CMPE 275 Section 2
Lab 2 - MVC and Persistence

Last updated: 11/02/2016

In this lab, you build a mini web application to manage office phones and phone users through create, view, update, and delete. This application needs to be hosted in either Amazon EC2 or Google AppEngine. You must use Spring's MVC framework for the UI implementation, and use JPA for the persistence. This is a group assignment with up to two members.

Incomplete definitions of user, phone, and address are given below..

```
package edu.sjsu.cmpe275.lab2;

public class User {
    private String id;
    private String firstname;
    private String lastname;
    private String title;
    private Address address;
    Private List<Phone> phones;
    ...
}

public class Phone {
    private String id;
    private String number;  // Note, phone numbers must be unique
    private String description;
    private Address address;
    ...
}

public class Address {
    String street;
    String city;
    String state;
    String zip;
    ...
}
```

Relationship Between Objects

- Each phone can be assigned to multiple users, and each user can have multiple phones.
- Each phone (and user) has an address, which is an ***embedded*** object with four fields mapped to the corresponding four columns in the phone (or user) table.
- When a person is deleted, all phones assigned to him are automatically unassigned from him.
- A phone ***cannot*** be deleted if it is assigned to at least one user.

Your app, running in either Amazon EC2 or Google AppEngine, 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 need to support. For simplicity, no authentication or authorization is enforced for these requests. The specification below uses *hostname* to represent your DNS or IP.

(1) Get a user as HTML

URL: https://hostname/user/userId
Method: GET

This returns an HTML that renders the given user ID's user record. The user fields are part of an HTML form.

- Firstname, lastname, address, and title must be editable.
- List of assigned phones must be shown: you must show at least the phone numbers. The phones are NOT editable here.
- NO need to show the owners of the phones here.
- The HTML page should contain **two** buttons, *Update* and *Delete*. When Update is clicked, it updates the user as specified in (4), using HTTP POST. When the Delete button is created, it deletes the user using HTTP DELETE.
- It is up to you to decide the layout and look-and-feel of the rendered user record.
- If the user of the given user ID does not exist, a customized 404 HTML page with the message "Sorry, the requested user 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.

(2) Get a user back as JSON

URL: https://hostname/user/userId?json=true
Method: GET

This returns the given user's record in JSON format.

```
    {
        "id":"2",
        "firstname": "John",
        "lastname": "Oliver",
        "title": "Manager",
        "address": {
                "street": "1 Washington Square",
                "city": "San Jose",
                "state": "CA",
                "zip": "95012"
        },
        "phones" : [
        {"id":"100", number":"3231214567", "description":"home"},
        {"id":"101", "number":"3231210000", "description":"office"}
        ]
}
```

This JSON is meant for readonly, and is not an HTML page or form.

- The content you put in the JSON must match what you have in (1).
- All error handlings should be the same as the previous request in (1) as well.

(3) Get the user creation HTML

URL: https://hostname/user
Method: GET

This returns an HTML form that should be almost the **same** as (1), except that

- All fields including the ID should be initially empty and should be editable except the ID.
- The page should contain one button, labelled *Create*. When Create is clicked, it creates the user as specified in (4), using HTTP POST.
- The return code should follow the HTTP convention.

(4) Create or update a user

URL:
https://hostname/user/userId?firstname=XX&lastname=YY&title=abc&street=AAA&city=BBB&state=CCC&zip=95012

Method: POST

This request creates or update the user for the given user ID.

- For simplicity, all the user fields other than the ID (firstname, lastname, address, and title) are passed as query parameters, and you can assume the request always comes with all the fields specified.
- The corresponding user should be created/updated accordingly.
- In the end, the request returns the newly created/updated user in HTML, the same as GET https://hostname/user/userId

(5) Delete a user

URL: https://hostname/user/userId
Method: DELETE

This request deletes the user for the given user ID.

- If the user does not exist, it should return the same 404 page as in (1) with error code 404.
- Otherwise, delete the corresponding user, and redirect the request to the user creation page at https://hostname/user.
- The phones should also unassigned from the user.

(6) Get a Phone as HTML

URL: https://hostname/phone/phoneId
Method: GET

This returns an HTML that renders the phone of the given ID. The phone fields are part of an HTML form.

- Phone number, description, and address must be editable.
- List of assigned users must be shown: you must show at least the assigned users' names and IDs. You must also provide widgets to allow add/remove of users. Do NOT show other phones a user is assigned.
- The HTML page should also contain **two** buttons, *Update* and *Delete*. When Update is clicked, it updates the phone as specified in (4), using HTTP POST. When the Delete button is created, it deletes the phone using HTTP DELETE.
- A phone cannot be deleted if there is still a user assigned to it: a 400 error should be returned for such request.
- It is up to you to decide the layout and look-and-feel of the rendered phone record.
- If the phone of the given phone ID does not exist, a customized 404 HTML page with the message "Sorry, the requested phone 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.

(7) Get a phone back as JSON

URL: https://hostname/phone/phoneId?json=true
Method: GET

This returns the given phone's record in JSON format.

```
        {
                "id":"23",
                "number": "1234567890",
                "description": "home",
                "address": {
                        "street": "San Salvador",
                        "city": "San Jose",
                        "state": "CA",
                        "zip": "95012"
                },
                "users" : [
                {"id":"2", "firstname":"John", lastname: "Oliver"},
                {"id":"21", "firstname":"John", lastname: "Mayer"}
                ]
        }
```

This JSON is meant for readonly, and is not an HTML page or form.

- The content you put in the JSON must match what you have in (6).
- All error handlings should be the same as the previous request in (6) as well.

(8) Get the phone creation HTML

URL: https://hostname/phone
Method: GET

This returns an HTML form that should be almost the **same** as (1), except that

- All fields including the ID should be initially empty and should be editable except the ID.
- The page should contain one button, labelled *Create*. When Create is clicked, it creates the phone as specified in (4), using HTTP POST.
- The return code should follow the HTTP convention.

(9) Create or update a phone

URL:
https://hostname/phone/phoneId?number=XX&description=YY&street=AAA&city=BBB&state=CCC&zip=95012&users[]=id1&users[]=id2
Method: POST

This request creates or update the phone for the given phone ID.

- For simplicity, all the phone fields other than the ID (number and description) are passed as query parameters, and you can assume the request always comes with all the fields specified. The users assignments must be taken care of too. (Users are assigned to a phone using UserId, refer the URL above. Only existing users can be assigned to a phone.)
- The corresponding phone should be created/updated accordingly.
- In the end, the request returns the newly created/updated phone in HTML, the same as GET https://hostname/phone/phoneId

(10) Delete a phone

URL: https://hostname/phone/phoneId

Method: DELETE

This request deletes the phone for the given phone ID.

- If the phone does not exist, it should return the same 404 page as in (1) with error code 404.
- A phone cannot be deleted if it's still assigned, or 400 errors will be returned.
- Otherwise, delete the corresponding phone, and redirect the request to the phone creation page at https://hostname/phone.
- The phones should also unassigned from the user.

Additional Requirements

a. You must follow the MVC design pattern and use Spring's MVC framework; particularly, (1) and (2) should share the same model, even though they implement different views.
b. All the 10 operations should be transactional.
c. You must use JPA and persist the user data into a database. If you are on Amazon EC2, you need to use MySQL; For Google AppEngine, you can use either the built-in datastore, or Cloud SQL.
d. You MUST show your group number (e.g., <title>Group 2: User</title>) in the title of every HTML you return in (1)-(4).
e. Please add proper JavaDoc comments.
f. 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 in 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 on your own group.
- You must include a file named readme.pdf, which contains:
    - Group member info
    - Cloud Service Choice: Amazon or AppEngine
    - Hostname (or IP address)
    - Six screenshots, one for each HTML returned by the five requests, and one for the 404 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 8, with 7 points for correctness (including using MVC, persistence, transaction, etc), and 1 point for documentation, code structure, and unit testing.

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. Google AppEngine credit to be updated.