

# PROGETTO BASI DI DATI - SETTEMBRE 2020

Gruppo:

M&M

Opzione:

compagnia aerea

Componenti:

Melania Gottardo 874240

Mario Coci 876422

DBMS:

MySQL

Versione di python:

3.7.8

## INDICE

---

- 0 Premessa iniziale
- 1 Esposizione di cos'è il progetto con tutte le idee e le funzionalità
- 2 Tutte le diverse schermate presenti nella web app
- 3 Progettazione concettuale e logica dello schema della base di dati e i ruoli
- 4 Scelte progettuali:
  - a. Vincoli
  - b. Trigger
  - c. Transazioni
- 5 Porzioni di codice interessante e tecnologie specifiche:
  - a. SQL
  - b. Flask
  - c. Html
  - d. Javascript
- 6 Visualizzazione immagini
- 7 Divisione dei compiti nel progetto
- 8 Futuro della web app

## 0 – PREMESSA

---

Per la realizzazione di questo progetto, l'applicazione è stata sviluppata con Visual Studio 2019 in Flask, utilizzando SQLAlchemy Core (Textual SQL) per interfacciarsi con il DBMS sottostante (MySQL). Le pagine per la web app sono state sviluppate a parte in html (con l'ausilio di css e js dove necessario) e poi renderizzate da Flask.

Il tema scelto per il progetto è: compagnia aerea.

La grafica delle pagine html è stata realizzata prendendo spunto da dei template di Bootstrap (<https://getbootstrap.com/>), infatti nelle pagine html ci sono dei link e riferimenti a file .css e .js presi da Bootstrap.

La grafica della pagina che mostra la mappa dei sedili, invece, è stata realizzata sulla base del lavoro di omarmahili (<https://github.com/omarmahili/seatchart.js>) e modificata nelle parti interessate.

Le immagini utilizzate sono tutte senza copyright e sono state scaricate da un sito di stock foto (<https://pixabay.com/it/>).

Prima di far avviare l'applicazione, è necessario eseguire alcune procedure:

1. Nella cartella del progetto è presente una cartella chiamata 'MySQL\_import' da importare in MySQL per poter ottenere le tabelle con i dati.
2. In MySQL, aprire l'editor di testo, incollare ed eseguire il contenuto del file 'DataBase\_aggiungere.sql' per ottenere i ruoli, i permessi e i trigger.
3. Sempre nell'editor di MySQL, incollare ed eseguire il contenuto del file 'DataBase\_utenti.sql' per ottenere gli utenti con assegnati i ruoli.
4. Nella sezione 'Users and Privileges' di MySQL, cliccare sull'utente "anonimo" e su 'Administrative Roles' spuntare "MaintenanceAdmin", "ProcessAdmin", "UserAdmin" e "SELECT", quindi premere 'Apply'.
5. Sempre nella sezione 'Users and Privileges', cliccare sul ruolo "Operatore" e su 'Administrative Roles' spuntare "UserAdmin" e "SecurityAdmin", quindi premere 'Apply'.

Ora è possibile avviare l'applicazione web.

## 1 – IL PROGETTO: IDEE E FUNZIONALITÀ

---

Il progetto si basa sulla realizzazione di un'applicazione web per una compagnia aerea. Il nome della compagnia è "Tappeto volante".

L'idea di base era realizzare un sito dove si avessero due possibilità principali: prenotare dei voli e registrare dei voli. Sono stati individuati quindi due ruoli: il cliente e l'operatore. Ognuno doveva avere un accesso personalizzato dove l'altro non potesse accedere e dove chiunque non fosse autenticato non potesse accedere. Serviva quindi un'area comune e due aree riservate ben distinte.

L'area comune parte dalla home page dove chiunque può accedere, anche se non autenticato. Da qui si può accedere a sei sezioni diverse del sito:

- L'elenco di tutti i voli ordinati per partenza dalla più imminente;
- L'elenco dei dieci voli più economici;
- L'elenco dei voli che presentano degli sconti;
- L'elenco di tutte le mete;
- L'elenco delle mete più richieste;
- L'elenco dei privilegi che caratterizzano le classi.

Questo elenco di possibili sezioni a cui accedere è presente anche nel menù rapido che si trova nella barra di navigazione. Quest'ultima è presente in tutte le pagine e da essa si può accedere anche alla home, alla pagina di login (o fare il logout se si è già autenticati) ed accedere all'area riservata (se si è autenticati). In aggiunta, qui possono apparire anche messaggi di errore per l'utente che tenta di compiere azioni non lecite.

Le pagine che riguardano i voli permettono tutte di accedere direttamente ai dettagli che riguardano un determinato volo. Se prima veniva visualizzato solo il nome delle città di provenienza e di arrivo con il prezzo dell'economy class, qui ci sono i dettagli specifici quali data e orario di partenza e di arrivo, durata del volo, luogo di partenza (città, nazione e aeroporto) e di arrivo e numero di scali. Da questa pagina si può accedere alla pagina di prenotazione dei posti sul volo, ma solo se si è autenticati.

Le pagine che riguardano le mete permettono di visualizzare l'elenco delle città e delle nazioni che sono presenti nel database.

La pagina dedicata alle classi permette di visualizzare quali privilegi sono concessi a chi prenota un posto in una determinata classe rispetto ad un'altra. Le classi sono tre: first class, business class ed economy class.

La pagina di login permette di inserire il proprio ID e la propria password per l'autenticazione e permette di accedere alla pagina di registrazione. In quest'ultima è possibile registrarsi inserendo tutti i dati richiesti nei campi specifici. Non è possibile effettuare più registrazioni con lo stesso ID, di conseguenza, per facilitare l'identificazione, è chiesto di inserire il codice fiscale che è unico per ogni individuo. Quando avviene la registrazione si diventa automaticamente un cliente, per registrarsi come operatore è necessario essere registrati da un altro operatore. Questa scelta è stata fatta per non permettere ad un individuo qualsiasi di poter scegliere se essere un cliente o un operatore per impedire che così facendo accedesse e modificasse i dati del database. Facendo una nuova registrazione si viene poi reindirizzati alla pagina di login per effettuare l'accesso. Una volta effettuato, si viene reindirizzati alla propria area riservata (ora accessibile).

Nell'area riservata, ogni utente (cliente o operatore) ha accesso alla pagina di modifica dei propri dati. Per il resto, le due pagine sono distinte.

Consideriamo ora le operazioni che può svolgere un cliente. Partendo dall'area riservata, il cliente può visualizzare tutte le sue prenotazioni (con data di partenza non superiore alla data di visualizzazione) e accedere o ai dettagli di un volo o ai dati di una prenotazione. Entrando in questa sezione sono disponibili i codici dei posti prenotati, il numero di bagagli a mano assegnati e il numero di bagagli in stiva selezionato. Da questa pagina, il cliente può decidere di tornare alla sua area riservata, cancellare la prenotazione o aggiungere altri bagagli in stiva.

Se il cliente si trova nella pagina dei dettagli di un volo, può andare alla pagina per eseguire una prenotazione relativa a quel volo. In questa pagina si vede la mappa di tutti i posti dell'aereo associato al volo scelto: i posti sono identificati tramite un codice alfanumerico e da un colore. Il colore permette di capire se un posto è selezionato, libero, già prenotato o appartenente ad una classe diversa da quella scelta. Infatti, il cliente prima seleziona la classe in cui desidera effettuare la prenotazione e poi seleziona i

posti che preferisce. Ogni volta che viene selezionato un posto viene anche creato un biglietto nel carrello che specifica il codice alfanumerico del posto e a che classe appartiene. I posti selezionati possono essere rimossi uno alla volta o tutti insieme. Il totale da pagare viene visualizzato sotto al carrello. Sopra al carrello c'è anche lo spazio per andare ad inserire il numero di bagagli in stiva desiderato. Quando l'utente è soddisfatto può cliccare su "Concludi" per effettuare la prenotazione e si ritroverà in una pagina riepilogativa di conferma. In questa pagina sono disponibili i codici dei posti prenotati, il numero di bagagli a mano assegnati, il numero di bagagli in stiva selezionato e il totale pagato applicato lo sconto che era previsto per quel volo (potrebbe anche essere 0%).

Consideriamo ora le operazioni che può fare un operatore. Partendo dall'area riservata, l'operatore può scegliere un'azione tra una serie di azioni quale compiere. Questa pagina serve solo come centro operativo per rimandare l'operatore alla pagina che veramente gli interessa. Le azioni possibili (oltre alla modifica dei propri dati) sono:

- Visualizzare le statistiche della compagnia;
- Registrare un nuovo volo / aereo / meta;
- Modificare le informazioni riguardanti un volo / aereo / meta;
- Cancellare un volo / aereo / meta;
- Inserire un nuovo operatore.

La pagina che permette la visualizzazione delle statistiche a fini commerciali prende le informazioni dal database presenti in quel momento. Si possono avere informazioni generali quali il totale degli incassi della compagnia, il numero di clienti, operatori e aerei della compagnia e il prezzo medio di un biglietto, anche per classe. Poi, ci sono informazioni riguardo al numero e alla percentuale di posti prenotati in una classe rispetto alle altre ed è presente l'elenco di tutti i voli con i rispettivi prezzi (divisi per classe), luoghi di partenza e arrivo, durata, scali ed eventuale sconto.

Le pagine per andare a registrare un volo, un aereo o una meta sono simili a quella per la registrazione di un utente. Si inseriscono i dati relativi a cosa si vuole registrare e si dà la conferma. Per inserire un volo, l'operatore deve selezionare da che luogo partire, in quale luogo arrivare e il numero di posti necessari per l'aereo di quel volo.

Le pagine per andare a modificare o cancellare un volo, un aereo o una meta sono simili: l'operatore seleziona quale volo, aereo o meta gli interessa, poi, nel caso di una modifica, va ad inserire i dati relativi e preme la conferma; invece, se si tratta di una cancellazione, basta che prema il pulsante per la cancellazione.

La pagina che permette di registrare un nuovo operatore funziona alla stessa maniera di quella per registrare un cliente, ma il nuovo utente sarà identificato come un operatore con tutti i permessi che ne derivano e alla fine non si viene reindirizzati alla pagina di login.

Se l'operatore si trova nella pagina dei dettagli di un volo, può andare alla pagina per vedere la mappa dell'aereo relativo a quel volo: i posti sono identificati tramite un codice alfanumerico e da un colore. Il colore permette di capire se un posto è selezionato, libero o prenotato. Ogni volta che viene selezionato un posto viene anche creato un biglietto fittizio nel carrello che specifica il codice alfanumerico del posto. I posti selezionati possono essere rimossi uno alla volta o tutti insieme.

Se un utente non autenticato o non autorizzato prova ad accedere ad una pagina a cui non dovrebbe accedere, gli verrà impedito l'accesso o verrà reindirizzato su un'altra pagina.

## 2 – SCHERMATE

### BARRA DI NAVIGAZIONE – utente non autenticato

The screenshot shows a top navigation bar with links: Tappeto volante, Home, Pianifica ▾, Area riservata, and Login. A vertical dropdown menu is open under the 'Pianifica' link, showing options: Voli, Voli low cost, Offerte, Mete, Mete gettonate, and Classi e privilegi.

### BARRA DI NAVIGAZIONE – cliente / operatore

The screenshot shows a top navigation bar with links: Tappeto volante, Home, Pianifica ▾, Area riservata, and Logout. The 'Logout' link is highlighted.

### BARRA DI NAVIGAZIONE – esempio di messaggio di errore

The screenshot shows a top navigation bar with links: Tappeto volante, Home, Pianifica ▾, Area riservata, and Login. A red error message box contains the text: "Errore: inserire un ID diverso".

### HOME PAGE (CAROUSEL 1)

The screenshot shows a large banner with a sunset over water and rocks. Overlaid text reads: "Vola con noi" and "Con noi viaggerai comodamente e senza pensieri. Avrai la testa tra le nuvole.". A blue button says "Vedi le offerte". The top navigation bar is visible.

#### Voli low cost

Sai che oggi e' davvero il tuo giorno fortunato?  
Scopri dove puoi volare praticamente gratis.

[Voli low cost](#)

#### Mete preferite

Le mete piu' gettonate di quest'anno sono qui  
per te! Le vostre mete preferite sono qui.

[Mete gettonate](#)

#### Classi e privilegi

A seconda della classe che sceglierai, avrai diversi  
comfort e vantaggi a disposizione.

[Classi e privilegi](#)

### CAROUSEL 2 e 3

The screenshot shows a banner with a tropical island and turquoise water. Overlaid text reads: "Prenota un viaggio intorno al mondo in due semplici click" and "Cerca il volo piu' adatto a te e alle tue esigenze. Seleziona i posti e i bagagli.". A blue button says "Prenota subito".

#### Visita le mete piu' belle

Avrai un'ampia gamma di mete, tra cui scegliere per la tua prossima avventura in giro per il mondo.

[Visualizza mete](#)

## VOLI

Tappeto volante Home Pianifica ▾ Area riservata Login

**Tutti i voli**  
Scorri per trovare il volo che stai cercando.

**Milano - Amsterdam** [Vedi dettagli](#) Da 30.0 euro

**Barcellona - San Pietroburgo** [Vedi dettagli](#) Da 70.0 euro

**Bordeaux - Dublino** [Vedi dettagli](#) Da 60.0 euro

## VOLI LOW COST

Tappeto volante Home Pianifica ▾ Area riservata Login

**I voli piu' economici**  
Scorri per trovare il volo che stai cercando ad un prezzo minore.

**Milano - Amsterdam** [Vedi dettagli](#) Da 30.0 euro

**Venezia - Istanbul** [Vedi dettagli](#) Da 50.0 euro

**Bordeaux - Dublino** [Vedi dettagli](#) Da 60.0 euro

## OFFERTE

Tappeto volante Home Pianifica ▾ Area riservata Login

**Le nostre offerte**  
Scopri quali sconti stanno aspettando te.

**Londra - Berlino** [Vedi dettagli](#) Sconto del 5%

**Atene - Nizza** [Vedi dettagli](#) Sconto del 5%

**Madrid - San Francisco** [Vedi dettagli](#) Sconto del 10%

## METE

Tappeto volante Home Pianifica ▾ Area riservata

Login

### Le nostre mete

Scorri per scoprire quali mete ti attendono.

#### Sydney

Scopri questa bellissima citta'...



#### Australia

... in questo magnifico paese.



#### Vienna

Scopri questa bellissima citta'...



#### Austria

... in questo magnifico paese.



## METE PREFERITE

Tappeto volante Home Pianifica ▾ Area riservata

Login

### Mete piu' attese

Scorri per scoprire le mete piu' scelte.

#### Dublino

Scopri questa bellissima citta'...



#### Irlanda

... in questo magnifico paese.



## CLASSI E PRIVILEGI

Tappeto volante Home Pianifica ▾ Area riservata

Login

## Classi

Scopri quale classe si adatta di piu' alle tue esigenze.

First	Business	Economy
<input checked="" type="checkbox"/> Wi-fi gratuito <input checked="" type="checkbox"/> Schermo per i film <input checked="" type="checkbox"/> Pasto gratuito <input checked="" type="checkbox"/> Presa di corrente	<input checked="" type="checkbox"/> Wi-fi gratuito <input checked="" type="checkbox"/> Schermo per i film <input type="checkbox"/> Pasto gratuito <input type="checkbox"/> Presa di corrente	<input type="checkbox"/> Wi-fi gratuito <input type="checkbox"/> Schermo per i film <input type="checkbox"/> Pasto gratuito <input type="checkbox"/> Presa di corrente

## DETTAGLI

Tappeto volante Home Pianifica ▾ Area riservata

Logout

### Milano - Amsterdam

Qui sotto sono elencate le specifiche relative a questo volo.

Cerca posti

Partenza	Arrivo	Durata viaggio
2020-09-07 14:38:00	2020-09-07 16:43:00	2h 5m
Partenza da	Arrivo a	Scali
Aeroporto di Milano-Malpensa Milano - Italia	Aeroporto di Amsterdam-Schiphol Amsterdam - Olanda	Numero scali: 0

## LOGIN

Tappeto volante Home Pianifica ▾ Area riservata

Login

Accedi

ID
Password

**Login**

**Registrati**

## REGISTRAZIONE

Tappeto volante Home Pianifica ▾ Area riservata Login

### Registrati

Nome utente  
Codice fiscale - ID  
Indirizzo email  
Password

Fatto

## MODIFICA PROFILO – cliente / operatore

Tappeto volante Home Pianifica ▾ Area riservata Logout

Inserisci i tuoi nuovi dati

Nome utente  
Indirizzo email

Modifica

## POSTI – cliente

Tappeto volante Home Pianifica ▾ Area riservata Logout

### Posti

Selezione i posti che desideri prenotare per il volo Milano - Amsterdam.

Classe ▾

- Classe ▾
- First class
  - Business class
  - Economy class

Cabina						
	A	B	C	D	E	F
1		1B	1C	1D	1E	
2	2A	2B	2C	2D	2E	2F
3	3A	3B	3C	3D	3E	3F
4	4A	4B	4C	4D	4E	4F
5	5A	5B	5C	5D	5E	5F
6	6A	6B	6C	6D	6E	6F

#### Legenda

- First euro 150.00
- Disponibile
- Prenotato
- Classe diversa

Numero bagagli in stiva

Carrello (0)

Cabina						Cabina						Cabina						Cabina						Cabina										
	A	B	C	D	E	F		A	B	C	D	E	F		A	B	C	D	E	F		A	B	C	D	E	F		A	B	C	D	E	F
1	1A	1B	1C	1D	1E	1F	2	2A	2B	2C	2D	2E	2F	3	3A	3B	3C	3D	3E	3F	4	4A	4B	4C	4D	4E	4F	5	5A	5B	5C	5D	5E	5F
4	4A	4B	4C	4D	4E	4F	5	5A	5B	5C	5D	5E	5F	6	6A	6B	6C	6D	6E	6F	7	7A	7B	7C	7D	7E	7F	8	8A	8B	8C	8D	8E	8F
9	9A	9B	9C	9D	9E	9F	10	10A	10B	10C	10D	10E	10F	11	11A	11B	11C	11D	11E	11F	12	12A	12B	12C	12D	12E	12F	13	13A	13B	13C	13D	13E	13F
14	14A	14B	14C	14D	14E	14F	15	15A	15B	15C	15D	15E	15F	16	16A	16B	16C	16D	16E	16F	17	17A	17B	17C	17D	17E	17F	18	18A	18B	18C	18D	18E	18F
19	19A	19B	19C	19D	19E	19F	20	20A	20B	20C	20D	20E	20F	21	21A	21B	21C	21D	21E	21F	22	22A	22B	22C	22D	22E	22F	23	23A	23B	23C	23D	23E	23F
24	24A	24B	24C	24D	24E	24F	25	25A	25B	25C	25D	25E	25F	26	26A	26B	26C	26D	26E	26F	27	27A	27B	27C	27D	27E	27F	28	28A	28B	28C	28D	28E	28F
29	29A	29B	29C	29D	29E	29F	30	30A	30B	30C	30D	30E	30F	31	31A	31B	31C	31D	31E	31F	32	32A	32B	32C	32D	32E	32F	33	33A	33B	33C	33D	33E	33F
34	34A	34B	34C	34D	34E	34F	35	35A	35B	35C	35D	35E	35F	36	36A	36B	36C	36D	36E	36F	37	37A	37B	37C	37D	37E	37F	38	38A	38B	38C	38D	38E	38F
39	39A	39B	39C	39D	39E	39F	40	40A	40B	40C	40D	40E	40F	41	41A	41B	41C	41D	41E	41F	42	42A	42B	42C	42D	42E	42F	43	43A	43B	43C	43D	43E	43F
44	44A	44B	44C	44D	44E	44F	45	45A	45B	45C	45D	45E	45F	46	46A	46B	46C	46D	46E	46F	47	47A	47B	47C	47D	47E	47F	48	48A	48B	48C	48D	48E	48F
49	49A	49B	49C	49D	49E	49F	50	50A	50B	50C	50D	50E	50F	51	51A	51B	51C	51D	51E	51F	52	52A	52B	52C	52D	52E	52F	53	53A	53B	53C	53D	53E	53F
54	54A	54B	54C	54D	54E	54F	55	55A	55B	55C	55D	55E	55F	56	56A	56B	56C	56D	56E	56F	57	57A	57B	57C	57D	57E	57F	58	58A	58B	58C	58D	58E	58F
59	59A	59B	59C	59D	59E	59F	60	60A	60B	60C	60D	60E	60F	61	61A	61B	61C	61D	61E	61F	62	62A	62B	62C	62D	62E	62F	63	63A	63B	63C	63D	63E	63F
64	64A	64B	64C	64D	64E	64F	65	65A	65B	65C	65D	65E	65F	66	66A	66B	66C	66D	66E	66F	67	67A	67B	67C	67D	67E	67F	68	68A	68B	68C	68D	68E	68F
69	69A	69B	69C	69D	69E	69F	70	70A	70B	70C	70D	70E	70F	71	71A	71B	71C	71D	71E	71F	72	72A	72B	72C	72D	72E	72F	73	73A	73B	73C	73D	73E	73F
74	74A	74B	74C	74D	74E	74F	75	75A	75B	75C	75D	75E	75F	76	76A	76B	76C	76D	76E	76F	77	77A	77B	77C	77D	77E	77F	78	78A	78B	78C	78D	78E	78F
79	79A	79B	79C	79D	79E	79F	80	80A	80B	80C	80D	80E	80F	81	81A	81B	81C	81D	81E	81F	82	82A	82B	82C	82D	82E	82F	83	83A	83B	83C	83D	83E	83F
84	84A	84B	84C	84D	84E	84F	85	85A	85B	85C	85D	85E	85F	86	86A	86B	86C	86D	86E	86F	87	87A	87B	87C	87D	87E	87F	88	88A	88B	88C	88D	88E	88F
89	89A	89B	89C	89D	89E	89F	90	90A	90B	90C	90D	90E	90F	91	91A	91B	91C	91D	91E	91F	92	92A	92B	92C	92D	92E	92F	93	93A	93B	93C	93D	93E	93F
94	94A	94B	94C	94D	94E	94F	95	95A	95B	95C	95D	95E	95F	96	96A	96B	96C	96D	96E	96F	97	97A	97B	97C	97D	97E	97F	98	98A	98B	98C	98D	98E	98F
99	99A	99B	99C	99D	99E	99F	100	100A	100B	100C	100D	100E	100F	101	101A	101B	101C	101D	101E	101F	102	102A	102B	102C	102D	102E	102F	103	103A	103B	103C	103D	103E	103F
104	104A	104B	104C	104D	104E	104F	105	105A	105B	105C	105D	105E	105F	106	106A	106B	106C	106D	106E	106F	107	107A	107B	107C	107D	107E	107F	108	108A	108B	108C	108D	108E	108F
109	109A	109B	109C	109D	109E	109F	110	110A	110B	110C	110D	110E	110F	111	111A	111B	111C	111D	111E	111F	112	112A	112B	112C	112D	112E	112F	113	113A	113B	113C	113D	113E	113F
114	114A	114B	114C	114D	114E	114F	115	115A	115B	115C	115D	115E	115F	116	116A	116B	116C	116D	116E	116F	117	117A	117B	117C	117D	117E	117F	118	118A	118B	118C	118D	118E	118F
119	119A	119B	119C	119D	119E	119F	120	120A	120B	120C	120D	120E	120F	121	121A	121B	121C	121D	121E	121F	122	122A	122B	122C	122D	122E	122F	123	123A	123B	123C	123D	123E	123F
124	124A	124B	124C	124D	124E	124F	125	125A	125B	125C	125D	125E	125F	126	126A	126B	126C	126D	126E	126F	127	127A	127B	127C	127D	127E	127F	128	128A	128B	128C	128D	128E	128F
129	129A	129B	129C	129D	129E	129F	130	130A	130B	130C	130D	130E	130F	131	131A	131B	131C	131D	131E	131F	132	132A	132B	132C	132D	132E	132F	133	133A	133B	133C	133D	133E	133F
134	134A	134B	134C	134D	134E	134F	135	135A	135B	135C	135D	135E	135F	136	136A	136B	136C	136D	136E	136F	137	137A	137B	137C	137D	137E	137F	138	138A	138B	138C	138D	138E	138F
139	139A	139B	139C	139D	139E	139F	140	140A	140B	140C	140D	140E	140F	141	141A	141B	141C	141D	141E	141F	142	142A	142B	142C	142D	142E	142F	143	143A	143B	143C	143D	143E	143F
144	144A	144B	144C	144D	144E	144F	145	145A	145B	145C	145D	145E	145F	146	146A	146B	146C	146D	146E	146F	147	147A	147B	147C	147D	147E	147F	148	148A	148B	148C	148D	148E	148F
149	149A	149B	149C	149D	149E	149F	150	150A	150B	150C	150D	150E	150F	151	151A	151B	151C	151D	151E	151F	152	152A	152B	152C	152D	152E	152F	153	153A	153B	153C	153D	153E	153F
154	154A	154B	154C	154D	154E	154F	155	155A	155B	155C	155D	155E	155F	156	156A	156B	156C	156D	156E	156F	157	157A	157B	157C	157D	157E	157F	158	158A	158B	158C	158D	158E	158F
159	159A	159B	159C	159D	159E	159F	160	160A	160B	160C	160D	160E	160F	161	161A	161B	161C	161D	161E	161F	162	162A	162B	162C	162D	162E	162F	163	163A	163B	163C	163D	163E	163F
164	164A	164B	164C	164D	164E	164F	165	165A	165B	165C	165D	165E	165F	166	166A	166B	166C	166D	166E	166F	167	167A	167B	167C	167D	167E	167F	168	168A	168B	168C	168D	168E	168F
169	169A	169B	169C	169D	169E	169F	170	170A	170B	170C	170D	170E	170F	171	171A	171B	171C	171D	171E	171F	172	172A	172B	172C	172D	172E	172F	173	173A	173B	173C	173D	173E	173F
174	174A	174B	174C	174D	174E																													

## AREA RISERVATA – cliente

Tappeto volante Home Pianifica ▾ Area riservata Logout

### Le tue prenotazioni

Visualizza le tue prenotazioni.

Modifica profilo

Roma - Tokyo [Dettagli](#) [La tua prenotazione](#)

Barcellona - San Pietroburgo [Dettagli](#) [La tua prenotazione](#)

Sydney - Miami [Dettagli](#) [La tua prenotazione](#)

## DATI PRENOTAZIONE

Tappeto volante Home Pianifica ▾ Area riservata Logout

### La tua prenotazione

Torna nella tua area riservata per vedere tutte le tue prenotazioni.

Area riservata

Posti prenotati	Bagagli a mano	Bagagli in stiva
<input type="checkbox"/> 2A <input type="checkbox"/> 2B <input type="checkbox"/> 2C	Total: 3  Un bagaglio a mano a persona. Il bagaglio starà' nella cappelliera.	Total: 5  Ogni bagaglio in stiva deve pesare al massimo 30kg, altrimenti si pagherà una sanzione per ogni kg in più.

[Cancella prenotazione](#) [Piu' bagagli in stiva](#)

Vuoi aggiungere altri posti? Fai una nuova prenotazione.

## AGGIUNTA BAGAGLI

Tappeto volante Home Pianifica ▾ Area riservata Logout

Aggiungi il numero di bagagli in stiva che ti serve.

3

Aggiungi

## AREA RISERVATA – operatore

Tappeto volante Home Pianifica ▾ Area riservata

Logout

# Buongiorno, operatore

Come desideri procedere?

[Modifica profilo](#)

[Vedi statistiche](#)

[Inserisci una nuova meta](#)

[Inserisci un nuovo volo](#)

[Inserisci un nuovo aereo](#)

[Modifica i dati di una meta](#)

[Modifica un volo](#)

[Modifica i dati di un aereo](#)

[Elimina una meta](#)

[Elimina un volo](#)

[Elimina un aereo](#)

[Inserisci un nuovo operatore](#)

## STATISTICHE

Tappeto volante Home Pianifica ▾ Area riservata

Logout

### Informazioni compagnia

Incassi totali = 344005.25 euro

Clienti totali = 12

Operatori totali = 6

Numero aerei = 37

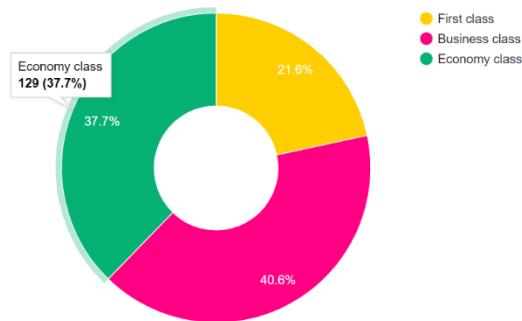
Prezzo medio biglietto = 1251 euro

Prezzo medio biglietto first class = 2084 euro

Prezzo medio biglietto business class = 1254 euro

Prezzo medio biglietto economy class = 416 euro

### Classi scelte



### Informazioni sui voli

Codice volo	First	Business	Economy	Durata	Partenza	Arrivo	Scali	Sconto
117	10000.0 euro	6000.0 euro	2000.0 euro	15h 40m	Napoli	San Francisco	1	25%
110	5000.0 euro	3000.0 euro	1000.0 euro	17h 20m	Nuova Delhi	Shanghai	1	30%
119	4500.0 euro	2700.0 euro	900.0 euro	24h 30m	Sydney	Istanbul	2	0%
115	4000.0 euro	2400.0 euro	800.0 euro	22h 50m	Sydney	Miami	2	0%
106	2875.0 euro	1725.0 euro	575.0 euro	15h 25m	Bangkok	Bruxelles	1	15%
116	2300.0 euro	1380.0 euro	460.0 euro	20h 20m	Las Vegas	Valencia	2	0%
103	2000.0 euro	1200.0 euro	400.0 euro	13h 5m	Vienna	New York	1	25%
101	1875.0 euro	1125.0 euro	375.0 euro	17h 45m	Roma	Tokyo	1	20%
102	1500.0 euro	900.0 euro	300.0 euro	17h 10m	Pechino	Parigi	1	0%
107	1375.0 euro	825.0 euro	275.0 euro	13h 0m	Vancouver	Amsterdam	0	0%
114	1250.0 euro	750.0 euro	250.0 euro	20h 30m	Madrid	San Francisco	1	10%
111	750.0 euro	450.0 euro	150.0 euro	3h 35m	Milano	Mosca	0	0%
108	425.0 euro	255.0 euro	85.0 euro	1h 55m	Londra	Berlino	0	5%
104	350.0 euro	210.0 euro	70.0 euro	4h 15m	Barcellona	San Pietroburgo	0	0%
113	350.0 euro	280.0 euro	70.0 euro	2h 40m	Atene	Nizza	0	5%
120	350.0 euro	210.0 euro	70.0 euro	1h 5m	Roma	Venezia	0	10%
112	300.0 euro	180.0 euro	60.0 euro	1h 55m	Bordeaux	Dublino	0	0%
105	250.0 euro	150.0 euro	50.0 euro	2h 25m	Venezia	Istanbul	0	0%

## REGISTRAZIONE META

Tappeto volante Home Pianifica ▾ Area riservata Logout

Registra una nuova meta

Citta'

Nazione

Nome aeroporto

Fatto

## REGISTRAZIONE VOLO

Tappeto volante Home Pianifica ▾ Area riservata Logout

Registra un nuovo volo

Luogo partenza

108 - Amsterdam - Aeroporto di Am ▾

Luogo arrivo

Data e orario partenza

Data e orario arrivo

Durata volo

Numero posti nell'aereo

Posti: 124 ▾

Numero scali

Prezzo first class

Prezzo business class

Prezzo economy class

Sconto

Fatto

## REGISTRAZIONE AEREO

Tappeto volante Home Pianifica ▾ Area riservata Logout

Registra un nuovo aereo

Tipologia aereo

Numero file

Peso della stiva in kg

Fatto

## MODIFICA META

Tappeto volante Home Pianifica ▾ Area riservata

Logout

Seleziona la meta che vuoi modificare.

Lista mete

Venezia - Aeroporto di Venezia-Marco Polo - Codice: 101 ▾

Nome aeroporto

Modifica

## MODIFICA VOLO

Tappeto volante Home Pianifica ▾ Area riservata

Logout

Seleziona il volo che vuoi modificare.

Lista voli

Roma - Tokyo - Durata: 17h 45m - Codice: 101 ▾

Prezzo first class

Prezzo business class

Prezzo economy class

Sconto

Modifica

## MODIFICA AEREO

Tappeto volante Home Pianifica ▾ Area riservata

Logout

Seleziona l'aereo che vuoi modificare.

Lista aerei

Airbus A220 - Codice: 101 ▾

Numero file da aggiungere

Numero kg in stiva da aggiungere

Modifica

## CANCELLAZIONE META

Tappeto volante Home Pianifica ▾ Area riservata

Logout

Seleziona la meta che vuoi cancellare.

Lista mete

Venezia - Aeroporto di Venezia-Marco Polo - Cc ▾

[Cancella](#)

## CANCELLAZIONE VOLO

Tappeto volante Home Pianifica ▾ Area riservata

Logout

Seleziona il volo che vuoi cancellare.

Lista voli

Roma - Tokyo - Codice: 101 ▾

[Cancella](#)

## CANCELLAZIONE AEREO

Tappeto volante Home Pianifica ▾ Area riservata

Logout

Seleziona l'aereo che vuoi cancellare.

Lista aerei

Airbus A220 - Codice: 101 ▾

[Cancella](#)

## REGISTRAZIONE OPERATORE

The screenshot shows a user interface for registering a new operator. At the top, there is a navigation bar with links: 'Tappeto volante', 'Home', 'Pianifica', and 'Area riservata'. On the right side of the header is a 'Logout' link. The main content area has a title 'Registra un nuovo operatore'. Below the title are four input fields: 'Nome utente', 'Codice fiscale - ID', 'Indirizzo email', and 'Password'. A large blue button labeled 'Fatto' is positioned below these fields.

## 3 – SCHEMA DELLA BASE DI DATI E RUOLI

---

Alla luce di quanto detto al punto 1 e mostrato al punto 2, adesso presentiamo lo schema della base di dati e spiegheremo il perché di queste scelte di progettazione.

Le tabelle in totale sono sette e servono per gestire: utenti, voli, prenotazioni, posti a sedere, aerei, località e classi.

Innanzitutto, per gestire gli utenti c’è un’unica tabella con una colonna “Ruolo” che va, appunto, ad identificare quale ruolo appartiene ad un utente. Questa colonna permette di gestire l’autorizzazione più facilmente all’interno delle funzioni di Flask, ma non va a sostituire i ruoli veri e propri che sono poi assegnati agli utenti. Quindi abbiamo una tabella per tutti gli utenti: clienti, operatori e l’utente anonimo.

La tabella dei voli serve a mantenere tutte le informazioni riguardanti ogni singolo volo ed è il perno, oltre alla tabella prenotazioni, dello schema. Infatti, nella tabella voli ci sono le chiavi per la tabella degli aerei e quella delle località. Queste due tabelle potrebbero essere inglobate nella tabella voli? La risposta è no perché più voli potrebbero utilizzare lo stesso aereo (in momenti diversi) e quindi ci sarebbero una serie di dati che andrebbero a ripetersi, come in numero di file nell’aereo, il modello e il peso della stiva. Per evitare questo problema basta andare a creare una tabella aerei che salva tutte le informazioni riguardanti un certo aereo e il volo associato manterrà il codice dell’aereo. La stessa cosa vale per la tabella località: è molto facile che ci siano più voli che partono e arrivano nelle stesse città, quindi è bene che tutte le informazioni di una località (come il nome della città, della nazione e dell’aeroporto) stiano in una tabella separata e che il volo che parte o arriva in una certa località ne mantenga il codice. In più pensiamo al caso in cui si dovesse andare ad aggiornare un dato di un aereo o di una località: bisognerebbe andare ad aggiornare quel dato in tutte le righe della tabella voli, mentre così, basterà andare a modificarlo una volta sola.

Come detto sopra, l'altra tabella perno dello schema è la tabella delle prenotazioni, in quanto da questa tabella possiamo sapere il volo a cui fa riferimento, chi ha effettuato la prenotazione e quali posti sono associati alla prenotazione. Questa tabella è necessaria? Assolutamente sì, perché serve tenere traccia di chi ha prenotato cosa, quando e in che quantità; non ci sarebbe altro modo. La tabella per i posti è necessaria o si poteva inglobare nella tabella delle prenotazioni? No, non si poteva inglobare nella tabella delle prenotazioni perché altrimenti per ogni prenotazione non ci sarebbe stata una sola riga, ma molte righe, per la precisione, una per posto e tutte con a fianco una serie di dati generali sulla prenotazione ripetuti x volte. Vogliamo evitare ripetizioni che si possono evitare, quindi usiamo una tabella esterna per i posti con qualche informazione utile, come la classe a cui appartengono.

Resta la tabella delle classi che contiene tutte le informazioni sulle peculiarità di ogni classe. I posti fanno riferimento a questa tabella per avere tutte le informazioni sulla classe a cui appartengono. Ancora una volta, è bene avere una tabella esterna che mantenga informazioni altrimenti ripetute, anche nel caso si debbano fare delle modifiche.

Tutte le tabelle sono interconnesse, ma non ci sono connessioni che potrebbero essere rimosse.

Qui sotto è riportato lo schema della base di dati con le varie connessioni.



## RUOLI

Parliamo ora di ruoli. Anche se abbiamo parlato solo di "Cliente" e "Operatore" esiste un terzo ruolo, l'"Anonimo", quindi i ruoli totali che accedono al sito e che sono presenti sono tre.

Come spiegato precedentemente, c'è una sola tabella che contiene tutti gli utenti e c'è una colonna che identifica se un utente è un cliente, un operatore oppure se è anonimo.

L'utente anonimo è uno soltanto ed è quello con cui si "accede" al sito quando non si è autenticati. È l'unico che ha il ruolo di "Anonimo".

Gli utenti che hanno il ruolo di "Cliente" sono gli effettivi clienti del sito, quelli che possono effettuare le prenotazioni e che non hanno la possibilità di toccare con mano i dati del database o di vedere le statistiche. Possono gestire il proprio profilo e proprie prenotazioni, per il resto sono come un utente anonimo.

Gli utenti che hanno il ruolo di "Operatore" sono gli effettivi operatori del sito. Loro possono gestire tutti i dati e possono compiere tutte le azioni disponibili sul sito, ad eccezione di effettuare prenotazioni. Hanno accesso diretto ai dati su voli, località e aerei, possono vedere le statistiche, possono registrare altri operatori e possono vedere le prenotazioni dei clienti.

Ecco riportati nello specifico i permessi garantiti ad ogni ruolo.

### ANONIMO

```
CREATE ROLE Anonimo;
GRANT INSERT ON users TO Anonimo;
```

### OPERATORE

```
CREATE ROLE Operatore;
GRANT SELECT ON * TO Operatore;
GRANT SELECT ON voli TO Operatore;
GRANT SELECT ON aerei TO Operatore;
GRANT SELECT ON localita TO Operatore;
GRANT INSERT ON users TO Operatore;
GRANT INSERT ON aerei TO Operatore;
GRANT INSERT ON localita TO Operatore;
GRANT INSERT ON voli TO Operatore;
GRANT DELETE ON aerei TO Operatore;
GRANT DELETE ON localita TO Operatore;
GRANT DELETE ON voli TO Operatore;
GRANT UPDATE ON aerei TO Operatore;
GRANT UPDATE ON localita TO Operatore;
GRANT UPDATE ON voli TO Operatore;
```

### CLIENTE

```
CREATE ROLE Cliente;
GRANT INSERT ON prenotazioni TO Cliente;
GRANT INSERT ON posti TO Cliente;
GRANT SELECT ON users TO Cliente;
GRANT SELECT ON aerei TO Cliente;
GRANT SELECT ON prenotazioni TO Cliente;
GRANT SELECT ON voli TO Cliente;
GRANT SELECT ON localita TO Cliente;
GRANT SELECT ON posti TO Cliente;
GRANT SELECT ON classi TO Cliente;
GRANT DELETE ON prenotazioni TO Cliente;
GRANT DELETE ON users TO Cliente;
GRANT UPDATE ON prenotazioni TO Cliente;
GRANT UPDATE ON users TO Cliente;
```

## 4 – SCELTE PROGETTUALI

---

### VINCOLI

In questa sezione si farà riferimento al grafico mostrato al punto 3.

#### AUTOINCREMENT

Le tabelle aerei, voli, localita e prenotazioni hanno la primary key settata su autoincrement perché, a differenza di un utente che può essere identificato con un codice (come il codice fiscale), qui bisognerebbe che l'utente decida un codice che non c'è ancora nella tabella interessata, quindi dovrebbe avere il permesso di vedere tutta la tabella e andare a verificare che il codice da lui scelto non sia già presente. Questo non è un buon metodo per scegliere un codice identificativo di dati (anche protetti, privati o riservati), meglio optare per una soluzione alternativa, dove il codice viene creato automaticamente.

#### NOT NULL

Ci sono alcune righe delle tabelle che sono segnate come NN, ovvero Not Null, infatti, quelle righe non ammettono che quel determinato valore non venga inserito. In particolare, nella tabella users sono tutte le righe, nella tabella prenotazioni sono tutte le righe meno ‘Bagagli’, nella tabella posti sono ‘CodPosti’ e ‘CodPrenotazioni’, nella tabella classi è ‘CodClassi’, nella tabella voli sono tutte le righe eccetto ‘Partenza’, ‘Arrivo’, ‘Durata’ e ‘NumScali’, nella tabella localita sono tutte le righe e nella tabella aerei sono tutte le righe. In ogni caso, per questa web app, non è possibile ottenere un caso in cui uno di questi campi non venga riempito.

#### PRIMARY KEY

Tutte le tabelle presentano un codice identificativo come primary key, ad eccezione della tabella posti: qui avere un codice per ogni posto non aveva molto senso, perché un posto ha il suo codice alfanumerico per essere identificato tra i posti di uno stesso volo, ma non tra diversi aerei e neanche tra diversi voli che usano lo stesso aereo. Al posto di inserire nella tabella tutti i posti presenti in ogni aereo per ogni volo, abbiamo optato per creare una tabella che salvasse solo i posti prenotati e li salvasse con una coppia come primary key: il posto con il suo codice alfanumerico e il codice della prenotazione a cui appartiene. Non sarebbe servito salvare sia i posti vuoti che quelli occupati, così inseriamo meno dati, ma più significativi.

#### FOREIGN KEY

Il posizionamento delle foreign key è banale, come si evince dal grafico. Tra la tabella aerei e la tabella voli c'è una chiave da voli ad aerei perché un aereo può essere usato in più voli, ma un volo usa un aereo. Tra la tabella localita e la tabella voli ci sono due chiavi da voli a localita perché (sia per la partenza che per il ritorno) una località ospita più voli, ma un volo parte e arriva in due soli posti. Tra la tabella prenotazioni e la tabella voli c'è una chiave da prenotazioni a voli perché un volo può essere soggetto a più prenotazioni, ma la prenotazione è di un volo soltanto. Tra la tabella prenotazioni e la tabella users c'è una chiave da prenotazioni ad users perché un utente può avere più prenotazioni, ma una prenotazione appartiene ad un solo utente. Tra la tabella prenotazioni e la tabella posti c'è una chiave da posti a prenotazioni perché una prenotazione può includere più posti, ma un posto può appartenere ad una sola prenotazione. Tra la tabella posti e la tabella classi c'è una chiave da posti a classi perché una classe contiene più posti, ma un posto appartiene ad una sola classe.

Ogni foreign key è impostata su ‘ON UPDATE CASCADE’ e ‘ON DELETE CASCADE’.

#### CHECK

Non ci sono check su attributi, né su tuple.

## TRIGGER

Nel nostro progetto sono presenti tre trigger; due di questi sono incentrati sul numero di bagagli in stiva in relazione al peso delle stive degli aerei, l'altro si riferisce all'aggiornamento dei dati degli utenti.

Cominciamo dai trigger sui bagagli. La scelta è stata dettata dal fatto che, mentre il numero di bagagli a mano è semplicemente calcolato sul numero di posti prenotati, il numero di bagagli in sitiva è a discrezione del cliente. Quando un cliente effettua una prenotazione, digita il numero di bagagli in stiva di cui necessita, ma non c'è un limite al numero di bagagli che può effettivamente inserire, quindi, inverosimilmente potrebbe inserire un milione di bagagli. Un esempio più plausibile è che tutti i posti dell'aereo, o quasi, siano stati prenotati e che il numero massimo di bagagli in stiva sia stato raggiunto; se un cliente dovesse chiedere di aggiungere bagagli in stiva alla sua prenotazione, questo chiaramente non sarebbe ammissibile perché la stiva di un aereo ha una certa capacità.

Per andare a controllare che il numero di bagagli in stiva rispetti il limite della capienza della stiva dell'aereo associato al volo selezionato, ci affidiamo ad un trigger, o meglio a due trigger. Il primo controlla l'inserimento di una prenotazione, mentre il secondo ne controlla l'aggiornamento. In entrambi i casi viene calcolato il numero di bagagli già presenti in stiva e il numero di bagagli massimo a cui si può arrivare per l'aereo in questione sulla base del fatto che la compagnia aerea dichiara che i bagagli in stiva devono pesare massimo 30 kg.

```
delimiter |
CREATE TRIGGER PesoStivaInsert BEFORE INSERT ON prenotazioni
FOR EACH ROW
BEGIN
    DECLARE NumBagagli int;
    DECLARE NumBagagliMax int;

    SELECT SUM(p.Bagagli)
    INTO NumBagagli
    FROM prenotazioni p
    WHERE p.CodVoli = NEW.CodVoli;

    SELECT a.PesoStiva
    INTO NumBagagliMax
    FROM aerei a NATURAL JOIN voli v
    WHERE v.CodVoli = NEW.CodVoli;

    IF NEW.Bagagli > ((NumBagagliMax / 30) - NumBagagli) THEN
        SET NEW.Bagagli = ((NumBagagliMax / 30) - NumBagagli);
    END IF;
END;
|
delimiter ;
```

Per il caso dell'inserimento, vediamo qui che se il numero di bagagli inserito è maggiore di quello massimo, verrà assegnato il numero massimo di bagagli per il peso non sfruttato della stiva, che potrebbe anche essere 0.

```

delimiter |
CREATE TRIGGER PesoStivaUpdate BEFORE UPDATE ON prenotazioni
FOR EACH ROW
BEGIN
    DECLARE NumBagagli int;
    DECLARE NumBagagliMax int;

    SELECT SUM(p.Bagagli)
    INTO NumBagagli
    FROM prenotazioni p
    WHERE p.CodVoli = NEW.CodVoli;

    SELECT a.PesoStiva
    INTO NumBagagliMax
    FROM aerei a NATURAL JOIN voli v
    WHERE v.CodVoli = NEW.CodVoli;

    IF NEW.Bagagli > ((NumBagagliMax / 30) - NumBagagli) THEN
        SET NEW.Bagagli = ((NumBagagliMax / 30) - NumBagagli + OLD.Bagagli);
    END IF;
END;
|
delimiter ;

```

Per il caso dell'aggiornamento, vediamo qui che se il numero di bagagli da aggiungere inserito è maggiore di quello massimo, verrà aggiunto al numero di bagagli già presente, il numero massimo di bagagli per il peso non sfruttato della stiva.

```

delimiter |
CREATE TRIGGER ProfiloUpdate BEFORE UPDATE ON users
FOR EACH ROW
BEGIN
    IF NEW.Nome = null OR NEW.Nome = '' THEN
        SET NEW.Nome = OLD.Nome;
    END IF;
    IF NEW.Email = null OR NEW.Email = '' THEN
        SET NEW.Email = OLD.Email;
    END IF;
END;
|
delimiter ;

```

Per quanto riguarda il trigger sui dati degli utenti, la questione è molto più semplice. Non si vuole che gli utenti modifichino accidentalmente un dato precedentemente inserito con un niente, cioè con "" . Se nella modifica dei dati del profilo, un utente volesse cambiare, per esempio, solo l'email, non serve che compili anche il campo del nome e questo non subirà modifiche. Questo tipo di trigger sfrutta il meccanismo di NEWROW e OLDROW per evitare di sovrascrivere per sbaglio dei dati.

Questo è un esempio di trigger usato per permettere all'utente di non dover compilare tutti i campi da modificare che potrebbe essere sfruttato per tutti gli aggiornamenti di dati, ma in questo progetto è stato implementato solo in questo caso a scopo esemplificativo.

## TRANSAZIONI

La scelta del livello di isolamento per ogni transazione è stata presa in esame singolarmente e poi riesaminata alla luce delle altre scelte prese. Ora, vedremo per ogni caso quale livello di isolamento è stato scelto e perché con un esempio.

### REGISTRAZIONE DI UN UTENTE (cliente) – livello: SERIALIZABLE

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "SERIALIZABLE". Infatti, se due utenti tentano di registrarsi con lo stesso codice ID nello stesso momento, non possono più essere distinti correttamente; quindi serve che avvenga un lock sulla tabella, per permettere ad uno alla volta di accedervi e fare l'inserimento.

### MODIFICA DEL PROFILO DI UN UTENTE – livello: READ UNCOMMITTED

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "READ UNCOMMITTED". Infatti, anche se l'utente modifica i suoi dati, nessuna altra operazione può essere intaccata.

### AGGIUNTA DI BAGAGLI ALLA PRENOTAZIONE – livello: SERIALIZABLE

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "SERIALIZABLE". Infatti, se, mentre un utente vuole aggiungere altri bagagli alla sua prenotazione, un altro utente sta facendo una nuova prenotazione ed è rimasto solo un posto per i bagagli in stiva, entrambi lo sceglieranno, causando un eccesso di bagagli in stiva; quindi serve che avvenga un lock sulla tabella, per permettere ad uno alla volta di accedervi e fare l'inserimento o la modifica.

### CANCELLAZIONE PRENOTAZIONE – livello: READ UNCOMMITTED

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "READ UNCOMMITTED". Infatti, anche se l'utente cancella i suoi dati, nessuna altra operazione può essere intaccata.

### CONFERMA DELLA PRENOTAZIONE – livello: SERIALIZABLE

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "SERIALIZABLE". Infatti, se, mentre un utente vuole aggiungere altri bagagli alla sua prenotazione, un altro utente sta facendo una nuova prenotazione ed è rimasto solo un posto per i bagagli in stiva, entrambi lo sceglieranno, causando un eccesso di bagagli in stiva; quindi serve che avvenga un lock sulla tabella, per permettere ad uno alla volta di accedervi e fare l'inserimento o la modifica.

### REGISTRAZIONE DI UN UTENTE (operatore) – livello: SERIALIZABLE

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "SERIALIZABLE". Infatti, se due utenti tentano di registrarsi con lo stesso codice ID nello stesso momento, non possono più essere distinti correttamente; quindi serve che avvenga un lock sulla tabella, per permettere ad uno alla volta di accedervi e fare l'inserimento.

## REGISTRAZIONE DI UN VOLO – livello: SERIALIZABLE

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "SERIALIZABLE". Infatti, se due utenti tentano di inserire un volo per lo stesso giorno con lo stesso numero di posti e rimane un solo aereo disponibile con quel numero di posti, entrambi lo sceglieranno, causando un conflitto nell'uso degli aerei per quel giorno; quindi serve che avvenga un lock sulla tabella, per permettere ad uno alla volta di accedervi e fare l'inserimento.

## REGISTRAZIONE DI UN AEREO – livello: READ COMMITTED

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "READ COMMITTED". Infatti, se un utente sta inserendo un nuovo aereo e un altro utente sta inserendo un nuovo volo e quest'ultimo seleziona quell'aereo (attraverso la ricerca dei posti), ma poi il primo utente non completa l'inserimento, il secondo utente si troverà con un volo che ha un aereo inesistente; quindi serve che per visualizzare il nuovo aereo, questo deve aver finito e portato a buon termine il suo inserimento.

## REGISTRAZIONE DI UNA META – livello: READ COMMITTED

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "READ COMMITTED". Infatti, se un utente sta inserendo una nuova meta e un altro utente sta inserendo un nuovo volo e quest'ultimo seleziona quella meta come luogo di arrivo, ma poi il primo utente non completa l'inserimento, il secondo utente si troverà con un volo che arriva in un luogo inesistente; quindi serve che per visualizzare la nuova meta, questo deve aver finito e portato a buon termine il suo inserimento.

## MODIFICA DI UN VOLO – livello: REPEATABLE READ

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "REPEATABLE READ". Infatti, se un utente sta aggiornando un volo e un altro utente sta facendo una prenotazione questo potrebbe ottenere un totale da pagare che poi è diverso da quello che paga veramente perché nel frattempo il valore del biglietto o dello sconto è cambiato; quindi, per non avere lost update, serve che, per calcolare il prezzo da pagare per un volo, l'aggiornamento di quest'ultimo sia terminato.

## MODIFICA DI UN AEREO – livello: REPEATABLE READ

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "REPEATABLE READ". Infatti, se due utenti stanno modificando il peso della stiva dello stesso aereo e il primo va ad aggiungere 20 kg, mentre il secondo va ad aggiungere 30 kg, se il primo conclude dopo che il secondo ha iniziato, il secondo avrà preso come riferimento il peso della stiva senza i 20 kg che il primo utente aveva aggiunto, finendo con il riportare il peso sbagliato alla fine; quindi, per non avere lost update, serve che l'aggiornamento del peso della stiva sia terminato prima di farne un altro.

## MODIFICA DI UNA META – livello: READ COMMITTED

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "READ COMMITTED". Infatti, se un utente sta inserendo una nuova meta e un altro utente sta inserendo un nuovo volo e quest'ultimo seleziona quella meta come luogo di arrivo e questa meta sta cambiando nome dell'aeroporto con un nome di un altro aeroporto nella stessa città, ma poi il primo utente non completa l'inserimento, il secondo utente si troverà con un volo che arriva in un luogo con un aeroporto inesistente; quindi serve che per visualizzare la meta modificata, l'aggiornamento di quest'ultima sia terminato.

## CANCELLAZIONE DI UN VOLO – livello: SERIALIZABLE

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "SERIALIZABLE". Infatti, se un utente sta prenotando un volo e un altro utente sta cancellando quel volo, il primo utente si ritroverà con la prenotazione di un volo che non esiste; quindi serve che avvenga un lock sulla tabella, per permettere al secondo utente di cancellare il volo senza causare danni.

## CANCELLAZIONE DI UN AEREO – livello: SERIALIZABLE

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "SERIALIZABLE". Infatti, se un utente sta prenotando un volo e un altro utente sta cancellando l'aereo di quel volo, il primo utente si ritroverà con la prenotazione di un volo che non esiste (per l'ON DELETE CASCADE); quindi serve che avvenga un lock sulla tabella, per permettere al secondo utente di cancellare l'aereo senza causare danni.

## CANCELLAZIONE DI UNA META – livello: SERIALIZABLE

Questa sezione di codice prevede che ci sia una transazione con livello di isolamento "SERIALIZABLE". Infatti, se un utente sta prenotando un volo e un altro utente sta cancellando la meta di partenza di quel volo, il primo utente si ritroverà con la prenotazione di un volo che non esiste (per l'ON DELETE CASCADE); quindi serve che avvenga un lock sulla tabella, per permettere al secondo utente di cancellare la meta senza causare danni.

## 5 – CODICE INTERESSANTE E TECNOLOGIE SPECIFICHE

---

Vediamo ora un excursus di porzioni di codice interessante estrapolate da diverse sezioni del progetto.

### SQL

In questa sezione non identificheremo dove sono ritrovabili queste query, anche per il fatto che alcune sono ripetute (magari con qualche leggera differenza) in più punti.

```
INSERT INTO users VALUES(:codice,:nome,:email,:password,'Cliente')
create user :codice@'localhost' identified with mysql_native_password by :password
GRANT Cliente to :codice@'localhost'
```

Per registrare un utente vengono passati i diversi campi da riempire come variabili in Flask. Una volta inserito l'utente nella tabella users, viene creato anche un user per dargli accesso alla base di dati e gli viene assegnato il ruolo giusto (in questo caso "Cliente", ma vale lo stesso per "Operatore")

```

SELECT v.CodVoli, v.PrezzoEconomy, l1.Citta AS Parte, l2.Citta AS Arriva
FROM voli v JOIN localita l1 ON v.Da = l1.CodLocalita JOIN localita l2 ON v.Verso = l2.CodLocalita
WHERE v.Partenza >= NOW()
ORDER BY v.Partenza

```

In questa query notiamo come vengono utilizzate le chiavi esterne presenti nella tabella voli verso la tabella localita. Per ottenere un doppio collegamento serve una doppia JOIN con la stessa tabella. Per distinguere poi i nomi delle città delle due località, vengono identificati con un nome diverso grazie dall'AS. In questo caso è presente anche la clausola WHERE dove andiamo a selezionare solo le partenze dal momento corrente in poi; infine ordiniamo per data di partenza.

```

SELECT v.CodVoli, v.PrezzoEconomy, l1.Citta AS Parte, l2.Citta AS Arriva
FROM voli v JOIN localita l1 ON v.Da = l1.CodLocalita JOIN localita l2 ON v.Verso = l2.CodLocalita
WHERE v.Partenza >= NOW()
ORDER BY v.PrezzoEconomy, v.CodVoli
LIMIT 10

```

In questa query ritorna il precedente ragionamento sulle chiavi tra le tabelle voli e localita, ma vediamo che viene aggiunto LIMIT 10 per limitare il numero di righe alle prime 10.

```

SELECT v.CodVoli, v.Sconto, l1.Citta AS Parte, l2.Citta AS Arriva
FROM voli v JOIN localita l1 ON v.Da = l1.CodLocalita JOIN localita l2 ON v.Verso = l2.CodLocalita
WHERE v.Sconto > 0
ORDER BY v.Sconto, v.CodVoli

```

In questa query ritorna il precedente ragionamento sulle chiavi tra le tabelle voli e localita, ma vediamo che la clausola WHERE cambia: viene verificato se lo sconto è maggiore di 0, quindi, in termini pratici, se esiste uno sconto da applicabile.

```

SELECT l1.Citta AS CittaParte, l1.Nazione AS NazioneParte, l1.NomeAeroporto AS AeroportoParte, l2.*, v./*
FROM voli v JOIN localita l1 ON v.Da = l1.CodLocalita JOIN localita l2 ON v.Verso = l2.CodLocalita
WHERE v.CodVoli =:volo

```

In questa query ritorna il precedente ragionamento sulle chiavi tra le tabelle voli e localita, ma vediamo che la clausola WHERE cambia: al posto di ragionare su un valore, andiamo a controllare che il codice del volo corrisponda a quello che ci passa la pagina html. Notiamo anche che nella clausola SELECT andiamo a prendere più informazioni riguardanti le località, quindi non basta ridenominare la città, ma serve che le colonne di almeno una delle due tabelle siano tutti ridenominate.

```

SELECT * FROM localita l
ORDER BY l.Nazione, l.Citta

```

In questa query molto breve, notiamo che l'ordine con cui vogliamo prendere i dati è per nazione e poi per città.

```

SELECT l.Città, l.Nazione, COUNT(*) AS Scelta
FROM posti o NATURAL JOIN prenotazioni p NATURAL JOIN voli v JOIN località l ON v.Verso=l.CodLocalità
GROUP BY v.Verso
ORDER BY Scelta DESC
LIMIT 5

```

In questa query andiamo ad unire quattro tabelle: dobbiamo trovare tutte le mete che sono state scelte maggiormente come arrivo di un volo, quindi serve che si colleghino le località ai posti prenotati. Se ci fossimo fermati alle prenotazioni, senza tener conto dei posti, una prenotazione di venti posti avrebbe avuto lo stesso peso di una prenotazione di un posto. Per collegare località a posti, serve collegare località a voli, voli a prenotazioni e prenotazioni a posti. Andiamo a raggruppare per codice della località di arrivo e ordiniamo in senso decrescente per ottenere come primi risultati quelli con un conteggio maggiore; infine, limitiamo il numero delle righe a cinque.

```

SELECT p.*, l1.Città AS Parte, l2.Città AS Arriva
FROM prenotazioni p NATURAL JOIN voli v JOIN località l1 ON l1.CodLocalità = v.Da JOIN località l2 ON l2.CodLocalità = v.Verso
WHERE p.CF =:id AND v.Partenza >= NOW()

```

In questa query ritorna il precedente ragionamento sulle chiavi tra le tabelle voli e località, ma vediamo che andiamo ad unire anche la tabella delle prenotazioni. Per trovare tutte le prenotazioni e permettere all'utente di visualizzare i nomi delle città di partenza e arrivo è necessario collegare prenotazioni a località. Fermadosi a voli avremmo ottenuto solo i codici delle località. Infine, controlliamo di selezionare solo le prenotazioni dell'utente attivo in quel momento e il cui momento di partenza non superi quello attuale.

```

SELECT p.CodPrenotazioni, COUNT(p.CodPosti) AS BagagliMano, o.Bagagli AS BagagliStiva
FROM posti p NATURAL JOIN prenotazioni o
WHERE o.CodPrenotazioni =:book
GROUP BY o.CodPrenotazioni

```

In questa query andiamo a calcolare il numero di bagagli a mano e in stiva riguardanti una prenotazione specifica. Per calcolare il numero di bagagli a mano, andiamo a contare il numero di posti associati alla prenotazione, così che ogni persona abbia un bagaglio a mano. Per calcolare il numero di bagagli in stiva, invece, andiamo semplicemente a prendere il valore salvato in prenotazioni. Nella clausola WHERE andiamo a controllare di prendere solo i dati della prenotazione con il codice corrispondente a quello della prenotazione selezionata.

```

SELECT p.CodPosti
FROM posti p NATURAL JOIN prenotazioni o NATURAL JOIN voli v
WHERE v.CodVoli =:volo

```

In questa query andiamo a prendere tutti i posti già prenotati riguardo ad un volo. Per farlo andiamo ad unire tre tabelle: voli con prenotazioni e prenotazioni con posti. Otteniamo così tutti i posti. Nella clausola WHERE andiamo a controllare che il codice del volo corrisponda con quello del volo selezionato.

```

SELECT p.CodPrenotazioni
FROM prenotazioni p
WHERE p.CF =:id AND p.CodVoli =:volo AND p.DataPrenotazione = (SELECT MAX(q.DataPrenotazione)
                                                               FROM prenotazioni q
                                                               WHERE q.CF=p.CF AND q.CodVoli=p.CodVoli)

```

In questa query andiamo a cercare il codice di una prenotazione specifica. Dobbiamo trovare la prenotazione che appartiene ad un utente di cui abbiamo l'id e che faccia riferimento ad un volo specifico. Se gli utenti potessero effettuare una sola prenotazione per volo sarebbe fatta, ma possono effettuarne molteplici, e a noi interessa la più recente, quindi andiamo a cercare quella la cui data di prenotazione è la più recente con una sottoquery. Dobbiamo verificare che la data di prenotazione sia uguale alla data più recente per le prenotazioni di quel volo, quindi andiamo a prendere il massimo tra le date di prenotazione (per ottenere quella più recente) e andiamo ad assicurarcoci che il codice dell'utente e del volo siano quelli sopra indicati nella clausola WHERE facendo un confronto con i codici nella query principale.

```

SELECT SUM(p.Prezzo) AS Totale
FROM prenotazioni p

```

In questa query andiamo a trovare il totale degli incassi della compagnia aerea, quindi il totale che hanno pagato tutti i clienti.

```

SELECT u.Ruolo, COUNT(*) AS Tipo
FROM users u
GROUP BY u.Ruolo

```

In questa query andiamo a calcolare quanti utenti ci sono nella tabella users divisi per ruolo. Questo significa che sarà compreso anche "Anonimo", di cui però poi non ci interesserà; il focus sarà su "Cliente" e "Operatore".

```

SELECT COUNT(*) AS Aerei
FROM aerei

```

In questa query andiamo semplicemente a trovare quanti aerei ci sono nella tabella aerei, quindi di quanti aerei dispone la compagnia aerea.

```

SELECT AVG(v.PrezzoFirst) AS First, AVG(v.PrezzoBusiness) AS Business, AVG(v.PrezzoEconomy) AS Economy
FROM voli v

```

In questa query andiamo a trovare il prezzo medio del biglietto di ogni classe per ogni volo. Per identificare poi ogni media calcolata, utilizziamo la ridenominazione con AS.

```

SELECT c.CodClassi, COUNT(p.CodClassi) AS Numero
FROM classi c LEFT JOIN posti p ON c.CodClassi = p.CodClassi
GROUP BY c.CodClassi

```

In questa query andiamo a trovare quanti posti prenotati ci sono per ogni classe, indipendentemente dal volo a cui sono associati. Per trovare il numero, andiamo ad unire le tabelle posti e classi e poi raggruppiamo per codice di classe. Andiamo anche a rinominare il conteggio per avere accesso più facilmente al numero calcolato.

```
SELECT v.CodVoli, v.PrezzoFirst, v.PrezzoBusiness, v.PrezzoEconomy, v.Durata, l1.Citta AS Parte, l2.Citta AS Arriva, v.NumScali, v.Sconto
FROM voli v JOIN localita l1 ON v.Da = l1.CodLocalita JOIN localita l2 ON v.Verso = l2.CodLocalita
ORDER BY v.PrezzoFirst DESC
```

In questa query andiamo a selezionare tutti i voli con le loro informazioni, ma anche il nome delle città di partenza e arrivo. Per ottenere questo risultato, andiamo ad unire come in precedenza la tabella voli con la tabella localita due volte. Infine, andiamo ad ordinare le righe per il prezzo del biglietto di first class, per ottenere i voli in ordine dal più costoso al più economico.

```
SELECT DISTINCT a.NumFile
FROM aerei a
ORDER BY a.NumFile
```

In questa query andiamo a prendere il numero di file per ogni aereo. Andiamo ad ordinare il numero di file in ordine crescente e, dato che ci interessa vedere i numeri di file possibili, ma non i doppiioni di uno stesso numero, andiamo ad utilizzare DISTINCT nella clausola SELECT.

```
SELECT a.CodAerei
FROM aerei a
WHERE a.NumFile =:file AND a.CodAerei NOT IN (SELECT v.CodAerei
                                                FROM voli v
                                                WHERE DATE(v.partenza) = DATE(:partenza))
LIMIT 1
```

In questa query andiamo a prendere i codici degli aerei che rispettano certe condizioni. Innanzitutto, il numero di file deve essere quello selezionato, poi, l'aereo deve essere disponibile per la data indicata come partenza, quindi ci serve una sottoquery per fare questo controllo. Andiamo a verificare che il codice dell'aereo non sia presente nella nostra sottoquery con un NOT IN. Nella sottoquery andiamo a prendere tutti i codici degli aerei associati a dei voli che partono dalla data indicata. Una volta ottenuto l'elenco di tutti i codici degli aerei validi, andiamo a limitare il numero delle righe ad una, poiché ci interessava trovare solo un aereo.

## FLASK

Passiamo ora ad osservare direttamente il codice scritto in Flask.

```
@app.route('/', methods=['GET', 'POST'])
def home():
    return render_template ('home.html', logged=current_user.is_authenticated) #render della pagina html "home"
```

Qui notiamo che quando andiamo a renderizzare la pagina html, passiamo alla funzione anche un'altra cosa, oltre al nome della pagina da renderizzare. Passiamo la variabile 'logged' che assume il valore dell'espressione "current\_user.is\_authenticated". Questa espressione restituisce il valore della verifica di

autenticazione dell'utente. La variabile 'logged' serve poi alla pagina html (in realtà a base.html che home.html estende) per capire come impostare la barra di navigazione.

```

@app.route('/login', methods=['GET', 'POST'])
def login():
    global engine #engine globale
    if request.method == 'POST': #route triggered con metodo POST
        conn = engine.connect() #connessione aperta
        s=text('SELECT * FROM users u WHERE u.ID =:id') #ricerca utente con l'ID specificato
        rs = conn.execute(s, id=request.form['ID']).fetchone()
        if rs: #controllo esistenza di quell'utente
            real_pwd = rs['Password'] #salvataggio password
            conn.close() #connessione chiusa
            if(request.form['password']==real_pwd): #controllo correttezza della password
                user = User(rs['ID'],rs['Nome'],rs['Email'],rs['Password'],rs['Ruolo']) #creazione utente con costruttore
                login_user(user) #chiamata a flask-login
                engine=create_engine("mysql://"+rs['ID']+":"+rs['Password']+"@localhost/compagniaaereaamm") #accesso con le credenziali
                return redirect(url_for('areaRiservata')) #rimando all'area riservata dell'utente
            else: #password errata
                flash('Errore: ID o password errati','error') #messaggio di errore
                return redirect(url_for('loginPagina')) #rimando alla pagina di login
        else: #id non esistente
            flash('Errore: ID errato o non esistente','error') #messaggio di errore
            conn.close() #connessione chiusa
            return redirect(url_for('loginPagina')) #rimando alla pagina di login
    else: #route triggered con metodo GET
        return redirect(url_for('home')) #rimando alla home page

```

Prendiamo la funzione di login come esempio di diversi aspetti interessanti. Il primo è il riferimento all'engine globale, poi andiamo ad aprire la connessione con il database che verrà chiusa alla fine della funzione. Notiamo l'uso di Textual SQL per eseguire le query, ma usiamo text() per salvare il contenuto nella variabile 's' e andare poi ad eseguire s. Un'altra cosa interessante è l'uso delle variabili all'interno della query (come abbiamo visto precedentemente): possiamo passare dei valori alle query per andare a confrontare certi elementi con essi, per esempio, nella clausola WHERE. Possiamo anche usare questi valori per inserirli nel database con un INSERT INTO, come avviene, per esempio, nella funzione di registrazione. Interessante anche l'uso del costruttore della classe User per andare a salvare i dati dell'utente corrente. Notiamo anche che, colleghiamo l'utente al database richiamando l'engine. Vediamo l'uso della funzione di redirect che rimanda direttamente ad un'altra pagina. L'ultima cosa da osservare è l'uso di flash per passare messaggi informativi alla pagina html: in questo caso è stato sfruttato per passare messaggi di errore per l'utente.

```

@app.route ('/areaRiservata', methods=['GET', 'POST'])
@login_required #richiede autenticazione
def areaRiservata ():
    if (current_user.get_ruolo()=="Cliente"): #Cliente
        return redirect(url_for('prenotazioni')) #rimando alla pagina delle prenotazioni
    elif (current_user.get_ruolo()=="Operatore"): #Operatore
        return redirect(url_for('operatore')) #rimando alla pagina principale dell'operatore
    else: #altro
        return redirect(url_for('home')) #rimando alla home

```

Nella funzione 'areaRiservata' vediamo un meccanismo di autorizzazione dove andiamo a richiamare la funzione get.ruolo() della classe User per andare a ricavare il ruolo dell'utente e reindirizzare ciascun utente verso la propria area riservata. Questo meccanismo è utilizzato in molte funzioni anche per evitare che un utente non autorizzato o non autenticato possa accedere a delle aree del sito a cui non dovrebbe poter accedere. La funzione è richiamata su current\_user per ottenere i dati dell'utente corrente.

```

@app.route ('/posti', methods=['GET', 'POST'])
@login_required #richiede autenticazione
def posti():
    if (current_user.get_ruolo() == "Cliente"):
        volo = request.form['volo'] #codice del volo
        classe = request.form['classe'] #classe
        da = request.form['da'] #luogo di partenza
        verso = request.form['verso'] #luogo di arrivo
        conn=engine.connect() #connessione aperta
        t=text("SELECT v.PrezzoFirst, v.PrezzoBusiness, v.PrezzoEconomy, a.NumFile FROM voli v NATURAL JOIN aerei a WHERE v.CodVoli =:volo")
        #ricerca dei prezzi dei biglietti e del numero di file dell'aereo
        prezzi = conn.execute (t, volo=int(request.form['volo'])).fetchone()
        s=text("SELECT p.CodPosti FROM posti p NATURAL JOIN prenotazioni o NATURAL JOIN voli v WHERE v.CodVoli =:volo")
        #ricerca dei posti già prenotati per il volo scelto
        prenotati = conn.execute (s, volo=int(request.form['volo'])) 
        conn.close() #connessione chiusa
        file = prezzi['NumFile'] #numero delle file
        sedili=[] #lista per i posti con codice alfanumerico
        seats = [] #lista per i posti con codice numerico
        for posto in prenotati: #ciclo sui posti prenotati con codice alfanumerico
            sedili.append(posto['CodPosti']) #aggiunge il sedile alla lista
        for p in sedili:
            s = re.sub("[^A-Z]+","",p) #rimozione dei numeri dal nome del sedile
            n = re.sub("[^0-9]+","",p) #rimozione delle lettere dal nome del sedile
            n = int(n) #conversione da string a int
            n = n - 1 #decremento perché l'elenco delle file deve partire da 0 e non da 1
            if n >= int(int(file)*0.1): #stacco - fila saltata tra prima e seconda classe
                n = n+1
            if n >= int(int(file)*0.4)+1: #stacco - fila saltata tra seconda e terza classe
                n = n+1
            n = n * 7 #per ogni fila ci sono 6 posti e il corridoio
            if s == 'B': #secondo posto nella fila
                n = n+1
            elif s == 'C': #terzo posto nella fila - corridoio
                n = n+2
            elif s == 'D': #quarto posto nella fila
                n = n+4
            elif s == 'E': #quinto posto nella fila
                n = n+5
            elif s == 'F': #sesto posto nella fila
                n = n+6
            seats.append(n) #posto aggiunto con codice numerico corretto
        return render_template("posti.html", logged=current_user.is_authenticated, volo=volo, righe=file, classe=classe, prenotati=seats, da=da, verso=verso, pren)
    elif (current_user.get_ruolo() == "Operatore"):
        volo=int(request.form['volo']) #codice del volo
        da = request.form['da'] #luogo di partenza
        verso = request.form['verso'] #luogo di arrivo
        conn=engine.connect() #connessione aperta
        t=text("SELECT a.NumFile FROM voli v NATURAL JOIN aerei a WHERE v.CodVoli =:volo") #ricerca del numero di file dell'aereo
        righe = conn.execute (t, volo=volo).fetchone()
        s=text("SELECT p.CodPosti FROM posti p NATURAL JOIN prenotazioni o NATURAL JOIN voli v WHERE AND v.CodVoli =:volo")
        #ricerca dei posti già prenotati per il volo scelto
        prenotati = conn.execute (s, volo=volo)
        conn.close() #connessione chiusa
        file=righe['NumFile'] #numero di file
        sedili=[] #lista per i posti con codice alfanumerico
        seats = [] #lista per i posti con codice numerico
        for posto in prenotati: #ciclo sui posti prenotati con codice alfanumerico
            sedili.append(posto['CodPosti']) #aggiunge il sedile alla lista
        for p in sedili:
            s = re.sub("[^A-Z]+","",p) #rimozione dei numeri dal nome del sedile
            n = re.sub("[^0-9]+","",p) #rimozione delle lettere dal nome del sedile
            n = int(n) #conversione da string a int
            n = n - 1 #decremento perché l'elenco delle file deve partire da 0 e non da 1
            if n >= int(int(file)*0.1): #stacco - fila saltata tra prima e seconda classe
                n = n+1
            if n >= int(int(file)*0.4)+1: #stacco - fila saltata tra seconda e terza classe
                n = n+1
            n = n * 7 #per ogni fila ci sono 6 posti e il corridoio
            if s == 'B': #secondo posto nella fila
                n = n+1
            elif s == 'C': #terzo posto nella fila - corridoio
                n = n+2
            elif s == 'D': #quarto posto nella fila
                n = n+4
            elif s == 'E': #quinto posto nella fila
                n = n+5
            elif s == 'F': #sesto posto nella fila
                n = n+6
            seats.append(n) #posto aggiunto con codice numerico corretto
        return render_template("posti_op.html", logged=current_user.is_authenticated, righe=file, prenotati=seats, da=da, verso=verso) #render della pagina html
    else: #altro
        return redirect(url_for('home')) #rimanda alla home page

```

La funzione 'posti' è una delle più interessanti, se non la più interessante per la meccanica della realizzazione della mappa dei posti. La prima parte del meccanismo si trova in Flask, le altre verranno riprese nei punti successivi. Cominciamo ad addentrarci nel meccanismo, spiegando l'obiettivo di quanto scritto: la pagina 'posti' html si appoggia su un file javascript per disegnare la mappa dei posti. Questo file necessita di un elenco di posti codificati numericamente. Con una query sulla tabella posti possiamo ottenere un elenco di posti con un codice alfanumerico per fila e colonna. L'obiettivo è quindi trasformare questi codici alfanumerici in codici numerici che corrispondano univocamente l'uno all'altro. Sappiamo che

i codici alfanumerici partono da 1A e arrivano al numero della fila finale con lettera F. Le colonne rimangono sempre le stesse (da A ad F) mentre il numero di file varia. Sappiamo che l'elenco numerico partirebbe da 0 e non da 1 e, inoltre, sappiamo che i buchi della mappa per identificare le divisioni tra le classi e i corridoi sono in realtà codificati come posti normali, quindi bisogna ricordarsi di calcolare di saltarli. Per prima cosa bisogna dividere i numeri e le lettere del codice di un posto, poi il numero (che rappresenta la fila) va ridotto di uno, per far corrispondere le file con la struttura che parte da 0. Dopodiché si calcola se va aggiunto o meno il salto per la divisione delle classi. Poi si moltiplica per sette il risultato ottenuto per arrivare a posto in posizione A della fila corretta. Ora che abbiamo la codifica numerica della fila corretta, dobbiamo solo scorrere nella fila per trovare il posto giusto. Banalmente, basta guardare la lettera e sommare da uno a sei per ottenere il posto. L'unica accortezza a questo punto è ricordarsi di saltare il corridoio andando ad aumentare di uno di più per le lettere D, E ed F. Una volta ottenuto l'elenco di posti con codice numerico, basta passarlo alla pagina html. La spiegazione del resto del meccanismo è spiegata più avanti.

```

@app.route('/aggiungiBagagli', methods=['GET', 'POST'])
def aggiungiBagagli():
    codice = request.form['codice'] #codice prenotazione
    aggiungere = request.form['aggiungere']#bagagli in stiva da aggiungere
    stiva = request.form['stiva']#bagagli in stiva già presenti
    conn=engine.connect() #connessione aperta
    conn.execute("SET TRANSACTION ISOLATION LEVEL SERIALIZABLE") #livello di isolamento SERIALIZABLE
    conn.execute("START TRANSACTION") #inizio transazione
    try:
        s=text("UPDATE prenotazioni SET Bagagli =:numero WHERE CodPrenotazioni =:codice") #aggiornamento numero di bagagli in stiva
        rs = conn.execute(s, codice=codice, numero=(int(aggiungere)+int(stiva)))
        conn.execute("COMMIT") #commit delle operazioni
        conn.close() #connessione chiusa
        return redirect(url_for('areaRiservata')) #rimando all'area riservata dell'utente
    except: #qualcosa non e' andato a buon fine
        conn.execute("ROLLBACK") #rollback delle operazioni
        conn.close() #connessione chiusa
        flash('Errore: modifica non riuscita, riprovare','error') #messaggio di errore
        return redirect(url_for('areaRiservata')) #rimando all'area riservata dell'utente

```

Nella funzione 'aggiungiBagagli' vediamo un esempio di transazione con scelta di livello di isolamento. Lasceremo stare la spiegazione della scelta del livello di isolamento perché spiegata in un altro punto del documento, e ci concentreremo sulla forma. Una volta aperta la connessione, si stabilizza il livello e si inizia la transazione. Nel blocco 'try' si tenta di fare un aggiornamento dei dati con le variabili che arrivano dall'html. Notiamo che nel caso di 'numero' il valore passato non è prestabilito, ma è calcolato sulla base di due variabili. Se l'operazione riesce viene fatto il commit delle operazioni e si chiude la connessione, altrimenti si va nel blocco 'except' dove si fa il rollback delle operazioni per annullare gli effetti di ciò che è stato fatto e si chiude la connessione.

```

@app.route('/modificaVolo', methods=['GET', 'POST'])
def modificaVolo():
    codice = request.form['codice'] #informazioni del volo
    tipo=codice.split(' - ') #separazione delle informazioni del volo
    for p in tipo: #ciclo for sulle informazioni del volo
        cod = p
    volo = re.sub("[^0-9]+","",cod) #rimozione delle lettere per ottenere il codice del volo

    conn=engine.connect() #connessione aperta
    conn.execute("SET TRANSACTION ISOLATION LEVEL REPEATABLE READ") #livello di isolamento REPEATABLE READ
    conn.execute("START TRANSACTION") #inizio transazione
    try:
        s=text("UPDATE voli SET PrezzoFirst =:first, PrezzoBusiness =:business, PrezzoEconomy =:economy, Sconto =:sconto WHERE CodVoli =:volo")
        #aggiornamento dei prezzi e dello sconto del volo in questione
        rs = conn.execute(s, first=request.form['first'], business=request.form['business'], economy=request.form['economy'], sconto=request.form['sconto'], volo=volo)
        conn.execute("COMMIT") #commit delle operazioni
        conn.close() #connessione chiusa
        return redirect(url_for('operatore')) #rimando all'area riservata dell'utente
    except: #qualcosa non e' andato a buon fine
        conn.execute("ROLLBACK") #rollback delle operazioni
        conn.close() #connessione chiusa
        flash('Errore: modifica non riuscita, riprovare','error') #messaggio di errore
        return redirect(url_for('modificaVoloPagina')) #rimando alla pagina di modifica del volo

```

Nella funzione 'modificaVolo' vediamo un esempio di estrapolazione di un codice da una serie di caratteri. In questo caso, l'html non passa un valore da poter prendere così com'è, serve manipolare la stringa che viene passata. Sappiamo che la stringa è suddivisa in blocchi separati da " - ", quindi possiamo usare la funzione split per andare a separare le parti della stringa. Sappiamo anche che il codice si trova nell'ultima parte della stringa, quindi con un ciclo for andiamo a prendere l'ultima parte. Una volta ottenuta la parte

desiderata, andiamo a rimuovere tutti i caratteri che non sono cifre. A questo punto abbiamo una stringa contenente solo il codice numerico che ci interessa. Se serve si può convertire in int.

```

@app.route ('/conferma', methods=['GET', 'POST'])
@login_required #richiesta autenticazione
def conferma():
    if (current_user.get_ruolo()=='Cliente'): #Cliente
        if request.form['biglietti'] != "": #verifica che sia stato selezionato almeno un posto
            biglietti = request.form['biglietti'].split(',') #posti selezionati (viene usata la funzione split per ottenere una lista dei posti da una stringa di testo)
            bagagli = request.form['bagagli'] #numero di bagagli in stiva
            volo = request.form['volo'] #codice del volo
            cl = request.form['classe'] #classe di riferimento
            if cl == '1':
                classe = 'First' #first class
            elif cl == '2':
                classe = 'Business' #business class
            else:
                classe = 'Economy' #economy class

            conn=engine.connect() #connessione aperta
            conn.execute("SET TRANSACTION ISOLATION LEVEL SERIALIZABLE") #livello di isolamento SERIALIZABLE
            conn.execute("START TRANSACTION") #inizio transazione
            try:
                if classe == 'First': #first class
                    v=text("SELECT v.PrezzoFirst, v.Sconto FROM voli v WHERE v.CodVoli =:volo") #ricerca prezzo e sconto
                    rs = conn.execute (v, volo=volo).fetchone()
                    sc = int(rs['Sconto']) #sconto x/100
                    prezzo = float(rs['PrezzoFirst']) * len(biglietti) #calcolo prezzo senza sconto
                elif classe == 'Business': #business class
                    v=text("SELECT v.PrezzoBusiness, v.Sconto FROM voli v WHERE v.CodVoli =:volo") #ricerca prezzo e sconto
                    rs = conn.execute (v, volo=volo).fetchone()
                    sc = int(rs['Sconto']) #sconto x/100
                    prezzo = float(rs['PrezzoBusiness']) * len(biglietti) #calcolo prezzo senza sconto
                else: #economy class
                    v=text("SELECT v.PrezzoEconomy, v.Sconto FROM voli v WHERE v.CodVoli =:volo") #ricerca prezzo e sconto
                    rs = conn.execute (v, volo=volo).fetchone()
                    sc = int(rs['Sconto']) #sconto x/100
                    prezzo = float(rs['PrezzoEconomy']) * len(biglietti) #calcolo prezzo senza sconto
                    prezzo = prezzo - (prezzo * sc / 100) #calcolo prezzo con sconto applicato

                w=text("INSERT INTO prenotazioni (CodVoli,CF,DataPrenotazione,Prezzo,Bagagli) VALUES(:volo,:cf,:data,:prezzo,:bagagli)") #inserimento prenotazione nella tabella prenotazioni con autoincrement sul codice delle prenotazioni
                rs = conn.execute (w, volo=volo, cf=current_user.get_id(), data=date.today(), prezzo=prezzo, bagagli=bagagli)

                u=text("SELECT p.CodPrenotazioni FROM prenotazioni p WHERE p.CF =:id AND p.CodVoli =:volo AND p.DataPrenotazione = (SELECT MAX(q.DataPrenotazione) #ricerca del codice della prenotazione appena inserito
                rs = conn.execute (u, id=current_user.get_id(), volo=volo).fetchone()
                cp = rs['CodPrenotazioni'] #codice prenotazione

                for s in biglietti: #ciclo for sulla lista dei posti selezionati
                    t=text("INSERT INTO posti VALUES(:classe,:posto,:prenotazione)") #inserimento posto selezionato
                    rs = conn.execute (t, classe=classe, posto=s, prenotazione=cp)

                conn.execute("COMMIT") #commit delle operazioni
                conn.close() #connessione chiusa
                return render_template("conferma.html", logged=current_user.is_authenticated, biglietti=biglietti, bs=bagagli, prezzo=prezzo, sconto=sc) #render
            except: #qualcosa non e' andato a buon fine
                conn.execute("ROLLBACK") #rollback delle operazioni
                conn.close() #connessione chiusa
                flash('Errore: prenotazione non andata a buon fine','error') #messaggio di errore
                return redirect(url_for('volo')) #rimando alla pagina dei voli
            else: #nessun posto selezionato
                flash('Errore: nessun posto selezionato','error') #messaggio di errore
                return redirect(url_for('volo')) #rimando alla pagina dei voli
            else: #altro
                return redirect(url_for('home')) #rimando alla home page
        
```

Nella funzione 'conferma' andiamo a vedere due aspetti: la divisione per classe e l'inserimento di dati nel database all'interno di un ciclo for. Per la questione delle classi la faccenda non è complicata: basta creare un if-elif-else per le tre classi e dentro ciascun blocco andar a prendere e modificare i dati specifici di quella classe. In questo caso i prezzi cambiano da classe a classe, quindi serviva differenziare i conti in base alla classe scelta. L'altro aspetto da osservare è che quando arriva la sequenza di posti prenotati da inserire, arriva sottoforma di un'unica stringa. Prima andiamo a separare tutti i posti e poi andiamo ad inserire i posti nel database. Visto che non sappiamo quanti possano essere, utilizziamo un ciclo per scorrere tutto l'elenco ed inserire ogni posto con i suoi dati (sappiamo che il codice della prenotazione e della classe rimarrà lo stesso per tutti i posti selezionati).

## HTML

<pre> 38     &lt;!-- Blocco per aggiungere contenuti alla sezione head --&gt; 39     {% block head %} 40     {% endblock %} </pre>	<pre> 129    &lt;!-- Blocco per aggiungere contenuti alla sezione body --&gt; 130    {% block pagebody %} 131    {% endblock %} </pre>
--	--

Questi due blocchi si trovano in 'base.html' e servono alle pagine che la estendono per scrivere codice all'interno del blocco <head> o <body> per non avere ridondanza di codice.

```

98     <!-- Messaggi di errore per l'utente -->
99     [% with messages = get_flashed_messages(with_categories=true) %]
100     [% for category, message in messages %]
101         <li class="alert-{{ category }} form-inline my-2 my-lg-0 ml-5">{{ message }}</li>
102     <!-- Categoria del messaggio e messaggio -->
103     [% endfor %]
104     [% endwith %]

```

In questa sezione di html vediamo la resa dei messaggi informativi passati da Flask con flash. Se ci sono messaggi, questa sezione farà apparire dei messaggi d'errore all'utente. Notiamo anche che la sintassi `{%%}` viene utilizzata per utilizzare dei comandi di Flask.

```

107     <!-- Input di login o logout per l'utente autenticato o meno -->
108     <form class="form-inline my-2 my-lg-0" action="{{url_for('loginPagina')}}" method="post">
109         <input type="submit" id="login" class="btn btn-light" name="Login" value="Login" />
110     </form>
111     <form class="form-inline my-2 my-lg-0" action="{{url_for('logout')}}" method="post">
112         <input type="submit" id="logout" class="btn btn-light" name="Logout" value="Logout" />
113     </form>

```

In questa sezione di html notiamo tre cose: innanzitutto, vediamo come html passa le variabili a Flask con `<form input>`, poi osserviamo che allo stato base, la barra di navigazione mostrerà l'input per il login, non per il logout. La modifica di far apparire l'uno e far scomparire l'altro viene gestita nello `<script>` che vedremo nella sezione di javascript. Infine, vediamo che per accedere alle variabili di Flask, vengono usate le `{()}`.

```

56     <form class="form-inline ml-5" action="{{url_for('dati')}}" method="post">
57         <!-- Passaggio del codice della prenotazione non visibile -->
58         <input type="hidden" name="prenotazioni" value="{{row['CodPrenotazioni']}}"/>
59         <!-- Input per vedere maggiori dettagli riguardanti una prenotazione -->
60         <input class="btn btn-outline-dark btn-lg" type="submit" name="La tua prenotazione" value="La tua prenotazione" />
61     </form>

```

In questa sezione di html osserviamo come html può passare delle variabili a Flask senza che queste siano visibili all'utente, ovvero con `type="hidden"`.

```

50     (% if first['Wifi'] == true %) <!-- C'e' il wifi -->
51     <div class="custom-control custom-checkbox">
52         <input type="checkbox" class="custom-control-input" id="defaultCheckedDisabled2" checked disabled>
53         <label class="custom-control-label" for="defaultCheckedDisabled2">Wi-fi gratuito</label>
54     </div>
55     (% else %) <!-- Non c'e' il wifi -->
56     <div class="custom-control custom-checkbox">
57         <input type="checkbox" class="custom-control-input" id="defaultUncheckedDisabled2" disabled>
58         <label class="custom-control-label" for="defaultUncheckedDisabled2">Wi-fi gratuito</label>
59     </div>
60     (% endif %)

```

In questa sezione di html, vediamo l'uso di un `if-else` dentro all'html utilizzando i comandi di Flask. Il funzionamento è lo stesso, ma in html non avremmo la possibilità di farlo normalmente.

```

53     <!-- Corpo della tabella -->
54     <tbody>
55         (% for row in prezzi %) <!-- Ciclo for per scorrere tutte le righe della tabella con le informazioni sui voli -->
56             <tr>
57                 <td>{{row['CodVoli']}}</td> <!-- Codice del volo -->
58                 <td>{{row['PrezzoFirst']}} euro</td> <!-- Prezzo di un biglietto di first class -->
59                 <td>{{row['PrezzoBusiness']}} euro</td> <!-- Prezzo di un biglietto di business class -->
60                 <td>{{row['PrezzoEconomy']}} euro</td> <!-- Prezzo di un biglietto di economy class -->
61                 <td>{{row['Durata']}}</td> <!-- Durata del volo -->
62                 <td>{{row['Parte']}}</td> <!-- Luogo di partenza del volo -->
63                 <td>{{row['Arriva']}}</td> <!-- Luogo di arrivo del volo -->
64                 <td>{{row['NumScali']}}</td> <!-- Numero di scali durante il volo -->
65                 <td>{{row['Sconto']}}%</td> <!-- Percentuale di sconto da applicare sui biglietti -->
66             </tr>
67         (% endfor %)
68     </tbody>

```

In questa sezione di html notiamo che per andare a riempire una tabella, non serve che indichiamo riga per riga cosa inserire, possiamo, ancora una volta, utilizzare i comandi `{%%}` di Flask per andare ad eseguire un ciclo for per ogni riga della nostra tabella di partenza da cui prendere i dati da inserire. Osservare che sia per l' `if-else`, sia per il ciclo for, al termine è presente la chiusura de blocco, come per `head` e `pagebody`.

```

18 <!-- Mappa dei posti e colonna a destra -->
19 <div class="content">
20   <!-- Mappa dei posti -->
21   <div id="map-container"></div>
22 
23   <div class="right">
24     <!-- Legenda dei posti -->
25     <div id="legend-container"></div>
26     <br />
27     <form action="{{url_for('conferma')}}" method="post">
28       <label for="bagagli" class="sr-only">Numero bagagli in stiva</label>
29       <!-- Numero di bagagli in stiva da inserire -->
30       <input type="number" min="0" name="bagagli" id="bagagli" class="form-control" placeholder="Numero bagagli in stiva" required autofocus />
31       <br />
32       <!-- carrello -->
33       <div id="cart-container"></div>
34       <br />
35       <!-- Passaggio del codice del volo non visibile -->
36       <input type="hidden" name="volo" value="{{volo}}"/>
37       <!-- Passaggio dei codici dei biglietti non visibile -->
38       <input id="biglietti" type="hidden" name="biglietti" />
39       <!-- Passaggio della classe come numero non visibile -->
40       <input id="classe" type="hidden" name="classe" value="{{classe}}"/>
41       <!-- Input per andare alla pagina di conferma della prenotazione -->
42       <input id="conferma" class="btn btn-success btn-block" type="submit" name="Concludi" value="Concludi" />
43     </form>
44   </div>
45 </div>

```

In questa sezione di html andiamo a riprendere il discorso dei posti. Se guardiamo bene, vediamo che in ogni `<input>`, a meno che non sia un valore da inserire manualmente da parte dell'utente, si deve dichiarare il 'value' che corrisponde al valore associato a quell'id, ovvero a quella variabile. C'è però un id che non ha associato un value: biglietti. Questa variabile viene passata a Flask da qui, ma il suo valore viene trovato dal file javascript 'seatchart.js'. Finiremo quindi di spiegare il funzionamento nella sezione di javascript.

```

72 <p class="text-left mt-2 mb-1 ml-2">Luogo partenza</p>
73 <!-- Luogo di partenza da inserire per essere registrato -->
74 <select id="da" name="da" class="form-control mb-1" required autofocus>
75   (% for row in citta %) <!-- Ciclo per scorrere tutte le righe della tabella con Le Localita' (codice, citta', nome aeroporto) -->
76   <!-- Per ogni riga della tabella -->
77   <option value="{{row['CodLocalita']}} - {{row['citta']}} - {{row['NomeAeroporto']}}>{{row['CodLocalita']}} - {{row['citta']}} - {{row['NomeAeroporto']}}</option>
78   (% endfor %)
79 </select>
80 
81 <p class="text-left mt-2 mb-1 ml-2">Luogo arrivo</p>
82 <!-- Luogo di arrivo da inserire per essere registrato -->
83 <select id="verso" name="verso" class="form-control mb-1" required autofocus>
84   <!-- Qui la lista delle localita' viene generata in base a quella delle localita' di partenza tramite lo script -->
85 </select>

```

In questa sezione di html troviamo il caso in cui l'operatore deve scegliere da quale meta far partire un volo e a quale meta farlo arrivare. Per evitare che possa accidentalmente scegliere la stessa meta per entrambe le mete, la lista viene inizialmente caricata solo per la partenza. La `<select>` per l'arrivo è impostata, ma non è riempita. Il suo riempimento con l'esclusione della selezione della partenza avverrà nello `<script>`, che vedremo nella sezione di javascript.

## JAVASCRIPT

Concludiamo col parlare del codice in javascript.

```

115   <script type="text/javascript">
116     (% if logged == true %) // se l'utente e' autenticato
117       $("login").attr("type", "hidden"); // L'input login viene nascosto
118       $("logout").attr("type", "submit"); // L'input logout viene mostrato
119       $("#areaRiservata").attr("disabled", false); // L'area riservata e' accessibile
120     (% else %) // se l'utente non e' autenticato
121       $("login").attr("type", "submit"); // L'input login viene mostrato
122       $("logout").attr("type", "hidden"); // L'input logout viene nascosto
123       $("#areaRiservata").attr("disabled", true); // L'area riservata non e' accessibile
124     (% endif %)
125   </script>

```

Questo pezzo di codice javascript si riferisce alla pagina 'base.html' dove si deve capire se attivare il login o il logout (e se rendere disponibile o meno l'aerea riservata). Questo script fa proprio questo. Va a controllare se la variabile 'logged' passata da Flask è true o false. Se è true, il login verrà nascosto e il logout verrà mostrato (e l'area riservata sarà resa disponibile); se è false, il login verrà mostrato e il logout verrà nascosto (e l'area riservata sarà resa non disponibile).

```

131 <script type="text/javascript">
132 /**
133 * Quando l'operatore sceglie una localita' per la partenza, l'elenco delle localita' di arrivo deve cambiare.
134 * Bisogna far sì che il luogo di partenza e quello di arrivo non coincidano.
135 */
136 $('#da').change(function () {
137     var copy = $('#da').clone(); // si copia l'elenco delle partenze
138     copy.attr("id", "verso"); // si mette l'id di "verso"
139     copy.attr("name", "verso"); // si mette il name di "verso"
140     $("#verso").replaceWith(copy); // si inserisce il nuovo elenco dove ci sono gli arrivi
141     $("#verso option[value='" + $(this).val() + "']").remove(); // si rimuove il luogo già selezionato come partenza
142 });
143 </script>

```

Questo pezzo di codice javascript si riferisce alla pagina “reg\_volo.html” in cui si va ad inserire un nuovo volo. La <selection> per scegliere la meta dell’arrivo del volo inizialmente è vuota, come abbiamo visto, ma quando viene selezionata una meta per la partenza, viene attivata questa funzione nello script che genera le <option> per la <selection> dell’arrivo, andando prima a copiare quella della partenza e poi andando a rimuovere la meta già selezionata. Infine, la nuova <selection> viene inserita al posto della precedente. Questo meccanismo avviene ogni qualvolta viene cambiata la meta della partenza.

```

580
581     var posti = [];
582     $("#conferma").unbind('click').bind('click', function () { // quando viene cliccato "Concludi"
583         $(".carrello").each(function () { // si prendono i biglietti creati nel carrello
584             posti.push($(this).text()); // si salvano i posti selezionati
585         })
586         $("#biglietti").attr("value", posti.toString()); // si inviano i posti selezionati
587     });
588 }

```

Questo pezzo di codice javascript si riferisce al discorso sui posti. Questa funzione si trova nel file ‘seatchart.js’: prende ogni biglietto generato nel carrello, trova il nome del posto e lo va ad inserire in un array. Una volta terminato con l’inserimento, alla variabile con id ‘biglietti’ va cambiato il value, o meglio generato, inserendo l’array con le codifiche dei posti. Questa funzione viene attivata quando il cliente preme sull’input “Concludi” che ha come id “conferma”.

```

52
53     var options; // opzioni per la creazione della mappa
54     var prenotati = JSON.parse('{{ prenotati|safe }}'); // elenco dei posti prenotati
55
56     /** FIRST CLASS ***/
57     if ({{ classe | int }} == 1) {
58
59         var other = []; // posti di altre classi
60         var k = 0;
61         var s = 6;
62
63         var start = ({{ righe | int * 0.1 | int }} + 1) * 7; // da dove comincia la business class
64         var end = ({{ righe | int }} + 2) * 7; // dove finisce l'economy class
65         for (var i = start; i < end; i++) { // ciclo for per identificare i posti da saltare (corridoi)
66
67             if (k < 3) {
68                 other.push(i); // posti nella prima fila a sinistra
69                 k++;
70             }
71             else {
72                 if (s < 6) {
73                     other.push(i); // tutti gli altri posti
74                     s++;
75                 }
76                 else {
77                     s = 0;
78                 }
79             }
80         }
81     }

```

Questo pezzo di codice javascript si riferisce al discorso sui posti, in particolare allo <script> della pagina html. Per generare la mappa dei posti, serve indicare le opzioni, qui denominate options, che vedremo nell’immagine successiva. Per il momento, concentriamoci sul calcolare quali posti andranno resi non disponibili perché già prenotati. Prendiamo la variabile “prenotati” passata da flask e con json andiamo a rendere l’elenco in un formato più adatto a javascript, ovvero un array. Qui c’è anche un esempio di calcolo di posti da oscurare perché di una classe diversa da quella selezionata. Avendo il numero di righe, possiamo andare a calcolare dove comincia una classe e dove finisce, andando anche a saltare il corridoio e le divisioni delle classi. La formula è sempre la stessa per calcolare le dimensioni delle classi, infatti alla first class spetta il 10%, alla business class il 30% e all’economy class il restante 60%. Di conseguenza è facile recuperare la posizione delle divisioni delle classi.

```

189 options = { // opzioni per la mappa
190   types: [ // biglietti di first class con il prezzo
191     { type: "First", backgroundColor: "#ffce00", price: {{prezzi['PrezzoFirst']}} },
192   ],
193   map: { // grandezza mappa che corrisponde alla grandezza dell'aereo corrispondente al volo
194     id: "map-container",
195     rows: {{ righe|int + 2 }}, // numero file
196     columns: 7, // numero colonne
197   },
198   reserved: {
199     seats: prenotati // posti prenotati
200   },
201   classe: {
202     seats: other // posti di altre classi
203   },
204   disabled: { // posti non esistenti
205     seats: [0, 6],
206     rows: [{(righe|int * 0.1)|int}, {(righe|int * 0.4)|int + 1 }], // divisori tra le classi
207     columns: [3] // corridoi
208   }
209 },
210 cart: { // carrello
211   id: "cart-container",
212   width: 280,
213   height: 250,
214   currency: 'euro ',
215 },
216 legend: { // Legenda
217   id: "Legend-container",
218 },
219 };

```

Questo pezzo di codice javascript si riferisce all'ultimo tassello del meccanismo dei posti. Qui andiamo ad inserire le opzioni per creare la mappa dei posti. Prima si inseriscono i dettagli della classe, come il biglietto; poi si stabiliscono le misure della grandezza della mappa, poi si indicano i posti già prenotati, andando finalmente ad utilizzare l'array con i codici numerici dei posti che abbiamo calcolato; si indicano i posti da oscurare perché di un'altra classe e si disabilitano i posti che non ci dovrebbero essere, ovvero quelli nelle divisioni di classi e nel corridoio (oltre ai posti in alto all'estrema destra e sinistra). Infine, andiamo ad indicare le dimensioni dl carrello. Per la mappa, il carrello e la legenda andiamo anche ad indicare il nome dell'id corrispondente.

## 6 – VISUALIZZAZIONE IMMAGINI

Oltre alle immagini presenti nel carousel della home page, nelle pagine relative alle mete c'è un'immagine per ogni città e nazione. Queste immagini sono cercate nelle cartelle del progetto all'interno di un ciclo for, quindi per trovarle ed inserirle è stato necessario denominarle come la città o la nazione in questione.

```
{% for row in localita %} <!-- Ciclo for per scorrere tutte le righe della tabella con le informazioni sulle mete -->
<!-- Per ogni riga della tabella -->


## {{row['Città']}}</h2> <!-- Nome città --> Scopri questa bellissima città!...</p>



<!-- Immagine della città -->
![Responsive image](static/images/{{row['Città']}}.jpg)



## {{row['Nazione']}}</h2> <!-- Nome nazione --> ... in questo magnifico paese.</p>



<!-- Immagine della nazione -->
![Responsive image](static/images/{{row['Nazione']}}.jpg)


</div>
{% endfor %}
```

Andando ad aggiungere nuove mete, è facile che la nazione non sia presente tra quelle già inserite, e tantomeno la città, quindi, quando verrà aperta una delle pagine delle mete, non si visualizzeranno le immagini per la nuova meta. Sarà necessario andare ad aggiungere delle immagini nella cartella 'images' che si chiamino esattamente con i nomi della città e della nazione inseriti.

## 7 – DIVISIONE DEI COMPITI

---

Il progetto deve dimostrare: conoscenza dell'utilizzo di Flask; conoscenza dell'utilizzo di SQLAlchemy; padronanza di SQL e di uno specifico DBMS.

L'intero progetto è stato realizzato in collaborazione. Qui sono riportate le specifiche di chi ha contribuito in maniera maggiore ad ogni aspetto del progetto, anche se non esclusivamente.

Sezione del progetto	Ricerca	Realizzazione	Revisione
Creazione schema della base di dati	Mario	Mario	Melania
Creazione e basi del progetto in Flask	Melania	Melania	Mario
Creazione utenti, ruoli e permessi	Melania	Melania	Mario
Base dell'html con barra di navigazione	Melania	Melania	Mario
Home page	Melania	Melania	Mario
Collegamenti con la base di dati	Mario	Mario	Melania
Meccanismo di login e logout	Mario	Mario	Melania
Registrazione utenti	Mario	Mario	Melania
Meccanismo di autenticazione	Melania	Mario	Melania
Meccanismo di autorizzazione	Mario	Melania	Mario
Pagine standard per voli, mete e classi	Melania	Mario	Melania
Pagine voli low cost, offerte e mete preferite	Melania	Melania	Mario
Pagine dettagli e dati	Mario	Mario	Melania
Struttura e divisione aree riservate	Melania	Melania	Mario
Pagine e procedura per la prenotazione e conferma	Melania	Melania	Mario
Aggiornamento e cancellazione prenotazione	Mario	Mario	Melania
Statistiche	Melania	Melania	Mario
Aggiornamento dei dati dell'utente	Melania	Mario	Melania
Inserimento, aggiornamento e cancellazione voli	Melania	Melania	Mario
Inserimento, aggiornamento e cancellazione aerei	Mario	Mario	Melania
Inserimento, aggiornamento e cancellazione mete	Mario	Mario	Melania
Messaggi di errore	Melania	Mario	Melania
Visualizzazione immagini	Melania	Melania	Mario
Trigger	Melania	Melania	Mario
Transazioni	Melania	Melania	Mario
Riempimento base di dati	-	Mario	Melania
Test per il funzionamento corretto	-	Mario	Melania
Commentare il codice Flask	-	Melania	Mario
Commentare il codice Html	-	Mario	Melania
Commentare il codice Javascript	-	Melania	Mario
Commentare i trigger e le transazioni	-	Melania	Mario
Stesura del documento	-	Melania	Mario

## 8 – FUTURO DELLA WEB APP

---

Ci sono degli aspetti che sicuramente si possono sviluppare maggiormente in questo progetto e che, se dovesse essere utilizzato per una vera compagnia aerea sarebbero significativi.

La prima cosa sarebbe aggiungere la possibilità per gli utenti di fare delle ricerche mirate, ovvero scegliere il luogo di partenza, quello di arrivo, la data in cui vorrebbero viaggiare, il numero di passeggeri e magari un budget entro cui stare. Un'altra opzione sarebbe quella di differenziare i prezzi non solo per classe, ma anche per fascia d'età (es. riduzione per bambini ed ultrasessantenni) e status (cliente normale o premium).

Un'altra variazione da apportare sarebbe quella di inserire la possibilità al cliente di cancellare uno o più posti, non solo l'intera prenotazione; oppure dargli l'occasione di cambiare uno o più posti con altri. Si potrebbe anche aggiungere la possibilità di selezionare in un'unica prenotazione posti di diverse classi.

Si potrebbe creare una tabella per monitorare gli scambi di denaro e dare la possibilità agli operatori di risarcire un cliente quanto un volo viene cancellato. L'operatore potrebbe anche segnalare ai clienti eventuali cancellazioni, cambiamenti o ritardi.

Una grossa aggiunta, che andrebbe anche a cambiare la struttura della base di dati, sarebbe quella di aggiungere un'altra categoria di persone che possono accedere al sito con diversi permessi: le agenzie di viaggi. Un agente avrebbe la possibilità di visualizzare, per esempio, la lista degli aerei e prenotarne uno interno per un viaggio che come provenienza e destinazione ha due località concordate con la compagnia aerea. Potrebbe anche registrare nuovi clienti ed ottenere sconti sostanziosi per grossi gruppi di persone che prenotano tutte insieme. Un agente potrebbe, inoltre, confrontare direttamente i prezzi di più voli con diverse opzioni per il monitoraggio.

Queste sono alcune tra le aggiunte che porterebbero la web app ad un livello molto più alto e migliore, ma ce ne sono molte altre che si potrebbero attuare.