National and Kapodistrian University of Athens

Department of Informatics and Telecommunications

WEB DEVELOPMENT TECHNOLOGIES

PROFESSIONAL NETWORKING APPLICATION

September 2021

Team No.25

Chalkias Spyridon - 1115 2018 00209

Laspias Christos - 1115 2018 00094

● BACKEND ● FRONTEND ● DBMS ● FILTERING

Table of Contents

0. Abstract

0.1 Main goal

The purpose of this project is multidimensional. First of all, *TeamUp* is designed to provide all the necessary services of a modern professional networking application. The main features which were taken into account during the design are:

- Minimalism.
- Ease of use.
- Simple and clean design.
- Security.

TeamUp smoothly combines the functionalities of a **social networking application** and a **job recruiting site**, so that everyone can:

- Publish posts (including photos, videos and sound clips!) and react to them.
- Chat with teammates.
- Make new connections.
- Seek and apply for a job position.
- Maintain a CV-like profile and have the chance to hide sections of it without having to display every information publicly.
- Have **100% control** over their personal data with the option to fully delete their account, if requested.

0.2 What follows

The documentation is separated into **5 main sections**:

1. Usage.
2. Backend.
3. Frontend.
4. Database Management.
5. Filtering algorithms applied.

Each of the **Backend** and **Frontend** sections:

- Starts with the presentation of the **framework** used to develop the respective end.
- Continues by analysing the **security features** added in order to shield the app.
- Ends with a detailed description of the **components** and their functional contribution to the app, along with some **business logic**.

Database Management section, refers to the object-relational database system used to store application's data and to the automated script that loads sample information during app's startup.

The **last section** offers a thorough description of the logic between **job** and **post filtering**. The aforementioned filtering is being implemented with the use of Matrix Factorization, a collaborative filtering algorithm used in Recommendation Systems.

1. Usage

1.1 SSL Certificate

Before launching the application, there may be need to trust the app's *self-signed certificate* "`server.crt`", which is located under:

"`socialnetworkingapp-front/src/ssl/`"

1.2 Compose & Run

In order to deal with environment disparity across different machines and platforms, the project is deployed using *Docker*.
In the root directory of the project run the script (install.sh) with root privileges.The script creates a docker image for the backend and the database and runs them in different containers.If this fails postgresql should be installed on your system and an empty database with name app should be created.Also the application.properties file needs to be changed respectively.For the front-end also run the script inside the socialnetworkingapp-front directory.Again if this fails just run npm start for the front end.Backend runs on https://localhost:8443 and front end runs on https://localhost:4200 while postgresql runs on localhost:5432.

2. Backend

2.1 Framework

The framework used to develop TeamUp's back end is **Spring Boot**. Spring Boot is an open source Java-based framework which provides a **RESTful API** and allows you to create stand-alone, production-grade Spring based Applications.

2.2 Security

In order to ensure that TeamUp is secure , spring security was used in the back-end. After a successfull login , every user gets a json web token and can use the web application. A non-registered/non-logged-in user can only either register or login .In the backend every http request should have a token attached to its headers in order to be authenticated.The validity of the token is checked with the help of sping security and jwt library .Some requests require admin privileges to be performed.This check is also performed with the help of json web tokens.The application runs over https with the help of self-signed certificate.The backend redirects all http traffic to https . Finally in order to authorize requests from front-end and get access to the api we implemented a cors filter allowing all traffic from front-end.

2.3 Model

2.3.1 Entities

The entities that make up the application are the following:

- Account: Every registered user has an account that stores their basic information.
- Bio: Abbreviation for *Biography.* An account can either have a bio associated with it, or not. If a user has a bio, he/she can edit or delete it at any time. On the other hand, if a user does not have a bio, he/she can create a new one.
- Like: A user can add a like to a post. If already liked, he/she can remove the like at any time.
- Comment: A user can add one or more comments to a post. Also, the user has the freedom to edit or delete his/her comment.
- Connection Request: When a user *A* visits the profile of another user *B* that is not present in his network of connected users, *A* can send a connection request to *B,* in order to form a connection. User *B* can either accept or reject incoming connection requests.
- Education: Entity that holds all basic information of education, such as School or University that someone studied, GPA, starting and ending date (if the user has already graduated) etc.
- Experience: Entity that holds all basic information of working experience, such as the Company's name that someone worked in, workplace, employment type, starting and ending date (if exists) etc.
- Job: Represents an article about a job in which users can apply.

- Job Application: Via this entity, a user can apply to a job and if necessary, cancel an already existing job application.
- Job View: When users click to a job article, their views are being collected in order for the recommendation system to work properly and tailor the job articles to each user's taste. The job creator's views are not being counted in the system.
- Message: A user can chat with his/her available connections at any time.
- Post: A user can publish a post that may contain text, image, video or even a sound clip. Also, a user can edit a post's caption, if needed.
- Post View: When users click to a post, their views are being collected in order for the recommendation system to work properly and tailor the posts to each user's taste. The post creator's views are not being counted in the system.
- Tags: Tags (or Interests) are a few keywords that represent different object fields and are associated with accounts and jobs. For example, a user that may be interested in Machine Learning and Software Engineering, will pick the tags MACHINE LEARNING and SOFTWARE ENGINEERING to be appended to his/her profile. Similarly, if a job article is e.g hardware-oriented, the job's creator will append the proper tags to the article. Tags are of major importance when it comes to recommending job articles to users (See section 4.).

2.3.2 Mappers

There's a significant difference between the internal entities of the application and the external objects that are being published back to the client. There are loads of fields that are completely unnecessary for the client to see, for both data integrity and security reasons. That's why mappers have been implemented in this application. For example, if a class *A* has 10 fields but there's only need for the 3 of them to be published back to the client, a mapper converts class *A* to another defined class *B*, which contains only the essential info. Mappers can be found under the `“/mapper”` folder in backend's source files.

3. Frontend

3.1 Web framework

The framework used to develop TeamUp's front end is **Angular**. Angular is a TypeScript-based free and open-source web application framework and one of the most popular platforms for building mobile and desktop web applications.

3.2 Security

Security in the front-end was achieved with the help of angular's Authentication Guard and angular's Local Storage. Every end-point of our application requires a user to be logged in.After every successfull login , local storage saves the token from back-end and user's unique email address.That way we can identify the currently logged-in users by looking up in the local storage.Front end also runs over https with the help of our self-signed certificate.Last but not least we implemented a token interceptor that attaches the token from local storage in every request.That way every request that has a token attached to it's headers can be authenticated from the backend.Also the interceptor attaches another header in order to make the backend capable of indicating front-end.

3.3 Model

3.3.1 Usage and functionality of pages

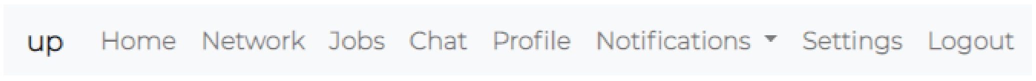
A brief description of the frontend's functionality is as follows:

- Landing page: Welcome page, where a guest can either sign in (if he/she already has an account) or register a new account.
- Home page: Page where a user can create a new post, or browse other user's posts and react to them. The user can see his/her full name and profile picture on the left part of the page and also visit his/her profile or his/her network with a single click.
- Profile: Overview of a user's information summed up in a page. The user can modify his/her profile information by adding, editing and deleting them, or by changing their visibility.
- Settings: Section where a user can change his personal information, including his/her email and password. Also, in this section the user has the freedom to completely delete his/her account.
- Network: Page where all user's contacts are being displayed in the form of animated cards. Each card shows the picture, the full name, the email and the current working position of the respective contact (if exists). By clicking to any of the contact cards, the user is being redirected to the corresponding contact's profile. The search bar on the top right, is implemented to support live search, in order for the user to seek registered users that are not present to his/her network.

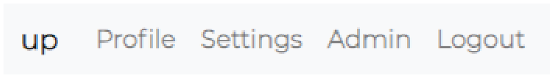
- Jobs: Section dedicated to job browsing. Every user can unfold a job article, read its information and apply to the indicated job position. Job creators can also edit and delete their job articles and furthermore see a list of the viewers and the applicants for each one of their articles.
- Notifications: Dropdown consisting of three (3) pop-ups. A logged user can review his/her received connection requests, the likes and the comments on his/her posts.
- Chat: Place where a user can chat with anyone from his/her network. The user can either continue a discussion that has already begun, or create a new one by clicking the appropriate button on top left. Chatting has been implemented using *Polling* technique.
- Log out: Sign out button.
- When a user *A* visits another user's *B* account, there may be differences in the page's appearance:
    - If user *A* is connected with *B*, then *A* can directly chat with *B* by clicking the corresponding button, or remove *B* from his/her network.
    - If user *A* is **not** connected with *B,* he/she can send a connection request to user *B*. Then, *A* can either wait for *B* to accept the request, or cancel it. When user *B* logs into the application, he/she can accept the connection request by checking his/her notifications, or by visiting *A*'s profile, where an indicative button appears.
- **Admin** has access to admin's page, where he/she can:
    - View a list of all the registered users in the application.
    - Visit any user's profile, by clicking on his/her full name.
    - View a brief overview of any user by clicking the button with the question mark on the right side.
    - Delete any user by clicking the button with the X mark on the right side.
    - Search for users by their full name.
    - Reload the user list.
    - Add a new account.
    - Export in XML and JSON format user data by selecting one or more of them.

3.3.2 Navigation bar

Navigation bar changes appearance depending on whether a user is logged into the app. The bar also changes during the admin's session. When a guest lands in the welcome page, the navigation bar is empty, because the user is not logged in yet. When a user is authenticated, the navigation bar looks like this:



On the other hand, when admin is authenticated, the bar changes to:



because admin’s account exists mainly for management purposes, such as monitoring the registered accounts and extracting their data.

4. DBMS

The database management system used to store TeamUp's data is **PostgreSQL**. Postgres is a free and open-source relational database management system (RDBMS) emphasizing extensibility and SQL compliance.

4.1 File System

In TeamUp, users can upload images, videos and sound clips. Therefore, in order for them to be properly stored in the database, a service has been implemented under **`"/filesystem"`** directory, in project's source files. Whenever a file is successfully uploaded in app's database, it can be retrieved by visiting the link:

```
"{backend server's address:port number}/api/files/{file_id}"
```

e.g: `"https://localhost:8443/api/files/e52fb3dd-19d0-4e7c-84bd-0f756a962bfb"`
(if the backend server is running on localhost, in port 8443)

4.2 Sample data

In order for TeamUp to provide sample data right out of the box, there is a file located in `"/src/main/resources"` called `"import.sql"`. The aforementioned file inserts accounts, network connections, biographies, posts, likes, comments, education, experience events, jobs, views and tags into the app's database.

Notes:

1. All of the sample users have the same password "`12345678`".
2. **Admin** initially has the following credentials:
   a. **email:** `admin@admin.com`
   b. **password:** `adminadmin`

Admin's credentials can then be changed by the admin, by visiting "Settings" page.

5. Matrix Factorization Collaborative Filtering

Both job and post filterings have been implemented with the use of Matrix Factorization technique, in order to tailor both job articles and posts to each individual user's preferences. Matrix-factorization based approaches prove to be highly accurate and scalable in addressing collaborative filtering problems.

M.F implementation in Java is present under the "`/util`" folder, in the project's source files. The recommendation algorithm for jobs can be found in:
"`src/main/java/com/example/socialnetworkingapp/model/job/JobService.java`"
in "`getJobs()`" function, while the recommendation algorithm for posts can be found in:
"`src/main/java/com/example/socialnetworkingapp/model/post/PostService.java`"
in "`findAllPosts()`" function.

The business logic behind each filtering is fully developed below.

5.1 Job filtering

1. Collect all jobs and every user's contact.
2. If the current authenticated user has seen every available job article, proceed to *Tag Filtering*. For every Job, see how many tags the job and the user have **in common** and for every matching tag, add one (1) view.
3. If current authenticated user has not seen all the available jobs:
    a. Run Matrix Factorization. The filter will take advantage of the jobs' views, in order to construct the vectors that represent users in the jobs space.
    b. Proceed to *Tag Filtering*. For every Job, see how many tags the job and the user have **in common** and for every matching tag, add +10% of the previous value of views.
4. Finally, sort by the largest value of views and return the resulting jobs.

5.2 Post filtering

1. Gather all posts **(P)**, that:
    a. A user has made.
    b. A user's friend has made.
    c. A user's friend has liked.
2. Sort **(P)** by creation date (latest first!).
3. If **(P)** have no likes and no comments:
    a. If **(P)** have no views or the current user has seen all of **(P)**:
        i. Sort and return a list **(L)** based on the degree of relationship of the current user with the publishers of the posts **(P)**. Initially, show the posts that connected users have posted (network first), then show the posts that connected users have liked (non-network posts) and finally present the posts that the current authenticated user has published.
    b. If **(P)**'s views are non-empty:

            i. Run Matrix Factorization to **(P)**. The filter will take advantage of the posts' views, in order to construct the vectors that represent users in the posts space.

4. If **(P)** have some likes or/and comments:
    a. If current user has not seen all of **(P)**:
        i. Run Matrix Factorization to **(P)** the same way as above.
    b. Parse the current user's vector of views in posts and:
        i. For each liked post add 100% of the already existing views in that post.
        ii. For each comment in a post add 50% of the already existing views in that post.
5. Sort by most viewed posts.
6. Sort and return the resulting list the same way as (3.a.i).

6. Summary

6.1 Organisation and elaboration

At the beginning of our work, we made an attempt to diagrammatically represent all the requirements of the model, in order for us to become more familiar with the project's structure and overall functionality. This helped us avoid misunderstandings and also prevented us from rewriting a large amount of code. Then, we started developing the back end of the application, by creating the corresponding classes for each necessary entity. Later on, we created a large amount of the controllers, services and repositories for all the entities, and then we moved on to the front end. The front end was developed one request at a time, because we had to design the overall page, generate the corresponding components and services in Angular and then link all of them with Spring Boot. Continuing, we started heavily testing and debugging the application by adding more data (sample data) and by executing edge cases. Finally, we cleaned up the code from unused code blocks and functions and we added security and SSL encryption.

6.2 Sources

https://stackoverflow.com/
https://www.baeldung.com/
https://www.gitmemory.com/
https://www.bezkoder.com/
https://www.bootdey.com/
https://getbootstrap.com/
https://amigoscode.com/
https://newbedev.com/
https://codecraft.tv/
https://roytuts.com/

END OF DOCUMENTATION