CMPE 275 Section 2
Fall 2015
Lab 2 - MVC, REST, ORM, and Transactions

Last updated: 10/26/2015

In this lab, you build a set of REST APIs to manage entities and relationships in a social app. You also get to exercise MVC, ORM, and transactions. There are two primary types of entities you are concerned with: persons and organizations. They have the following relationships and constraints:

- Persons can be friends with each other. Friendship is bidirectional. A person can have zero or more friends.
- A person belongs to up to one organization. Different persons can belong to the same organization.
- The **firstname**, **lastname**, and **email** fields are required for any person. Emails have to be **unique** across persons.
- The **name** field is required for any organization, and does not need to be unique.

Partial definition of the related classes are provided below. While the Address class is defined for convenience and clarity, you must embed addresses and not to store them as separate entities.

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

public class Person {
    private long id;
    private String firstname;
    private String lastname;
    private String email;
    private String description;
    private Address address;
    private Organization org;
    private List<Person> friends;
    // constructors, setters, getters, etc.
    ...
}

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

public class Organization {
    private long id;
    private String name;
    private String description;
    private Address address;
    ...
}
```

You **must** persist these entities in a relational database, preferably MySQL. You need to create at least three tables, PERSON, ORGANIZATION, and FRIENDSHIP.

To manage these entities and their relationships, you are asked to provide the following REST APIs. The paths below are relative to the base URL of your app.

Person APIs

(1) Create a person

Path: person?firstname=XX&lastname=YY&email=ZZ&description=UU&street=VV$...
Method: POST

This API creates a person object.

- For simplicity, all the person fields (firstname, lastname, email, street, city, organization, etc), except ID and friends, are passed in as query parameters. Only the firstname, lastname, and email are required. Anything else is optional.
- Friends is not allowed to be passed in as a parameter.
- The organization parameter, if present, must be the ID of an existing organization.
- The request returns the newly created person object in JSON in its HTTP payload, including all attributes. (Please note this *differs* from generally recommended practice of only returning the ID.)
- If the request is invalid, e.g., missing required parameters, the HTTP status code should be 400; otherwise 200.

(2) Get a person

Path:person/{id}?format={json | xml | html}
Method: GET

This returns a full person object with the given ID in the given format in its HTTP payload.

- All existing fields, including the optional organization and list of friends should be returned.
    - The payload should contain the full organization object, if present.

    - The list of friends can be either (a) list of person IDs, or (b) list of "*shallow*" person objects that do not have their friends list populated. If you take option (b), you want to use techniques like lazy loading to avoid serializing the whole social network starting from the requested person in the returned payload.
- If the person of the given user ID does not exist, the HTTP return code should be 404; otherwise, 200.
- The format parameter is optional, and the value is case insensitive. If missing, JSON is assumed.

(3) Update a person

Path: person/{id}?firstname=XX&lastname=YY&email=ZZ&description=UU&street=VV$...
Method: POST

This API updates a person object.

- For simplicity, all the person fields (firstname, lastname, email, street, city, organization, etc), except friends, should be passed in as query parameters. Required fields like email must be present. The object constructed from the parameters will completely replace the existing object in the server, except that it does not change the person's list of friends.
- Similar to the get method, the request returns the updated person object, including all attributes (first name, last name, email, **friends**, organization, etc), in JSON. If the person ID does not exist, 404 should be returned. If required parameters are missing, return 400 instead. Otherwise, return 200.

(4) Delete a person

URL: http://person/{id}
Method: DELETE

This deletes the person object with the given ID.

- If the person with the given ID does not exist, return 404.
- Otherwise, delete the person and remove any reference of this person from your persistence of friendship relations, and return HTTP status code 200 and the deleted person in JSON.

Organization APIs

(5) Create an organization

Path: org?name=XX&description=YY&street=ZZ&...
Method: POST

This API creates an organization object.

- For simplicity, all the fields (name, description, street, city, etc), except ID, are passed in as query parameters. Only name is required.

- The request returns the newly created organization object in JSON in its HTTP payload, including all attributes. (Please note this *differs* from generally recommended practice of only returning the ID.)
- If the request is invalid, e.g., missing required parameters, the HTTP status code should be 400; otherwise 200.

(6) Get a organization

Path:org/{id}?format={json | xml | html}
Method: GET

This returns a full organization object with the given ID in the given format.

- All existing fields, including the optional organization and list of friends should be returned.
- If the organization of the given user ID does not exist, the HTTP return code should be 404; otherwise, 200.
- The format parameter is optional, and the value is case insensitive. If missing, JSON is assumed.

(7) Update an organization

Path: org/{id}?name=XX&description=YY&street=ZZ&...
Method: POST

This API updates an organization object.

- For simplicity, all the fields (name, description, street, city, etc), except ID, are passed in as query parameters. Only name is required.
- Similar to the get method, the request returns the updated organization object, including all attributes in JSON. If the organization ID does not exist, 404 should be returned. If required parameters are missing, return 400 instead. Otherwise, return 200.

(8) Delete an organization

URL: http://org/{id}
Method: DELETE

This method deletes the organization object with the given ID.

- If there is still any person belonging to this organization, return 400.
- If the organization with the given ID does not exist, return 404.
- Return HTTP code 200 and the deleted object in JSON if the object is deleted;

Friendship APIs

(9) Add a friend

Path:friends/{id1}/{id2}
Method: PUT

This makes the two persons with the given IDs friends with each other.

- If either person does not exist, return 404.
- If the two persons are already friends, do nothing, just return 200. Otherwise,
- Record this friendship relation. If all is successful, return HTTP code 200 and any informative text message in the HTTP payload.

(10) Remove a friend

Path:friends/{id1}/{id2}
Method: DELETE

This request removes the friendship relation between the two persons.

- If either person does not exist, return 404.
- If the two persons are not friends, return 404. Otherwise,
- Remove this friendship relation. Return HTTP code 200 and a meaningful text message if all is successful.

Additional Requirements/Constraints

- This is a group assignment, with group size ranging from 2-3. If you want to do it alone, please seek permission from the TA first.
- You are recommended to use a local install of MySQL for your persistence, and you must use a relational database.
- You must use the ORM frameworks taught in the classes, and avoid directly using JDBC and writing queries.
- You must use transactions. It is OK to use the default isolation level.
- You must use Spring's MVC to implement the different returning formats of the two GET methods for person and organization.
- Please add proper Java documents.

Submission

- Please submit through Canvas. Do not include jars or compiled class files.
- Please include at least fourteen screenshots, one for each payload of the format, JSON/XML/HTML, returned by the two GETs, and one for each of the remaining eight requests.

Grading:

This lab has a total point of 10, with 8 points for correctness and 2 points code structure clarity and Java documentation.