Assignment 5: Facegraph - An enriched social network

In this assignment, you will implement a web application implementing a social network associated with data from a knowledge base. Your system will manage the information of a social network, including users and friendships between users. The hobbies of a user will link to the knowledge graph. The application will be implemented using Servlets and JSP (Java 1.8), to be run on Tomcat (8). For storing the social network and the knowledge base, you should use Neo4j. The application should be designed to be viewed by Firefox.

Assignment is 52% Takef.

Submission date: 3/4/2016 until 8:00 am.

TA in charge: Yoav, 236369cs@gmail.com.

This assignment is to be submitted in pairs. Students who wish to submit alone should contact the TA in charge.

In this assignment, you should present your application to the course teachers. You and your partner will need to register for the presentation (presentation times will be published later), and both of you should attend the presentation. Each student in a pair is expected to be familiar with the entire application, including the details of implementation, and to be able to answer questions about the application. Moreover, you will be asked about the previous assignments and their related course material. The final grade depends not only on the quality of the application, but also on the quality of the presentation (the correctness of the explanations) and the answers to our questions during the presentation. Hence, partners may receive different grades.
You should also submit a video clip of your application, a document with an answer to a few questions, and the code of the application, as explained below. The application should be deployed on a virtual machine and should be available for the course staff for examination.

Introduction

Knowledge Bases like Wikidata store complex information about various topics and facts. Such systems provide methods for both computers and humans to analyze the data, and to make use of it in applications. Besides facts, the data also presents relations between items, hence allowing for graph traversal.

Social networks, like Facebook, can make use of knowledge bases, by enriching their network representing relationships between users, with real life facts. One such enrichment is improving link prediction by using the knowledge base [1]. For example, consider two users that are not friends, but both live in the same city, working in the same place, and are interest in the same hobbies. Using the knowledge base, one can detect these common interests and recommend them on each other. This can be done, for example, by searching users, that are similar to a user (e.g., in the place they live in, work in or in their hobbies), but do not have are not friends with this user.

Social networks enable users to post messages that are spread to their friends. However, this feature might turn into a hazard, since users might be flooded with irrelevant information, and abandon the network. One of the solutions social networks adopt to deal with this problem is to block posts from a specific user.

In this assignment you will implement a social network, connecting between users, that is enriched by a Wikidata knowledge base. Each user in the network will be represented by several properties, that will be connected to entities in the knowledge base. Using these links, you will support advanced search options over the graph. Each user in your system will be able to post messages, that will be sent to all of her friends, besides the ones that blocked messages from the user.

Requirements

Your application should provide the utilities described below for managing the information in the system. You should choose appropriate types and sizes for each field you define.

Users

Users are logged-in clients, that may perform various actions in the system. A user should have the following properties:

1. Username – unique, Identifies the user, in the system.
2. **Password**
3. **Name** – the name of the user.
4. **Hobbies**\(^1\) - a set of Wikidata entities describing the hobbies of the user.
5. **Relationship status** - the user should choose this value out of a limited list. You may use same list as Facebook.

Users may have additional properties (optional). Below are a few suggestions for optional properties:

1. **Photo** - A photo of the user. You may store the photo as a file, and in the database store the file-system path to the photo’s file.
2. **Age**
3. **Gender**
4. **Address** - the address of the user, you may use Wikidata entities for the country and the city.

**Messages**

Each user may post messages. A *message* should have the following properties:

1. **Text** - you may limit this field to 1024 bytes.
2. **Timestamp**

Message may have additional properties (optional). Below are a few suggestions for optional properties:

1. Photo
2. Link

**User Management and User Actions**

Users should be able to

- **General:**
  - Register to the site.
  - Login/Logout from the site.
  - Delete herself.
  - See her profile.

- **Friends:**
  - See other users in the system.
  - Become a friend with other users. Note that friendship is a symmetric (undirected) relationship.
  - See her friends, and the friends of her friends.

- **Messages:**

\(^1\) the hobbies should be picked out of a given list of occupations, see [Dataset](#) section
Post a new message.
See all her posts.
See the posts of users she is friend with.
Block posts from a specific friend.

Search and Recommendations:
Search for a user by her name.
Search for a users by relationship status.
Advanced search of users - see Advanced Features.

General Actions
Users should login to their account for using the application; this will be done by providing a user name and a password. The users should be able to logout from the system whenever they wish.

You should manage a list of users and allow new users to register to the site. When a user registers to the site, she will be required to enter her details. Each user should have a unique user name.

Users should be able to delete themselves from the system. When a user is deleted, all her messages should be deleted, all the links to her hobbies should be deleted. Also, if the user was friend of other users, her friendships relations should be deleted.

Useful link - user authentication in Tomcat 8.

Users should be able to view their profile.

Friendships Actions
New users start alone, without any friends. They should be able to browse other users in the system and to connect to them. Once a user connects to another user, they become friends (for simplicity we skip the double approval process exists for example in Facebook).

Users should be able to see their friends, and the friends of their friends.

Messages Actions
Users should be able to post messages over the network. For simplicity in this assignment, you are not required to support deletion of messages.

Users should be able to see messages posted by their friends. The messages should be presented in a “feed” manner, showing the latest message first.

A user should be able to view her own messages.
If a user doesn't want to see posts from a specific friend, he should be able to block posts from this user. Once a user is blocked, her posts should not be available to the user who blocked him. For simplicity, you are not required to support unblocking a blocked friend.

**Search Actions**

Users should be able to search for other users in the system. The search functionality should at least support search by a name, or by a relationship status.

**Advanced Features**

Below we present a list of advanced features. You should choose only two out of the following four advanced features and implement only these two items. You should choose one of the options for a *web service feature* and one of the options for a *presentation feature*.

1. **Web-service feature**: use web services to recommend on directly share services/tasks among different applications. Use the web service you implemented in your system.
   a. *Keywords-based-search* web service
   b. *Graph-based-search* web service

2. **Presentation for integration**:
   a. Use RDF and RDFS for representing users, friendships and hobbies. Allow querying this repository using SPARQL.
   b. Produce *GraphML* storage of the users, relationships and hobbies. Transform the large graph to smaller subsets (keeping the GraphML format), using XSLT; and support the presentation of GraphML using a JS library.

Next, we further describe these items.

**Using Web Services for Sharing Resources**

Information sharing is bidirectional: allowing other sites to receive information from your application (i.e., providing information about users, relationships and hobbies from your application to other applications), and receiving information from other applications (i.e., present on your site information from other applications) using a web service of a remote site.

In this exercise you will need to implement a service that supports remote search in your data (only outgoing information). You should support either a *keyword-based search*, or a *graph-based search*. You should implement a web service with the API described below. Your web service should return a *JSON* array that contains JSON objects according to the format detailed below.

Your web service should be implemented using *Restlet*, and not by regular servlets on Tomcat. This is required in order to practice the concept of REST-API taught during the course. The
Your search should support one of the following two options:

- For a **keyword-based-search web service** you should implement a web service with the following API:

  `<machine ip>/search/?q=<keyword_1>+…+<keyword_n>&k=<num results>`

For example, “132.68.40.18/search?q=golf+football” to search users having the hobbies golf or football. The *keywords* part should be a sequence of keywords that describe the title of hobbies users may have, and *k* determines the maximal size of the result. The result should be an ordered list, in the size of *k*, of users according to their relevance to the search (relevance is a vague concept so you can choose how to define it, and you should implement the ranking accordingly).

The response should be in JSON, and conform to the following format:

`[<username_1>,...,<username_k>]`

If there isn’t any user with the given hobbies, you should return an empty list. If a hobby does not exist in your database, just ignore it.

If you choose this option, add an explanation in your documentation about how the search was performed, and about the way you calculated the recommendation.

Useful links – Neo4j full-text search

- For a **graph-based-search** you should implement a web service with a method that has the following signature:

  `<machine ip>/search/u/<username1>/u/<username2>`

For example, “132.68.40.18/search/u/user1/u/user2” to search the shortest path between two users in the graph. That search should return a **shortest path** between the two given users, measured over friendship relations. The response should include the distance, and the users in the path ordered from user1 to user2. The response should be in JSON, and conform to the following format:

`{dist: <distance>, [<username_1>,...,<username_n>]}`

The users in the query should be included in the answer. For example, if the graph contains only three users connected in a line: user1->user2->user3, then, for the query “132.68.40.18/search/u/user1/u/user3” you should
respond with: {dist:2,[user1,user2,user3]}.  
Note that the distance between a user to himself is 0. If there is no path between the users, you should return an empty JSON object. If there are several shortest paths, just return one of them.

If you choose this option, add an explanation in your documentation about how the feature was implemented, including the query you used.

You should use the web service you will implement in your system, as another functionality available to the users. I.e., support search for users by their hobbies or search for the path between two users over the social network.

You can read more about the importance role of network distance and people search in this reference paper.

Use RDF and RDFS for Representing Users, Friendships and Hobbies, and Allowing Querying this Repository using SPARQL

If you choose to implement this advanced feature, you should use RDF and RDFS for representing the information about users, friendship relationships, and hobbies. The information can be produced and stored in the file system or produced on-the-fly as a stream of characters. You should provide a utility that allows querying the data using SPARQL. Users should be able to pose SPARQL queries over the RDF data. The result of the query should be presented to the user. You should illustrate the result of a query that searches for services and users.

If you choose this option, add examples of three different SPARQL queries that can be used in your system.
Useful link – the Jena project:
● http://jena.apache.org/

Presenting Users, Friendships and Hobbies with GraphML, using XSLT

If you choose to implement this advanced feature, you should export your database including users, relationships, and hobbies to a GraphML object. GraphML is an XML standard for presenting graphs. Note that since the graph might be too big (especially the knowledge base part) you may export only the relevant hobbies. Given an initial GraphML, your application should be able to extract from this graph different subgraphs. To that end, you will transform the origin GraphML to a new GraphML using an XSL stylesheet. You should be able to produce the following results:
● All the users
● Friends of the current user
● Friends of the current user, who share at least one hobby with the current user, and the shared hobbies.
For presenting the results, the application should apply XSLT style sheets on the XML that describes the information.

Useful links – XSLT processors and GraphML:
SAXON, XALAN, TrAX: [http://www.ling.helsinki.fi/kit/2004k/ctl257/JavaXSLT/Ch05.html](http://www.ling.helsinki.fi/kit/2004k/ctl257/JavaXSLT/Ch05.html)
GraphML: GraphMLExporter, blueprints
Exporting Neo4j to GraphML: [neo4j tools](http://neo4j.com)

The GraphML result can be presented on the client side by merely presenting the XML or, by using a proper JS library. Here are some useful libraries for presenting graphs in JS. [D3JS](http://d3js.org), [visJS](http://visjs.org), Force Directed Static Graph, Sigma JS, Radial Reingold–Tilford Tree

**General Requirements**

- **Efficiency and Scalability**: Design your application to be as scalable (suitable to serve many concurrent users) and as efficient as possible. Define proper indices, deal with concurrency wisely and try to optimize your application.
- **Database Consistency**: Your database needs to remain consistent throughout the lifetime of your Web application.
- **Concurrent Site Accesses**: You should allow concurrent access of different users to your website. In addition, you should try minimizing the effect of concurrency management on efficiency. So, for example, you should avoid unnecessary synchronization of code segments.
- **Modularity and Code Reuse**: You should design your system to be modular so that it will be easy to replace components such as the database, and to reuse code in different components of the system or in other systems.
- **Valid Formats**: All the pages you use in this exercise should be valid according to their format. In particular, XML documents should have a DTD (and they should conform to their DTD).
- **Error Conditions**: Try to make sure that your site provides meaningful responses to the user even when a problem comes up. Note that a user of your application should not see an error page that tells her that a Java Exception has been thrown (and she should not see the stack trace of the exception).
- **Checking Input Correctness**: Anytime that you accept input from the user, you should make sure that the input is correct (e.g., required values are entered, number fields contain only numbers, etc.). Try to determine the best place for your input validation code: on the client side, on the server side, or both.
- **Choose the Appropriate Technology Per Page**: When designing your Web site, carefully choose which technology to use for each page (e.g., JSP vs. Servlets, HTML vs. XML with XSL, client-side vs. server-side dynamics, etc.).
Separate Technologies: You should stick to the principle of separating technologies. For instance, your XHTML pages should be bare of style (which is determined by external CSS files) and, as much as possible, dynamic code (which can be written in external JavaScript files). Similarly, your JSP pages should look more like XHTML (or XML) files rather than Java files, while Servlets should look more like Java files and not XML ones.

Uniform-Look Site: The different pages should have a uniform design. Try to avoid using "cut-and-paste" in order to achieve this uniformity. Instead, wisely use CSS, XSL, JavaScript and Servlet/JSP inclusions.

Design

Try to design your site such that the following two principles will hold. Your application will be checked using Firefox.

- Scalable Web Page Design: Try designing your web pages to be properly viewed on (reasonably) resized screens and screens with different (yet reasonable) resolutions. It should also be possible for users to change the font size (by configuring the browser) according to their preferences. In particular, when using CSS, try to use measurements that are proportional to the screen size (like percentage) and font size (like em) as opposed to absolute ones.

- Intuitive Site Navigation: Try ensuring that the pages in your site are linked one to another in a manner that enables easy site navigation. In particular, your application should avoid enforcing the user to use browser buttons like "Back" and "Reload".

What You Should Submit

You should submit

1. A details.txt file that contains:
   a. Your IDs (the two IDs of the submitting students), and your emails.
   b. A link to the Virtual Machine on which your application is deployed.
   c. A link to a video clip that presents your system and its features.

2. A short document with answers to the questions described below.

3. A Zipped project of your application, which includes all the sources as a JAR/WAR file. Make sure you mark the "export source files" option if you submit your project as a JAR/WAR file.

   Instead you may supply a link to a BitBucket repository containing the source code of your project with proper access rights to the course team.

All The files should be zipped and uploaded to the GR submission system.

Your application should include an "about us" description, located under the root directory in a file called 'about.html' or 'about.jsp' (e.g., Yael and Yuval will provide /yaelandyuval/about.html). This page should present at least the following details:
- Your names and emails
- The advanced features you have implemented
- Any other special remarks that concern your design, implementation, deployment and the running of your code.

**Deployment on a Virtual Machine**
Each student will receive a virtual machine (VM) on the faculty’s grid. First, you should approve (by sending an email to the course's email) our regulation form that will be uploaded in the next few days to the course website, in the Assignment 5 section. We are monitoring your VM constantly, if we will see any suspicious actions we may shut down your VM without any notifications.

On the VMs, that are running Linux OS, Tomcat 8 and Neo4j were installed as services. Your account has the privilege of a super user. You may install every other LEGAL software on the VM, by using `sudo apt-get <desired application>` command. For more information see [this link](http://example.com).

For more instruction on how to connect to the VMs we will provide a cheat sheet for your use.

**Dataset**

**Knowledge Base**
In each VM we supply an existing Neo4j database containing partial dump of Wikidata. Since multiple databases requires installing multiple instances of Neo4j, we recommend you to use this database to store the data on users and relationships. The initial database was created [using this project](http://example.com). A compressed version of the database (in case you need to restart it) can be found [here](http://example.com).

User hobbies should be a Wikidata objects, since there are many objects, and the dataset is partial, you should pick a list of hobbies supported by the partial dump.

A drawing for the database structure for your usage can be found [here](http://example.com).

**Initial dataset**
We will supply an initial dataset, that will be use for our testings of your system, you should supply a link, on your website, for uploading this dataset (and deleting all other users and relationships exists).

When you present your application, the database you use should already include several users, with relations between them. Make sure that the content of the database is diverse and sufficient for demonstrating all the capabilities of your application, according to the requirements.

**Video Clip of Application**
You should create a short (not longer than 10 minutes) screencast video clip that presents your application (see [https://en.wikipedia.org/wiki/Comparison_of_screencasting_software](https://en.wikipedia.org/wiki/Comparison_of_screencasting_software)), upload it to a video hosting Web site (e.g., YouTube, Vimeo, [http://en.wikipedia.org/wiki/List_of_video_hosting_services](http://en.wikipedia.org/wiki/List_of_video_hosting_services)) and provide a link to this video. The
video should include all the features you were asked to implement, and specifically, the user management tasks, posting, searching and matching operations, and the advanced features.

Document

You should submit a document that answers the following questions:
1) Describe the design of your database and explain how you handle concurrency and maintain consistency. Explain how your application deals with many concurrent operations of many different users and why your application deals with concurrency correctly.
2) Explain how you designed your code to provide modularity and to support code reuse.
3) Explain how you are handling Input Correctness issues and what makes the application flexible in supporting different types of clients.
4) Explain how you implemented the search task and what makes it effective.
5) Explain what makes your application scalable.
6) Explain how your application handles errors. The document may be written in Hebrew or in English. At the beginning of the document you should write and sign below the following statement:

   We hereby declare that we did the assignment by ourselves, without the help of other people and without looking at the code or at the answers of other students. Both of us contributed equally to this assignment.

Presentation of the Assignment

You will need to register for a presentation at the registration application. Registration is on the basis of "first come, first served", i.e. the earlier you will register, the more registration options you will have. We will specify how to register in a later stage. Each presentation will last about 40 minutes in which you will use a browser, in order to present your work, explain each of the technologies you used, where and how did you use them, and discuss the implementation and other elements of the course. You will also be asked to explain how your system complies with the assignment requirements. We will also have Eclipse in order to look at the application code.

Attendance of both students from each pair is required. Student card / ID is required. Also note that each student will be asked different questions, so it is possible that students of the same group will receive different grades. Students are asked not to discuss with other people other than the course teachers the questions they were asked during the presentation.

Additional notes:
- The place of the presentations will be announced later on.
- Make sure you show up to your scheduled presentation, AS POSTED at the presentations page. If any problem pops up – please notice us as soon as you can.
- Points will be reduced for not attending the scheduled presentation without a notice.
- Although presentations are planned for 40 minutes, it is usually takes a bit longer. Plan your availability accordingly.
- Check & double check your deployment before your submission and presentation, so you won’t have last minute deployment issues.

**Good luck!**

References: