1 Introduction

Progressive Web App (PWA) allows users to create and comment on both events and user stories. Social media features such as 'like', 'follow', 'interested', and 'going' were integrated. Users are able to tag an event with their stories, which will then appear in the 'Explore' page. Users can create a story by taking a picture with their front camera through the use of WebRTC or upload a picture locally. Each user story can receive likes and comments, for which the latter was implemented using Socket.IO [2, 10]. Socket.IO was also used for posting comments in the 'Discussion' of each event. Service worker was implemented to cache requests for offline usage. MongoDB was used to store and synchronise data between the client and server, while IndexedDB was used to store data loaded from MongoDB for offline usage. However, data stored in MongoDB can only be retrieved when user is online. The search function (via location) was implemented with Google Maps API, allowing autocomplete for the location field and ensuring that a valid location is used. For security purposes, users can only login with their respective Google Account.

2 Diagrams

Appendix **Figure 1** and Appendix **Figure 2** are the detailed description of the PWA system structure. **Figure 1** describes the interactions between the front-end, data storage, caching and data retrieval. Functionalities are limited and users are unable to have a personalised user experience if sign-in through Google Account is not completed. Data is retrieved from MongoDB and stored in IndexedDb when server connection is available. This enables user to access preloaded information when they are offline. Caching takes place with the use of service workers when users are online. **Figure 2** demonstrates the database model along with the relationship between documents. The types of data collected for each document and the functions used for data retrieval are also described.

3 Interface to Insert and Search Data via Forms

Challenges: Search speed must be efficient and search results must be accurate with respect to the given query. Users must be able to search for events, stories, and profile of other users. Basic text search will allow users to input any text query and return the results sorted by relevance. Users can use partial words to search for an event or profiles of other users. Advanced search will allow users to search for events based on event name, genres, venue (address) and date. When creating an event, Place Autocomplete by Google Maps API [5] is used to provide users with valid locations as options. **Solution:** FlexSearch.js [4] was chosen to implement basic text searching task due to its efficient, lightweight, and flexible properties [3]. It allows multi-field search and accepts multiple data format types as data index. Hence, it is used with both MongoDB and IndexedDB to provide searching functionality in both online and offline environment respectively. **Requirements:** Users should be able to search for events and user profiles while online and offline. Users should be able to obtain results through querying with partial information. The results returned are sorted by relevance. **Limitations:** On every page load, the data has to be retrieved from IndexedDB to populate the FlexSearch.js's search index. The performance loss might be negligible when the total data size is small, but it may cause longer loading time when the size of data exceeds a certain size. Google Maps API request could not be cached as it generates a random token for every request made, hence Place Autocomplete will not work offline.

4 Interface to Search Data via Map

Challenges: The use of map with accurate location where caching is able to take place. Custom marker containing the image and link to event is placed on the location of the event. **Solutions:** Map was implemented with Leaflet Geocoder, with markers displaying pop-up of image and link of ongoing and future events. Offline searches will display marker on cached location of event. **Requirements:** Allow users to search for events using a map in both online and offline environments. Past events should not appear on the map. Future and ongoing events should be displayed as custom markers on

the map. Custom markers should pop-up to display the event information when clicked. **Limitations:** The map takes time to load and does not generate immediately. Google Maps API was not chosen to be implemented in this task as request could not be cached due to the generation of random token for every request made, making it unavailable when user is offline. Caching the map tiles consumes more storage than other requests as map tiles are stored as images. If Place Autocomplete by Google Maps API [5] was not implemented, users would be able to insert invalid locations, causing markers of any previously created events with valid locations to not show up on the map. Hence, heavily relies on the input to be of valid locations.

5 PWA – Caching of the App Template Using a Web Worker

Challenges: Fetching cross-domain response will result in an opaque response, which will consume huge storage quota compared to normal responses [11]. Sending a fallback offline page response when user is offline and visit a page that does not exist in the cache. **Solutions:** 'Cache then network' was used and modified according to The Offline Cookbook [1]. **Requirements:** Users should be able to view a basic offline template with required data when page was not visited in the past and user is offline. Users should be able to view previously loaded events, stories, and profiles without the ability to make any changes. **Limitations:** Non-static files are not cached in the installation stage of the service worker. Page that has never been visited before will not be cached. This applies to events or stories which were updated, where instead of the updated information, the cached information, which might not be up-to-date, will be displayed when user is offline.

6 PWA: Caching Data Using IndexedDB

Challenges: Retrieve data from MongoDB and store them in IndexedDB. Display page content using data loaded from IndexedDB when no internet connection is present. **Solution:** Retrieve data from MongoDB using AJAX. Promise is used for both storing and loading of data from IndexedDB. **Requirements:** Data on events and user profiles must be stored locally for offline searching. Users should be able to access data retrieved from IndexedDB when they are offline. **Limitations:** A known bug of IndexedDB storage is usage increases with every *put*() operation [6, 7].

7 NodeJS Server Including Non-Blocking Organisation of Multiple Dedicated Servers

Challenges: Uses asynchronous functions to handle file upload, data retrieval from server, or other function calls that require time to complete. Errors might occur during the execution of asynchronous functions. **Solution:** Uses the Promise and async/await features of JavaScript to handle both successful and failed function operations. Keep the code nesting shallow to make code more maintainable and readable. **Requirements:** Able to use Promise to order the sequence of asynchronous processes. Able to use callbacks to handle the success and failure of asynchronous functions. **Limitations:** Single threaded mechanism, not suitable for CPU-intensive operations. Relies heavily on callbacks, which might possibly result in several callbacks nested within order callbacks.

8 MongoDB

Challenges: Creating schemas that clearly define the respective models. Verifying user inputs when creating a new document. **Solution:** Uses mongoose's validation middleware [9] to define the values allowed for an index. **Requirements:** Data stored in MongoDB must be synchronised with IndexedDB when the client has connection to the server. **Limitations:** Lacks flexibility as it does not support joins between multiple documents. Document size has a limit of 16MB, additional configurations were required for storing large images.

9 Quality of the Web Solution

Challenges: Implementation of an authentication system. Researching and implementing social features such as user profile, marking events as 'Interested' or 'Going'. Ensuring that the features implemented will work and do not cause system failure. **Solution:** Used passport-google-oauth20 [8] to implement the authentication system. AJAX requests are used for a non-page reload form submission. Socket.IO was used for live update of comments on stories and discussion posts on each event. Exhaustively testing the system to minimise the probability of bug occurrences. **Requirements:** Users should be able to select events to be tagged to their stories, 'like' and comment on stories, 'follow' other users, and click 'Interested' or 'Going' on events. Events which users attended are recorded in their profile. Events which were created by a user is set to be an event the organiser is 'Going' to by default. Through following others and posting themselves, users are able to view the activity of followed users and themselves, be it events or stories, in the 'Home' page. Users require a Google Account to login for security purposes. With the implementation of WebRTC, users are able to take pictures at events with a range of filters to choose from. Another extra feature is in the implementation of map search where users are able to view the event name as a link which would lead the user to the page displaying details of that specific event. Socket.IO is used to have a live update of contents. Hence, it was implemented for comments to be added to each posted story. It was also used for the creation of stories and posts for each event, allowing other users to view all activities related to an event, be it stories or just text. Users are able to post comments on events, have discussions with other users, all through live updates on story and post creation in the 'Discussion' section of each event. Having Google Maps API Place Autocomplete also helps users to quickly and accurately insert a valid location. Users are also able to list their favourite genres and write a bios to guide other users in finding those of similar interest, and possibly follow them. This in done to improve users' experience to have information of interest displayed on their 'Home' page upon login. **Limitations:** Users without a Google Account could not sign in into the system and experience personalised functionalities. Users who are not signed-in are able to view events, stories, and use the search functionalities, but are unable to show their interest through clicking on 'Interested' or 'Going', which would be displayed in their profiles. They would not be able to update any information, follow or unfollow other users, or like and comment on stories. As mentioned above, Google Maps API do not work offline. However, this is a minor limitation here since users are not able to make changes when they are offline. User accounts are all public, hence users should be careful when sharing personal information.

10 Conclusions

This assignment highlights the importance of caching and the importance of usability both during online and offline. It taught us to consider other implementations to improve user experience and web security. This assignment exposed us to multiple tools (Google Maps API for address searching, WebRTC for photo capturing, Socket.IO for live comment updates, Leaflet Geocoder for map searching with markers, etc.) while allowing us to implement the basic knowledge of web building at a higher level through the use of promises, service workers and memory caching.

11 Division of Work

The solution was designed jointly. **Zer Jun Eng** lead implementation of MongoDB, search by map, PWA caching, and front-end implementation while taking part in WebRTC, testing. **Lim Jia Mei Grace (Jia Lim)** lead implementation of WebRTC, testing, and documentation while taking part in the construction of MongoDB database, search by map and front-end implementation. The final document was jointly edited.

12 Extra Information

Run **npm run initdb** to populate the database with initial data, then run **npm start** to start localhost and MongoDB.

References

[1] J. Archibald, *The offline cookbook*. [Online]. Available: https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook/ (visited on 31/03/2019).

[2] F. Ciravegna, *Week 6 - mongodb and socket.io*, 2019.

[3] *Flexsearch benchmark comparison*, Nextapps GmbH. [Online]. Available: https://github.com/nextapps-de/flexsearch#benchmark-ranking (visited on 31/03/2019).

[4] *Flexsearch.js*, Nextapps GmbH. [Online]. Available: https://github.com/nextapps-de/flexsearch (visited on 31/03/2019).

[5] *Google places autocomplete api*, Google. [Online]. Available: https://developers.google.com/maps/documentation/javascript/examples/places-autocomplete (visited on 31/03/2019).

[6] *Google/leveldb github issues #593*. [Online]. Available: https://github.com/google/leveldb/issues/593 (visited on 24/03/2019).

[7] *Google/leveldb github issues #603*. [Online]. Available: https://github.com/google/leveldb/issues/603 (visited on 24/03/2019).

[8] J. Hanson, *Passport google oauth 2.0*. [Online]. Available: https://github.com/jaredhanson/passport-google-oauth2 (visited on 31/03/2019).

[9] *Mongoose 5.4.20 validation*, mongosoe. [Online]. Available: https://mongoosejs.com/docs/validation.html (visited on 31/03/2019).

[10] *Socket.io*. [Online]. Available: https://socket.io/ (visited on 31/03/2019).

[11] *Understanding storage quota*, Google. [Online]. Available: https://developers.google.com/web/tools/workbox/guides/storage-quota#beware_of_opaque_responses (visited on 31/03/2019).

A Appendix

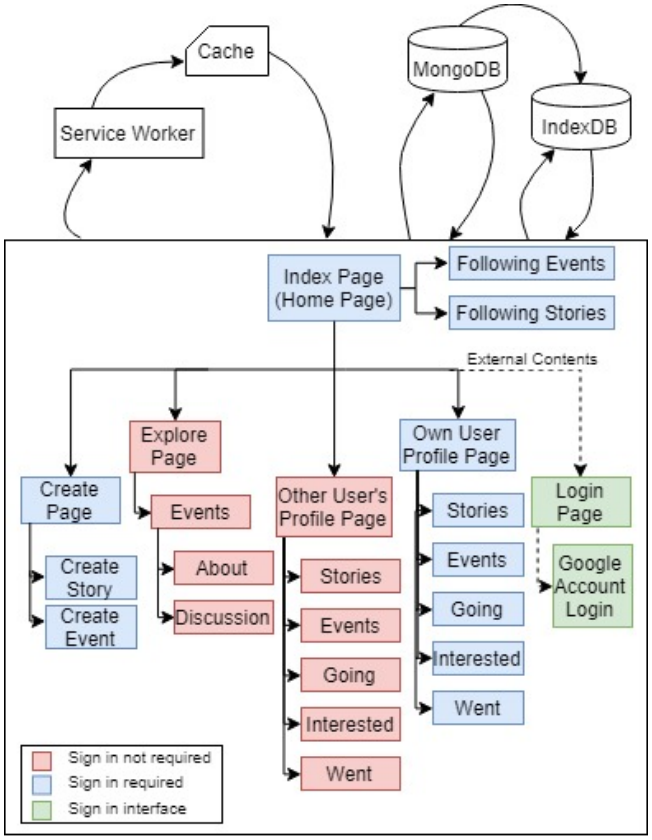

Figure 1: Demonstrates the flow of each web page in this PWA system along with the respective partial pages and external content pages.

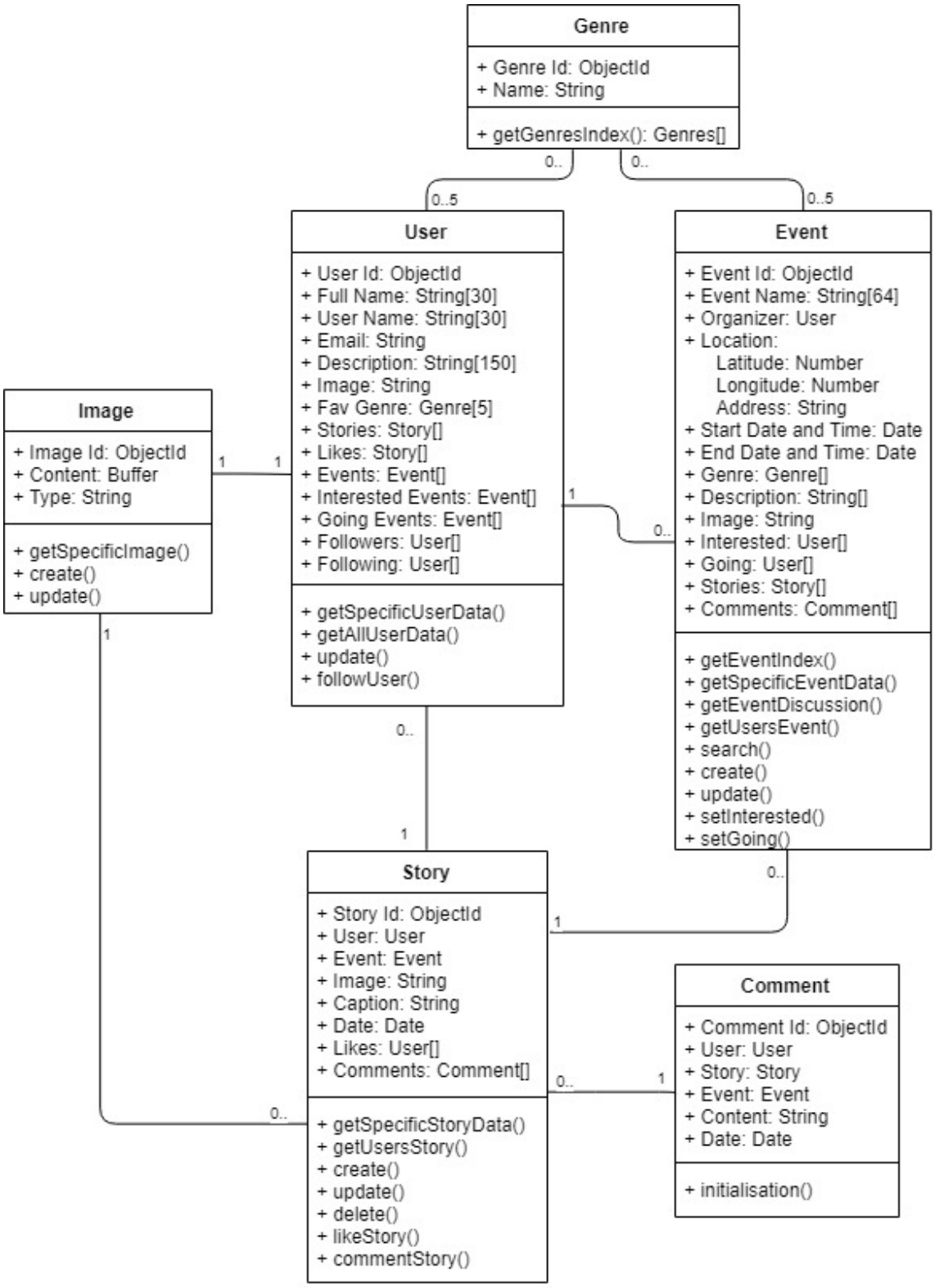

Figure 2: Displays the structure of the database along with the types of content stored.