrono tra browser e server e dei linguaggi *HTML* e *CSS*;
- interagire lato client attraverso l'uso del linguaggio di programmazione *Javascript* per la gestione dei controlli e per le semplici interazioni con l'utente;
- effettuare i test dell'applicazione utilizzando *JUnit* e *Mockito*, e *Jacoco* per verificarne la copertura.

2.3 Progettazione

La parte di progettazione dell'applicazione si svolge in più fasi. Inizialmente viene analizzato il problema ed i requisiti associati per estrarre un modello concettuale non dettagliato. In una fase successiva viene raffinato lo schema aggiungendo dettagli e metodologie di progettazione (e.g. design pattern) per ottimizzare il modello tenendo conto dei casi d'uso che emergono dall'analisi del problema. Vengono quindi realizzati degli schemi o draft (mockups) che mostrano dei modelli approssimati dell'interfaccia che presenterà l'applicazione, oltre ad un diagramma che mostra i collegamenti tra le varie pagine di presentazione.

2.3.1 Modello Concettuale

In Figura 2.1 è riportato in modo schematico il modello concettuale derivante da una prima analisi dalle specifiche presentate nel Paragrafo 2.1.

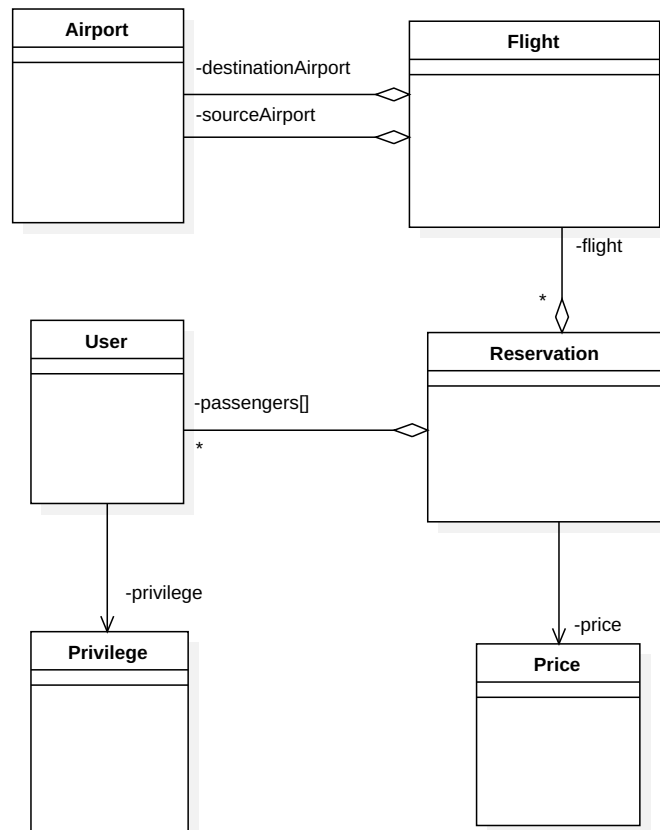


Figura 2.1: Modello concettuale.

Possiamo notare come lo schema proposto identifichi e rappresenti le entità principali che interagiscono nel sistema. Al centro del modello troviamo l'entità *Reservation* che identifica una prenotazione alla quale è associato un prezzo *Price* e contenente degli utenti *User*. Alla prenotazione è collegato un volo *Flight* al quale sono associati due aeroporti *Airport* distinguendo il ruolo di aeroporto di partenza *sourceAirport* e di destinazione *destinationAirport*. Inoltre è stato assegnato un privilegio *Privilege* all'entità utente in modo da considerare anche il ruolo di amministratore nello schema.

2.3.2 Modello di Dominio

In Figura 2.2 viene illustrato lo schema del modello di dominio in forma di *class diagram* costruito a partire dal modello concettuale in Figura 2.1 e dall'analisi dei requisiti nel Capitolo 2.2.

Lo schema si presenta molto più dettagliato rispetto al modello concettuale riportato in Figura 2.1. In particolare sono riportate le classi comprensive di attributi e relazioni tra esse, oltre all'aggiunta del *design pattern Strategy* che consente di selezionare diverse strategie per il calcolo dell'eventuale sconto sul prezzo, sulla base di alcuni parametri.

Questa scelta progettuale prevede di istanziare oggetti diversi nella gerarchia sulla base di alcuni parametri (come ad esempio il giorno di prenotazione, il numero di passeggeri ecc..) che si rivela un'ottima pratica nei casi in cui le diverse strategie sono scelte nella fase di raccolta dei requisiti e non vi è quindi la necessità di aggiungerne nuovi durante l'esecuzione dell'applicazione. Nel caso in cui invece si volesse fornire ad un utente amministratore la possibilità di inserire diverse strategie in un secondo momento senza dover interrompere il servizio, questa non si rivela la scelta più appropriata. In tal caso un'alternativa valida potrebbe essere quella di consentire all'amministratore di inserire dei parametri, persistiti poi nel database, che verranno utilizzati per determinare lo sconto finale. In questa applicazione è stata scelta la prima soluzione

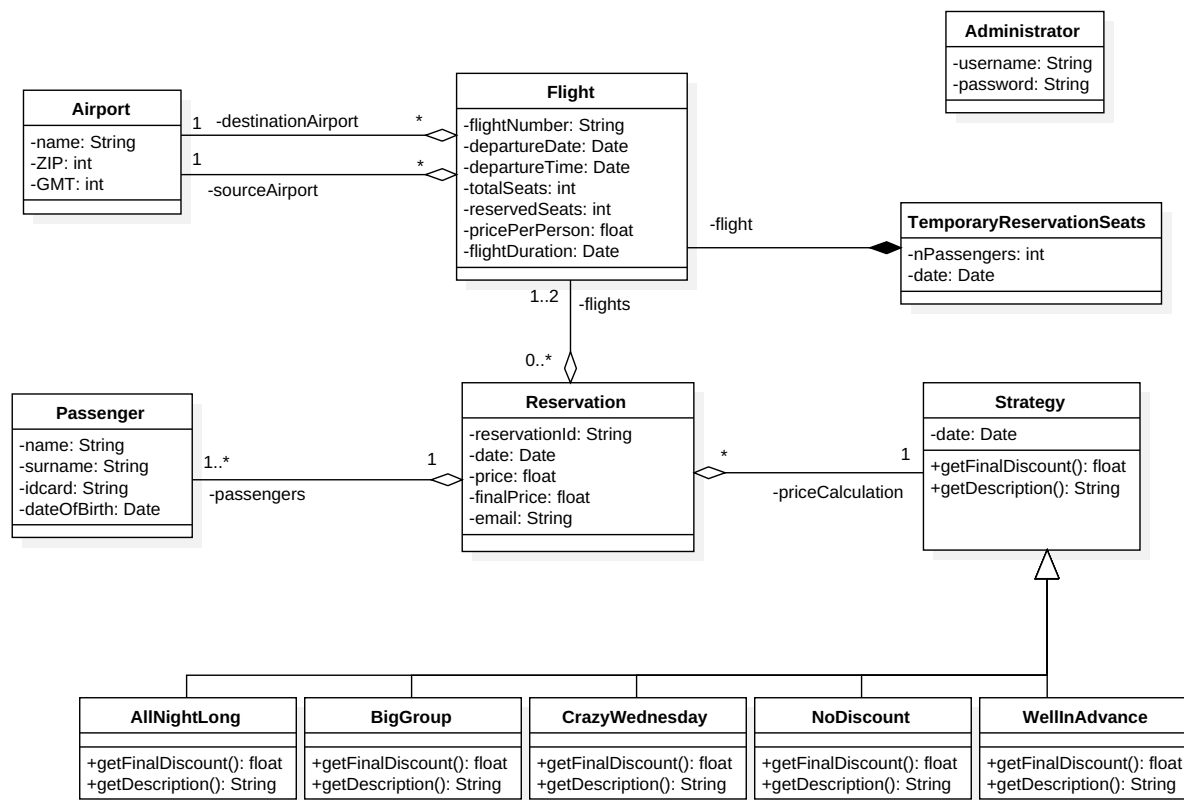


Figura 2.2: Diagramma delle classi riferite al modello di dominio.

principalmente a causa della complessità nell'implementazione della seconda in quanto sarebbe stato necessario definire un metodo di interpretazione dei parametri adatto a questo particolare tipo di soluzione e che va oltre gli scopi di questo elaborato. Un'altra motivazione di questa scelta è stata la sperimentazione nell'introduzione del design pattern *Strategy* e di come viene gestita la persistenza nel database.

Una divergenza rispetto al modello concettuale la troviamo anche nell'entità *Price* che non è stata rappresentata come classe nel modello di dominio, bensì come attributo della classe *Reservation* perché non identificato come punto di interesse in questa applicazione.

Un'altra differenza importante si ritrova nell'entità *Privilege*, che è stata completamente rimossa a causa di una scelta organizzativa differente dal modello nella separazione tra i ruoli di utente e amministratore. Sono stati distinti i ruoli di utente che ricerca dei voli e che eventualmente effettua una prenotazione, di passeggero facente parte della prenotazione e di amministratore. Nel nuovo schema infatti l'amministratore non rappresenta un'estensione di un utente generico.

È stata invece inserita l'entità *TemporaryReservationSeats* che ha il compito di tenere traccia delle prenotazioni in corso di svolgimento ma non ancora confermate. In particolare nel momento in cui un utente sceglie i voli di andate ed eventualmente di ritorno, viene creata un'istanza di questo oggetto per riservare in modo temporaneo i posti sui voli scelti in modo da gestire la concorrenza evitando quindi che un altro utente possa prenotare dei posti su un volo che in realtà potrebbero non essere disponibili. La questione verrà affrontata in modo più dettagliato nel prossimo capitolo.

Il sistema non tiene traccia dei dati dell'utente che crea la prenotazione e pertanto non viene rappresentato nel modello né persistito nel database, a meno che non venga inserito come passeggero della prenotazione e in tal caso viene solamente associato alla prenotazione stessa senza alcuna assegnazione di un ruolo specifico. L'amministratore invece possiede un'entità persistita

contenente i dati di accesso all'area personale per la gestione delle tratte aeree. L'accesso ai dettagli e alla gestione della prenotazione, diversamente, viene effettuato attraverso l'e-mail ed il codice di prenotazione che sono attribuiti alla prenotazione stessa; dunque non è l'utente che ha creato la prenotazione ad effettuare l'accesso, ma è un qualsiasi utente che possiede i dati della stessa.

2.3.3 Use Case Diagram

In Figura 2.3 è riportato un diagramma dei casi d'uso dell'applicazione che mostra le azioni che possono effettuare i tre attori che accedono al sistema. Verranno utilizzate le notazioni *User* per identificare il soggetto che vuole ricercare ed eventualmente effettuare una nuova prenotazione, *Reservation User* per identificare il soggetto che possiede i dati di accesso ad una prenotazione ed *Administrator* per identificare l'amministratore che gestisce i voli aerei.



Figura 2.3: Use Case Diagram.

L'utente standard che accede alla piattaforma avrà l'obiettivo di inserire una nuova prenotazione (*Confirm Reservation*) all'interno del sistema che sarà possibile solo dopo aver effettuato la selezione della tratta aerea desiderata attraverso la ricerca (*Search Flights*) e la scelta (*Choose Flights*) dei voli, e dopo l'inserimento dei dati dei passeggeri (*Insert Passengers Data*).

L'utente che possiede i dati di una prenotazione invece avrà la possibilità di effettuare il login all'interno della piattaforma (*Login*) che gli consentirà di visualizzare i dettagli della prenotazione (*View Reservation*), stampare il resoconto (*Print Reservation*), modificare i dati dei

passengeri (*Update Reservation*), cancellare la prenotazione (*Delete Reservation*) o effettuare il logout dal sistema (*Logout*).

L'amministratore potrà effettuare il login all'interno del piattaforma (*Login*) che gli consentirà di inserire nuove tratte aeree (*Insert Flight*), di cancellare alcuni voli (*Delete Flight*), di ricercare i voli presenti nel sistema (*Search Flights*) oppure di effettuare il logout (*Logout*).

2.3.4 Page Navigation Diagram

In Figura 2.4 è riportato il diagramma di navigazione delle pagine del sistema che mostra i collegamenti tra le varie pagine di presentazione che compongono l'applicazione, mostrando i pulsanti che conducono alle varie pagine. Al centro dello schema troviamo la *home page* dalla quale si accede alle diverse aree relative ai tre flussi mostrati nel diagramma dei casi d'uso in Figura 2.3 e descritti meglio in seguito nel Capitolo 5.1.

In alto viene mostrato il flusso dell'amministratore che è in grado di accedere alla pagina di inserimento delle credenziali attraverso il pulsante *administration* dalla home page e in seguito accedere all'area riservata attraverso il pulsante *login*. A questo punto l'amministratore è in grado di inserire un nuovo volo o di accedere alla pagina di ricerca voli attraverso il pulsante *show flights* dalla quale è possibile tornare indietro con il tasto *back to insert flights*. Da ogni pagina è possibile terminare la sessione e tornare alla *home page* attraverso il pulsante *logout*.

In basso viene mostrato il flusso relativo all'utente generico che è in grado di ricercare i voli dalla *home page* attraverso il pulsante *search* che lo porterà alla pagina dei risultati della ricerca. A questo punto l'utente potrà selezionare i voli desiderati e andare alla pagina di conferma attraverso il pulsante *confirm flights* dove dovrà completare l'inserimento dei dati e confermare definitivamente la prenotazione attraverso il pulsante *confirm reservation* che lo porterà verso la pagina di resoconto della prenotazione. Durante queste fasi è sempre possibile tornare alla *home page* ed effettuare una nuova ricerca attraverso il pulsante *cancel* o *back to home*.

A sinistra invece viene mostrato il flusso dell'utente che possiede i dati di una prenotazione attiva, che sarà in grado di accedere alla pagina di login attraverso il tasto *my booking* dalla home page e in seguito accedere alla pagina che mostra i dettagli della prenotazione dopo l'autenticazione, attraverso il pulsante *enter*. Da questa pagina è possibile accedere alla schermata di modifica dei passeggeri attraverso il pulsante *edit passengers* dalla quale sarà possibile tornare al resoconto della prenotazione in seguito all'aggiornamento dei dati dei passeggeri tramite il tasto *update* o semplicemente annullando le modifiche tramite il tasto *cancel*. È inoltre possibile tornare alla home page in seguito alla cancellazione della prenotazione con il pulsante *delete reservation* o attraverso il tasto *back to home*. Da ogni pagina è sempre possibile terminare la sessione e tornare alla home page attraverso il pulsante *logout*.

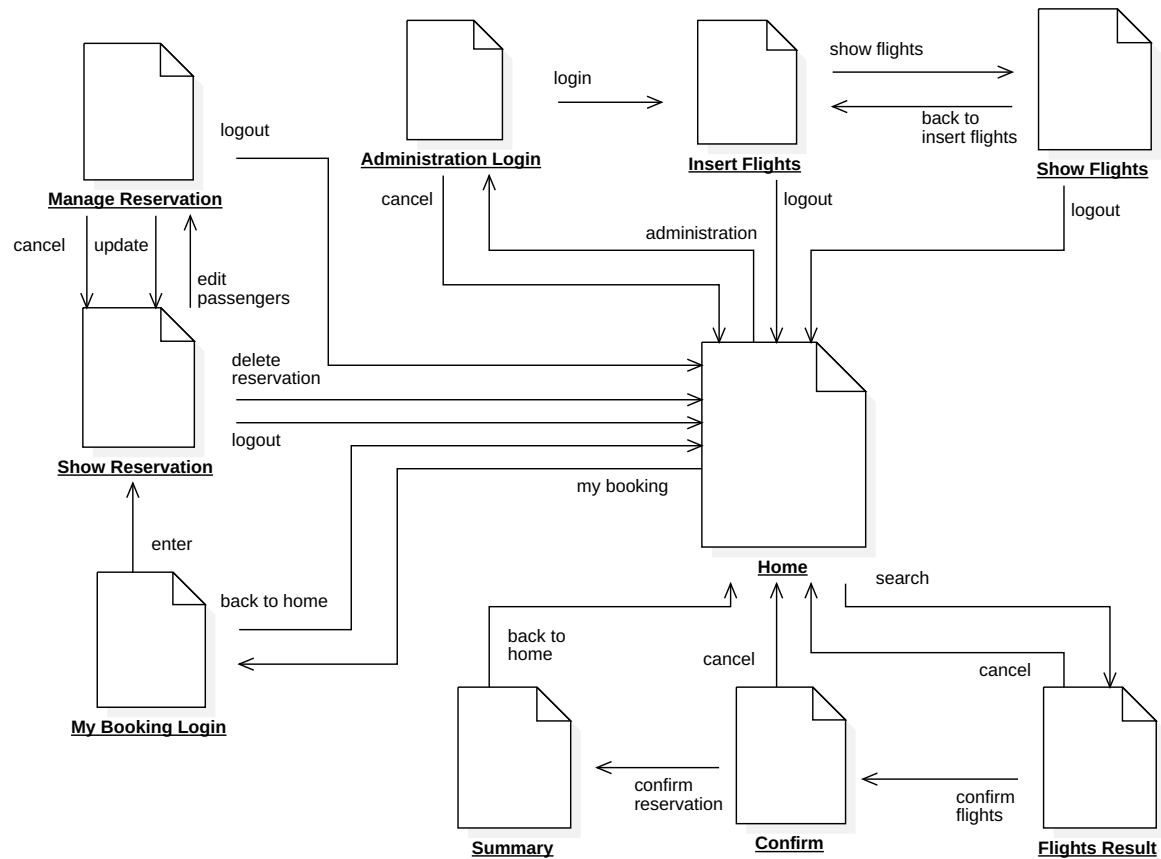


Figura 2.4: Page Navigation Diagram.

2.3.5 Mockups

Di seguito, dalla Figura 2.5 alla Figura 2.12, vengono mostrati i mockups delle pagine relative alla presentazione all'applicazione, ovvero le schermate approssimative che evidenziano le caratteristiche grafiche che le compongono. Ogni pagina contiene le informazioni in modo tale da permettere e valorizzare la gestione dei casi d'uso mostrati in Figura 2.3 mantenendo un'interfaccia utente semplice ed intuitiva (*user friendly*). Inoltre possiamo notare i diversi nomi dei pulsanti presenti nelle pagine che consentono di navigare nell'applicazione come descritto nel Page Navigation Diagram mostrato in Figura 2.4.

Flight Manager - Home

http://www.flight-manager.com

Flight Manager

My booking Administration

from to

Pisa ✈ Palermo

☐ one way ☒ return

date date

10/08/2017 18/08/2017

passengers

1

Search

Figura 2.5: Home page.

Flight Manager - Available flights

http://www.flight-manager.com

Flight Manager

Flights for Aug 10, 2017 - Passengers: 1

Forward flights

Pisa - 08:25	1h 10m	Palermo - 09:35	F1234	<input type="radio"/> 52,25€
Pisa - 22:25	1h 10m	Palermo - 23:35	F1235	<input checked="" type="radio"/> 45,50€

Flights for Aug 18, 2017 - Passengers: 1

Backward flights

Palermo - 10:00	1h 10m	Pisa - 11:10	F1236	<input checked="" type="radio"/> 70,45
Palermo - 00:00	1h 10m	Pisa - 01:10	F1237	<input type="radio"/> 65,40

Select

Figura 2.6: Risultati della ricerca dei voli.

Flight Manager - Confirm

http://www.flight-manager.com

Flight Manager

Pisa - 22:25	1h 10m	Palermo - 23:35	F1235	45,50€
Palermo - 10:00	1h 10m	Pisa - 11:10	F1236	70,45€

Insert email paul@stevenson.edu

Insert passengers data

	Name	Surname	Identity card number	Date of birth
Passenger 1	Paul	Stevenson	AB12345	10/11/1980

Total price: 115,95€

Confirm

Figura 2.7: Conferma della prenotazione.

Flight Manager - Summary
http://flights-manager.com

Flight Manager

Your flight has been booked successfully!

Summary data of your reservation:

Reservation ID: 5
Date of reservation: Jul 10, 2017
Passengers: 1
Price: 115,95€
E-mail: paul@stevenson.edu

Flight details:

Forward flight					
Pisa - 22:25	→ 1h 10m	Palermo - 23:35	F1235	45,50€	
Backward flight					
Palermo - 10:00	→ 1h 10m	Pisa - 11:10	F1236	70,45€	

Delete reservation

Figura 2.8: Riepilogo della prenotazione.

Flight Manager - My Booking
http://www.flight-manager.com

Flight Manager

Reservation ID

5

E-mail

paul@stevenson.edu

Login

Figura 2.9: Accesso all'area riservata per la visualizzazione della prenotazione.

Flight Manager - Admin Login
http://www.flight-manager.com

Flight Manager

Username

mario.rossi

Password

Login

Figura 2.10: Pagina di login lato amministratore.

Flight Manager - Insert a flight

http://www.flight-manager.com

Flight Manager

mario.rossi log out

Show all flights

Flight Number: F4652 Departure Time: 18:50 Duration: 1:25 Price per Person (€): 39.90 Date: 12/04/2017 Repeat for: 4 day(s) Total seats: 100

☒ existing ☐ new

From: Cagliari Destination: ZIP:

☐ existing ☒ new

To: Cagliari London Stansted 64321

Confirm

Figura 2.11: Inserimento di un volo (lato amministratore).

Flight Manager - Show all flights

http://www.flight-manager.com

Flight Manager

mario.rossi log out

Insert flight

Show Flights

Flight Number	From	To	Departure Time	Arrival Time	Price per Person
F1234	Pisa	Palermo	08:25	09:35	52.25€
F1235	Pisa	Palermo	22:25	23:35	45.50€
F1236	Palermo	Pisa	10:00	11:10	70.45€
F1237	Palermo	Pisa	00:00	01:10	65.40€

Figura 2.12: Visualizzazione dei voli (lato amministratore).

2.3.6 Architettura 3-tier

In Figura 2.13 è rappresentata una versione semplificata del progetto del layer logico dell'architettura 3-tier dell'applicazione, comprendente anche DAOs e Controllers.

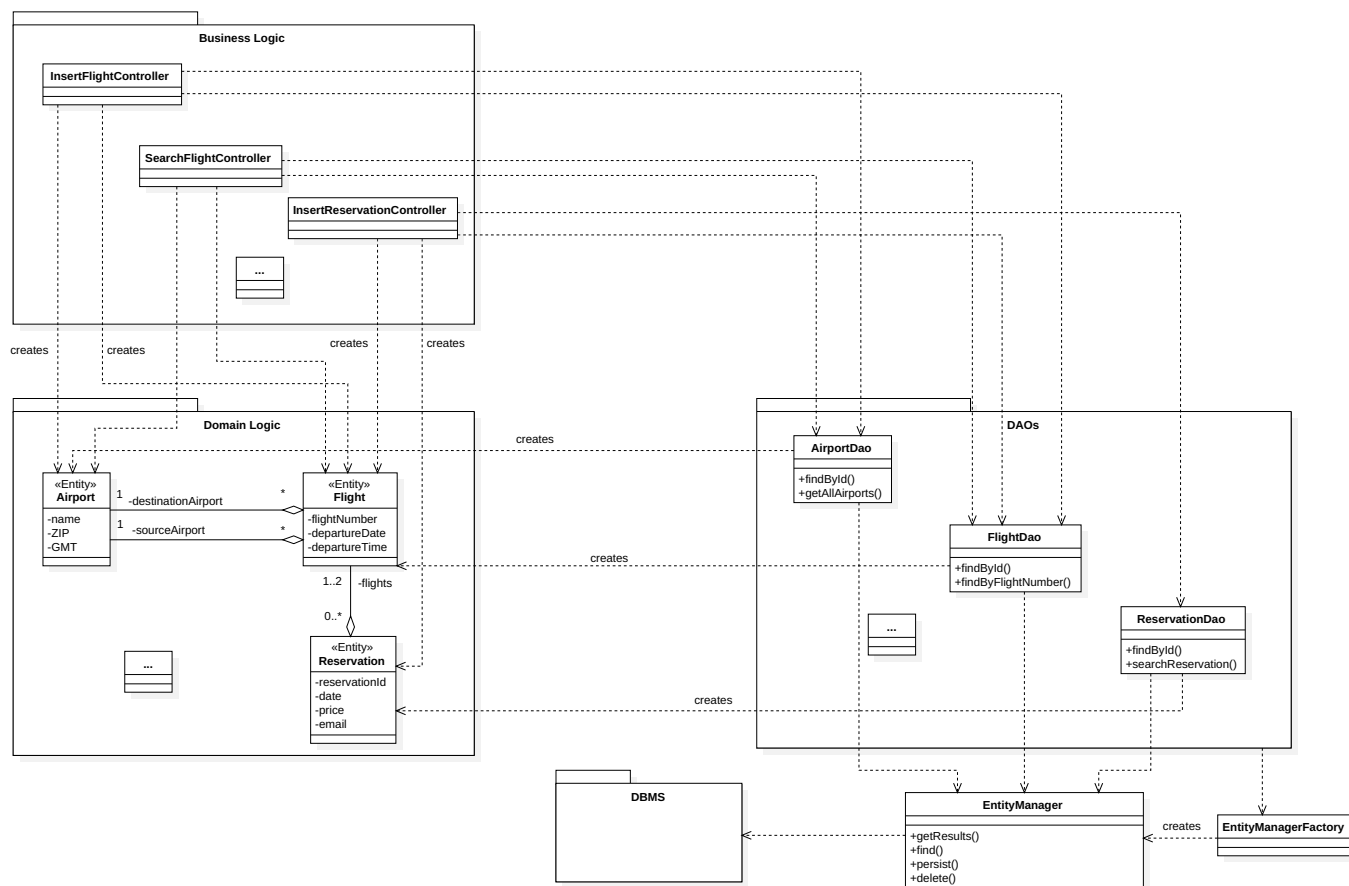


Figura 2.13: Relazioni tra Business Logic, Domain Model e DAOs.

Dal diagramma è possibile notare come i Controllers della Business Logic utilizzino oggetti della Domain Logic insieme ai DAO, i quali utilizzano un *EntityManager*, creato da un *EntityManagerFactory*, per collegare al database gli oggetti del modello logico.

Capitolo 3

Implementazione

Quanto descritto finora è stato implementato utilizzando lo stack tecnologico J2EE, che comprende Java, JPA/Hibernate e CDI; per l'implementazione dell'interfaccia è stato utilizzato invece JSF. Di seguito verranno mostrati in modo dettagliato tutti i livelli dell'architettura realizzata.

3.1 Domain Logic

Le classi Java facenti parte della logica di dominio sono state realizzate usando le annotazioni JPA. Come mostrato in Figura 2.2 abbiamo:

BaseEntity: È una classe astratta che viene estesa dalle altre classi del modello di dominio, mappata sul database attraverso l'annotazione *@MappedSuperclass*. Possiede attributi comuni a tutte le classi della logica di dominio, come *id* e *uuid*, ed effettua la ridefinizione, attraverso l'override, del metodo *equals()* per il confronto fra oggetti. L'attributo *id* ha l'annotazione *@id*, identificativo di ogni entità generato automaticamente dal DBMS con la strategia *IDENTITY*. L'attributo *uuid*, invece, è contrassegnato come unico.

ModelFactory: È una classe non persistita sul database, avente come unico compito quello di creare, attraverso metodi statici, delle istanze di oggetti della domain logic impostandogli un *uuid* corretto.

Administrator: È un'entità con attributi *username* e *password* e rappresenta l'utente amministratore utilizzato per l'accesso all'area riservata per la gestione dei voli.

Airport: È un'entità contenente gli attributi *name*, *ZIP* e *GMT* che rappresentano rispettivamente il nome dell'aeroporto, il suo codice postale e il Greenwich Mean Time, un numero intero compreso tra -12 e 12 che identifica il fuso orario di appartenenza dell'aeroporto.

Passenger: È un'entità che presenta gli attributi *firstName*, *surname*, *dateOfBirth* e *idCard*, che sono i dati necessari ad identificare un passeggero.

Flight: È un'entità che contiene tutti i dettagli di un volo, come *flightNumber*, *departureDate*, *DepartureTime*, *flightDuration* ed altri attributi fra i quali un riferimento all'aeroporto di partenza *sourceAirport* e all'aeroporto di destinazione *destinationAirport*, entrambi con annotazione *@ManyToOne*. Espone il metodo *copyFlight()* che, dato in ingresso un oggetto di tipo *Flight*, provvede ad effettuarne una copia degli attributi.

Reservation: È un'entità atta a contenere i dettagli di una prenotazione. Contiene gli attributi *reservationId*, *date*, *price*, *finalPrice*, *email*, una lista contenente i riferimenti ai passeggeri

passengers (*@OneToMany*), una lista contenente i riferimenti ai voli *flights* (*@ManyToMany*) ed un attributo *priceCalculation* (*@ManyToOne*) di tipo *Strategy* per il calcolo del prezzo con eventuali sconti.

Nei paragrafi che seguono verranno descritti altri aspetti implementativi del progetto, quali l'impiego del pattern *Strategy* per la scelta dello sconto da applicare, la creazione di una classe *Util* necessaria a svolgere operazioni di carattere generale, come ad esempio le operazioni e i controlli tra le date, e l'introduzione di un'entità *TemporaryReservationSeats* per l'allocazione temporanea di posti su un volo.

3.1.1 Strategy

Per una corretta realizzazione della scelta presentata nel paragrafo 2.3.2 relativa alla gestione dello sconto sul prezzo finale di una prenotazione è stata implementata una soluzione che teneva conto della possibilità di molteplici scelte dettate da una migliore convenienza. Questo meccanismo è stato quindi implementato attraverso l'utilizzo del design pattern *Strategy* come mostrato in Figura 3.1.

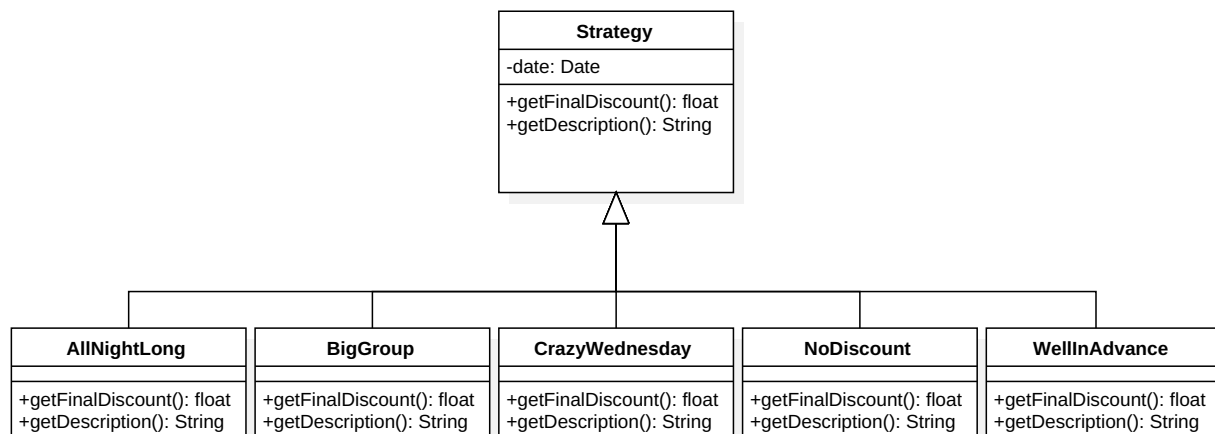


Figura 3.1: Pattern strategy per il calcolo dello sconto da applicare alla prenotazione.

A questo scopo è stata creata la generica classe astratta *Strategy* e alcune diverse implementazioni di strategie come descritto di seguito.

Strategy: È una classe astratta che rappresenta la generica strategia per l'applicazione dello sconto sul prezzo finale di una prenotazione. Possiede i codici identificativi *uuid* e *id* con i relativi metodi getter e setter che sono dichiarati rispettivamente come il codice univoco dell'istanza e il campo *@id* di ogni entità generato dal *DBMS* con strategia *IDENTITY*. Possiede inoltre la data di creazione della strategia e i metodi astratti *getFinalDiscount()* e *getDescription()* che delegano alle classi concrete la responsabilità di calcolare lo sconto finale di una data prenotazione e mostrare una breve descrizione della strategia utilizzata.

Ognuna delle seguenti classi estende la classe astratta *Strategy* ed implementa entrambi i metodi *getFinalDiscount()* e *getDescription()* secondo la propria strategia descritta di seguito:

- **NoDiscount:** Non viene applicato nessuno sconto alla prenotazione.
- **BigGroup:** Viene calcolato uno sconto del 5% ogni 5 passeggeri della stessa prenotazione fino ad un massimo del 15%.

- **AllNightLong:** Viene calcolato uno sconto dell'8% sul prezzo finale, applicato a prenotazioni notturne, cioè effettuate dalle ore 0:00 alle ore 6:00.
- **CrazyWednesday:** Viene calcolato uno sconto del 10% applicato a prenotazioni effettuate il giorno *Mercoledì*.
- **WellInAdvance:** Viene calcolato uno sconto del 25% applicato a prenotazioni effettuate con largo anticipo, ovvero con data di partenza superiore a 3 mesi dalla data di prenotazione.

L'applicazione consente di usufruire di un solo sconto per ogni prenotazione e viene applicato il metodo più conveniente per l'utente, ovvero quello che concede la maggiore percentuale sul prezzo finale tra i possibili sconti applicabili alla prenotazione stessa.

La gestione dell'ereditarietà all'interno del database, al fine di rappresentare lo schema mostrato in Figura 3.1, viene effettuata attraverso l'utilizzo della notazione *@Inheritance* specificando come tipo di ereditarietà *SINGLE_TABLE*. Questa scelta consente di mappare tutte le entità della struttura nella stessa tabella del database; questo approccio rende alcuni tipi di query (es. query polimorfiche) più efficienti e consente di ottenere migliori performance. Questo tipo di organizzazione risulta essere la soluzione più adatta a questo particolare tipo di scenario poiché le strategie concrete non aggiungono attributi o metodi alla classe padre dunque non necessitano di tabelle differenziate. Quando vengono persistite tutte le entità nella medesima tabella, *Hibernate* necessita di un modo per determinare a quale classe entità appartiene ogni record. Questa informazione è memorizzata in una colonna discriminativa che non fa parte degli attributi dell'entità, definita attraverso l'annotazione *@DiscriminatorColumn* nella classe padre, che nel nostro caso è di tipo stringa e chiamata *Strategy_Type*.

3.1.2 Util

Per ottenere una migliore separazione delle responsabilità delle diverse classi e per rendere il codice più omogeneo ed organizzato è stata creata la classe *Util* che racchiude una serie di metodi statici utili al calcolo di semplici funzioni.

Util: È una classe *stateless* composta da soli metodi statici utilizzata per il calcolo di funzioni utili al sistema, come ad esempio semplici operazioni sulle date o per l'estrazione di informazioni da un dato complesso.

Tra i metodi che espone questa classe sull'elaborazione delle date vi sono:

- **getOnlyHoursAndMinutes():** presa in ingresso una data completa, restituisce una stringa formata solo dall'ora e dai minuti nel formato "*HH:mm*";
- **getOnlyHoursAndMinutes():** restituisce una stringa sempre nel formato precedente, data però in ingresso una quantità espressa in minuti;
- **addMinutesToDate():** aggiunge i minuti inseriti ad una data in ingresso; *incrementDay()* che aggiunge un giorno ad una data in ingresso;
- **addDays():** utilizza il metodo precedente per aggiungere un numero di giorni scelto ad una data;
- **getOnlyDate():** data in ingresso una data completa, restituisce una stringa nel formato "*dd/MM/yyyy*";
- **subOneHour():** sottrae un'ora ad una data in ingresso.

Tra i metodi che non riguardano le date troviamo invece:

- **round()**: restituisce un numero reale approssimato con un numero di decimali scelto;
- **getArrivalTime()**: dato in ingresso un volo, restituisce una stringa contenente l'orario di arrivo dello stesso considerando la data di partenza, la durata del volo e i fusi orari degli aeroporti di partenza e di destinazione;
- **digest()**: a partire da una password in chiaro passata in ingresso, genera attraverso la funzione crittografica *SHA-1*, il digest corrispondente, utilizzato per la memorizzazione degli amministratori e per il controllo degli accessi in fase di login.

3.1.3 Temporary Reservation

L'applicazione consente ad un utente di ricercare e scegliere un volo in relazione ai posti disponibili nel sistema ed al numero di passeggeri selezionati nella ricerca. Una volta selezionato un volo che risulta disponibile, il sistema consente all'utente di completare la prenotazione entro lo scadere della sessione fissata da un tempo massimo. In questo intervallo l'applicazione deve tenere traccia del numero di posti temporaneamente occupati dalla prenotazione in corso e deve quindi mostrare ad eventuali nuovi utenti che effettuano la ricerca dello stesso volo una disponibilità inferiore in modo da evitare che i nuovi utenti possano occupare gli ultimi posti prima del completamento della prenotazione precedente. Se la prenotazione viene annullata o la sessione scade prima della conferma finale, il sistema provvederà a rendere nuovamente disponibili per nuove ricerche i posti precedentemente occupati. In questo modo il database rimarrà consistente in ogni fase della prenotazione differenziando le prenotazioni "reali" ovvero già completate dalle prenotazioni temporanee "attive" ovvero ancora in corso o quelle "scadute" ovvero quelle annullate o con sessione scaduta.

Questo meccanismo viene gestito dalla classe *TemporaryReservationSeats* che crea istanze di prenotazioni temporanee atte a tenere conto dei posti "virtualmente" occupati senza modificare i posti realmente occupati. In questo modo viene soddisfatto il requisito non funzionale R11.

TemporaryReservationSeats: È una classe che gestisce le prenotazioni temporanee per la corretta organizzazione delle occupazioni dei posti a sedere dei voli. Possiede come attributi un riferimento ad un volo *flight* con una annotazione *@ManyToOne* in modo da consentire ad istanze diverse di riferirsi allo stesso volo, il numero di passeggeri da occupare *nPassengers* e la data di creazione *date* con i relativi metodi getter e setter.

Per comprendere meglio il meccanismo di funzionamento delle prenotazioni temporanee, viene riportato di seguito un caso a scopo esemplificativo. Nelle Tabelle 3.1 e 3.2 sono mostrate alcune istanze delle tabelle *Flight* e *TemporaryReservationSeats* dove sono stati prenotati 115 posti su 120 totali e un utente sta completando la propria prenotazione riferita allo stesso volo (*flightId* in *TemporaryReservationSeats* è chiave esterna della tabella *Flight*) selezionando 2 passeggeri. In Figura 3.2 viene infatti mostrato come un utente generico visualizza i risultati di ricerca di un volo mentre un altro utente sta completando una prenotazione dello stesso mostrando solamente 3 posti disponibili.

id	flightNumber	departureDate	...	totalSeats	reservedSeats
54	F8520	09/09/2017	...	120	115
...

Tabella 3.1: Esempio di istanza di *Flight* nel database

id	date	flightId	nPassengers
121	09/09/2017 15:35:10	54	2
...

Tabella 3.2: Esempio di istanza valida di *TemporaryReservationSeats* nel database

From	Duration	To	Flight number	Price per person	
Parigi - 22:00	02:00	Firenze - 00:00 +1	F8520	€ 56.0	Only 3 seats available!
GMT: +1		GMT: +1			

Figura 3.2: Vista di un utente nella ricerca dei voli con altre prenotazioni temporanee attive.

3.2 Data Access Objects (DAO)

I Data Access Objects consentono di gestire la persistenza: consistono in classi Java che rappresentano delle entità tabellari del DBMS relazionale usate per stratificare ed isolare l'accesso ad una tabella tramite query, creando così un maggiore livello di astrazione ed una più facile manutenibilità. Consentono quindi di relazionare Domain Logic e database, sfruttando JPA (Hibernate) ed il linguaggio JPQL (Java Persistence Query Language) [3].

Tutti i DAO utilizzati possiedono degli attributi e dei metodi comuni anche se di tipi diversi. In particolare possiedono un *EntityManager* che viene iniettato attraverso l'utilizzo dell'annotazione *@PersistenceContext* di CDI ed espongono i metodi *save()*, *delete()* e *findById()*. Tutti i DAO, inoltre, implementano l'interfaccia *Serializable*, in modo che siano poi iniettabili in controllori della Business Logic aventi scope superiore a *@RequestScoped*. È stato dunque realizzato un DAO per ogni entità del modello che fosse persistito nel database:

AdministratorDao: Espone i metodi *existsAdministrator()* e *login()*. Il primo implementa la query che verifica se esiste almeno un'amministratore nel database: restituisce *true* in caso positivo, *false* in caso negativo. Il secondo metodo controlla se *username* e *password* di un oggetto *Administrator* corrispondono ad un amministratore sul database che, in caso positivo, verrà restituito in output.

AirportDao: Espone i metodi *getAllAirports()*, *getAllAirportsNames()* e *getAirport()*. I primi due implementano query che estraggono dal database tutti gli aeroporti presenti e restituiscono rispettivamente una lista di *Airport* ed una lista di nomi di aeroporti di tipo *String*. L'ultimo metodo implementa una query che, dato l'id dell'aeroporto, verifica se questo esiste sul database e lo restituisce in caso positivo.

FlightDao: Questa classe espone i metodi per la ricerca di voli nel database:

- *findByFlightNumber()* che, dato un *flightNumber*, ritorna il volo che possiede tale attributo;
- *getAllFlights()* che restituisce tutti i voli disponibili a partire da una data fornita in ingresso;
- *getFlights()* che, dati un aeroporto di partenza, uno di destinazione ed una data, restituisce la lista di voli che coprono quella tratta nel giorno specificato;
- *getFlightsFromDate()* che, dati un aeroporto di partenza, uno di destinazione ed una data, restituisce la lista di voli che coprono quella tratta a partire dal giorno specificato;

- *getFlightsFromDestination()* che restituisce la lista di voli in un determinato giorno che presentano un dato aeroporto come destinazione;
- *getFlightsFromSource()* che restituisce la lista di voli in un determinato giorno che partono da un dato aeroporto;
- *getSourceAirports()* e *getDestinationAirports()* che restituiscono rispettivamente una lista di aeroporti di partenza e una lista di aeroporti di destinazione dei voli presenti nel database;
- *getDestinationAirportsFromSource()* che, dato un aeroporto di partenza, restituisce una lista di aeroporti che sono destinazione dei voli presenti nel sistema;
- *getAvailableDateFromSourceToDestination()* che, dati un aeroporto di partenza ed uno di destinazione, restituisce tutte le date per le quali esiste almeno un volo disponibile su tale tratta.

PassengerDao: Espone semplicemente i metodi *save()*, *delete()* e *findById()* per la persistenza, l'aggiornamento, la cancellazione e la ricerca tramite id di un oggetto di tipo *Passenger*.

ReservationDao: Espone i metodi *searchReservation()* e *getIdFromLastReservation()*. Il primo implementa una query che cerca nel database una prenotazione avente *reservationId* e *email* dati; il secondo recupera l'id dall'ultima prenotazione effettuata.

StrategyDao: Espone semplicemente i metodi *save()*, *delete()* e *findById()* per la persistenza, l'aggiornamento, la cancellazione e la ricerca tramite id di un oggetto di tipo *Strategy*.

TemporaryReservationSeatsDao: Espone i metodi *getTemporaryReservedSeats()*, *getLastAdded()* e *cleanExpiredTemporaryReservation()*. Il primo restituisce il numero di passeggeri per i quali sono stati allocati dei posti in modo temporaneo per un dato volo; il secondo restituisce la più recente *TemporaryReservationSeats* persistita; il terzo provvede ad eliminare tutte le *TemporaryReservationSeats* scadute, ovvero presenti da un tempo superiore rispetto a quello ricevuto in ingresso.

3.3 Business Logic

La *Business Logic* mostrata in Figura 2.13 utilizza la logica di dominio ed è strettamente connessa ai casi d'uso dell'applicazione. Essa rappresenta un livello che si frappone tra l'interfaccia e la logica di dominio, focalizzandosi in particolar modo sulla creazione, modifica e gestione dei dati piuttosto che nella loro rappresentazione. La Business Logic dell'applicazione comprende *beans* e *controllers*.

3.3.1 Bean

I beans sono classi Java che presentano annotazioni CDI. All'avvio dell'applicazione il container CDI naviga le classi per identificare quali di loro possono rappresentare dei beans. All'interno di questa applicazione sono presenti principalmente due beans: *AdministratorSessionBean* è utilizzato per gestire il login/logout dell'amministratore; *ReservationSessionBean* è utilizzato invece per l'accesso ad una prenotazione. A questi si aggiunge il terzo bean *StartupBean* utile a creare un amministratore, qualora non esistesse, all'avvio dell'applicazione.

AdministratorSessionBean: Questo bean viene utilizzato per controllare l'accesso degli amministratori alla zona riservata alla gestione dei voli. È stata utilizzata l'annotazione *@SessionScoped* in modo da assegnare a questo bean un ciclo di vita pari a quello di una sessione

HTTP. La classe, inoltre, implementa l'interfaccia *Serializable* al fine di gestire l'insieme delle richieste HTTP. Gli attributi che questa classe presenta sono: *administratorId* riferito al codice identificativo dell'amministratore, *administrator* un oggetto di tipo *Administrator* e *administratorDao* di tipo *AdministratorDao* iniettato attraverso CDI. Vengono inoltre esposti i metodi che consentono di verificare se l'amministratore ha effettuato l'accesso o è scollegato.

ReservationSessionBean: Questo bean è molto simile al precedente, in quanto serve a controllare gli accessi di un utente che intende visualizzare o modificare una prenotazione. Anche per questa classe è stato utilizzato uno scope di tipo *@SessionScoped* ed estende l'interfaccia *Serializable*, per assegnargli un tempo di vita pari a quello di una sessione HTTP. Analogamente al bean precedente, questa classe presenta un attributo *id* riferito all'id della prenotazione, l'attributo *reservation* di tipo *Reservation* e l'oggetto *reservationDao* di tipo *ReservationDao* iniettato attraverso CDI.

StartupBean: Questo bean si occupa di popolare il database con un amministratore predefinito (*username: admin, password: password*) qualora il database non ne presentasse alcuno. Le annotazioni *@Startup* e *@Singleton* consentono di creare il bean all'avvio dell'applicazione, mentre l'annotazione *@PostConstruct* impone che il metodo annotato venga eseguito solamente dopo che l'oggetto di tipo *AdministrationDao* sia stato iniettato.

3.3.2 @HttpParam

Nella classe *HttpParam* viene definito il qualificatore *@HttpParam* che presenta il solo membro *value()*. Un qualificatore non è altro che un'annotazione Java che può essere applicata ad un bean. Dal momento che il membro *value()* presenta l'annotazione *@NonBinding*, questo verrà ignorato dal Container CDI poiché questa annotazione definisce una proprietà di un bean che non è rilevante nella definizione di un tipo.

Alla classe *HttpParams* è delegato il compito di restituire il valore del parametro attraverso la funzione *getParamValue()* che ha come unico parametro in ingresso un oggetto di tipo *InjectionPoint*. *InjectionPoint* è un'interfaccia che consente di avere accesso ad alcuni metadati riguardanti il punto di iniezione del bean, fra cui, in questo caso, anche il valore del parametro. Il metodo *getParamValue()* è inoltre annotato con *@Produces* in quanto dovrà essere in grado di creare un *@HttpParam*.

All'atto pratico la creazione di questa annotazione sarà poi utile per recuperare nei controllers alcuni parametri passati tra pagine XHTML attraverso il metodo GET, dunque contenuti nell'URL.

3.3.3 Controller

I controllers sono esattamente dei beans chiamati in modo differente per indicare la presenza di metodi che implementano le funzionalità vere e proprie dell'applicazione. All'interno dei controllers è racchiusa tutta la logica operativa dell'applicazione, che a sua volta utilizza i servizi messi a disposizione dai DAOs (per le transazioni sul database) e dai beans e che si serve dalla Domain Logic per la rappresentazione e la memorizzazione dei dati. Come già accennato precedentemente, i controllers costituiscono uno strato che si interpone tra l'interfaccia (JSF in questo caso), la Domain Logic ed i DAO. Di seguito vengono mostrati singolarmente tutti i controllers sviluppati per l'applicazione; per la loro descrizione si farà riferimento al Page Navigation Diagram rappresentato in Figura 2.4 e ai casi d'uso mostrati in Figura 2.3. Si noti che tutti gli attributi presenti nei controllers sono privati; proprio per questo motivo sono stati creati i metodi getter e setter per tutti gli attributi per i quali è necessario accedervi in lettura e scrittura dall'esterno della classe.

Di seguito verranno presentati i controller utilizzati dall'applicazione mostrando solo le caratteristiche più importanti e rimandando i particolari all'Appendice A.

- **SearchFlightsController:** Questo controller si occupa di gestire le pagine *Home* e *Flights Result* ed espone i metodi per la ricerca di voli e per la presentazione dei risultati in modo da soddisfare i requisiti funzionali R1 e R2. Questa classe presenta uno scope di tipo *Conversation* per poter controllare in modo arbitrario il tempo di vita della stessa. Tra i suoi attributi sono presenti dei *DAO* che consentono di estrarre informazioni dal database riguardanti i voli.
- **InsertReservationController:** Gestisce le pagine *Confirm* e *Summary* ed ha il compito di creare una prenotazione a partire dalle scelte compiute nella pagina iniziale e dai dati inseriti nella pagina di conferma in modo da soddisfare il requisito funzionale R3. Questo controller ha inoltre il compito di gestire le prenotazioni temporanee associate agli utenti che non hanno ancora confermato i voli mentre continua la conversazione iniziata nel controller precedente per poi chiuderla nel momento della conferma o in caso di annullamento della procedura. Questa classe si occupa inoltre di scegliere la tipologia di sconto da applicare.
- **ReservationLoginController:** Utilizza la pagina *My Booking Login* ed ha come scopo quello di gestire l'accesso di un utente alla sezione relativa alla visualizzazione, modifica ed eliminazione di una prenotazione.
- **ManageReservationController:** Questo controller si occupa della gestione delle pagine che mostrano il riepilogo delle prenotazioni, la modifica dei dati dei passeggeri e la cancellazione delle prenotazioni in modo da soddisfare rispettivamente i requisiti funzionali R4, R5 e R6. Presenta uno scope di tipo *Session* poiché dovrà rimanere in vita per un'intera sessione HTTP.
- **AdministratorLoginController:** Questo controller è del tutto simile a *ReservationLoginController*, salvo per il fatto che si occupa di gestire, attraverso la pagina *Administrator Login*, l'accesso di un utente amministratore per alla sezione riservata.
- **InsertFlightController:** Gestisce le pagine *Insert Flights* e *Show Flights* per l'inserimento di nuovi voli e nuovi aeroporti, e la visualizzazione dei voli presenti nel sistema in modo tale da soddisfare rispettivamente i requisiti funzionali R7, R8, R9. Questo controller consente inoltre di cancellare i voli nel sistema che non hanno prenotazioni attive per soddisfare anche il requisito funzionale R10.

3.4 Presentation Pages

L'interfaccia dell'applicazione è gestita attraverso la tecnologia *JSF*, che consente di unire la gestione e l'organizzazione dei dati all'interno dei controller utilizzando il linguaggio di markup *HTML5* e i fogli di stile *CSS3*, l'interazione e i controlli lato client attraverso *JavaScript*, e l'interazione asincrona lato server attraverso *AJAX*.

Le pagine ed i contenuti che compongono il livello di presentazione dell'applicazione sono presenti all'interno della cartella “\WebContent”. Come è possibile notare dalla Figura 3.3, all'interno di questa cartella sono presenti le pagine *.xhtml* oltre ad un'organizzazione gerarchica nelle sottocartelle “\css”, “\img” e “\layout”.

L'applicazione è stata configurata in modo tale da mostrare nella pagina degli indirizzi tutte le pagine con estensione *.xhtml*, come specificato all'interno del file di configurazione *web.xml*. La gestione delle pagine è organizzata includendo prima di ogni contenuto una pagina chiamata *template.xhtml*, presente nella cartella “\layout”, che ha il compito di inserire i tag HTML di

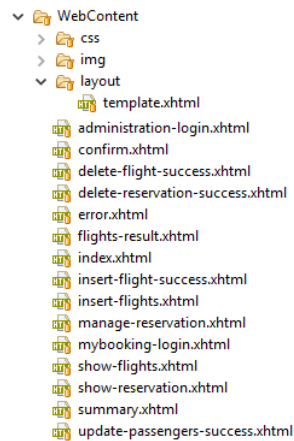


Figura 3.3: Organizzazione della cartella `\WebContent` per la gestione della grafica dell'applicazione.

base di una pagina web e di inserire i riferimenti ai file CSS esterni ed interni, presenti nella cartella `"\css"`.

3.4.1 JSF

JSF (JavaServer Faces) è una tecnologia Java che può essere considerata come un framework, per componenti lato server, di ambiente grafico con l'obiettivo di semplificare lo sviluppo dell'interfaccia web [4]. Questo è possibile grazie all'utilizzo di diversi *View Declaration Languages* [5], che consentono di interagire anche con i dati dei Controller o dei Bean della Business Logic e di inserirli all'interno di tag HTML standard. Un esempio di utilizzo di JSF è mediante l'utilizzo del tag `<h:panelGroup>` che è in grado di creare un tag HTML `` o `<div>` solo se il contenuto dell'attributo `rendered` restituisce un valore booleano pari a `true`. Questo tag viene utilizzato nel file `template.xhtml`, come mostrato nell'esempio in Figura 3.4, per la visualizzazione dello stato degli accessi riferiti ad un amministratore o ad una prenotazione, mostrando dunque l'username o il codice della prenotazione nella barra in alto della piattaforma insieme al logo e al nome dell'applicazione sempre visibili.

```
<h:panelGroup rendered="#{reservationSessionBean.isLoggedIn()}"> ... </h:panelGroup>
```

Figura 3.4: Esempio di utilizzo di tag JSF.

In Figura 3.4 viene mostrato un esempio di tag JSF all'interno del quale viene utilizzata un'istruzione del linguaggio *Java Unified Expression Language* [6] per fare riferimento a metodi e attributi privati dei Controller o dei Bean mediante la notazione `#{controller.method()}` o `#{controller.attribute}`. L'Expression Language infatti consente di accedere ad un attributo privato all'interno di una classe attraverso la definizione dei metodi getter e setter definiti con nomi standard (*description* \rightarrow `getDescription()`, `setDescription()`). Per non incorrere in errori di tipo `nullPointerException` sono stati ridefiniti tutti i metodi getter in modo tale da creare un'istanza dell'oggetto alla prima chiamata qualora abbia valore `null`.

3.4.2 CSS

La piattaforma utilizza, tramite riferimento remoto, i framework *Milligram* [7] e *Bootstrap* [8] che insieme consentono di ottenere una grafica semplice ed intuitiva, adattabile a diverse tipologie di

schermo e che, anche attraverso l'uso dei colori e dei piccoli particolari personalizzati, consente di navigare con estrema semplicità all'interno dell'applicazione. Come anticipato sono stati inoltre creati i file personalizzati “*FM-index.css*”, “*FM-reset-buttons.css*” e “*FM-template.css*” al fine di migliorare alcuni particolari della piattaforma e inserire delle personalizzazioni grafiche.

3.4.3 JavaScript

Per l'interazione ed i controlli lato client sono state create numerose funzioni javascript e sono state inserite all'interno delle pagine che le utilizzano. Queste funzioni sono state inserite all'interno del tag `<script>` e sono state implementate soprattutto nelle pagine che hanno il compito di gestire degli input per verificarne la correttezza o che devono inviare dati complessi che necessitano di controlli. Le funzioni più importanti sono descritte in Appendice B raggruppate per pagina.

3.4.4 AJAX

Per quanto riguarda le richieste asincrone tra browser e server, JSF mette a disposizione il tag `<f:ajax>` che permette di scegliere a quale evento associare l'aggiornamento di uno specifico componente della pagina o quale metodo di un Controller o un Bean chiamare attraverso l'utilizzo degli attributi *event* e *execute* o *render*. Questa funzionalità è stata ad esempio utilizzata nella home page, come mostrato nell'esempio in Figura 3.5, per aggiornare le date selezionabili una volta scelta la tratta desiderata, in modo da disabilitare le date con voli non esistenti.

```
<f:ajax event="click" render="form:availableDates" execute="form:source  
form:destination"/>
```

Figura 3.5: Esempio di aggiornamento delle date attraverso AJAX.

Questo tipo di controllo non può essere effettuato semplicemente lato client con JavaScript, poiché è necessaria una connessione al database per ottenere le date disponibili una volta selezionata la tratta desiderata.

Capitolo 4

Test

Il testing è un aspetto fondamentale nello sviluppo di un'applicazione, poiché permette di verificare la presenza di errori nel codice introdotti durante la fase di implementazione. Come ben noto, tuttavia, garantire la totale correttezza del codice è un problema *NP-Complete*.

I test effettuati per questa applicazione coprono tutte le classi ed i metodi, fatta eccezione per i metodi *getter*, *setter* ed *equals*, poiché assunti corretti. In questo capitolo verranno analizzati i test di ogni livello dell'architettura 3-tier, ossia Domain Logic, Data Access Objects e Business Logic. Questa parte del progetto è stata realizzata attraverso l'implementazione di test *JUnit*; nella parte di Business Logic si è inoltre introdotto *Mockito*.

4.1 Domain Logic

Nella parte di logica di dominio sono state implementate le classi a partire dal Class Diagram rappresentato in Figura 2.2, che rappresenteranno dunque entità ed attributi nel database. Non vengono quindi esposte funzionalità o logiche particolari dell'applicazione, ma principalmente metodi *getter* e *setter* che si occupano dell'accesso in lettura e scrittura agli attributi. Proprio per questo motivo, come già accennato in precedenza, la maggior parte dei metodi sono stati assunti corretti.

L'unica classe testata è *BaseEntity* che contiene il metodo *equals()*, centrale per la verifica dell'uguaglianza tra oggetti della logica di dominio. I test sul metodo provvedono a coprire i casi che possono presentarsi durante il confronto tra due oggetti. In particolare, nei casi in cui vi sono oggetti diversi dello stesso tipo, si confronta l'attributo UUID (Universally Unique Identifier), indispensabile per descrivere in modo univoco un oggetto sia all'interno dell'applicazione, sia durante la persistenza su database.

4.1.1 Strategy

Sono stati creati alcuni test sull'utilizzo delle strategie in modo da verificare la corretta creazione delle istanze delle classi durante una prenotazione fittizia. In questo modo viene soddisfatto il requisito non funzionale R12.

4.2 DAO

I Data Access Objects rappresentano uno strato che si interpone tra la Business Logic ed il DBMS. Essi contengono principalmente i metodi per l'accesso al database, dunque anche il codice JPQL (Java Persistence Query Language) per le query. I DAO dunque rappresentano un componente fondamentale per l'applicazione; sono quindi stati testati tutti i metodi per

verificare il corretto funzionamento di tutte le operazioni di persistenza ed interrogazione sul database.

Per agevolare la stesura dei test e favorire il riutilizzo del codice è stata creata la classe astratta *JpaTest*, classe padre di tutte le classi che effettuano test sui singoli DAO. Questa classe si occupa di inizializzare l'*EntityManager* e di svuotare tutte le tabelle presenti nel database prima che i test vengano eseguiti. Una volta eseguiti i test questa classe si occupa di effettuare il rollback delle operazioni effettuate durante ogni singolo test. Per effettuare i test è stato utilizzato un database diverso rispetto a quello usato dall'applicazione.

Le classi nelle quali sono presenti i test per i DAO estendono la classe astratta *JpaTest* ed al loro interno implementano la funzione *init()* della classe padre, atta ad inizializzare gli oggetti necessari ai test. I test di ogni singola classe coprono in modo esaustivo tutti i metodi presenti nei rispettivi DAO: *save()*, *delete()*, *findById()* ed altri metodi che effettuano interrogazioni specifiche al DBMS.

4.3 Business Logic

La Business Logic rappresenta tutta la logica applicativa che rende operativa l'applicazione e determina quindi come vengono creati, memorizzati e modificati i dati. Anche in questa parte, come nella Domain Logic, i metodi getter e setter sono stati assunti corretti. Per ogni controller sono stati testati tutti gli altri metodi come login e logout lato amministratore e l'accesso alle prenotazioni, verificando, anche a fronte di input nulli o non corretti, che avessero un comportamento corretto.

Per questi test, a differenza dei precedenti, è stato utilizzato *Mockito* principalmente per effettuare il mock sui DAO. Questo framework consente infatti di astrarre dall'effettiva implementazione dei metodi, associando un loro valore di ritorno per determinati input. Oltre a questa è stata utilizzata un'altra funzionalità che permette di verificare il numero di volte che viene eseguito un metodo del DAO.

4.4 Util

Una classe rilevante nella stesura dei test è *Util* che contiene i metodi statici utili ricorrenti nei metodi dei controller. Tra essi troviamo il metodo *digest()* i cui test mirano a verificare sia la correttezza del calcolo del digest attraverso il metodo crittografico SHA-1, sia la corretta memorizzazione dello stesso. In particolare date due password in chiaro p_1 e p_2 , con $p_1 = p_2$, deve valere che $d(p_1) = d(p_2)$, dove $d(\cdot)$ è la funzione che calcola il digest data la password in chiaro. L'applicazione di questo metodo consente infatti di soddisfare il requisito non funzionale R13.

4.5 Valutazioni Quantitative

Al fine di avere una valutazione quantitativa riguardante la copertura del codice è stato utilizzato EclEmma, un plugin di Eclipse basato su JaCoCo (Java Code Coverage).

In Figura 4.1 sono mostrate le percentuali di copertura sia dell'intera applicazione (88.2%), sia dei singoli package. Si noti che le percentuali non alte di copertura sono dovute alla numerosa presenza di metodi getter e setter assunti corretti.





































Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
▼  swam-flights-manage	 88,2 %	9.364	1.255	10.619
▼  src/main/java	 75,5 %	2.858	929	3.787
>  it.unifi.ing.swam.controller	 70,1 %	1.467	625	2.092
>  it.unifi.ing.swam.dao	 88,1 %	649	88	737
>  it.unifi.ing.swam.bean	 41,4 %	58	82	140
>  it.unifi.ing.swam.model.strategy	 49,2 %	63	65	128
>  it.unifi.ing.swam.model	 90,7 %	312	32	344
>  it.unifi.ing.swam.bean.producer	 0,0 %	0	17	17
>  it.unifi.ing.swam.util	 94,9 %	261	14	275
>  it.unifi.ing.swam.model.temp	 78,6 %	22	6	28
>  it.unifi.ing.swam.bean.startup	 100,0 %	26	0	26
▼  src/test/java	 95,2 %	6.506	326	6.832
>  it.unifi.ing.swam.controller	 93,1 %	3.481	257	3.738
>  it.unifi.ing.swam.dao	 98,4 %	2.550	42	2.592
>  it.unifi.ing.swam.bean	 89,8 %	106	12	118
>  it.unifi.ing.swam.util	 96,9 %	283	9	292
>  it.unifi.ing.swam.bean.startup	 87,5 %	42	6	48
>  it.unifi.ing.swam.model	 100,0 %	44	0	44

Figura 4.1: Analisi con JaCoCo sulla copertura dei test.

Capitolo 5

Funzionalità e Utilizzo

In questo capitolo verrà presentato il funzionamento generale dell'applicazione. Verranno in primo luogo presentati i tre principali flussi di utilizzo, seguiti da una panoramica dettagliata dell'intera applicazione.

5.1 Flussi di Utilizzo

L'applicazione è stata ideata con l'intento di sviluppare diverse funzionalità nell'ambito della gestione e ricerca di voli aerei creando diversi scenari utili in contesti differenti e con obiettivi diversi. Sono stati individuati principalmente tre flussi di navigazione indipendenti accessibili dalla home page e mirati a tre tipologie di utenti differenti:

- La ricerca e la scelta di voli per la creazione di una prenotazione disponibile a tutti gli utenti che accedono all'applicazione
- La visualizzazione e la gestione della propria prenotazione accessibile tramite credenziali fornite al termine della creazione della prenotazione del punto precedente
- La visualizzazione e l'inserimento di nuovi voli nell'applicazione accessibile solo agli amministratori attraverso credenziali personali

5.2 Panoramica dell'Applicazione

L'applicazione è chiamata *Flight Manager* e si presenta con la pagina principale *index.xhtml* che consente di cercare un volo e iniziare la procedura per una nuova prenotazione. Ogni pagina possiede una barra celeste in alto con il nome e il logo dell'applicazione che se cliccati consentono di tornare alla pagina principale (Figura 5.1). Nella stessa barra sulla destra saranno riportate le informazioni riferite alla sessione in corso.



Figura 5.1: Flight Manager - Logo dell'applicazione.

5.2.1 Ricerca dei Voli e Nuova Prenotazione

La pagina principale consente di cercare un volo selezionando gli aeroporti di partenza e destinazione e dà la possibilità di scegliere voli di sola andata o anche di ritorno scegliendo le date partenza e di ritorno e selezionando il numero di passeggeri. La home page consente inoltre di

accedere alle sezioni riservate alla gestione delle prenotazioni e alla sezione di amministrazione attraverso i pulsanti in alto a destra rispettivamente *My Booking* e *Administration* (Figura 5.2).

Search flight

MY BOOKING ADMINISTRATION

One Way Return

From Bologna To Londra Stansted

Fly out 21/08/17 Fly back 24/08/17

Passengers 1

SEARCH

Figura 5.2: Pagina principale di ricerca voli.

Nell'elenco degli aeroporti di partenza sono visualizzati tutti gli aeroporti disponibili ma sono selezionabili solo quelli che hanno almeno un volo disponibile dall'aeroporto scelto e analogamente verranno proposti solo gli aeroporti di destinazione disponibili. Selezionata la tratta verranno mostrate solo le date disponibili dalla data odierna sia per l'andata che per il ritorno (Figura 5.3).

August 2017

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

21/08/17

Figura 5.3: Scelta della data.

Cliccando il pulsante di ricerca verranno visualizzati tutti i voli disponibili per la tratta e la data selezionata mostrando in alto il riepilogo della ricerca e subito sotto i risultati. Nella schermata vengono divisi i voli di andata e di ritorno mostrando per ogni volo gli aeroporti della tratta, gli orari di partenza e di arrivo con particolare attenzione al fuso orario di appartenenza dell'aeroporto, la durata del volo, il numero del volo e il prezzo per persona (Figura 5.4).

Se il numero di posti disponibili del volo è terminato o è minore del numero di passeggeri scelto verrà visualizzato un messaggio di errore che non consentirà di selezionare il volo. In tutti i casi se sono rimasti pochi posti disponibili verrà visualizzato un messaggio di allerta con il numero di posti rimasti.

Se la ricerca del volo ha mostrato almeno un risultato disponibile, sarà possibile confermare il volo scelto attraverso il pulsante *Confirm Flights* per procedere con la fase di inserimento

Flights results

Forward flights - From **Bologna** to **Londra Stansted** on **21/08/2017** for **6** passengers

From	Duration	To	Flight number	Price per person
Bologna - 16:45 GMT: +1	01:10	Londra Stansted - 16:55 GMT: 0	F9846	€ 49.99 ●
Bologna - 08:15 GMT: +1	01:30	Londra Stansted - 08:45 GMT: 0	F6491	€ 79.0 ○

Backward flights - From **Londra Stansted** to **Bologna** on **24/08/2017** for **6** passengers

From	Duration	To	Flight number	Price per person
Londra Stansted - 08:56 GMT: 0	01:32	Bologna - 11:28 GMT: +1	F9164	€ 39.0 ●

CANCEL

CONFIRM FLIGHTS

Figura 5.4: Risultati di ricerca dei voli.

dei dati della prenotazione. In ogni momento è possibile tornare alla pagina principale per una nuova ricerca attraverso il pulsante rosso *Cancel*.

Dal momento in cui vengono scelti i voli e si prosegue alla fase successiva per la conferma dei dati, vengono riservati i posti relativi alla prenotazione in atto fino al termine della sessione dell'utente, o fino a quando la prenotazione non viene cancellata. L'utente avrà quindi la possibilità di completare l'inserimento dei dati personali mantenendo la prenotazione sui posti disponibili e impedendo ad utenti successivi di completare la disponibilità dei voli.

Allo scadere della sessione, stabilita da un tempo massimo, se l'utente non avrà completato la prenotazione i posti riservati saranno nuovamente liberi e l'utente dovrà ricominciare la procedura dalla ricerca dei voli (Figura 5.5).

ATTENTION: session expired! Retry to insert data!

Figura 5.5: Sessione scaduta.

Nella schermata di inserimento dei dati della prenotazione l'utente dovrà inserire un indirizzo email valido che sarà poi utilizzato per l'accesso all'area relativa alla gestione della prenotazione come specificato dall'indicatore giallo accanto.

In uno scenario reale l'indirizzo email inserito è utilizzato per l'invio delle informazioni relative alla prenotazione mentre nell'applicazione corrente l'indirizzo email inserito non sarà utilizzato per inviare informazioni all'utente ma sarà utile solo per la fase di accesso alla prenotazione (Figura 5.6).

L'utente dovrà inoltre inserire tutti i dati dei passeggeri quali nome, cognome, numero di carta d'identità e data di nascita; eventuali informazioni incomplete o scorrette verranno segnalate (Figura 5.7). In questa fase non è possibile utilizzare i dati di un passeggero che è già stato inserito all'interno dell'applicazione attraverso una precedente prenotazione, pertanto sarà necessario inserire tutti i dati dei passeggeri anche se già presenti nel sistema e verranno trattati come nuovi passeggeri.

Insert email address

Email Address

The email address allows you to view and modify the reservation in **My Booking** section

Figura 5.6: Inserimento indirizzo email.

Insert passengers data

	First name	Surname	ID-Card	Date of birth
Passenger 1	<input type="text" value="First name"/>	<input type="text" value="Surname"/>	<input type="text" value="Identity card number"/>	<input type="text" value="dd/mm/yy"/>
Passenger 2	<input type="text" value="First name"/>	<input type="text" value="Surname"/>	<input type="text" value="Identity card number"/>	<input type="text" value="dd/mm/yy"/>
Passenger 3	<input type="text" value="First name"/>	<input type="text" value="Surname"/>	<input type="text" value="Identity card number"/>	<input type="text" value="dd/mm/yy"/>

CANCEL BACK CONFIRM RESERVATION

Figura 5.7: Inserimento dati dei passeggeri.

Nella pagina sarà visualizzato un box in alto a destra contenente i dettagli del prezzo totale della prenotazione specificando il prezzo di andata e di ritorno per persona e calcolato sul numero di passeggeri. L'applicazione mette inoltre a disposizione la possibilità di effettuare sconti alle tariffe in base a diversi criteri quali ad esempio il numero elevato di passeggeri, una prenotazione notturna etc. Viene applicato al massimo un criterio di sconto ed è relativo al maggiore, in percentuale, tra quelli di cui l'utente ha diritto. Il prezzo finale verrà visualizzato in fondo al box ed è possibile conoscere i dettagli dello sconto (se applicato) tramite l'indicatore giallo (Figura 5.8).

Price		
	per person	x 6
Forward flight:	€ 49.99	€ 299.94
Backward flight:	€ 39.0	€ 234.0
Total:		€ 533.94
Special offer!		€ 507.24

Discount applied for big groups (5% each 5 people)

Figura 5.8: Box del prezzo finale con dettaglio dello sconto.

Completato l'inserimento dei dati e confermata la prenotazione tramite il tasto verde *Confirm reservation* verrà visualizzata la pagina contenente il resoconto finale della stessa con i dati relativi ai voli prenotati e le credenziali di accesso per la visualizzazione e gestione della prenotazione nell'apposita area riservata (Figura 5.9).

È possibile e consigliato stampare il resoconto della prenotazione utilizzando il pulsante blu *Print* presente in fondo alla pagina per non perdere le credenziali di accesso.

Your flights have been booked successfully! ✓

Reservation date:	Mon Aug 21 12:41:07 CEST 2017
Email:	myemail@mail.com
Reservation ID:	FMID-74
Passengers:	6
Total price:	€ 507.24

Flight number	Date	From	Duration	To
F9846	2017-08-21	Bologna - 16:45	01:10	Londra Stansted - 16:55
F9164	2017-08-24	Londra Stansted - 08:56	01:32	Bologna - 11:28

BACK TO HOME PRINT

Figura 5.9: Resoconto della prenotazione.

5.2.2 Gestione della Prenotazione

La sezione relativa alla gestione della prenotazione consente ad un utente che ha già effettuato una prenotazione di visualizzarne i dettagli, stampare il resoconto dettagliato, modificare le informazioni dei passeggeri e anche cancellare l'intera prenotazione.

Per accedere alla prenotazione dalla pagina principale l'utente dovrà cliccare sul pulsante *My Booking* ed effettuare l'accesso inserendo il codice di prenotazione fornito al momento della conferma della prenotazione insieme all'indirizzo email scelto durante l'inserimento dei passeggeri (Figura 5.10).

My booking login

Reservation ID

Email

BACK TO HOME ENTER

Figura 5.10: Pagina di login per la gestione prenotazione.

Se i dati inseriti sono errati verrà visualizzato un messaggio di errore, altrimenti l'utente verrà reindirizzato direttamente alla pagina di resoconto della prenotazione.

Una volta effettuata correttamente l'autenticazione verranno visualizzati in alto a destra di ogni pagina il codice di prenotazione, un avatar e il pulsante di *Logout* che consente di terminare la sessione e tornare alla pagina di login (Figura 5.11).

Come mostrato in Figura 5.12, la pagina di resoconto è suddivisa in tre sezioni:



Figura 5.11: Profilo prenotazione.

- I dettagli della prenotazione come la data di prenotazione, l'indirizzo email scelto, il codice di prenotazione, il numero di passeggeri e il prezzo totale
- I voli scelti (di sola andata o anche di ritorno se selezionato) con i dettagli completi come il nome degli aeroporti della tratta, la data e l'orario di partenza, l'orario di arrivo e la durata del volo
- Le informazioni dei passeggeri inseriti durante la prenotazione

Your booking details

Reservation date:	2017-08-21 12:41:07.0
Email:	myemail@mail.com
Reservation ID:	FMID-74
Passengers:	6
Total price:	€ 507.24

Your flights

Flight number	Date	From	Duration	To
F9846	2017-08-21	Bologna - 16:45	01:10	Londra Stansted - 16:55
F9164	2017-08-24	Londra Stansted - 08:56	01:32	Bologna - 11:28

Passengers

First Name	Surname	ID-Card	Date of birth
Paolo	Mengoni	R34234	12/04/1956
Marco	Lippi	T43524	29/04/1993

Figura 5.12: Pagina di resoconto della prenotazione.

Al termine della pagina sono presenti quattro pulsanti che consentono rispettivamente di tornare alla pagina di ricerca voli (*Back to home*), stampare la pagina di resoconto (*Print*), di cancellare l'intera prenotazione (*Delete Reservation*) e di modificare i dati dei passeggeri (*Edit passengers*).

Cliccando il pulsante di cancellazione della prenotazione, verrà richiesta la conferma per procedere che porterà alla eliminazione dell'intera prenotazione insieme ai passeggeri ad essa

associati; verrà inoltre effettuato il logout mostrando un messaggio di avvenuta cancellazione (Figura 5.13).

Your reservation has been deleted successfully! ✓

BACK TO HOME

Figura 5.13: Conferma di avvenuta cancellazione.

Cliccando il pulsante di modifica dei passeggeri, l'utente verrà reindirizzato in una pagina che gli consentirà di modificare il nome, il cognome, il numero di carta d'identità e la data di nascita di ognuno dei passeggeri attraverso dei campi precompilati dalle informazioni inserite in precedenza. L'utente potrà aggiornare le nuove informazioni cliccando il tasto *Update* che mostrerà i nuovi dati inseriti, o annullare le modifiche cliccando il pulsante *Cancel* per tornare alla sezione precedente (Figura 5.14).

Edit passengers

First Name	Surname	ID-Card	Date of birth
Paolo	Mengoni	R34234	12/04/56
Marco	Lippi	T43524	29/04/93
Francesco	Morra	FA12043	06/08/84

CANCEL UPDATE

Figura 5.14: Pagina di modifica dei passeggeri.

5.2.3 Area di Amministrazione

La gestione amministrazione consente ad un amministratore di inserire nuovi voli selezionando aeroporti esistenti o creandone di nuovi; consente inoltre di visualizzare i voli già presenti nel sistema filtrando per aeroporto di partenza o di destinazione.

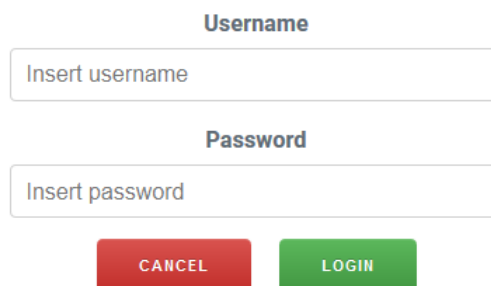
Per accedere alla sezione di amministrazione, dalla pagina principale è possibile cliccare sul pulsante *Administration* ed inserire username e password nella pagina di login (Figura 5.15). Le credenziali di accesso di un amministratore vengono create lato backend e inserite direttamente nel sistema; pertanto non possono essere create o modificate dall'applicazione, ma solo utilizzate per la fase di accesso all'area di amministrazione.

Se i dati inseriti sono errati verrà visualizzato un messaggio di errore, altrimenti l'amministratore verrà reindirizzato alla pagina di inserimento di nuovi voli.

Una volta effettuata correttamente l'autenticazione verrà visualizzato in ogni pagina in alto a destra lo username, un avatar e il pulsante di *Logout* che consente di terminare la sessione e tornare alla pagina di login (Figura 5.16).

La pagina per l'inserimento di un nuovo volo consente di scegliere il numero del volo, la data e l'orario di partenza, la durata del volo, il numero massimo di passeggeri, il prezzo per persona e il numero di giorni consecutivi per i quali si vuole ripetere lo stesso volo per un massimo di 31. Appena sotto è possibile selezionare la tratta desiderata selezionando l'aeroporto di partenza e

Administration login



The login form consists of two input fields and two buttons. The first field is labeled 'Username' and contains the placeholder text 'Insert username'. The second field is labeled 'Password' and contains the placeholder text 'Insert password'. Below the fields are two buttons: a red 'CANCEL' button and a green 'LOGIN' button.

Figura 5.15: Pagina di login per la gestione amministrazione.



Figura 5.16: Profilo amministratore.

di destinazione tra l'elenco di tutti gli aeroporti. È inoltre possibile creare un nuovo aeroporto selezionando la voce *New* e inserendo il nome, il codice ZIP dell'aeroporto e scegliendo il fuso orario di appartenenza (tra -12 a 12) (Figura 5.17).

Eventuali informazioni incomplete o scorrette verranno segnalate come il formato dell'ora di partenza errato o l'inserimento di nuovi aeroporti con nomi già esistenti.

L'amministratore è inoltre in grado di cercare tutti i voli inseriti nel sistema cliccando sul pulsante *Show flights* che lo porterà alla schermata di ricerca. In questa pagina è possibile selezionare un aeroporto di partenza, uno di destinazione e una data dalla quale effettuare la ricerca; cliccando sul pulsante *View flights* verranno quindi visualizzati tutti i voli disponibili su quella tratta a partire dalla data scelta. I risultati di ricerca sono completi di numero di volo, data e ora di partenza, durata del volo, posti totali, posti occupati e prezzo per persona; i voli risultanti sono ordinati per data crescente.

È anche possibile selezionare l'opzione *Any* nella scelta degli aeroporti di partenza o di destinazione in modo tale da mostrare i risultati rispettivamente da un qualsiasi aeroporto di partenza verso la destinazione selezionata, o dalla partenza selezionata verso qualsiasi destinazione. In questi casi verranno mostrati tra i risultati di ricerca anche gli aeroporti di partenza o di destinazione mancanti. Selezionando la voce *Any* su entrambi gli aeroporti verranno visualizzati tutti i voli disponibili da e verso qualsiasi destinazione sempre a partire dalla data scelta (Figura 5.18).

L'amministratore sarà inoltre in grado di cancellare i voli che non presentano prenotazioni attive, ovvero quelli con posti prenotati pari a zero, cliccando il pulsante rosso *Delete* di fianco al volo; dopo aver confermato l'operazione di cancellazione l'amministratore verrà reindirizzato ad una pagina di conferma dell'avvenuta operazione riportandolo alla gestione dei voli.

È sempre possibile tornare alla pagina per l'inserimento di un nuovo volo cliccando sul pulsante *Back to insert flight*.

Insert flight

Flight number	Date	Departure time (HH:MM)	Flight duration (minutes)	Passengers	Price per person (€)	Repeat for (days)
<input type="text" value="F1234"/>	<input type="text" value="21/08/17"/>	<input type="text" value="17:55"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0.0"/>	<input type="text" value="1"/>

Source airport existing ☐ new ☒

From

Destination airport existing ☒ new ☐

To

Figura 5.17: Pagina di inserimento nuovi voli.

Show flights

Starting date

From To

Flight number	Date	Departure time (HH:MM)	Flight duration (minutes)	Total seats	Reserved seats	Price per person	From	
F5256	01/08/2017	21:00	70	300	6	€ 75.0	Londra Stansted	
F5221	01/08/2017	21:00	70	300	0	€ 75.0	Palermo	<input type="button" value="DELETE"/>
F4646	01/08/2017	18:42	56	78	1	€ 130.0	Londra Stansted	
F4984	03/08/2017	12:10	98	87	0	€ 15.0	Pisa	<input type="button" value="DELETE"/>

Figura 5.18: Pagina di ricerca voli.

Capitolo 6

Conclusioni e sviluppi futuri

Lo scopo di questo elaborato è stato quello di progettare e realizzare un'applicazione per la gestione dei voli aerei. A partire da una panoramica del problema si è proceduto con la progettazione e la realizzazione dell'applicazione secondo le specifiche delineate nella fase iniziale del lavoro. I test formali e funzionali mostrano ampiamente la correttezza delle funzionalità che l'applicazione è tenuta a svolgere.

Questo lavoro è nato fondamentalmente con l'intento di realizzare un'applicazione a scopo puramente accademico, che potesse racchiudere molte funzionalità e strumenti di un progetto reale seguendo le linee guida per l'utilizzo di diversi linguaggi di programmazione, tecniche di progettazione varie e implementazioni ottimizzate. Per questo scopo molti dettagli riguardo la realizzazione di una vera piattaforma per la gestione e prenotazione di voli aerei sono stati omessi, poiché avrebbero aumentato notevolmente i tempi di realizzazione e non sarebbero stati utili ai fini del progetto.

Ciò che è stato realizzato in questo elaborato è comunque solido rispetto alle funzioni di base implementate e le scelte progettuali consentono di realizzare eventuali estensioni in modo piuttosto agevole. Di seguito sono elencate alcune possibili idee che possono essere utilizzate per migliorare od estendere l'applicazione:

- Creare il biglietto in formato PDF identificato attraverso un codice a barre o QR-code;
- Inviare un'e-mail di conferma di avvenuta prenotazione all'indirizzo e-mail inserito con un resoconto della prenotazione e/o il biglietto in formato PDF;
- Abilitare un metodo di pagamento (Paypal, bonifico);
- Aggiungere la possibilità di modificare o cancellare i voli: al momento l'applicazione consente ad un amministratore di cancellare i voli che non presentano alcuna prenotazione attiva, ma potrebbe essere implementata la funzione di modifica dei voli presenti nel sistema (con conseguente notifica alle prenotazioni attive) o cancellazione con possibilità di cambio volo o rimborso del prezzo totale;
- Modificare il prezzo dei biglietti aerei: per come è stata progettata e realizzata l'applicazione, è possibile modificare il prezzo di un volo aereo senza influire sul calcolo del prezzo finale delle prenotazioni precedenti, dando la possibilità alla compagnia aerea di applicare rincari o sconti in base al numero di prenotazioni attive o a promozioni particolari;
- Creare e modificare un amministratore: l'applicazione non presenta la sezione per la creazione degli amministratori nè per la modifica dei dati come username e password;
- Consentire all'utente che effettua una prenotazione di utilizzare i dati di un passeggero che è già stato inserito nel sistema da una prenotazione precedente mediante, ad esempio, un altro tipo di login alla piattaforma;

- Potrebbe essere di interesse per la compagnia aerea l'inserimento di politiche riguardanti i bagagli dei passeggeri. Ad esempio, potrebbe essere compreso nel prezzo un bagaglio di piccole dimensioni, ma l'utente potrebbe comunque avere necessità di trasportare più di un bagaglio o con dimensioni superiori. In fase di prenotazione, dunque, l'idea potrebbe essere quella di visualizzare all'utente un'opzione che gli permetta di acquistare, per ogni passeggero, il permesso per il trasporto in stiva di bagagli aggiuntivi;
- Consentire ad un utente amministratore di inserire nuove strategie o metodi per il calcolo del prezzo. Un'idea potrebbe essere quindi quella di creare una pagina nella quale l'utente amministratore può inserire dei criteri per l'applicazione della strategia di sconto e i parametri (ad esempio la percentuale, il valore massimo, etc...) sulla base dei quali calcolare lo sconto. Questi verranno persistiti su database per poi essere recuperati ed interpretati al momento della chiamata di un metodo in grado di decidere lo sconto da applicare.

Appendice A

Controllers

A.1 SearchFlightsController

Questo controller si occupa di gestire le pagine *Home* e *Flights Result* ed espone i metodi per la ricerca di voli e per la presentazione dei risultati; questo copre il caso d'uso UC4. Gli oggetti che utilizza dovranno quindi avere uno scope che sia in grado di gestire un numero arbitrario di richieste HTTP dipendente dal numero di operazioni che l'utente desidera compiere. Questa classe, proprio per questo motivo, è stata annotata con *@ConversationScoped* che impone quindi di avere al suo interno un oggetto di tipo *Conversation*, iniettato con CDI, e di specificare l'inizio e la fine della conversazione attraverso i metodi *conversation.begin()* e *conversation.end()*; al momento della chiamata del primo metodo è necessario inoltre impostare un tempo massimo (timeout) dopo il quale la conversazione viene interrotta automaticamente perdendo così tutti i dati memorizzati fino a quel momento. La conversazione inizia quindi al momento della ricerca dei voli e viene interrotta al momento dell'avvenuta prenotazione o in caso di timeout scaduto. Questa classe presenta inoltre molti attributi per la ricerca dei voli, per la presentazione dei risultati e per la gestione degli errori, fra cui un *FlightDao*, un *AirportDao* ed un *TemporaryReservationSeatsDao*, tutti con annotazione *@Inject*.

La maggior parte dei metodi di questo controller si occupa semplicemente di estrarre informazioni sui voli servendosi dei metodi esposti dai DAO per effettuare interrogazioni specifiche al DBMS. Nel momento in cui l'utente seleziona la tratta del volo e la data di interesse, viene anzitutto chiamato il metodo *isAvailable()* che, dato un volo, controlla se effettivamente è disponibile verificando il numero di posti non ancora prenotati. Questo metodo viene chiamato a causa del fatto che nel tempo che intercorre tra la ricerca dei voli e l'effettiva conferma della prenotazione da parte dell'utente, un altro utente potrebbe aver iniziato e concluso una prenotazione riservando nuovi posti. Nel caso in cui questo metodo fornisca un risultato positivo viene consentito all'utente di proseguire con la prenotazione rimandandolo alla pagina successiva gestita dal controller *InsertReservationController* che avrà il compito di riservare i posti del volo scelto; in caso negativo non viene consentito di proseguire oltre. Questa funzionalità soddisfa il caso d'uso UC2.

A.2 InsertReservationController

Gestisce le pagine *Confirm* e *Summary* ed ha il compito di creare una prenotazione a partire dalle scelte compiute nella pagina iniziale e dai dati inseriti nella pagina di conferma. Presenta attributi atti alla creazione della prenotazione, come i dati dei passeggeri, la strategia scelta per il calcolo del prezzo finale, ed altri, di tipo *ReservationDao*, *FlightDao* e *TemporaryReservationSeatsDao* con annotazione *@Inject*, per la persistenza e l'estrazione di informazioni dal database. Anche questo controller, analogamente a quello descritto precedentemente, presenta l'annotazione CDI *@ConversationScoped* poiché facente sempre parte della conversazione iniziata nel

SearchFlightController.

Quando l'utente passa alla pagina *Confirm*, vengono anzitutto creati e persistiti degli oggetti di tipo *TemporaryReservationSeats* (uno per ogni volo) attraverso il metodo *reserveSeats()* avente annotazione *@Transactional* per poter creare una transizione sul database. Questi oggetti specificano il numero di posti da riservare in modo temporaneo su ciascun volo selezionato e verranno utilizzati nella fase iniziale di ricerca dei voli per mostrare i posti effettivamente disponibili considerando le prenotazioni completate e quelle ancora da concludersi. Successivamente, attraverso il metodo *chooseStrategy()*, viene determinata la strategia di calcolo dello sconto e conseguentemente del prezzo finale. Questa viene gestita dal pattern Strategy come descritto in Figura 3.1 creando un oggetto da associare alla prenotazione e applicando lo sconto relativo.

Nella pagina *Confirm* si dà la possibilità all'utente di inserire tutti i dati necessari alla creazione della prenotazione (e-mail, dati dei passeggeri) e si mostra l'eventuale sconto applicato; questo copre il caso d'uso UC3. Si noti che i dati di eventuali passeggeri che sono stati già inseriti nel sistema attraverso una precedente prenotazione non potranno essere recuperati per una nuova: al momento della conferma l'utente dovrà inserire nuovamente i dati del passeggero poiché al livello organizzativo il passeggero stesso verrà trattato come un nuovo passeggero e quindi non sarà associato ad un passeggero già esistente.

Alla conferma dell'utente viene chiamato il metodo *saveReservation()* (con annotazione *@Transactional*) che provvede a generare il *reservationId* della prenotazione, ad aggiornare il numero di posti disponibili sui voli selezionati ed a rimuovere le prenotazioni temporanee create in precedenza. Attraverso questa procedura si copre il caso d'uso UC1.

Esiste un ulteriore metodo chiamato *cleanTemporaryReservation()*, con annotazioni *@PreDestroy* e *@Transactional*, che consente di eliminare tutte le prenotazioni temporanee scadute, ovvero che sono presenti nel database da più di *maxMinutesTemporaryReservation* minuti (che nel nostro caso coincide con la sessione massima dell'utente). Questo metodo viene utilizzato a causa del fatto che un utente potrebbe chiudere il browser nella pagina di conferma senza cliccare il pulsante di annullamento oppure viene impiegato troppo tempo per confermare la prenotazione, tenendo così occupati dei posti che potrebbero essere di interesse per altri utenti che utilizzano l'applicazione e mantenendo il database inconsistente. L'annotazione *@PreDestroy* viene utilizzata generalmente sui metodi come notifica di callback per segnalare che l'istanza sta per essere rimossa dal container.

A.3 ReservationLoginController

Utilizza la pagina *My Booking Login* ed ha come scopo quello di gestire il login e logout di un utente per l'accesso alla sezione relativa alla visualizzazione, modifica ed eliminazione di una prenotazione. Presenta attributi di tipo *ReservationSessionBean* e *ReservationDao* annotati con *@Inject*, ed un attributo di tipo *Reservation* per contenere tutti i dati della prenotazione qualora il login andasse a buon fine.

Questo controller è annotato con *@Model* ed ha uno scope di tipo *@RequestScoped* in quanto è necessaria una sola richiesta HTTP per l'accesso e l'oggetto dovrà avere un ciclo di vita che comprende tutte le successive richieste HTTP appartenenti alla stessa sessione.

Espone principalmente i metodi:

- *login()*: attraverso il *reservationDao*, verifica se esiste una prenotazione avente *email* e *reservationId* specificati dall'utente: in caso positivo l'oggetto di tipo *Reservation* conterrà i dati della prenotazione specificata e l'utente verrà rimandato alla pagina che mostra i dettagli della prenotazione (gestita dal *ManageReservationController*) e che consente di effettuarne le operazioni precedentemente menzionate; in caso negativo verrà restituito un messaggio di errore all'utente, che lo inviterà a riprovare nuovamente rimanendo quindi nella stessa pagina; questo copre il caso d'uso UC5;

- *logout()*: questo metodo imposta semplicemente il parametro *id* della reservation al valore *null* e reindirizza l'utente alla home page chiudendo la sessione, soddisfacendo quindi il caso d'uso UC10.

A.4 ManageReservationController

Questo controller si occupa della gestione delle pagine che mostrano il riepilogo delle prenotazioni, la modifica dei dati dei passeggeri e la cancellazione delle prenotazioni. La classe è stata annotata con *@SessionScoped*. Presenta gli attributi di tipo *ReservationDao*, *FlightDao* e *ReservationSessionBean* annotati con *@Inject* ed altri attributi necessari a contenere i dettagli della prenotazione e dei passeggeri.

Espone i metodi:

- *updatePassengers()*: viene chiamato in caso di aggiornamento dei dati dei passeggeri attraverso l'apposita pagina JSF; il secondo viene chiamato per cancellare la prenotazione;
- *deleteReservation()*: consente di rendere nuovamente disponibili i posti a sedere sui voli prenotati pari al numero di passeggeri della prenotazione e successivamente viene eliminata la prenotazione stessa dal sistema.

Questo controller consente di coprire i casi d'uso UC6, UC7, UC8 e UC9.

A.5 AdministratorLoginController

Questo controller è del tutto simile a *ReservationLoginController*, salvo per il fatto che si occupa di gestire, attraverso la pagina *Administrator Login*, il login ed il logout di un utente amministratore per l'accesso alla sezione riservata. Utilizza quindi attributi di tipo *AdministratorSessionBean* e *AdministratorDao* annotati con *@Inject*, ed un attributo di tipo *Administrator* per contenere i dati dell'amministratore che intende effettuare l'accesso. La classe è stata annotata con *@Model*. Espone i metodi:

- *login()*: attraverso l'*administratorDao*, verifica se username e password inseriti dall'utente sono presenti all'interno del database: in caso positivo l'attributo di tipo *Administrator* conterrà i dati dell'amministratore che ha appena effettuato l'accesso e l'utente verrà rimandato alla pagina *Insert Flights* (gestita dal *InsertFlightController*); in caso negativo verrà restituito un messaggio di errore all'utente e rimarrà nella pagina di login; questo copre il caso d'uso UC11;
- *logout()*: questo metodo si occupa di impostare l'attributo *id* dell'amministratore al valore *null* e reindirizzare l'utente alla home page, soddisfacendo il caso d'uso UC15.

A.6 InsertFlightController

Gestisce le pagine *Insert Flights* e *Show Flights* per l'inserimento di nuovi voli e la visualizzazione dei voli presenti nel sistema. La classe è stata annotata con *@SessionScoped*.

Presenta diversi attributi atti a svolgere le operazioni descritte, fra cui un *FlightDao* ed un *AirportDao* entrambi annotati con *@Inject*.

Espone tre metodi principali:

- *searchFlights()*: è utilizzato nella pagina *Show Flights* per la ricerca di voli, a partire da una data specificata nella tratta selezionata dall'utente; l'utente è in grado di selezionare anche la voce *any* sia come aeroporto di partenza, sia come aeroporto di destinazione: in tal caso verranno mostrati i voli rispettivamente da e verso qualsiasi aeroporto presente nel database;

- *saveFlights()*: presenta l'annotazione *@Transactional* ed è utilizzato dalla pagina *Insert Flights* per l'inserimento di un nuovo volo. Gli aeroporti di partenza e destinazione del volo potranno essere scelti dall'elenco di quelli disponibili o inserire un nuovo aeroporto. Nel secondo caso verrà chiesto all'utente di inserire i dettagli degli aeroporti oltre a quelli del volo: il metodo provvederà quindi a creare e persistere un nuovo aeroporto, prima di effettuare la stessa operazione per il volo. All'utente verranno quindi richiesti tutti i dettagli del volo, come numero del volo, giorno e orario di partenza, durata, etc., prima che il metodo effettui la persistenza su database dell'oggetto di tipo *Flight*;
- *deleteFlight()*: presenta sempre l'annotazione *@Transactional* e permette di eliminare un volo purché non vi siano delle prenotazioni associate ovvero con attributo *reservedSeats* pari a zero.

Questi tre metodi consentono di coprire rispettivamente i casi d'uso UC14, UC12 e UC13.

Appendice B

Funzioni JavaScript

Di seguito vengono descritte le funzioni JavaScript più importanti utilizzate all'interno dell'applicazione e in particolare all'interno delle pagine di presentazione.

- **insert-flights.xhtml**

- *checkData()*: ha il compito di controllare il corretto inserimento dei dati di un volo da parte dall'amministratore, con particolare attenzione al riempimento di tutti i campi presenti, al formato dell'orario di partenza del volo (*HH:mm*), la non uguaglianza degli aeroporti di partenza e destinazione ed il riempimento corretto di tutti i campi qualora venga inserito un nuovo aeroporto (e.g. il codice postale *ZIP* deve essere un numero intero);
- *showHideSource()* e *showHideDestination()*: sono semplici funzioni che hanno il solo scopo di mostrare o nascondere i campi del nuovo aeroporto nelle sezioni corrispondenti qualora vengano selezionate rispettivamente le voci “*new*” o “*existing*” nella scelta degli aeroporti della tratta.

- **index.xhtml**

- *showHide()*: analogamente alle funzioni precedenti ha il solo scopo di mostrare o nascondere la data di ritorno del viaggio per consentire anche la scelta del volo di ritorno nel caso in cui vengano selezionate rispettivamente le opzioni “*return*” o “*one way*”;
- *setMinDateForBack()*: ha il compito di disabilitare tutte le date del volo di ritorno (se selezionato) antecedenti alla data selezionata per il volo di andata; questa operazione viene svolta per consentire la scelta di soli voli di ritorno successivi alla data di partenza;
- *disableDays()* e *disableDaysBack()*: sono due metodi uguali che hanno il compito di disabilitare tutte le date per le quali non esistono voli per la tratta scelta, rispettivamente per le date di andata e di ritorno (se selezionato).

- **flights-result.xhtml**

- *checkSelected()*: ha il ruolo di controllare che l'utente selezioni almeno un volo di andata ed eventualmente uno di ritorno (se è stato selezionato), qualora ne esista almeno uno disponibile nell'elenco. Se non fosse disponibile nessun volo di andata ma almeno uno di ritorno, il sistema consentirà di procedere con la prenotazione considerando il volo di ritorno come un volo di andata con gli aeroporti invertiti e data uguale a quella che era stata scelta per il ritorno.

- **confirm.xhtml**

- *validateEmail()*: ha il compito di controllare il corretto inserimento dell'indirizzo e-mail scelto per la prenotazione seguendo il formato standard *a@b.cd* e mostrando un messaggio di errore in caso di e-mail non valida;
- *checkData()*: controlla il corretto inserimento di tutti i dati dei passeggeri all'interno della pagina con particolare attenzione alla formato corretto per la data di nascita (*dd/mm/yy*).

Bibliografia

- [1] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.
- [2] C. Bauer and G. King, *Java Persistence with Hibernate*. Dreamtech Press, 2006.
- [3] S. M. I. 2007-08-02", "Core J2EE Patterns - Data Access Objects." <http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>. [Online; accessed 10-September-2017].
- [4] O. Corporation", "JavaServer Faces Technology." <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>.
- [5] O. Corporation", "JSF 2.2 View Declaration Language: Facelets Variant." <https://docs.oracle.com/javaee/7/javaserver-faces-2-2/vlddocs-facelets/toc.htm>.
- [6] O. Corporation", "Expression Language." <https://docs.oracle.com/javaee/7/tutorial/jsf-el.htm>.
- [7] "Milligram." <http://milligram.io/>.
- [8] "Bootstrap." <http://getbootstrap.com/>.