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 [3, 14]. 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 provided as an option to users. 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: 1) Search speed must be efficient and search results must be accurate with respect to the given query. **2**) Users must be able to search for events, stories, and profile of other users. **3**) Basic text search should allow users to input any text query and return the results sorted by relevance. **4**) All search functions should work for both offline and online usage. **Solution: 1**) FlexSearch.js [5] was chosen to implement basic text searching task due to its efficient, lightweight, and flexible properties [4]. 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. This was implemented in search function for both events and user profiles, where when users provide partial information, the display of search results sorted by relevance (most relevant on top) in a drop-down menu is returned. **2**) Users are able to search for events via event name, location, genre, and date of event. Users can also search for stories via event name, location, and date posted. This allows users to post stories related to an event from another location (not restricted to the event location) and they are able to post throwbacks (not restricted to date and time of event occurrence). **3**) The location input utilise Google Place Autocomplete plugin [6], which allows the user to use a precise existing location as input. **4**) Given a date, it will return all events which has that specific date as its start or end date, or within the start and end date (event which is more than a single day) **5**) When online, the search request is processed by server-side NodeJS code. If the search parameters are valid, then NodeJS will return status code 200 along with the search results fetched from MongoDB in JSON format, otherwise it will return status code 500; When offline, the search request is processed by client-side JavaScript code. If the search parameters are valid, then the search results are processed and fetched from IndexedDB, otherwise an error notification will be shown and the search request will not be processed. **Requirements: 1**) When searching for events, given a name, location, or date by the users, the search function should return all events that match the given search criteria only. **2**) When searching for stories, given an event name, location, or date by the users, the search function should return all stories that match the given search criteria

only. **3**) Users should be able to search for events, stories, and user profiles while online and offline. **4**) Search results should be fetched from MongoDB when user is online, while search results should be fetched from IndexedDB when user is offline. **5**) Users should be able to obtain results through querying with partial information. The results returned are sorted by relevance (most relevant on top). **6**) All requirements are met. **Limitations: 1**) On every page load, the data has to be retrieved from IndexedDB to populate the FlexSearch.js's search index. Performance loss may be negligible when total data size is small, but this may cause longer loading time when data exceeds a certain size. **2**) Google Maps API request could not be cached as it generates a random token for every request made, which indicates that Place Autocomplete will not work offline. Therefore, the search query has to be more specific to retrieve precise results.

4 Interface to Search Data via Map

Challenges: 1) The use of map with accurate location where caching is able to take place. **2**) Plotting events and stories as custom markers on an interactive map. **3**) If Place Autocomplete by Google Maps API [6] 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. **Solutions: 1**) Map was implemented with Leaflet [10], and geocoding was implemented with Leaflet Control Geocoder [11]. **2**) In the implementation of searching for events, markers are placed to display pop-up of image and link of upcoming events upon clicking, while in the implementation of searching for stories, markers are placed to display pop-up of image, username and caption of stories upon clicking. Offline searches will perform the same as online functionalities if they were cached prior to lost of internet connection. **3**) To mitigate stacking of multiple markers (multimarker), Leaflet Marker Cluster [12] was used. This would result in number of markers of each cluster to be displayed. Upon clicking, the area of interest will me zoomed in and each marker would be displayed. **4**) Valid locations entered by users (selected from Place Autocomplete by Google Maps API [6]) are stored as longitude and latitude to be used to plot markers, while invalid locations (not found in Google Maps API) are not plotted. Google Maps API covers a large range of locations and if it is not found, there is a large possibility that the location entered is either invalid or made-up by the user, hence deemed impractical to plot as marker, though it is still deemed practical to be used as an event venue upon creation. **Requirements: 1**) Allow users to search for events using a map in both online and offline environments. **2**) Past events should not appear on the map. **3**) Future and ongoing events should be displayed as custom markers on the map. **4**) Events and stories should be plotted on a map as custom marker. **5**) Custom markers should pop-up when clicked to display the event information when clicked. **6**) All requirements are met. **Limitations: 1**) The map takes time to load and does not generate immediately (latency). **2**) 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. **3**) The implementation of markers heavily relies on the input to be of valid locations, where invalid locations will not appear as markers. **4**) When there exist multiple markers, E.g. 20 or more, on the exact same location, stacking of markers will occur, leading to confusion and poor user experience. **5**) Markers are loaded all at once regardless of whether or not the marker is within the frame of screen. This could be handled with the implementation of lazy loading, but is not implemented in the current site due to time constraint.

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

Challenges: 1) Fetching cross-domain response will result in an opaque response, which will consume large amount of storage quota as compared to normal responses [15] due to unnecessary overhead. **2**) When invalid or un-cached URL path is requested, a fallback response should take place to prevent unplausible contents (E.g. Error displaying or loading image due to broken or unknown path) from displaying on the application. **Solutions: 1**) 'Cache then network' was used and modified according to The Offline Cookbook [1]. The service worker will make two requests, one to the cache and one to the network. If the network request is successful, then the service worker will update the cache and return

the network response, otherwise, it will return the cached response. **2**) In offline environment, when the user is visiting a page that has never been cached before, the service worker will send a fallback offline template response to prevent the web browser from displaying the default 'No internet connection' page (i.e. Chrome dinosaur). **Requirements: 1**) Users would have a basic offline template displayed if page was not visited in the past and user is offline. **2**) Users would be able to view cached data if the page was visited prior to offline usage. However, the displayed data might not be up-to-date. **3**) Users should be able to view previously loaded events, stories, and profiles without the ability to make any changes. **4**) All requirements are met. **Limitations: 1**) Non-static files are not cached in the installation stage of the service worker. **2**) Page that has never been visited in the past are not cached. This applies to events and 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. **3**) Service worker may not work in older versions of a browser.

6 PWA: Caching Data Using IndexedDB

Challenges: 1) Maintaining the same object structure for both IndexedDB and MongoDB, so that data could be retrieved from MongoDB when connection is present, data can then be retrieved from IndexedDB when internet connection is not present. **2**) Fetching the data from MongoDB and storing them into IndexedDB using Ajax and Promises. **Solution: 1**) When the user is online, the application will automatically retrieve data from MongoDB through the use of Ajax and store the retrieved data into IndexedDB. **2**) The GET request is performed over Ajax calls. Promises are used for both storing and loading of data from IndexedDB. **3**) In offline environment, Ajax GET and POST request will always return an error. As a result, the required information is fetched from previously stored data (cache) in IndexedDB. **Requirements: 1**) Sync and update the local IndexedDB with the latest data fetched from remote NodeJS server when online. **2**) All data should be stored locally to allow offline searching of events, stories, and user profiles. **3**) Users should be able to access data retrieved from IndexedDB when they are offline. **4**) All requirements are met. **Limitations: 1**) A known bug of IndexedDB storage is that usage increases with every *put*() operation [7, 8]. **2**) Maximum browser storage is dynamic, which is based on the size of user's hard drive. In the unlikely case where the IndexedDB has reached its storage limit, an error will be thrown which may cause adverse effect [2].

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

Challenges: 1) 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: 1**) 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: 1**) Able to use Promise to order the sequence of asynchronous processes. Able to use callbacks to handle the success and failure of asynchronous functions. **2**) All requirements are met. **Limitations: 1**) Single threaded mechanism, not suitable for CPU-intensive operations. **2**) Asynchronous nature. Extensive use of Promise and callback functions to handle success and error events, which might result in heavily nested code.

8 MongoDB

Challenges: 1) Creating schemas that clearly define the respective models. **2**) Verifying format and correctness of user inputs when creating a new document. **Solution: 1**) Design UML diagram prior to creation of database as shown in Appendix **Figure 2**. **2**) Use of mongoose's validation middleware [13] to define and check type and format of values allowed for an index. **Requirements: 1**) Data stored in MongoDB must be synchronised with IndexedDB when the client has connection to the server. **2**) All requirements are met. **Limitations: 1**) Lacks flexibility as it does not support joins between multiple documents. **2**) Document size has a limit of 16MB, additional configurations were

required for storing large images.

9 Quality of the Web Solution

Challenges: 1) Implementation of an authentication system. **2**) Researching and implementing social features such as user profile, marking events as 'Interested' or 'Going'. **3**) Ensuring that the features implemented will work and do not cause system failure or errors. **Solution: 1**) Used passport--google-oauth20 [9] to implement the authentication system. **2**) 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. **3**) Exhaustively testing the system to minimise the probability of bug occurrences. **4**) Events which users attended are recorded in their profile. The organiser (creator) of an event is by default set as 'Going' to that event. However, delete function is not available as in the case where events are cancelled, it should be announced in the 'Discussion' of the event. Merely deleting an event would cause confusion for users. **5**) 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. **6**) Users require a Google Account to login for security purposes. **7**) With the implementation of WebRTC, users are able to take pictures at events with a range of filters to choose from. **8**) 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. **9**) 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. **10**) Having Google Maps API Place Autocomplete also helps users to quickly and accurately insert a valid location. **11**) Users are also able to list their favourite genres and write a bio 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. **Requirements: 1**) Users should be able to experience basic social media functionalities after logging in, and only searching functions otherwise. **2**) Users should be able to select events to be tagged to their stories, check-in through posting of stories, 'like' and comment on stories, 'follow' other users, and show their interests by clicking on 'Interested' or 'Going' on events. **3**) Users should be able to post announcements or discussions on events. **4**) Users should be able to view posts and events of interest through following other users. **5**) All requirements are met. **Limitations: 1**) 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 are not be able to update any information, follow or unfollow other users, or like and comment on stories. Creating events and stories are also unavailable without signing-in. **2**) 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. **3**) User accounts are all public, hence, users should be careful when sharing personal information. **4**) In the current site, lazy loading is not implemented, hence, when multiple comments, stories, events, or discussion posts are created, they will all appear in the respective pages, resulting in a cluttered interface. **5**) The 'Discussion' section of each event could not be edited or deleted, and is not limited by time, at which users can continue to create discussion posts even after the event has ended. This has its pros and cons at which users can still ask about potential future events or provide feedback, but would also clutter the interface. **6**) Filters are provided in the implementation of WebRTC but users are unable to change the degree of each filter as they please. **7**) Due to time constraint, users are able to view number of followers, their followed users, number of likes on their created stories, but cannot view the names of users involved in all of the stated cases. **8**) External sharing of events and stories are not made available as well. **9**) Due to time constraint, 'Notification' page has also been removed from the initial design of the site. However, it would be useful to have this function to remind users of upcoming events near

them, upcoming events which user showed interest in attending, and to notify users of events which many from the list of their following users are attending. **10**) This site does not allow the sending of invites by organiser. **11**) There cannot be the case where multiple organisers share the same rights to make changes to the event. It is assumed that an organising body (non-individual) is to have a Google Account of its own. **12**) Due to time constraint, events created could not be set to private or public. **13**) Markers on map does not differenciate an ongoing event from a future event. This can be done with colour difference on custom markers but was not implemented due to time constraint. **14**) A useful feature to have is to suggest events to users. Events can be sorted by popularity, at which there are a large number of people 'Interested' and 'Going' to the event, or suggestions based on the interest of user and that of their followed users.

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

All the members of the group contributed equally to the assignment solution. 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. Go to `https://localhost:3000` to visit the home page.

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] *Browser storage limits and eviction criteria*, mozilla. [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Browser_storage_limits_and_eviction_criteria` (visited on 17/05/2019).

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

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

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

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

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

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

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

[10] *Leaflet - javascript library for interactive maps*, Leaflet. [Online]. Available: `https://leafletjs.com/` (visited on 17/05/2019).

[11] P. Liedman, *Leaflet control geocoder*. [Online]. Available: `https://github.com/perliedman/leaflet-control-geocoder` (visited on 17/05/2019).

[12] *Marker clustering plugin for leaflet*, Leaflet. [Online]. Available: `https://github.com/Leaflet/Leaflet.markercluster` (visited on 17/05/2019).

[13] *Mongoose 5.5.9 validation*, mongoose. [Online]. Available: `https://mongoosejs.com/docs/validation.html` (visited on 16/05/2019).

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

[15] *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

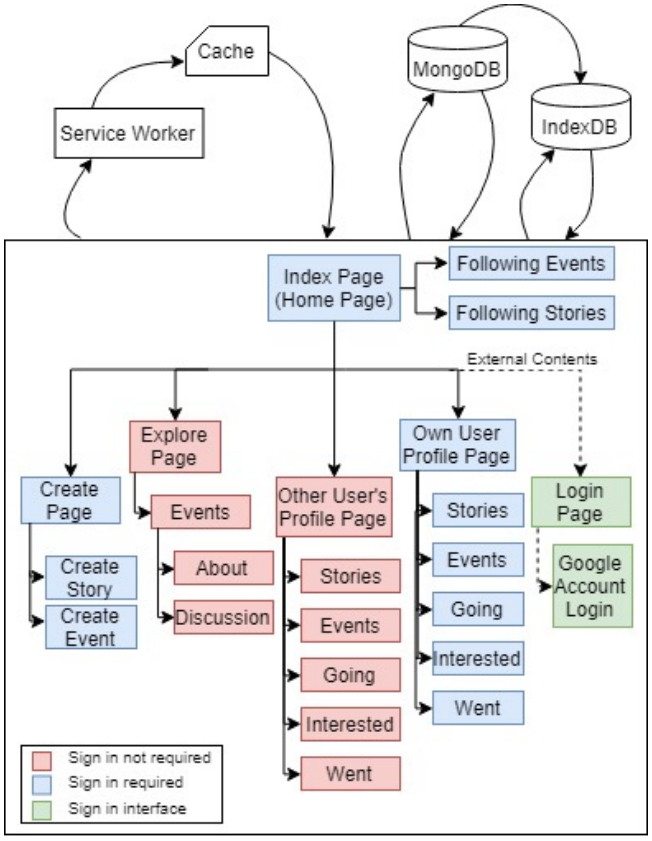

Figure 1: Demonstrates the flow of each web page in this PWA system along with the respective partial pages and external content pages.

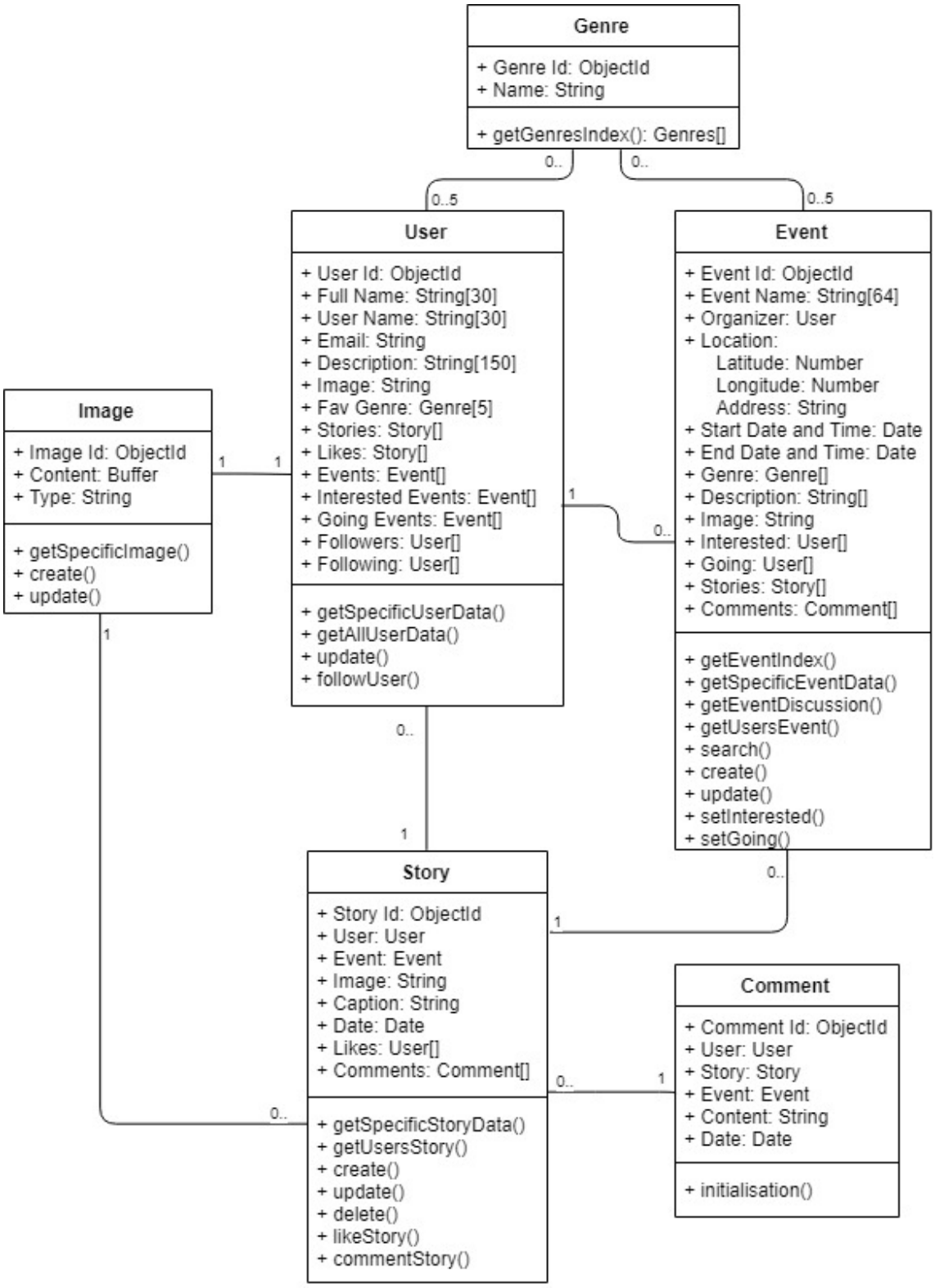

Figure 2: Displays the structure of the database along with the types of content stored.