

FlyAway.com

Airline Booking Portal

Prototype of the Application

Name: RAJDEEP KUMAR

GAUTAM

GitHub: <https://github.com/rajdeepkgautam/JavaFSDP2.git>

The prototype of the application starts from the frontend, and it can also directly start from the project folder. This portal allows us to do flight management across administrator and provide flight booking facilities across client side which will at the end page goes to the payment portal(dummy). This prototype is built through various webpages (mainly .jsp file) which are interconnected with backend (servlets, database, models).

The implementation is done with the help of Hibernate, maven, Servlet, Java EE, Apache tomcat v8.5 and Eclipse.

Sprint Planning

The Implementation is done in two sprints which are mentioned below:

Sprint 1:

- Clarify the specification and requirements.
- Implement a framework of certain pages such as admin login, homepage.
- Implement a blueprint of Controller, Models part at backend and its core functionality.
- Identifying the various association for mapping of Passenger with Flight database along with its attributes containing primary key in both the tables.

Sprint 2:

- Building a platform for the prototype with hibernate, maven (webapp archetype) integration along with MySQL as a database which will run on local server (Tomcat v8.5) and required dependencies.
- Creating JSP Page as a starting point containing a hyperlink that will take us to the Admin Login page and contains a html table containing booking details such as source, destination, date of boarding and number of persons.
- Afterward, as this part is broadly categorized into two parts: admin and passenger section.
- Introducing a single controller (Servlet) as the data will be share not just within admin or passenger section but also with each other, packages consisting of models, hibernate configuration, data transfer objects for database connectivity.

Sprint 3:

- Implement functionality in controller for validation in admin login page consisting of email and password which is stated as email (admin@test.com) and password (admin).

- Implement functionality for changing password in admin section which consist of new password and confirmation password (same as new password for recheck).
- Implementing a web page for admin login page and after successful login will jump to the admin main page.
- Implementing another JSP page for changing password which is not connected currently.

Sprint 4:

- Developing the main page displaying flight details along with add, change password and logout button.
- Adding functionality for adding flights acting as hyperlink that will take us to next page asking for Flight Details such as flight number, airline, origin, destination, flight date.
- Once the admin submits the required details, it will store all the valid data into the database through servlet along with auto increment primary key not null values. And it will display in the main page along with a “edit” and “delete” button on status column.

Sprint 5:

- From the passenger side, once a user registered the details for their travel, a filter operation is performed in backend by retrieving the flight details from database and filter it according to source, destination, and boarding date.
- Along each flight details, there will be a hyperlink button name “Book Now”.
- From the “Book Now” button, it will jump to register page where user must put his/her details according to the number of persons.

Sprint 6:

- In the registration page, there will be an option to select add passenger and a view table where user can view all the passenger details and modifications can also be done through status column option.
- Once the registration is done, it will show all the flight summary such as source, destination, date of boarding and total flight price.
- If a user registered passenger details more than number of persons, it will go back to the homepage again.

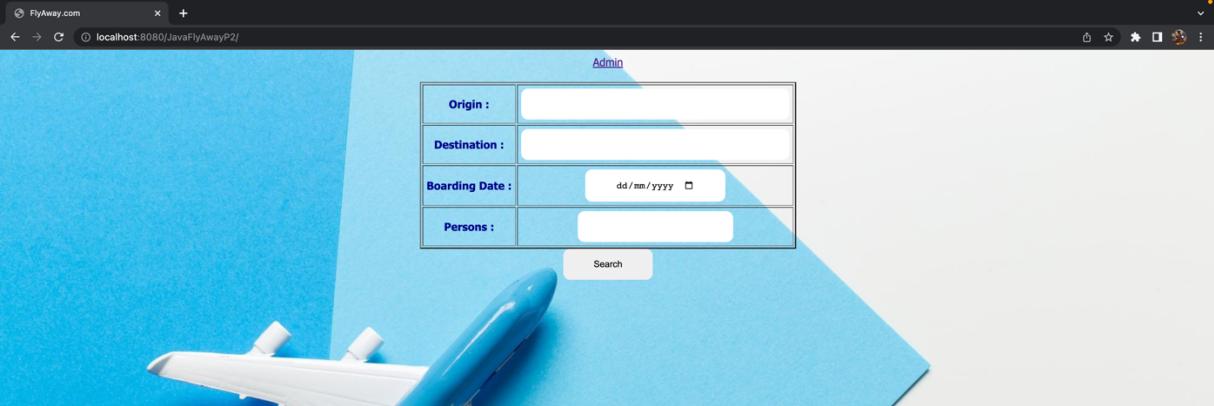
Sprint 7:

- From the passenger side, once a user registered the details for their travel, a filter operation is performed in backend by retrieving the flight details from database and filter it according to source, destination, and boarding date.
- Afterward, a dummy payment page will be displayed having an option to go back to home page as a hyperlink.
- Ensuring all the operations are tight and working well.
- Documentation.

Documentation of the functionality:

Here is the various static Java Servlet Pages that user will come across along the way.

1: Home Page

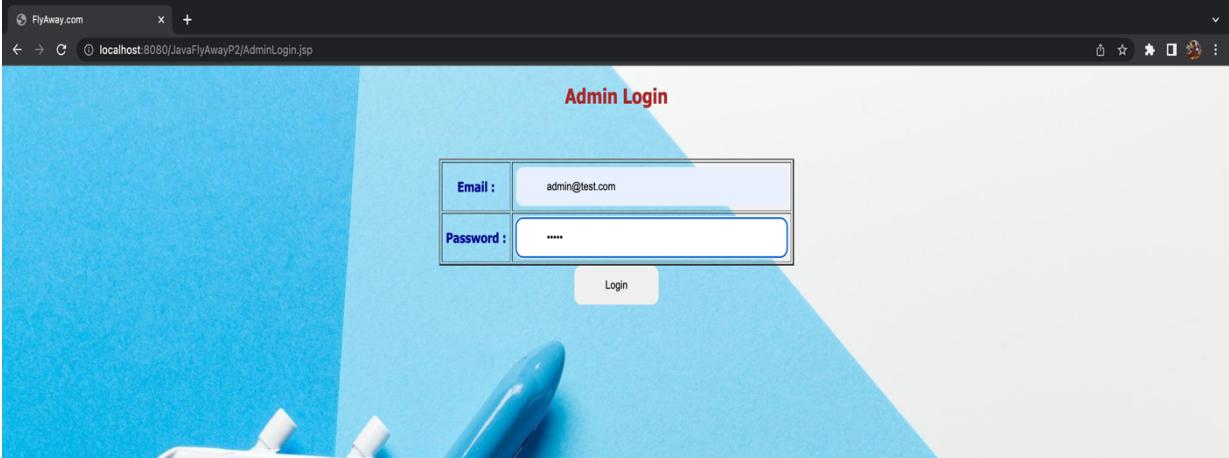


The screenshot shows a web browser window titled "FlyAway.com" with the URL "localhost:8080/JavaFlyAwayP2/". The page has a light blue background featuring a white airplane silhouette. At the top right, there is a link labeled "Admin". Below it is a form with four input fields:

Origin :	<input type="text"/>
Destination :	<input type="text"/>
Boarding Date :	<input type="text"/> dd/mm/yyyy
Persons :	<input type="text"/>

Below the form is a "Search" button.

2: Admin Login Page (From admin side)

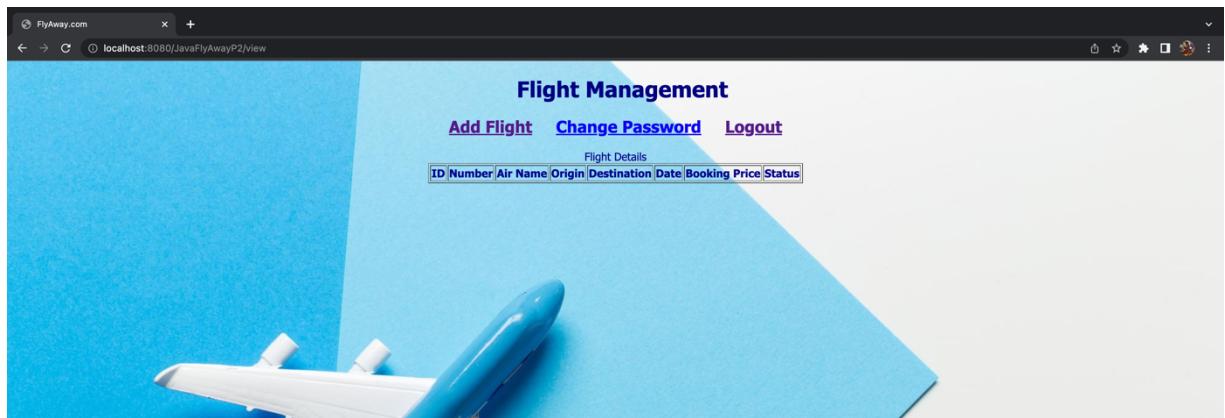


The screenshot shows a web browser window titled "FlyAway.com" with the URL "localhost:8080/JavaFlyAwayP2/AdminLogin.jsp". The page has a light blue background featuring a white airplane silhouette. The title "Admin Login" is centered at the top in red text. Below it is a form with two input fields:

Email :	<input type="text"/> admin@test.com
Password :	<input type="password"/>

Below the form is a "Login" button.

3: Admin main page (From admin side)



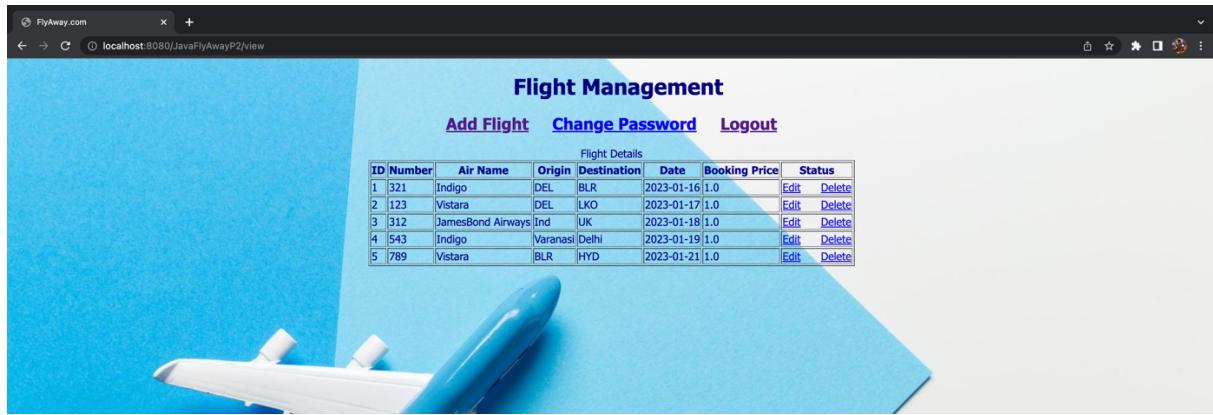
4: Add Flight Page (From admin side)

The screenshot shows a web browser window titled "FlyAway.com" with the URL "localhost:8080/JavaFlyAwayP2/edit?id=3". The page displays a form for adding a flight. The form fields are as follows:

Flight Number :	312
Airline :	JamesBond Airways
Origin :	Ind
Destination :	UK
Flight Date :	18/01/2023
Ticket Price :	60000
<input type="button" value="Save"/>	

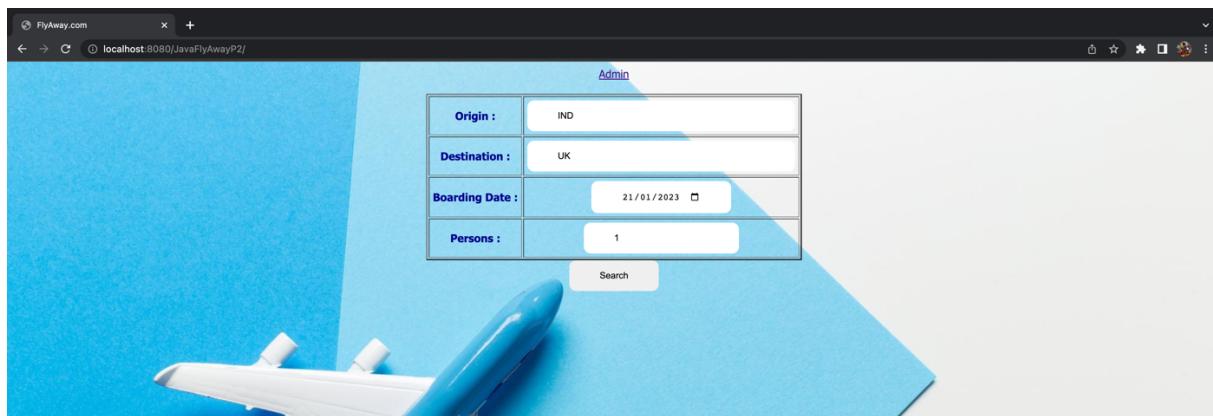
The background of the page features a blue airplane model.

*Note: Once the user saves it after inserting details, it will be inserted into the database.

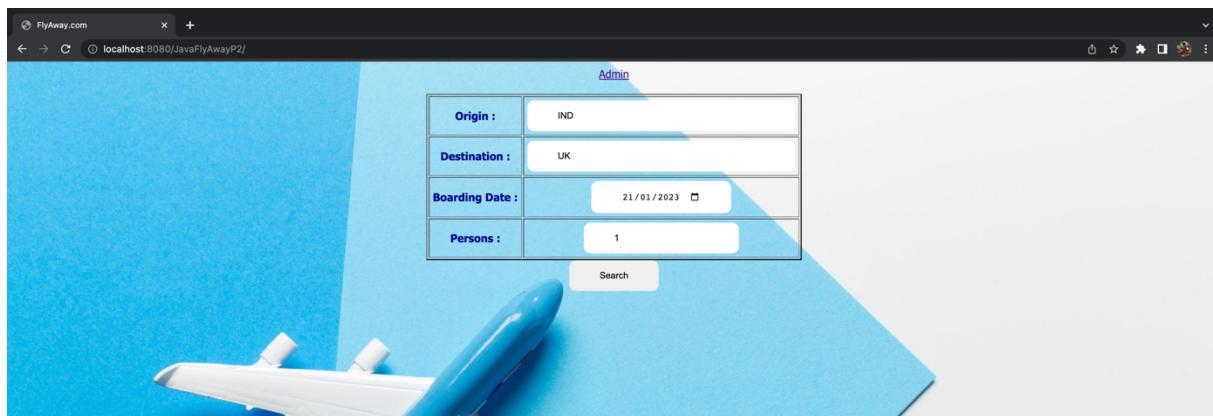


After, Admin can logout back to the homepage.

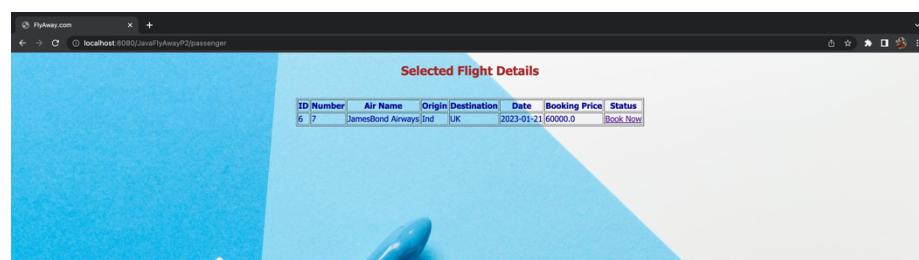
5: Insert the booking details from client side.



6: After inserting booking details, when user click the “search” button.



7: After filtering flight details and user pressed the “Book Now” button. It will surf to Passenger Details.



8: Add Passenger Page, Entering Passenger Details

The image contains two identical screenshots of a web browser displaying the 'Add Passenger' form. The browser title bar reads 'FlyAway.com' and the address bar shows 'localhost:8080/JavaFlyAwayP2/AddPassenger.jsp'. The form consists of five input fields: 'First Name' (Albert), 'Last Name' (Einstein), 'Contact' (9876543210), 'Age' (37), and 'Email' (albert@test.com). A 'Save' button is located at the bottom right of the form.

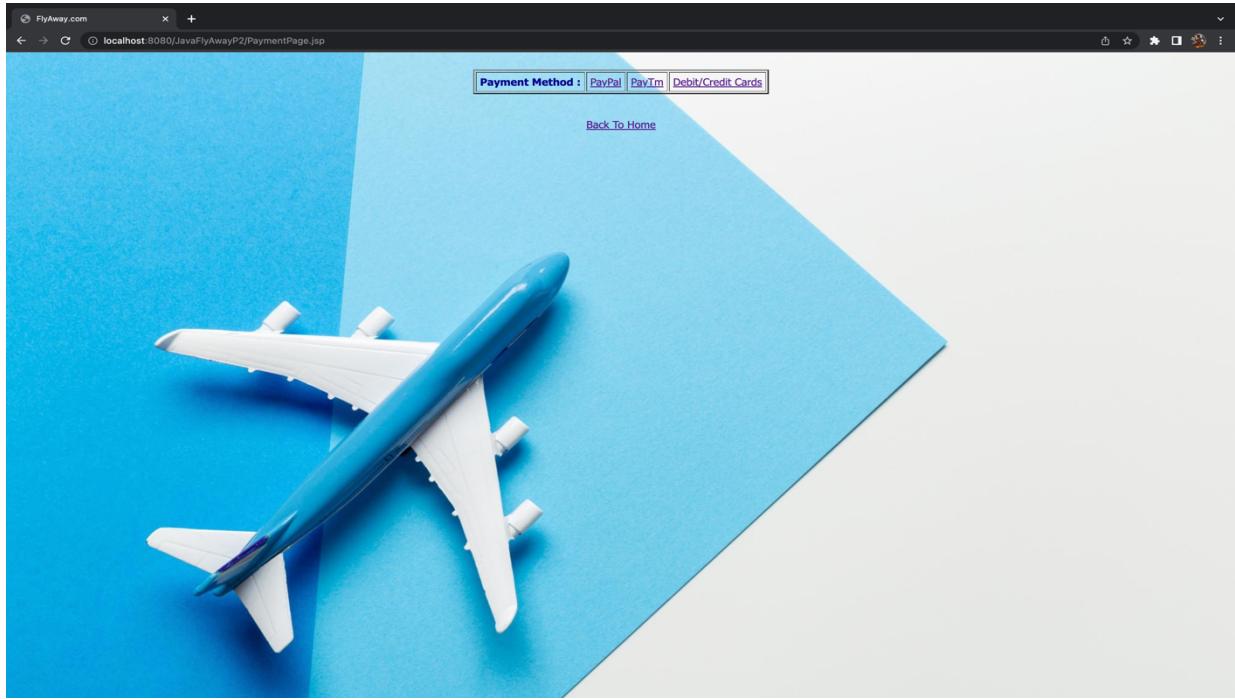
9: Summary Page

The image shows a screenshot of a web browser with the title 'FlyAway.com' and address 'localhost:8080/JavaFlyAwayP2/register'. The main heading is 'Summary Details' in red. Below it is a table with the following flight information:

Flight Number :	7
Flight Name :	JamesBond Airways
Flight From :	Ind
Flight To :	UK
Flight Boarding Date :	2023-01-21
Ticket Price :	60000.0

A 'Payment' button is located at the bottom center of the summary section.

10: Payment Page



11: In MySQL Database, we have implemented @ManyToMany associations, here is the two tables flight and passengers table with its attributes and primary key.

```
|mysql> use rkg;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> desc flight;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| flight_id | int | NO | PRI | NULL | auto_increment |
| flight_name | varchar(255) | YES | | NULL |
| Boarding_Date | varchar(255) | YES | | NULL |
| flight_number | int | YES | | NULL |
| Source | varchar(255) | YES | | NULL |
| Ticket_price | float | YES | | NULL |
| Destination | varchar(255) | YES | | NULL |
+-----+-----+-----+-----+-----+
7 rows in set (0.01 sec)

mysql> |
```

```
mysql> desc passenger;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| passenger_id | int | NO | PRI | NULL | auto_increment |
| passenger_age | int | YES | | NULL |
| passenger_mob | bigint | YES | | NULL |
| passenger_email | varchar(255) | YES | | NULL |
| passenger_fname | varchar(255) | YES | | NULL |
| passenger_lname | varchar(255) | YES | | NULL |
+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> select * from passenger;
+-----+-----+-----+-----+-----+
| passenger_id | passenger_age | passenger_mob | passenger_email | passenger_fname | passenger_lname |
+-----+-----+-----+-----+-----+
| 1 | 37 | 9876543210 | albert@test.com | Albert | Einstein |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

And a third table named (flight_passenger) containing the mapping primary key of both the table is done.

```
mysql> select * from flight_passenger;
+-----+-----+
| flight_id | passenger_id |
+-----+-----+
| 6 | 1 |
+-----+-----+
1 row in set (0.00 sec)

mysql> select * from flight;
+-----+-----+-----+-----+-----+-----+
| flight_id | flight_name | Boarding_Date | flight_number | Source | Ticket_price | Destination |
+-----+-----+-----+-----+-----+-----+
| 1 | JamesBond Airways | 2023-01-21 | | 7 | IND | 60000 | UK |
| 2 | Indigo | 2023-01-16 | | 6332 | DEL | 4500 | BLR |
| 3 | Indigo | 2023-01-20 | | 4332 | DEL | 3500 | LKO |
| 4 | Vistara | 2023-01-25 | | 9883 | BLR | 4500 | HYD |
| 5 | Vistara | 2023-01-26 | | 7654 | Varanasi | 3600 | Delhi |
| 6 | Indigo | 2023-01-16 | | 6332 | DEL | 4500 | BLR |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

```
1 package models;
2
3 import java.util.ArrayList;
4 import java.util.HashSet;
5 import java.util.List;
6
7 import javax.persistence.CascadeType;
8
9 import javax.persistence.Column;
10 import javax.persistence.Entity;
11 import javax.persistence.GeneratedValue;
12 import javax.persistence.GenerationType;
13 import javax.persistence.Id;
14 import javax.persistence.Table;
15 import javax.persistence.JoinColumn;
16 import javax.persistence.JoinTable;
17 import javax.persistence.ManyToOne;
18
19 @Entity
20 @Table(name = "Flight")
21 public class Flight {
22
23     @Id
24     @GeneratedValue(strategy = GenerationType.
IDENTITY)
25     @Column(name = "flight_id")
26     private int flightId;
27
28     @Column(name = "flight_number")
29     private int flightNumber;
30
31     @Column(name = "flight_name")
32     private String airline;
33
34     @Column(name = "Source")
35     private String origin;
36
37     @Column(name = "Destination")
38     private String target;
39
40     @Column(name = "Boarding_Date")
```

```
41     private String dob;  
42  
43     @Column(name = "Ticket_price")  
44     private float price;  
45  
46     public Flight() {  
47         }  
48  
49     @ManyToMany(cascade = CascadeType.ALL)  
50     @JoinTable(name = "flight_passenger",  
51                 joinColumns = {  
52                     @JoinColumn(name = "flight_id")  
53                 },  
54                 inverseJoinColumns = {  
55                     @JoinColumn(name = "passenger_id")  
56                 }  
57             )  
58     List<Passenger> passenger = new ArrayList<  
      Passenger>();  
59  
60  
61     public Flight(int flightId, int flightNumber,  
       String airline, String origin, String target,  
       String dob, float price) {  
62         super();  
63         this.flightId = flightId;  
64         this.flightNumber = flightNumber;  
65         this.airline = airline;  
66         this.origin = origin;  
67         this.target = target;  
68         this.dob = dob;  
69         this.price = price;  
70     }  
71  
72     public Flight(int flightNumber, String airline  
       , String origin, String target, String dob, float  
       price) {  
73         super();  
74         this.flightNumber = flightNumber;  
75         this.airline = airline;  
76         this.origin = origin;
```

```
77         this.target = target;
78         this.dob = dob;
79         this.price = price;
80     }
81
82     public int getFlightId() {
83         return flightId;
84     }
85
86     public void setFlightId(int flightId) {
87         this.flightId = flightId;
88     }
89
90     public int getFlightNumber() {
91         return flightNumber;
92     }
93
94     public void setFlightNumber(int flightNumber
95 ) {
96         this.flightNumber = flightNumber;
97     }
98
99     public String getAirline() {
100        return airline;
101    }
102
103    public void setAirline(String airline) {
104        this.airline = airline;
105    }
106
107    public String getOrigin() {
108        return origin;
109    }
110
111    public void setOrigin(String origin) {
112        this.origin = origin;
113    }
114
115    public String getTarget() {
116        return target;
117    }
```

```
117
118     public void setTarget(String target) {
119         this.target = target;
120     }
121
122     public String getDob() {
123         return dob;
124     }
125
126     public void setDob(String dob) {
127         this.dob = dob;
128     }
129
130     public float getPrice() {
131         return price;
132     }
133
134     public void setPrice(float price) {
135         this.price = price;
136     }
137
138     public List<Passenger> getPassenger() {
139         return passenger;
140     }
141
142     public void setPassenger(List<Passenger>
143         passenger) {
144         this.passenger = passenger;
145     }
146 }
147
```

```
1 package models;
2
3 public class Password {
4
5     private static String pwd;
6
7     public Password() {
8         pwd = "admin";
9     }
10
11    public static String getPwd() {
12        return pwd;
13    }
14
15    public static void setPwd(String pwd) {
16        Password.pwd = pwd;
17    }
18
19 }
20
```

```
1 package models;
2
3 import java.util.ArrayList;
4 import java.util.HashSet;
5 import java.util.List;
6 import java.util.Set;
7
8 import javax.persistence.CascadeType;
9 import javax.persistence.Column;
10 import javax.persistence.Entity;
11 import javax.persistence.GeneratedValue;
12 import javax.persistence.GenerationType;
13 import javax.persistence.Id;
14 import javax.persistence.Table;
15 import javax.persistence.ManyToOne;
16
17 @Entity
18 @Table(name = "Passenger")
19 public class Passenger {
20
21     @Id
22     @GeneratedValue(strategy = GenerationType.
23     IDENTITY)
24     @Column(name="passenger_id")
25     private int pId;
26
27     @Column(name="passenger_fname")
28     private String fname;
29
30     @Column(name="passenger_lname")
31     private String lname;
32
33     @Column(name="passenger_age")
34     private int age;
35
36     @Column(name="passenger_mob")
37     private long contact;
38
39     @Column(name="passenger_email")
40     private String email;
```

```
41     @ManyToMany(mappedBy = "passenger")//, cascade
42         = CascadeType.MERGE)
43     private List<Flight> flight = new ArrayList<
44         Flight>();
45
46
47     public Passenger() {
48
49     }
50
51
52     public Passenger(int pId, String fname, String
53         lname, int age, long contact, String email) {
54         super();
55         this.pId = pId;
56         this.fname = fname;
57         this.lname = lname;
58         this.age = age;
59         this.contact = contact;
60         this.email = email;
61     }
62
63
64     public Passenger(String fname, String lname,
65         int age, long contact, String email) {
66         super();
67         this.fname = fname;
68         this.lname = lname;
69         this.age = age;
70         this.contact = contact;
71         this.email = email;
72     }
73
74
75     public int getpId() {
76         return pId;
77     }
78
79     public void setpId(int pId) {
80         this.pId = pId;
81     }
82
83
84     public String getFname() {
85         return fname;
86     }
87
88     public void setFname(String fname) {
89         this.fname = fname;
90     }
91
92     public String getLname() {
93         return lname;
94     }
95
96     public void setLname(String lname) {
97         this.lname = lname;
98     }
99
100    public int getAge() {
101        return age;
102    }
103
104    public void setAge(int age) {
105        this.age = age;
106    }
107
108    public long getContact() {
109        return contact;
110    }
111
112    public void setContact(long contact) {
113        this.contact = contact;
114    }
115
116    public String getEmail() {
117        return email;
118    }
119
120    public void setEmail(String email) {
121        this.email = email;
122    }
123
124    public List<Flight> getFlight() {
125        return flight;
126    }
127
128    public void setFlight(List<Flight> flight) {
129        this.flight = flight;
130    }
131
132    @Override
133    public String toString() {
134        return "Passenger{" +
135            "pId=" + pId +
136            ", fname='" + fname + '\'' +
137            ", lname='" + lname + '\'' +
138            ", age=" + age +
139            ", contact=" + contact +
140            ", email='" + email + '\'' +
141            '}';
142    }
143
144    @Override
145    public boolean equals(Object o) {
146        if (this == o) return true;
147        if (o == null || getClass() != o.getClass()) return false;
148        Passenger passenger = (Passenger) o;
149        return pId == passenger.pId &&
150            fname.equals(passenger.fname) &&
151            lname.equals(passenger.lname) &&
152            age == passenger.age &&
153            contact == passenger.contact &&
154            email.equals(passenger.email);
155    }
156
157    @Override
158    public int hashCode() {
159        return Objects.hash(pId, fname, lname, age, contact, email);
160    }
161
162    @Override
163    public void persist() {
164        em.persist(this);
165    }
166
167    @Override
168    public void remove() {
169        em.remove(this);
170    }
171
172    @Override
173    public void merge() {
174        em.merge(this);
175    }
176
177    @Override
178    public void flush() {
179        em.flush();
180    }
181
182    @Override
183    public void clear() {
184        em.clear();
185    }
186
187    @Override
188    public void close() {
189        em.close();
190    }
191
192    @Override
193    public void lock() {
194        em.lock(this);
195    }
196
197    @Override
198    public void unlock() {
199        em.unlock(this);
200    }
201
202    @Override
203    public void evict() {
204        em.evict(this);
205    }
206
207    @Override
208    public void refresh() {
209        em.refresh(this);
210    }
211
212    @Override
213    public void refresh(Object... objects) {
214        em.refresh(objects);
215    }
216
217    @Override
218    public void evict(Object... objects) {
219        em.evict(objects);
220    }
221
222    @Override
223    public void flush(Object... objects) {
224        em.flush(objects);
225    }
226
227    @Override
228    public void clear(Object... objects) {
229        em.clear(objects);
230    }
231
232    @Override
233    public void lock(Object... objects) {
234        em.lock(objects);
235    }
236
237    @Override
238    public void unlock(Object... objects) {
239        em.unlock(objects);
240    }
241
242    @Override
243    public void refresh(Object... objects) {
244        em.refresh(objects);
245    }
246
247    @Override
248    public void evict(Object... objects) {
249        em.evict(objects);
250    }
251
252    @Override
253    public void flush(Object... objects) {
254        em.flush(objects);
255    }
256
257    @Override
258    public void clear(Object... objects) {
259        em.clear(objects);
260    }
261
262    @Override
263    public void lock(Object... objects) {
264        em.lock(objects);
265    }
266
267    @Override
268    public void unlock(Object... objects) {
269        em.unlock(objects);
270    }
271
272    @Override
273    public void refresh(Object... objects) {
274        em.refresh(objects);
275    }
276
277    @Override
278    public void evict(Object... objects) {
279        em.evict(objects);
280    }
281
282    @Override
283    public void flush(Object... objects) {
284        em.flush(objects);
285    }
286
287    @Override
288    public void clear(Object... objects) {
289        em.clear(objects);
290    }
291
292    @Override
293    public void lock(Object... objects) {
294        em.lock(objects);
295    }
296
297    @Override
298    public void unlock(Object... objects) {
299        em.unlock(objects);
300    }
301
302    @Override
303    public void refresh(Object... objects) {
304        em.refresh(objects);
305    }
306
307    @Override
308    public void evict(Object... objects) {
309        em.evict(objects);
310    }
311
312    @Override
313    public void flush(Object... objects) {
314        em.flush(objects);
315    }
316
317    @Override
318    public void clear(Object... objects) {
319        em.clear(objects);
320    }
321
322    @Override
323    public void lock(Object... objects) {
324        em.lock(objects);
325    }
326
327    @Override
328    public void unlock(Object... objects) {
329        em.unlock(objects);
330    }
331
332    @Override
333    public void refresh(Object... objects) {
334        em.refresh(objects);
335    }
336
337    @Override
338    public void evict(Object... objects) {
339        em.evict(objects);
340    }
341
342    @Override
343    public void flush(Object... objects) {
344        em.flush(objects);
345    }
346
347    @Override
348    public void clear(Object... objects) {
349        em.clear(objects);
350    }
351
352    @Override
353    public void lock(Object... objects) {
354        em.lock(objects);
355    }
356
357    @Override
358    public void unlock(Object... objects) {
359        em.unlock(objects);
360    }
361
362    @Override
363    public void refresh(Object... objects) {
364        em.refresh(objects);
365    }
366
367    @Override
368    public void evict(Object... objects) {
369        em.evict(objects);
370    }
371
372    @Override
373    public void flush(Object... objects) {
374        em.flush(objects);
375    }
376
377    @Override
378    public void clear(Object... objects) {
379        em.clear(objects);
380    }
381
382    @Override
383    public void lock(Object... objects) {
384        em.lock(objects);
385    }
386
387    @Override
388    public void unlock(Object... objects) {
389        em.unlock(objects);
390    }
391
392    @Override
393    public void refresh(Object... objects) {
394        em.refresh(objects);
395    }
396
397    @Override
398    public void evict(Object... objects) {
399        em.evict(objects);
400    }
401
402    @Override
403    public void flush(Object... objects) {
404        em.flush(objects);
405    }
406
407    @Override
408    public void clear(Object... objects) {
409        em.clear(objects);
410    }
411
412    @Override
413    public void lock(Object... objects) {
414        em.lock(objects);
415    }
416
417    @Override
418    public void unlock(Object... objects) {
419        em.unlock(objects);
420    }
421
422    @Override
423    public void refresh(Object... objects) {
424        em.refresh(objects);
425    }
426
427    @Override
428    public void evict(Object... objects) {
429        em.evict(objects);
430    }
431
432    @Override
433    public void flush(Object... objects) {
434        em.flush(objects);
435    }
436
437    @Override
438    public void clear(Object... objects) {
439        em.clear(objects);
440    }
441
442    @Override
443    public void lock(Object... objects) {
444        em.lock(objects);
445    }
446
447    @Override
448    public void unlock(Object... objects) {
449        em.unlock(objects);
450    }
451
452    @Override
453    public void refresh(Object... objects) {
454        em.refresh(objects);
455    }
456
457    @Override
458    public void evict(Object... objects) {
459        em.evict(objects);
460    }
461
462    @Override
463    public void flush(Object... objects) {
464        em.flush(objects);
465    }
466
467    @Override
468    public void clear(Object... objects) {
469        em.clear(objects);
470    }
471
472    @Override
473    public void lock(Object... objects) {
474        em.lock(objects);
475    }
476
477    @Override
478    public void unlock(Object... objects) {
479        em.unlock(objects);
480    }
481
482    @Override
483    public void refresh(Object... objects) {
484        em.refresh(objects);
485    }
486
487    @Override
488    public void evict(Object... objects) {
489        em.evict(objects);
490    }
491
492    @Override
493    public void flush(Object... objects) {
494        em.flush(objects);
495    }
496
497    @Override
498    public void clear(Object... objects) {
499        em.clear(objects);
500    }
501
502    @Override
503    public void lock(Object... objects) {
504        em.lock(objects);
505    }
506
507    @Override
508    public void unlock(Object... objects) {
509        em.unlock(objects);
510    }
511
512    @Override
513    public void refresh(Object... objects) {
514        em.refresh(objects);
515    }
516
517    @Override
518    public void evict(Object... objects) {
519        em.evict(objects);
520    }
521
522    @Override
523    public void flush(Object... objects) {
524        em.flush(objects);
525    }
526
527    @Override
528    public void clear(Object... objects) {
529        em.clear(objects);
530    }
531
532    @Override
533    public void lock(Object... objects) {
534        em.lock(objects);
535    }
536
537    @Override
538    public void unlock(Object... objects) {
539        em.unlock(objects);
540    }
541
542    @Override
543    public void refresh(Object... objects) {
544        em.refresh(objects);
545    }
546
547    @Override
548    public void evict(Object... objects) {
549        em.evict(objects);
550    }
551
552    @Override
553    public void flush(Object... objects) {
554        em.flush(objects);
555    }
556
557    @Override
558    public void clear(Object... objects) {
559        em.clear(objects);
560    }
561
562    @Override
563    public void lock(Object... objects) {
564        em.lock(objects);
565    }
566
567    @Override
568    public void unlock(Object... objects) {
569        em.unlock(objects);
570    }
571
572    @Override
573    public void refresh(Object... objects) {
574        em.refresh(objects);
575    }
576
577    @Override
578    public void evict(Object... objects) {
579        em.evict(objects);
580    }
581
582    @Override
583    public void flush(Object... objects) {
584        em.flush(objects);
585    }
586
587    @Override
588    public void clear(Object... objects) {
589        em.clear(objects);
590    }
591
592    @Override
593    public void lock(Object... objects) {
594        em.lock(objects);
595    }
596
597    @Override
598    public void unlock(Object... objects) {
599        em.unlock(objects);
600    }
601
602    @Override
603    public void refresh(Object... objects) {
604        em.refresh(objects);
605    }
606
607    @Override
608    public void evict(Object... objects) {
609        em.evict(objects);
610    }
611
612    @Override
613    public void flush(Object... objects) {
614        em.flush(objects);
615    }
616
617    @Override
618    public void clear(Object... objects) {
619        em.clear(objects);
620    }
621
622    @Override
623    public void lock(Object... objects) {
624        em.lock(objects);
625    }
626
627    @Override
628    public void unlock(Object... objects) {
629        em.unlock(objects);
630    }
631
632    @Override
633    public void refresh(Object... objects) {
634        em.refresh(objects);
635    }
636
637    @Override
638    public void evict(Object... objects) {
639        em.evict(objects);
640    }
641
642    @Override
643    public void flush(Object... objects) {
644        em.flush(objects);
645    }
646
647    @Override
648    public void clear(Object... objects) {
649        em.clear(objects);
650    }
651
652    @Override
653    public void lock(Object... objects) {
654        em.lock(objects);
655    }
656
657    @Override
658    public void unlock(Object... objects) {
659        em.unlock(objects);
660    }
661
662    @Override
663    public void refresh(Object... objects) {
664        em.refresh(objects);
665    }
666
667    @Override
668    public void evict(Object... objects) {
669        em.evict(objects);
670    }
671
672    @Override
673    public void flush(Object... objects) {
674        em.flush(objects);
675    }
676
677    @Override
678    public void clear(Object... objects) {
679        em.clear(objects);
680    }
681
682    @Override
683    public void lock(Object... objects) {
684        em.lock(objects);
685    }
686
687    @Override
688    public void unlock(Object... objects) {
689        em.unlock(objects);
690    }
691
692    @Override
693    public void refresh(Object... objects) {
694        em.refresh(objects);
695    }
696
697    @Override
698    public void evict(Object... objects) {
699        em.evict(objects);
700    }
701
702    @Override
703    public void flush(Object... objects) {
704        em.flush(objects);
705    }
706
707    @Override
708    public void clear(Object... objects) {
709        em.clear(objects);
710    }
711
712    @Override
713    public void lock(Object... objects) {
714        em.lock(objects);
715    }
716
717    @Override
718    public void unlock(Object... objects) {
719        em.unlock(objects);
720    }
721
722    @Override
723    public void refresh(Object... objects) {
724        em.refresh(objects);
725    }
726
727    @Override
728    public void evict(Object... objects) {
729        em.evict(objects);
730    }
731
732    @Override
733    public void flush(Object... objects) {
734        em.flush(objects);
735    }
736
737    @Override
738    public void clear(Object... objects) {
739        em.clear(objects);
740    }
741
742    @Override
743    public void lock(Object... objects) {
744        em.lock(objects);
745    }
746
747    @Override
748    public void unlock(Object... objects) {
749        em.unlock(objects);
750    }
751
752    @Override
753    public void refresh(Object... objects) {
754        em.refresh(objects);
755    }
756
757    @Override
758    public void evict(Object... objects) {
759        em.evict(objects);
760    }
761
762    @Override
763    public void flush(Object... objects) {
764        em.flush(objects);
765    }
766
767    @Override
768    public void clear(Object... objects) {
769        em.clear(objects);
770    }
771
772    @Override
773    public void lock(Object... objects) {
774        em.lock(objects);
775    }
776
777    @Override
778    public void unlock(Object... objects) {
779        em.unlock(objects);
780    }
781
782    @Override
783    public void refresh(Object... objects) {
784        em.refresh(objects);
785    }
786
787    @Override
788    public void evict(Object... objects) {
789        em.evict(objects);
790    }
791
792    @Override
793    public void flush(Object... objects) {
794        em.flush(objects);
795    }
796
797    @Override
798    public void clear(Object... objects) {
799        em.clear(objects);
800    }
801
802    @Override
803    public void lock(Object... objects) {
804        em.lock(objects);
805    }
806
807    @Override
808    public void unlock(Object... objects) {
809        em.unlock(objects);
810    }
811
812    @Override
813    public void refresh(Object... objects) {
814        em.refresh(objects);
815    }
816
817    @Override
818    public void evict(Object... objects) {
819        em.evict(objects);
820    }
821
822    @Override
823    public void flush(Object... objects) {
824        em.flush(objects);
825    }
826
827    @Override
828    public void clear(Object... objects) {
829        em.clear(objects);
830    }
831
832    @Override
833    public void lock(Object... objects) {
834        em.lock(objects);
835    }
836
837    @Override
838    public void unlock(Object... objects) {
839        em.unlock(objects);
840    }
841
842    @Override
843    public void refresh(Object... objects) {
844        em.refresh(objects);
845    }
846
847    @Override
848    public void evict(Object... objects) {
849        em.evict(objects);
850    }
851
852    @Override
853    public void flush(Object... objects) {
854        em.flush(objects);
855    }
856
857    @Override
858    public void clear(Object... objects) {
859        em.clear(objects);
860    }
861
862    @Override
863    public void lock(Object... objects) {
864        em.lock(objects);
865    }
866
867    @Override
868    public void unlock(Object... objects) {
869        em.unlock(objects);
870    }
871
872    @Override
873    public void refresh(Object... objects) {
874        em.refresh(objects);
875    }
876
877    @Override
878    public void evict(Object... objects) {
879        em.evict(objects);
880    }
881
882    @Override
883    public void flush(Object... objects) {
884        em.flush(objects);
885    }
886
887    @Override
888    public void clear(Object... objects) {
889        em.clear(objects);
890    }
891
892    @Override
893    public void lock(Object... objects) {
894        em.lock(objects);
895    }
896
897    @Override
898    public void unlock(Object... objects) {
899        em.unlock(objects);
900    }
901
902    @Override
903    public void refresh(Object... objects) {
904        em.refresh(objects);
905    }
906
907    @Override
908    public void evict(Object... objects) {
909        em.evict(objects);
910    }
911
912    @Override
913    public void flush(Object... objects) {
914        em.flush(objects);
915    }
916
917    @Override
918    public void clear(Object... objects) {
919        em.clear(objects);
920    }
921
922    @Override
923    public void lock(Object... objects) {
924        em.lock(objects);
925    }
926
927    @Override
928    public void unlock(Object... objects) {
929        em.unlock(objects);
930    }
931
932    @Override
933    public void refresh(Object... objects) {
934        em.refresh(objects);
935    }
936
937    @Override
938    public void evict(Object... objects) {
939        em.evict(objects);
940    }
941
942    @Override
943    public void flush(Object... objects) {
944        em.flush(objects);
945    }
946
947    @Override
948    public void clear(Object... objects) {
949        em.clear(objects);
950    }
951
952    @Override
953    public void lock(Object... objects) {
954        em.lock(objects);
955    }
956
957    @Override
958    public void unlock(Object... objects) {
959        em.unlock(objects);
960    }
961
962    @Override
963    public void refresh(Object... objects) {
964        em.refresh(objects);
965    }
966
967    @Override
968    public void evict(Object... objects) {
969        em.evict(objects);
970    }
971
972    @Override
973    public void flush(Object... objects) {
974        em.flush(objects);
975    }
976
977    @Override
978    public void clear(Object... objects) {
979        em.clear(objects);
980    }
981
982    @Override
983    public void lock(Object... objects) {
984        em.lock(objects);
985    }
986
987    @Override
988    public void unlock(Object... objects) {
989        em.unlock(objects);
990    }
991
992    @Override
993    public void refresh(Object... objects) {
994        em.refresh(objects);
995    }
996
997    @Override
998    public void evict(Object... objects) {
999        em.evict(objects);
1000    }
1001
1002    @Override
1003    public void flush(Object... objects) {
1004        em.flush(objects);
1005    }
1006
1007    @Override
1008    public void clear(Object... objects) {
1009        em.clear(objects);
1010    }
1011
1012    @Override
1013    public void lock(Object... objects) {
1014        em.lock(objects);
1015    }
1016
1017    @Override
1018    public void unlock(Object... objects) {
1019        em.unlock(objects);
1020    }
1021
1022    @Override
1023    public void refresh(Object... objects) {
1024        em.refresh(objects);
1025    }
1026
1027    @Override
1028    public void evict(Object... objects) {
1029        em.evict(objects);
1030    }
1031
1032    @Override
1033    public void flush(Object... objects) {
1034        em.flush(objects);
1035    }
1036
1037    @Override
1038    public void clear(Object... objects) {
1039        em.clear(objects);
1040    }
1041
1042    @Override
1043    public void lock(Object... objects) {
1044        em.lock(objects);
1045    }
1046
1047    @Override
1048    public void unlock(Object... objects) {
1049        em.unlock(objects);
1050    }
1051
1052    @Override
1053    public void refresh(Object... objects) {
1054        em.refresh(objects);
1055    }
1056
1057    @Override
1058    public void evict(Object... objects) {
1059        em.evict(objects);
1060    }
1061
1062    @Override
1063    public void flush(Object... objects) {
1064        em.flush(objects);
1065    }
1066
1067    @Override
1068    public void clear(Object... objects) {
1069        em.clear(objects);
1070    }
1071
1072    @Override
1073    public void lock(Object... objects) {
1074        em.lock(objects);
1075    }
1076
1077    @Override
1078    public void unlock(Object... objects) {
1079        em.unlock(objects);
1080    }
1081
1082    @Override
1083    public void refresh(Object... objects) {
1084        em.refresh(objects);
1085    }
1086
1087    @Override
1088    public void evict(Object... objects) {
1089        em.evict(objects);
1090    }
1091
1092    @Override
1093    public void flush(Object... objects) {
1094        em.flush(objects);
1095    }
1096
1097    @Override
1098    public void clear(Object... objects) {
1099        em.clear(objects);
1100    }
1101
1102    @Override
1103    public void lock(Object... objects) {
1104        em.lock(objects);
1105    }
1106
1107    @Override
1108    public void unlock(Object... objects) {
1109        em.unlock(objects);
1110    }
1111
1112    @Override
1113    public void refresh(Object... objects) {
1114        em.refresh(objects);
1115    }
1116
1117    @Override
1118    public void evict(Object... objects) {
1119        em.evict(objects);
1120    }
1121
1122    @Override
1123    public void flush(Object... objects) {
1124        em.flush(objects);
1125    }
1126
1127    @Override
1128    public void clear(Object... objects) {
1129        em.clear(objects);
1130    }
1131
1132    @Override
1133    public void lock(Object... objects) {
1134        em.lock(objects);
1135    }
1136
1137    @Override
1138    public void unlock(Object... objects) {
1139        em.unlock(objects);
1140    }
1141
1142    @Override
1143    public void refresh(Object... objects) {
1144        em.refresh(objects);
1145    }
1146
1147    @Override
1148    public void evict(Object... objects) {
1149        em.evict(objects);
1150    }
1151
1152    @Override
1153    public void flush(Object... objects) {
1154        em.flush(objects);
1155    }
1156
1157    @Override
1158    public void clear(Object... objects) {
1159        em.clear(objects);
1160    }
1161
1162    @Override
1163    public void lock(Object... objects) {
1164        em.lock(objects);
1165    }
1166
1167    @Override
1168    public void unlock(Object... objects) {
1169        em.unlock(objects);
1170    }
1171
1172    @Override
1173    public void refresh(Object... objects) {
1174        em.refresh(objects);
1175    }
1176
1177    @Override
1178    public void evict(Object... objects) {
1179        em.evict(objects);
1180    }
1181
1182    @Override
1183    public void flush(Object... objects) {
1184        em.flush(objects);
1185    }
1186
1187    @Override
1188    public void clear(Object... objects) {
1189        em.clear(objects);
1190    }
1191
1192    @Override
1193    public void lock(Object... objects) {
1194        em.lock(objects);
1195    }
1196
1197    @Override
1198    public void unlock(Object... objects) {
1199        em.unlock(objects);
1200    }
1201
1202    @Override
1203    public void refresh(Object... objects) {
1204        em.refresh(objects);
1205    }
1206
1207    @Override
1208    public void evict(Object... objects) {
1209        em.evict(objects);
1210    }
1211
1212    @Override
1213    public void flush(Object... objects) {
1214        em.flush(objects);
1215    }
1216
1217    @Override
1218    public void clear(Object... objects) {
1219        em.clear(objects);
1220    }
1221
1222    @Override
1223    public void lock(Object... objects) {
1224        em.lock(objects);
1225    }
1226
1227    @Override
1228    public void unlock(Object... objects) {
1229        em.unlock(objects);
1230    }
1231
1232    @Override
1233    public void refresh(Object... objects) {
1234        em.refresh(objects);
1235    }
1236
1237    @Override
1238    public void evict(Object... objects) {
1239        em.evict(objects);
1240    }
1241
1242    @Override
1243    public void flush(Object... objects) {
1244        em.flush(objects);
1245    }
1246
1247    @Override
1248    public void clear(Object... objects) {
1249        em.clear(objects);
1250    }
1251
1252    @Override
1253    public void lock(Object... objects) {
1254        em.lock(objects);
1255    }
1256
1257    @Override
1258    public void unlock(Object... objects) {
1259        em.unlock(objects);
1260    }
1261
1262    @Override
1263    public void refresh(Object... objects) {
1264        em.refresh(objects);
1265    }
1266
1267    @Override
1268    public void evict(Object... objects) {
1269        em.evict(objects);
1270    }
1271
1272    @Override
1273    public void flush(Object... objects) {
1274        em.flush(objects);
1275    }
1276
1277    @Override
1278    public void clear(Object... objects) {
1279        em.clear(objects);
1280    }
1281
1282    @Override
1283    public void lock(Object... objects) {
1284        em.lock(objects);
1285    }
1286
1287    @Override
1288    public void unlock(Object... objects) {
1289        em.unlock(objects);
1290    }
1291
1292    @Override
1293    public void refresh(Object... objects) {
1294        em.refresh(objects);
1295    }
1296
1297    @Override
1298    public void evict(Object... objects) {
1299        em.evict(objects);
1300    }
1301
1302    @Override
1303    public void flush(Object... objects) {
1304        em.flush(objects);
1305    }
1306
1307    @Override
1308    public void clear(Object... objects) {
1309        em.clear(objects);
1310    }
1311
1312    @Override
1313    public void lock(Object... objects) {
1314        em.lock(objects);
1315    }
1316
1317    @Override
1318    public void unlock(Object... objects) {
1319        em.unlock(objects);
1320    }
1321
1322    @Override
1323    public void refresh(Object... objects) {
1324        em.refresh(objects);
1325    }
1326
1327    @Override
1328    public void evict(Object... objects) {
1329        em.evict(objects);
1330    }
1331
1332    @Override
1333    public void flush(Object... objects) {
1334        em.flush(objects);
1335    }
1336
1337    @Override
1338    public void clear(Object... objects) {
1339        em.clear(objects);
1340    }
1341
1342    @Override
1343    public void lock(Object... objects) {
1344        em.lock(objects);
1345    }
1346
1347    @Override
1348    public void unlock(Object... objects) {
1349        em.unlock(objects);
1350    }
1351
1352    @Override
1353    public void refresh(Object... objects) {
1354        em.refresh(objects);
1355    }
1356
1357    @Override
1358    public void evict(Object... objects) {
1359        em.evict(objects);
1360    }
1361
1362    @Override
1363    public void flush(Object... objects) {
1364        em.flush(objects);
1365    }
1366
1367    @Override
1368    public void clear(Object... objects) {
1369        em.clear(objects);
1370    }
1371
1372    @Override
1373    public void lock(Object... objects) {
1374        em.lock(objects);
1375    }
1376
1377    @Override
1378    public void unlock(Object... objects) {
1379        em.unlock(objects);
1380    }
1381
1382    @Override
1383    public void refresh(Object... objects) {
1384        em.refresh(objects);
1385    }
1386
1387    @Override
1388    public void evict(Object... objects) {
1389        em.evict(objects);
1390    }
1391
1392    @Override
1393    public void flush(Object... objects) {
1394        em.flush(objects);
1395    }
1396
1397    @Override
1398    public void clear(Object... objects) {
1399        em.clear(objects);
1400    }
1401
1402    @Override
1403    public void lock(Object... objects) {
1404        em.lock(objects);
1405    }
1406
1407    @Override
1408    public void unlock(Object... objects) {
1409        em.unlock(objects);
1410    }
1411
1412    @Override
1413    public void refresh(Object... objects) {
1414        em.refresh(objects);
1415    }
1416
1417    @Override
1418    public void evict(Object... objects) {
1419        em.evict(objects);
1420    }
1421
1422    @Override
1423    public void flush(Object... objects) {
1424        em.flush(objects);
1425    }
1426
1427    @Override
1428    public void clear(Object... objects) {
1429        em.clear(objects);
1430    }
1431
1432    @Override
1433    public void lock(Object... objects) {
1434        em.lock(objects);
1435    }
1436
1437    @Override
1438    public void unlock(Object... objects) {
1439        em.unlock(objects);
1440    }
1441
1442    @Override
1443    public void refresh(Object... objects) {
1444        em.refresh(objects);
1445    }
1446
1447    @Override
1448    public void evict(Object... objects) {
1449        em.evict(objects);
1450    }
1451
1452    @Override
1453    public void flush(Object... objects) {
1454        em.flush(objects);
1455    }
1456
1457    @Override
1458    public void clear(Object... objects) {
1459        em.clear(objects);
1460    }
1461
1462    @Override
1463    public void lock(Object... objects) {
1464        em.lock(objects);
1465    }
1466
1467    @Override
1468    public void unlock(Object... objects) {
1469        em.unlock(objects);
1470    }
1471
1472    @Override
1473    public void refresh(Object... objects) {
1474        em.refresh(objects);
1475    }
1476
1477    @Override
1478    public void evict(Object... objects) {
1479        em.evict(objects);
1480    }
1481
1482    @Override
1483    public void flush(Object... objects) {
1484        em.flush(objects);
1485    }
1486
1487    @Override
1488    public void clear(Object... objects) {
1489        em.clear(objects);
1490    }
1491
1492    @Override
1493    public void lock(Object... objects) {
1494        em.lock(objects);
1495    }
1496
1497    @Override

```

```
78
79     public void setFname(String fname) {
80         this.fname = fname;
81     }
82
83     public String getLname() {
84         return lname;
85     }
86
87     public void setLname(String lname) {
88         this.lname = lname;
89     }
90
91     public int getAge() {
92         return age;
93     }
94
95     public void setAge(int age) {
96         this.age = age;
97     }
98
99     public long getContact() {
100        return contact;
101    }
102
103    public void setContact(long contact) {
104        this.contact = contact;
105    }
106
107    public String getEmail() {
108        return email;
109    }
110
111    public void setEmail(String email) {
112        this.email = email;
113    }
114
115    public List<Flight> getFlight() {
116        return flight;
117    }
118
```

```
119     public void setFlight(List<Flight> flight) {  
120         this.flight = flight;  
121     }  
122  
123 }  
124
```

```
1 package Controller;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5 import java.sql.SQLException;
6 import java.util.List;
7
8 import javax.servlet.annotation.WebServlet;
9 import javax.servlet.Dispatcher;
10 import javax.servlet.ServletException;
11 import javax.servlet.http.HttpServlet;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14
15 import models.Flight;
16 import models.Passenger;
17 import transferobjectaccess.FlightDAO;
18 import transferobjectaccess.PassengerDAO;
19
20 @WebServlet("/")
21 public class MasterServlet extends HttpServlet {
22
23     private FlightDAO ob;
24     private Flight f;
25     private PassengerDAO ob1;
26     private Passenger p;
27     private int num;
28     private String pd = null;
29     private int count, flag;
30     private List<Passenger> list1;
31
32     private static final long serialVersionUID = 1L
33 ;
34
35     public void init() {
36
37         ob = new FlightDAO();
38         ob1 = new PassengerDAO();
39         pd = "admin";
40         count = 0;
41         flag = 0;
```

```
41         list1 = null;
42
43     }
44
45     public MasterServlet() {
46         count = 0;
47         list1 = null;
48     }
49
50     @Override
51     public void service(HttpServletRequest request
52         , HttpServletResponse response) throws
53         ServletException, IOException {
54
55         String action = request.getServletPath();
56         try {
57             switch (action) {
58
59                 case "/add":
60                     showNewForm(request, response);
61                     break;
62                 case "/delete":
63                     deleteDetails(request, response);
64                     break;
65                 case "/edit":
66                     EditDetails(request, response);
67                     break;
68                 case "/insert":
69                     insertDetails(request, response);
70                     break;
71                 case "/update":
72                     updateDetails(request, response);
73                     break;
74                 case "/reset":
75                     changePassword(request, response);
76                     break;
77                 case "/register":
78                     register(request, response);
79                     break;
80                 case "/storage":
81                     storage(request, response);
```

```
80                     break;
81             case "/find":
82                 getflightDetailsById(request,
83                                         response);
84                     break;
85             case "/login":
86                 login(request, response);
87                     break;
88             case "/passenger":
89                 passengerFlightDetails(request,
90                                         response);
91                     break;
92             case "/insertPassenger":
93                 count++;
94                 passengerInsertDetails(request,
95                                         response);
96                     break;
97             case "/deletePassenger":
98                 count--;
99                 passengerDeleteDetails(request,
100                                         response);
101                     break;
102             case "/booking":
103                 BookingDetails(request, response);
104                     break;
105     }
106 } catch (Exception e) {
107     e.printStackTrace();
108 }
```

109 }

110 }

111

112 private void BookingDetails(HttpServletRequest
request, HttpServletResponse response)
113 throws ServletException, IOException {
114
115 List<Passenger> list = ob1.getAllDetails

```
115 () ;  
116         list1 = list ;  
117         request.setAttribute("list", list) ;  
118         RequestDispatcher rd = request.  
119             getRequestDispatcher("RegisterPage.jsp") ;  
120             rd.forward(request, response) ;  
121     }  
122  
123     private void passengerInsertDetails(  
124         HttpServletRequest request, HttpServletResponse  
125         response) throws IOException {  
126         if (count > num)  
127             response.sendRedirect("HomePage.jsp") ;  
128         else {  
129             String fname = request.getParameter("  
129             fname") ;  
130             String lname = request.getParameter("  
130             lname") ;  
131             long contact = Long.parseLong(request.  
131             getParameter("contact")) ;  
132             int age = Integer.parseInt(request.  
132             getParameter("age")) ;  
133             String email = request.getParameter("  
133             email") ;  
134             Passenger p = new Passenger(fname,  
134             lname, age, contact, email) ;  
135             ob1.insertPassengerInDB(p) ;  
136             response.sendRedirect("booking") ;  
137     }  
138 }  
139  
140     private void passengerDeleteDetails(  
141         HttpServletRequest request, HttpServletResponse  
142         response) throws IOException {  
143         int id = Integer.parseInt(request.  
143             getParameter("id")) ;  
143         Passenger p = ob1.getPassengerById(id) ;
```

```
144         ob1.deletePassenger(p);
145         response.sendRedirect("booking");
146     }
148
149     private void EditDetails(HttpServletRequest
150                             request, HttpServletResponse response)
150                             throws SQLException, ServletException
150                             , IOException {
151
152         int id = Integer.parseInt(request.
152                                     getParameter("fid"));
153         Flight f = ob.getFlightById(id);
154         RequestDispatcher dispatcher = request.
154                                     getRequestDispatcher("AddFlight.jsp");
155         request.setAttribute("f", f);
156         dispatcher.forward(request, response);
157
158     }
159
160     private void updateDetails(HttpServletRequest
160                               request, HttpServletResponse response)
160                               throws SQLException, IOException {
162
163         int fid = Integer.parseInt(request.
163                                     getParameter("fid"));
164         int fnumber = Integer.parseInt(request.
164                                     getParameter("fnumber"));
165         String fname = request.getParameter("fname");
166         String forigin = request.getParameter("forigin");
167         String ftarget = request.getParameter("ftarget");
168         String date = request.getParameter("fdate");
169         float fprice = Float.parseFloat(request.
169                                     getParameter("fprice"));
170         Flight fl = new Flight(fid, fnumber, fname,
170                               forigin, ftarget, date, fprice);
171         ob.updateFlight(fl);
```

```
172         response.sendRedirect("view");
173     }
175
176     private void deleteDetails(HttpServletRequest
177 req, HttpServletResponse resp) throws IOException
178     {
179         int id = Integer.parseInt(req.getParameter
180 ("fid"));
181         Flight f = ob.getFlightById(id);
182         ob.deleteFlight(f);
183         resp.sendRedirect("view");
184     }
185     private void showAllDetails(
186     HttpServletResponse response, HttpServletRequest
187 request)
188         throws ServletException, IOException {
189
190         List<Flight> list = ob.getAllDetails();
191         System.out.println(list);
192         request.setAttribute("list", list);
193         RequestDispatcher rd = request.
194 getRequestDispatcher("FlightDetails.jsp");
195         rd.forward(request, response);
196     }
197     private void showNewForm(HttpServletRequest
198 request, HttpServletResponse response)
199         throws ServletException, IOException {
200
201         RequestDispatcher rd = request.
202 getRequestDispatcher("AddFlight.jsp");
203         rd.forward(request, response);
204     }
205     private void insertDetails(HttpServletRequest
```

```

204 request, HttpServletResponse response) throws
205     IOException {
206         int fnumber = Integer.parseInt(request.
207             getParameter("fnumber"));
208         String fname = request.getParameter("fname");
209         String forigin = request.getParameter("forigin");
210         String ftarget = request.getParameter("ftarget");
211         String date = request.getParameter("fdate");
212         float fprice = Float.parseFloat(request.
213             getParameter("fprice"));
214         Flight fl = new Flight(fnumber, fname,
215             forigin, ftarget, date, fprice);
216         ob.insertFlightInDB(fl);
217         response.sendRedirect("view");
218     }
219     private void passengerFlightDetails(
220         HttpServletRequest request, HttpServletResponse
221         response)
222             throws ServletException, IOException {
223         String origin = request.getParameter("origin");
224         String target = request.getParameter("target");
225         String date = request.getParameter("date");
226         num = Integer.parseInt(request.
227             getParameter("qty"));
228         List<Flight> list = ob1.
229             getAllDetailsByOriginDate(origin, date, target);
230         // System.out.println(list);
231         // request.setAttribute("num", num);
232         request.setAttribute("Selectedlist", list

```

```
229 );
230         RequestDispatcher rd = request.
231             getRequestDispatcher("PassengerFlights.jsp");
232             rd.forward(request, response);
233 }
234
235     private void getflightDetailsById(
236         HttpServletRequest request, HttpServletResponse
237         response)
238             throws ServletException, IOException {
239
240         int id = Integer.parseInt(request.
241             getParameter("fid"));
242         FlightDAO x = new FlightDAO();
243         f = x.getFlightById(id);
244         // request.setAttribute("num", num);
245         RequestDispatcher rd = request.
246             getRequestDispatcher("booking");
247             rd.forward(request, response);
248
249     }
250
251     private void register(HttpServletRequest
252         request, HttpServletResponse response)
253             throws ServletException, IOException {
254
255         if (count != 0) {
256             ob.relation(f, list1);
257             request.setAttribute("n", num);
258             request.setAttribute("f", f);
259             // request.setAttribute("p", p);
260             RequestDispatcher rd = request.
261                 getRequestDispatcher("SummaryPage.jsp");
262                 rd.forward(request, response);
263             } else
264                 response.sendRedirect("HomePage.jsp");
265
266     }
267
268     private void storage(HttpServletRequest
```

```
262 request, HttpServletResponse response) throws
263     IOException {
264         ob1.insertPassengerInDB(p);
265         response.sendRedirect("HomePage.jsp");
266     }
267 }
268
269     private void changePassword(HttpServletRequest
270         request, HttpServletResponse response)
271             throws IOException, ServletException {
272
273         String newpwd = request.getParameter("newpwd");
274         String confpwd = request.getParameter("confpwd");
275
276         if (newpwd.compareTo(confpwd) == 0) {
277             pd = newpwd;
278             response.sendRedirect("AdminLogin.jsp");
279         } else {
280             RequestDispatcher rd = request.
281                 getRequestDispatcher("ResetPage.jsp");
282             PrintWriter out = response.getWriter
283             ();
284             rd.include(request, response);
285             out.println("<center> <span style='
286             color:red'> Invalid Credentials!!! </span></center
287             >");
288
289         }
290     }
291
292     private void login(HttpServletRequest request
293         , HttpServletResponse response) throws
294         ServletException, IOException {
295
296         String email = request.getParameter("email
297         ");
298         String pwd = request.getParameter("pwd");
```

```
291
292         RequestDispatcher rd = null;
293
294         if (email.equalsIgnoreCase("admin@test.com
295             ") && pwd.equals(pd)) {
295             rd = request.getRequestDispatcher("view");
296             rd.forward(request, response);
297         } else {
298             rd = request.getRequestDispatcher("AdminLogin.jsp");
299             PrintWriter out = response.getWriter
300             ();
300             rd.include(request, response);
301             out.println("<center> <span style='
301             color:red'> Invalid Credentials!!! </span></center
302             >");
302
303
304     }
305
306 }
307
```

```
1 package Configuration;
2
3 import java.util.Properties;
4
5 import org.hibernate.SessionFactory;
6 import org.hibernate.boot.registry.
    StandardServiceRegistryBuilder;
7 import org.hibernate.cfg.Configuration;
8 import org.hibernate.cfg.Environment;
9 import org.hibernate.service.ServiceRegistry;
10
11 import models.Flight;
12 import models.Passenger;
13
14 public class HibernateConfig {
15
16     private static SessionFactory sFactory;
17
18     public static SessionFactory getSessionFactory()
19     {
20         if(sFactory == null) {
21             try {
22                 Configuration cfg = new
23                     Configuration();
24
25                 Properties settings = new
26                     Properties();
27
28                 settings.put(Environment.DRIVER, "com.mysql.cj.jdbc.Driver");
29                 settings.put(Environment.URL, "jdbc:mysql://localhost:3306/demo?useSSL=false");
30                 settings.put(Environment.USER, "root");
31                 settings.put(Environment.PASS, "cisco");
32
33                 settings.put(Environment.DIALECT, "org.hibernate.dialect.MySQL5Dialect");
34                 settings.put(Environment.SHOW_SQL,
35                     "true");
36
37                 settings.put(Environment.
38                     CURRENT_SESSION_CONTEXT_CLASS, "thread");
39
40             }
41         }
42     }
43 }
```

```
31          //settings.put(Environment.  
32          HBM2DDL_AUTO, "update");  
33          settings.put(Environment.  
34          HBM2DDL_AUTO, "create");  
35  
36          cfg.setProperties(settings);  
37          cfg.addAnnotatedClass(Flight.class  
38 );  
39          cfg.addAnnotatedClass(Passenger.  
40          class);  
41          ServiceRegistry servReg = new  
42          StandardServiceRegistryBuilder()  
43          .applySettings(cfg.  
44          getProperties()).build();  
45          sFactory = cfg.buildSessionFactory(  
46          servReg);  
47          } catch (Exception e) {  
48              e.printStackTrace();  
49          }  
50      }  
51      return sFactory;  
52  }  
53 }  
54 }
```

```
1 package transferobjectaccess;
2
3 import java.util.List;
4
5 import org.hibernate.Session;
6 import org.hibernate.Transaction;
7
8 import Configuration.HibernateConfig;
9 import models.Flight;
10 import models.Passenger;
11
12 public class FlightDAO {
13
14     public void insertFlightInDB(Flight f) {
15
16         Transaction transaction = null;
17         Session dbSession = null;
18         try {
19             dbSession = HibernateConfig.
19             getSessionFactory().openSession();
20             transaction = dbSession.
20             beginTransaction();
21             dbSession.save(f);
22             transaction.commit();
23         }catch (Exception e) {
24             if(transaction != null) {
25                 transaction.rollback();
26             }
27             e.printStackTrace();
28         }finally {
29             dbSession.close();
30         }
31
32     }
33
34     @SuppressWarnings("unchecked")
35     public List<Flight> getAllDetails() {
36
37         Session dbSession = null;
38         List<Flight> list= null;
39         try {
```

```
40             dbSession = HibernateConfig.  
41                 getSessionFactory().openSession();  
42                 list = dbSession.createQuery("from  
43 Flight").list();  
44             }  
45         }catch (Exception e) {  
46             e.printStackTrace();  
47         }  
48     return list;  
49  
50 }  
51  
52 public Flight getFlightById(int flightId) {  
53  
54     Transaction transaction = null;  
55     Session dbSession = null;  
56     Flight f = null;  
57     try {  
58         dbSession = HibernateConfig.  
59             getSessionFactory().openSession();  
60             transaction = dbSession.  
61         beginTransaction();  
62         f = dbSession.get(Flight.class,  
63 flightId);  
64  
65     }catch (Exception e) {  
66         if(transaction != null) {  
67             transaction.rollback();  
68         }  
69         e.printStackTrace();  
70     }  
71     finally {  
72         dbSession.close();  
73     }  
74     return f;  
75 }
```

```
76         Transaction transaction = null;  
77         Session dbSession = null;  
78         try {  
79             dbSession = HibernateConfig.  
80                 getSessionFactory().openSession();  
81             transaction = dbSession.  
82                 beginTransaction();  
83             dbSession.update(f);  
84             transaction.commit();  
85         }catch (Exception e) {  
86             if(transaction != null) {  
87                 transaction.rollback();  
88             }  
89             e.printStackTrace();  
90         }finally {  
91             dbSession.close();  
92         }  
93     }  
94  
95     public Flight deleteFlight(Flight f) {  
96  
97         Transaction transaction = null;  
98         Session dbSession = null;  
99         try {  
100             dbSession = HibernateConfig.  
101                getSessionFactory().openSession();  
102             transaction = dbSession.  
103                 beginTransaction();  
104             dbSession.delete(f);  
105             transaction.commit();  
106         }catch (Exception e) {  
107             if(transaction != null) {  
108                 transaction.rollback();  
109             }  
110             e.printStackTrace();  
111         }finally {  
112             dbSession.close();  
113         }  
114     }
```

```
113
114      }
115
116      public void relation(Flight f, List<Passenger
> list) {
117
118          Transaction transaction = null;
119          Session dbSession = null;
120          try {
121              dbSession = HibernateConfig.
122                  getSessionFactory().openSession();
123              transaction = dbSession.
124                  beginTransaction();
125              f.setPassenger(list);
126              dbSession.save(f);
127              transaction.commit();
128          }catch (Exception e) {
129              if(transaction != null) {
130                  transaction.rollback();
131              }
132              e.printStackTrace();
133          }finally {
134              dbSession.close();
135          }
136
137      }
138
```

```
1 package transferobjectaccess;
2
3 import java.util.List;
4
5 import org.hibernate.Session;
6 import org.hibernate.Transaction;
7 import org.hibernate.query.Query;
8
9 import Configuration.HibernateConfig;
10 import models.Flight;
11 import models.Passenger;
12
13 public class PassengerDAO {
14
15     @SuppressWarnings("unchecked")
16     public List<Flight> getAllDetailsByOriginDate(
17         String origin, String date, String target) {
18
19         Transaction transaction = null;
20         Session dbSession = null;
21         List<Flight> list = null;
22         try {
23             dbSession = HibernateConfig.
24                 getSessionFactory().openSession();
25             transaction = dbSession.
26                 beginTransaction();
27             @SuppressWarnings("rawtypes")
28             Query query = dbSession
29                     .createQuery("from Flight f
30             where f.origin = :origin and f.target = :target and
31             f.dob = :date");
32             query.setParameter("origin", origin);
33             query.setParameter("target", target);
34             query.setParameter("date", date);
35             list = query.list();
36         } catch (Exception e) {
37             if (transaction != null) {
38                 transaction.rollback();
39             }
40             e.printStackTrace();
41         } finally {
```

```
37             dbSession.close();
38         }
39         return list;
40     }
41
42
43     public void insertPassengerInDB(Passenger p) {
44
45         Transaction transaction = null;
46         Session dbSession = null;
47         try {
48             dbSession = HibernateConfig.
49                 getSessionFactory().openSession();
50             transaction = dbSession.
51                 beginTransaction();
52             dbSession.save(p);
53             transaction.commit();
54         } catch (Exception e) {
55             if (transaction != null) {
56                 transaction.rollback();
57             }
58             e.printStackTrace();
59         } finally {
60             dbSession.close();
61         }
62
63     @SuppressWarnings("unchecked")
64     public List<Passenger> getAllDetails() {
65
66         Session dbSession = null;
67         List<Passenger> list = null;
68         try {
69             dbSession = HibernateConfig.
70                 getSessionFactory().openSession();
71             list = dbSession.createQuery("from
72             Passenger").list();
73         } catch (Exception e) {
74             e.printStackTrace();
75         }
76     }
77 }
```

```
74         } finally {
75             dbSession.close();
76         }
77         return list;
78     }
79
80
81     public Passenger getPassengerById(int id) {
82
83         Transaction transaction = null;
84         Session dbSession = null;
85         Passenger p = null;
86         try {
87             dbSession = HibernateConfig.
88                 getSessionFactory().openSession();
89             transaction = dbSession.
90                 beginTransaction();
91             p = dbSession.get(Passenger.class, id
92 );
93         }catch (Exception e) {
94             if(transaction != null) {
95                 transaction.rollback();
96             }
97             e.printStackTrace();
98         }finally {
99             dbSession.close();
100        }
101
102        public Passenger deletePassenger(Passenger p
103 ) {
104
105         Transaction transaction = null;
106         Session dbSession = null;
107         try {
108             dbSession = HibernateConfig.
109                 getSessionFactory().openSession();
110             transaction = dbSession.
111                 beginTransaction();
```

```
109             dbSession.delete(p);
110             transaction.commit();
111         }catch (Exception e) {
112             if(transaction != null) {
113                 transaction.rollback();
114             }
115             e.printStackTrace();
116         }finally {
117             dbSession.close();
118         }
119         return p;
120     }
122
123 }
124
```

```
1 <%@ page language="java" contentType="text/html;
  charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="ISO-8859-1">
7 <title>FLYAWAY.COM</title>
8 </head>
9 <body>
10    <div class="container" align="center">
11        <a href="AdminLogin.jsp">Admin</a> <br>
12        <br>
13        <form action="passenger" method="post">
14            <table border="2" cellpadding="5"
15              position="bottom">
16                <tr>
17                    <th>Origin :</th>
18                    <td><input type="text" name="
  origin" size="45" required>
19                      </td>
20                </tr>
21                <tr>
22                    <th>Destination :</th>
23                    <td><input type="text" name="
  target" size="45" required>
24                      </td>
25                </tr>
26                <tr>
27                    <th>Boarding Date :</th>
28                    <td><input type="date" name="
  date" size="45" required>
29                      </td>
30                </tr>
31                <tr>
32                    <th>Persons :</th>
33                    <td><input type="number" name="
  qty" size="45" required>
34                      </td>
35                </tr>
            </table>
```

```
36          <input type="submit" value="Search">
37      </form>
38  </div>
39 </body>
40 </html>
```

```
1 <%@ page language="java" contentType="text/html;
   charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1" isELIgnored="false"%>
3
4 <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
   prefix="c"%>
5
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="ISO-8859-1">
10 <title>FLYAWAY.COM</title>
11 </head>
12 <body>
13     <div align="center">
14         <c:if test="${f != null}">
15             <form action="update" method="post">
16             </c:if>
17             <c:if test="${f == null}">
18                 <form action="insert" method="post">
19             </c:if>
20             <table border="1" cellpadding="5">
21                 <c:if test="${f != null}">
22                     <input type="hidden" name="fid"
23                         value=<c:out value='${f.
flightId}' />" />
24                 </c:if>
25                 <tr>
26                     <th>Flight Number :</th>
27                     <td><input type="number" name="
fnumber" size="45"
28                         value=<c:out value='${f.
flightNumber}' />" required /></td>
29                 </tr>
30                 <tr>
31                     <th>Airline :</th>
32                     <td><input type="text" name="fname"
size="45"
33                         value=<c:out value='${f.
airline}' />" required /></td>
34                 </tr>
```

```
35          <tr>
36              <th>Origin :</th>
37              <td><input type="text" name="
38                  forigin" size="45"
39                  value=<c:out value='${f.origin
39 }' />" required /></td>
40          </tr>
41          <tr>
42              <th>Destination :</th>
43              <td><input type="text" name="
44                  ftarget" size="45"
45                  value=<c:out value='${f.target
45 }' />" required /></td>
46          </tr>
47          <tr>
48              <th>Flight Date :</th>
49              <td><input type="date" name="fdate"
50                  size="45"
51                  value=<c:out value='${f.dob
51 }' />" required /></td>
52          </tr>
53          <tr>
54              <th>Ticket Price :</th>
55              <td><input type="number" name="
56                  fprice" size="45"
57                  value=<c:out value='${f.price
57 }' />" required /></td>
58          </tr>
59      </table>
60      </form>
61  </div>
62 </body>
63 </html>
```

```
1 <%@ page language="java" contentType="text/html;
  charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="ISO-8859-1">
7 <title>FLYAWAY.COM</title>
8 </head>
9 <body>
10    <div align="center">
11        <h2>Reset Password</h2>
12        &nbsp;&nbsp;&nbsp;
13        <br><br>
14        <form action="reset" method="post">
15            <table border="2" cellpadding="5"
16              position="bottom">
17                <tr>
18                    <th>Enter new Password :</th>
19                    <td><input type="password" name
="newpwd" size="45" required>
20                    </td>
21                </tr>
22                <tr>
23                    <th>Confirm Password :</th>
24                    <td><input type="password" name
="confpwd" size="45" required>
25                    </td>
26                </tr>
27            </table>
28            <input type="submit" value="Save">
29        </form>
30    </div>
31 </body>
32 </html>
```

```
1 <%@ page language="java" contentType="text/html;
  charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1" %>
3
4
5
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="ISO-8859-1">
10 <title>FLYAWAY.COM</title>
11 </head>
12 <body>
13     <div align="center">
14         <h2>Admin Login</h2>
15         &nbsp;&nbsp;&nbsp;
16         <br><br>
17         <form action="login" method="post">
18             <table border="2" cellpadding="5"
19               position="bottom">
20                 <tr>
21                     <th>Email :</th>
22                     <td><input type="email" name="
23 email" size="45" required>
24                         </td>
25                 </tr>
26                 <tr>
27                     <th>Password :</th>
28                     <td><input type="password" name=
29 "pwd" size="45" required>
30                         </td>
31                 </tr>
32             </table>
33             <input type="submit" value="Login">
34         </form>
35     </div>
36 </body>
37 </html>
```

```
1 <%@ page language="java" contentType="text/html;
  charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1" isELIgnored="false"%>
3
4 <%@taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c"%>
5
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="ISO-8859-1">
10 <title>FLYAWAY.COM</title>
11 </head>
12 <body>
13     <div align="center">
14         &nbsp;&nbsp;&nbsp;
15         <table border="2" cellpadding="5" position="
  bottom">
16             <tr>
17                 <th>Payment Method :</th>
18                 <td><a href="#">PayPal</a></td>
19                 <td><a href="#">PayTm</a></td>
20                 <td><a href="#">Debit/Credit Cards
  </a></td>
21             </tr>
22         </table>
23         <br><br>
24         <a href="HomePage.jsp">Back To Home</a>
25     </div>
26 </body>
27 </html>
```

```
1 <%@ page language="java" contentType="text/html;
  charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1" isELIgnored="false"%>
3
4 <%@taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c"%>
5
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="ISO-8859-1">
10 <title>FLYAWAY.COM</title>
11 </head>
12 <body>
13     <div align="center">
14         <h2>Summary Details</h2>
15         &nbsp;&nbsp;&nbsp;
16         <form action="PaymentPage.jsp" method="post"
17 " >
18             <table border="1">
19                 <tr>
20                     <th>Flight Number :</th>
21                     <td><c:out value="${f.
22           flightNumber}" /></td>
23                     </tr>
24                     <tr>
25                         <th>Flight Name :</th>
26                         <td><c:out value="${f.
27           airline}" /></td>
28                         </tr>
29                     <tr>
30                         <th>Flight From :</th>
31                         <td><c:out value="${f.
32           origin}" /></td>
33                     </tr>
34                     <tr>
```

```
35 <th>Flight Boarding Date
36 </th>
37 <td><c:out value="${f.dob}"
38 /></td>
39 </tr>
40 <tr>
41 <th>Ticket Price :</th>
42 <td><c:out value="${f.price
43 * n}" /></td>
44 </tr>
45 </table>
46 <input type="submit" value="Payment">
47 </form>
48 </div>
49 </body>
50 </html>
```

```

1 <%@ page language="java" contentType="text/html;
  charset=ISO-8859-1"
2   pageEncoding="ISO-8859-1" isELIgnored="false"
  %>
3
4 <%@taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c" %>
5
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="ISO-8859-1">
10 <title>FLYAWAY.COM</title>
11 </head>
12 <body>
13   <div align="center">
14     <c:if test="${p != null}">
15       <form action="updatePassenger" method="
  post">
16       </c:if>
17       <c:if test="${p == null}">
18         <form action="insertPassenger" method="
  post">
19         </c:if>
20         <table border="1" cellpadding="5">
21           <c:if test="${f != null}">
22             <input type="hidden" name="id"
23               value=<c:out value='${p.pId
} ' />" />
24           </c:if>
25           <tr>
26             <th>First Name :</th>
27             <td><input type="text" name="fname"
28               size="45"
29               value=<c:out value='${p.fname
} ' />" required /></td>
30           </tr>
31           <tr>
32             <th>Last Name :</th>
             <td><input type="text" name="lname"
               size="45"

```

```
33          value=<c:out value='${p.lname}>
34      > required /></td>
35      </tr>
36      <tr>
37          <th>Contact :</th>
38          <td><input type="number" name="
39 contact" size="45"
40          value=<c:out value='${p.
41 contact}' /> required /></td>
42      </tr>
43      <tr>
44          <th>Age :</th>
45          <td><input type="text" name="age"
46 size="45"
47          value=<c:out value='${p.age}>
48      > required /></td>
49      </tr>
50      <tr>
51          <td colspan="2" align="center"><
52 input type="submit"
53          value="Save" /></td>
54      </tr>
55      </table>
56      </div>
57 </body>
58 </html>
```

```
1 <%@ page language="java" contentType="text/html;
  charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1" isELIgnored="false"%>
3
4 <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c"%>
5
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="ISO-8859-1">
10 <title>FLYAWAY.COM</title>
11 </head>
12 <body>
13     <h1 align="center">Passenger Details</h1>
14     <h2 align="center">
15         <a href="AddPassenger.jsp">Add Passenger</a>
16         &nbsp;&nbsp;&nbsp;
17     </h2>
18     <div align="center">
19         <table border="1">
20             <caption>Passenger Details</caption>
21             <tr>
22                 <th>ID</th>
23                 <th>First Name</th>
24                 <th>Last Name</th>
25                 <th>Contact</th>
26                 <th>Age</th>
27                 <th>Email</th>
28             </tr>
29             <c:forEach var="p" items="${list}">
30                 <tr>
31                     <td><c:out value="${p.pId}">
32                         /></td>
33                     <td><c:out value="${p.fname}">
34                         /></td>
35                     <td><c:out value="${p.lname}">
36                         /></td>
37                     <td><c:out value="${p.contact}">
38                         /></td>
39                     <td><c:out value="${p.age}">
40                         /></td>
41             </tr>
42         </table>
43     </div>
44 </body>
45 </html>
```

```
34  /></td>
35          <td><c:out value="${p.email}">
36          /></td>
37          <td><a href="deletePassenger?id=<c:out value='${p.pId}' />">Delete</a></td>
38      </tr>
39      </c:forEach>
40  </table>
41  <a href="register">Confirm</a>
42 </body>
43 </html>
```

```
1 <%@ page language="java" contentType="text/html;
   charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1" isELIgnored="false"
3 %
4 <%@taglib uri="http://java.sun.com/jsp/jstl/core"
   prefix="c" %>
5
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="ISO-8859-1">
10 <title>FLYAWAY.COM</title>
11 </head>
12 <body>
13     <h1 align="center">Flight Management</h1>
14     <h2 align="center">
15         <a href="add">Add Flight</a>
16         &nbsp;&nbsp;&nbsp;
17         <a href="ResetPage.jsp">Change Password</a>
18         &nbsp;&nbsp;&nbsp;
19         <a href="HomePage.jsp">Logout</a>
20     </h2>
21     <div align="center">
22         <table border="1">
23             <caption>
24                 Flight Details
25             </caption>
26
27             <tr>
28                 <th>ID</th>
29                 <th>Number</th>
30                 <th>Air Name</th>
31                 <th>Origin</th>
32                 <th>Destination</th>
33                 <th>Date</th>
34                 <th>Booking Price</th>
35                 <th>Status</th>
36             </tr>
37             <c:forEach var="f" items="${list}">
38                 <tr>
```

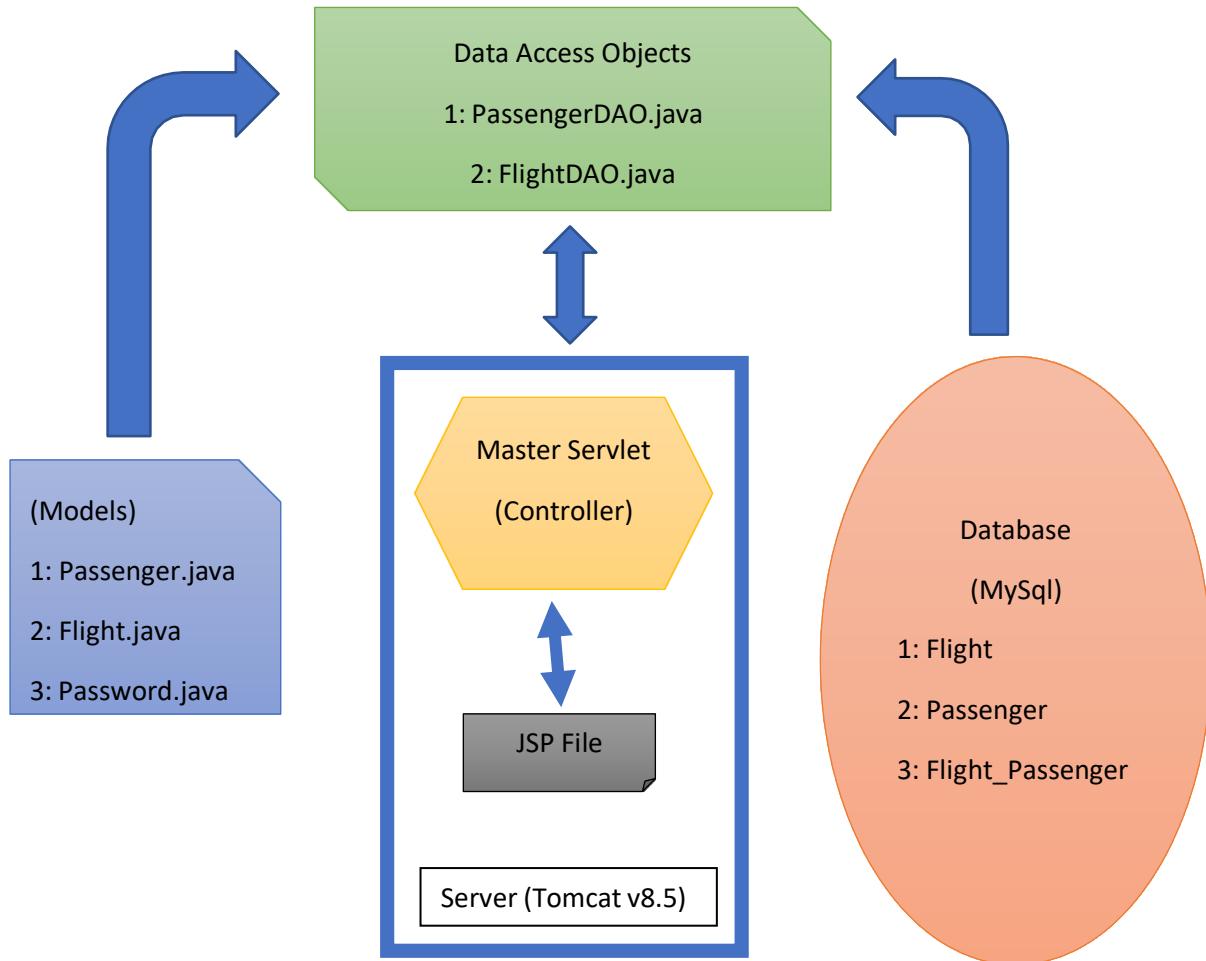
```
39 <td><c:out value="${f.flightId}" /></td>
40 <td><c:out value="${f.flightNumber}" /></td>
41 <td><c:out value="${f.airline}" /></td>
42 <td><c:out value="${f.origin}" /></td>
43 <td><c:out value="${f.target}" /></td>
44 <td><c:out value="${f.dob}" /></td>
45 <td><c:out value="${f.price}" /></td>
46 <td>
47 <a href="edit?fid=<c:out value = '${f.flightId}' />">Edit</a>
48 &nbsp;&nbsp;&nbsp;
49 <a href="delete?fid=<c:out value='${f.flightId}' />">Delete</a>
50 >
51 </td>
52 </tr>
53 </c:forEach>
54 </table>
55 </div>
56 </body>
57 </html>
```

```
1 <%@ page language="java" contentType="text/html;
  charset=ISO-8859-1"
2     pageEncoding="ISO-8859-1" isELIgnored="false"%>
3
4 <%@taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c"%>
5
6 <!DOCTYPE html>
7
8 <html>
9 <head>
10 <meta charset="ISO-8859-1">
11 <title>FLYAWAY.COM</title>
12 </head>
13 <body>
14     <div align="center">
15         <h2>Selected Flight Details</h2>
16         &nbsp;&nbsp;&nbsp;
17         <table border="1">
18             <tr>
19                 <th>ID</th>
20                 <th>Number</th>
21                 <th>Air Name</th>
22                 <th>Origin</th>
23                 <th>Destination</th>
24                 <th>Date</th>
25                 <th>Booking Price</th>
26                 <th>Status</th>
27             </tr>
28             <c:forEach var="f" items="${
  Selectedlist}">
29                 <tr>
30                     <td><c:out value="${f.flightId
  }" /></td>
31                     <td><c:out value="${f.
  flightNumber}" /></td>
32                     <td><c:out value="${f.airline}"
  /></td>
33                     <td><c:out value="${f.origin}"
  /></td>
34                     <td><c:out value="${f.target}"
```

```
34  /></td>
35          <td><c:out value="${f.dob}">
36          /></td>
37          <td><c:out value="${f.price}">
38          /></td>
39          <td><a href="find?fid=<c:out
value='${f.flightId}' />">Book Now</a></td>
40          </tr>
41      </c:forEach>
42  </table>
43 </div>
44 </body>
45 </html>
```

```
1 <!DOCTYPE web-app PUBLIC
2   "-//Sun Microsystems, Inc.//DTD Web Application 2.
3//EN"
4   "http://java.sun.com/dtd/web-app_2_3.dtd" >
5
6 <web-app>
7   <display-name>Archetype Created Web Application</
8     display-name>
9   <servlet>
10    <servlet-name>MasterServlet</servlet-name>
11    <display-name>MasterServlet</display-name>
12    <description></description>
13    <servlet-class>Controller.MasterServlet</
14      servlet-class>
15    </servlet>
16    <servlet-mapping>
17      <servlet-name>MasterServlet</servlet-name>
18      <url-pattern>/MasterServlet</url-pattern>
19    </servlet-mapping>
20    <welcome-file-list>
21      <welcome-file>HomePage.jsp</welcome-file>
22    </welcome-file-list>
23 </web-app>
```

Flow Diagram



- ❖ Core Concepts used in this project are mostly maven, hibernate, MySQL, jdbc connector, associations, java, CRUD operations in a database, web development.

Algorithm

Step 1> Start.

Step 2> Two options in the home page:

 Case 1: If user select “admin” section then go to step 3.

 Case 2: If user goes to passenger side through registering all booking details, then go step 7.

Step 3> Once a user select admin, it will prompt for admin email and admin password.

Step 4> An admin main page will be displayed containing three options and showing list of flights that admin has entered:

 Case 1: Add Flights -> step 5.

 Case 2: Change Password -> step 6.

Case 3: Logout and go back to step 2.

Step 5> A new window will be shown where admin can enter flight details and go back step 4.

Step 6> For changing password, it will ask for new and confirmation password from admin. Once it is validated correctly it will go back to home page.

Step 7> This will show a window having flights details that are filtered out through booking details. And the user has to select the flight for further action.

Step 8> Once, the user has selected the flight, user has to register his/her details and should be less than or equal to number of persons that user has specified.

Step 9> After continuation, it will show the summary of user's flight and will prompt the user for payment (Dummy Payment Gateway).

Step 10> Once payment is successful, it will take the user back to step 2.

Step 11> Stop

Conclusion

1: The prototype is robust and platform independent.

2: User can easily use the prototype and safely exit out of it.

3: As a developer, we can enhance it by introducing several new features such as guards along each web pages as currently its statically connected with each along with backend as will not allow to go back once admin has been logout, routing, custom validators and can have more user-friendly by adding styling (CSS, Bootstrap), custom loaders.

4: Though this prototype is tightly connected, the data will only persist in database until server is running and gets reset with each restarting of sever because of manual configuration of hibernate.

5: This prototype can also be implemented with multithreading to enable better performance.

6: And lastly, this prototype can be upgraded by implementing with securities patches to make it more versatile and secure in both local environment and global and later can be configured dynamically with connection of database through hibernate.