# Title Page

**Title**: Website Development MERN stack

**My Name**: Nikoloz Muladze

**Supervisor**: Alba Garcia

**Second Assessor**: Ana MATRAN-FERNANDEZ

**My degree course**: BSC G400 – Computer Science

# Acknowledgements

First of all, I am very thankful to my friends and parents who supported me and my wellbeing throughout the whole academic year and this project. Second of all, I am very thankful to those people and companies who made these platforms and packages used in this project available for public to use, for people like me. Furthermore, I am glad to mention all those people who make wonderful tutorials on YouTube that are free to watch and use for self-studies and development. Enormous thanks to Facebook/Meta for creating React and making it free and open-source. Mostly, throughout the development I used YouTube, Stack Overflow and official documentations as a main source of problem solving. I used official documentations of React, Express, MongoDB, Mongoose, and Next.js by Vercel, Socket.io, Node.js and npm packages that had their documentations on individual npm website pages respectively. I did not use JavaScript documentation, as odd as it would sound, as both server and client scripts are in JavaScript. The reason to why I did not use JavaScript documentation is that any knowledge I needed to have I received from tutorials and I have experience with it.

# Abstract/Summary (200 words)

For my project I decided to make a Social Media website, with features such as posting on a newsfeed, commenting and reacting on posts. Also being able to follow other users and chat with them. This includes authentication system which allows you to either login or register. This website was made using MERN stack. MongoDB, Express.js, React.js and Node.js.

Chatting feature gives you an ability to either send messages in private or in a group chat!

# Table of contents

Contents

[1. Title Page 1](#_Toc102116207)

[2. Acknowledgements 2](#_Toc102116208)

[Abstract/Summary (200 words) 3](#_Toc102116209)

[3. Table of contents 4](#_Toc102116210)

[4. List of symbols (1 page) 6](#_Toc102116211)

[5. Main Text (30 to 40 pages) 8](#_Toc102116212)

[1. About 8](#_Toc102116213)

[2. Motivation 8](#_Toc102116214)

[3. What did I use to make this project? And why? 8](#_Toc102116215)

[4. Literature review 10](#_Toc102116216)

[6. npm packages & external packages 12](#_Toc102116217)

[7. Server side packages 12](#_Toc102116218)

[1. bcrypt 12](#_Toc102116219)

[2. Cloudinary 13](#_Toc102116220)

[3. cors 13](#_Toc102116221)

[4. dotenv 13](#_Toc102116222)

[5. esm 13](#_Toc102116223)

[6. express-formidable 14](#_Toc102116224)

[7. express-jwt 14](#_Toc102116225)

[8. jsonwebtoken 14](#_Toc102116226)

[9. mongoose 14](#_Toc102116227)

[10. morgan 14](#_Toc102116228)

[11. nanoid & uuid 15](#_Toc102116229)

[12. nodemon 15](#_Toc102116230)

[13. path (not in use) 15](#_Toc102116231)

[14. socket.io 15](#_Toc102116232)

[8. Client side packages 15](#_Toc102116233)

[1. antd & @ant-design/icons 16](#_Toc102116234)

[2. Axios 16](#_Toc102116235)

[3. Bootstrap 16](#_Toc102116236)

[4. Moment 16](#_Toc102116237)

[5. React-dom 16](#_Toc102116238)

[6. React-toastify 16](#_Toc102116239)

[7. Socket.io-client 17](#_Toc102116240)

[9. External applications used for testing 17](#_Toc102116241)

[10. How to run the project on a fresh environment (tested only on Windows 10/11) 17](#_Toc102116242)

[1. Database 18](#_Toc102116243)

[2. Back-end 19](#_Toc102116244)

[3. Front-end 20](#_Toc102116245)

[11. GitLab repository link: 21](#_Toc102116246)

[12. Jira link: 21](#_Toc102116247)

[13. About application 21](#_Toc102116248)

[1. Home Page 21](#_Toc102116249)

[2. Register 22](#_Toc102116250)

[3. Login 23](#_Toc102116251)

[4. Reset Password 24](#_Toc102116252)

[5. Newsfeed 26](#_Toc102116253)

[6. Posting 27](#_Toc102116254)

[7. Commenting 28](#_Toc102116255)

[8. Editting & Deleting 30](#_Toc102116256)

[9. Search 32](#_Toc102116257)

[10. Follow & Unfollow 32](#_Toc102116258)

[11. Profile 35](#_Toc102116259)

[12. Chatting 38](#_Toc102116260)

[13. Sending messages 44](#_Toc102116261)

[14. Code 46](#_Toc102116262)

[1. server folder 46](#_Toc102116263)

[2. client folder 51](#_Toc102116264)

[6. Project Planning 60](#_Toc102116265)

[7. Conclusions (1-3 pages) 61](#_Toc102116266)

# List of symbols (1 page)

News Feed - list of newly published content on a website

PHP - Hypertext Preprocessor, a general-purpose scripting language

Backend & Frontend - In software engineering, the terms frontend and backend (or sometimes referred to as back end or back-end) refer to the separation of concerns between the presentation layer (frontend), and the data access layer (backend) of a piece of software, or the physical infrastructure or hardware. In the client–server model, the client is usually considered the frontend and the server is usually considered the backend, even when some presentation work is actually done on the server itself

Full stack - A full stack web developer is a person who can develop both client and server software

API - acronym for Application Programming Interface

URL - Uniform Resource Locator, colloquially termed a web address

Firebase - platform developed by Google for creating mobile and web applications

SDK - a set of software tools and programs provided by hardware and software vendors that developers can use to build applications for specific platforms.

The Twelve-Factor App methodology - methodology for building software-as-a-service applications. These best practices are designed to enable applications to be built with portability and resilience when deployed to the web

ECMAScript - ECMAScript is commonly used for client-side scripting on the World Wide Web, and it is increasingly being used for writing server applications and services using Node.js

UI – User interface

DOM - Document Object Model

Port - software-defined number associated to a network protocol that receives or transmits communication for a specific service

JWT\_SECRET - JWT is created with a secret key and that secret key is private to you which means you will never reveal that to the public or inject inside the JWT token. When you receive a JWT from the client, you can verify that JWT with this that secret key stored on the server

Bearer Token - predominant type of access token used with OAuth 2.0. A Bearer Token is an opaque string, not intended to have any meaning to clients using it. Some servers will issue tokens that are a short string of hexadecimal characters, while others may use structured tokens such as JSON Web Tokens

SEO - Search Engine Optimization

Args & Props & Params - a value that is passed between programs / functions

# Main Text (30 to 40 pages)

## About

This project is about Social Media web application, which involves features such as authentication, posting your thoughts on news feed so that people who follow you can reply or react to your thoughts through commenting on your post and liking it. Alternatively, you can chat with other users, either privately or through creating a group chat with several users in it. Being that said, you are also able to view other users profiles with information such as how many people they are following, how many followers they have, what do they have in description, and when did they join this social media.

## Motivation

Inspiration of this project is mainly Facebook, as it is known for being made by a well-known Harvard dropout Mark Zuckerberg in 2004 for, which he programmed in just two weeks using PHP. But, why did I make this social media, don’t we already have enough social media platforms? Well the motivation of this project is not making a product and then trying to outrun corporations that have net worth of billions of dollars with thousands of developers. The real motivation of this project lies in time. It took almost three to four years for Facebook to introduce chatting feature, which was released in March of 2008 that later in 2011 became Messenger that we know of today. So eventually the motivation is how fast the technology evolved and how much easier it is for developers to create features such as authentication, chatting and posting on news feed, that previously took others a lot of time, and hereby I made it in 7 months, which is probably much less considering how much time it took to do other assignments. Note that I worked on both backend and frontend, which in industry is called full-stack and companies usually have separate employees, one of them working on frontend and other working on backend.

## What did I use to make this project? And why?

In short, I used MERN stack, where MERN stands for MongoDB, Express, React, Node, four key technologies that make up this stack. But, why did I choose it? What makes it special? First and foremost, it is very important to choose the right technology to make any programming project successful, choosing C to make backend for this application would have been a nightmare and a waste of precious time. Let us make more reasonable example, before I discuss different stacks and platforms, let us start from Node.js, I mean clearly there is a huge list of choices I could make, for example PHP, Python or even Java. Well everyone could argue that PHP is somehow better than Node, because it has more features and it even can contain JavaScript, HTML and CSS. But it’s not always about the number of features something has right? Clearly, quality over quantity is always the answer, but this is another topic to argue about quality of both languages. So the only thing that comes into mind is experience with both languages that I have. And I can say with honour that Node.js has much easier learning curve than PHP. To fully unleash the powers of PHP, one should learn its famous framework Laravel. Well still personally I find Node much easier to grasp and maintain than PHP. As for community there is a huge number of sources for both PHP and Node. And what really makes Node stand out is that it is a server-side JavaScript run-time built on Chrome’s JavaScript Engine V8, so whatever stack I would choose, frontend would still be JavaScript as all browsers use JavaScript, and choosing Node just allowed me to have JavaScript both on frontend and backend and this is what I think makes it amazing.

Now that we chose Node, it only makes sense to use its framework Express.js to develop APIs faster and simpler, because using raw Node.js would take more time. Actually there are a lot of other choices, but community recommended to use Express.js so I just went with it, because it is too easy to get overwhelmed and I don’t yet know which one is the best. Express is a web framework that lets you structure a web application to handle multiple different http requests at a specific URL, which makes it really easy to understand, and application called Postman makes everything so much easier to test API calls

Okay but, why did I choose MongoDB over let us say MySQL, which is also so popular, well you would be amazed but MongoDB is actually way faster than MySQL and less complex. Studies show, if we believe them that MongoDB handled inserts nearly five times faster, selects 65% as fast and updates seven times faster compared to SQL. But, why is this, how? Well from personal experience, that’s because MySQL is a table and stores data in rows and columns and it is a relational database, because you are able to make joins from one table to another, whereas MongoDB is a collection of data, basically a JSON file, so instead of storing data in rows, MongoDB stores data in documents and fields and even though it is a non-relational database, you can still link one field to another. To put it simpler, in MongoDB you store data as an object, that can have an array of objects with data in it, and each of these objects can have arrays of objects.

Fine, but what about frontend? I have chosen that Node path already, I could use just raw html with JavaScript, but that would be more or less complicated and I wouldn’t be able implement some features, such as real-time rendering of components without refreshing the page. So I had to choose a JavaScript framework, from 3 different ones, which were React.js, Vue.js and AngularJS, and actually each of them have a stack name respectively: MERN, MEVN and MEAN. I crossed AngularJS out straight away as I didn’t really like it from the start as how complex it is, and so I had to choose either React or Vue, I have experience in both and it is really hard to say which one is better, because both are as simple as possible, both have huge communities, React being slightly bigger. The reason why I chose React is that it was actually made by an engineer at Facebook, yes the motivator of this project, and another reason was that actually React developer jobs pay better, which is also an important topic to consider when choosing a framework, just like C++ jobs are so highly regarded.

Now that is not where everything ends, I actually I could also choose between React frameworks, such as Next.js and Gatsby.js and I went with Next.js as it supports both static site generation and SSR (server side rendering), whereas Gatsby.js only supports static site generation

## Literature review

After choosing a technology, learning comes next so now I will discuss how I learned everything mentioned above, before I provide references. The first time I encountered website development was back in year one of this degree, in an associated course, CE154 to be specific. In an assignment I made a game browsing website using HTML, CSS, PHP and a use of JavaScript was optional, without any frameworks, so later in summer I felt like I really liked the subject so I read all the documentations for HTML, CSS, PHP and JavaScript on <https://www.w3schools.com> after reading all of these I found roadmaps for website development that I later eagerly followed:

Front-end roadmap: <https://roadmap.sh/frontend>

Back-end roadmap: <https://roadmap.sh/backend>

I watched a lot of tutorials regarding this subject, I will provide references only to ones that are related to this project. So I stumbled upon MERN stack in January 2021 on a YouTube channel called Clever Programmer <https://www.youtube.com/c/CleverProgrammer>

On this channel they are making clones of existing applications, like Amazon, Discord, even Facebook, but the thing I don’t like about them is their heavy use of Firebase (online database), and websites like Pusher (real-time experience), which are not suitable for commercial use in my opinion, because you will have to pay if the database gets bigger and bigger, which is an absolute rip off for anyone who prefers to spend time implementing such simple features themselves, without paying, instead of firebase it is much simpler to host the database yourself or find hosting services, which are much cheaper, as for Pusher, using Socket.io is much more sufficient, if you know MERN stack, might as well invest some time learning Socket.io which allows implementing real-time experience applications, without paying anything and suitable for commercial use.

So after discovering MERN stack, I searched more tutorials and learned everything from scratch. Here are some tutorials I did:

1. Node.js and Express.js – Full Course (8 hours and 16 minutes) <https://youtu.be/Oe421EPjeBE>
2. JavaScript Crash Course for Beginners (1 hour and 40 minutes) <https://youtu.be/hdI2bqOjy3c>
3. React JS Full Course for beginners (8 hours and 49 minutes) <https://youtu.be/RVFAyFWO4go> (I did this one as a refresher) As I mostly used official documentation to learn React
4. MongoDB Tutorial For Beginners (1 hour and 8 minutes) <https://youtu.be/Www6cTUymCY>
5. Next.js Crash Course <https://youtu.be/mTz0GXj8NN0>

Official documentations used to learn MERN stack:

1. React (version 17) Docs - <https://reactjs.org/blog/2020/10/20/react-v17.html>
2. Next Docs - <https://nextjs.org/docs>
3. Node (version 16.x) Docs - <https://nodejs.org/dist/latest-v16.x/docs/api/>
4. Express (version 4.x) Docs - <https://expressjs.com/en/4x/api.html>
5. MongoDB (version 5.0) Docs - <https://www.mongodb.com/docs/manual/tutorial/getting-started/>

## npm packages & external packages

Before I discuss any external packages, I know we are required to use GitLab as repository, so it makes sense to talk about Git bash that I’m using for version control, pushing commits to the GitLab, to download external packages using npm, run server, client and database. I will explain how to use npm later, for now let us focus on external packages I used. Everything that is installed from external packages does not mean it is used in current implementation of the application, but it means I tested them during development

## Server side packages

Let us start from server side packages:

1. Bcrypt - <https://www.npmjs.com/package/bcrypt>
2. Cloudinary - <https://www.npmjs.com/package/cloudinary>
3. Cors - <https://www.npmjs.com/package/cors>
4. Dotenv - <https://www.npmjs.com/package/dotenv>
5. Esm - <https://www.npmjs.com/package/esm>
6. Express - <https://www.npmjs.com/package/express> (yes, this is Node framework)
7. Express-formidable - <https://www.npmjs.com/package/express-formidable>
8. Express-jwt - <https://www.npmjs.com/package/express-jwt>
9. Jsonwebtoken - <https://www.npmjs.com/package/jsonwebtoken>
10. Mongoose - <https://www.npmjs.com/package/mongoose>
11. Morgan - <https://www.npmjs.com/package/morgan>
12. Nanoid - <https://www.npmjs.com/package/nanoid>
13. Nodemon - <https://www.npmjs.com/package/nodemon>
14. Path - <https://www.npmjs.com/package/path>
15. Socket.io - <https://www.npmjs.com/package/socket.io>
16. Uuid - <https://www.npmjs.com/package/uuid>

These packages are so called dependencies for the project, if they don’t get installed, application will not run!

### bcrypt

Bcrypt is a password hashing function designed by Niels Provos and David Mazières, based on the Blowfish cipher and presented at USENIX in 1999. Besides incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power. Wiki: [bcrypt - Wikipedia](https://en.wikipedia.org/wiki/Bcrypt)

### Cloudinary

I really do not want to include this package in explanation as it was planned to be temporary and later be replaced by package called multer, but time is my enemy and I really don’t want this to be regarded as part of my project, as I consider it as a part where I failed to implement something. It is an SDK which allows you to integrate the application with Cloudinary (<https://cloudinary.com/>) an application where you can upload files, such as images.

### cors

I used cors, because without it I was getting an error in the client side and was not able to fetch the data. Essentially it is a node package for providing middleware to connect and express, to avoid errors. Middleware is just a function that executes before executing the goal function. (I will explain what a middleware in details is when I will start discussing code) I do not know how it works as I did not make it, and it is just one line and avoids any error in my server-side application so as long as it works I’m fine with it

### dotenv

Dotenv allows JavaScript files, in this case Node files to read from .env file by typing process.env to access constant variables in env file. Env is separate from the code and is based on The Twelve-Factor App methodology. These type of files are usually excluded from git repositories as it mostly contains sensitive information, such as password hashing algorithm name, database link etc.

### esm

Esm package allows the use of ECMAScript in Node.js. To put it simply, you are allowed to use keywords that are otherwise not available in basic Node.js. For example instead of writing:

const express = require("express");

You are allowed to write:

import express from "express";

Which is essentially same

### express-formidable

This is what I used with cloudinary, but it works well with multer, basically it allows you to parse form data including multipart/form-data. This is the data received from client side, which has a type of a file. In our case an image.

### express-jwt

This node module provides a middleware for express so it is able to validate JWT (JSON web tokens) **Warning:** JWTs are credentials, which can grant access to resources. Be careful where you paste them! We do not record tokens, all validation and debugging is done on the client side. [JSON Web Tokens - jwt.io](https://jwt.io/) basically used for encoding and decoding hashed passwords

### jsonwebtoken

<https://datatracker.ietf.org/doc/html/rfc7519> – “JSON Web Token (JWT) is a compact, URL-safe means of representing

claims to be transferred between two parties. The claims in a JWT

are encoded as a JSON object that is used as the payload of a JSON

Web Signature (JWS) structure or as the plaintext of a JSON Web

Encryption (JWE) structure, enabling the claims to be digitally

signed or integrity protected with a Message Authentication Code

(MAC) and/or encrypted.”

### mongoose

Mongoose is one of the important packages used as it allows the code to read MongoDB and model it, such as write or read. And it works in asynchronous environment, which I am using.

### morgan

Morgan is HTTP request logger, so let us say we got a GET or POST request from client side to fetch data or to upload it. Basically, in server controllers’ folder, each file has asynchronous functions that have (req, res) as arguments, or (req, res, next) this will be discussed thoroughly in code section. Req and Res stand for Request and Response, so to simplify these HTTP requests, morgan comes in handy to output simplified logs in console. Morgan is essentially a logger, on any requests being made, it generates logs automatically.

### nanoid & uuid

These two have the same purpose, generate random strings, in my project’s case, for example usernames are generated using Nanoid. Nanoid is two times faster than uuid if we believe what is written in [nanoid - npm (npmjs.com)](https://www.npmjs.com/package/nanoid) here

### nodemon

The only purpose this package had for me was to run the server and keep it up and running, basically a live-server for Node application, if I made any changes to code and saved the file it would automatically refresh the server, so I would not have to go to command line and manually refresh the server.

### path (not in use)

Path is for accessing and interacting with the file system

### socket.io

“Socket.IO enables real-time bidirectional event-based communication”. This package by itself is nothing, there is a client-side part of this package that is important to run to see any results

## Client side packages

1. @ant-design/icons - <https://ant.design/components/icon/>
2. Antd - <https://ant.design/components/overview/>
3. Axios - <https://axios-http.com/docs/intro>
4. Bootstrap - <https://getbootstrap.com/>
5. Moment - <https://www.npmjs.com/package/moment>
6. Next (React Framework) - <https://nextjs.org/docs>
7. React (mentioned above) - <https://reactjs.org/blog/2020/10/20/react-v17.html>
8. React-dom () - <https://www.npmjs.com/package/react-dom>
9. React-toastify - <https://www.npmjs.com/package/react-toastify>
10. Socket.io-client - <https://socket.io/docs/v4/client-api/>

### antd & @ant-design/icons

Antd – Ant Design is a UI Library for reusable components such as buttons, Inputs, Modals, Drawers and etc. More in detail at: [Components Overview - Ant Design](https://ant.design/components/overview/) same goes for @ant-desing/icons which only includes icons [Icon - Ant Design](https://ant.design/components/icon/). It comes in handy during the development as you will not have to go to CSS every time and style every single button there is in the project. Actually there were other better choices that I regret not making, like Material UI and Chakra UI. The reason why I chose Ant Design was my friend, who recommended me to use it as he is working in this industry and said that it is a decent UI library, but apparently it all depends on what you prefer to use.

### Axios

One of the most important packages in client side, it allows client to send HTTP requests to the server asynchronously and fetch the data from it.

### Bootstrap

Bootstrap is a famous front end development framework for creation of websites, which consists of raw CSS and Javascript. There are traces of heavy use of Bootstrap in this project, but you will see that the line where Bootstrap is imported is commented out, instead parts of it are located in my CSS file, largely to the reason of some Bootstrap 4 styles not being available in Bootstrap 5 and vice versa, and I avoided importing both versions as there would have been a lot of clashes.

### Moment

Moment package is used for formatting dates, this library is the reason why application is able to display for example: posted “11 days ago”

### React-dom

This package allows interaction with the DOM (Document Object Model)

### React-toastify

This package is used for error handling in the client side, if any error occurs the notification message is displayed at the top centre of the page.

### Socket.io-client

Socket.io-client is an extension of a package Socket.io from the back-end, which gives the purpose to Socket.io as both are integral parts of each other and gives ability to implement real-time bidirectional event-based communication

## External applications used for testing

I used Postman and MongoDB Compass for testing the backend.

Postman - <https://www.postman.com/>

MongoDB Compass - <https://www.mongodb.com/try/download/compass>

Postman is mainly used for API testing, by sending HTTP requests to the server.

MongoDB Compass is a GUI (Graphical User Interface) for checking the MongoDB database, command line could be used instead of this application, but it is a real game changer that saves a lot of time

As for front-end nothing was used for testing, apart from Chrome browser’s built-in inspect elements features such as Console, Network, Application and Elements

## How to run the project on a fresh environment (tested only on Windows 10/11)

First of all, I would recommend downloading Git Bash on the system to be able to clone the repository to the preferred path or simply download the zip file from GitLab.

Download the repository as a ZIP file: <https://cseegit.essex.ac.uk/ce301_21-22/CE301_muladze_nikoloz/-/archive/master/CE301_muladze_nikoloz-master.zip>

Or

SSH:

git clone [git@cseegit.essex.ac.uk:ce301\_21-22/CE301\_muladze\_nikoloz.git](mailto:git@cseegit.essex.ac.uk:ce301_21-22/CE301_muladze_nikoloz.git)

HTTPS:

git clone <https://cseegit.essex.ac.uk/ce301_21-22/CE301_muladze_nikoloz.git>

I prefers using https method.

Git version I am using is 2.35.2

### Database

(IMPORTANT\*) Now following onto the next step, MongoDB Community Sever must be installed!

MongoDB version I am using is 5.0.3

Installation link: <https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-5.0.3-signed.msi>

How to install MongoDB: <https://youtu.be/3wqzr-GJoS0>

**It is important to add the mongodb bin folder to the path to run the database server**

Database by default is located in the folder where the MongoDB is installed but it is also possible to create a manual directory on a C drive.

Following will do that:

**cd ~**

**mkdir data && cd data && mkdir db**

By default MongoDB database runs on port 27017 <http://localhost:27017>

But to ensure MongoDB is running do the following

**mongod**

After that run the following to check what the MongoDB port is

**mongo**

You will see information such as “connecting to: mongodb://127.0.0.1:27017/…”

If 27017 is different you will have to modify .env file in **/server/.env** and change 27017 with the port numbers from the command line respectively.

You will have to leave the command line opened where you are running **mongod**. You can exit out of any other program.

### Back-end

**(IMPORTANT\*) now you must install Node.js**

The version of Node I am using is 16.13.0

Download Node: <https://nodejs.org/download/release/v16.13.0/node-v16.13.0-x64.msi>

As far as I know npm version is tied to Node version, but still I will mention it.

Npm version I am using in this project is 8.1.0

**(IMPORTANT\*) when installing Node.js, remember to add it to PATH as without it, it will not be possible to run the server and client.**

How to install Node.js - https://youtu.be/3F5IaPqj7ds

Now open up a command prompt or a git bash and go to the directory where you cloned this project to, then type

cd server

and type:

npm i

“npm i" will check the package.json file and install all the dependencies the server has, once it’s done installing you will see a new folder appear, call node\_modules. (This step is optional: if command prompt asks you to do “npm audit fix” you can run that, it will remove deprecated files, though I would recommend first trying to run the server to see if it works. If it works, leave it be.)

To run the server type in the following in the same CMD or Git bash:

npm start

Once “DB Connected!” is displayed in the CMD, means everything is fine and server is running.

### Front-end

Now to run the client, open another CMD or Git bash and go to the directory where you cloned this project to, then type:

cd client

and type:

npm i

“npm i" will do the same thing as in the back-end. node\_modules folder will appear and might ask to do “npm audit fix”.

Now to start the client you can choose from these 2 ways:

First way:

npm run dev

This will run the client and show the URL that will open the application which is:

<http://localhost:3000>

Second way:

If for some reason the previous step did not work I suggest building the app, this way will run client much faster and the app will be faster

To build the app type in:

npm run build

This will build the app and the /.next folder will appear in the client folder

Now to run the client type in:

npm start

And you are good to go and use the application.

It is important to first run the database, then server and client at last

Now, I know that took a while, but I would like to notice that these steps would not be here if the app was deployed and hosted, which I did not do, but is a possibility.

## GitLab repository link:

<https://cseegit.essex.ac.uk/ce301_21-22/CE301_muladze_nikoloz>

## Jira link:

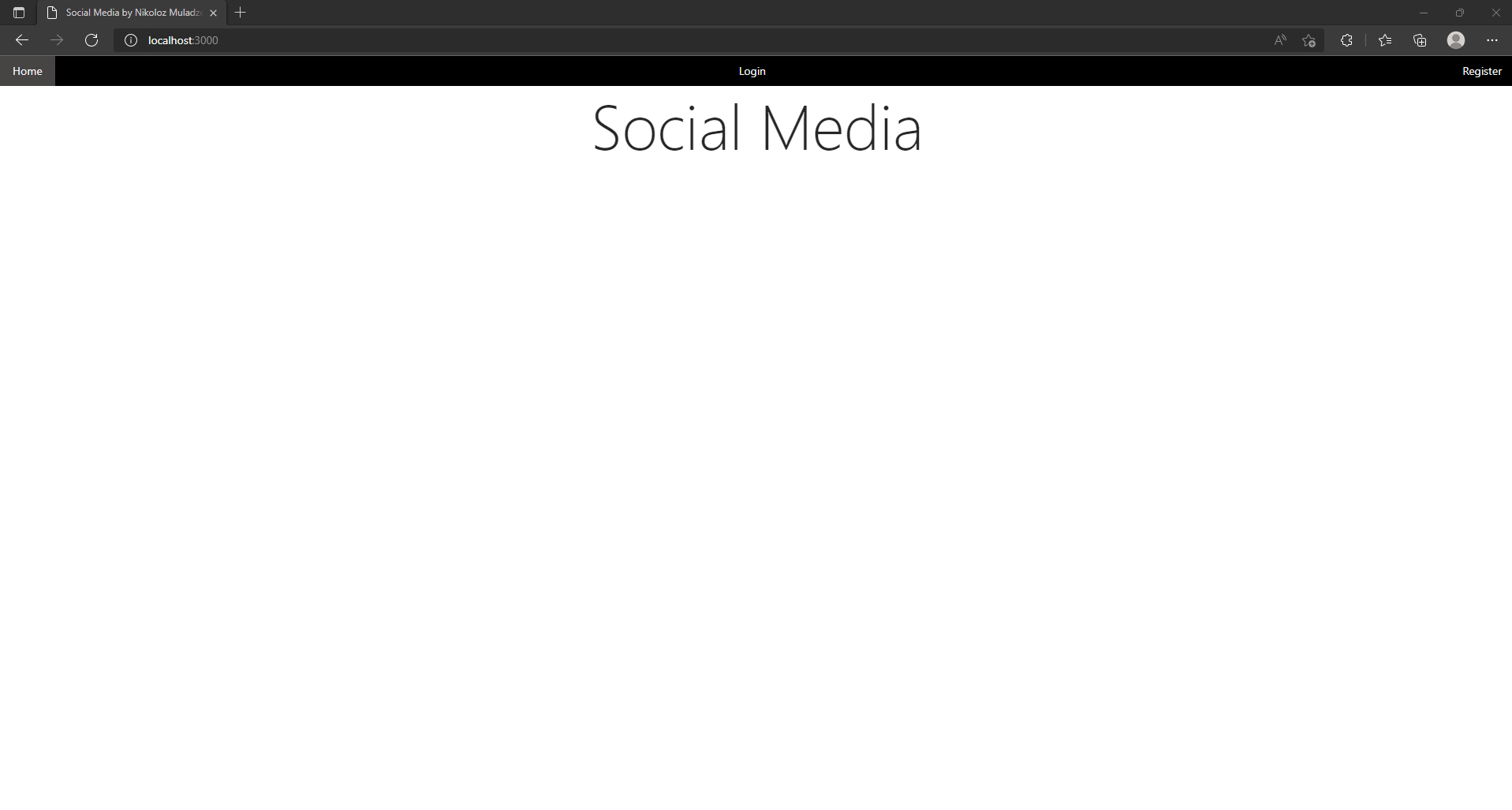
<https://cseejira.essex.ac.uk/secure/RapidBoard.jspa?rapidView=5596&projectKey=B301024>

I am not sure how to copy Jira properly, but this report will be included in Jira too

## About application

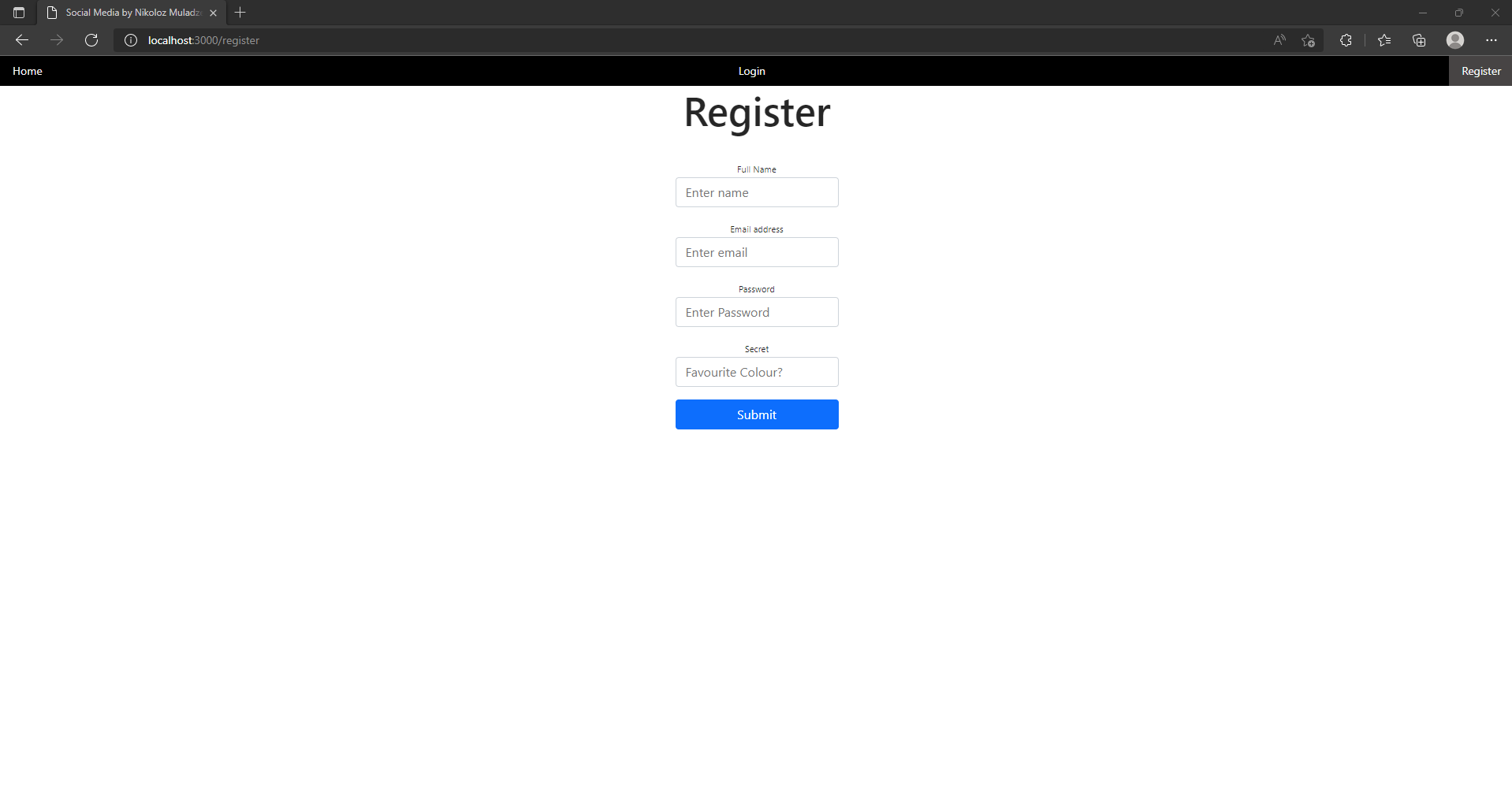
Now it is possible to check out the application. Once you visit <http://localhost:3000> you will be directed to the home page:

### Home Page



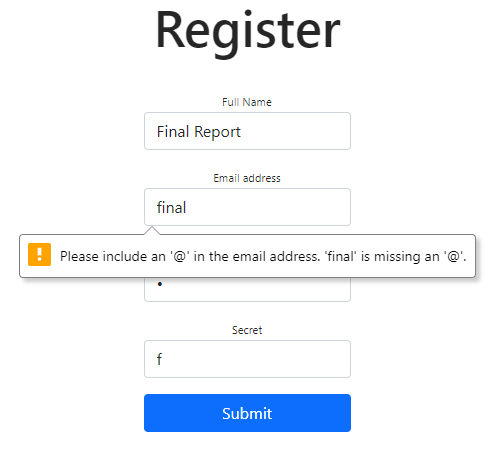
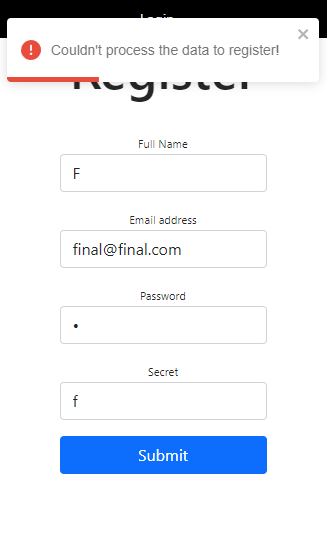
From here you can visit 3 other pages.

### Register

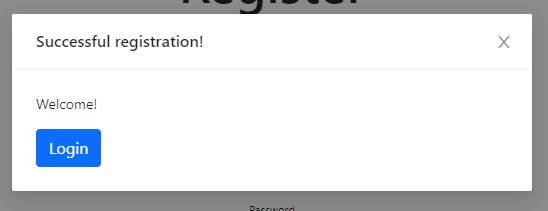
First let us visit Register page by pressing Register in the navigation bar: 

Here you will have to enter what is asked to register.

Possible errors:

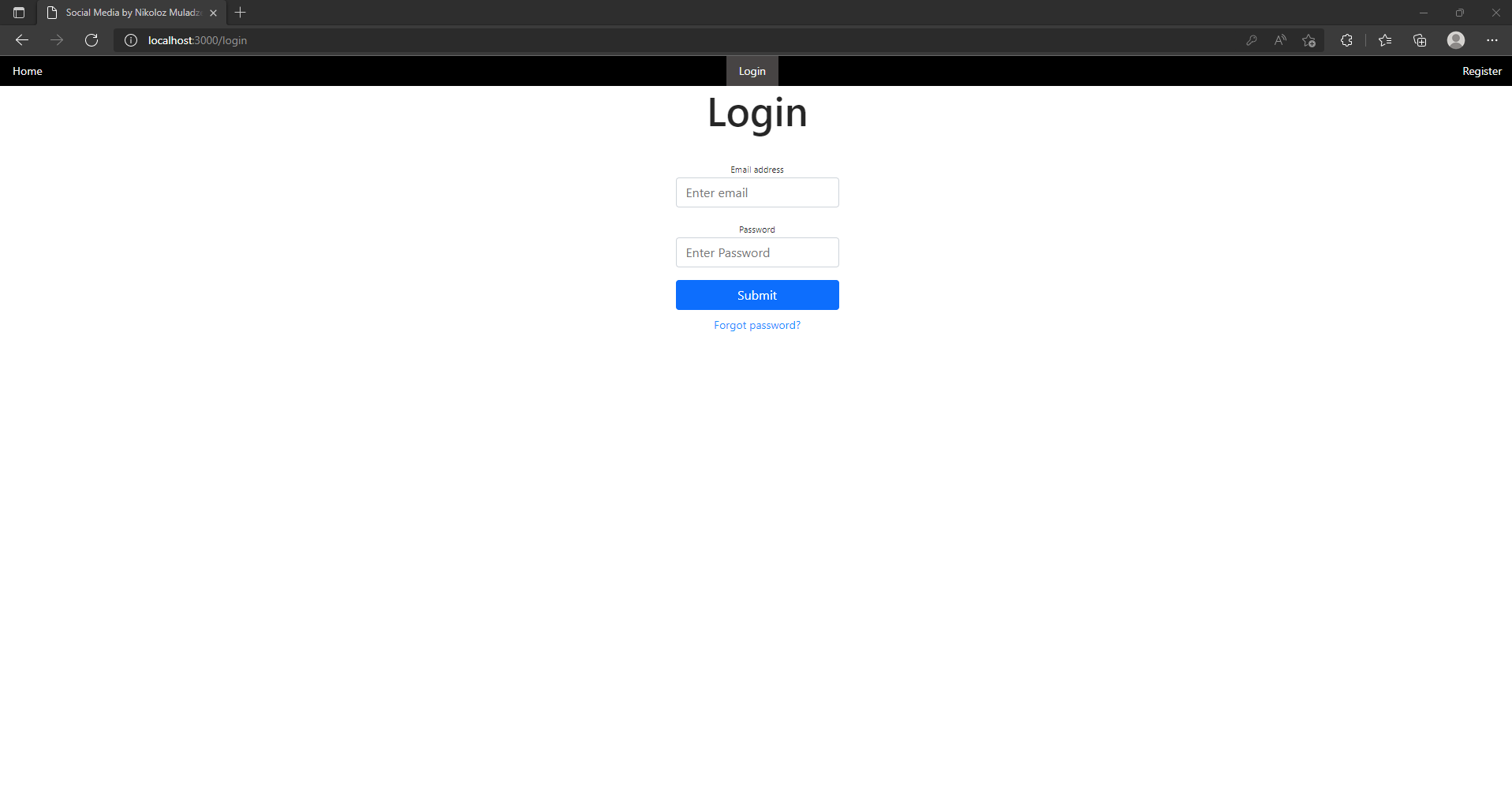


All 4 fields must be filled in, password must be either 6 characters long or more, email should not be from the existing ones.

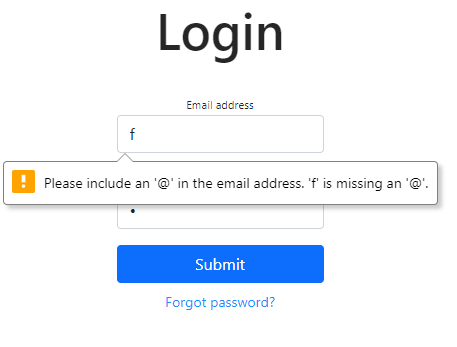
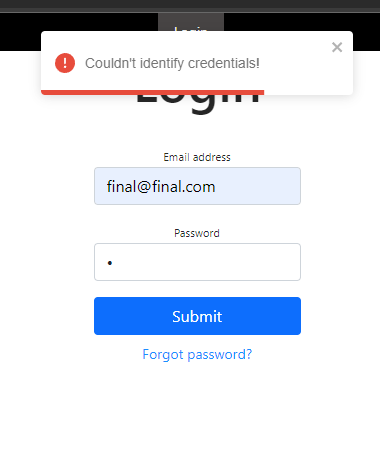
Once registration is successful you will see: 

### Login

Now you can press Login or close it and press Login in the Navigation bar and you will be directed to the login page:

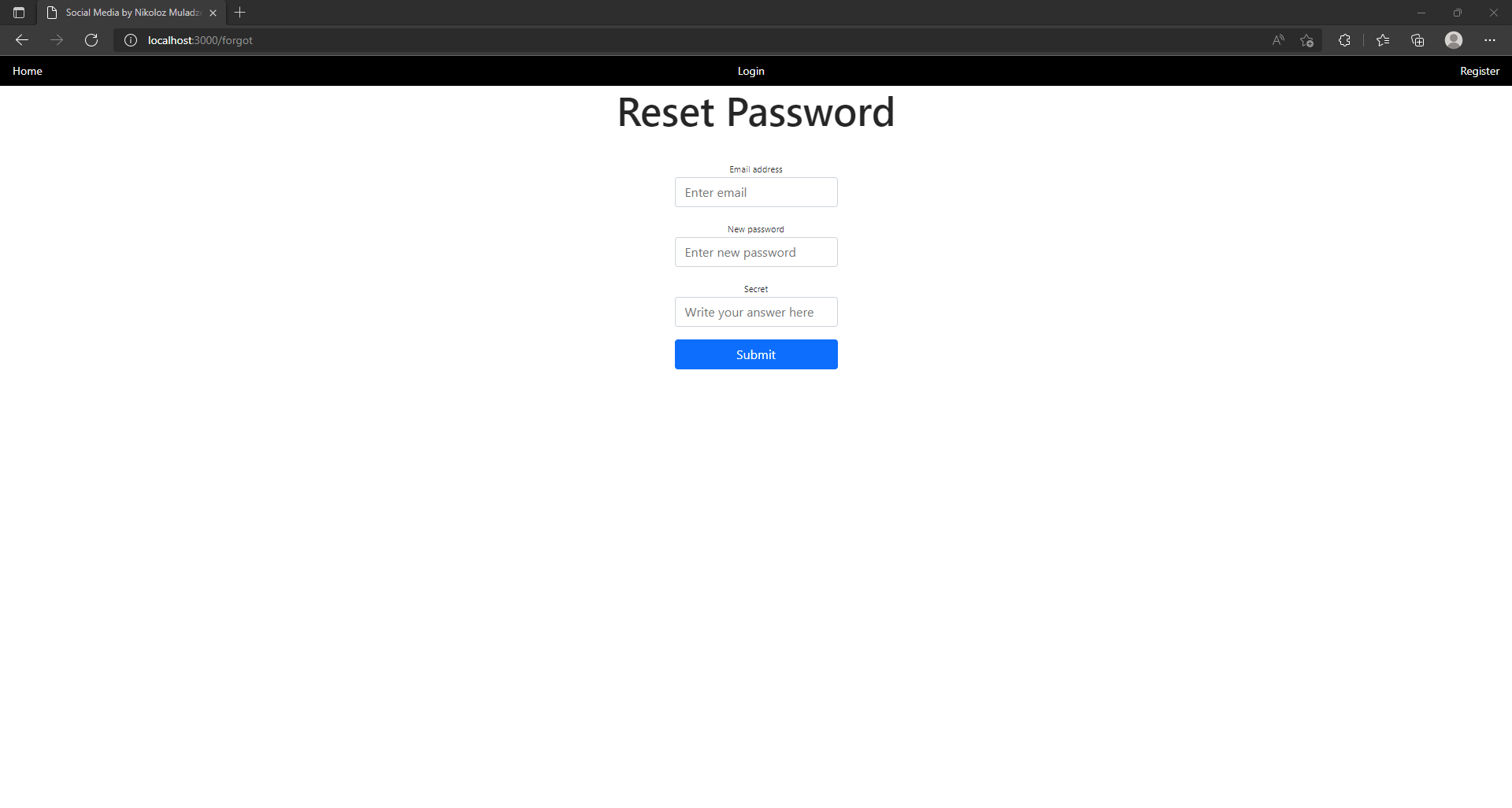


Possible errors:

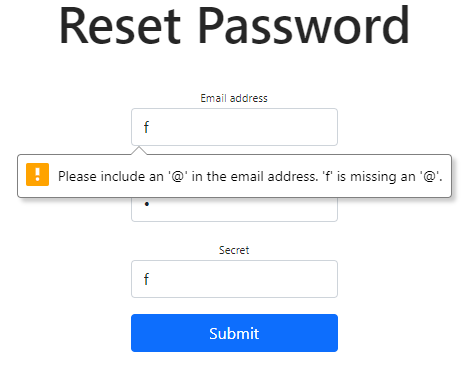
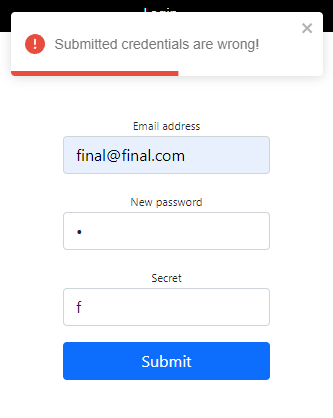


Both fields much be field in and match what you submitted during the registration. But if you forgot the password and still remember the email you can press “Forgot password?” and you will be directed to the page where you will be able to reset the password:

### Reset Password

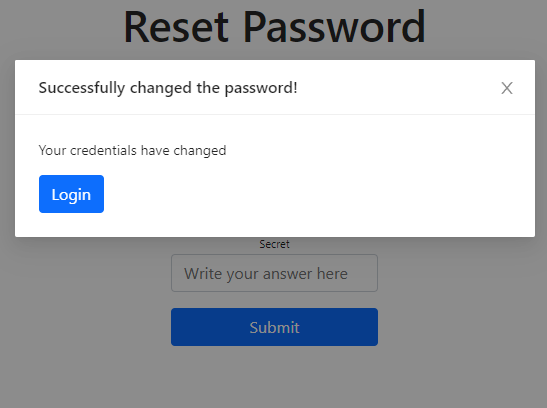


Possible errors:



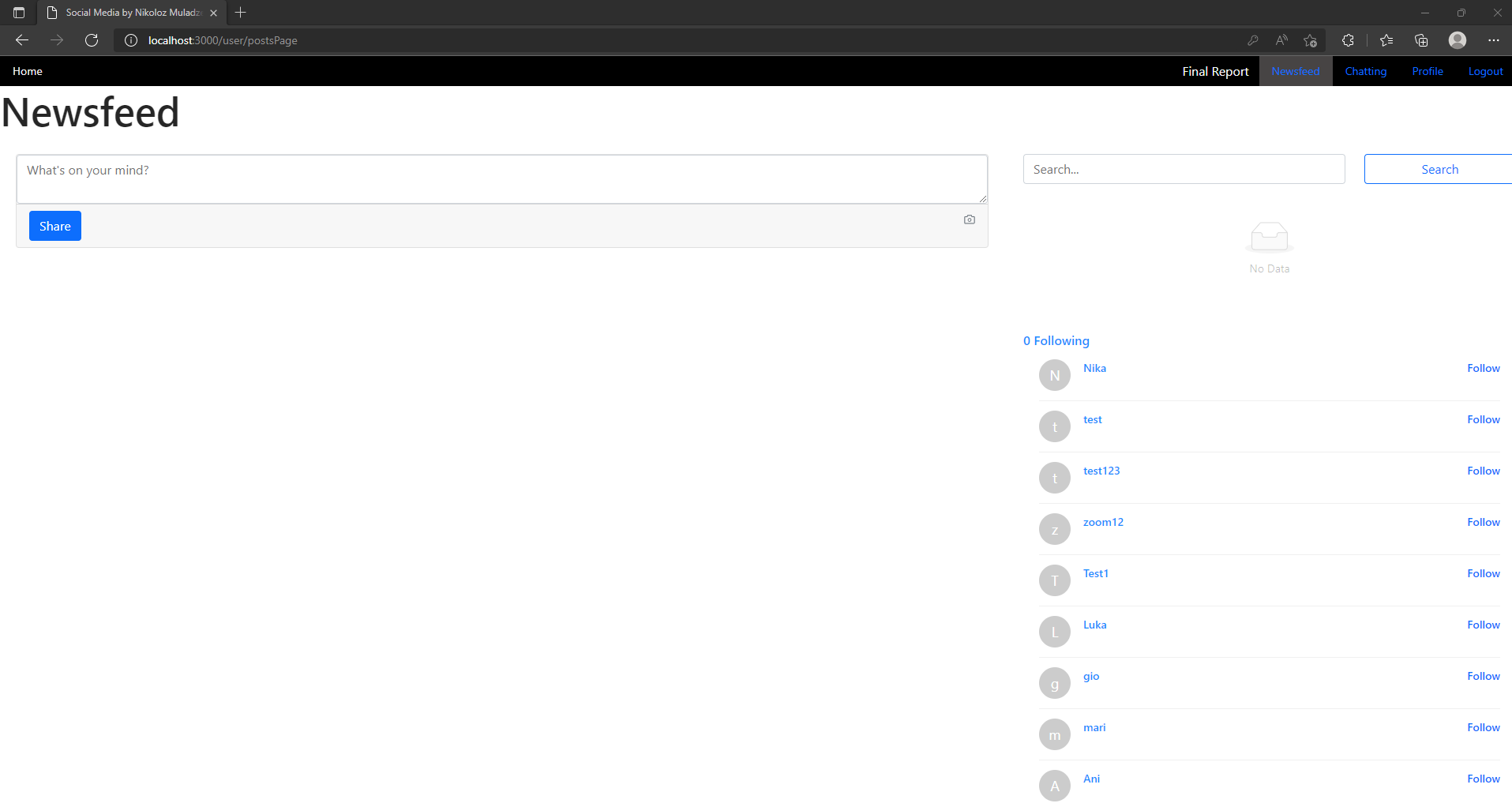
All the fields must be filled in, new password must be 6 characters long or more, email and secret must match data in the database.

Once you successfully change the password you will get the following notification:



Either press Login or close the notification and press Login in the navigation bar. When once again on the login page and you enter correct credentials you will be directed to the newsfeed:

### Newsfeed

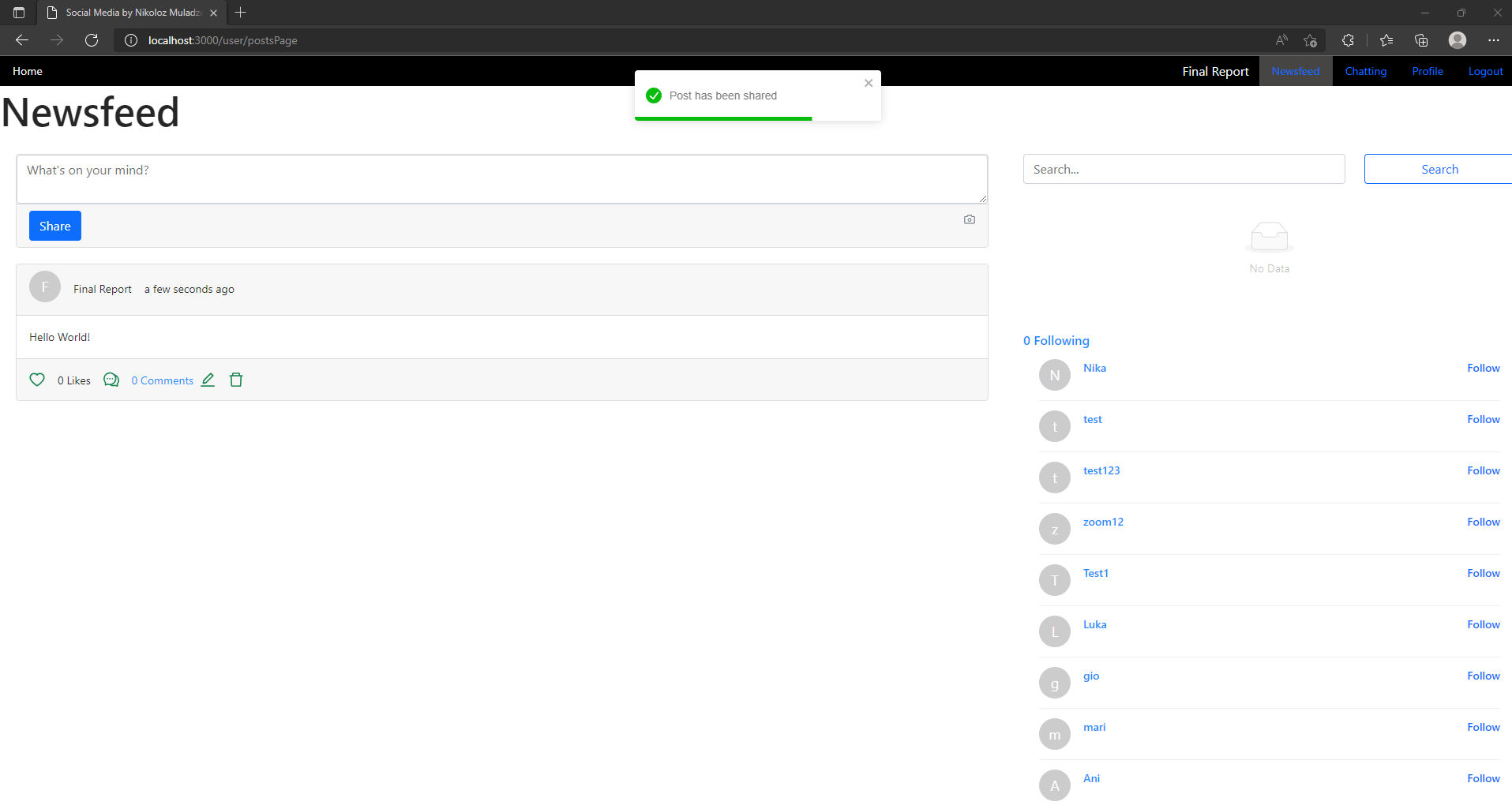


Once on this page you can do a lot of things, and the list on the right will empty for you if you are testing the application for the first time, I registered these users for testing purposes.

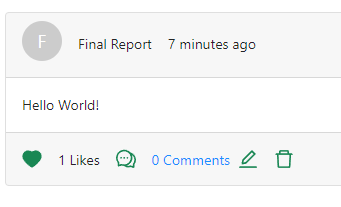
Let us discuss the page we are on called Newsfeed.

You can type in something in the field where it says “What’s on your mind?” and after pressing Share button it will be displayed below it immediately without refreshing the page

### Posting

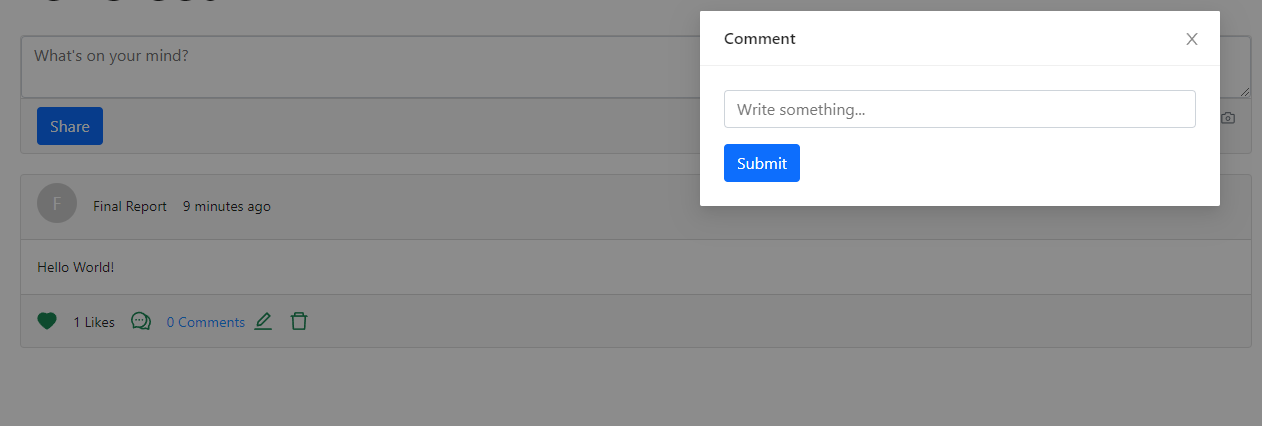


You can see the camera icon to the right of Share button, but it is deactivated, because it is related to Cloudinary, that I decided to exclude from this project.

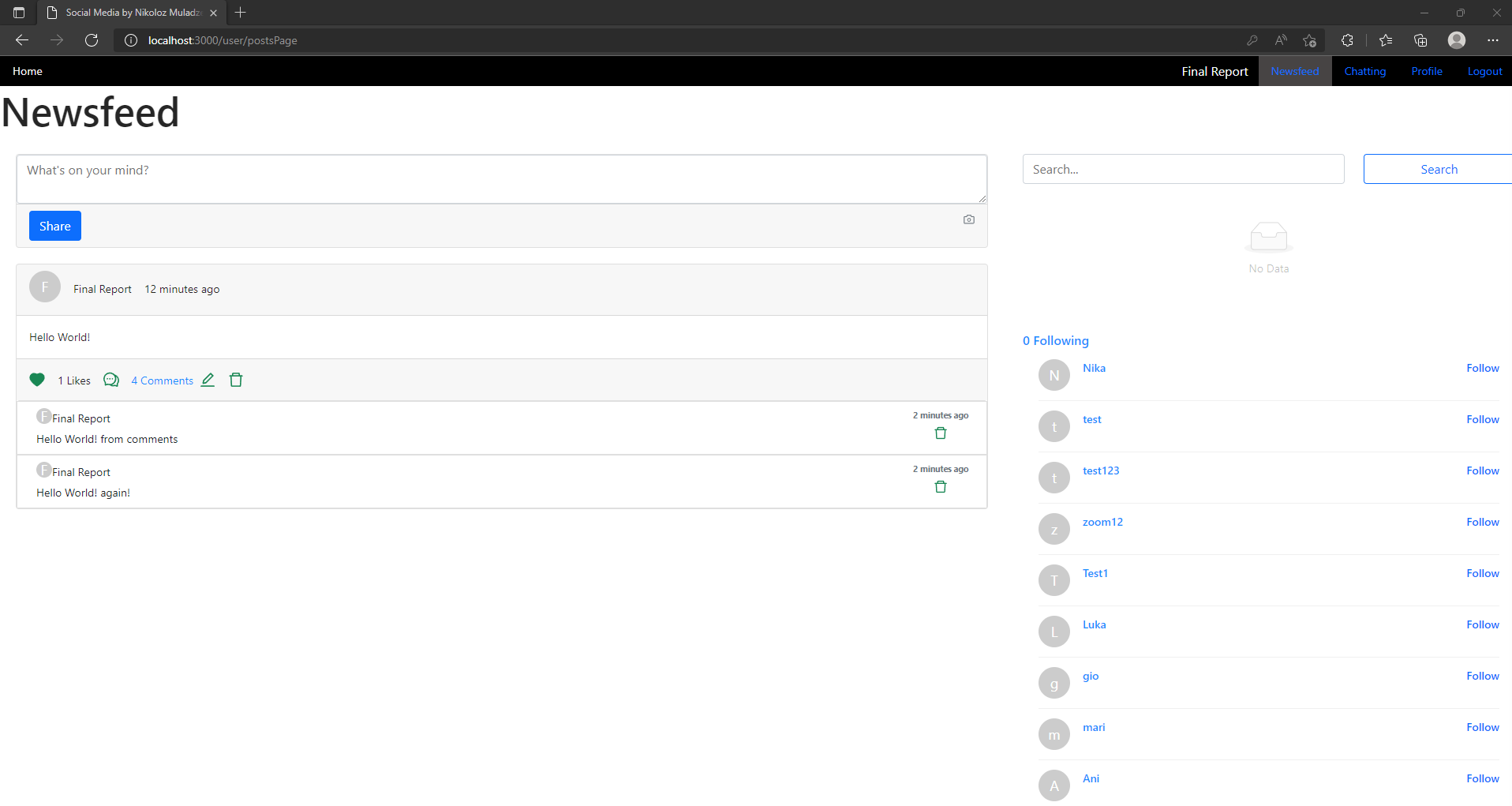
Once you see a post, there are quite a few things to discuss as you see quite a few information on it. First comes avatar/picture of the user who posted it, the character that is displayed on avatar is the first character of the user’s name who posted it. Then comes the Full name of the user and then the timestamp, when he/she posted it. Next line is the contents of the posts, in our case “Hello World!” Then comes the heart icon which you can press to react on the post: 

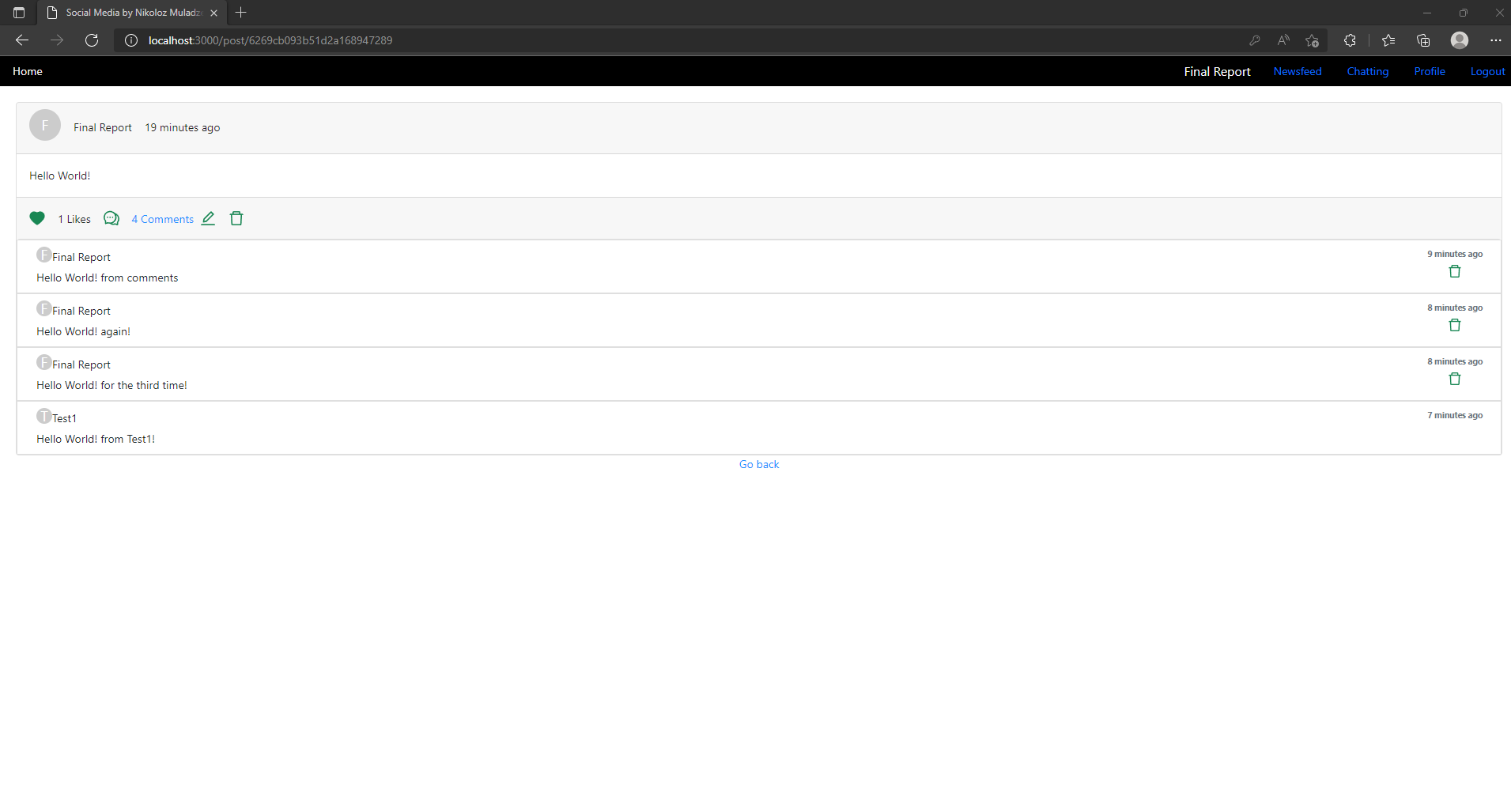
And If you change your mind you can press the icon again to unlike it.

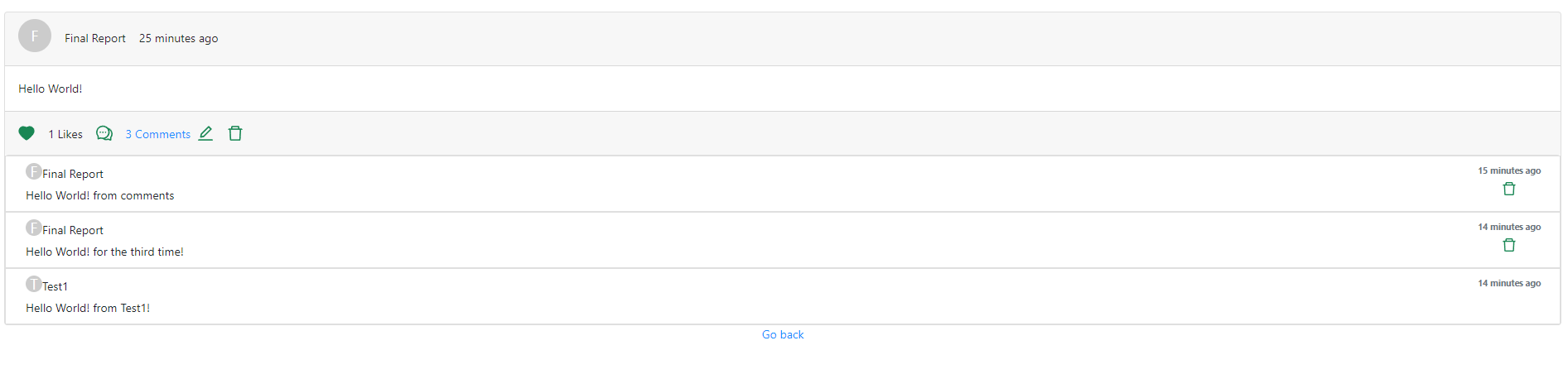
### Commenting

Next up is the comments icon, if you press it, the modal will appear so you can comment on the post: 

Once you fill in the field with the placeholder “Write something…” the comment will appear below the post, without refreshing the page



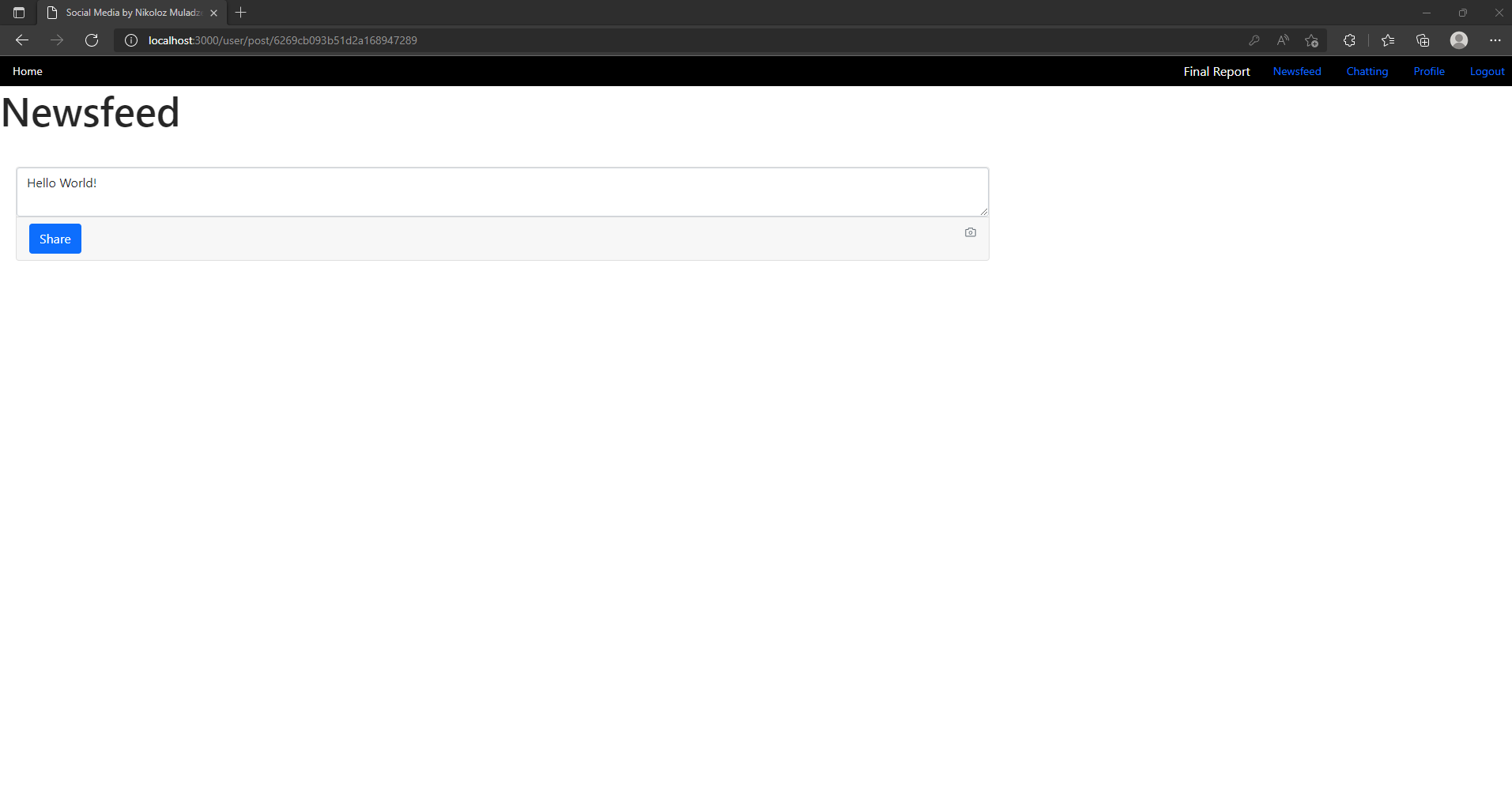
But what’s wrong? Clearly, the post displays 4 comments in blue, but we can only see 2 comments. That is because you can only see only first 2 comments on the newsfeed, once you press “4 comments” it will direct you to the separate page where you will be able to see only that specific post where the limit of comments is 10 million comments, but that’s only the limit of how many comments can be displayed. 

You can see on this page that URL has changed to /post/postID yes, these random characters in the url is the post ID, and now you can see all the comments. You can also see that all the comments have avatars, names, timestamps and some have delete icons. You can only delete your own comments, in this case the last comment is not your comment so you cannot delete it. 

As you can see I deleted one of my comments.

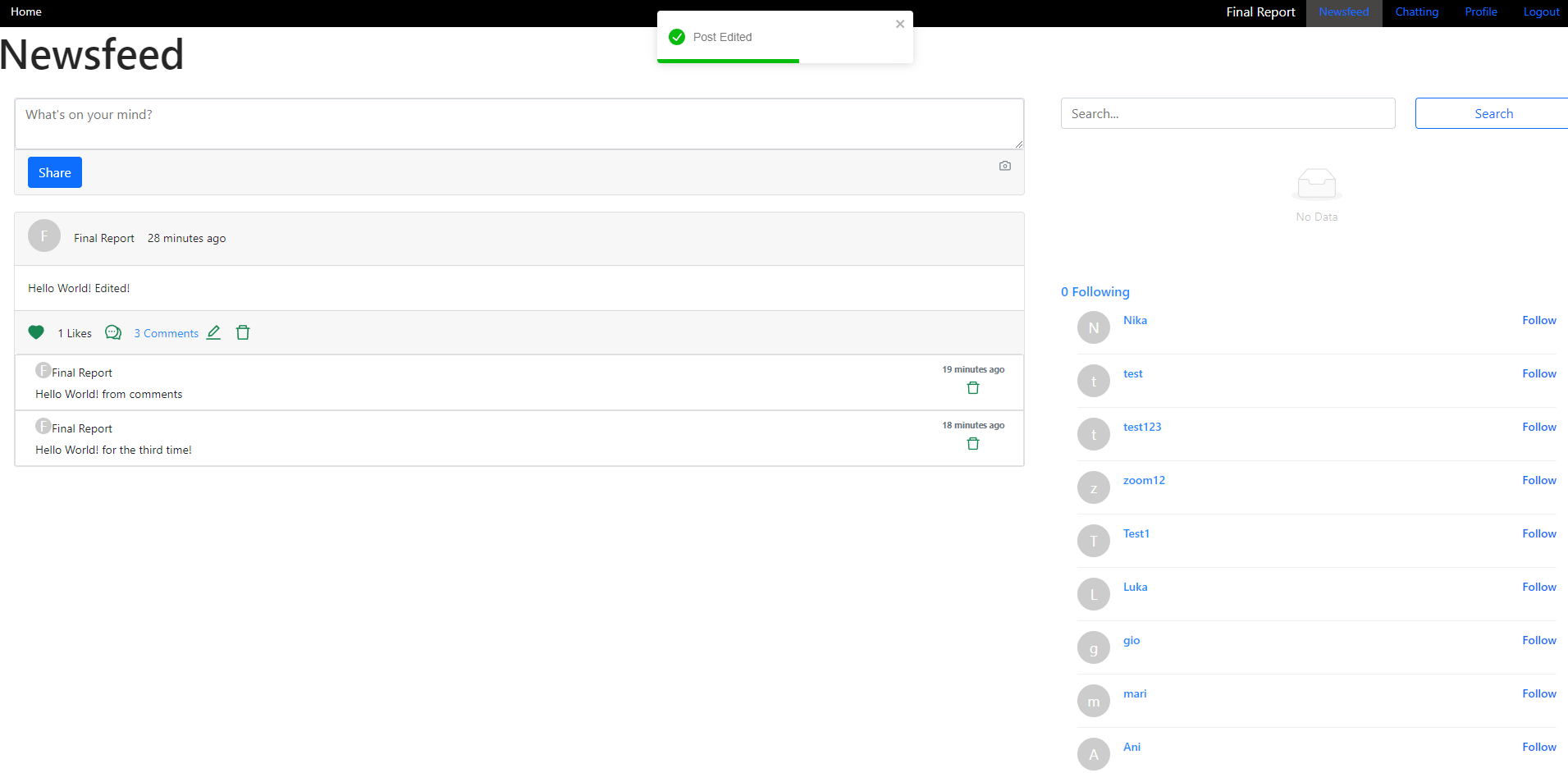
### Editting & Deleting

Now lets say I want to edit, or delete my post, you can see 2 icons after “Comments”. The first one is an edit icon and if you press it you will be directed to other page where you can edit your post

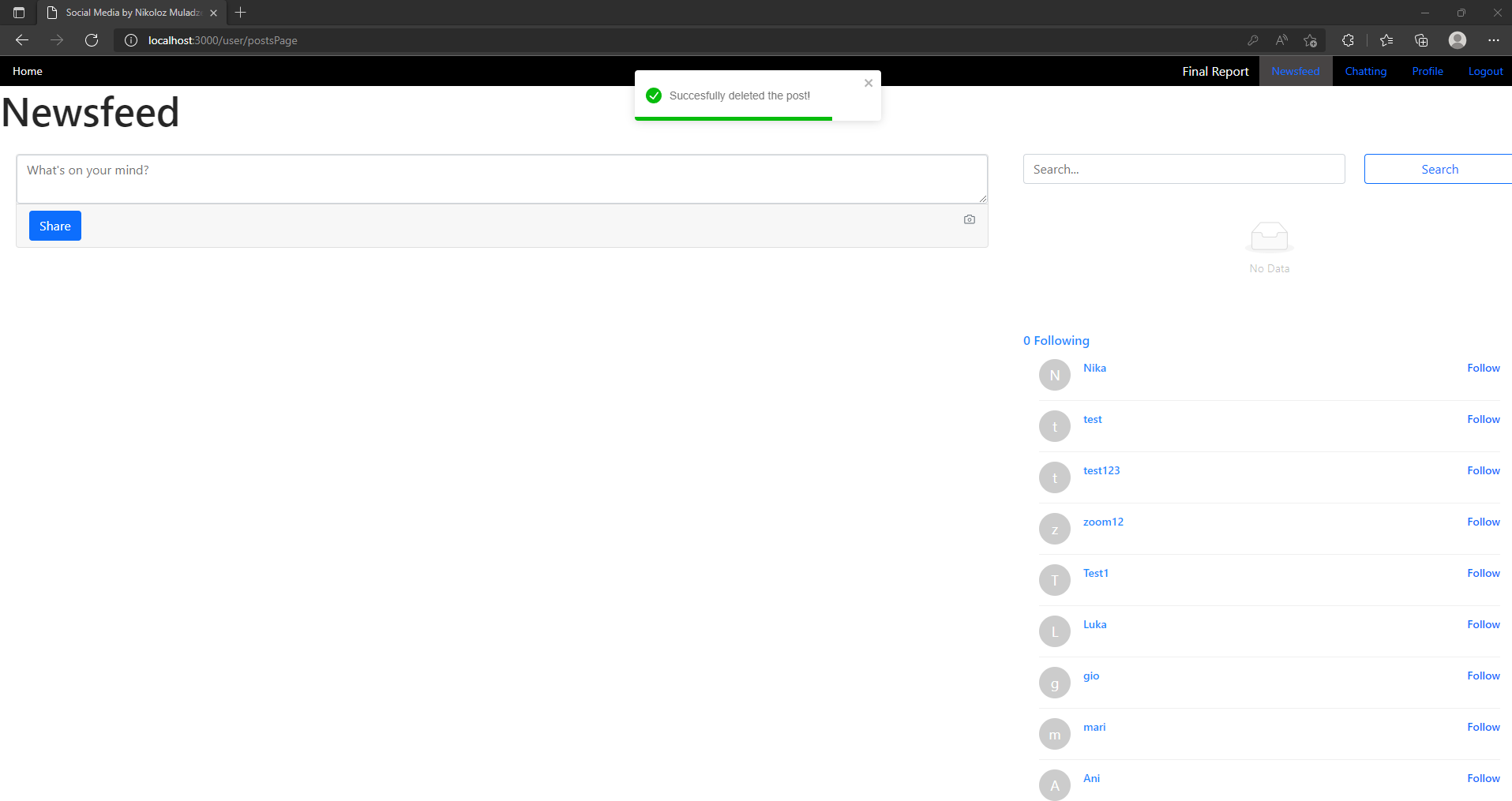


You can also see that URL slightly changed, now it is /user/post/postID

To change the post, just change the field and press Share:



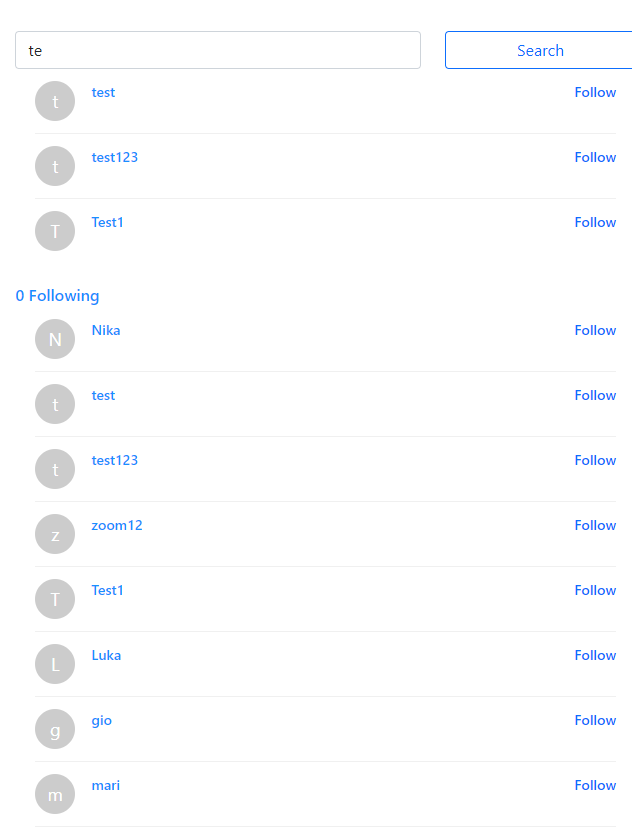
Now if you would like to delete this post, just press the delete icon to the right of the edit icon:



Notice, all of this operations did not refresh the page, everything so far was real-time. Maybe you could argue that, but the URL changed, that means that page refreshed too… wrong! Pay a close attention to the navigation bar, it did not re-render.

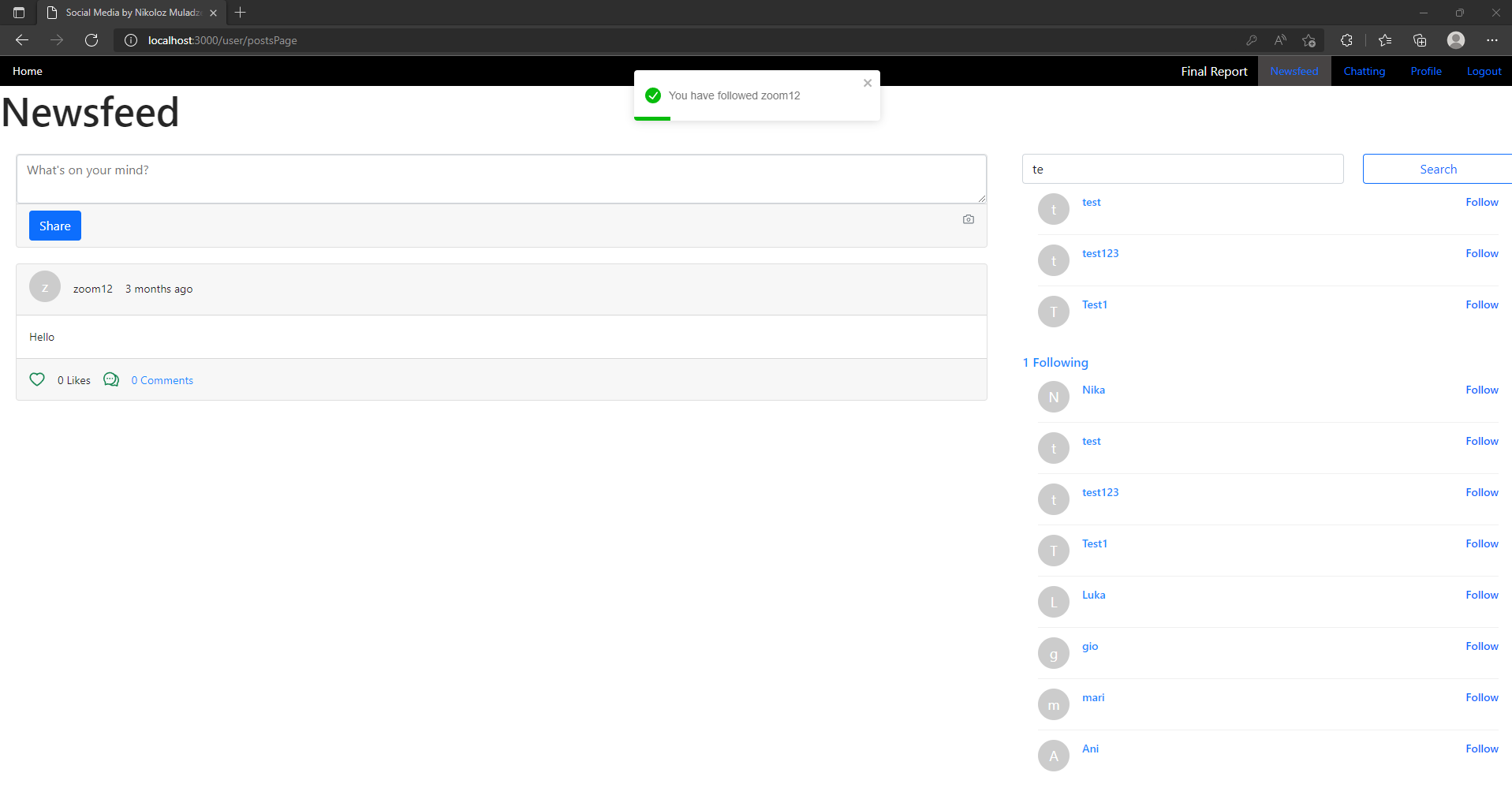
### Search

Now check the right side of the screen, where it says Search. There you can type in any name and it will display users with matching name:

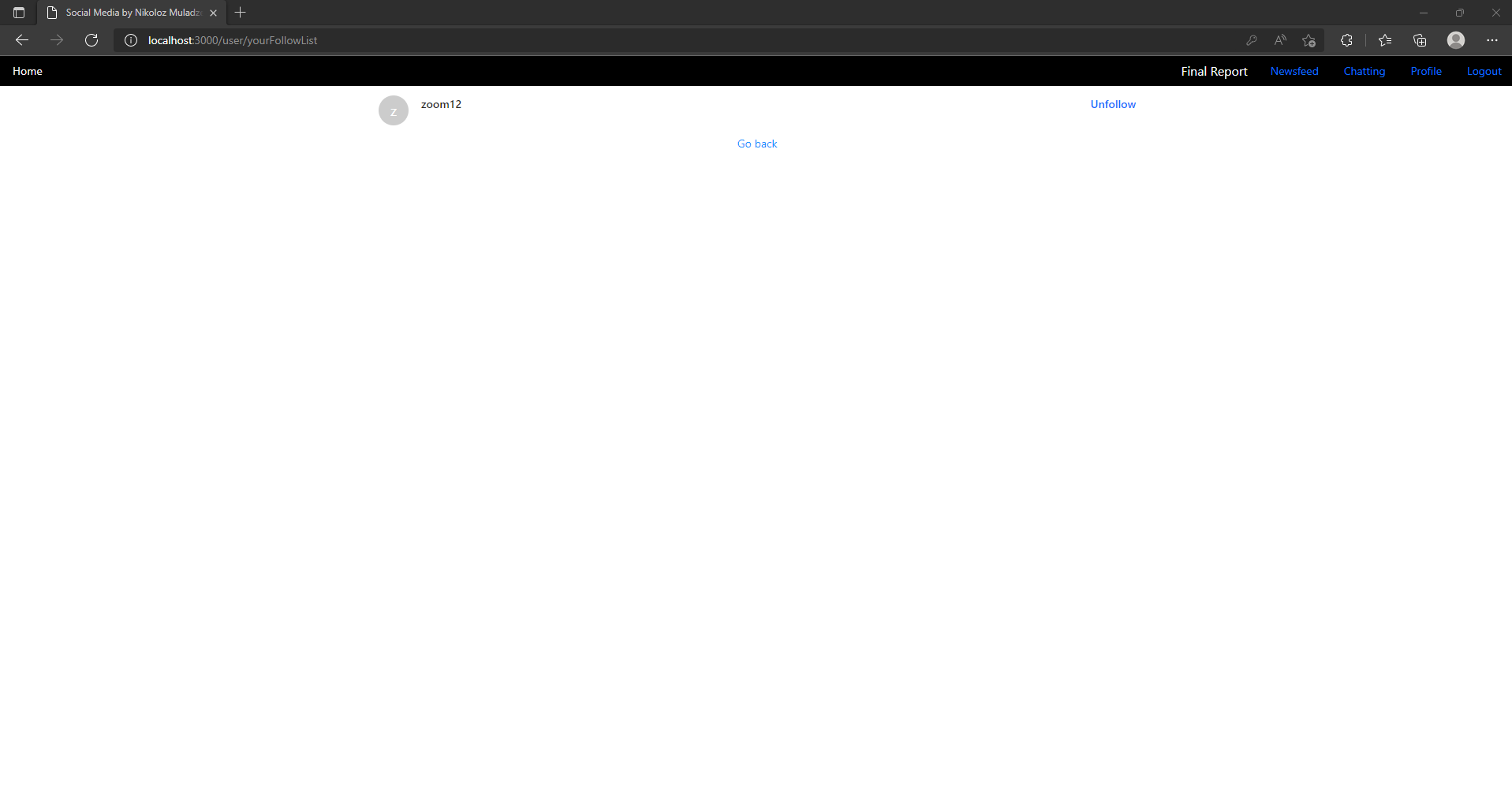


### Follow & Unfollow

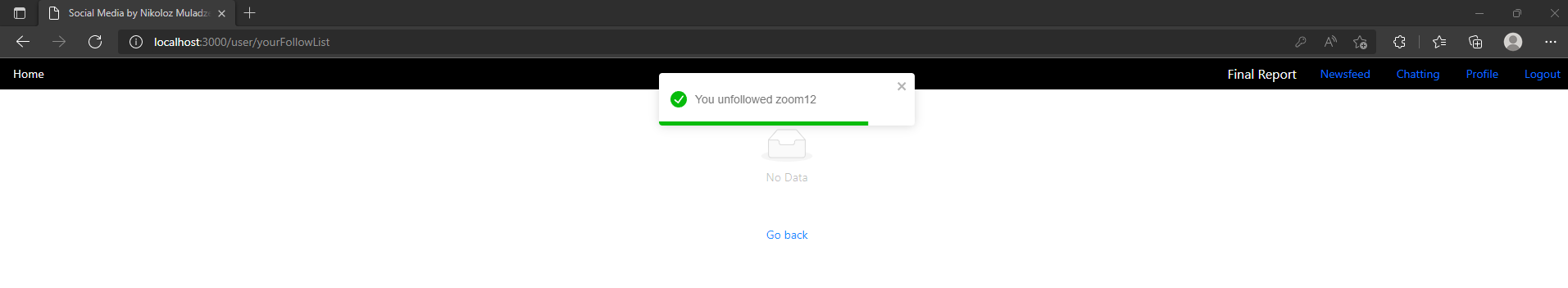
Now that you have found matched users, you can either follow found users, of the ones that are in the list below. But what does following give you? Well you will be able to see what they are posting on their newsfeed!



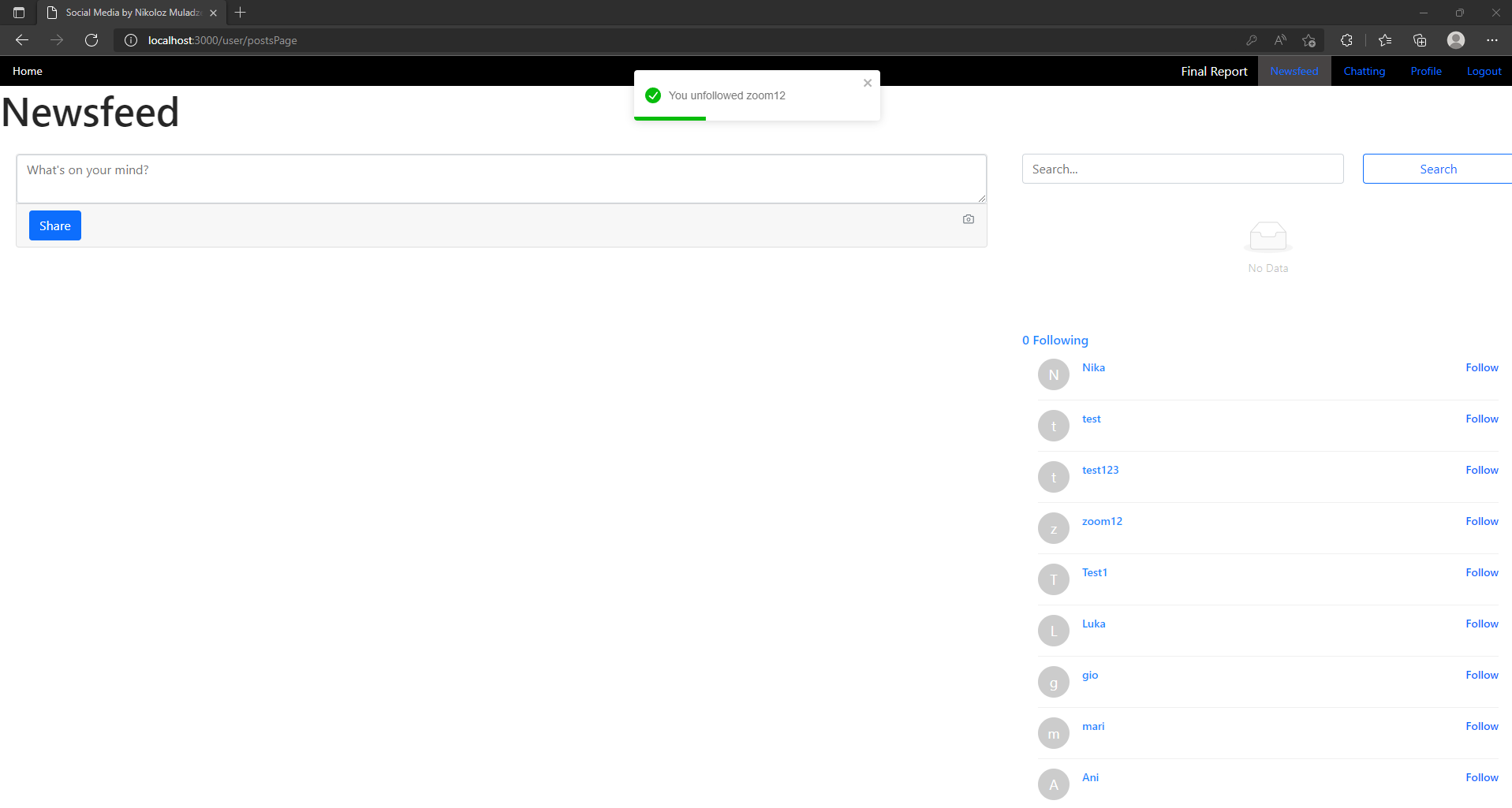
As you can see, I followed zoom12 and can now see the posts zoom12 has posted or will post. I can also react to zoom12’s posts and comment on them (I cannot edit them or delete them!). But what if I do not want to see zoom12’s posts anymore? You can press “1 Following” on the right side of the screen that will direct you to the other page where you can see everyone you are following:



You can also see that URL changed to /user/yourFollowList and if you press unfollow the user will be unfollowed:

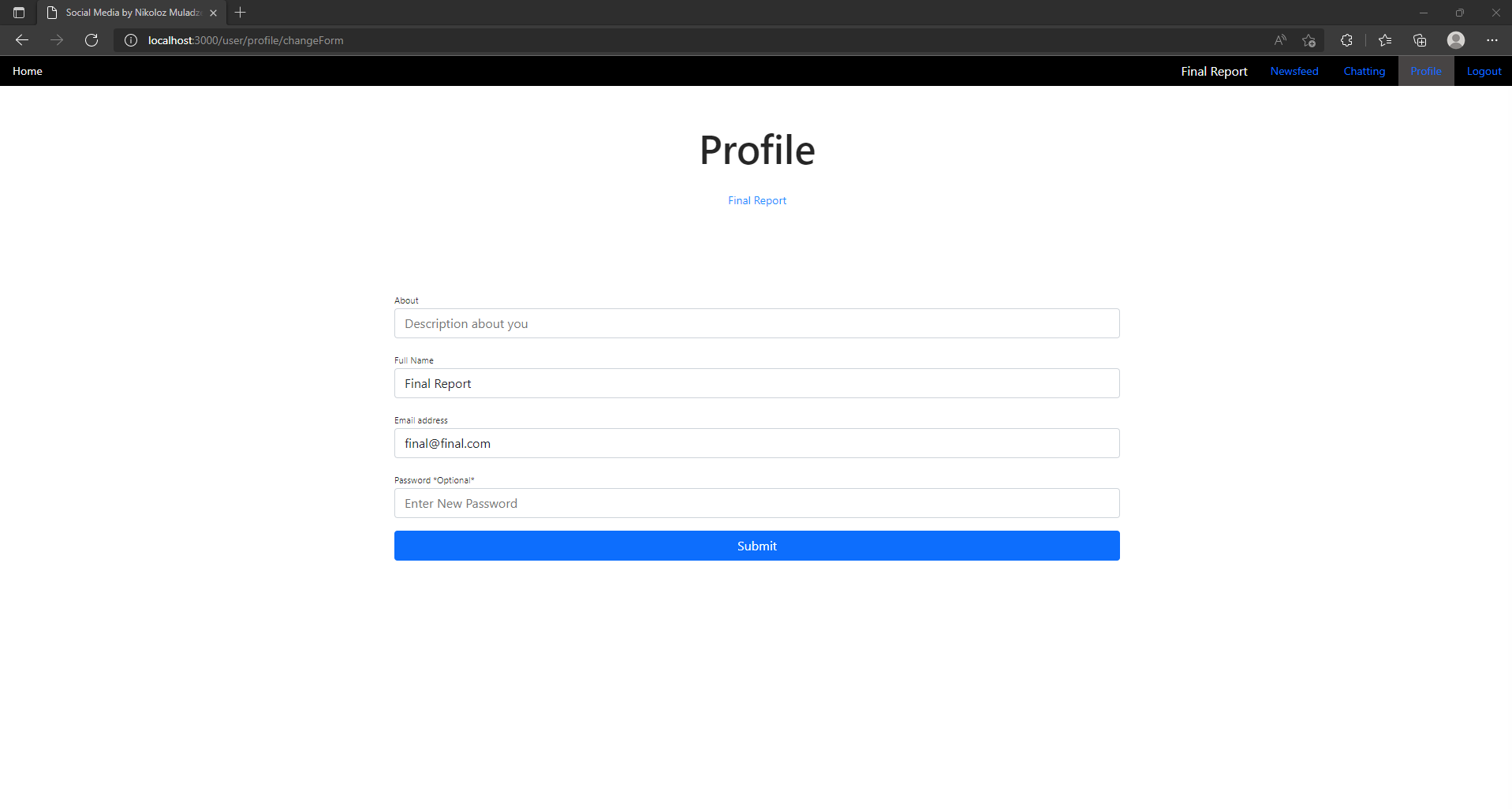


So if you go back and check the newsfeed again, you will not see zoom12 posts anymore:



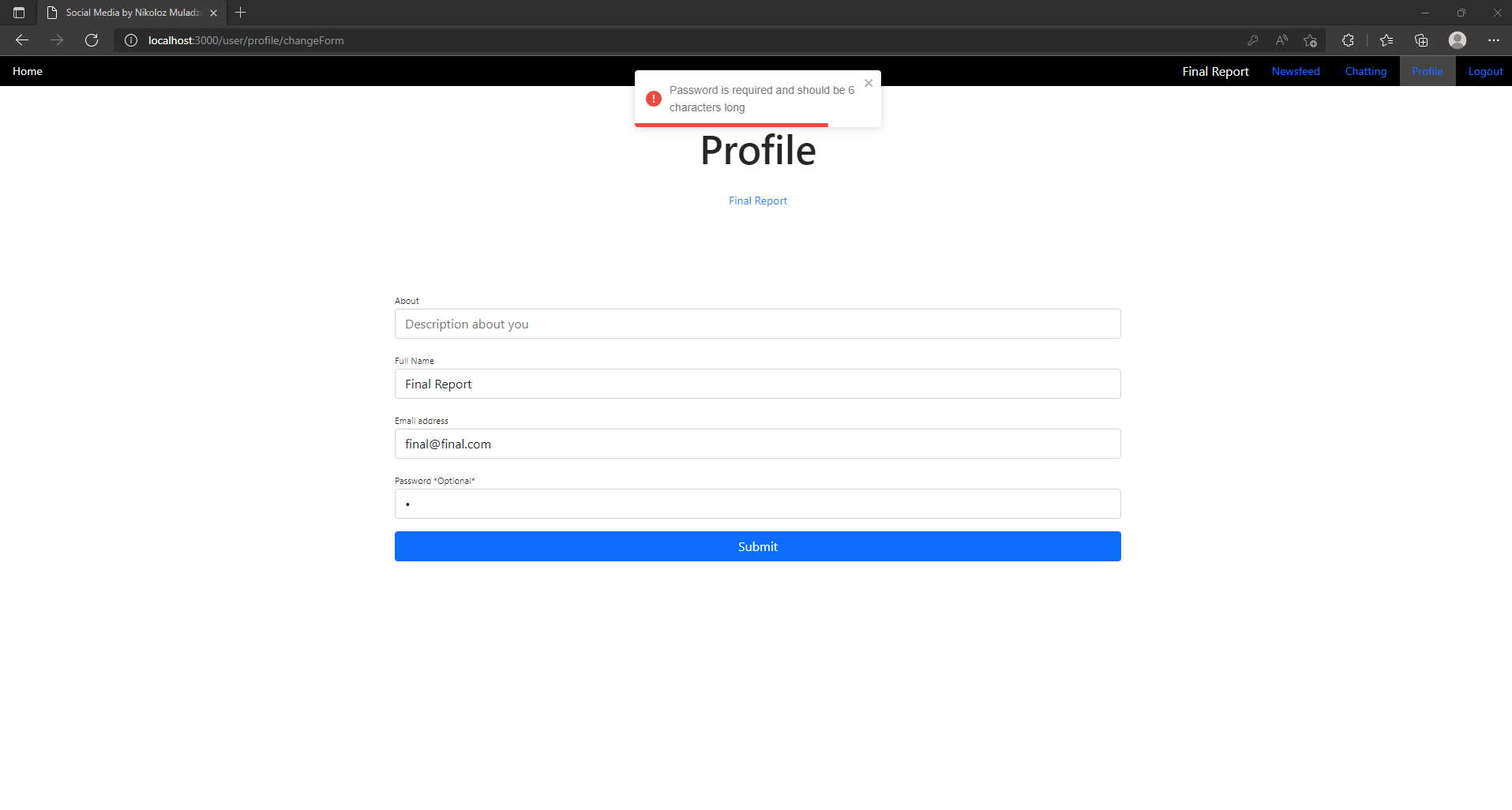
That’s all there is for newsfeed!

### Profile

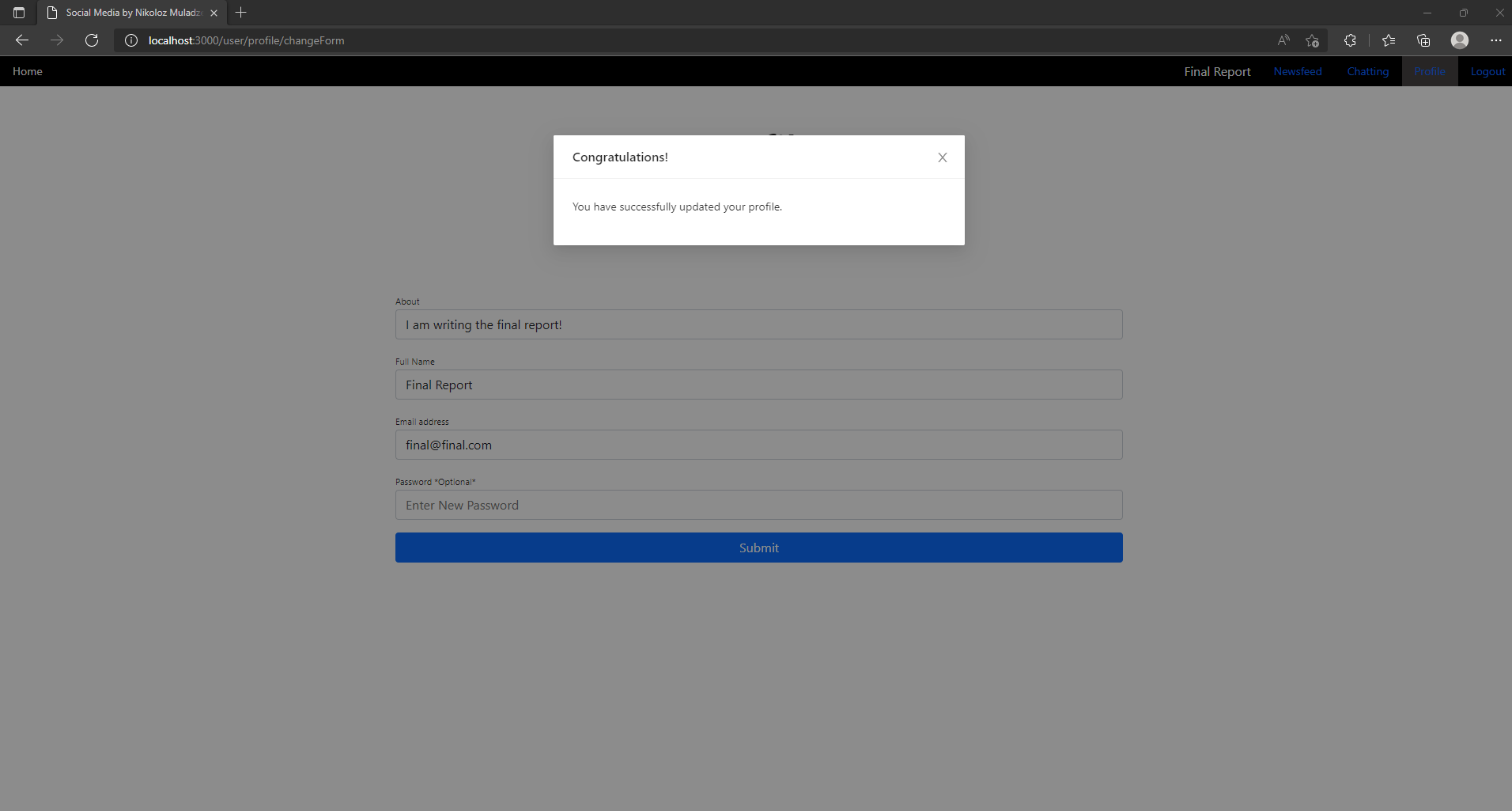
If you want to check your profile, you can press “Profile” on the Navigation bar and you will be directed to the page with the URL /profile/changeForm where you can edit your details: 

Actually you can change only 3 things, “About”, “Full Name” and “Password”. You can see the email, but that is there just to remind you, you cannot type in email field!

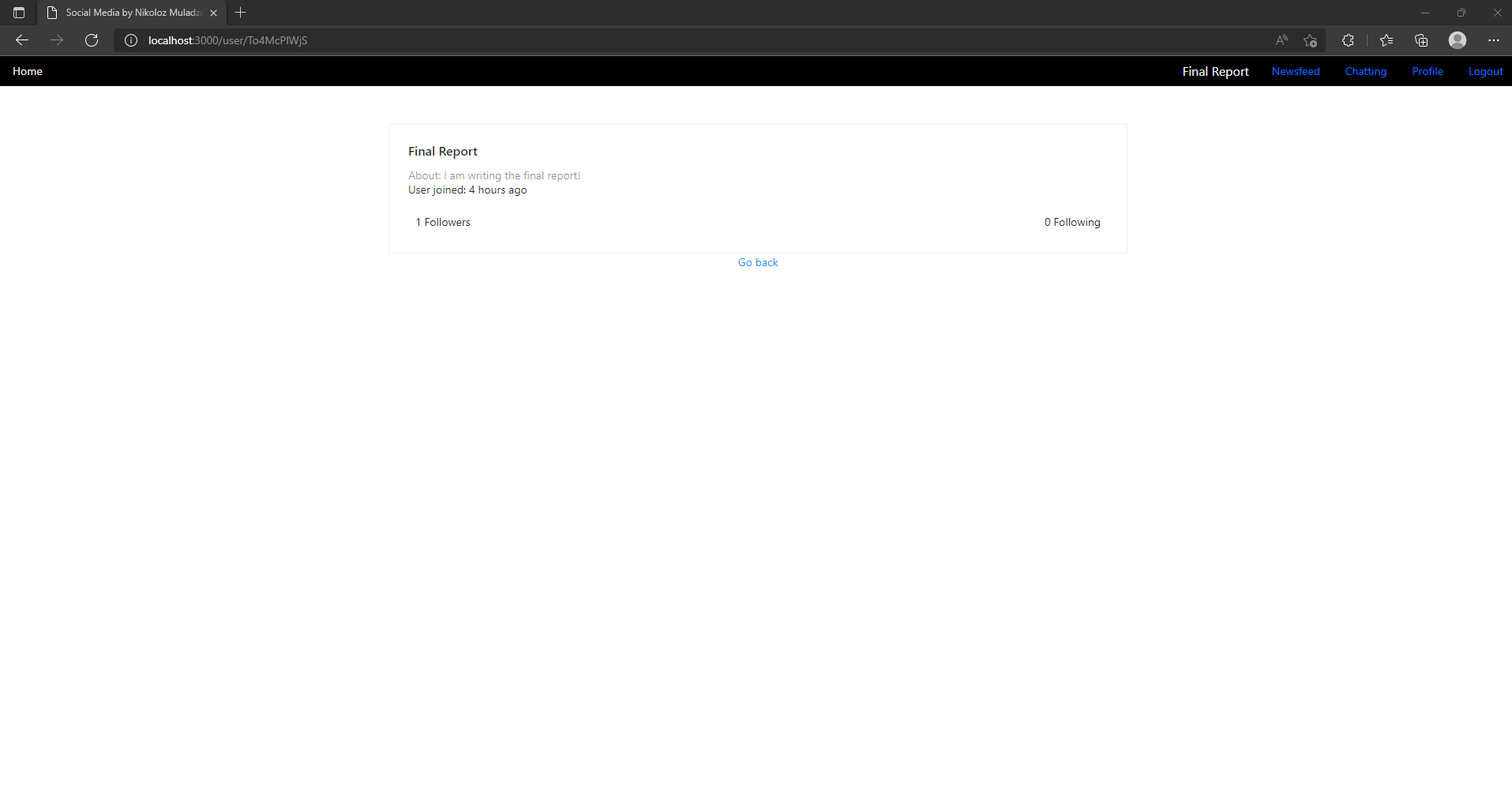
Changing the password is optional, but keep in mind, error still applies:



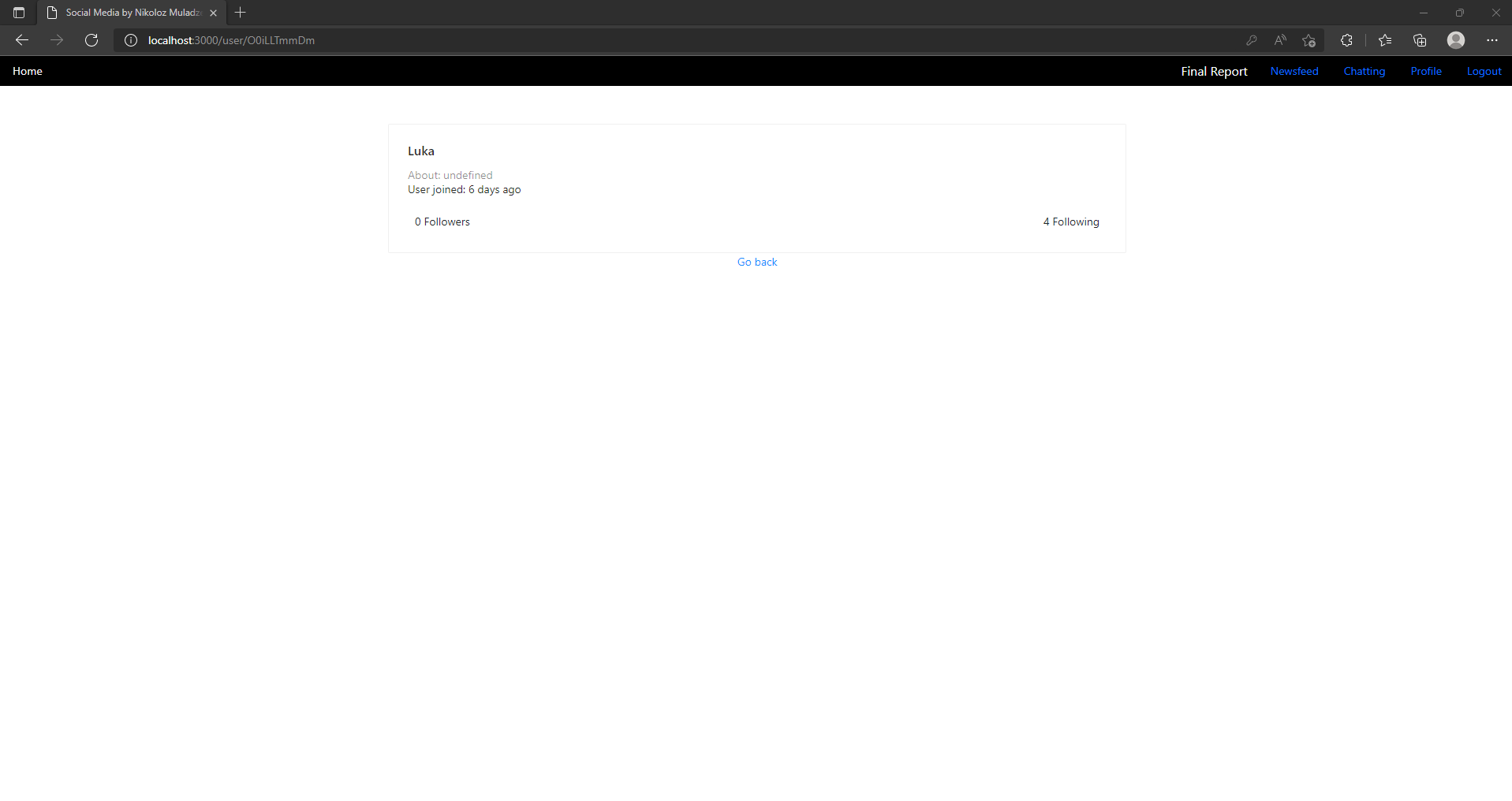
If you decide to change your details you will see the following window:



Now you can go and check your actual profile by pressing your name in the middle of the screen that will direct you to the page /user/userID

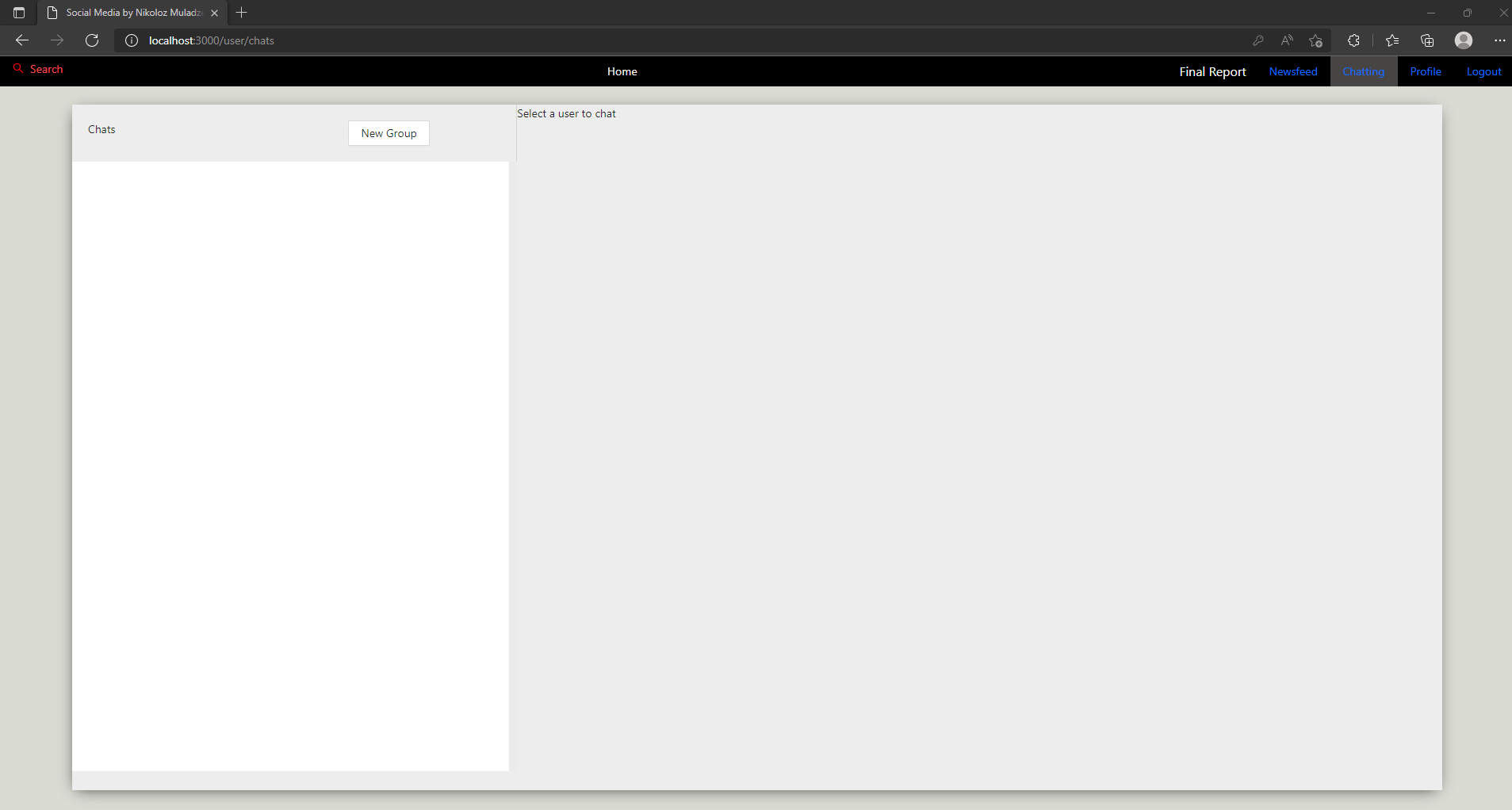


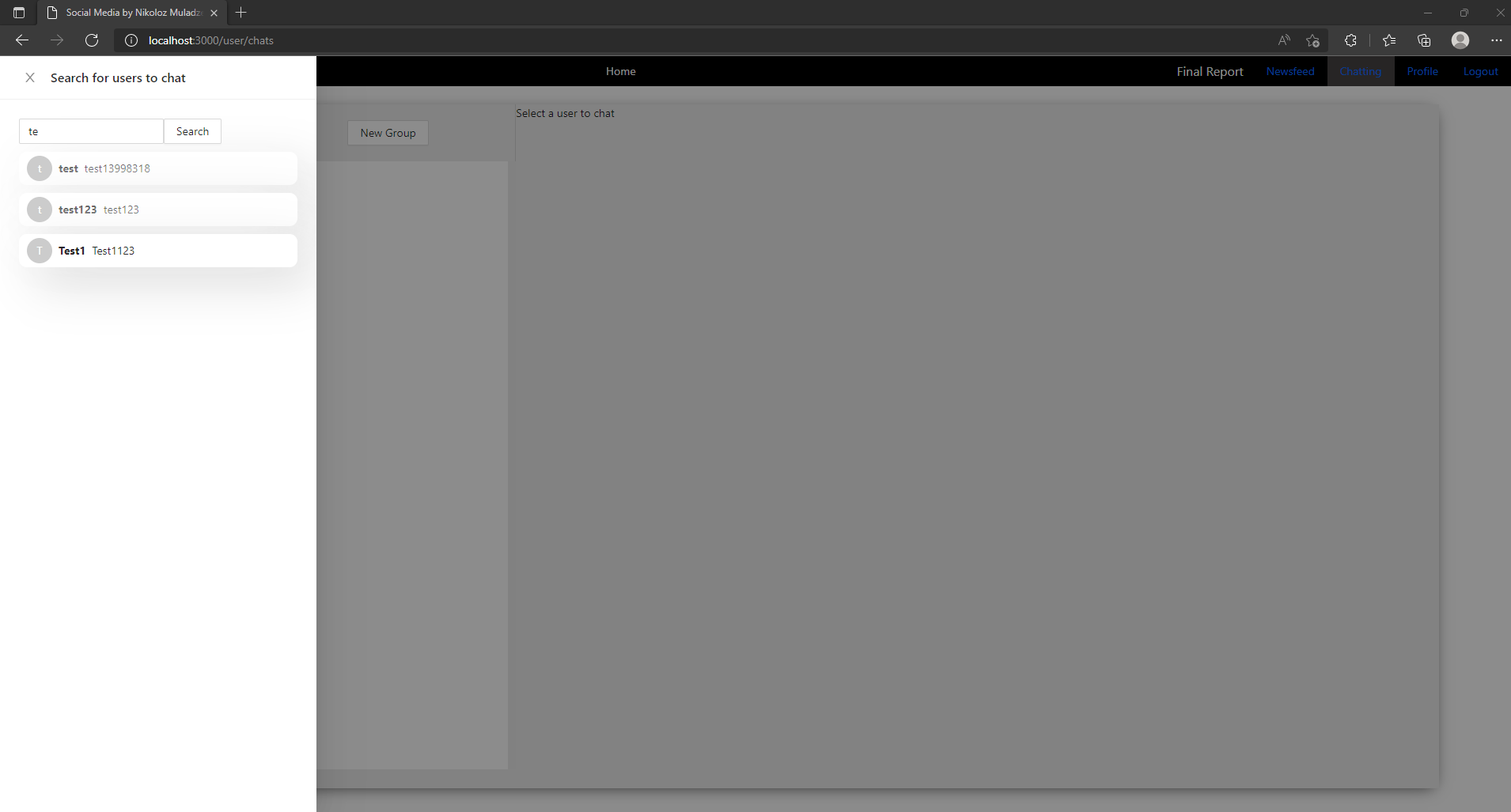
You can see your name, about, when you registered, how many people are following you and how many people you are following, but you cannot see who your followers are. You can also go to the Newsfeed and remember on the right side there where names of other users? You can actually press on their names, which will direct you to their profile pages respectively:



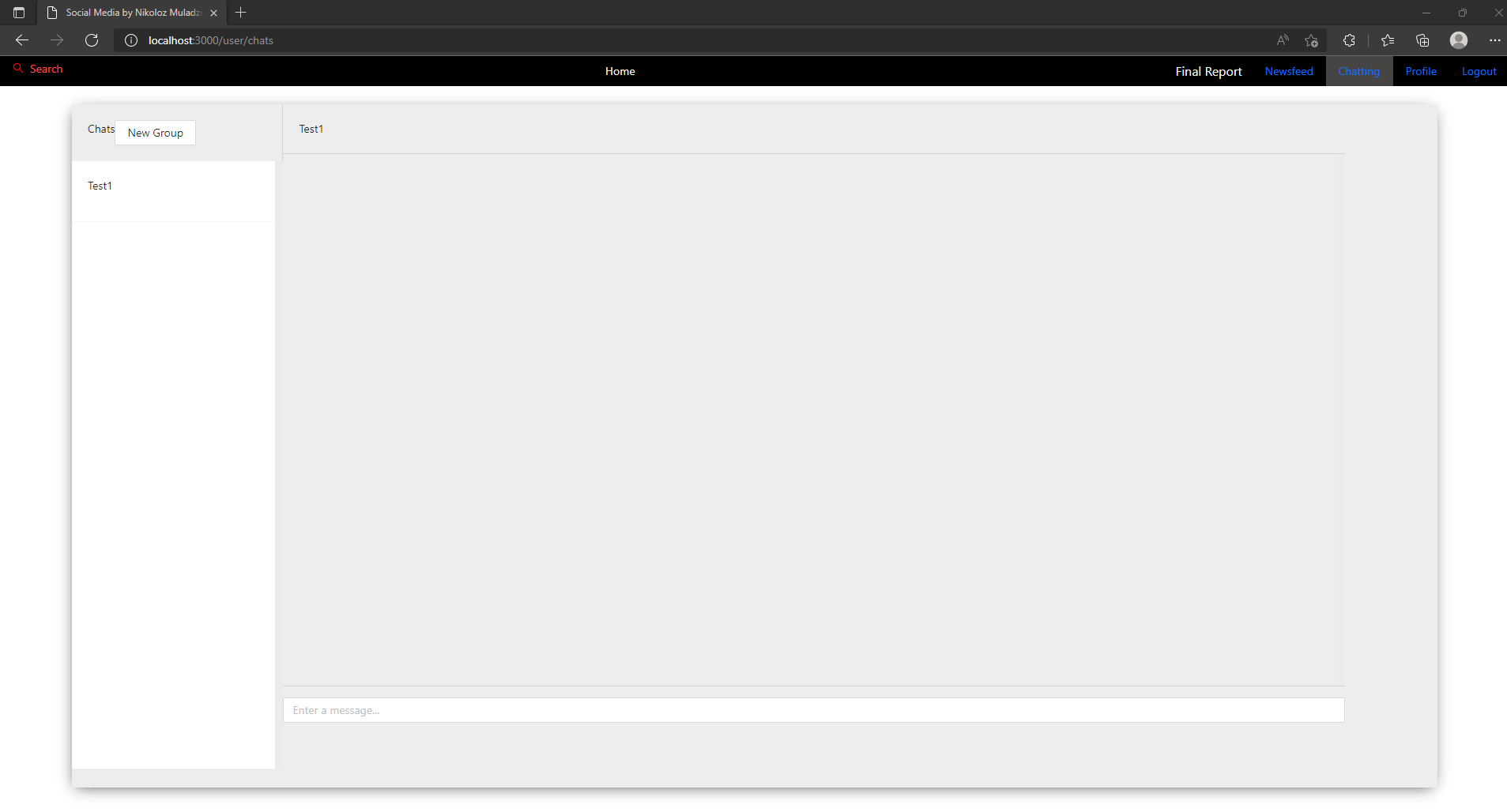
### Chatting

To access chats press Chatting in the Navigation Bar and you will be directed to the page with URL /user/chats



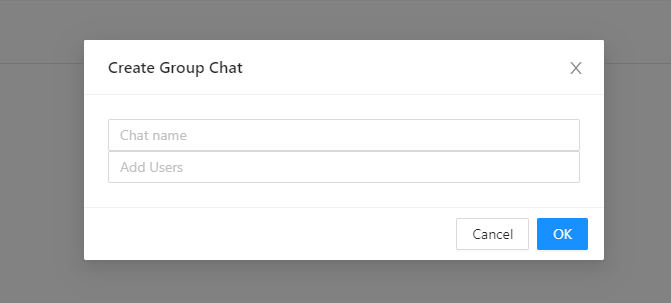
You will see straight away that a Search in red appeared on the Navigation bar, which indeed only appears on chatting page. Once you press it a Drawer will open up from the left side where you are able to search for users: 

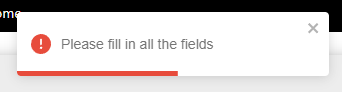
Pressing one of these users will open up a private chat:



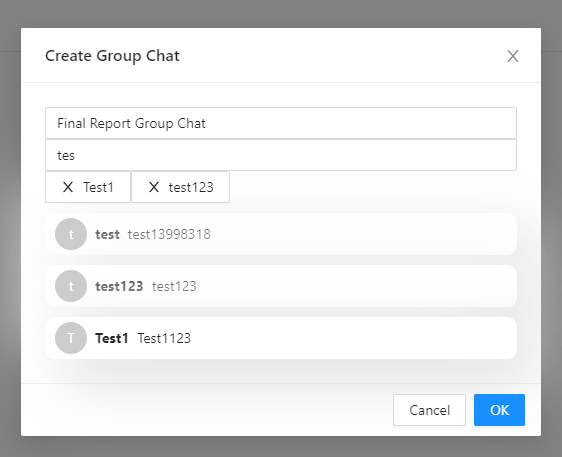
Now you can see this user’s name appeared on the left side of the screen and opened up a chat on the right.

If you do not want to chat privately, then maybe you want to create a group chat? You can do that by pressing New Group button on the top-left side of the screen, this action will open up a modal:



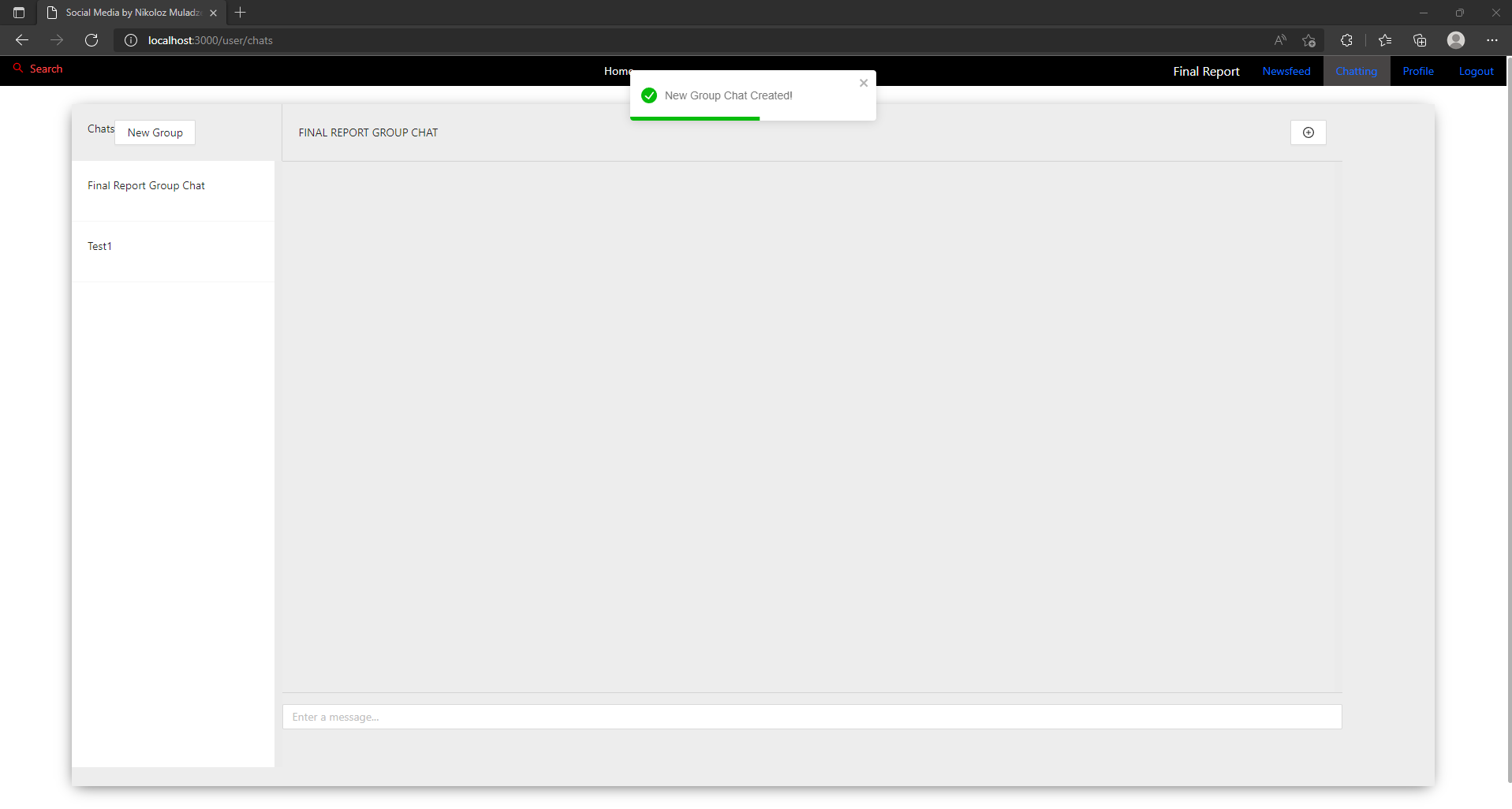
Here you are able to call the group chat anything you like and add as many users as you like, but if you press ok with empty fields you will get an error: 

So let us create a chat now and add some users:



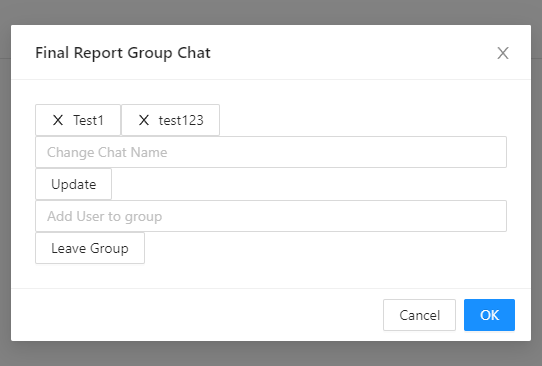
If you type something in the field where it says “Add Users” the list of the users will appear that match the query from the field. And if you click on these users, they will be appended above as Tags, which can be removed by clicking.

Now that we have decided on who will be a part of our group chat we can press “Ok” and that will create a chat:

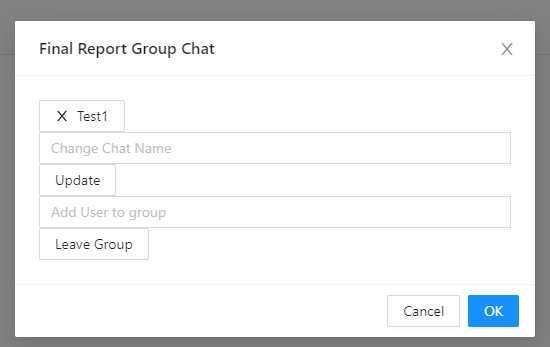


There is a slight issue with this step as after creating a group chat a spinner will appear, but pressing a chat on the left will solve this issue in no time.

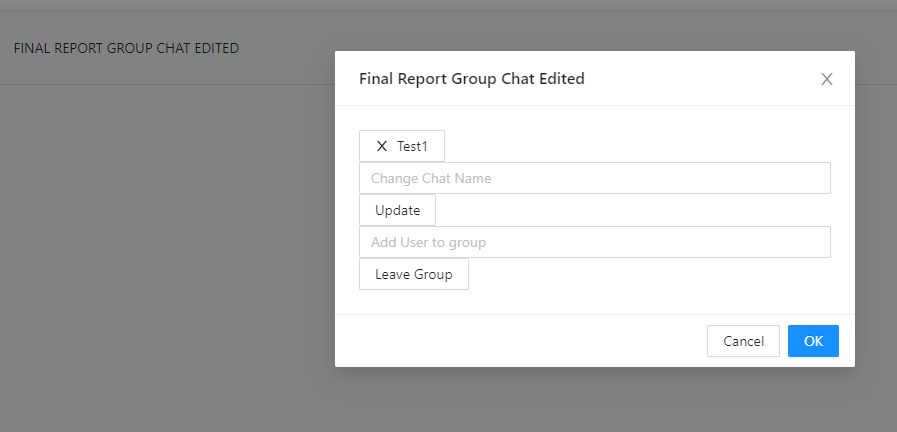
Now you can see two different things, a chat name and a button with the plus icon on the top-right side of the screen. If you press that button, a modal will open up where you will be able to change group chat’s name, its members or leave it (NOTE: whoever created a chat is so called Group admin and has a privilege to modify it):



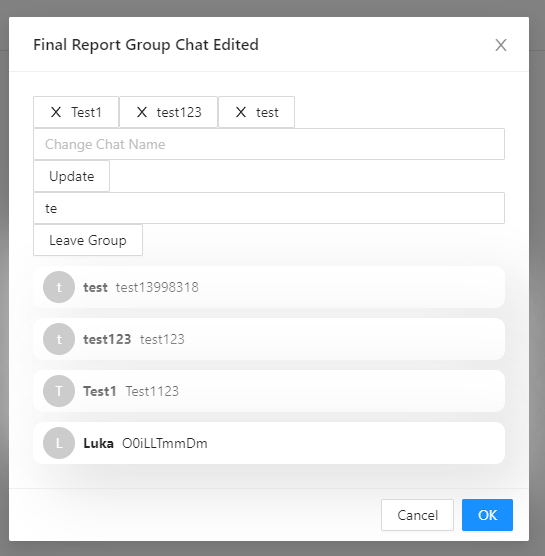
It is straight forward that you can remove users from the group chat by clicking on tags, if they disappear it means they are no longer in it.



When you decide to change the name, just type the new chat name in the first field and press Update button:

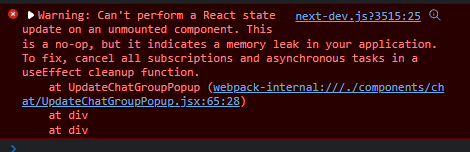


Then if you decide that you would like to add new users to the group chat just type their name in the second field and choose from the list:



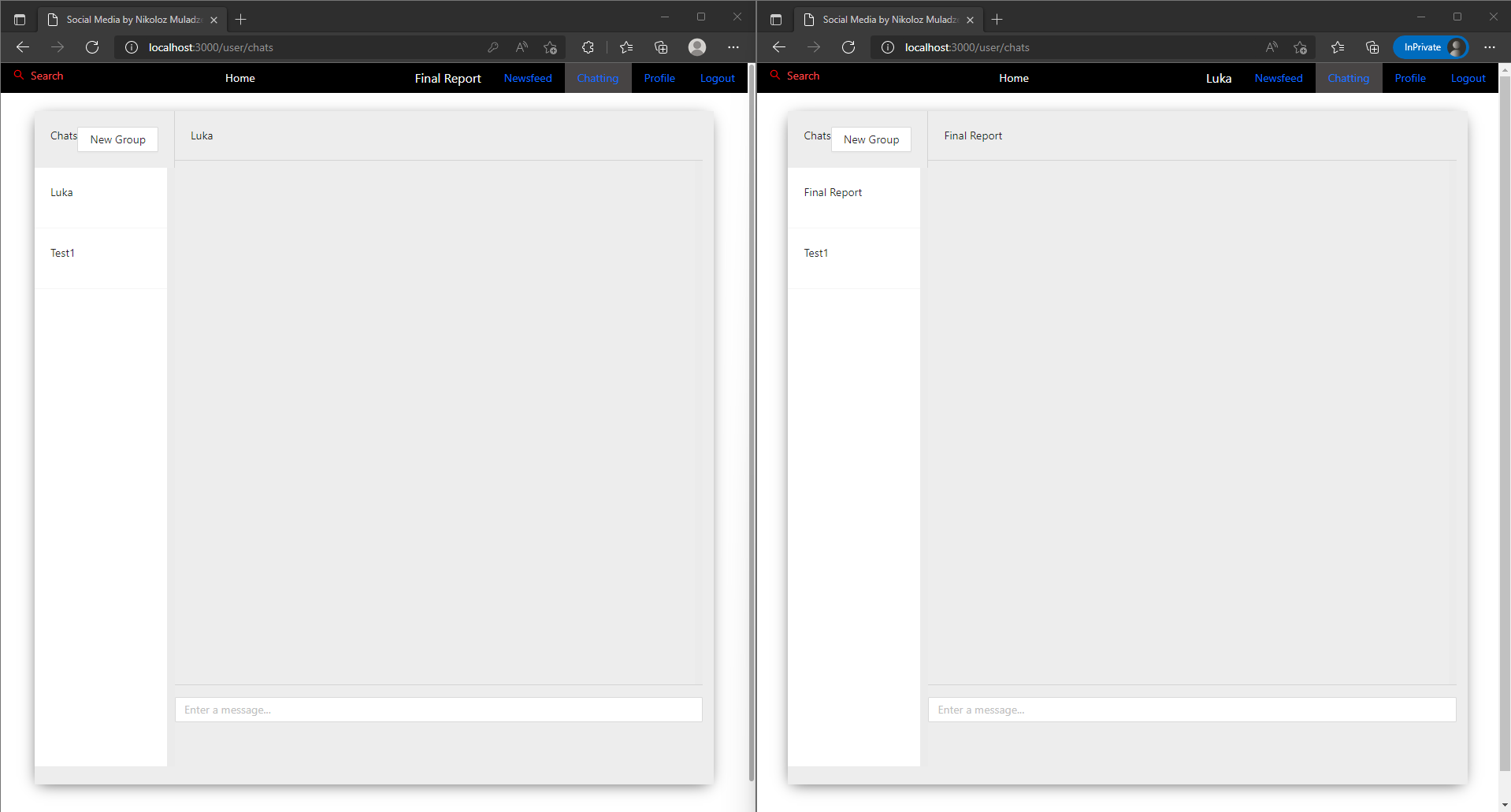
Once you press a user from the list, they are appended above as Tags with their respective names, which means they have been added to the group.

You can also press Leave Group button and you will leave the group, but the following error will appear in the console, but you still leave the group chat without any apparent issues apart from this error, that is only visible when viewing console, without viewing console you would not even know that an error occurred at all:

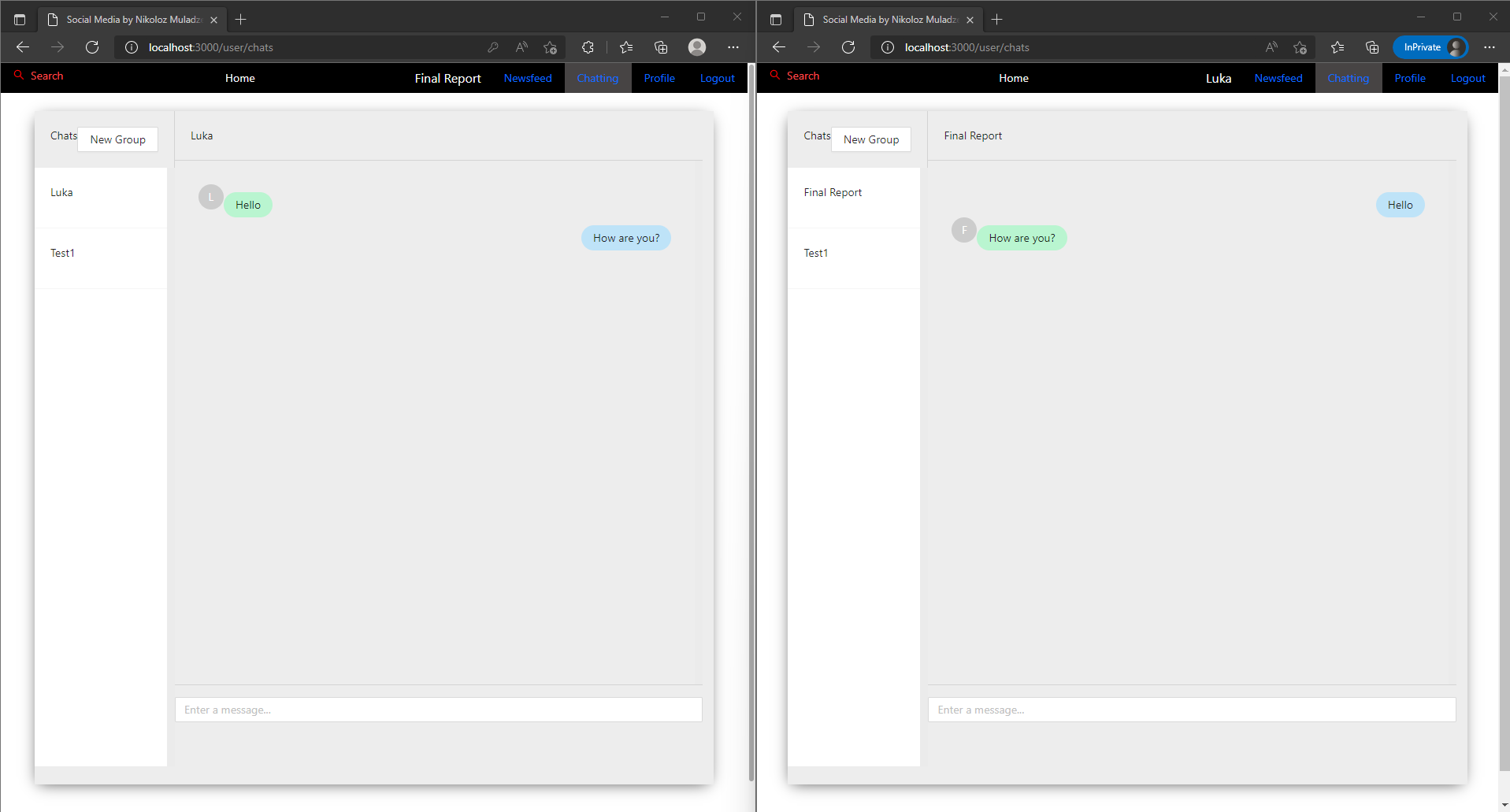


### Sending messages

Now let us review sending messages:



I opened up two separate clients and if I start sending messages from one, the other will see the result without having to refresh the page, this is where socket.io came into use and gave an ability for such real-time experience in chatting



Now, there is actually a bug with this feature, as when a message is received during chatting, all the previous messages are removed from display and only the new ones get displayed.

## Code

### server folder

#### .env

Env file / dotenv is a text configuration file that gives you an ability to control your application’s constants, which should not be viewed by public. So my server has 4 such constants: PORT, DATABASE, JWT\_SECRET and CLIENT\_URL.

#### package.json & package-lock.json

Package.json file is used for version control. This file is very important, first it indicates the name of the package, then its version, then description (these three are custom and do not affect the application in any way) next is “main  
:”server.js” now this line is very important as it means that the server.js file is a root file which will be executed first. If we check npm documentation, it says the following: “The main field is a module ID that is the primary entry point to your program”: <https://docs.npmjs.com/cli/v7/configuring-npm/package-json#main>

package-lock.json is “automatically generated for any operations where npm modifies either the node\_modules tree, or package. json . It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.”: <https://docs.npmjs.com/cli/v8/configuring-npm/package-lock-json>

Next we will see scripts field, this is also very important as we can write a whole command in the string for its sub-fields and just run:

npm {sub-field}

In our case:

npm start

Which is identical to running the following line:

nodemon –r esm server.js

nodemon is the external package I am using which allows to run the server.js without having to re-type the same line in case we make changes to server.js and any file that is related to it. “–r esm” is another external package I am using to enable ECMAScript in the server. Next up keywords, author and license fields do not affect the application. Now dependencies field is the most important one, it contains all the external packages with their respective versions needs to install to run the application.

#### server.js

require("dotenv").config();

Makes possible to access and read from .env files.

const app = express();

So we can use the express framework

Next we create the server, create the socket io instance and connect to the MongoDB with the mongoose.connect

Next we run middlewares.

And then we have routes:

**// automatically load routes**

readdirSync("./routes").map((r) => app.use("/api", require(`./routes/${r}`)));

This line is my favourite as it just loads all the files that are in the folder ./routes/...

**// socketio**

io.on("connection", (socket)

This is socket io, just like a listener, it connects client to server socket.io

Then it waits until user selects the chat to display their id in the command line which is socket.on(“setup”)

Then there are three things socket.io does, wait for client to join chat, wait for client to write a new message and within the new message wait for message to be received. Basically io.on is a sender and io.in is a receiver. Io.off is when socket disconnects

And at last we are running the server by http.listen() to also receive http requests

#### Authentication

Let us start from /server/models/user.js

This file creates a Schema that is sent to the MongoDB database and creates a collection called “users” the name is given at the last line:

export default mongoose.model("User", userSchema);

NOTE: here it says “User” but in database it just adds puts it to lowercase and adds an “s” so it becomes a collection of “users” same goes for the rest of the collections

User schema contains:

Name, email, password, secret, username, about, photo (which is not used anymore but is there in case I will work on this project after the university), following (Array), followers (Array), isAdmin (is not used either) and timestamps (when user registered). All of these are pretty much self-explanatory.

Moving on to /server/routes/authentication.js

It starts with express routing (refer to official docs: [Express routing (expressjs.com)](https://expressjs.com/en/guide/routing.html))

Next we are importing functions from the /server/controllers/authentication.js

And setting up different HTTP request receivers (the first argument they take is the string, actual URL where HTTP request will be sent to, which is not a full URL, it actually starts with /api/… but that is solved in server.js, at the line where it loads routes automatically, so it essentially does a concatenation), the second argument runs a middleware, that has to be satisfied to execute the third function. That middle is requireSignIn which is located in /server/middlewares/index.js and checks whether user has a Token (is logged in) and if the user is indeed logged in, if it is required then we execute functions from /server/controllers/authentication.js

All of the functions: register, hashedPassword, login, thisUser, forgotPassword, profileChange, findUsers, addFollowers, followUser, followingUser, removeFollowerUser, unfollowUser, findUser, getUser, checkForAllUsers.

All of these functions are pretty much self-explanatory but I will mention that all of these functions are asynchronous because they have these two keywords async / await which indicate that they act asynchronously. There are thousands of references to this topic, but I will say that Node.js is actually single-threaded, that is why we need asynchronous functions for it to work

Async / Await in Node.js: [Modern Asynchronous JavaScript with Async and Await (nodejs.dev)](https://nodejs.dev/learn/modern-asynchronous-javascript-with-async-and-await)

Furthermore, after typing await, we are referring to the specific collection in the MongoDB database and fetching the data from it.

One thing is also where we are calling function hashThePassword(password) in hashedPassword function and function CompareThePassword(password, user.password) in login function. These two functions are from /server/helpers/authentication.js

When you enter a password, it is hashed for security purposes, meaning the password field viewed in the database shows some huge random string that is encoded (password-hashing: [Cryptography | Free Full-Text | Password-Hashing Status (mdpi.com)](https://www.mdpi.com/2410-387X/1/2/10))

#### Posting

Let us start from /server/models/post.js just like /server/models/user.js this file creates a MongoDB schema and creates a collection of data in the database.

The collection name is “posts” and has fields: content (what user posted), postedBy (reference to users collection (ref:”User”)), image (not used anymore), likes (reference to users collection), comments. Comments field has sub-fields, such as when it was created and who it is postedBy (again reference to users collection). Reference to collection means it is connected to that collection. And timestamps (for when the post was shared).

Moving onto routes, you can ignore “express-formidable” importation, even though it is used, its purpose is for processing multipart form data, such as files and images. Next importation of requireSignIn and postChangable. requireSignIn was explained in previous section, as for postChangable in server/middlewares/index.js searches for post by Id and checks if the user that shared the post is same as the user who is logged in. all the routers are self-explanatory as they are same as /server/routes/authentication.js

As for some of these routers, that have postChangable as third argument, have two middleware, which means both of the middlewares must be satisfied so the last argument function is executed.

So to say last arguments are the controller functions from /server/controllers/posting.js

In /server/controllers/posting.js instantly ignore anything that is related to cloudinary

Posting.js consists of these functions: sharePost, uploadPicture (related to cloudinary), allPost, oneUserPost, changePost, deleteThePost, renderNewsFeed, likeThePost, unlikeThePost, commentOnPost, removeTheComment.

#### Chatting

Moving onto last one, chatting. First let us start from /server/models/chat.js and /server/models/message.js

Chat.js creates a collection called “chats” in the database with the following fields: chatName, isGroupChat, users (reference to “users” collection), latestMessage (reference to “messages” collection), groupAdmin (reference to “users” collection) and timestamps.

As for Message.js it creates a collection called “messages” in the database with the following fields: sender (reference to “users” collection), content, chat (reference to “chats” collection), readBy (reference to “users” collection) and timestamps.

Nothing is different in /server/routes/chatting.js everything that applies to previously discussed routes, applies to this one too, something I might have forgotten to mention is the first argument string in routers that might look something like this “/text-message/:chatId”. Something different about this is that it has a colon after a slash “/:chatId”. This indicates that chatId is a query and it will differ for all the requests, in this case the query will be a chat id.

Next up, /server/controllers/chatting.js contains all the functions for routes, same as before.

/server/controllers/chatting.js functions: openChat (this specific one is executed when you press a user from the list in the sidedrawer and it opens the private chat), renderChats, createGroupChat, renameGroupChat, removeFromGroupChat, addToGroupChat, sendMessageToChat, checkAllMessages.

Again, functions are pretty self-explanatory.

#### Other

One thing that might be a question is .populate() function. When fetching the data from the database (where it says await): “Population is the process of automatically replacing the specified paths in the document with document(s) from other collection(s)” in some cases you might see .populate(“-password”) here a dash symbol (-) means that the password field from the collection will be excluded when fetching the data.

One also common thing is this line:

res.json(data)

This line sends a response to the client, and is usually executed when everything went well.

This is the end of server folder, it seems a lot but it actually follows the same pattern, even all the non-middleware functions, start with fetching the data from the database and end with sending a response back to the client

### client folder

#### .env.local

Env.local file / dotenv is a text configuration file that gives you an ability to control your application’s constants, which should not be viewed by public. So my client has 2 such constants: NEXT\_PUBLIC\_API and NEXT\_PUBLIC\_SOCKETIO

#### package.json & package-lock.json

Package.json file is used for version control. This file is very important, first it indicates the name of the package, then its version, then description (these three are custom and do not affect the application in any way) next is “main  
:”index.js” now this line is very important as it means that the index.js will be executed first. If we check npm documentation, it says the following: “The main field is a module ID that is the primary entry point to your program”: <https://docs.npmjs.com/cli/v7/configuring-npm/package-json#main>

Next we will see scripts field, this is also very important as we can write a whole command in the string for its sub-fields and just run:

npm {sub-field}

In our case:

npm run dev

npm run build

npm start

npm run lint

npm run dev run the client as a developer, npm run build builds the client application for deployment of the app, and creates a /.next folder. npm start runs the client only if client application was previously built and /.next folder is present. npm run lint is a “static code analysis tool for identifying problematic patterns found in JavaScript code.” ESLint: [ESLint - Wikipedia](https://en.wikipedia.org/wiki/ESLint)

package-lock.json is “automatically generated for any operations where npm modifies either the node\_modules tree, or package. json . It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.”: <https://docs.npmjs.com/cli/v8/configuring-npm/package-lock-json>

And dependencies field is the most important one, it contains all the external packages with their respective versions needs to install to run the application as described above.

#### public/styles

I won’t start discussing each line of this file as it contains over 1000 lines, but I will say that, after this comment:

**/\* Layouts \*/**

**/\* IMPORTANT: THE REST OF STYLINGS BELOW ARE FROM BOOTSTRAP \*/**

Is both bootstrap 4 and bootstrap 5 and I mentioned previously why I did this.

#### Context

/client/context/index.js is very important because it is a user provider, so to say the values

value={[state, setState, selectedChat, setSelectedChat, chats, setChats]}

Are used throughout the whole client, almost all components.

UserContext.Provider docs: [Context – React (reactjs.org)](https://reactjs.org/docs/context.html#contextprovider)

For example you will see “state” and “setState” almost everywhere. Because these variables, provide the logged in user’s information. “selectedChat” and “setSelectedChat” are variables that indicate which chat is selected on the left side of the screen on the chats page. “chats” and “setChats” are for displaying the right chat on the right side after selecting the chat on the left side.

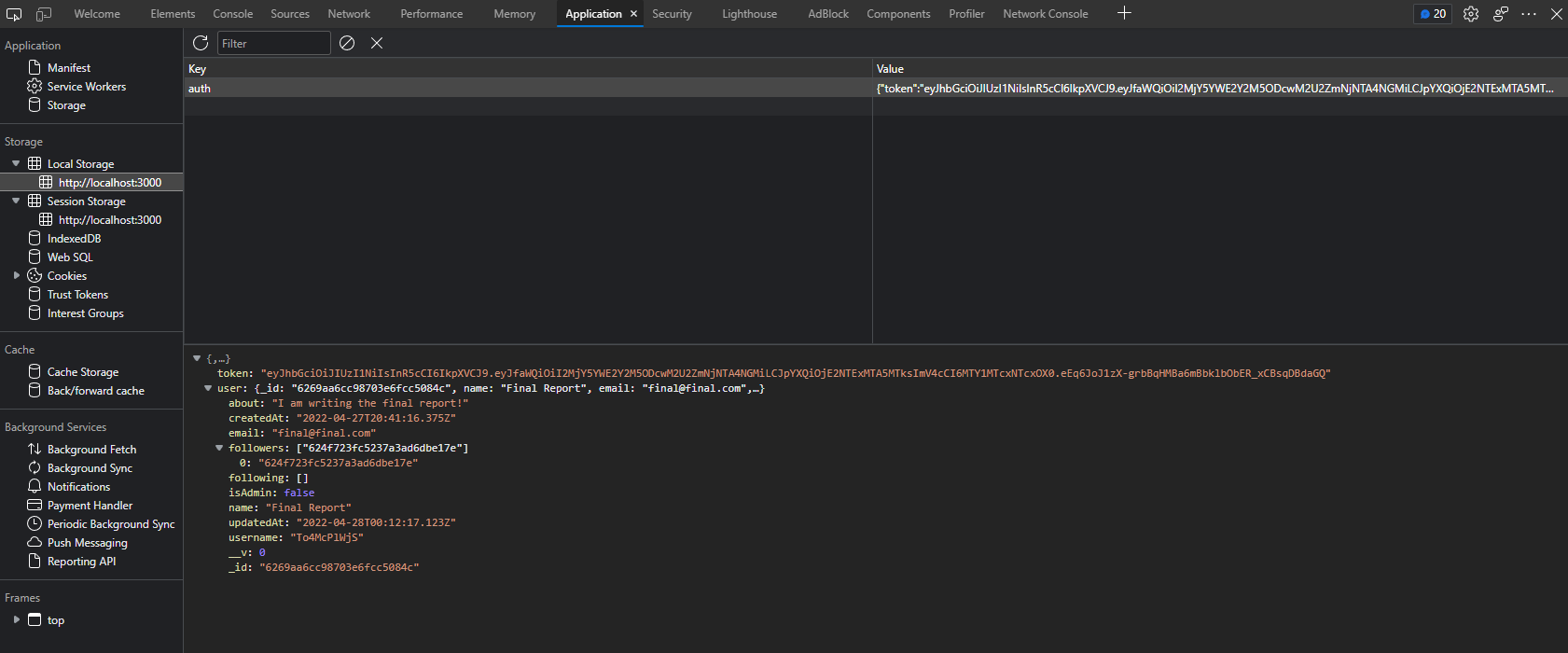
Also the comment where it says Axios, that’s where we start communicating with the server, and set the Bearer Token if the token is not present, then user is directed to /login page.

useEffect(() => {

    setState(JSON.parse(window.localStorage.getItem("auth")));

  }, []);

useEffect() runs on whenever page is loaded, where this specific component is present, and this component is present on every single page. So it sets the state of the user, if the user is logged in and has a “Bearer Token” it is saved in the local storage of the browser and if the user turns off the browser or closes the application in the browser. Whenever user re-opens the web application, the local storage is checked and if the token is present user is considered as logged in and is logged in automatically without having to enter the credentials again on the /login page. Bearer Token can be seen in the Inspect > Application > Local Storage > <http://localhost:3000>:



#### Config

/client/config/ChatFunctions.js has functions that are used mainly in ChatScrolling.jsx component file, apart from “getSenderUser” function which is used in ActiveChat.jsx and ChatSelector.jsx component files.

ChatFunctions.js has five functions: getSenderUser, isIdenticalSenderMargin, isIdenticalUserSender, isIdenticalUser and isLastSentText. Yes these functions seem pretty complicated, but that is not the case, these functions are just boolean / if statement logic. getSender shows which one is the logged in user and which one is the sender of the message, so it returns the sender of the message.

isIdenticalSender and isLastSentText is for displaying the avatar icon on the bottom of the last message sent by the other user (we are the receiving user).

IsIdenticalSenderMargin is for locating the right position of the message, sender left and receiver right.

And isIdenticalUser function is if we (the receiver) send several messages, then this creates some space between these messages that we sent.

#### Pages (refer to components)

Pages folder actually indicates all the possible pages in the applications so each file is a page. Also file names match URL too.

These are possible pages:

Home - <http://localhost:3000/>

Login - <http://localhost:3000/login>

Register - <http://localhost:3000/register>

Reset Password - <http://localhost:3000/forgot>

Individual Post - [http://localhost:3000/post/[\_id](http://localhost:3000/post/%5b_id)]

Newsfeed - <http://localhost:3000/user/postsPage>

Chats - <http://localhost:3000/user/chats>

People you are following - <http://localhost:3000/user/yourFollowList>

Any user profile - [http://localhost:3000/user/[username](http://localhost:3000/user/%5busername)]

Edit your profile - <http://localhost:3000/user/profile/changeForm>

Edit your post - [http://localhost:3000/user/post/[\_id](http://localhost:3000/user/post/%5b_id)]

The way this routing is possible is all thanks to Next.js

#### \_app

In package.json it is mentioned that he first file that executes it is index.js and yes that is true, whenever you open <http://localhost:3000> you will see home page, but the root file of the application is \_app.js and as you can see it is wrapped around <UProvider> a user provider (refer to /client/context/index.js)

Head is where you can provide SEO, styles, description that will be displayed on the tag and so on. Refer to official docs for <Head> component from next: [next/head | Next.js (nextjs.org)](https://nextjs.org/docs/api-reference/next/head)

Next up we see NavigationBar component, which will be mentioned in Components section, and is in fact displayed on all pages, so if you do not see it refresh / re-render it means that application did not restart and everything so far was and is real-time.

<ToastContainer position=”top-center”> component is from “react-toastify” external package, meaning it will display a notification with errors or successful completion of something in the top-centre of the screen

And as last <Component {…pageProps} /> are from the args as you see, meaning all the pages will display below NavigationBar and ToastContainer components, even though ToastContainer, but because it has an absolute position we do not see that.

#### Login page and JSX files

This is the first .jsx file I will discuss, there might be a question, but what is jsx: [Introducing JSX – React (reactjs.org)](https://reactjs.org/docs/introducing-jsx.html)

“It is neither a string not HTML. It is called JSX, and it is a syntax extension to JavaScript”. In fact JSX is a template language. Moreover, if you change all the .jsx files to .js, there will be absolutely no change in the application. .jsx just tells us that, this is a JSX file, it will have its own structure, and it is a React file.

The structure – You will notice that all of the JSX files have the same structure, first import all the necessary functions, then create a const function which is in fact a component, then inside this you will weird first few lines, and it is your first time seeing such variable assignment. Yes, useState() is used for assigning variables, which should have been mentioned previously. If you work with it for some time, you will understand that it is not actually weird assignment of variables and start treating useStates as traditional getters and setters. First variable being a getter and second variable being a setter respectively.

Another thing you will see is a declaration of a router, which is also a quite common line throughout an application. That is used for directing user to a specific URL / page.

After variable assignments you will see either regular functions or asynchronous functions (which means async functions are interacting with the server and database respectively).

Then you would see useEffect() or either before functions, you can just treat it as a function that executes, if given dependencies in [] brackets are satisfied upon start of the page

useEffect hook: [Using the Effect Hook – React (reactjs.org)](https://reactjs.org/docs/hooks-effect.html)

And then you will see a return either will parentheses or squiggly brackets, and inside of it you will see HTML-like syntax, which is in fact JSX, sometimes you might stumble upon <></> tag, which is same as <div></div> tags, defined as “division” or a section in HTML document. Div tag is used as a container for HTML elements.

I will not explain any HTML tags, for that reason refer to any HTML documentation (my preference: [HTML Tutorial (w3schools.com)](https://www.w3schools.com/html/))

But the difference is still there, for example, tag parameters. If let us say tag parameters want to interact with variables a squiggly brackets need to be included, for example: “email={email}” where email in squiggly brackets is a variable from useStates and email is a parameter from the component that is shown in its args in squiggly brackets.

To display any code instead of plain text, squiggly brackets need to be supplied too, for example to display a string in a variable that holds a string from the database like a name or an email.

Then, another common thing is inline styles that need double squiggly brackets like “style={{}}” and any first fields need to be written in camel case syntax instead. For that, refer to docs: [React CSS (w3schools.com)](https://www.w3schools.com/react/react_css.asp)

And at last, very important step, that is often missed is

export default Login;

This line exports the const Login = () => {} function so that whichever file imports this login component file will be able to able use Login like a component, though it will not happen in this application as pages are not used as components, even though they are components.

Another common thing is in JSX, you will see Custom components, everywhere, that is a beauty of React, and it will be explain in the later section.

If I talk about each file in pages folder, I will just repeat myself, so it is much better to explain the syntax.

Furthermore, touching async functions, all of them in the client side are interacting with the server and database as previously mentioned, but how is it done? Here is where Axios package comes in, it sends an HTTP request to the supplied URL and saves the data into the const {data} variable, which is handled accordingly and outputs an error in case of a fail.

Another common line throughout the application is:

e.preventDefault();

It prevents page from refreshing, which gives the feeling of real-time rendering, it is there because, by default pages refresh after an event.

You will also notice that some files are surrounded by square brackets, that exact file is actually a variable, a query. There are 3 cases of such use:

1. [http://localhost:3000/post/[\_id](http://localhost:3000/post/%5b_id)]
2. [http://localhost:3000/user/post/[\_id](http://localhost:3000/user/post/%5b_id)]
3. [http://localhost:3000/user/[username](http://localhost:3000/user/%5busername)]

It does not have to be an ID, it is just a query, and it sure will be sent to the server to be processed.

Now this is the pattern that is followed in all other JSX files, for further technical, thorough and detailed explanations, please refer to official documentations of React ([Getting Started – React (reactjs.org)](https://reactjs.org/docs/getting-started.html)) and Next ([Getting Started | Next.js (nextjs.org)](https://nextjs.org/docs/getting-started))

#### Components

React components are the core feature of react, the whole client is built on custom components. Component can be anything, from big to small, it can be a whole page, or just a small button. For further explanation refer to [Components and Props – React (reactjs.org)](https://reactjs.org/docs/components-and-props.html)

Everything said above in JSX section, applies to components too.

There are a lot of components, and different types, here is the list:

1. Cards
   1. People.jsx
   2. Post.jsx
   3. PostCatalogue.jsx
   4. UserinList.jsx
   5. UserTag.jsx
2. Chat
   1. ActiveChat.jsx
   2. ChatContainer.jsx
   3. ChatGroupPopup.jsx
   4. ChatLoading.jsx
   5. ChatScrolling.jsx
   6. ChatSelector.jsx
   7. UpdateChatGroupPopup.jsx
3. Forms
   1. AuthenticationForm.jsx
   2. CommentingForm.jsx
   3. ForgotThePasswordForm.jsx
   4. SharePostForm.jsx
4. Routes
   1. RouterForUser.js
5. NavigationBar.jsx
6. Search.jsx

You might notice that RouterForUser.js does not indicate .jsx when that is because I forgot to write it, but I will leave it there to prove that both works, this components is for a spinner, if the whole page is loading.

This is the end of technical documentation, for any possible questions refer to official documentation.

# Project Planning

In this section you should discuss your project planning. How successful were you at maintaining momentum, adapting to change, identifying and dealing with risks. You should reflect on your overall achievements, performance, and what you have learnt. Did your methodology prove suitable?

First of all I was thinking about making the frontend, but that did not work because I was bad at CSS and overall styling or design, so I started learning bootstrap and managed somehow to mix both bootstrap and custom styles together, on the challenge week I made Authentication system with registration and login, but it was overcomplicated for no reason and I later simplified it. Then I implemented sharing posts and had issues with database management and fetching data from it, but eventually I managed to fix it, added all the other features, but failed at implementing feature for uploading images to the server and displaying them, so I had to use some third-party web application for it, but it was only for testing purposes and yes there are traces of It in the code but I consider it as a part where I failed to accomplish my ideas as I do not want to rely on third-part web application that require connection to internet in this project. And at last started working on the chatting feature, which did not go well, but did not fail and achieved most of the things I wanted to, excluding some minor bugs. Overall there are little to no errors, existing errors are mentioned in other sections of this report, unless I did not do enough testing that obviously requires some more quality assurance testing, with help of other people, because testing the application of this scale does require quite some time to polish it well.

# Conclusions (1-3 pages)

You should conclude your report by drawing together the work you have undertaken and presenting your overall results. Make clear what has been produced against what was intended. Make comment on what future work could be undertaken to improve or further your work.

This project was very interesting and tiring, the amount of information gained in such a short period of time is very pleasing though and I can definitely say that I benefited from this doing this project. First of all, I will be able to send a CV and include this project in it and it will give me a huge advantage. Later I will definitely brag about what I have done in this project with my friends and maybe even deploy this application to have a private area with my friends possibly, maybe after fixing some bugs. Then this whole project, both back-end and front-end were made by just one person, me. Understanding both back-end and front-end makes me a full-stack developer and it is quite an advantage on itself already, as people usually know either back-end or front-end.

From here, I learned quite a few things about overall creating applications of this scale. I understood, that I am really bad at CSS and will have to practice on that later, as the current CSS styles file is just an abomination from my point of view and I believe many will agree. To fix that I would just stop using CSS at all, instead I will use SCSS/SASS, which is an absolute must do, when making styles. I even know from my friend who is in this industry that everyone uses SCSS and raw CSS is widely condemned. I could also just continue using bootstrap, but honestly, that was the worst mistake I made in this project, because bootstrap is its own thing, it has its own layouts, styles, everything, and if these strict rules are not followed, well you see how awfully the chats page design is made, and completely unresponsive design. I will have to spend hours in practicing for improving on that, I will probably have to learn some tools like Webflow and Figma, I try drawing things there first, to fully understand layouts and website design as it’s not something you can learn from Stack Overflow and it requires a lot of practice.

Moreover, this application, as of now, is by no means perfect, there are bugs! Just like the most evident one there might be smaller ones, like a TypeError that occurs during autofilling credentials on login page. Or the most common one with the unmounted component. Well, all of this is definitely fixable, but as of now is not possible due to the time limit.