CMPE 275 Lab 2

REST API, Persistence, and Transactions

*Last updated:* ***04/04/2018***

In this lab, you build a REST API that can be used to implement a simple **airline reservation system** through. Your API provides *create*, *get*, *update*, *delete,* and *query* operations on flights, passengers, and reservations. This system needs to be hosted on Amazon EC2, Google Cloud Platform (GCP), including Compute Engine and App Engine, or any other cloud offering. You must use Spring’s @RestController for the implementation and use JPA for the persistence. You are encouraged to use Spring Boot as well. This is a group assignment with up to **two** members. You can read this Spring guide on [how to build a RESTful Web Service](https://spring.io/guides/gs/rest-service/).

Please design your data model to hold information for the airline reservation system. We define the following requirements and constraints:

* Each **passenger** can make one or more **reservations**. **Time overlap is not allowed** among any of his reservation.
* Each reservation may consist of one or more flights.
* Each **flight** can carry one or more passengers.
* Each flight uses one **plane**, which is an embedded object with four fields mapped to the corresponding four columns in the airline table.
* The total amount of passengers can not exceed the capacity of an plane.
* When a passenger is deleted, all reservation made by him are automatically canceled for him.
* A flight can not be deleted if it needs to carry at least one passenger.

**Incomplete** definitions of passenger, reservation, flight and plane are given below..

package edu.sjsu.cmpe275.lab2;

public class Passenger {

private String id;

private String firstname;

private String lastname;

private int age;

private String gender;

private String phone; // Phone numbers must be unique

...

}

public class Reservation {

private String reservationNumber;

private Passenger passenger;

private double price; // sum of each flight’s price.

private List<Flight> flights;

...

}

public class Flight {

private String flightNumber; // Each flight has a unique flight number.

private double price;

private String ~~from~~ origin;

private String to;

/\* Date format: **yy-mm-dd-hh,** do not include minutes and seconds.

\*\* Example: 2018-03-22-19

\*/ The system only needs to supports PST. You can ignore other time zones.

private Date departureTime;

private Date arrivalTime;

private int seatsLeft;

private String description;

private Plane plane; // Embedded

private List<Passenger> passengers;

...

}

public class Plane {

private int capacity;

private String model;

private String manufacturer;

private int year;

}

Your app, running in cloud, must be made accessible to the TA, through DNS (e.g., cmpe275-lab2-minisocial.appspot.com), or an IP address. There are five types of requests your app needs to support. For simplicity, no authentication or authorization is enforced for these requests. The specification below uses the ***hostname*** to represent your DNS or IP.

For the output encoding, you ***must*** use XML when “xml=true” is expected, otherwise JSON.

## (1) Get a passenger back as JSON

|  |  |
| --- | --- |
| URL | [https://hostname/](http://hostname/profile)passenger/id |
| Method | GET |
| Return | If the passenger can be found with the given ID, return the passenger's record in JSON format:  {  "passenger": {  "id": " 123 ",  "firstname": " John ",  "lastname": " Oliver ",  "age": " 21 ",  "gender": " male ",  "phone": " 4445556666 ",  "reservations": {  "reservation": [  {  "reservationNumber": "123",  "price": "240.00",  "flights": {  "flight": [  {  "number": " GH2Z1 ",  "price": "120.00",  "origin": "Seattle, WA",  "to": "San Jose, CA",  "departureTime": "2018-04-12-09 ",  "arrivalTime": "2018-04-12-14",  "description": "xxxx",  "plane": {  "capacity": "120",  "model": "Boeing 757",  "manufacturer": "Boeing",  "year": "1998"  }  },  {  "number": " HZ124 ",  "price": "120.00",  "origin": "San Jose, CA",  "to": "Seattle, WA",  "departureTime": "2018-04-14-09 ",  "arrivalTime": "2018-04-14-14",  "description": "xxxx",  "plane": {  "capacity": "120",  "model": "Boeing 757",  "manufacturer": "Boeing",  "year": "1998"  }  }  ]  }  },  {  "reservationNumber": "345",  "price": "100.00",  "flights": {  "flight": {  "number": " KJ124 ",  "price": "100.00",  "origin": "San Jose, CA",  "to": "Washton, DC",  "departureTime": "2018-04-15-09 ",  "arrivalTime": "2018-04-15-15",  "description": "xxxx",  "plane": {  "capacity": "100",  "model": "Boeing 757",  "manufacturer": "Boeing Airplanes",  "year": "1999"  }  }  }  }  ]  }  }  }  Otherwise, return:  {  "BadRequest": {  "code": " 404 ",  "msg": " Sorry, the requested passenger with id **XXX** does not exist"  }  }  Note: XXX is the ID specified in the request, and you **must return HTTP error code 404** as well. |
| Description | * This JSON is meant for consumption of APIs, and may not render well in browsers unless extensions/plugs are installed. |

## 

## (2) Get a passenger back as XML

|  |  |
| --- | --- |
| URL | [https://hostname/](http://hostname/profile)passenger/id?xml=true |
| Method | GET |
| Return | If the passenger can be found with the given ID, return the passenger's record in XML format:  <passenger>  <id> 123 </id>  <firstname> John </firstname>  <lastname> Oliver </lastname>  <age> 21 </age>  <gender> male </gender>  <phone> 4445556666 </phone>  <reservations>  <reservation>  <reservationNumber>123</reservationNumber>  <price>240.00</price>  <flights>  <flight>  <number> GH2Z1 </number>  <price>120.00</price>  <origin>Seattle, WA</origin>  <to>San Jose, CA</to>  <departureTime>  2018-04-12-09  </departureTime>  <arrivalTime>2018-04-12-14</arrivalTime>  <description>xxxx</description>  <plane>  <capacity>120</capacity>  <model>Boeing 757</model>  <manufacturer>Boeing</manufacturer>  <year>  1998  </year>  </plane>  </flight>  <flight>  <number> HZ124 </number>  <price>120.00</price>  <origin>San Jose, CA</origin>  <to>Seattle, WA</to>  <departureTime>  2018-04-14-09  </departureTime>  <arrivalTime>2018-04-14-14</arrivalTime>  <description>xxxx</description>  <plane>  <capacity>120</capacity>  <model>Boeing 757</model>  <manufacturer>Boeing</manufacturer>  <year>  1998  </year>  </plane>  </flight>  </flights>  </reservation>  <reservation>  <reservationNumber>345</reservationNumber>  <price>100.00</price>  <flights>  <flight>  <number> KJ124 </number>  <price>100.00</price>  <origin>San Jose, CA</origin>  <to>Washton, DC</to>  <departureTime>  2018-04-15-09  </departureTime>  <arrivalTime>2018-04-15-15</arrivalTime>  <description>xxxx</description>  <plane>  <capacity>100</capacity>  <model>Boeing 757</model>  <manufacturer>Boeing</manufacturer>  <year>  1999  </year>  </plane>  </flight>  </flights>  </reservation>  </reservations>  </passenger>  Otherwise return:  {  "BadRequest": {  "code": " 404 ",  "msg": " Sorry, the requested passenger with id **XXX** does not exist"  }  }  Note: XXX is the ID specified in the request, and you **must return HTTP error code 404** as well. |
| Description | This XML is meant for consumption of APIs, and may not render well in browsers unless extensions/plugs are installed. |

## (3) Create a passenger

|  |  |
| --- | --- |
| URL | [https://hostname/passenger?firstname=XX&lastname=YY&age=11&gender=famale&phone=123](https://hostname/user/userId?firstname=XX&lastname=YY&title=abc&street=AAA&city=BBB&state=CCC&zip=95012) |
| Method | POST |
| Return | If the passenger is created successfully, the request returns the newly created~~/updated~~ passenger in **Json**, the same as GET [https://hostname/](http://hostname/profile)passenger/id~~?json=true~~.  Otherwise, return proper HTTP **error code** and an error message of the following format  {  "BadRequest": {  "code": "400",  "msg": "xxx”  }  }  Note: xxx here is the failure reason; e.g., “another passenger with the same number already exists.” |
| Description | This request creates a passenger’s record in the system.   * For simplicity, all the passenger's fields including the phone number (firstname, lastname, age, and gender) are passed as query parameters, and you can assume the request always comes with all the fields specified. * The uniqueness of phone numbers must be enforced here. |

## (4) Update a passenger

|  |  |
| --- | --- |
| URL | [https://hostname/passenger/id?firstname=XX&lastname=YY&age=11&gender=famale&phone=123](https://hostname/user/userId?firstname=XX&lastname=YY&title=abc&street=AAA&city=BBB&state=CCC&zip=95012) |
| Method | PUT |
| Return | If the passenger is updated successfully, the request returns the newly updated passenger in **Json**, the same as GET [https://hostname/](http://hostname/profile)passenger/id.  Otherwise, return  {  "BadRequest": {  "code": "404 ",  "msg": "xxx"  }  } |
| Description | * This request updates a passenger’s record in the system. * For simplicity, all the passenger's fields including the phone number (firstname, lastname, age, and gender) are passed as query parameters, and you can assume the request always comes with all the fields specified. |

## (5) Delete a passenger

|  |  |
| --- | --- |
| URL | [https://hostname/passenger/id](http://hostname/profile/userId?firstname=XX&lastname=YY&...http://hostname/profile) |
| Method | DELETE |
| Return | If the passenger does not exist, return:  {  "BadRequest": {  "code": "404 ",  "msg": "Passenger with id XXX does not exist"  }  }  You **must return HTTP error code 404** as well.  Otherwise, return:  <Response>  <code> 200 </code>  <msg> Passenger with id XXX is deleted successfully </msg>  </Response>  Note: xxx here is the given ID in the request |
| Description | * This request deletes the user with the given user ID. * The reservation made by the passenger should also be deleted. * You must update the number of available seats for the involved flights. * All successful response will use the same XML format. |

## (6) Get a reservation back as JSON

|  |  |
| --- | --- |
| URL | [https://hostname/reservation](http://hostname/profile/userId?firstname=XX&lastname=YY&...http://hostname/profile)/number |
| Method | GET |
| Return | If the reservation can not be found with the given number, return:  {  "BadRequest": {  "code": " 404 ",  "msg": " Reserveration with number XXX does not exist "  }  }  You **must return HTTP error code 404** as well.  Otherwise, return:  {  "reservation": {  "reservationNumber": "123",  "price": "240.00",  "passenger": {  "id": " 123 ",  "firstname": " John ",  "lastname": " Oliver ",  "age": " 21 ",  "gender": " male ",  "phone": " 4445556666 "  },  "flights": {  "flight": [  {  "number": " GH2Z1 ",  "price": "120.00",  "origin": "Seattle, WA",  "to": "San Jose, CA",  "departureTime": "2018-04-12-09 ",  "arrivalTime": "2018-04-12-14",  "seatsLeft": "15",  "description": "xxxx",  "plane": {  "capacity": "120",  "model": "Boeing 757",  "manufacturer": "Boeing Airplanes",  "year": "1998"  }  },  {  "number": " HZ124 ",  "price": "120.00",  "origin": "San Jose, CA",  "to": "Seattle, WA",  "departureTime": "2018-04-14-09 ",  "arrivalTime": "2018-04-14-14",  "seatsLeft": "15",  "description": "xxxx",  "plane": {  "capacity": "120",  "model": "Boeing 757",  "manufacturer": "Boeing Airplanes",  "year": "1998"  }  }  ]  }  }  } |
| Description | * This JSON is meant for consumption of APIs, and may not render well in browsers unless extensions/plugs are installed. |

## 

## (7) Make a reservation

|  |  |
| --- | --- |
| URL | [https://hostname/reservation](http://hostname/profile/userId?firstname=XX&lastname=YY&...http://hostname/profile)?passengerId=XX&flightLists=AA,BB,CC |
| Method | POST |
| Return | If the reservation is created successfully, the request returns the newly created reservation’s record in **XML**, like:  <reservation>  <reservationNumber>123</reservationNumber>  <price>240.00</price>  <passenger>  <id> 123 </id>  <firstname> John </firstname>  <lastname> Oliver </lastname>  <age> 21 </age>  <gender> male </gender>  <phone> 4445556666 </phone>  </passenger>  <flights>  <flight>  <number> GH2Z1 </number>  <price>120.00</price>  <origin>Seattle, WA</origin>  <to>San Jose, CA</to>  <departureTime>2018-04-12-09 </departureTime>  <arrivalTime>2018-04-12-14</arrivalTime>  <seatsLeft>15</seatsLeft>  <description>xxxx</description>  <plane>  <capacity>120</capacity>  <model>Boeing 757</model>  <manufacturer>  Boeing Airplanes  </manufacturer>  <year>  1998  </year>  </plane>  </flight>  <flight>  <number> HZ124 </number>  <price>120.00</price>  <origin>San Jose, CA</origin>  <to>Seattle, WA</to>  <departureTime>2018-04-14-09 </departureTime>  <arrivalTime>2018-04-14-14</arrivalTime>  <seatsLeft>15</seatsLeft>  <description>xxxx</description>  <plane>  <capacity>120</capacity>  <model>Boeing 757</model>  <manufacturer>Boeing Airplanes</manufacturer>  <year>  1998  </year>  </plane>  </flight>  </flights>  </reservation>  Otherwise, return:  {  "BadRequest": {  "code": "400 ",  "msg": "xxx"  }  }  Note: xxx here is the failure reason, and you **must return HTTP error code 404** as well. |
| Description | * This request makes a reservation for a passenger. * Time-Overlap is not allowed for a certain passenger. * The total amount of passengers can not exceed the capacity of the reserved plane. * You would receive a list of flights as input. |

## 

## (8) Update a reservation

|  |  |
| --- | --- |
| URL | [https://hostname/reservation](http://hostname/profile/userId?firstname=XX&lastname=YY&...http://hostname/profile)/number?flightsAdded=AA,BB,CC&flightsRemoved=XX,YY |
| Method | POST |
| Return | If the reservation is updated successfully, the request returns the newly updated reservation in **Json**, the same as GET <https://hostname/reservation/number>.  Otherwise, return:  {  "BadRequest": {  "code": "404 ",  "msg": "xxx"  }  }  Note: xxx here is the failure reason, and you **must return HTTP error code 404** as well. |
| Description | * This request update a reservation by adding and/or removing some flights * If flightsAdded (or flightsRemoved) param exists, then its list of values cannot be empty. * **If both additions and removals exist, the non-overlapping constraint should not consider the flights to be removed.** * **Update a reservation triggers a recalculation of its total price by summing up the price of each contained flight.** |

## (9) Search for reservations

|  |  |
| --- | --- |
| URL | [https://hostname/reservation](http://hostname/profile/userId?firstname=XX&lastname=YY&...http://hostname/profile)?passengerId=XX&origin=YY&to=ZZ&flightNumber=GH2Z1 |
| Method | GET |
| Return | Return the search result in XML format:  <reservations>  <reservation>  <reservationNumber>123</reservationNumber>  <price>240.00</price>  <passenger>  <id> 123 </id>  <firstname> John </firstname>  <lastname> Oliver </lastname>  <age> 21 </age>  <gender> male </gender>  <phone> 4445556666 </phone>  </passenger>  <flights>  <flight>  <number> GH2Z1 </number>  <price>120.00</price>  <origin>Seattle, WA</origin>  <to>San Jose, CA</to>  <departureTime>  2018-04-12-09  </departureTime>  <arrivalTime>2018-04-12-14</arrivalTime>  <description>xxxx</description>  <plane>  <capacity>120</capacity>  <model>Boeing 757</model>  <manufacturer>  Boeing  </manufacturer>  <year>  1998  </year>  </plane>  </flight>  </flights>  </reservation>  <reservation>  <reservationNumber>345</reservationNumber>  <passenger>  <id> 234 </id>  <firstname> Emma </firstname>  <lastname> Latin </lastname>  <age> 20 </age>  <gender> female </gender>  <phone> 22312332 </phone>  </passenger>  <price>100.00</price>  <flights>  <number> GH2Z1 </number>  <price>120.00</price>  <origin>Seattle, WA</origin>  <to>San Jose, CA</to>  <departureTime>  2018-04-12-09  </departureTime>  <arrivalTime>2018-04-12-14</arrivalTime>  <description>xxxx</description>  <plane>  <capacity>120</capacity>  <model>Boeing 757</model>  <manufacturer>  Boeing  </manufacturer>  <year>  1998  </year>  </plane>  </flight>  </flights>  </reservation>  </reservations> |
| Description | * This request allow to search for reservations by any combination of single passenger ID, departing city, arrival city, and flight number * You can assume that at least one request parameter is specified |

## 

## (10) Cancel a reservation

|  |  |
| --- | --- |
| URL | [https://hostname/reservation](http://hostname/profile/userId?firstname=XX&lastname=YY&...http://hostname/profile)/number |
| Method | DELETE |
| Return | If the reservation does not exist, return:  {  "BadRequest": {  "code": " 404 ",  "msg": " Reservation with number XXX does not exist "  }  }  You **must return HTTP error code 404** as well.  Otherwise, return:  <Response>  <code> 200 </code>  <msg> Reservation with number XXX is canceled successfully </msg>  </Response>  Note: xxx here is the given number in the request |
| Description | * This request cancel a reservation for a passenger. * You need to update the number of available seats for the involved flight. |

## (11) Get a flight back as JSON

|  |  |
| --- | --- |
| URL | [https://hostname/](http://hostname/profile)flight/flightNumber |
| Method | GET |
| Return | The flight record with given flight number in JSON format.  {  "flight": {  "flightNumber": " HX837 ",  "price": "120.00",  "origin": " San Jose, CA ",  "to": " Seattle, WA ",  "departureTime": " 2018-03-12-09 ",  "arrivalTime": " 2018-03-12-14 ",  "description": " xxx ",  "seatsLeft": " 11 ",  "plane": {  "capacity": " 150 ",  "model": " Boeing 747 ",  "manufacturer": "Boeing Commercial Airplanes ",  "year": "1997"  },  "passengers": {  "passenger": [  {  "id": "123",  "firstname": " John ",  "lastname": " Oliver ",  "age": " 21 ",  "gender": " male ",  "phone": " 4445556666 "  },  {  "id": "234",  "firstname": " Ali ",  "lastname": " Swan ",  "age": " 30 ",  "gender": " female ",  "phone": " 444555777 "  }  ]  }  }  }  If the flight can not be found with the given number, return:  {  "BadRequest": {  "code": " 404 ",  "msg": " Sorry, the requested flight with number **XXX** does not exist"  }  }  You **must return HTTP error code 404** as well. |
| Description | This JSON is meant for consumption of APIs, and may not render well in browsers unless extensions/plugs are installed. |

## (12) Get a flight back as XML

|  |  |
| --- | --- |
| URL | [https://hostname/](http://hostname/profile)flight/flightNumber?xml=true |
| Method | GET |
| Return | The flight record with given flight number in XML format.  <flight>  <flightNumber> HX837 </flightNumber>  <price>120.00</price>  <origin> San Jose, CA </origin>  <to> Seattle, WA </to>  <departureTime> 2018-03-12-09 </departureTime>  <arrivalTime> 2018-03-12-14 </arrivalTime>  <description> xxx </description>  <seatsLeft> 11 </seatsLeft>  <plane>  <capacity> 150 </capacity>  <model> Boeing 747 </model>  <manufacturer>Boeing Commercial Airplanes </manufacturer>  <year>1997</year>  </plane>  <passengers>  <passenger>  <id>123</id>  <firstname> John </firstname>  <lastname> Oliver </lastname>  <age> 21 </age>  <gender> male </gender>  <phone> 4445556666 </phone>  </passenger>  <passenger>  <id>234</id>  <firstname> Ali </firstname>  <lastname> Swan </lastname>  <age> 30 </age>  <gender> female </gender>  <phone> 444555777 </phone>  </passenger>  </passengers>  </flight>  If the flight can not be found with the given number, return:  {  "BadRequest": {  "code": " 404 ",  "msg": " Sorry, the requested flight with number XXX does not exist"  }  }  You **must return HTTP error code 404** as well. |
| Description | * This XML is meant for consumption of APIs, and may not render well in browsers unless extensions/plugs are installed. |

## 

## (13) Create or update a flight

|  |  |
| --- | --- |
| URL | https://hostname/flight/flightNumber?price=120&origin=AA&to=BB&departureTime=CC&arrivalTime=DD&description=EE&capacity=GG&model=HH&manufacturer=II&year=1997 |
| Method | POST |
| Return | If the flight is created/updated successfully, it should return the newly created/updated flight in XML, the same as GET [https://hostname/](http://hostname/profile)flight/flightNumber?xml=true  Otherwise, return the appropriate error code, 400 or 404, and error message, e.g.,  {  "BadRequest": {  "code": " 404 ",  "msg": "xxx"  }  }  Note: xxx here is the failure reason, and you **must return HTTP error code** as well. |
| Description | * This request creates/updates a new flight for the system. * For simplicity, all the fields are passed as query parameters, and you can assume the request always comes with all the fields specified. * The corresponding flight should be created/updated accordingly. * You may need to update the seatsLeft when capacity is modified. * When attempting reduce the existing capacity of a flight, the request must fail with error code 400 if active reservation count for this flight is higher than the target capacity. * If change of a flight’s departure and/or arrival time causes a passenger to have **overlapping flight time**, this update cannot proceed and hence **fails** with error code 400. * Changing the price of a flight will not affect the total price of existing reservations. |

## 

## (14) Delete a flight

|  |  |
| --- | --- |
| URL | <https://hostname/airline/flightNumber> |
| Method | DELETE |
| Return | If the flight with the given flight number does not exist, return:  {  "BadRequest": {  "code": " 404 ",  "msg": " xxx "  }  }  Note, xxx here is the reason why you can not delete the flight. You **must return HTTP error code 404** as well.  Otherwise, return:  <Response>  <code> 200 </code>  <msg> Flight with number XXX is deleted successfully </msg>  </Response>  Note: xxx here is the given number in the request |
| Description | * This request deletes the flight for the given flight number. * You can not delete a flight that has one or more reservation, in which case, the deletion should fail with error code 400. |

## 

## Additional Requirements

1. All these operations should be **transactional**.
2. You must use **JPA** and persist the user data into a database. If you are on Amazon EC2, you need to use MySQL; For GCP, you can use either [Cloud Datastore](https://cloud.google.com/datastore/docs/), or [Cloud SQL](https://cloud.google.com/sql/docs/).
3. Please add proper **JavaDoc comments**.
4. You must keep your server running for at least three weeks upon submission. Once your code is submitted to Canvas, you cannot make any further deployment/upload to your app on the server, or it will be considered as late submission or even cheating. You may be asked to show the server log and deployment history upon the TA’s request.

## Submission

Please submit through Canvas. Do not include jars or compiled class files.

* This is a group assignment, even though you can be in your own group.
* You must include a file named readme.pdf, which contains:
  + Group member info
  + Cloud Service Choice: Amazon, GCP App Engine, GCP Compute Engine, etc.
  + Hostname (or IP address)
  + 28 screenshots, one for each properly formatted result returned by the requests, and one for the according error page. For each screenshot, you must also paste the URL text (clearly readable) right before your image, so that it’s handy for the TA to copy and paste as needed for grading purpose

## Grading

This lab has a total point of 10, with 9 points for correctness (including persistence, transaction, etc), and 1 point for documentation, code structure, and unit testing.

## Cheating Policy

If code similarity is detected through [JPlag](https://jplag.ipd.kit.edu), you automatically get zero points for this lab and run the risk of failing the class.

## Additional Info

1. Since SJSU is an AWS Educate Institution member, students should be able to sign up and apply for free credits at <https://aws.amazon.com/education/awseducate/members>
2. The free tier of GCP should be sufficient for this lab.