CS/SE/CE 3354 Software Engineering
Final Project Deliverable 1

LinkedOut

Group 3

Benjamin Schuh, Mytri Nair, Manthra Natarajan, Adam Hassan
Mahmoud Elsharydah, Will Arato, Shreyas Ankolekar, Maya Gangadharan

1. Project Implementation

Our project will be focused on a web application implementation deployed via existing cloud services:

- Possible Cloud/Serverless Solutions: Google Cloud, AWS, Azure, Render, Railway, Netlify, Vercel, etc.

Main Goal:

- To give students an outlet to network with others with similar experiences.
- To give students motivation to build and update LinkedIn accounts.
-

Software for Development:

- MERN stack (Mongo, Express, React, Node).
- Python/selenium web scraping AND/OR import LinkedIn profile to generate profiles/matches via MongoDB
- An integrated chatbot (i.e to draft messages to another student based on interests)

Features:

- Profile maker to analyze LinkedIn profiles and get interests/skills
- Most of user profiles will be on Linkedin and users can add additional information if needed
- Import a LinkedIn profile and fetch everything from the profile or can manually add for yourself (you as user) in another section.
- Web Scraping/LinkedIn import for the profile data.
- Ai that connects based on similar linked in profiles
- Users can swipe right if they want to connect with another student
- Users swipe left if they want to move on
- If two users swipe right on each others profile notification will appear and their linked in profile will appear and allow users to connect with each other

Motivation

Students often struggle to find the right connections that align with their skills. Traditional job boards don't have a personalized approach, or a chance for students to connect going through the same struggles. **LinkedOut** simplifies the process by analyzing your LinkedIn profile and matching you with the best-fit students—just like a dating app, but for your career. It would help students connect with like-minded peers for hackathons, projects, and networking.

2. Github Repository Link

a. https://github.com/manthranatarajan/LinkedOut

3. Task Delegation

Benjamin Schuh:	**Functional and nonfunctional Requirements, UseCase diagram, swipe Sequence Diagrams, Architectural Design**
Manthra Natarajan:	**Git, SWmodel justification, part traceability matrix**
Will Arato:	**Non-functional Requirements, Badge Sequence Diagram**
Mahmoud Elsharydah:	**Functional/Non-Functional Requirements, Use-Case Diagram, deleteProfile and viewProfile Sequence Diagrams, Class Diagram**
Maya Gangadharan:	**Edit and Create Sequence Diagrams, Use Case diagram, Functional and NonFunctional Requirements**
Shreyas Ankolekar:	**Functional and NonFunctional Requirements and ViewLeaderBoard Sequence Diagram**
Mytri Nair:	**Creating Sequence Diagrams log in and view Notifications**
Adam Hassan:	**Created Notification Sequence Diagrams, Class Diagram, FR & NFR.**

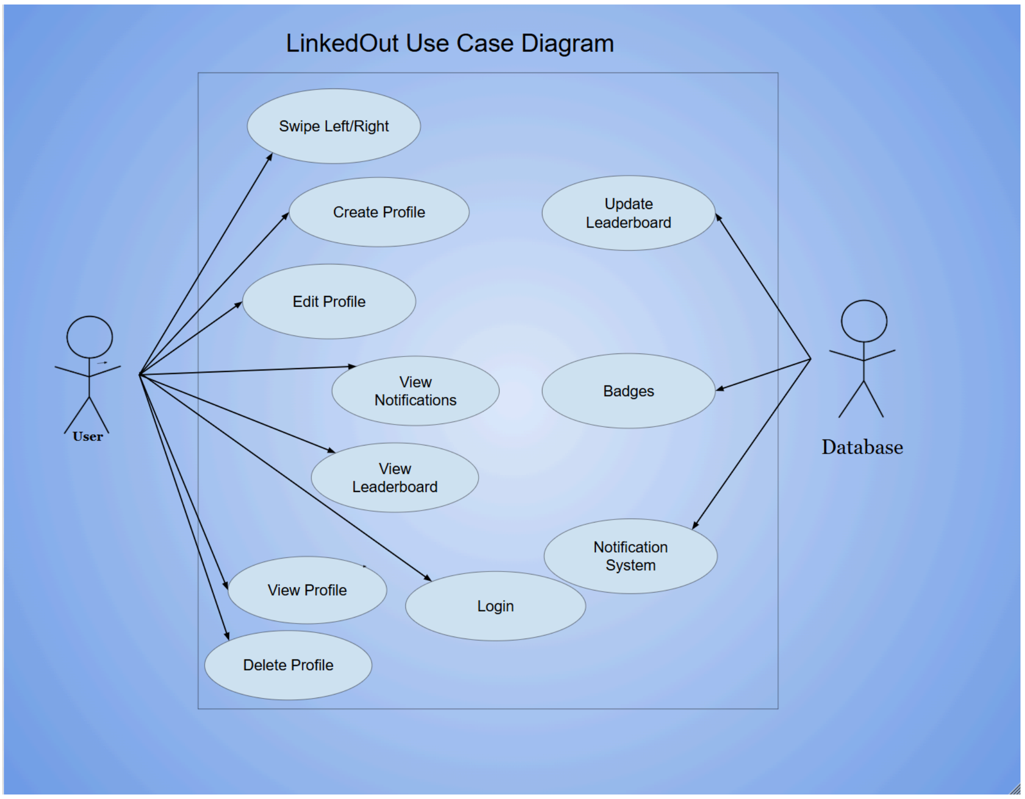
4. Software Process Model

We have chosen an Agile approach for this project, as it allows for an incremental and iterative development process. Given that this project is consumer-focused, it is essential to keep the customer actively involved throughout the planning and testing phases. Agile enables the system to dynamically adapt to user needs, allowing for flexibility and continuous improvement. Additionally, maintaining a simple and intuitive design ensures ease of use and efficient implementation of changes when necessary.

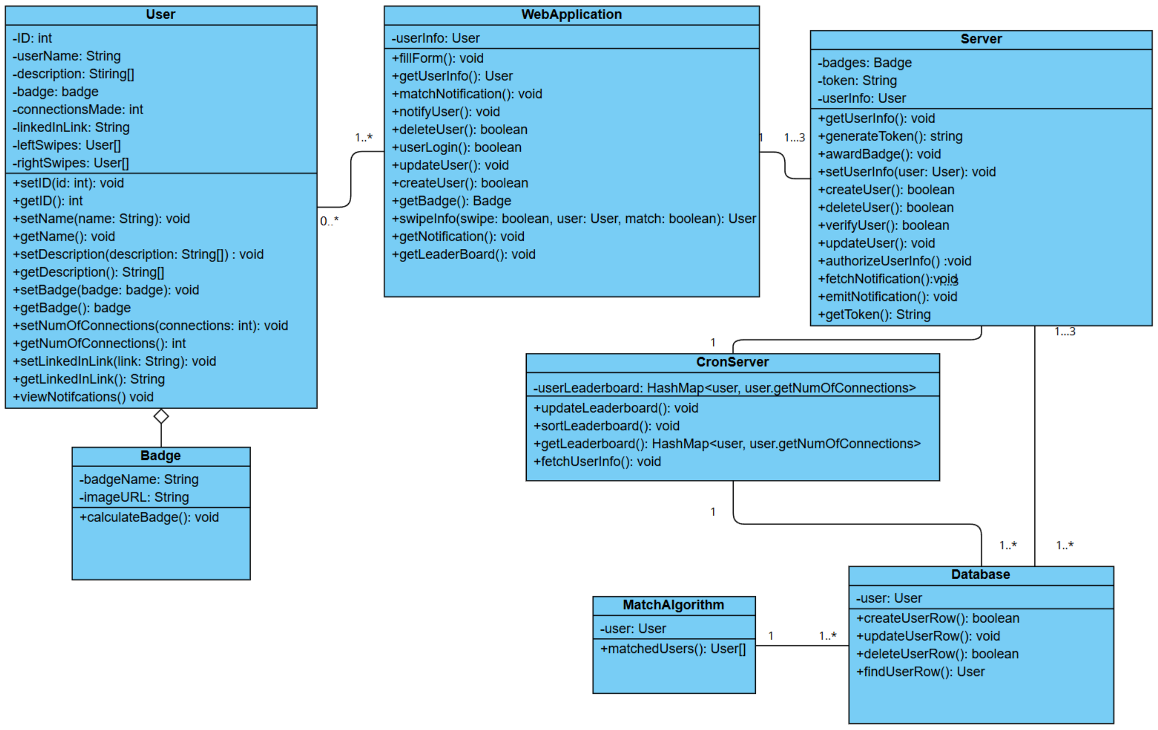
The system will be flexible and will allow changes to be incorporated based on continuous feedback loops from end-users. Using Agile will let us have iterations; each iteration will undergo code reviews, unit testing, and integration testing. This approach would make our project more scalable and user friendly.

5. Diagrams

a. UseCase Diagram

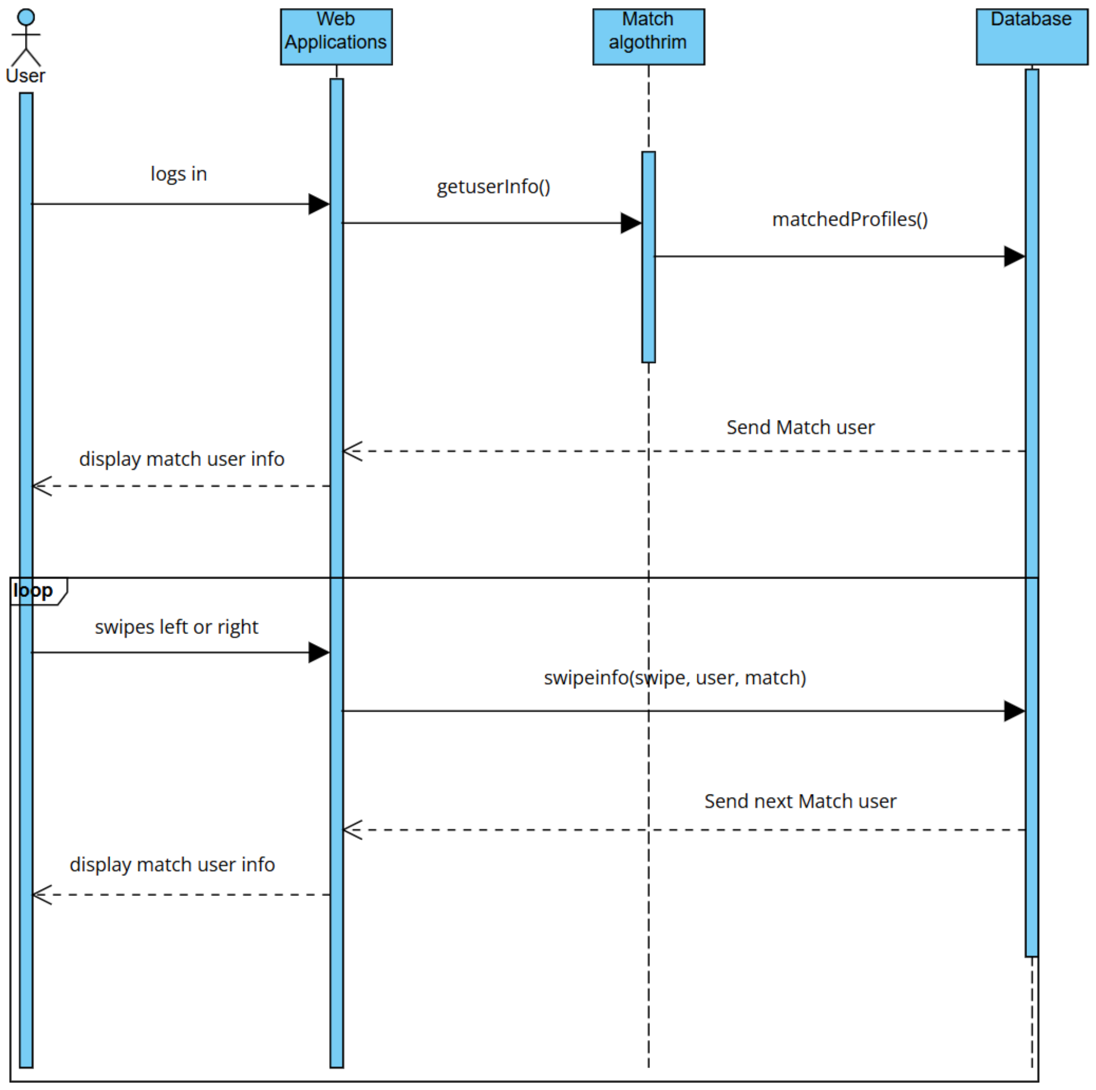


b. Class Diagram

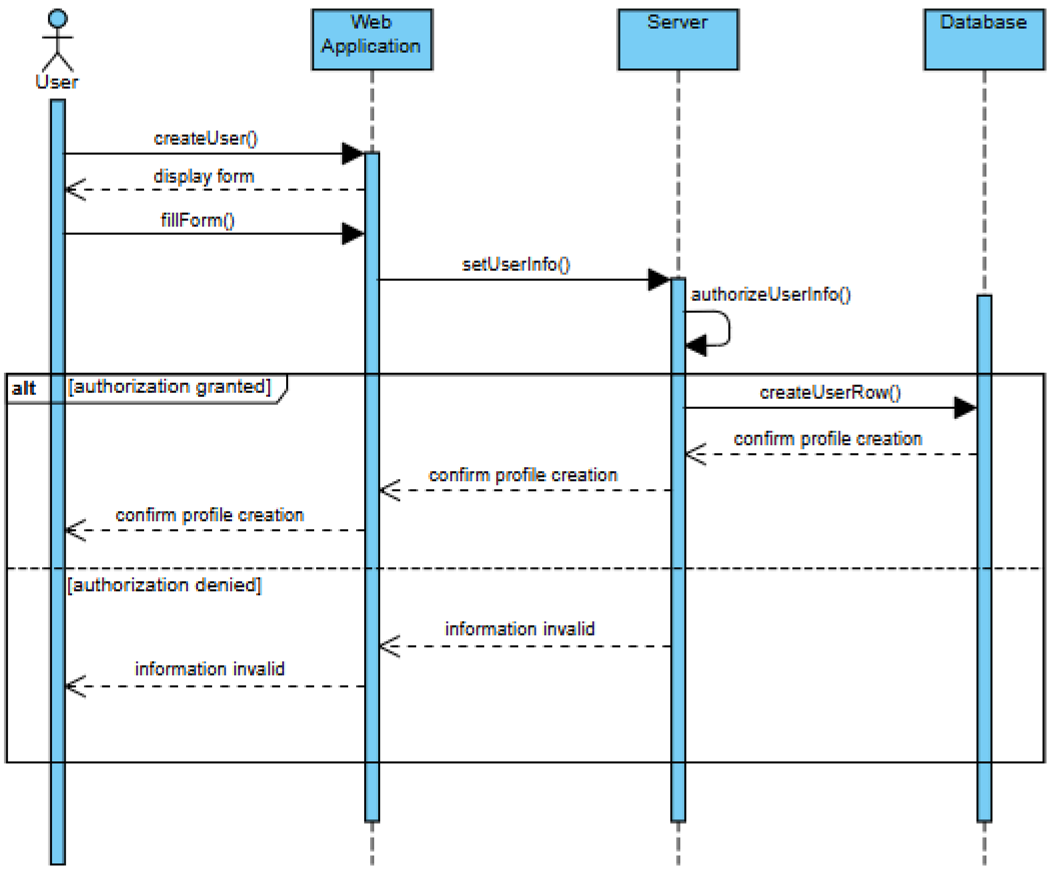


c. Sequence Diagrams

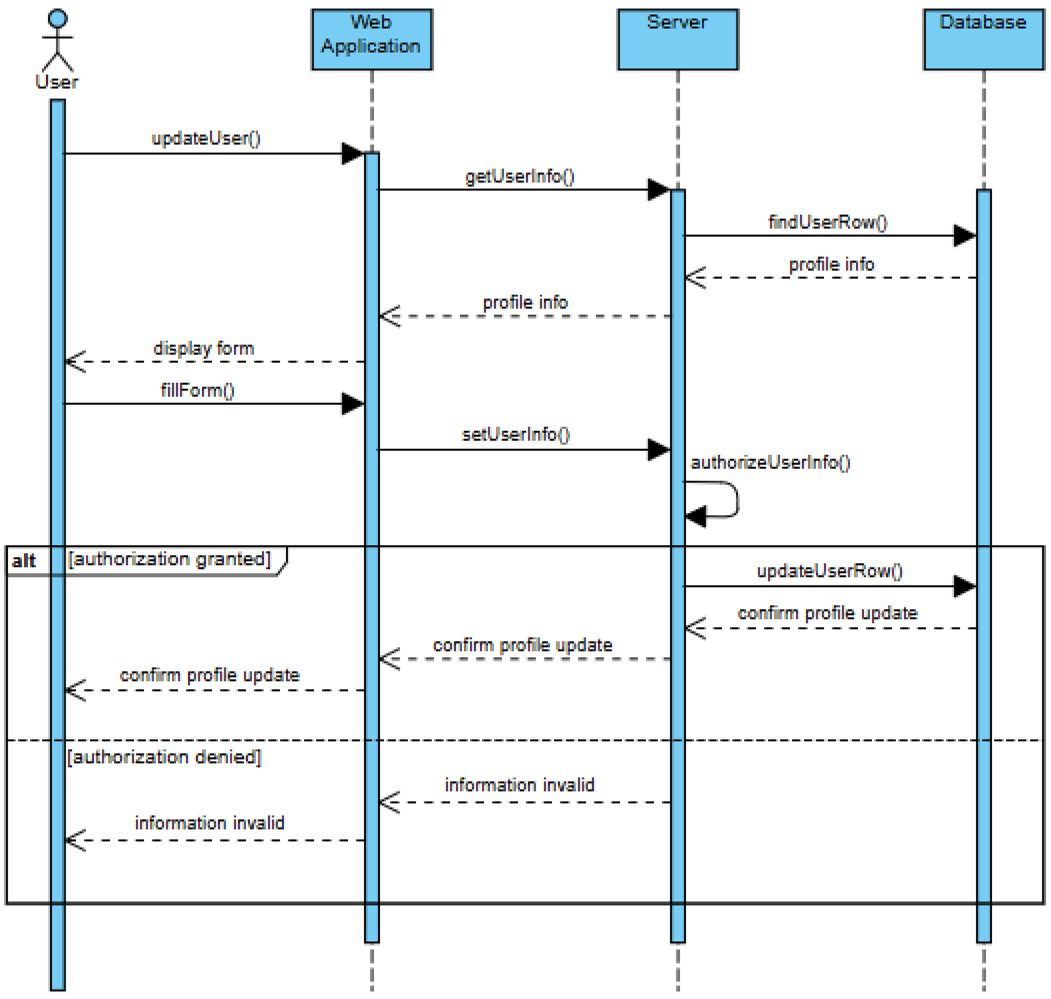
Swipe Sequence Diagram:



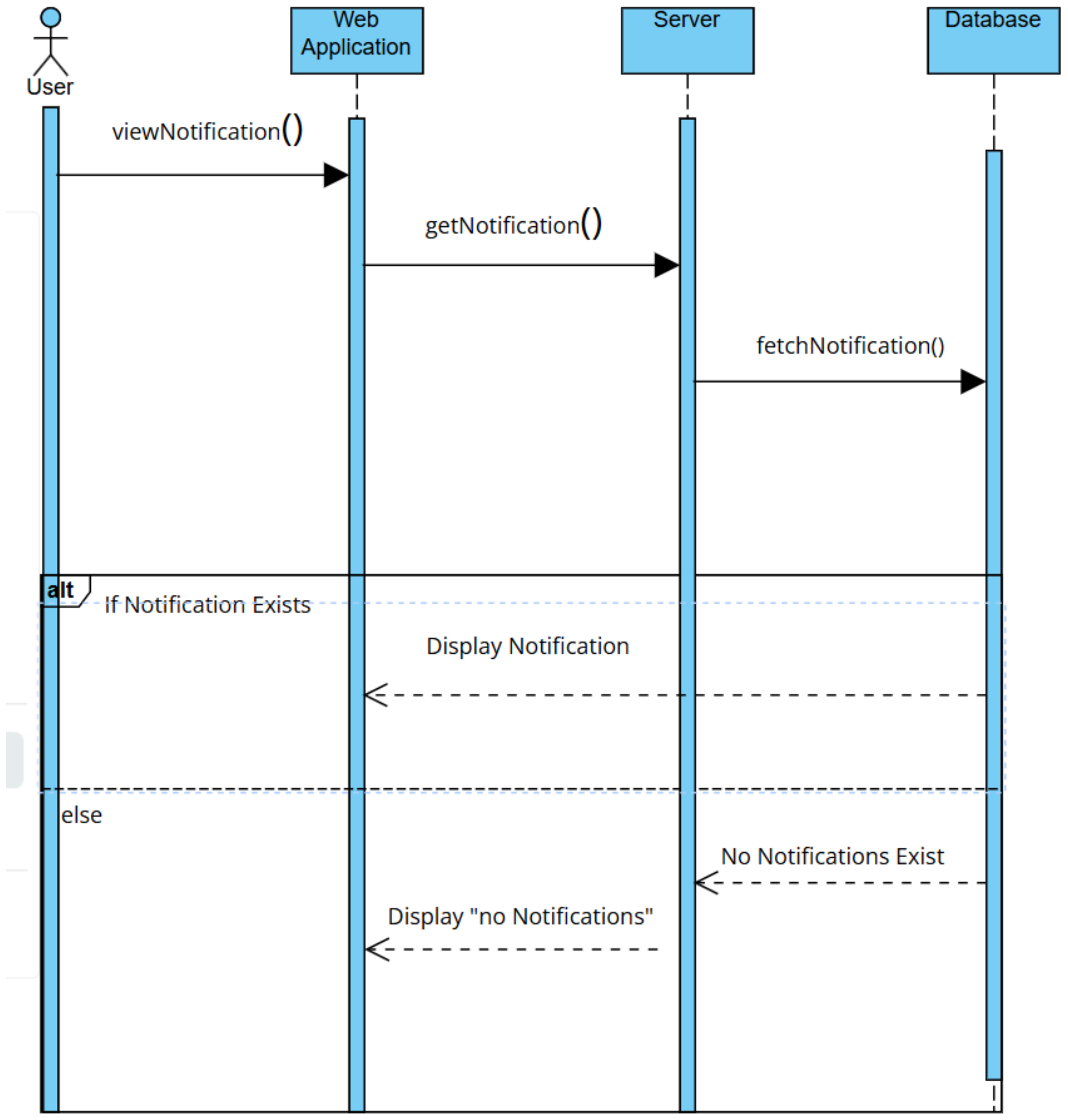
Create Profile Diagram



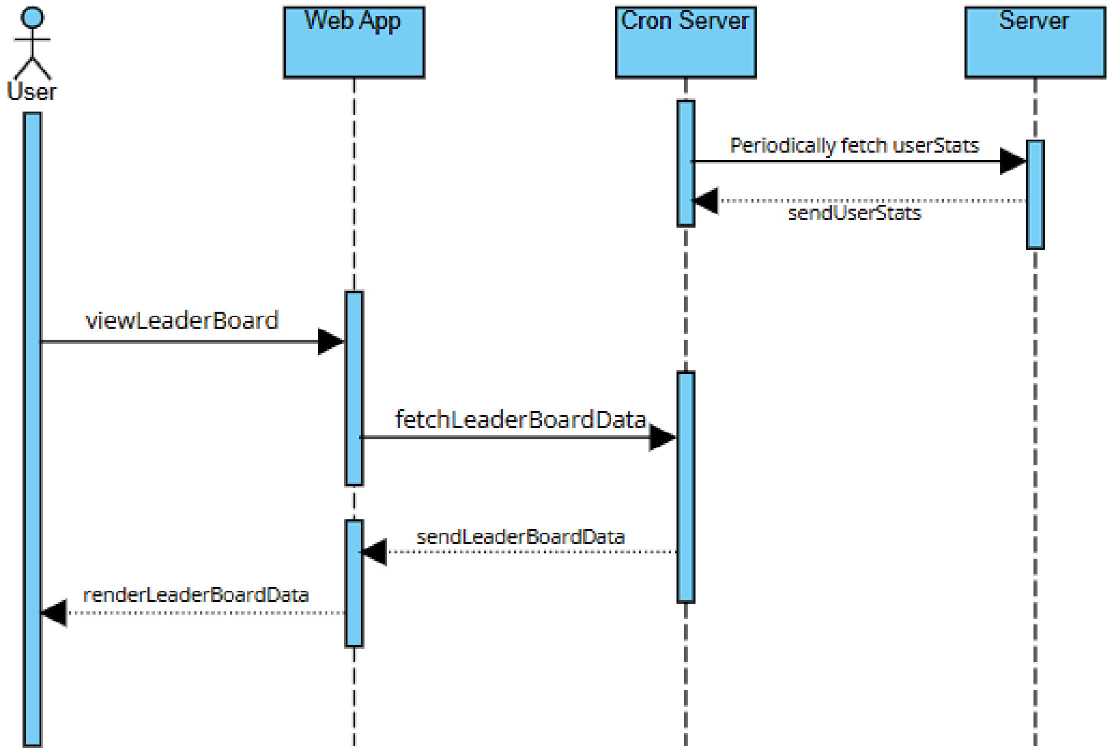
Edit Profile Diagram



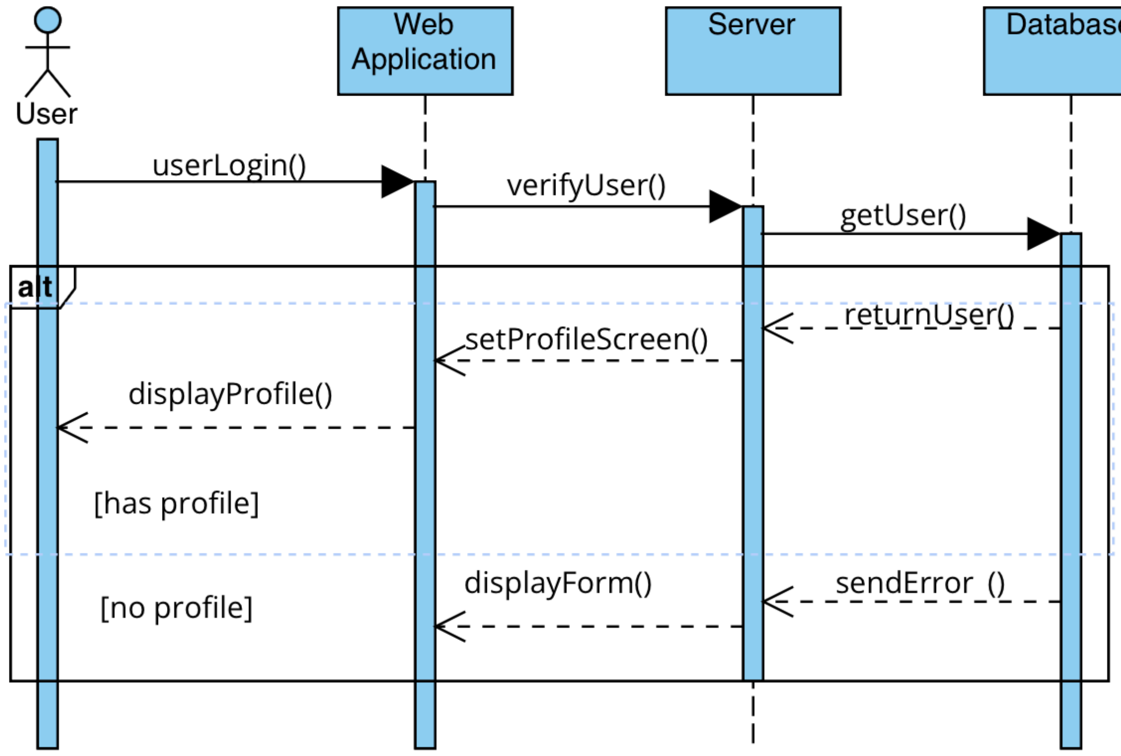
View Notifications Diagram



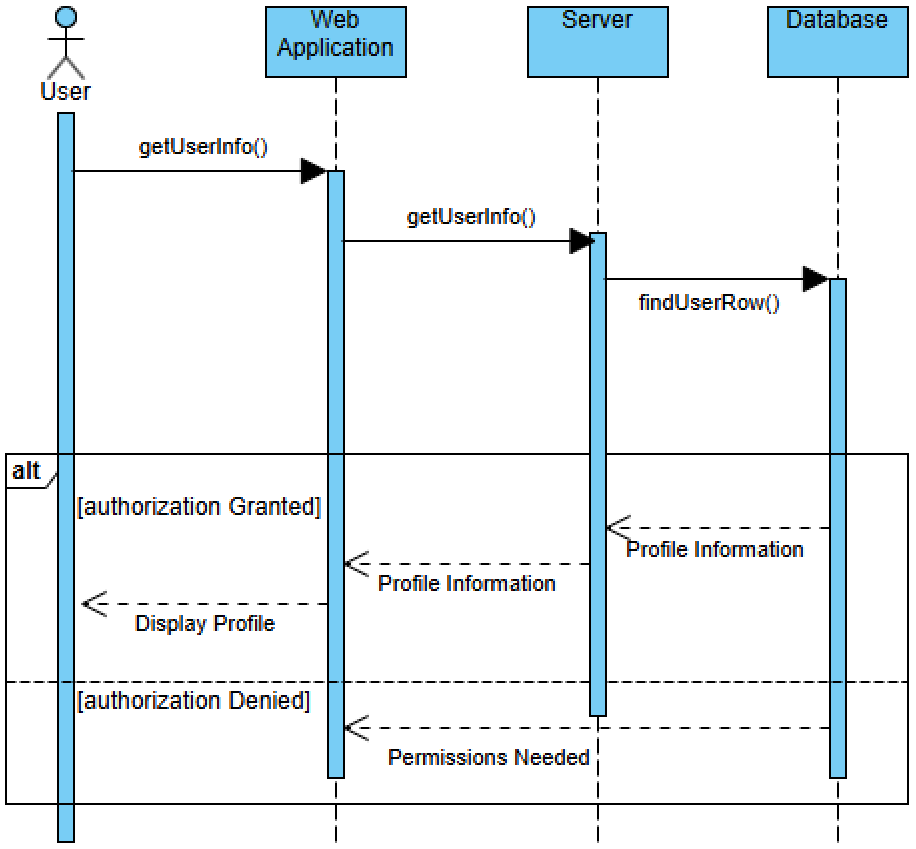
View LeaderBoard Diagram


User
Web App
Cron Server
Server
Periodically fetch userStats
sendUserStats
viewLeaderBoard
fetchLeaderBoardData
sendLeaderBoardData
renderLeaderBoardData

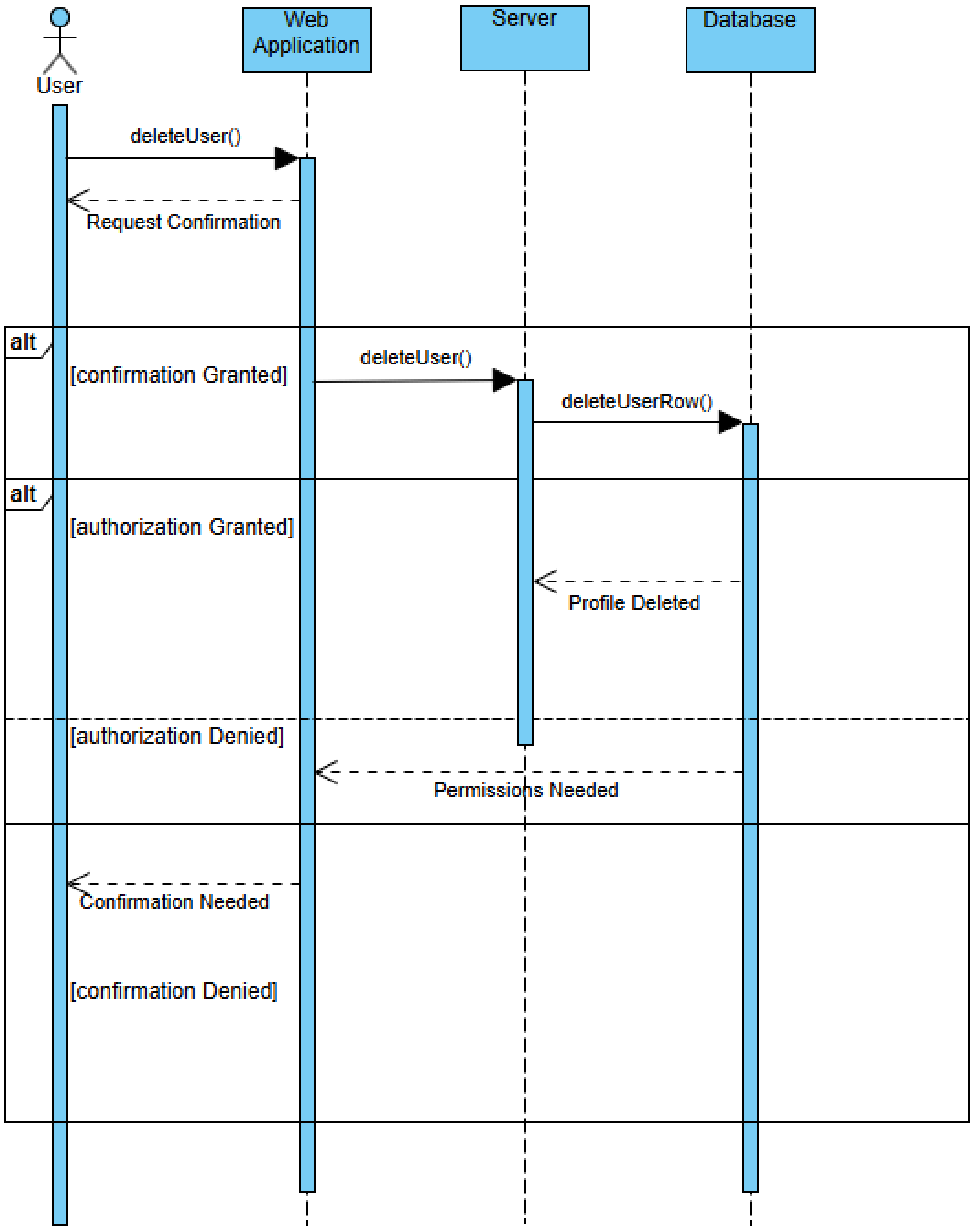
Login Diagram



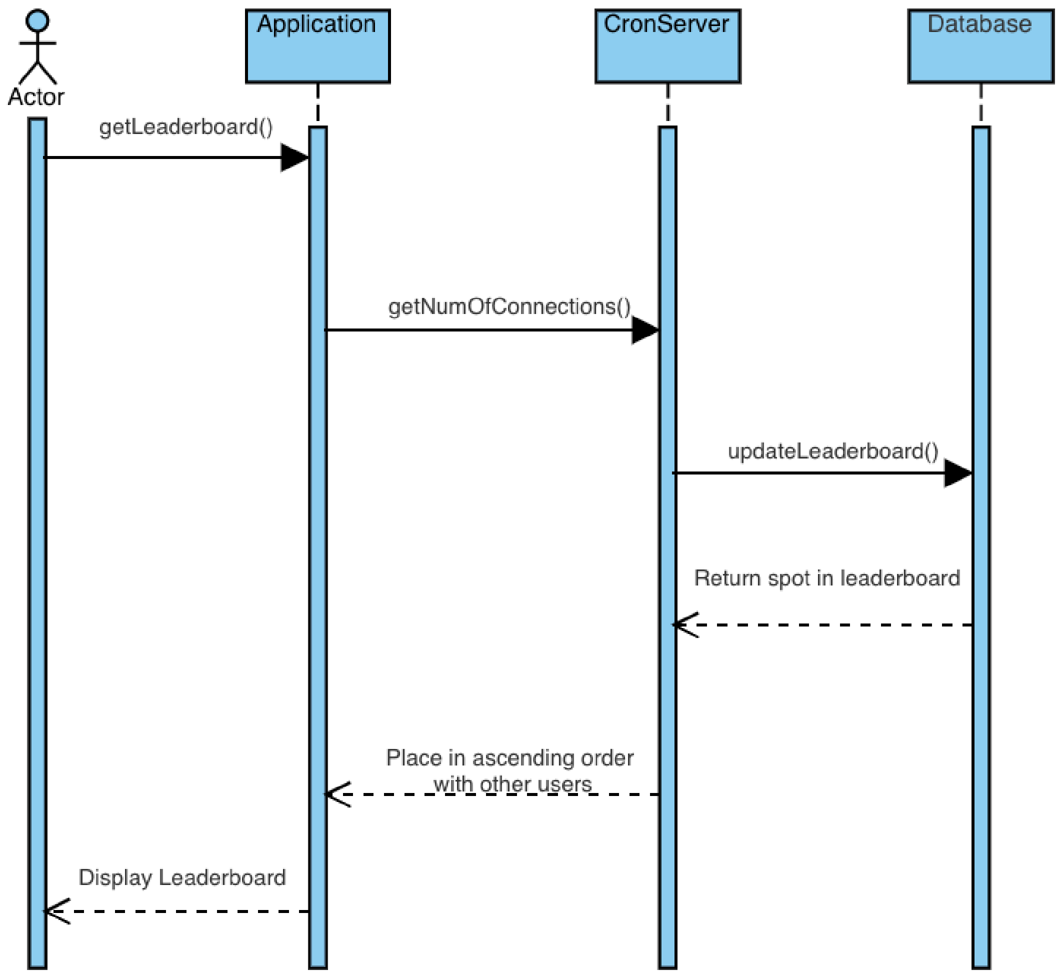
View Profile Diagram



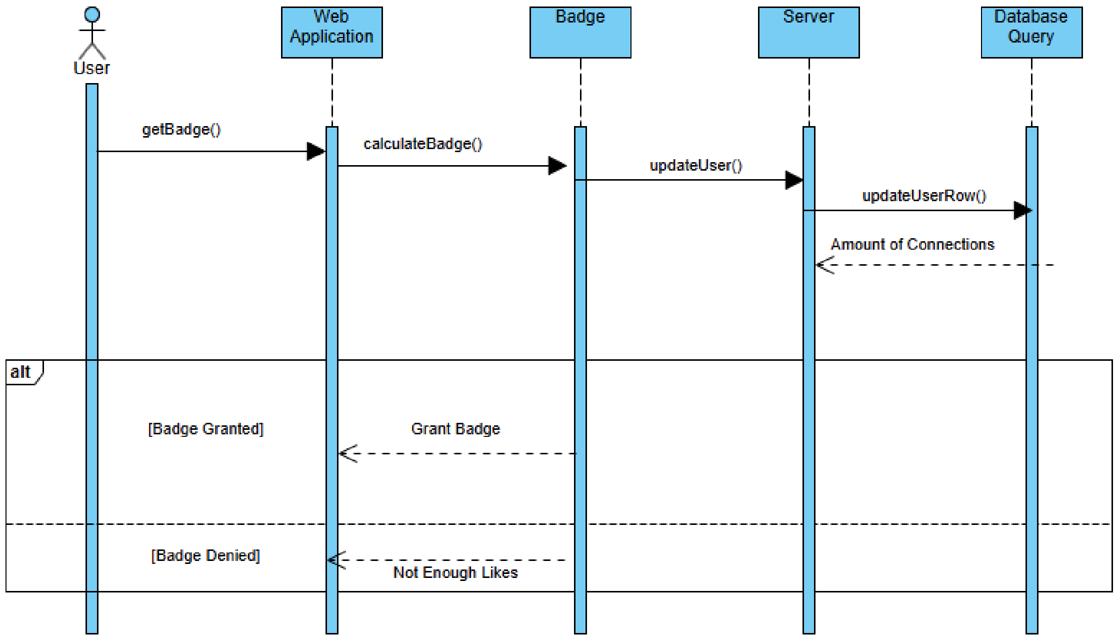
Delete Profile Diagram



Update LeaderBoard Diagram

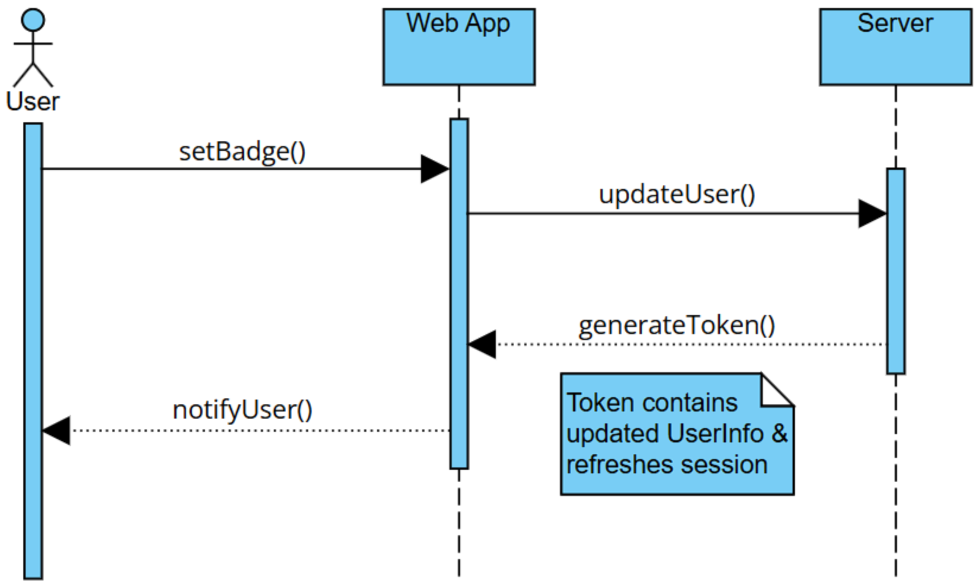


Badge Diagram

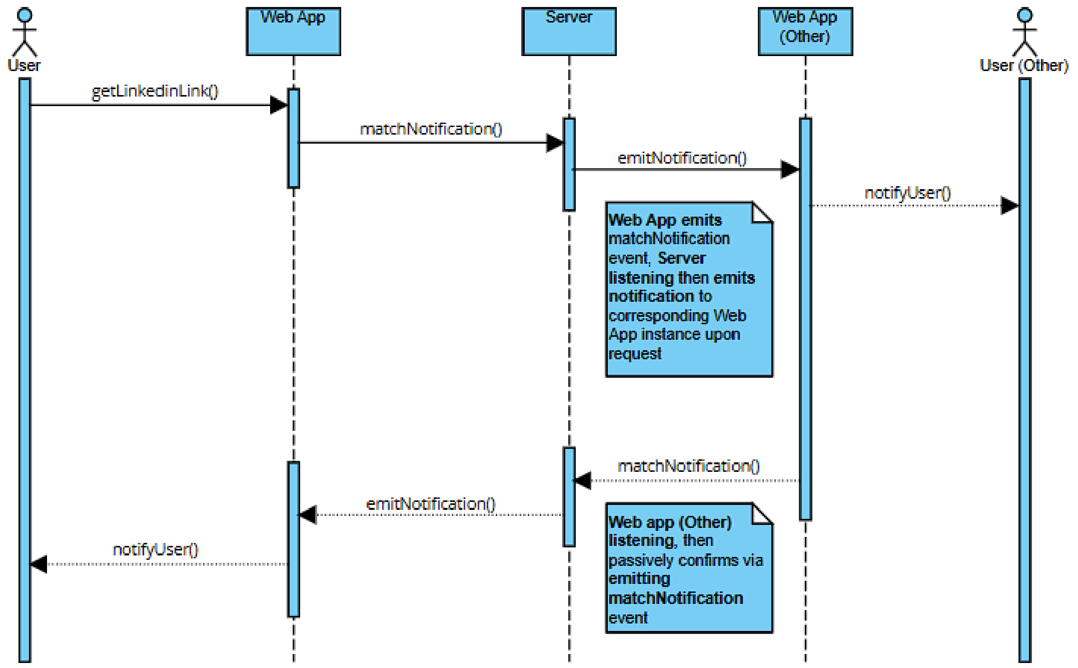

User
Web Application
Badge
Server
Database Query
getBadge()
calculateBadge()
updateUser()
updateUserRow()
Amount of Connections
alt
[Badge Granted]
Grant Badge
[Badge Denied]
Not Enough Likes

Notification System Diagrams

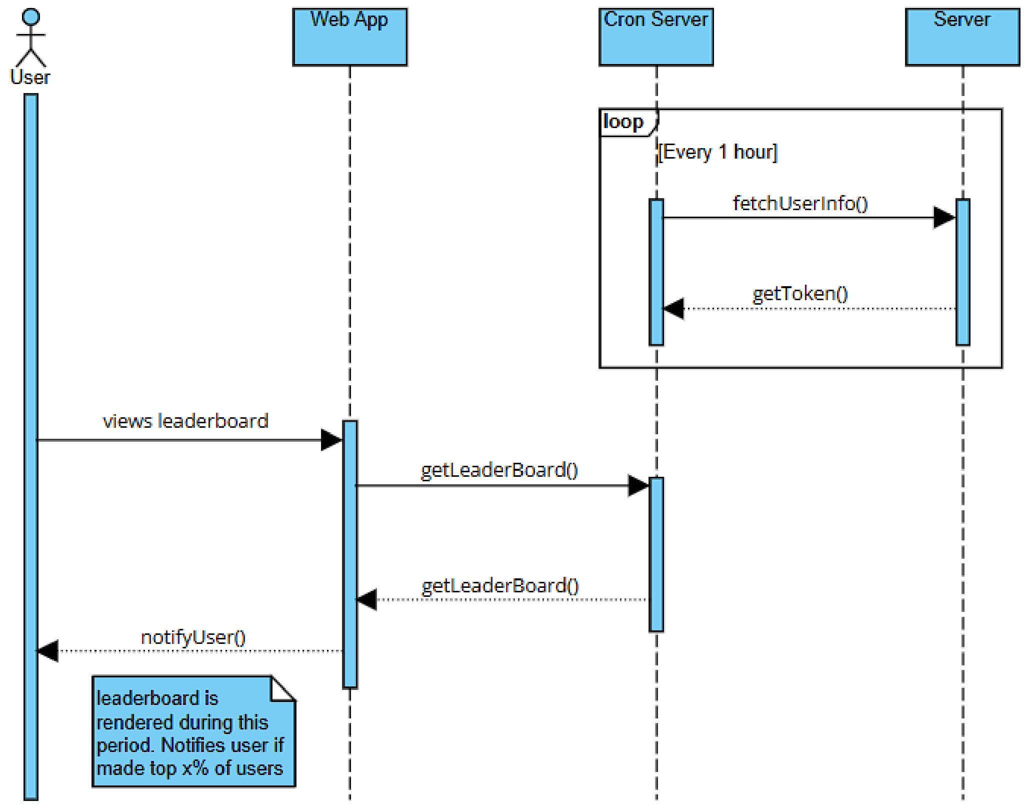
Badge



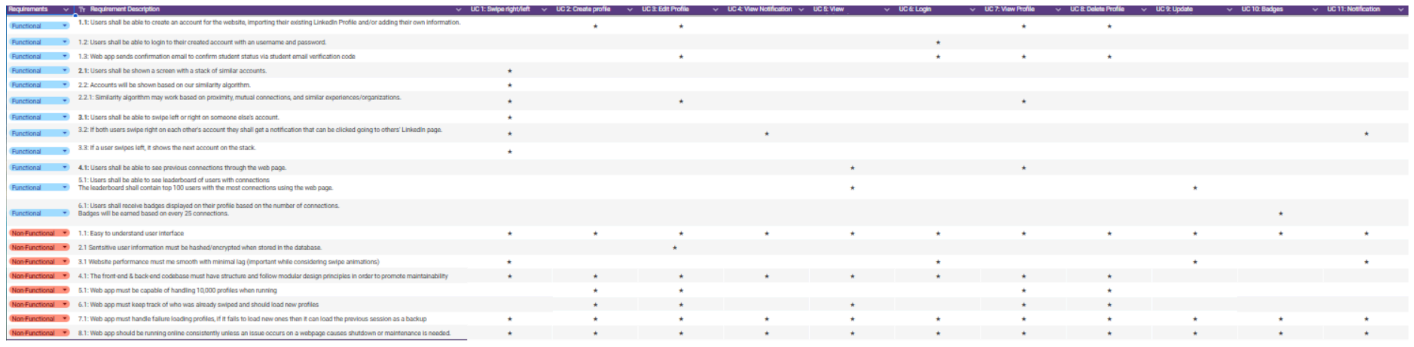
Match


User
Web App
Server
Web App (Other)
User (Other)
getLinkedinLink()
matchNotification()
emitNotification()
notifyUser()
Web App emits matchNotification event, Server listening then emits notification to corresponding Web App instance upon request
matchNotification()
emitNotification()
notifyUser()
Web app (Other) listening, then passively confirms via emitting matchNotification event

Leaderboard



d. Traceability Matrix



Requirements	Requirement Description	UC 1: Swipe right/left	UC 2: Create profile	UC 3: Edit Profile	UC 4: View Notification	UC 5: View	UC 6: Login	UC 7: View Profile	UC 8: Delete Profile	UC 9: Update	UC 10: Badges	UC 11: Notification
Functional	**1.1:** Users shall be able to create an account for the website, importing their existing LinkedIn Profile and/or adding their own information.		★	★				★	★			
Functional	1.2: Users shall be able to login to their created account with an username and password.						★					
Functional	1.3: Web app sends confirmation email to confirm student status via student email verification code			★			★	★	★			
Functional	**2.1:** Users shall be shown a screen with a stack of similar accounts.	★										
Functional	2.2: Accounts will be shown based on our similarity algorithm.	★										
Functional	2.2.1: Similarity algorithm may work based on proximity, mutual connections, and similar experiences/organizations.	★		★				★				
Functional	**3.1:** Users shall be able to swipe left or right on someone else's account.	★										
Functional	3.2: If both users swipe right on each other's account they shall get a notification that can be clicked going to others' LinkedIn page.	★			★							★
Functional	3.3: If a user swipes left, it shows the next account on the stack.	★										
Functional	**4.1:** Users shall be able to see previous connections through the web page.					★		★				
Functional	5.1: Users shall be able to see leaderboard of users with connections The leaderboard shall contain top 100 users with the most connections using the web page.					★				★		
Functional	6.1: Users shall receive badges displayed on their profile based on the number of connections. Badges will be earned based on every 25 connections.										★	
Non-Functional	1.1: Easy to understand user interface	★	★	★	★	★	★	★	★	★	★	★
Non-Functional	2.1 Sentsitive user information must be hashed/encrypted when stored in the database.			★								
Non-Functional	3.1 Website performance must me smooth with minimal lag (important while considering swipe animations)	★					★			★		★
Non-Functional	4.1: The front-end & back-end codebase must have structure and follow modular design principles in order to promote maintainability	★	★	★	★	★	★	★	★			
Non-Functional	5.1: Web app must be capable of handling 10,000 profiles when running		★	★				★	★			
Non-Functional	6.1: Web app must keep track of who was already swiped and should load new profiles		★	★		★		★	★			
Non-Functional	7.1: Web app must handle failure loading profiles, if it fails to load new ones then it can load the previous session as a backup	★	★	★	★	★	★	★	★	★	★	★
Non-Functional	8.1: Web app should be running online consistently unless an issue occurs on a webpage causes shutdown or maintenance is needed.	★	★	★	★	★	★	★	★	★	★	★

6. Software Requirements

a. Functional Requirements

- 1.1: Users shall be able to create an account for the website, importing their existing LinkedIn Profile and/or adding their own information.
    - 1.2: Users shall be able to login to their created account with an username and password.
    - 1.3: Web app sends confirmation email to confirm student status via student email verification code

- 2.1: Users shall be shown a screen with a stack of similar accounts.
    - 2.2: Accounts will be shown based on our similarity algorithm.
        - 2.2.1: Similarity algorithm may work based on proximity, mutual connections, and similar experiences/organizations.
- 3.1: Users shall be able to swipe left or right on someone else's account.
    - 3.2: If both users swipe right on each other's account they shall get a notification that can be clicked going to others' LinkedIn page.
    - 3.3: If a user swipes left, it shows the next account on the stack.
- 4.1: Users shall be able to see previous connections through the web page.
- 5.1: Users shall be able to see leaderboard of users with connections
    - The leaderboard shall contain top 100 users with the most connections using the web page.
- 6.1: Users shall receive badges displayed on their profile based on the number of connections.
    - Badges will be earned based on every 25 connections.

b. **Non Functional Requirements**

- 1.1: Easy to understand user interface
- 2.1: Sensitive user information must be hashed/encrypted when stored in db.
- 3.1: Website performance must be smooth with minimal lag (important when considering swipe animations)
- 4.1: The front-end & back-end codebase must have structure and follow modular design principles in order to promote maintainability.
- 5.1: Web app must be capable of handling 10,000 profiles when running
- 6.1: Web app must keep track of who was already swiped and should load new profiles
- 7.1: Web app must handle failure loading profiles, if it fails to load new ones then it can load the previous session as a backup
- 8.1: Web app should be running online consistently unless an issue occurs on a webpage causes shutdown or maintenance is needed.

7. Architectural Design - Client Service Architecture

7.1 An client server architecture will support real time interaction, frequent communication with databases, and scalability. Real time interaction is needed by LinkedOut due to constant feedback from users swiping on profiles, which is done by protocols by the web application. To get profiles to swipe for the database will request constant interaction with the database and with the structure of client server allows simple communication frequently between the user. Since this application will be a publicly available website it would be needed that the software is scalable due to the possible large amount of traffic that can interact with the system. Client service architecture is both horizontally scalable(more servers can be added) and vertical scalable(servers can be upgraded).