



Professional Networking Application (LinkedIn type)

Contents

INTRODUCTION:.....	2
Goal:	3
Installation Instructions:	3
1. Front-end:	3
2. Back-end:.....	3
3. Database:	4
Design Decisions:.....	4
Contents:	4
Chapter 1: Creation and management of professional profiles by users	5
Profile Creation:	5
Backend:	6
Frontend:.....	6
Return to Home Page:	6
User Login:	6
Backend:	7
Frontend:.....	7
Chapter 2: Connecting with other professionals and managing the contact network	7
Connection:	8
Backend:	8
Frontend:.....	8
Contact Network Management and Profile View:	8
Backend:	8
Frontend:.....	9
Chapter 3: Posting Articles and Commenting on Other Users' Posts	9
Backend:	9
Frontend:.....	10

Post Example:	11
Chapter 4: Job Ads Management and Display	11
Job Ad Creation:	11
Job Ads Viewing:	12
Job Ads Management:	12
View Tracking:	12
Example of Job Ads Display:	12
Chapter 5: Private Conversations Between Users	13
Backend:	13
Frontend:	14
Example of a Private Conversation:	14
Chapter 6: User Data Export Capability for the Administrator	15
Backend:	15
Frontend:	15
Example of Admin Page:	15
Chapter 7: Matrix Factorization	16
Post Recommendation:	16
JobAd Recommendation:	17
Random Dataset Generation:	17
Matrix Representation for the Algorithm:	17
Chapter 8: Notification Display and Management	18
Chapter 9: User Settings	18
Chapter 10: Initial Setup of Roles and Users	19
Chapter 11: Authentication	19
JwtAuthenticationFilter:	19
JwtUtilities:	19
SpringSecurityConfig:	19
Conclusion:	20

INTRODUCTION:

Goal:

This project was implemented as part of the course "Internet Application Technologies." Its goal is the development of a professional networking application called **ConnectX**, which is similar to the LinkedIn platform. It allows users to create and manage their professional profiles, connect with other professionals, and search for or publish job advertisements.

The application offers two main user roles: **Administrator** and **Professional**.

- The **Administrator** role allows for user management and the export of their data in XML and JSON formats.
- The **Professional** role enables the creation and editing of a professional profile, interaction with other users, posting articles and job ads, as well as participating in private discussions.

The application is designed to operate in a **web-based environment**, offering users a user-friendly and secure platform. All communications are performed through encrypted HTTP requests (SSL/TLS), ensuring the security of user data.

Installation Instructions:

1. Front-end:

- Programming Languages: JavaScript and CSS
- Framework: React
- Development Environment: PhpStorm
- To install all the required dependencies, navigate to the project's root directory and run the following command:
`npm install --legacy-peer-deps`
- After the installation is complete, start the development server and launch the front end by running:
`npm start`

2. Back-end:

- Programming Language: Java 17

- Framework: Spring Boot
- Development Environment: IntelliJ IDEA Ultimate
- Build Tool: Maven

3. Database:

- Database System: Oracle
- The Oracle database is hosted within a Docker container, ensuring easy management, development, and maintenance of the database environment.
- To create the Oracle container, execute the following command:

```
docker container create -it --name tedi2024 -p 1521:1521 -e ORACLE_PWD=tedi2024  
container-registry.oracle.com/database/express:latest
```

Design Decisions:

1. Architecture:
 - The application is built based on the Model-View-Controller (MVC) architecture, ensuring a clear separation of concerns and maintainable code structure.
2. Security:
 - Secure communication between client and server is ensured through SSL/TLS encryption, protecting user data during transactions.
 - Database encryption mechanisms are also in place for sensitive information.
3. Backend Models:
 - The backend is structured into two main model types: DAO (Data Access Objects) and DTO (Data Transfer Objects).
 - DAO represents the data as it is stored in the database.
 - DTO represents the data as it is sent to the frontend, providing an additional layer of security and abstraction.

This design approach ensures scalability, security, and efficient data handling across different layers of the application.

Contents:

The development of the application includes the following features:

1. Creation and management of professional profiles by users
2. Connecting with other professionals and managing the contact network
3. Posting articles and commenting on other users' posts
4. Management and display of job advertisements
5. Private conversations between connected users
6. Data export capabilities for administrators
7. Matrix Factorization (recommendation system)
8. Display and management of notifications
9. User settings
10. Initial setup of roles and users
11. Authentication

Chapter 1: Creation and management of professional profiles by users

Profile Creation:

A user can create their own profile by following these steps:

Once on the application's home page, they can click the REGISTER NOW button. Then, a form will appear where they will fill in their details, which include:

- First Name
- Last Name
- Email
- Password

- Confirm Password

Additionally, the user can optionally provide:

- Phone Number
- CV (Curriculum Vitae)
- Profile Image

Backend:

In the backend implementation, there are two models: User and UserDto, both containing all the necessary fields for the user. User data is stored in the UserRepository, while user-related operations are implemented in UserServiceImpl.

Within this service, the register() method is responsible for creating user profiles.

In the UserController, appropriate endpoints are created to handle requests from the frontend. Specifically, there is a register() endpoint (of type POST) that calls the register() method to register new users.

Frontend:

On the frontend side, the profile creation process is managed through the files RegistrationForm.js and RegistrationForm.module.css (which is used for styling the form).

In RegistrationForm.js, when the user clicks the REGISTER button, a request is sent to register the user. If all fields are correctly filled out, the user is saved to the database.

Return to Home Page:

If a user wants to return to the home page, whether they have not completed their registration or after completing it, they can do so by clicking on the **ConnectX** icon.

User Login:

A user can log in to their profile through the **Login** form by entering their **Email** and **Password**.

Backend:

The `authenticate()` method in the `UserController` accepts a `LoginDto` object through a POST request, which contains the user's login credentials.

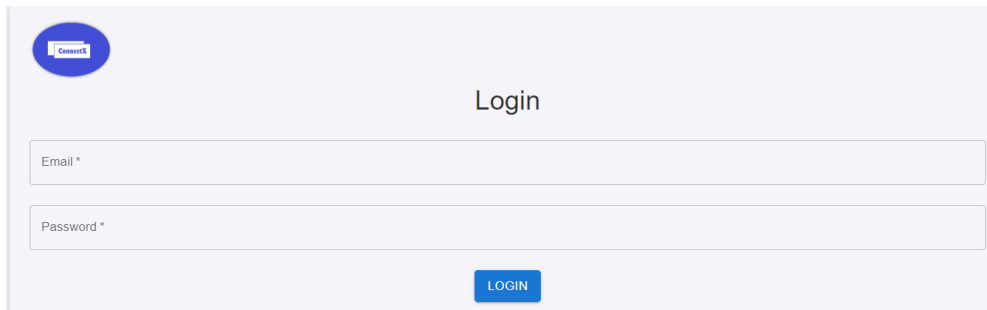
This method calls the corresponding `authenticate()` method of `UserServiceImpl`, which uses the `AuthenticationManager` to validate the user's credentials.

- If the authentication is successful, it stores the authentication in the `SecurityContextHolder`.
- It then searches the database for the user based on their email.
 - If the user is found, it returns a `LoginResponse` object containing the user's relevant information.
 - If the user is not found, it throws a `UsernameNotFoundException`.

Frontend:

In `LoginForm.js`, when the user submits the form, a request is sent to verify their credentials.

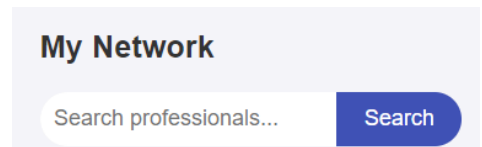
- If the login is successful, the user's information is stored, and they are redirected to the appropriate page based on their role.
- If the login fails, an error message is displayed to the user.

A screenshot of a login form. In the top left corner, there is a blue circular button with a white 'Connect' label. The form itself is centered and has a light gray background. The title 'Login' is centered at the top of the form area. Below the title, there are two input fields: the first is labeled 'Email *' and the second is labeled 'Password *'. At the bottom center of the form, there is a blue rectangular button with the text 'LOGIN' in white capital letters.

Chapter 2: Connecting with other professionals and managing the contact network

Connection:

A user can search for another user based on their name. After finding the desired user, they can send a connection request to add that user to their professional network.



Backend:

The `searchConnections` method handles the GET request for searching users based on a search query (keyword) and calls the corresponding service `searchConnections`.

On the other hand, the `connect` method manages the POST request for connecting with other users. It calls the `connect` service, passing the ID of the user that the current user wants to connect with.

These methods are implemented in the files `NetWorkController` and `NetWorkService`, and the connections are stored within the `User` model of each user.

Frontend:

In `NetworkPage.js`, the user search is performed through an input field, where the user types the desired search term. This term is stored in the state variable called `searchQuery`.

- When the value of `searchQuery` changes, an automatic GET request is sent to the API to search for users.
- The search results are stored in the `searchResults` state and displayed in a list.

Additionally, the user's existing connections are retrieved during the initial page load through a GET request to the API and are displayed in a card layout.

Contact Network Management and Profile View:

A user can manage their contacts by viewing them in a list format and selecting the profile of a specific user.

- From that profile, the user can initiate a private conversation with them.
- Additionally, the user can explore the contact's skills, professional experience, and education, as long as this information is publicly available.

Backend:

In the backend, the `getProfile` method in the `NetWorkController` class handles the GET request to retrieve a user's details based on their `userId`.

- This method calls the `getProfile` method of the `NetworkService`, which returns the requested user's profile, using the connected user's email for authentication.

Frontend:

On the frontend, the **ProfilePage** class uses the **useEffect** hook to execute an **API request** when the page loads.

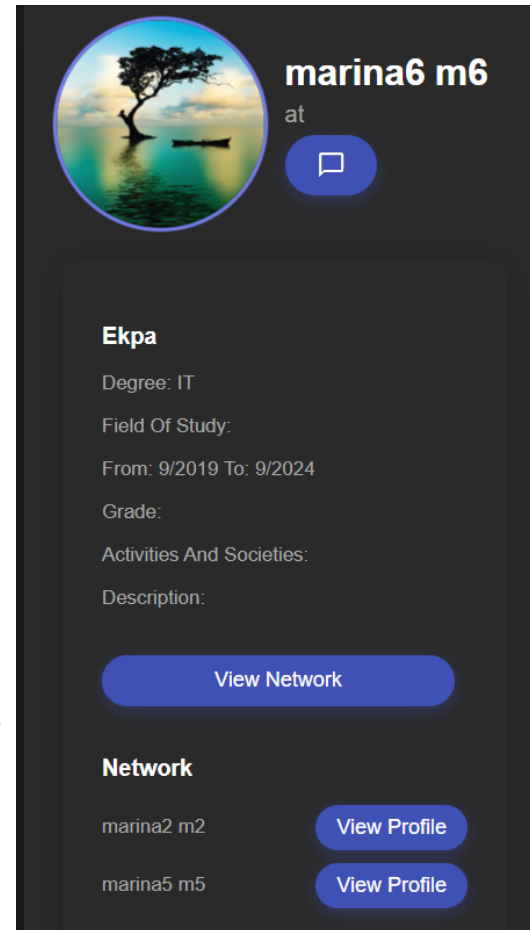
- This request fetches the user's profile from the backend through **Axios**, storing the retrieved data in the application's **state**.

The user profile is then displayed with details, allowing for easy interaction and exploration of professional information.

Chapter 3: Posting Articles and Commenting on Other Users' Posts

On the Posts page, the user has the ability to:

1. **Post Articles:** The user can create a new post with text, images, videos, or voice messages.
2. **Commenting:** The user can add comments to posts made by other users, facilitating interaction and discussion.
3. **Reactions:** Users can express reactions (e.g., Like, Love, or Care) on posts.
4. **Edit/Delete:** The user is also able to edit or delete their own posts if needed.



Backend:

In the backend, all functionalities related to posts, such as creation, editing, deletion, and retrieval, are managed in the PostController file.

- When a user submits a new post, the `createPost()` method handles the request by collecting the necessary data, including:
 - The content of the post
 - Attached files (such as images, videos, or voice messages)
 - The user's information
- This data is organized into a `PostDto` object, which is then forwarded to the `PostService` for processing.

Apart from creating posts, the controller also supports:

- Commenting on posts
- Reactions (e.g., like)

- Displaying posts from the user's network

User authentication is managed through Spring Security, ensuring that only authorized users can access and perform these actions. This is done with the help of the `@AuthenticationPrincipal` annotation, which identifies the user with every request.

Frontend:

Posts are retrieved from the backend using the method `PostService.getPostsByUserAndNetwork()`, which loads the posts of both the user and their network.

Post Creation:

- Users can create new posts by typing text or uploading files through HTML input fields.
- State management is handled with `useState` in React.
- When a post is submitted, `PostService.createPost()` sends the content and attached files to the backend.

Dynamic Updates:

- Posts and reactions are dynamically updated on the page without the need for a full refresh.

Commenting and Reactions:

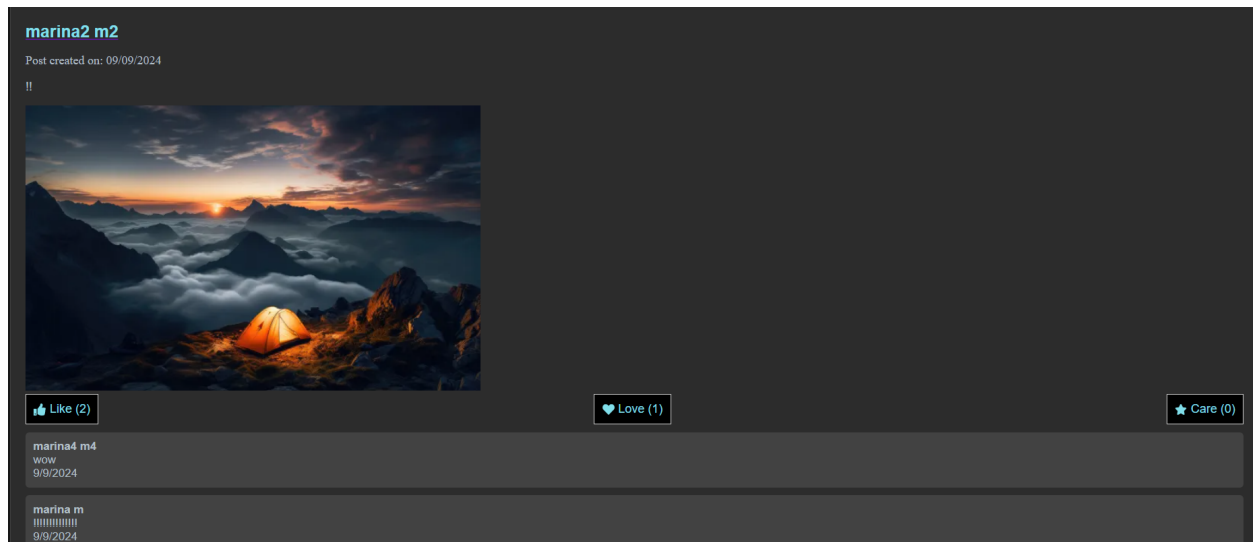
- Users can add comments and reactions to posts, which are sent back to the backend for storage.

Post Display:

- The `Post` component is responsible for displaying each post.
- Reaction buttons are linked to the `handleReaction` function, which:
 - Updates the application's state.
 - Sends the corresponding request to the backend.

Data is fetched from the backend using `Axios` with the appropriate methods from `PostService`.

Post Example:



Chapter 4: Job Ads Management and Display

Job Ad Creation:

The createJobAd method in the JobAdServiceImpl class handles the creation of a new job advertisement.

- The user sends a JobAdvertisementDto object through a POST endpoint in the JobAdController, which is converted into a JobAdvertisement entity.
- The method assigns the current user as the creator of the job ad and saves it to the database via the jobAdRepository.

On the frontend, the user can click the "Create Job" button to open a form.

- In the form, the user fills in details such as:
 - Job title
 - Description
 - Company

- Location
- Type (e.g., full-time)
- Level (e.g., junior, senior)
- Required skills

Once the form is submitted, an axios request is sent to the API to create the job advertisement.

Job Ads Viewing:

Within the `getJobAds` method, the user retrieves the available job advertisements.

- Based on the user's skills, the job ads are filtered to return only those matching the user's qualifications.
- The job ads are converted into `JobAdvertisementDto` objects before being sent back to the user.

When the page initially loads, API calls are made to fetch all available job ads.

- These ads are stored in the state and displayed as cards.
- The user can view detailed information for each job advertisement.

Job Ads Management:

Through the `applyToJobAd` method, users can apply for a job advertisement only once.

- The user is added to the list of candidates for the respective job ad.

For each job ad, the system checks whether the current user is the creator of the ad.

- Creators can view the candidates who have applied to their job advertisements.

Additionally, the `deleteJobAd` method allows the creators of job ads to delete their ads.

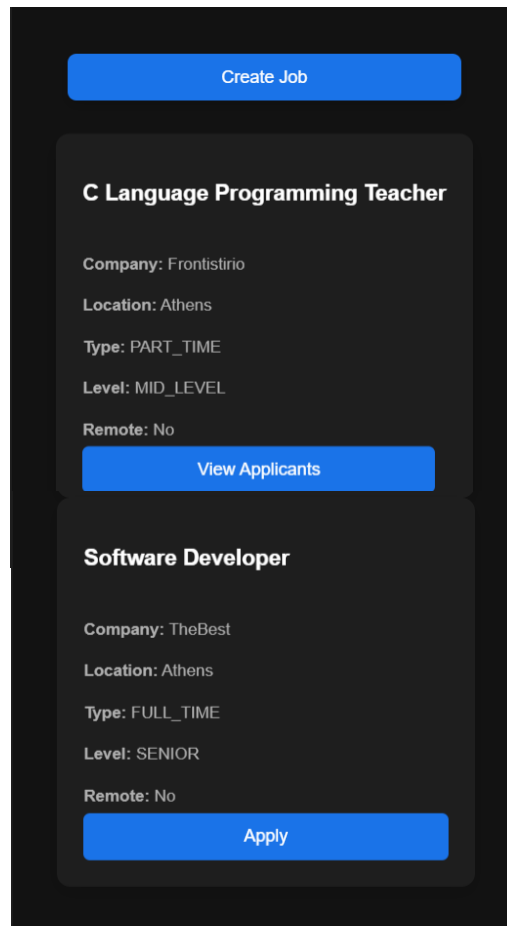
- Before deletion, it verifies that the user attempting the deletion is indeed the creator of the job ad.

View Tracking:

Through the `trackJobAdView` method, the application records how many users have viewed a job advertisement by adding the user to the list of viewers.

- When a user hovers the mouse over a job ad, the view is recorded and tracked.

Example of Job Ads Display:



Chapter 5: Private Conversations Between Users

A user can send private messages to other users, whether they belong to their network or not.

They have the ability to search for any user they wish and start a conversation with them.

All existing conversations are displayed in the left column, while the right side shows the current conversation with the selected user.

Backend:

The MessageServiceImpl manages the logic for sending and retrieving messages.

- When a user sends a message, the system checks if there is an existing chat between the two users; if not, it creates a new one.

- The message is then stored within the chat.
- Users can also create a new conversation with another user via the /createChat endpoint.

The MessageController handles HTTP requests for sending messages, retrieving messages, and creating conversations, using the authenticated user obtained through the SecurityContextHolder.

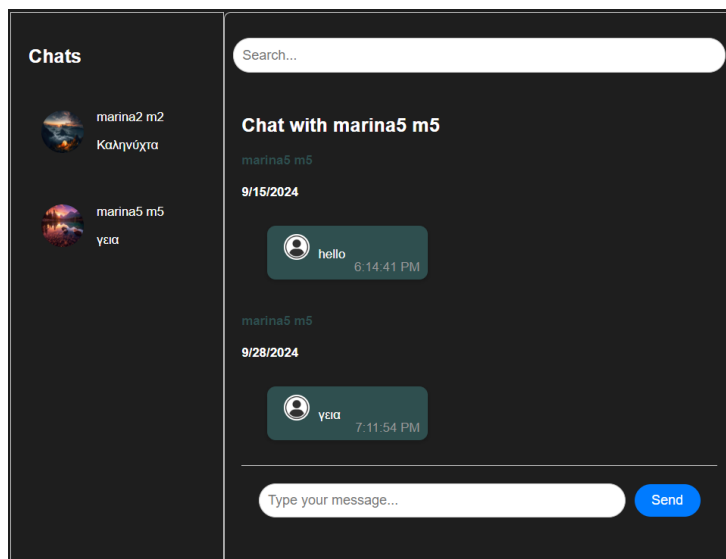
Frontend:

The system loads the user's conversations during the initial rendering of the page.

It includes functionality for automatic message loading when a conversation is clicked, as well as for displaying real-time search results.

Using profile pictures, each conversation provides a more personalized experience, while messages are sorted by date and sender, creating a clean and organized chat interface.

Example of a Private Conversation:



Chapter 6: User Data Export Capability for the Administrator

Backend:

The AdminController provides three main functions for user management:

- getUsers(): Returns the list of all users from the database via the UserService.
- getUserById(): Allows the admin to retrieve details for a specific user.
- exportUsers(): Enables exporting user data in XML or JSON formats, depending on the selected fields.

Frontend:

The AdminPage component offers several user management features:

- It uses the useEffect hook to fetch the list of users from the backend through axios, storing the data in the users state.
- The admin can select users of interest via checkboxes.
- Additionally, the admin can choose specific fields they wish to export.
- By clicking the export buttons, the admin can download the selected user data as a file in the chosen format.

Example of Admin Page:

Admin Email: admin@gmail.com

Admin Password: adminPassword12!

The screenshot displays the 'Admin Page' interface. On the left, under 'Select Users', there is a list of users with checkboxes and 'View Details' buttons. The users listed are: admin@gmail.com, marina mylona - marinamylona123@gmail.com, marina m - marina@marina.com, marina2 m2 - marina2@marina2.com, marina3 m3 - marina3@marina3.com, and marina4 m4 - marina4@marina4.com. On the right, under 'Select Fields', there is a list of fields with checkboxes: FIRSTNAME, LASTNAME, EMAIL, TITLE, BIO, CV, and POSTS. At the bottom, there are three buttons: 'Export to XML', 'Export to JSON', and 'Log out'.

Chapter 7: Matrix Factorization

The recommendation system operates through the following steps:

1. Creation of a two-dimensional ratings matrix.
2. Initialization of the MatrixFactorization class.
3. Model training.
4. Prediction of ratings.
5. Recommendations based on the predictions.

Post Recommendation:

The PostRatingMatrixBuilder class creates a user-post interaction matrix, where the rows correspond to users and the columns correspond to posts.

- Ratings are based on different types of interactions (e.g., LIKE, LOVE, CARE, COMMENT) with specific weights assigned to each type.

Next, using the MatrixFactorization class, the relevant matrices are initialized:

- The user matrix and the post matrix are initialized randomly.

The model is trained using the Stochastic Gradient Descent (SGD) technique, which updates the parameters of the matrices (userMatrix and postMatrix) to minimize the error between the actual ratings and the predicted ratings.

After training, the model uses the predict method to calculate possible ratings for each post that the user has not yet rated.

- This is done by computing the dot product of the user and post factor matrices.

Posts are then sorted based on the predicted ratings in descending order, and the top recommendations are returned to the user.

Additionally, priority is given to posts from the user's network as well as their own posts, which are displayed in chronological order.

- If the number of posts does not reach the predefined limit (numRecommendation = 20), the system fills the remaining slots with top recommendations.

The entire implementation is found in the recommendPost method of the PostService, the RecommendationService, the MatrixFactorization class, and the PostRatingMatrixBuilder class.

JobAd Recommendation:

The process is similar to the Post Recommendation system.

Specifically, a ratings matrix is created from users and job ads, where the ratings are based on user applications and views of the job ads. The Matrix Factorization algorithm remains the same.

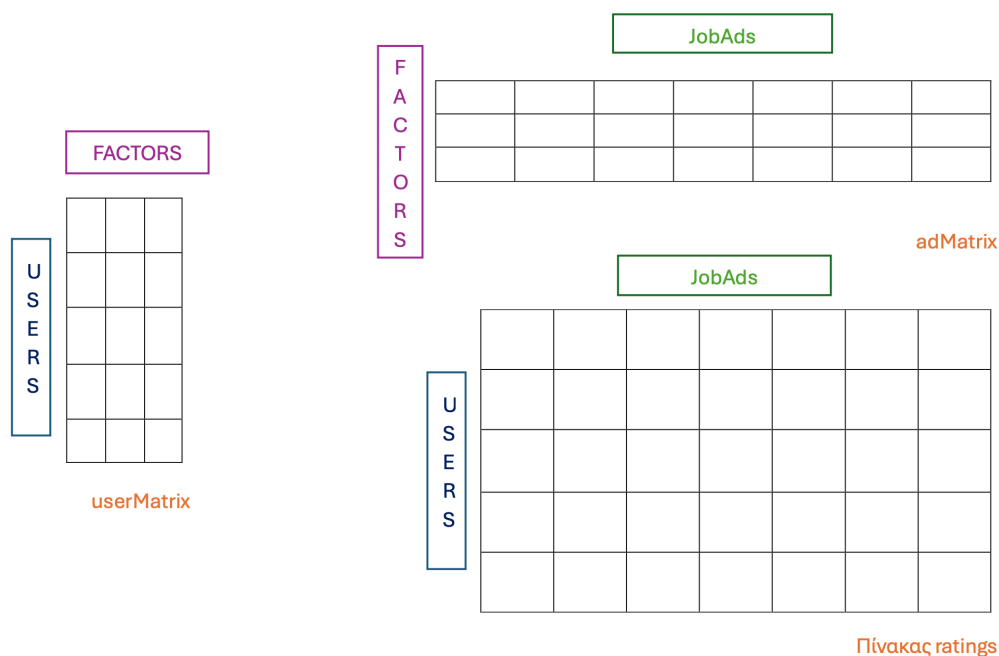
Additionally, in the RecommendationService, there is the method recommendAdForUser, which follows this procedure for recommending job ads:

- First, the list shows the job ads posted by the user themselves.
- Next, it includes job ads that match at least one of the user's skills, starting with those that match the most skills.
- Finally, based on the algorithm's predictions, recommended job ads are displayed without duplicates.
- If the number of recommended ads has not reached the preset limit (numRecommendations = 20), the list is filled with further top suggestions.

Random Dataset Generation:

Using the ArtificialDataGenerator file, users, posts, reactions, comments, and job ads are created. These datasets are used only for the recommendation algorithm.

Matrix Representation for the Algorithm:



Chapter 8: Notification Display and Management

On the Notifications page of the application, users can view notifications relevant to them. The page is divided into two main sections:

1. Top Section:

- Users can see their pending connection requests here.
- The backend manages these requests via the `NetWorkController`, using a GET request to the `/requested` endpoint to retrieve pending requests.
- Acceptance or rejection of requests is handled via a POST request to the `updateConnectionRequest` method.

2. Bottom Section:

- This part displays notifications related to reactions or comments from other users on the user's posts.
- These notifications are managed on the backend via the `NotificationController`, which uses the `getNotifications` method from the `NotificationService`.

On the frontend, both sections are handled in the `NotificationPage.js` file, where data is fetched and rendered based on the server's response.

Chapter 9: User Settings

On the Settings page, users can change their password by filling out a form.

- On the backend, this process is handled by the `updateCredentials` method in the `UserService` class. This method is responsible for validating and updating the user's password.
- The method is called via a POST endpoint defined in the `UserController` class.
- On the frontend, the settings page functionality is implemented in the `SettingsPage.js` file, with styling handled in `SettingsPage.module.css`.

Chapter 10: Initial Setup of Roles and Users

In the config/BootstrapData file, a CommandLineRunner is implemented, which executes code when the application starts to set up initial roles and users.

- It checks if roles already exist in the database; if not, it creates the roles USER and ADMIN via the userService.
- It also creates a user with the email admin@gmail.com and the password adminPassword12!, encrypts the password, and saves the user to the database.
- Then, it retrieves the user and the ADMIN role and assigns this role to the user, saving the updated user information to ensure proper application functionality.

Chapter 11: Authentication

JwtAuthenticationFilter:

- This filter is responsible for authenticating users via JWT tokens.
- It reads the token from the HTTP request header ("Authorization"), verifies its validity, and if valid, extracts user details from the token.
- Upon successful authentication, it creates a UsernamePasswordAuthenticationToken object and stores it in the SecurityContextHolder, which manages the application's security.

JwtUtilities:

- This class handles operations such as creating tokens when a user successfully logs in, extracting information from tokens (e.g., email or claims like user roles), validating tokens (checking expiration and integrity), and authenticating users based on valid tokens.

SpringSecurityConfig:

- This class configures Spring Security in the application.
- It sets up security for application endpoints and provides beans for AuthenticationManager (handling authentication processes) and PasswordEncoder (using BCryptPasswordEncoder for password hashing).

Conclusion:

This project involves the development of a comprehensive networking application that offers many of the classic features found in platforms like LinkedIn.

The application combines a pleasant user experience with security and efficient data management through a modern backend system built with up-to-date tools and technologies.

It fully meets users' needs by providing capabilities such as:

- Posting and interacting with posts
- Managing and viewing contact networks
- Searching and adding job advertisements
- Engaging in private conversations
- Viewing notifications
- Editing profiles
- Changing passwords through user settings