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 or environment camera through the implementation of WebRTC or upload a picture locally. Each user story can receive likes and comments, for which the latter is implemented using Socket.IO [4, 1]. 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

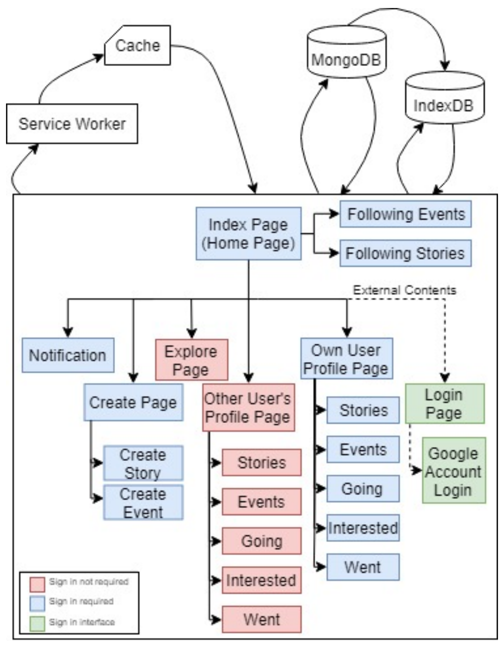

Figure 1: Demonstrates the flow of each web page in this PWA system along with the respective partial pages and external content pages.

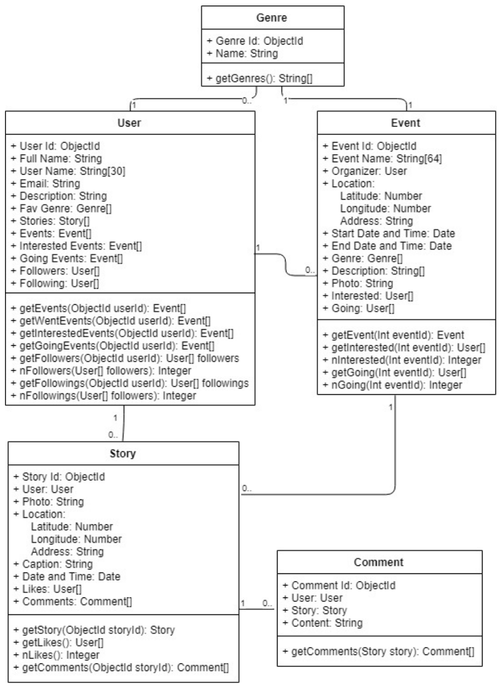

Figure 2: Displays the structure of the database along with the types of content stored.

Figure 1 and 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 describe the types of data stored in the database along with the relationship between documents.

3 Interface to Insert and Search Data via Forms (Yet to Implement)

- **Challenges**:
  - Usage of search libraries which would allow suggestions and auto complete.
  - ElasticSearch only allow online searching and would cause errors when users are offline.
- **Solution**: (Yet to Implement)
  - Online library that would support offline searching so that users can make use of this functionality both during online and offline.
- **Requirements**:
  - Users should be able to search for events both during online and offline.
  - Users should be able to search for events based on partial event names.

4 Interface to Search Data via Map (Yet to Implement)

- **Challenges**:
  - Implementation of Google API could not be cached, therefore could not be used offline.
- **Requirements**:
  - Allow users to search for events' location using map.
  - Users should be able to use this search both offline and online.

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

- **Challenges**:
  - Image of events are not cached on initial load.
  - Unable to cache user page through the use of service worker.
- **Solution**:
  - Alterations in service worker was conducted for imaged to load immediately and to cache user profile data which is stored as *json*.
- **Requirements**:
  - Users should be able to view a basic offline template with required data for when offline.
  - 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**:
  - Users are only able to view a basic offline template if no data were loaded in the past when the user was online.

6 PWA: Caching Data Using IndexedDB

- **Challenges**:
  - The usage of IndexedDB increases with every *put*() operation.
- **Solution**:
  - There is no known solution at the moment, please refer to [2, 3] for more information.
- **Requirements**:
  - Optimise IndexedDB to cache data to allow offline usage with limited data available.
  - Users should be able to access essential data when offline for the application to serve its purpose.
- **Limitations**:
  - IndexedDB is stored locally and could not be updated until user is online.
  - Disk Storage still increases with every *put*() operation

7 NodeJS Server Including Non-Blocking Organisation of Multiple Dedicated Servers (Yet to Implement)

- **Requirements**:
  - The application should be able to connect to MongoDB server without explicitly starting MongoDB server before starting the application.

8 MongoDB

- **Challenges**:
  - Loading of initial data to populate the database.
  - The retrieval and storage of images in the form of bits.
- **Solution**:
  - Initial population of database was resolved with the use of promises.
  - Image retrieval and storage was implemented using *url* format.

- **Requirements**:
  - Data stored in MongoDB is stored online. Data is not stored locally and cannot be retrieved offline. When user is online, data is retrieved from the database and displayed in the PWA application.
- **Limitations**:
  - Initialisation of database needs to be run separately (refer to 'Extra Information').
  - Data stored on MongoDB can only be accessed when user is online.

9 Quality of the Web Solution

- **Challenges**:
  - Implementation of a secure login system.
- **Solution**:
  - Using Google Account to conduct login, at which users should use their existing Google credentials to access the application.
- **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 use the system even though it is an unlikely case.
  - User accounts are all public, hence users should be careful when sharing personal information.

10 Conclusions (To date of first submission)

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 (like Google Maps API for address searching, WebRTC for photo capturing, 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 (To date of first submission)

The division of workload was lead by team leader, Eng Zer Jun. The solution was designed jointly and then each member lead the implementation of one specific part of the code. In particular:

- **Eng Zer Jun**: lead the implementation of all PWA caching, MongoDB, and front-end implementation
- **Lim Jia Mei Grace**: lead the report documentation, implementation of WebRTC, assisted in the construction of MongoDB database and front-end implementation.

12 Extra Information

- **System initialisation**: run 'npm install' prior to all of the following.
- **Connect to MongoDB**: (temporary) run 'mongod' to start serving MongoDB.
- **Initial population of MongoDB**: run 'npm run initdb' to drop database and repopulate database with default data.
- **Running the program**: run 'npm start' to start serving localhost.

References

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

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

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

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