Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Bogdan Andrei Chivescu
bac1g16
28404467

April 2019

3YePee - a minimalist, proof of concept social media platform

Project supervisor: Professor Leslie Carr
Second examiner: Dr Gary B Wills

A project report submitted for the award of Bachelor of Science in Computer Science

Abstract

This project is meant to highlight one of the overlooked faults of current social networking sites and present a simple approach to address this fault. The problem I identified is the management of stale data - old posts and inactive groups, along with their metadata. These data serve little to no purpose and waste valuable memory in the server storage unit. A straightforward solution is to delete such resources as soon as they become stale. I demonstrate this approach by developing and evaluating a compact, proof-of-concept social networking application, which has specifying a lifetime for resources as its main feature.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes
- I have acknowledged all sources, and identified any content taken from elsewhere. I have not used any resources produced by anyone else
- I did all the work myself, or with my allocated group, and have not helped anyone else
- The material in the report is genuine, and I have included all my data/code/designs
- I have not submitted any part of this work for another assessment
- My work did not involve human participants, their cells or data, or animals

Contents

1 Introduction **5**

2 Literature review **6**

3 Analysis **8**

3.1 What is out there 8

3.2 Forming requirements 9

4 Design **10**

4.1 JavaScript 10

4.2 Serverless 11

4.3 Backend - Amazon Web Services 11

4.4 Frontend - ReactJS 13

5 Implementation **15**

5.1 Project setup 15

5.2 AWS-Amplify 15

5.3 Lambda Functions and API 16

5.4 React 17

5.5 Timeline 19

5.6 Challenges 21

5.7 Deployment 22

6 Testing **23**

6.1 Frontend 23

6.2 Backend 23

7 Evaluation **24**

7.1 Social networking 24

7.2 Security 25

7.3 Stale data 25

7.4 Possible improvements 26

8 Conclusion **26**

References **28**

A Estimated work schedule **30**

B Login **30**

C Example test event **30**

D Contents of design archive **31**

E Initial project brief **33**

List of Figures

1 Group pages. Screenshots from my smartphone 8

2 Backend diagram. Made with `www.lucidchart.com` 12

3 Groups page mock-up. Made with `www.creately.com` 13

4 User profile mock-up. Made with `www.creately.com` 14

5 Component hierarchy. Made with `www.lucidchart.com` 18

6 Groups page. Screenshot from my device 18

7 User profile. Screenshot from my device 19

8 Commit history, per week 20

9 Stale data deletion flow. Made with `www.lucidchart.com` 21

10 Work schedule from progress report 30

1 Introduction

In the era of the Internet and the World Wide Web, social networking sites play an important role in the development of inter-human relationships. People can connect instantly with family and friends, across great distances. The social circle is not limited by proximity anymore and it can even include employers or complete strangers. Moreover, people use social media (text posts, photos, videos, shared articles, likes, comments...) to express their feelings and opinions better, thus enhancing communication.

However, the underlying, technical side of social networking sites is overlooked by the common user. Aspects such as data representation, memory and bandwidth usage are of interest to developers and software engineers, who are concerned about efficiency, performance and scalability.

As a developer, I observed that some social networking platforms accumulate immense amounts of data in the form of posts (with associated up-votes and comments), shared content and other structural metadata. These data serve an ephemeral purpose while they are relevant to a certain situation or context, then continue to exist on the platform despite their staleness. For instance, think of a "Happy Birthday" written on a friend's timeline five years ago, on Facebook. It is most likely still there, if one would care enough to scroll down and find it. Another example is an online Facebook group created for a school trip. I am now in my third year of university and I can still find such a group (along with the pictures and videos shared in it), even though nobody has been active in that group for more than six years.

All this accumulated stale data occupies valuable space on the storage unit of the server. Instead of resources being freed and reused, they are wasted on storing information which is no longer relevant. One may argue that in 2019 storage is not a problem anymore because memory is cheap, but this has grater implications. As social networking sites grow larger, they will amass more and more data which will become hard to manage at some point. Network traffic will grow needlessly too. If metadata for ALL groups is fetched when one accesses the groups page of a social networking site, considering one browses less than half of those groups, then more than half of the bandwidth was used for no real purpose.

Looking at the issue of stale data from a professional career point of view, users' old activity on social networking sites can affect their chances of being employed. More and more companies require social media background checks for their prospects and surprisingly many candidates fail them because of a controversial picture, video or text post shared years ago. While some content is indeed vindictive and deserves a sanction, many users find themselves in a prejudicial situation. For example, imagine a person posting a photo of oneself with a marijuana plant while on a vacation in Amsterdam and years later being accused of drug use when interviewing for a job. This problem is serious and it is highlighted by many articles in the media (Preston 2011) (Sherman 2013) (Halper 2015).

A recurring concept in this project is the online group, which I define as a smaller

and sometimes limited network within the larger, more diverse social networking service. Members of a group usually have similar interests, which in turn dictate the activity and content in a group. Most groups have one or more admins who regulate member behaviour, but it is not a requirement.

In the following sections I will explain how I built a lightweight social networking application that addresses the problem of stale data.

2 Literature review

According to Boyd & Ellison (2007), social networking applications are "web-based services that allow individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections". Additional to these three fundamental properties, social networking platforms can have other features, such as media sharing or instant messaging (Boyd & Ellison 2007).

For the task of building a social networking application, one has to understand the functional blocks of such a service. Kietzmann et al. (2011) identifies seven such blocks: identity, presence, reputation, relationships, conversations, sharing and groups. The first three categories focus on the individual, while the rest focus on connections with other users. Different social networking sites emphasise on different blocks. For instance Kietzmann et al. (2011) points out that LinkedIn accentuates identity and reputation, while YouTube insists on sharing. Even though it can be argued that all platforms touch on all functional blocks, most social apps focus on one or two.

Social networking literature covers each functional block independently. Lampinen et al. (2009) looks at group interactions on Facebook and shows that "implicit groups" - friends in the offline overlap greatly with "explicit groups" - created on a social network. Moreover, subjects of the study (Lampinen et al. 2009) stated that it was easier to manage the network of friends by organising them into smaller groups and choosing what each group can see and do. The co-presence of multiple groups didn't affect the already experienced user of social media.

With so many social networks out there, one may ask why build another one. The answer is simple: it generates social capital. Williams (2006) defines social capital with an analogy: "Some social actors interact and form a network of individuals - a 'social network' - resulting in positive affective bonds. These in turn yield positive outcomes such as emotional support or the ability to mobilise others". Drawing from the definition, one can say that social capital is used to create more social capital. Putnam (2000) identifies two types of social capital: "bridging" and "bonding". The former refers to connections with individuals from a different background (race, religion, sexual orientation), while the latter represents relationships with close friends or relatives. Another classification introduces the term "maintained social

capital" (Ellison et al. 2007), that is, preserving of social connections after a life-changing event, such as moving home. Note that this maintenance is done over the Internet, with the help of social media.

Even though naysayers believe that social networking services subtract from the social capital by reducing face to face interactions. Ellison et al. (2007) and Norris (2002) demonstrate in their studies that it is not the case. Moreover, Norris (2002) shows that online groups contribute to the social capital with both bridging and bonding functions. The study was however conducted in the United States, so the previous statement is a generalisation.

An important consideration when discussing social networking is the use of smartphones. Modern mobile devices allow individuals to stay connected to their peers and receive updated information anytime and anywhere. Chang et al. (2016) note that users enjoy using social networking applications on smartphones despite privacy concerns, especially when travelling, which in turn builds trust with these services and other users. Moreover, Chang et al. (2016) point out that mobiles make use of location-based services much more effectively than a personal computer. Examples include recommendations for places to go, friend suggestions and local weather and news reports.

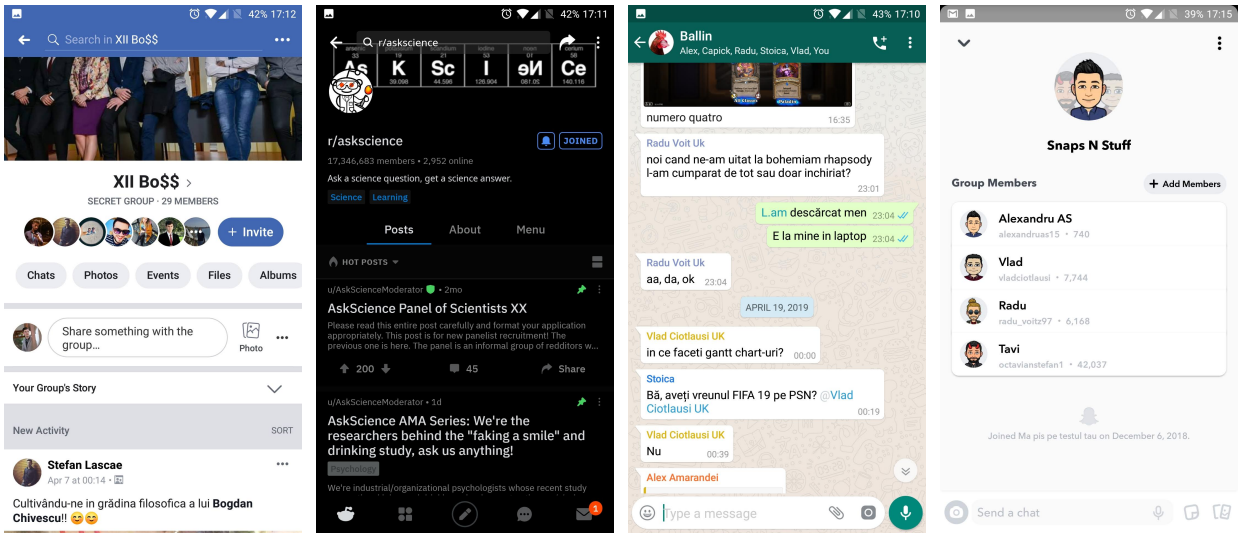
A new trend in the social media environment is the use of self-destructive content, as exemplified by Snapchat and Flickr. Charteris et al. (2014) argue that the ephemeral messages promote a new type of online identity and presence, similar to in-the-moment face-to-face interactions. Given that their messages, photos and videos disappear after a specified time, users, mostly teenagers, are prone to appreciate the content better, due to the scarcity effect (Lynn 1991). Nevertheless, there are also risks associated with ephemeral messages, such as sexting (Charteris et al. 2014), cyber-bullying or trolling (Bergstorm 2011). It is up to the individual to not share sensible content and not encourage inadequate behaviour online.

Despite abundant research on social networking sites, literature doesn't seem to cover the technical side of social media platforms. Academic papers overlook considerations such as system design, performance and efficiency, scalability or cyber-security, as they are the concern of developers. The need for standardised Web development principles is noted by Murugesan et al. (2001), who introduce "Web Engineering, an emerging new discipline, which advocates a process and a systematic approach to development of high quality Internet- and Web-based systems". This field is meant to be dynamic and keep up with the ever-evolving Web. Currently, best practices about building online applications are scattered in Web development manuals, framework documentations, GitHub and StackOverflow.

3 Analysis

3.1 What is out there

Drawing from my observation in the introduction, I noted that most stale data accumulates in online groups, particularly, those groups which are created for the purpose of short-lived events. Examples of such events are trips, vacations, camps, meetings, contests, sport events, parties, courses and so forth. The common pattern is that a group eventually becomes inactive, and it just continues to exist in the online environment, despite not serving any purpose. To elaborate on this issue further, I will discuss how some of the established social networking platforms which offer group functionalities handle the problem of stale data. My chosen subjects are Facebook, Reddit, WhatsApp and Snapchat (Figure 1).



(a) Facebook (b) Reddit (c) WhatsApp (d) Snapchat

Figure 1: Group pages. Screenshots from my smartphone

Facebook offers straightforward group creation. To see, like and comment on the content shared in a group, one has to be a member. Groups can be public (everyone can find and join the group), closed (everyone can find the group, but joining needs approval from an admin) or secret (only members can find the group and request other users to be added by an admin). Any type of file with a maximum size of 100MB can be shared in a group. However, media files will be compressed and lose quality. A group can exist indefinitely and it can be deleted by an admin only when ALL members have left or have been removed from the group. A group can also be archived, meaning that members can see old posts, but not add new ones.

Reddit is a massive collection of forums. Groups take the form of subreddits - threads with a specific topic within the larger forum. Subreddits are mostly public, with few exceptions (users with a paid membership can create private subreddits). Members can join a

public subreddit simply by subscribing. However, in order to create a subreddit or post on existing subreddits, one needs "karma points", which are obtained by getting up-votes on previous posts or comments. Most content on Reddit is text, but images and videos are also supported. A subreddit cannot be deleted, not even by the creator/moderator. A subreddit is similar to a stream of data, so it cannot be archived either.

WhatsApp is not so much about social media as it is a messaging application. Content, that is text or files sent in chats, cannot be up-voted or commented on. Group chat creation is straightforward, although all groups are private. Admins are responsible for adding new members or assigning new admins. As with Facebook, any type of file smaller that 100MB can be shared in a group and media is compressed. What is unique about WhatsApp is that it doesn't store anything on its servers apart from explicitly backed-up conversations, without files. Therefore, any files shared in chats exist solely on the device of the user. Another unique feature is that chats can be exported, including the media.

Snapchat is a new type of social networking service. It is based on ephemeral chats and stories - slideshows of photos and videos which disappear after 24 hours. A group on Snapchat is a shared story ("our story") and an associated group chat, but only for text and media. Stories are public, any member can add content to the story, but only images and short videos. All content in a chat disappears as soon as the chat window is closed, so Snapchat, same as WhatsApp, doesn't store anything on its servers apart from temporary stories. Stories can be saved locally before they self-destroy, but only in the slideshow format. An important note, Snapchat is available only on mobile devices, as a native application.

3.2 Forming requirements

Having looked at what is out there, I can now work out the requirements of this project, keeping in mind the problem that I am trying to solve. The application I am developing needs the following functionalities as a minimum:

- Basic account creation, using a valid email
- Profile customisation - adding a profile picture, setting a nickname and an optional status, deleting the account
- Group creation, setting associated attributes such as name, description, public or private, initial members and LIFETIME
- Adding members to groups
- Removing members from groups
- Searching and joining public groups
- Posting content in groups - adding text and media posts (at least pictures)
- Content rating, similar to Facebook likes

- Commenting on posts
- Changing group settings, such as description or lifetime
- Browsing a list of contacts
- Ability to get in touch with contacts
- Notifications for relevant events, in-app or by email

The lifetime of a group is how long the group will be accessible for members. During this lifetime, new content can be added and existing content can be up-voted or commented on. An important requirement is that after the lifetime is exceeded, the group, along with all the content which was shared in it and the metadata, should be permanently deleted from the web because it is stale. Another requirement is that when a user joins a group, all members of that group should be added to the user's contact list and persist after group deletion. This way, a user can form a network and add fresh contacts to other groups.

The functionality of searching for users is intentionally left out to increase privacy. Moreover, in order for two users to be in each other's contact list, they need to be or to have been members of the same group. Instant messaging is also left out as it is not relevant to the problem of stale data and is therefore outside the scope of this project.

My goal is to build a lightweight social networking application which satisfies the requirements stated above. Additionally, I intend to make the frontend of the system adaptive, meaning that it will change according to screen size (it will be mobile friendly). The strategy to achieve this goal is to construct a skeleton of the application and then add each functionality gradually, both in the server side and the client side. The challenges of this project are (1) finding a method to delete stale data behind the scenes, without any user intervention, (2) implementing a notifications system and (3) making the frontend adaptive.

4 Design

4.1 JavaScript

Development of the application starts with design. The first decision to be made is choosing the appropriate programming language. My choice is JavaScript because I have worked extensively with it during previous coursework and it can be used for both frontend (it runs in the browser) and backend (it runs in the Node.js environment). Moreover, it is the language of most modern Web development frameworks, package management is made easy with the command line tool `npm` and there is plenty of support in the online community. Other viable options were Python or Java for the backend and PHP for the frontend.

4.2 Serverless

Traditionally, a Web application needed a physical or virtual server to provide the frontend and process incoming requests. The developer was responsible for the deployment and maintenance of the server and its resources, such as operating system, memory used for processing and storage units. However, with the rise of cloud computing, the hassle of setting up the underlying infrastructure of a Web service has been abstracted away from developers. The cloud provider is responsible for allocating the necessary resources for code execution, thus allowing developers to focus on writing the code and to deliver a finished product faster. A modern solution for building lightweight Web applications on the cloud is Serverless.

Serverless is an execution model in which the code is run inside ephemeral, stateless containers that can be triggered by various events, such as a HTTP request. The code executed in containers usually takes the form of a function, therefore Serverless is sometimes called "Function as a Service" or "FaaS". The main benefit of Serverless is that it scales automatically and the cloud provider charges only for what is computed. For instance, if a function takes 100ms to execute and it is triggered once a day, the total cost for a month would be 100 * 30 = 3000ms of computing time. This greatly reduces the cost of serving a Web application, as idle time is not charged. As the name suggests, in Serverless there is no server which runs continuously, but only temporary instances that are active for the duration of executing a function.

I chose to develop my social networking application with Serverless because of the benefits stated above and because it discourages me from writing monolithic code. Separating the functionality of the backend into independent functions allows me to write cleaner, more maintainable code which is easier to debug and test.

4.3 Backend - Amazon Web Services

There are three well-known cloud providers which support development in the Serverless architecture: Amazon Web Services (Lambda functions), Google Cloud (Google App Engine) and Microsoft Azure (Azure Functions). I picked Amazon Web Services because, as with JavaScript, I have worked with it in previous coursework and because it offers a one-year free tier which includes all the services I need for this project. Therefore, the cost for developing, testing and demonstrating my social networking platform will be minimal, or even nil. Additionally, the online developer console is intuitive and, if preferred, one can interact with it from a command line interface.

Figure 2 represents a schema of the backend of my application. Even though Lambdas are the central piece of the system, they have to be combined with other services in order to provide the necessary functionality. Luckily, Amazon has great support for integration of its services. First, the frontend resources are hosted in a S3 bucket, which serves them to the Web. The client makes requests, or API calls, to API Gateway, which is nothing but

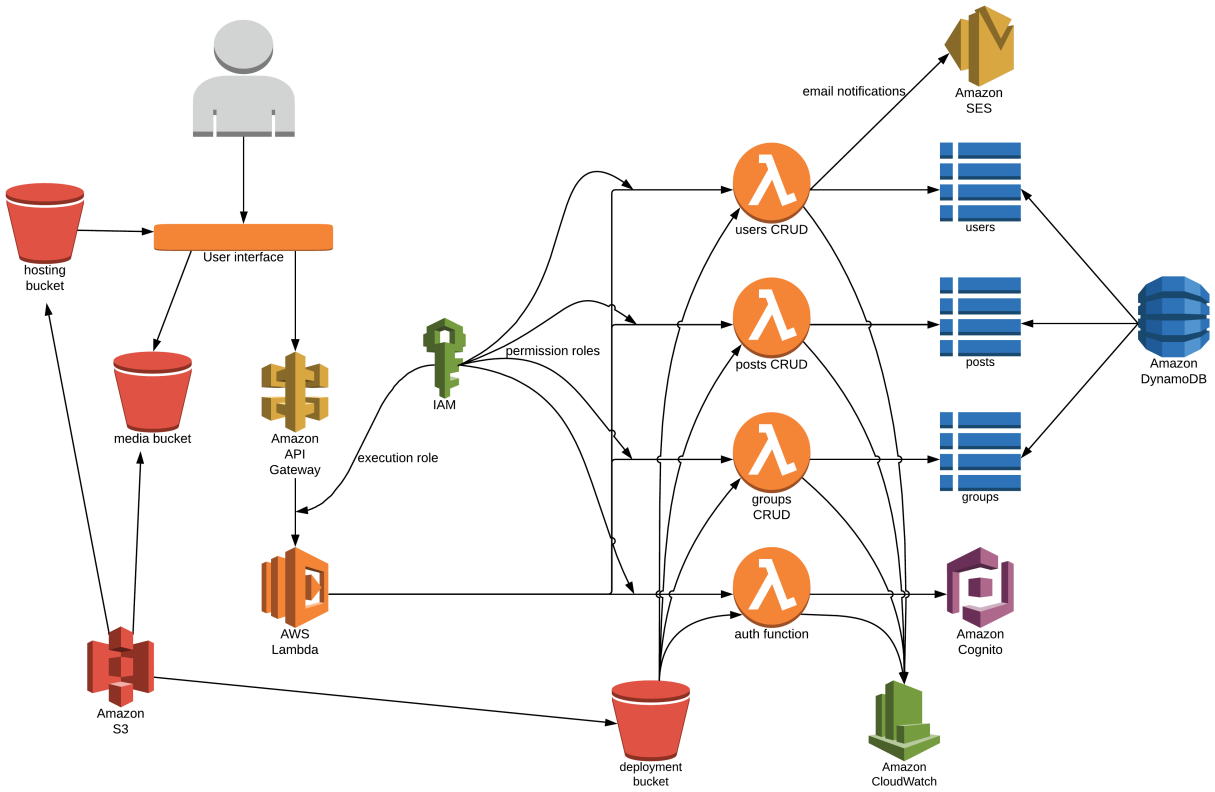


Figure 2: Backend diagram. Made with `www.lucidchart.com`

a HTTP server (the only continuously active server in the Serverless architecture of AWS). The gateway has an execution role from Identity and Access Management attached to it, which allows it to execute Lambda functions when a request comes in. Each Lambda also has an associated permission role, which specifies what other Amazon services it is allowed to interact with. As Lambdas can become quite large in size, they need to be stored in a deployment bucket. CloudWatch is used to monitor functions activity, such as how many times they were called, what event triggered them and how long they took to execute. It is also used as a log output, for debugging.

I need four Lambdas to implement the functionality of my application. The first is responsible for authentication of users into the app. The identity and credentials provider is Cognito, which handles email validation automatically and can even support Two Factor Authentication. The other three functions are each responsible for CRUD operations into a non-relational DynamoDB table (users, groups, posts). I chose Dynamo and not a relational database because it integrates nicely with Lambda and it supports storing of lists, maps and other JSON objects. Additionally, the Lambda which handles users is also responsible for sending emails with Simple Email Service. The associated media of each post is stored in yet another S3 bucket, which is accessed directly from the frontend.

4.4 Frontend - ReactJS

The user interface of my project will be built with ReactJS, a popular set of JavaScript libraries used for frontend development. React makes use of encapsulated, stateful components, constructed with the object-oriented programming syntax introduced in ECMAScript6. Views are built from these reusable components, which are either predefined in external libraries or written locally by the developer. However, the main feature of React is that it holds its own Virtual Document Object Model, which can be rendered and re-rendered server-side, keeping state away from the real DOM in the browser. The advantage of this technique is that changes in the state are reflected automatically in the UI without refreshing the page, which makes the app faster and more dynamic.

The other frameworks I considered for developing the frontend are AngularJS ans Vue.js. However, I chose React because of my familiarity with it due to earlier coursework and because it minimises the hassle of writing HTML and CSS (components are build with JavaSript classes, not HTML templates). The code is declarative and easy to debug and there is plenty of support from online communities. Additional styling of components will be made, where possible, with the well-established Bootstrap (a massive set of predefined CSS classes).

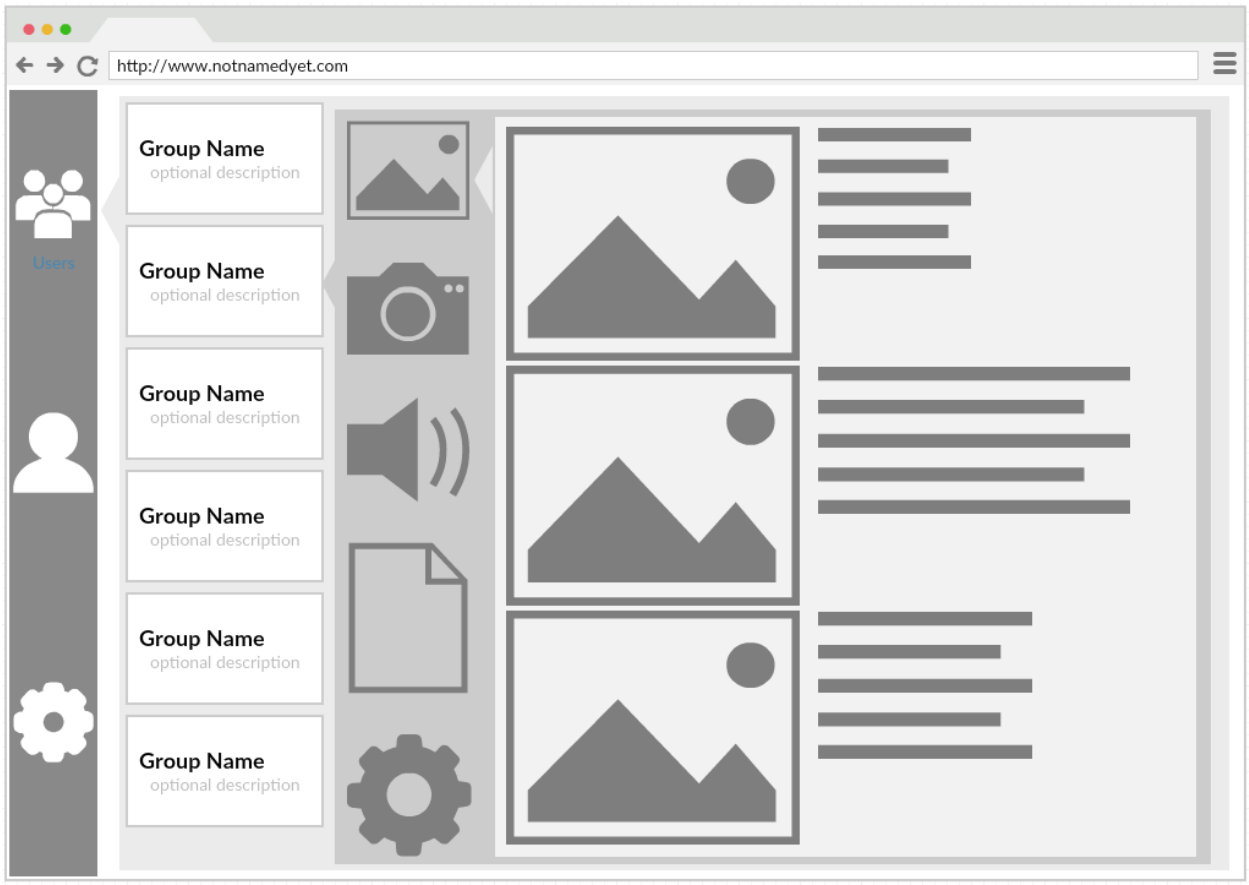


Figure 3: Groups page mock-up. Made with `www.creately.com`

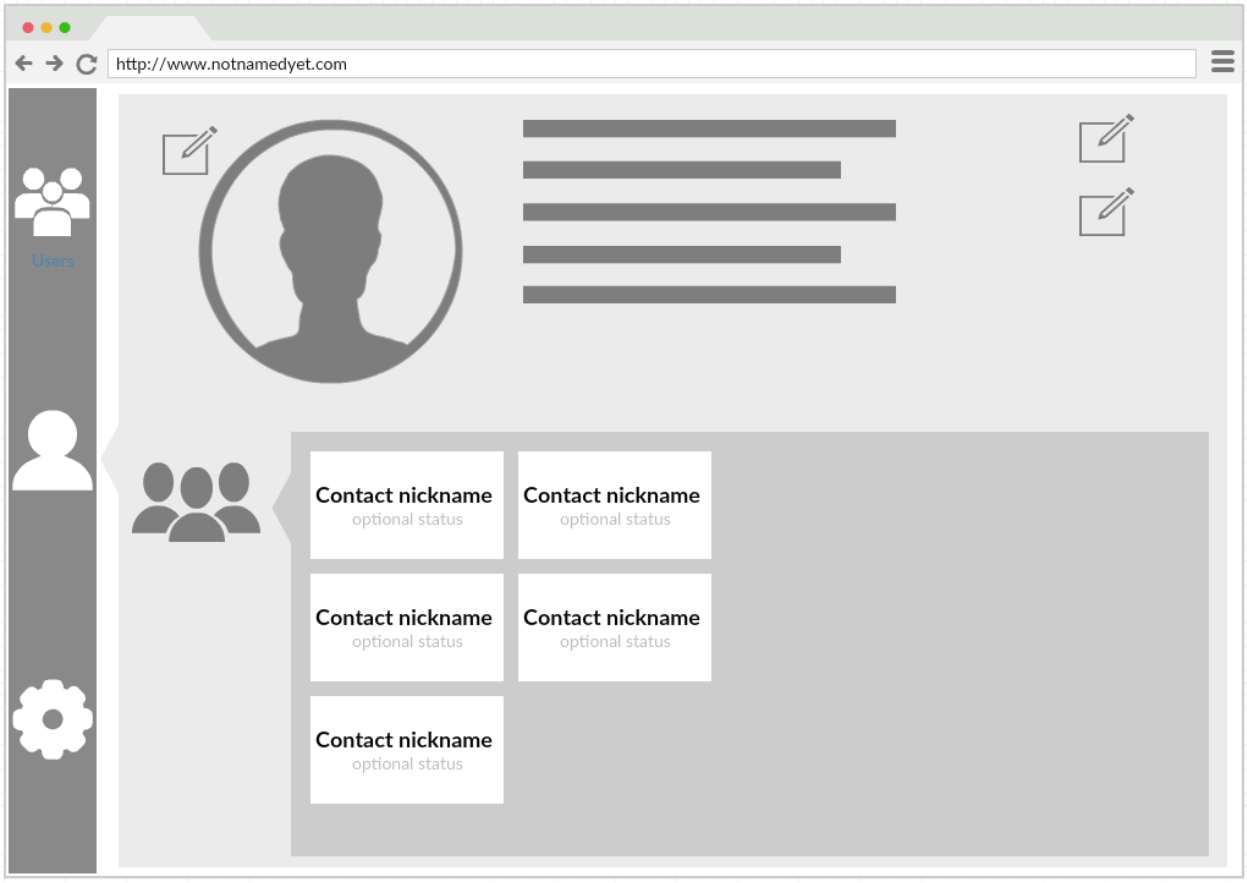


Figure 4: User profile mock-up. Made with `www.creately.com`

The storyboards in Figures 3 and 4 roughly illustrate how the UI would look like on a desktop. Figure 3 depicts a user browsing the images shared in a particular group and Figure 4 demonstrates viewing the profile page of the logged in user. Navigation is facilitated by stacking menus on the left side. I aim to make the interface as intuitive as possible by using suggestive icons and a simple, fluid layout. However, these illustrations are just prototypes. The finished application may use additional or even completely different icons, fonts and styles. The colour palette is also subject to change.

The mobile version of the UI is also an important detail of frontend development. On mobile devices, the layout and navigation will most certainly need to be different to those on desktops. Modern guides to designing Web products encourage the "Mobile First" rule, a principle which states that applications should have a responsive interface in respect to screen size. This is due to mobile internet usage surpassing desktop usage back in 2016. React and CSS3 are flexible enough to allow dynamic resizing and restructuring of the user interface such that mobiles will be able to use the social networking application with ease.

5 Implementation

5.1 Project setup

The first thing to do when starting development with React is initialise a new out-of-the-box project with the command line tool `create-react-app`. Under the hood, this tool uses other well-established projects to power the newly created application: Babel is used to compile newer versions of JavaScript into the browser-supported ECMAScript5 and Webpack is used to bundle all source files into a single, compact script to be used in the browser. Additionally, `create-react-app` also sets up a Git repository. To start a development server on the local host, I use the command `npm start`. A particularly useful feature of the local server is that it refreshes itself automatically when I make a change in the code, without my having to run `npm start` again. I do my development on a Ubuntu VM, using the IDE WebStorm.

5.2 AWS-Amplify

Amplify is a powerful framework used to develop applications with a serverless backend on Amazon Web Services. It is made up of three different items that integrate nicely with React to facilitate the development and deployment of a reliable Web application. I describe each item in the following paragraphs of this section.

The **Amplify Command Line Interface (CLI)** is a comprehensive toolchain used to create and manage resources on AWS. I use it in this project to build the serverless infrastructure on the cloud. Particularly, I do the following:

- add authentication with Cognito (user pool and function to handle authentication are created automatically)
- create tables in DynamoDB for users, groups and posts data (non-relational database only needs a partition key for each table)
- create Lambda functions to access the tables (code for basic reading and writing in the table is generated automatically and CloudWatch is set up automatically as well)
- create an API Gateway to trigger the functions (different path for each function and access restricted to authenticated users only)
- add a storage unit on S3 (bucket for media such as profile pictures and content in groups)
- add a hosting unit on S3 (public bucket to serve the UI of the application)

Additional, finer-grained operations, such as enabling batch operations on Dynamo tables and adding permissions to use Simple Email Service to Lambdas, have to be done manually,

from the online developer console.

The **`aws-amplify` package** is a JavaScript library used to interact with the backend from the React-enabled UI. I use three modules from this package:

- Auth - used to log the user in and out, change or reset the password and retrieve basic details, such as email
- API - used to make various requests to the Gateway and trigger Lambda functions
- Storage - used to upload files to or retrieve files from the media bucket

The **`aws-amplify-react` package** is another JavaScript library which contains higher-order components to be used with React. A higher-order component is a function that takes a component and returns a new component. The only such function which I use from this library is `withAuthenticator`. What this does is handle the authentication flow by restricting access to the app behind a login and sign up view. In other words, it prevents unauthenticated users from interacting with the underlying application. Higher-order components greatly reduce the workload of a developer by providing basic, recurring functionalities out-of-the-box.

5.3 Lambda Functions and API

The Lambdas run in the Node.js environment on the cloud. As they will be triggered by HTTP requests, they are written as Express applications. Express is a minimalist and flexible Web application framework which allows the developer to create a robust, RESTful application program interface quickly and with ease. What Express does is provide a means to define handlers for different HTTP methods on different endpoints. In the serverless architecture, a single Lambda contains handlers for multiple endpoints associated with the same resource (a DynamoDB table), hence the big size and the need for a deployment bucket. The endpoints of my project are as follows:

- `GET /profile` - retrieve data about the currently logged-in user
- `GET /profile/contacts` - retrieve a list of contacts of the currently logged-in user
- `POST /profile` - create a database record for a new user
- `POST /profile/contact` - notify a contact by email that the user wants to get in touch
- `PUT /profile` - edit attributes of the user, such as nickname or status
- `PUT /profile/join` - add new contacts to each other's lists
- `PUT /profile/delete` - remove stale data, such as expired groups, from the user record
- `PATCH /profile` - add a notification for the user

- `PATCH /profile/clear` - clear all notifications from the user record
- `DELETE /profile` - delete the user record
- `GET /groups` - retrieve a list of groups, while deleting stale ones
- `GET /groups/:name` - retrieve the complete record of the group with the given name
- `GET /groups/search/:token` - retrieve search results for the given token
- `POST /groups` - create a database record for a new group
- `PUT /groups/:name` - edit attributes of the group with the given name
- `PUT /groups/:action/:name` - add members to or remove members from groups
- `DELETE /groups/:name` - delete group record
- `GET /posts` - retrieve list of posts
- `POST /posts` - create a database record for a new post
- `PUT /posts/:action/:id` - add or remove up-votes for the post with the given ID
- `PUT /posts/comment/:id` - add a comment to the post with the given ID
- `DELETE /posts` - batch delete multiple posts

The external libraries used in the backend are:

- `aws-sdk` - module which provides the interface to Amazon Web Services (DynamoDB and Simple Email Service)
- `aws-serverless-express` - middleware for Express, which makes it compatible with the serverless architecture of AWS
- `body-parser` - middleware for Express, used to parse the body of HTTP request
- `express` - the Express application framework
- `nodemailer` - wrapper for Simple Email Service which simplifies sending emails
- `sillyname` - nickname generator for new users

5.4 React

Figure 5 shows the hierarchy of components that make up the application. Note, there is no inheritance relation in React, every component is a separate entity. The parent component in the tree passes properties to the components under it. Thus, the relevant API calls to retrieve information are made in key components, then data is passed down to the child components that need it. Therefore, the application doesn't make unnecessary API calls. Modifier functions are also passed down along with the data, such that modifications made to the application state are reflected everywhere. For instance, if a user changes their nickname in `ProfileInfo`, the updated nickname will appear in `GroupContent`, without the app having

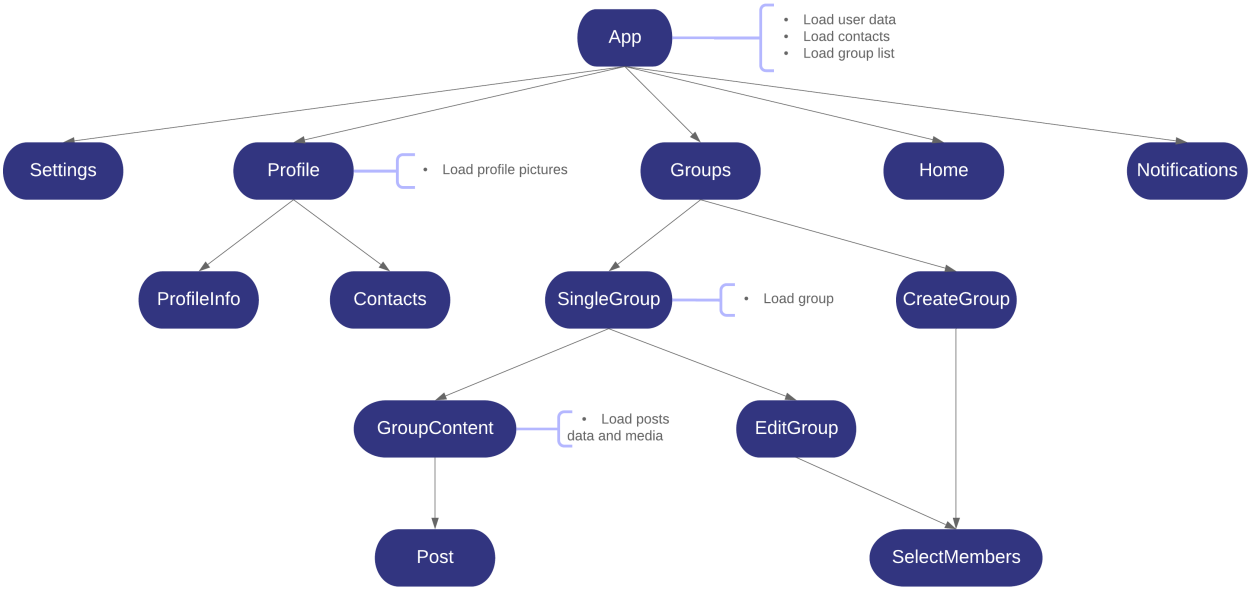
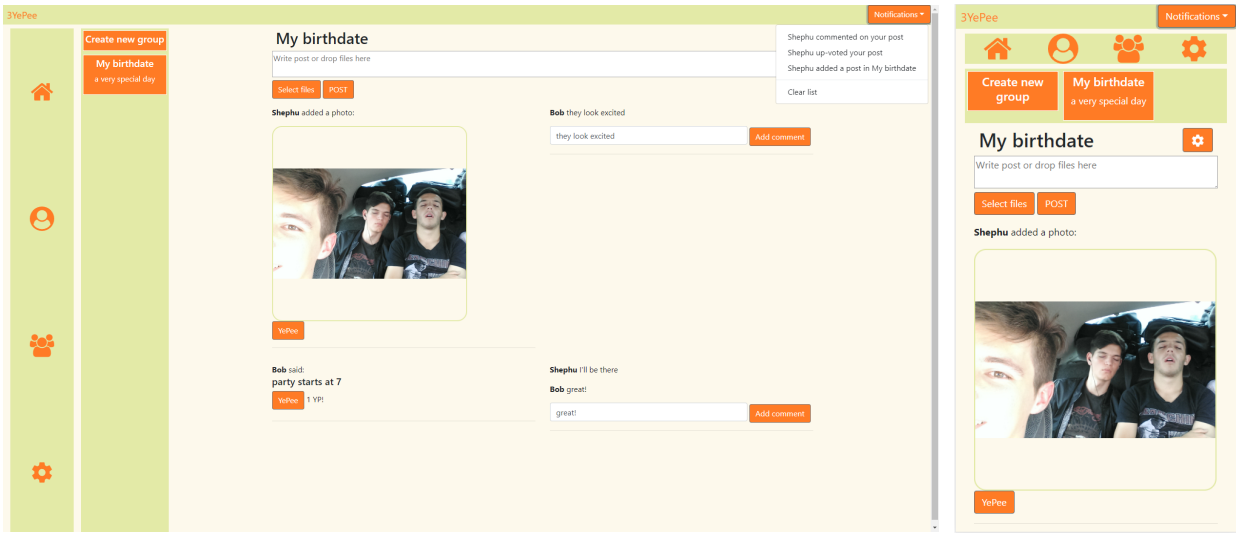


Figure 5: Component hierarchy. Made with `www.lucidchart.com`

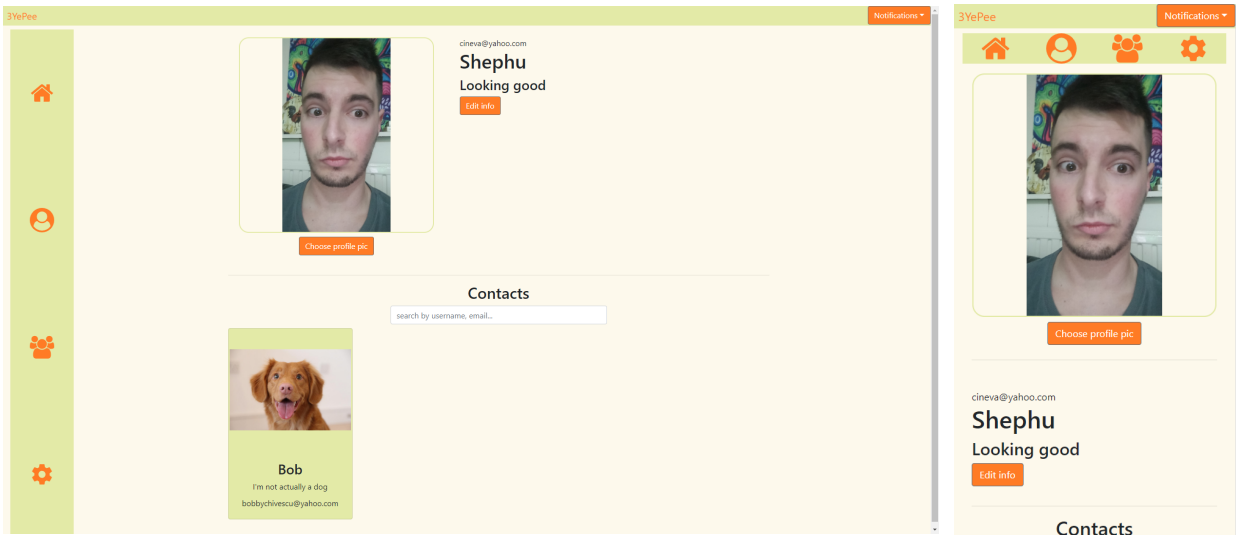
to refresh or make additional requests to the backend. All API calls are asynchronous or have a callback function associated with them, such that re-rendering of the UI or other automatic React operations are not hindered.



(a) Desktop (b) Mobile

Figure 6: Groups page. Screenshot from my device

Figure 6 shows how the group page looks like on the actual social networking application. The notification panel can be seen in the top right corner of Figure 6a. The implementation looks somewhat similar to the mock in Figure 3, although the colours are completely different. This is due to design decisions being revised during implementation because of feedback. The



(a) Desktop (b) Mobile

Figure 7: User profile. Screenshot from my device

mobile version, with the navigation menus moved to the top is shown in Figure 6b. The user profile page displayed in Figure 7 complies with the mock in Figure 4 as far as layout is concerned, but again, colours are different.

The external libraries used in the frontend are:

- `aws-amplify` and `aws-amplify-react` - explained in section 5.2
- `axios` - used to get the text from .txt files in the storage bucket
- `bootstrap` - used to style HTML elements
- `husky`, `lint-staged` and `prettier` - used to format code before each commit
- `react`, `react-dom` and `react-scripts` - the main packages of the React framework
- `react-router` and `react-router-dom` - used to handle routing of the app
- `react-datetime-picker` and `react-dropzone` - singular, high-level components
- `react-icons-kit` - source of icons in the navigation menu
- `reactstrap` - basic styled components, such as buttons, cards, inputs and so forth
- `uuid` - used to generate unique IDs for posts

5.5 Timeline

Development of this project (the code) has taken approximately two months, February and March 2019. The commit history is shown in Figure 8. In contrast with the work schedule estimated in December (available in Appendix A as Figure 10), development has

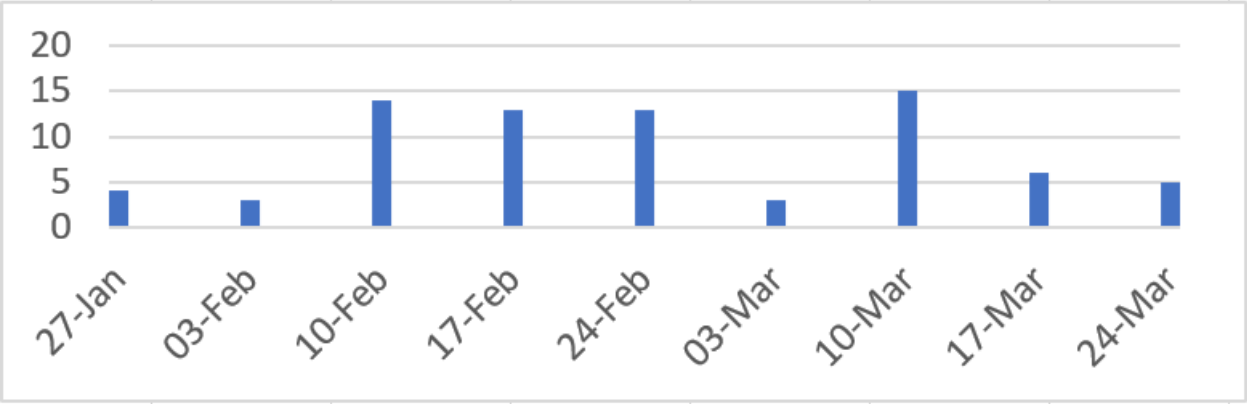


Figure 8: Commit history, per week

taken less time than expected, with individual tasks being offset by a few days. However, the order in which functionalities were added respected to some extent the initial plan. When implementing a new functionality, I made sure to do both backend and frontend before moving on to a new task. My timeline was as follows:

1. set up the project and quickly implement login, using the `withAuthenticator` higher-order component, as explained in section 5.2
2. create the user table and Lambda for manipulating it, along with the UI for changing nickname and status
3. add the storage bucket to the backend and profile picture to the frontend
4. create the settings page to allow logout and more profile customisation (password change, making the email private)
5. create the groups table and associated Lambda, along with basic UI for group creation
6. add frontend for post creation (upload text and images to storage bucket)
7. add contact list to user table and associated view in profile page
8. implement add members to groups functionality, along with contact list joining
9. create posts table and Lambda, add up-votes and comments to post component
10. implement automatic deletion of stale data from both the database and the storage bucket
11. add search and join group functionality
12. add notification functionality, in-app and by email

The enumeration is brief, as there were many more smaller, intermediate tasks. Important code refactoring took place between individual tasks as well, to make the code clearer to read and easier to enrich. Refactoring was also done to improve performance. For instance, a loop of API calls to a single backend operation (getting and deleting the posts) was replaced with a single API call and a batch operation. Another example is the emailing functionality,

which was initially implemented in a separate Lambda function, but was later moved to the function which manipulates the users table, to save API calls and improve latency. In the serverless architecture, each API call, Lambda execution, CloudWatch log or DynamoDB access adds to the total cost, so minimizing the number of operations is a priority when developing an application.

5.6 Challenges

As mentioned in section 3.2, this project had three challenges: resolving deletion of stale data, implementing notifications and making the user interface responsive.

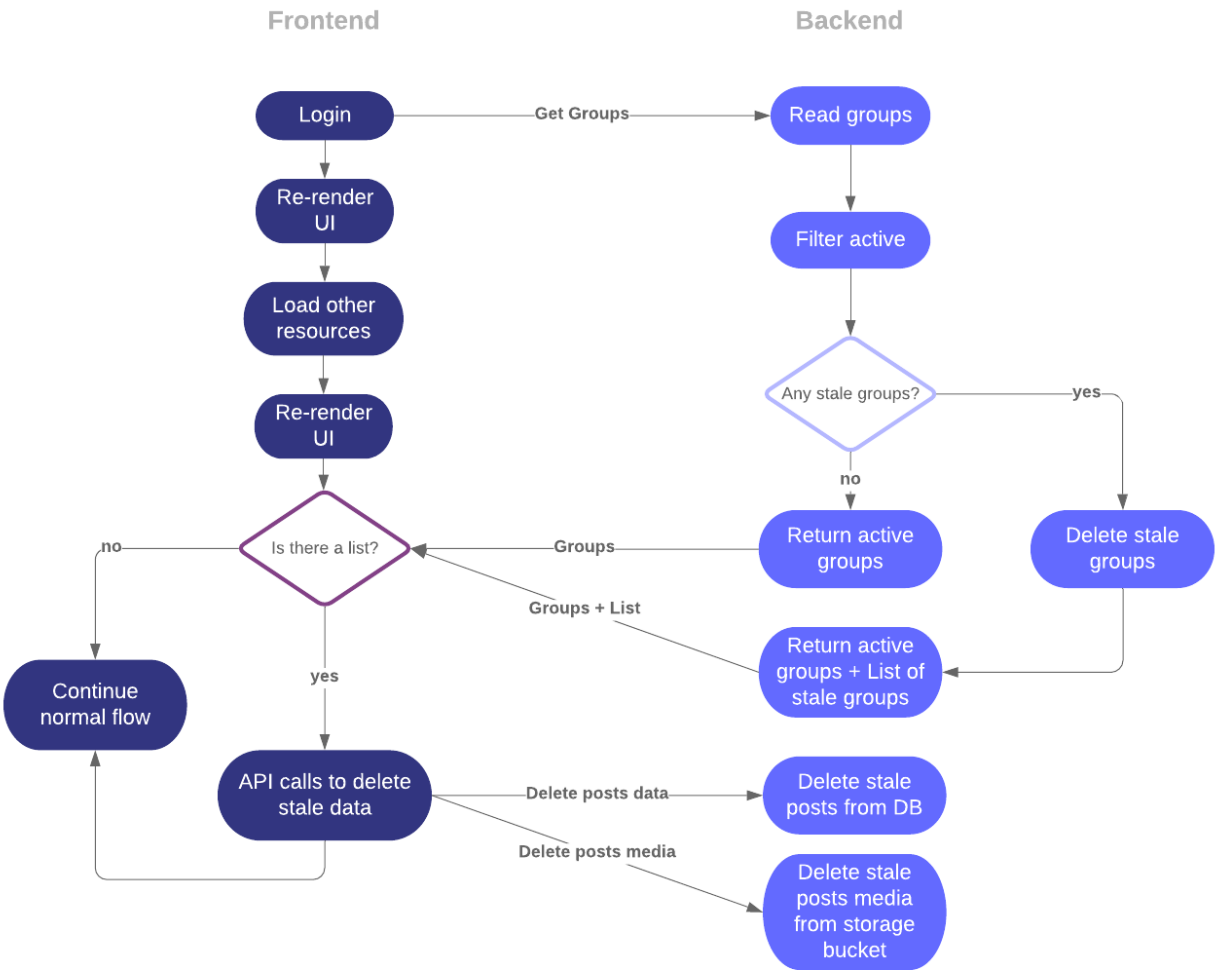


Figure 9: Stale data deletion flow. Made with `www.lucidchart.com`

In this social networking application, stale data takes the form of groups which exceeded their lifetime. Such groups are identified when a user logs in and the app tries to retrieve

their groups. The handler which reads the data applies a filter and selects only the active groups, while computing a list of the groups that need to be deleted, if they exist. The function then deletes all stale groups from the database before returning the active ones and the list. In the frontend, if a list of stale groups is received, a call is made to the storage bucket to delete all posts media associated with those groups and another call is made to the Lambda that deletes those posts from the database. These requests are made only once, from the application of the first user who logs in after one or more groups expire. A graphical representation of this explanation is available in Figure 9. The deletion process is not completed with a single API call because Lambdas have limited memory (128MB) and because it would cause too much latency in the application.

A notification is made of two elements: informative text and path to the group or post where something has happened. In the application, every time a user makes an action in a group, the other members of the group will be notified. This is done by an API call to add a notification object (text and path) to the relevant user records. This is where the non-relational database has the advantage: it allows storing of lists of JSON objects. An interval is also set to check for new notifications every five minutes. However, the users are responsible for clearing the list of notifications. The UI for the notifications is simple, a Facebook style drop-down menu (top right corner of Figure 6a).

The task of making the frontend adaptive has been a continuous challenge throughout development. The key to a responsive user interface is CSS3 media queries. The most important feature of a media query is that it lets the application change the styling of HTML elements based on screen size. Therefore, the same UI will look different on mobile and desktop. Built on top of media queries and the CSS flexbox is the Bootstrap Grid system, which divides the screen into rows and columns, like a table. The columns usually have a specified width, which can change according to screen width. For instance, if two elements are side-by-side on desktop, they will be top-bottom on mobile. I was able to make the UI of my application responsive by using a combination of media queries and Bootstrap grid classes. Examples are shown in Figures 6 and 7.

5.7 Deployment

This is the last step of development. The completed social networking application is available here. Still, a few things need to be mentioned. Amazon Web Services and the Amplify framework allow two types of deployment, production or development. In production, the app is deployed with CloudFront - the content deliver network (CDN) service of AWS. This would optimise the application to make it faster and secure it with HTTPS. However, this option would also be more expensive and, more importantly, this project is not ready to go into production just yet. Legal matters have not been considered and there is no Terms and Conditions or Privacy Policy put together. I believe these issues should be discussed with legal experts first. Therefore, for the purpose of demonstration, the app is deployed to the development environment. Details and credentials for login are available in Appendix B.

6 Testing

6.1 Frontend

When it comes to testing user interfaces, there is no standard. Testing can be split into three categories: unit, integration and end-to-end. Unit testing in React means that each defined component should get its own test, to verify if it behaves correctly when it receives or doesn't receive properties. Whether state updates correctly when certain methods are called can also be checked in unit testing, but it is discouraged. Integration testing goes one step further to evaluate if components interact with each other the right way. End-to-end testing is self-explanatory, it checks the functionality of the application as a whole. The common practice is to apply only one type of testing, however a combination of two is used if there is need for extended testing coverage.

A popular test runner for JavaScript which is commonly used with React is Jest. The Jest framework provides simple methods to setup the testing environment, assert exact behaviour and run tests in parallel using `npm`. Although it can be used for end-to-end as well, Jest is recommended for unit and integration testing. Another popular framework, this time recommended for end-to-end testing, is Cypress. Cypress simulates visiting a Web page and interacting with it. It uses a jQuery-like syntax to find elements in the page and has simple methods to assert values. Both frameworks have their advocates and enough support in the community.

Although testing is essential before deploying an application into production, it also presents a big drawback for developers: it takes a lot of time and effort. Most often, writing suitable tests takes as much time and lines of code as writing the application itself. This is valid for any type of software. In the case of testing the UI of a Web application, if it gets its data from an external source via API calls, those will need to be mocked too, using yet another framework. In serverless, it is absolutely necessary that API calls are mocked in testing in order to not cause unexpected expenses. These are the reasons why I have not written any tests for my UI. However, I performed extensive manual testing, going through every state I could when adding a new feature to the application.

6.2 Backend

As cloud computing is a relatively new technology, strategies and tools for testing it are also in their early stages. The developer is not aware of the infrastructure of the backend because the cloud takes care of it, so setting up a test runner similar to the actual production environment can become difficult. However, developing serverless applications deployed with AWS means that only the Lambdas need testing.

If Lambdas are written in JavaScript, they can be tested as Node applications, with `npm`,

just like the frontend. To that end, one can use the Jest framework, which was described in the previous section. Another option for testing comes from the Express framework. Mocha is a simple, parallel test runner frequently used with Express applications, which is particularly suited for working with asynchronous code. It is also flexible in the sense that it can be integrated easily with other frameworks or external libraries. The common practice is to use Chai, an assertion library, in combination with Mocha to facilitate testing even more. Still, testing Lambdas involves mocking of external services, just like in the frontend, where API calls must be simulated. In the backend of this project, interactions with DynamoDB and Simple Email Service need to be mocked in order to reduce expenses.

In addition to all the testing frameworks, AWS provides yet another way to test Lambdas, which is considered by many the best and the easiest. The online developer console allows configuration of test events simply by defining a JSON object (usually copied and pasted from the logs in CloudWatch). A test can then be run instantly by clicking a button in the page of the respective function. An example of such a test event can be found in Appendix C. The Amplify framework also provides a way to invoke functions locally in the same manner as in the online developer console, but as Amplify is a novel tool, support from the community is still limited.

7 Evaluation

I will evaluate this project from three points of view: I will categorise it as a social networking platform, I will talk about security considerations and I will explain how I believe it solves the problem of stale data. Finally, I will discuss what could be improved. The evaluation is performed based on my reflection on the usage of the application. I chose not to involve human participants in this assessment because the aspects I am trying to appraise are not obvious to the end user. A person only interacts with the UI and is usually not aware of or even cares about any implementation details such as what happens with stale data or what requests are made to which endpoints.

7.1 Social networking

As mentioned in Section 2, a social networking site is identified by its necessary functionalities: user profiles, lists of connections and, optionally, media sharing and instant messaging. The application I developed meets the first requirement by allowing the user to customise their account with a public nickname, status and profile picture. Furthermore, a user can choose if their email is private or not. When a user enters a group, the existing members are added to the user's contact list, which can be browsed in the profile page, thus meeting the second requirement. Groups support sharing, up-voting and commenting on text and image posts, hence integrating social media into the application as well. Even though my platform doesn't support instant messaging, users are encouraged to get in tough by email. If the

email of a user is public, this is simple. If not, the application offers the option to send a request to get in touch. Additional features meant to further improve the experience of the user are the notification system and the responsive UI.

7.2 Security

The cloud provider takes care of most cyber-security threats in serverless applications. To name a few, SQL injections are impossible because my app uses a non-relational database, code injections, cross-site request forgeries and brute-forcing are handled by API Gateway, bogus files cannot be uploaded to S3 and authorisation from Cognito and IAM cannot be bypassed. React is also a secure framework, which doesn't allow cross-site scripting and hides stack traces in the case of an error.

However, it is the responsibility of the developer to ensure that the logic of the application is foolproof. To that end, I made sure there are no loose endpoints, that is I implemented extra checks to restrict access. For instance, only the creator of a group can edit or delete it and only members of a group can add other members. One additional layer of protection for the user is provided by the use of nicknames or, better said, the non-use of real names and by the feature of concealing the email.

7.3 Stale data

The underlying method to destroy stale groups was explained in Section 5.6 and illustrated in a high-level perspective in Figure 9. From the end user point of view, this process happens automatically and seamlessly. The latency caused by eventual data deletion when retrieving the groups is minimal and hardly ever noticed. The subsequent API calls to delete posts data are made asynchronously, therefore do not impede the loading or the use of the platform at all. Moreover, this whole process is scalable and cheap, because the deletion of groups and posts from DynamoDB is made in batch, with a single operation. Even though posts media can only be deleted one-by-one from S3, this is not a problem because requests are asynchronous and S3 charges based on how much memory is used, not how many times it is accessed. I have tested the deletion of stale data with up to five expired groups at a time and I can confirm that all data (groups, posts, media) were deleted successfully and my experience was smooth and not obstructed at all.

Deleting stale data brings two benefits to this application (or any other platform that might use this approach): it cuts costs and it increases performance. By freeing memory dynamically, there is no need for a continuously growing and therefore expensive storage unit. The necessary memory is fairly proportional to the number of users, hence the growth rate for the required storage space is much smaller than that of a site which accumulates all data, such as Facebook. In Serverless and the other architectures used on the cloud, this is particularly advantageous because storage services charge based on how much memory is

used. Working with less data also means faster transfers over the Internet, lower latency and less risk for data loss, theft or corruption. All these considerations contribute to the overall performance of a Web service.

7.4 Possible improvements

Enhancing of my social networking application would start with moving more functionality to the backend and not relying so much on the frontend. I would have Lambdas call each other asynchronously instead of entrusting the UI to make all the necessary API calls. For instance, I would invoke the notification function, which is currently triggered by an independent API call, from the Lambdas responsible for creating posts, up-voting, commenting and adding members to groups. Another example is bundling the stale data deletion functionality in a separate, asynchronous Lambda and not involving the UI at all in this process. However, in order to implement these, I would require a lot more experience with AWS and Amplify.

Another improvement would be to use React in combination with Redux - a framework which simulates an easier-to-manage global state instead of individual component state. I would also work more on the design of the application, to make it more cost-efficient and reliable and easier to test. Looking at the functionalities, I would add more social media features, such as sharing posts among different groups, instant messaging and support for videos. Location-based services, such as finding nearby groups or searching groups by location, are also valid enhancements.

8 Conclusion

The problem addressed in this project is stale data on social networking sites. This data is represented by old posts and groups, which are no longer relevant to any context and just waste space on the storage unit of servers and, sometimes, create unnecessary traffic on the network. Moreover, some of this content may at times degrade the current professional image of individuals. Looking at some models of modern social networking platforms, I concluded that the best strategy to tackle this problem is to just destroy data as soon as it becomes stale.

To demonstrate this solution, I built a simple social networking application where content is shared only in groups with a clear end-date, specified when a group is created. Once groups expire, they and the content which was shared in them are permanently deleted from the Web without any user intervention, thus freeing up storage space in the database. The application was developed in the Serverless architecture on Amazon Web Services, where the accent is put on scalability and efficient use of computing resources. Based on my reflection on the usage of this system, I believe it solves the problem of stale data in a seamless manner, all

while being cost-efficient and user-friendly.

Future work includes the improvements described in the previous section, extensive testing and the development of a native application for Android and iOS.

References

Bergstorm, K. (2011), 'Don't feed the troll: Shutting down debate about community expectations on reddit.com', *First Monday* **16**. [Accessed April 2019].
URL: *https://journals.uic.edu/ojs/index.php/fm/article/view/3498/3029*

Boyd, D. M. & Ellison, N. B. (2007), 'Social network sites: Definition, history, and scholarship', *Journal of Computer-Mediated Communication* **13**, 210–230. [Accessed April 2019].
URL: *https://doi.org/10.1111/j.1083-6101.2007.00393.x*

Chang, S. E., Shen, W.-C. & Liu, A. Y. (2016), 'Why mobile users trust smartphone social networking services? a pls-sem approach', *Journal of Business Research* **69**, 4890–4895. [Accessed April 2019].
URL: *https://doi.org/10.1016/j.jbusres.2016.04.048*

Charteris, J., Gregory, S. & Masters, Y. (2014), 'Snapchat 'selfies': The case of disappearing data', *Rhetoric and Reality: Critical perspectives on educational technology* pp. 389–393. [Accessed April 2019].
URL: *https://www.researchgate.net/profile/Jennifer_Charteris/publication/277186349_Snapchat_'selfies'_The_case_of_disappearing_data/links/5564487808ae86c06b6987c3.pdf*

Ellison, N. B., Steinfield, C. & Lampe, C. (2007), 'The benefits of facebook "friends:" social capital and college students' use of online social network sites', *Journal of Computer-Mediated Communication* **12**. [Accessed April 2019].
URL: *https://doi.org/10.1111/j.1083-6101.2007.00367.x*

Halper, K. (2015), 'A brief history of people getting fired for social media stupidity', *Rolling Stone* . [Accessed April 2019].
URL: *https://www.rollingstone.com/culture/culture-lists/a-brief-history-of-people-getting-fired-for-social-media-stupidity-73456/*

Kietzmann, J. H., Hermkens, K., McCarthy, I. P. & Silvestre, B. S. (2011), 'Social media? get serious! understanding the functional building blocks of social media', *Business Horizons* **54**, 241–251. [Accessed April 2019].
URL: *http://summit.sfu.ca/system/files/iritems1/18103/2011_social_media_bh.pdf*

Lampinen, A., Tamminen, S. & Oulasvirta, A. (2009), 'All my people right here, right now: management of group co-presence on a social networking site', *Proceedings of the ACM 2009 international conference on Supporting group work* pp. 281–290. [Accessed April 2019].
URL: *https://dl.acm.org/citation.cfm?id=1531717*

Lynn, D. M. (1991), 'Scarcity effects on value: A quantitative review of the commodity theory literature', *Psychology & Marketing* **8**. [Accessed April 2019].

URL: *https://scholarship.sha.cornell.edu/cgi/viewcontent.cgi?referer=https://scholar.google.co.uk/&httpsredir=1&article=1181&context=articles*

Murugesan, S., Deshpande, Y., Hansen, S. & Ginige, A. (2001), 'Web engineering: a new discipline for development of web-based systems', *Web Engineering. Lecture Notes in Computer Science* **2016**, 3–13. [Accessed April 2019].
URL: *https://doi.org/10.1007/3-540-45144-7_2*

Norris, P. (2002), 'The bridging and bonding role of online communities', *The Harvard International Journal of Press/Politics* **7**. [Accessed April 2019].
URL: *https://www.researchgate.net/profile/Pippa_Norris/publication/240725744_The_Bridging_and_Bonding_Role_of_Online_Communities/links/569153d608aed0aed8148606.pdf*

Preston, J. (2011), 'Social media history becomes a new job hurdle', *The New York Times* . [Accessed April 2019].
URL: *https://www.nytimes.com/2011/07/21/technology/social-media-history-becomes-a-new-job-hurdle.html*

Putnam, R. D. (2000), *Bowling alone*, Simon & Schuster.

Sherman, E. (2013), '1 in 10 young job hunters rejected because of their social media', *America Online* . [Accessed April 2019].
URL: *https://www.aol.com/2013/06/04/applicants-rejected-social-media-on-device-research/*

Williams, D. (2006), 'On and off the 'net: Scales for social capital in an online era', *Journal of Computer-Mediated Communication* **11**, 593–628. [Accessed April 2019].
URL: *https://doi.org/10.1111/j.1083-6101.2006.00029.x*

A Estimated work schedule

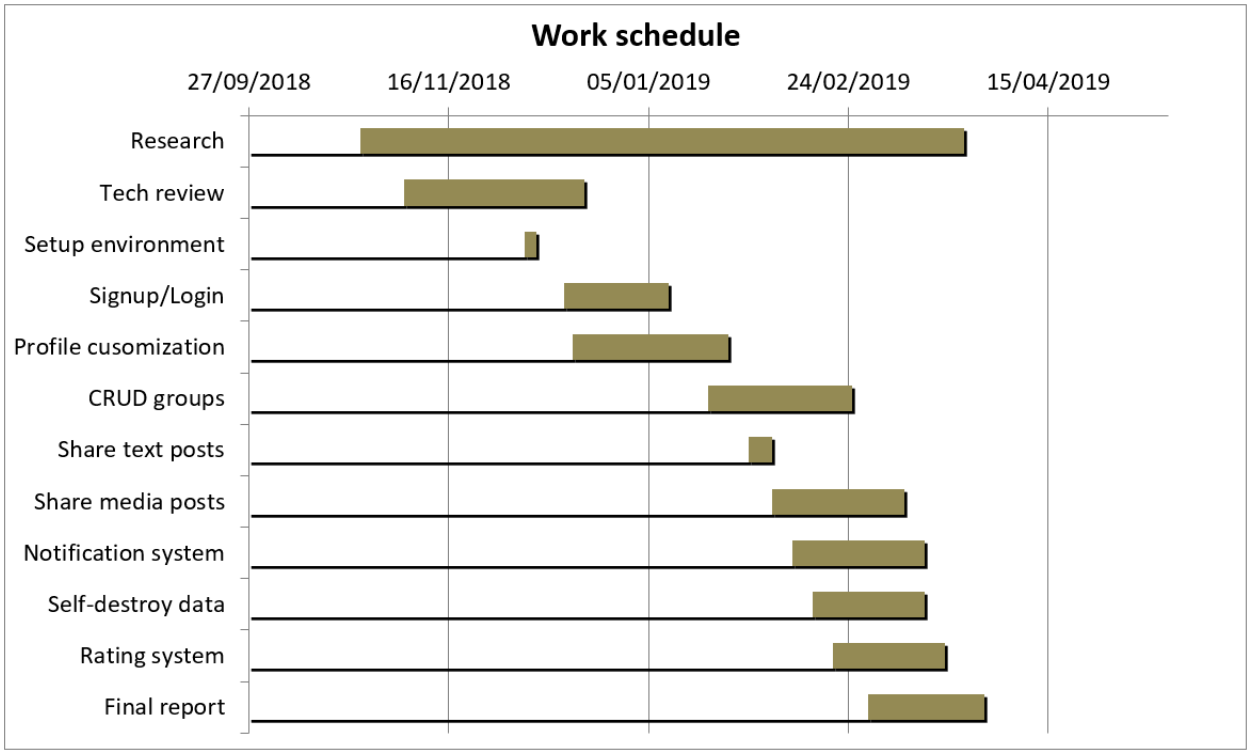


Figure 10: Work schedule from progress report

B Login

Access the app by clicking on this hyperlink. Use the username `test@soton.com` and password `Qwerty123!`. When asked to verify the email, click skip. Note, this account cannot demonstrate the email notifications. Alternatively, create and account with a valid email. When checking for emails, please look at the Spam folder too.

C Example test event

The following snippet is an example of a test event that is used to trigger a Lambda. It represents a request to retrieve the user data.

```
{
  "resource": "/profile",
  "path": "/profile",
  "httpMethod": "GET",
```

```
  "requestContext": {
    "identity": {
      "cognitoIdentityId":
        "eu-west-1:c6e3cffb-3572-419f-9431-fffa1e05c8d4"
    }
  }
}
```

D Contents of design archive

```
    .amplifyrc
    .gitignore
    package-lock.json
    package.json
    README.md

├──amplify
    ├──.config
            aws-info.json
            project-config.json

    └──backend
        |   amplify-meta.json
        |
        ├──api
        |   └──3YP
        |           3YP-cloudformation-template.json
        |           api-params.json
        |
        ├──auth
        |   └──cognito46197493
        |           cognito46197493-cloudformation-template.yml
        |           parameters.json
        |
        ├──awscloudformation
        |       nested-cloudformation-stack.yml
        |
        ├──function
        |   ├──3ypGroupsCRUD
        |   |   |   3ypGroupsCRUD-cloudformation-template.json
        |   |   |
        |   |   └──src
```

```
        │   │   │       app.js
        │   │   │       event.json
        │   │   │       index.js
        │   │   │       package-lock.json
        │   │   │       package.json
        │   │   │
        │   │   ├──3ypPostsCRUD
        │   │   │   │   3ypPostsCRUD-cloudformation-template.json
        │   │   │   │
        │   │   │   └──src
        │   │   │           app.js
        │   │   │           event.json
        │   │   │           index.js
        │   │   │           package-lock.json
        │   │   │           package.json
        │   │   │
        │   │   └──3ypUsersCRUD
        │   │       │   3ypUsersCRUD-cloudformation-template.json
        │   │       │
        │   │       └──src
        │   │               app.js
        │   │               event.json
        │   │               index.js
        │   │               package-lock.json
        │   │               package.json
        │   │
        │   ├──hosting
        │   │   └──S3AndCloudFront
        │   │           template.json
        │   │
        │   └──storage
        │       ├──3ypGroups
        │       │       3ypGroups-cloudformation-template.json
        │       │       parameters.json
        │       │
        │       ├──3ypPosts
        │       │       3ypPosts-cloudformation-template.json
        │       │       parameters.json
        │       │
        │       ├──3ypUsers
        │       │       3ypUsers-cloudformation-template.json
        │       │       parameters.json
        │       │
        │       └──s32feb684b
        │               parameters.json
```

```
                    s3-cloudformation-template.json

├──public
        favicon.ico
        index.html
        manifest.json
        user.png

└──src
    │   apiWrapper.js
    │   App.css
    │   App.js
    │   App.test.js
    │   aws-exports.js
    │   index.css
    │   index.js
    │   serviceWorker.js
    │
    └──components
            Contacts.js
            CreateGroup.js
            EditGroup.js
            GroupContent.js
            Groups.js
            Home.js
            Notifications.js
            Post.js
            Profile.js
            ProfileInfo.js
            SelectMembers.js
            Settings.js
            SingleGroup.js
```

To deploy the project locally, run `npm install` and then `npm start`.

E Initial project brief

CommonCrowd

Student: Bogdan-Andrei Chivescu (bac1g16, 28404467)
Supervisor: Professor Leslie Carr

Problem

The recent problem with social media platforms is privacy invasion, cyber security and data collection and sharing (or worse, selling). The news has been flooded with Zuckerberg's hearing and the whole scandal about Facebook's use of user data. A security beach in Google+ which dated in March 2018 was recently uncovered and led to the shutting down of Google+. These events not only create a gap in the social media environment, but also diminish the user's trust in these big names. However, what Google+ was trying to achieve with its Circles and Communities is still palpable. Some may say that Reddit is a valid alternative to the user who wants to share knowledge, get news updates and learn about their interests, all while interacting with other people. However, statistics show that more than 50% of Reddit users are in the US (content is biased or inapplicable/uninteresting for non-US) and almost 75% of all users are male (uneven distribution, making it less appealing for all genders). Furthermore, it is text-based, which makes it less accessible for inexperienced users or users who want a more visual, auditive or interactive experience.

Goals

The main goal of my project is to create a social media platform that is focused on groups of people rather than the individual. Its core use would be knowledge sharing, under any format (text, image, video, audio, other small files). Naturally, a group would have a topic or interest to determine the content and the members.

As a secondary goal, the platform should be visually engaging and intuitive, making it easy to use by unexperienced users and offering accustomed users a smooth sail. Another secondary goal is to make the application as lightweight as possible to keep it fast and secure and avoid a one-solution-to-all-problems situation.

A business goal is of course to generate revenue. Instead of running ads or marketing brands in any other way, the platform would manage a paid membership scheme to some groups, sharing the profits with the content creators.

Scope

- Ability to create and securely log into and out of an account
- A customizable, but limited user profile
- Options to create, search, join, request to join or leave groups
- Ability to create content in a group, in the form of writing or uploading files
- Ability to endorse, enrich and discuss the content
- No friend or follower relationship, just members of the same group
- No chatting functionality attached (keep the app lightweight plus this type of functionality can be outsourced if a good API is provided)
- No personal data needed (phone number, address, location)