1 Introduction

Progressive Web App (PWA) allows users to create and add comments to both events and user stories. Social media features such as 'like', 'follow', 'interested', and 'going' are integrated. Users are able to tag an event along with their stories, which will then appear in the event page. Users can create a story by taking a picture with their front camera through the use of 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, 8]. Service worker was implemented to cache requests for offline usage. MongoDB is used to store and synchronise data between the client and server, while IndexedDB is 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 auto complete for the location field and ensuring that it is a valid location. 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 with **Figure 1** describing the front-end, data storage, caching and data retrieval, while **Figure 2** demonstrates the database model along with the relationship between documents.

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 the user to input any text query and return the results sorted by relevance. Advanced search will allow the user to search for events of a specific genre or find events between two different dates. Address searching in the event creation page is done through Place Autocomplete by Google Maps API [3]. **Solution:** ChoicesJS was chosen to implement basic text searching task due to its efficient, lightweight, and flexible properties. It allows multi-field search and accepts multiple data format types for its data index, hence it could be used with both IndexedDB and MongoDB to provide searching functionality in both online and offline environment. **Requirements:** Users should be able to search for events, stories, and user profiles while online and offline. Users should be able to obtain results through querying with partial information. The results returned should be sorted according to relevance. **Limitations:** On every page load, the data has to be retrieved from IndexedDB to populate the 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.

4 Interface to Search Data via Map

Challenges: Allow users to search for ongoing or upcoming events via map, where 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 environment. 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, hence it will not work 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 [3] was not implemented, users would be able to insert invalid locations, hence causing markers of any previously created events with valid locations to not show up in the map. Hence, this feature highly replies on the input 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 [9]. 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 not connected to the internet. Users should be able to view previously loaded events and stories without the ability to make any changes to the events, stories, or their profiles. **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 the cached information, which might not be up-to-date, will be displayed when user is offline.

6 PWA: Caching Data Using IndexedDB

Challenges: Retrieving data from MongoDB and store them in IndexedDB. Display page content using data loaded from IndexedDB when no internet connection present. **Solution:** Retrieve the data from MongoDB using AJAX. Promise is used for both storing and loading of data from IndexedDB. **Requirements:** Events and user profile data 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 [4, 5].

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 functions call that requires time to complete. Errors might occurred 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 successful and failure events of asynchronous functions. **Limitations:** Single threaded mechanism, not suitable for CPU-intensive operations. Relies heaviliy 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 [7] 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 required for storing large images.

9 Quality of the Web Solution

Challenges: Implementation of an authentication system. 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 [6] to implement the authentication system. AJAX requests are used for a non-page reload form submittion. Exhaustively testing the system to minimise the probability of bug occurrences. **Requirements:** Users should be able to tag events in their stories, 'like' stories, 'follow' other users, click 'interested' or 'going' on events. Events which users attended are recorded in their profile. Users require a Google Account to login for security purposes. **Limitations:** Users without a Google Account could not sign in into the system and enjoy personalised content. 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] *Google places autocomplete api*, Google. [Online]. Available: `https://developers.google.com/maps/documentation/javascript/examples/places-autocomplete` (visited on 31/03/2019).

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

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

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

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

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

[9] *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

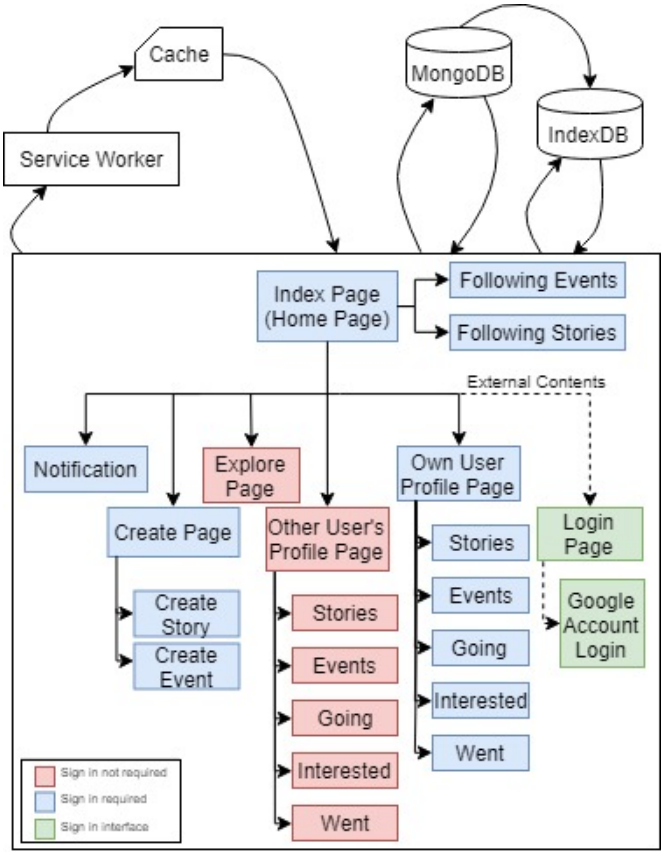

Figure 1: Demonstrates the flow of each web page in this PWA system along with the respective partial pages and external content pages.

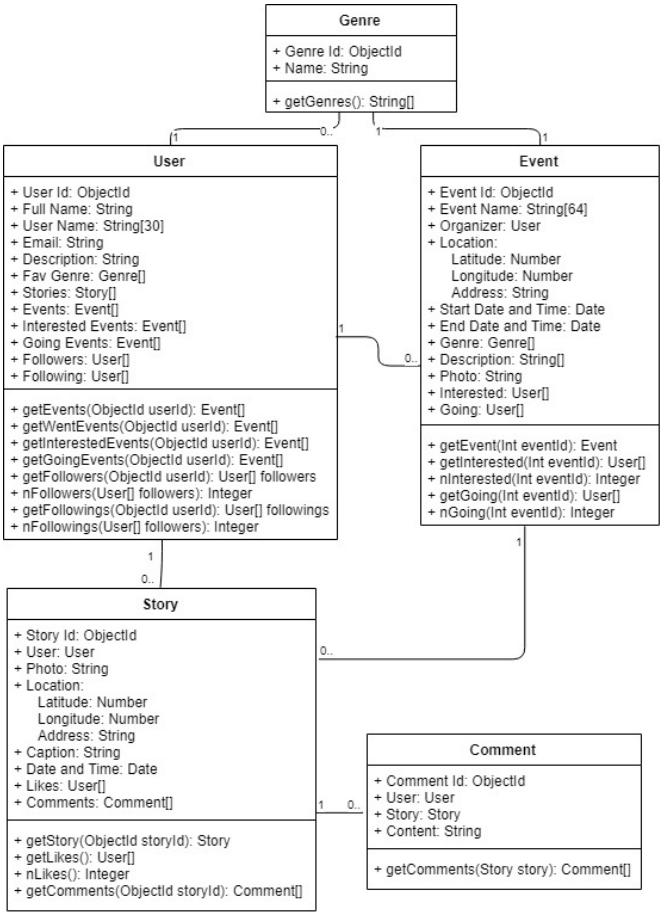

Figure 2: Displays the structure of the database along with the types of content stored.