Wayfinding in Highland Park Memorials

by

Balaji Jagadeesan

Project submitted in partial fulfillment of the requirements for the degree of Master of Science in Information Sciences and Technologies

Rochester Institute of Technology

B. Thomas Golisano College of Computing and Information Sciences

Department of Information Sciences and Technologies

June 04, 2018

Rochester Institute of Technology

B. Thomas Golisano College
of
Computing and Information Science

Master of Science in Information Sciences and Technologies

Project Approval Form

Student Name: Balaji Jagadeesan

Project Title: Wayfinding in Highland Park Memorials

Project Area(s): (√ primary area)

☑ Application Dev.	☐ Database	☑ Website Dev.
☐ Game Design	☐ HCI	☐ eLearning
☐ Networking	☐ Project Mngt.	☐ Software Dev.
☐ Multimedia	☐ System Admin.	☐ Informatics
☐ Geospatial	☐ Other _____	

Project Committee

Name	Signature	Date
Dr. Deborah Labelle Chair		
Prof. Bryan French Committee Member		

Table of Contents

List of Figures

List of Tables

1 Abstract

Trends in contributing trees and benches to parks in memory (Cloke & Pawson, 2008) of a departed soul are on the rise as they are affordable and environmentally friendly. Proper records on these memorial trees and benches are often not readily accessible and are forgotten in time. In addition to that, the exact location of the memorials is not always known. Sophisticated technology like digital wayfinding to track these features are not available especially for local parks like Highland Memorial Park, which is run predominantly by volunteers. The focus of this project is to build a scalable server-client system that will identify the location of these memorials and provide directions based on the user location. The first half of this paper presents the data collection and development of *GraphQL* server with Apollo framework along with the implementation of custom authentication using JWT. The second half of this paper discusses the development of mobile client using *React-Native*, *Apollo client*, *React-Native Maps* and development of administrative console using *React*. The mobile client has only minimal features and a few known bugs due to the inherent problem with the base library. The administrative console provides a basic interface to view edits, verify entry and accept new administrators. The project has a huge scope of turning into a robust open-source wayfinding application to identify the memorials as the react native and supporting libraries mature.

2 Introduction

Due to the sophistication of modern architectural design, the infrastructure of rural and urban settings evolved. The designers started improving the design patterns, incorporating the concept of "legibility" and "orientation" (Dogu & Erkip, 2000). We need to understand these two concepts before discussing way-finding. Legibility is the inherent information that is present in the structure or little quirks in certain spaces like a fountain in a mall which acts as a central hub that helps in extracting relevant information about navigation. Orientation, on the other hand, is the sense of direction. At any given place inside the structure, a person should able to answer "Where am I?" and "Which direction am I facing?" in accordance with the surrounding space.

Wayfinding, in general, is the different way of orienting oneself in the spatial direction to navigate from one place to another. Although wayfinding is an inherent ability of a living being, there is a possibility of getting "lost" in an environment because of the inability to recognize the immediate surroundings. In order to avoid these circumstances, human beings developed unique procedures to aid in their travel. Some examples include ancient Polynesians (Passini, 1981) using celestial navigation, bird observation, and studying the nature of waves for navigating the sea.

During the 15th to 17th century dubbed "the age of exploration" (Head & Isom, 2010) saw the advancement in tools in aiding wayfinding. Tools like cross-staff (used to measure the altitude of heavenly bodies) and improvements in cartography were astonishing, that it eventually led to many more advancement in navigation in the late 1900's. Modern-day way-finding is becoming so sophisticated that they use 3D visual mapping, digital directional sign boards, GPS enabled location finding, hand-held devices containing navigational maps and digital compass to help people find their way in a physical space.

Though wayfinding concepts are implemented during the design phase of the building, digital map of the infrastructure in the form of interactive information screen complement the design. These digital technologies are being made mandatory in hospitals, movie theaters, and malls to help people learn as well as to navigate around the infrastructure. These interactive maps are also available for some national parks like the Harper's Ferry National Historical Park by using NPMap - a digital map application created by the national park services (Services, 2016). Although these kind of services are available, they do not transition to local parks due to lack of government funding and often these parks are maintained by the local community. Another drawback is that this technology gives only high-level information about the park which in most cases is good enough for general parks. But local parks like Highland Memorial Park needs special attention.

Memorial parks contain a collection of memorials, erected in memory of someone close to our heart, or in remembrance of their heroic deeds and bravery which portrays the profound impact that they had in our life. These memorials can be classified as upright memorials, slant memorials, flat headstone, trees, benches or small gardens dedicated in memory of an individual. These memorials are often forgotten in the course of time or not maintained properly. These factors may lead to inconsistent records of intangible information which often results in the destruction of memorials, one such incident happened recently on July 20, 2017, where a memorial tree dedicated to 9/11 victim was cut down mistakenly during the renovation of a park (Baker, 2017).

Although information about funeral memorials are available to the public through their funeral website, information about the location of the memorial trees, memorial benches, and small gardens are not available digitally and are often recorded in physical form and/or accessible to the officials working towards the maintenance of that memorial. This information whilst accessible can be cumbersome at sometimes. The general public, wishing to pay a visit to

these memorials often spend a lot of time at the information center gathering the information about the location of the memorial. Sometimes this information can be vague and there is a possibility to navigate to an incorrect location.

The primary objective of the project is to collect and collate the data about memorials and people's information, from the official records (both physical and digital) and implement wayfinding features through Google maps on the client to direct the user to the memorials in the Highland Memorial Park.

3 Problem

3.1 Problem Statement

Information about the location of memorial trees, benches, small gardens and details about the deceased whilst claim to be publicly available are often cumbersome to access. Due to lack of digital services tailored to localized information needs, finding a memorial corresponding to a particular individual can be tedious and time-consuming. The focus is to solve this problem at Highland Memorial park in Rochester, New York where people spend a lot of time at the park office to get information about memorial trees and benches, as there is no digital map of these details available for the public use.

3.2 History of Highland Memorial Park

The origin of Highland Memorial Park is deeply rooted in the efforts of two horticulturists George Ellwanger, Patrick Barry, and designer Fredrick Law Olmstead (Peck, 1908). George Ellwanger emigrated from Wurttemberg, Germany to America and worked as an apprentice in a horticulturist firm in Stuttgart from 1831 to 1835 and then worked with William Reynolds and Michael Bateham of the Seed Store in Rochester from 1836 to 1838, who finally ended up starting a nursery business with Thomas Rogers, a mulberry tree salesman (McKelvey, 1940).

Patrick Barry emigrated from Ireland to America in 1836 and was employed as a nurseryman at Prince Nursery in Flushing, Long Island (Meehan, 1907).

George Ellwanger along with Patrick Barry established Ellwanger & Barry Nursery Co., in 1840 because Ellwanger's previous venture with Thomas Rogers was a failure (McKelvey, 1940). Ellwanger & Barry Nursery Co., business boomed and they relocated to a 7-acre plot in Mt. Hope Avenue which later grew to 500 acres by 1859. In 1856, the partners ventured into real estate business and started building houses in cypress street and developing Maplewood district. During Ellwanger & Barry Nursery Co.'s business peak, their market reached to California and as far as Japan, Korea, and Australia. The cultivation of nurseries spanned across 650 acres (McKelvey, 1940).

During their era, they contributed a lot to the city of Rochester and one such contribution is a 20-acre plot of land next to the foothill of the city (near Mt. Hope Avenue) in 1888. At that time, it served as the public area which eventually became Highland Park (Peck, 1908). This 20 acre of land instigated the formation of Parks Department in Rochester. The landscaping started in 1892 with Frederick Law Olmsted overseeing the park's design. Olmsted designed the park to have a natural look and planted 20 lilacs initially. Now the Highland Park is one of the most beautiful parks with over 500 varieties of lilacs and plants like azaleas and cherry blossoms. Through the course of time, the Highland Park expanded to accommodate memorials for Vietnam War veterans, a memorial for AIDS victims and memorial for worker's rights activist and victims of violent crimes.

3.3 Importance of Memorial Trees and Benches

Memorial trees and benches are used as a tool that transcends through time carrying the memories and the impact a person has made in one's life. According to (Cloke & Pawson, 2008), trees are considered for memorial purposes for the following reasons,

- Trees carry symbolic value. Willow, Cypress trees in Christian culture symbolizes grief and morality (Fussell, 2009).
- Trees can reflect the character of a person.

- Trees can grow through time, evolving and transforming to make an everlasting impact on the community.

Memorial Benches can be customized to reflect a person's character and can be placed in a location that the deceased person loved to visit or spend time. They are also portable and can be switched from place to place with low maintenance.

3.4 Significance of Project

The memorials, in general, are rich in history and memorial parks are no exception. Each memorial sings a rich heritage and history of the event or the contribution of the people. A lot of history can be found in those places and a lot can be learned from them but only a handful of information are present on-site. Despite their claim to be publicly available, most of them are not recorded accurately or contain just vague directional instructions.

The memorials can also be scattered throughout the park and it is often hard to find their physical location with a help of a conventional paper map of the park or directional instruction. Even though, if we are able to find them - information like the background details about the person or memorial, that person's service to the community or their history in the military if the memorial is dedicated to a veteran- may not be fully available. Although a quick Google search gives you the details about the person, there might be some conflicting information.

3.5 Project Goals

- Collecting and collating the data regarding the memorial and person information, from the valid resources like park administration.
- Implementing a scalable API server to serve the collected resource.
- Developing a mobile client to serve as an open source wayfinding platform to help the people navigate around the park and to find the memorials nearby.

- Building a simple administration console to manage and accept new moderators, verify content and make changes to the entries in the database.

4 Prior Work

Wayfinding tools like interactive maps, virtual tour are available to national parks. The local parks lack these features due to reduced funding from the government (Barrett, Pitas, & Mowen, 2017) and rely on volunteers to maintain the park through organizations like NRPA (National Recreation and Park Association) and volunteer program.

There are businesses that provide wayfinding tools like the creation of a virtual map for the park and help in tracking memorial trees and monuments. One such organization is web cemeteries (WebCemeteries: Memorial Solutions, 2018) where they provide paid services like map creation and memorial tracking in a park. They also offer a mobile application to track the memorials, tools to create online memorials. But local community driven park cannot afford such service as there are costs involved in maintenance. So open source wayfinding tools can be used for such scenarios.

Way marking (Waymarking, 2018) is a community run website that tracks places that are unique and interesting. In addition to monitoring all interesting spots, they manage places like monuments, memorial trees, and benches along with little details. Despite the fact that website track the memorials, there are not enough contribution by the community as these websites are not famous among the local community.

Other organizations tackle the problem at small scale by providing wayfinding tools on request by an acquaintance of deceased person. Qeepr (Qeepr: Free Online Memorial Sites & Tributes, 2018) is an online memorial website to remember the departed. Relatives of the deceased person can create a social profile enlisting the biography of the departed person. They can share memento, photos, personal notes and memories that they had with the

deceased person. It follows a similar trait of any social profile. Qeepr also introduced mobile application to geotag the deceased final burial spot. Using the mobile application, one can get directions to the burial site. So there is a need for an open-source platform that should be localized to a particular area that is contributed by the community, with ease of accessibility to track the memorials.

5 Solution Approach

The development of the application for the Highland Memorial Park is split into three stages. The first stage involves data collection, where memorial data is collected and a data model is built based on the data gathered. The second stage involves server development, where the business logic, authentication and GraphQL layer for the database is implemented. The third stage involves the development of a mobile client and a basic administrative console to manage the newly added data. The high-level architecture diagram of the entire stack is shown below in *Figure 1*.

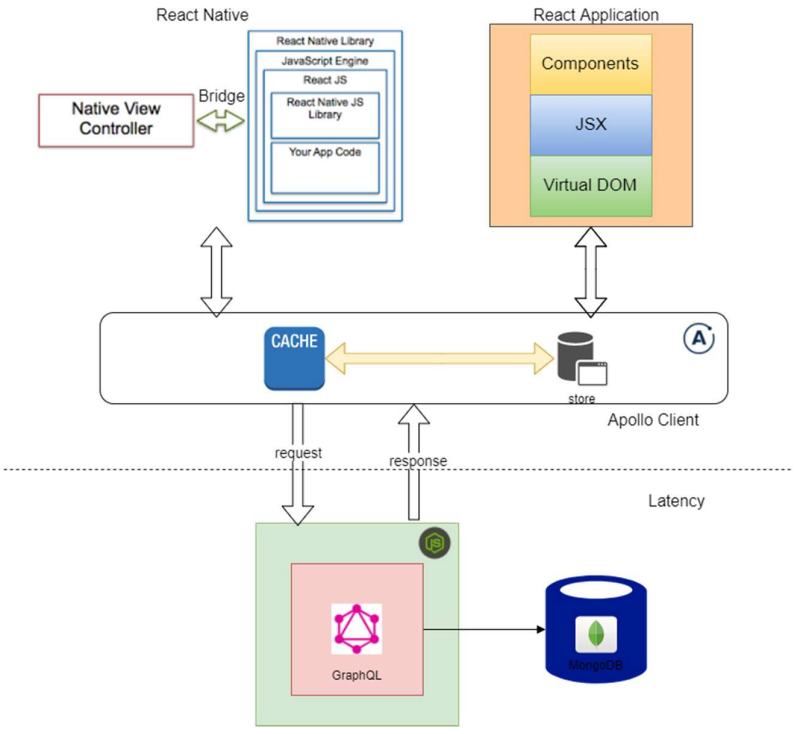


Figure 1 Architecture Overview

5.1 Database Design

The database design is the crux of the entire application. The model has to be solid with the room for improvement to accommodate future needs. The following sections discuss covers the data collection process, artifacts selection and database, structuring of user details, memorial information and relationships between them.

5.1.1 Data Collection

Partial data needed for the project was already collected for the application. *Figure 2* shows a sample data that was collected previously. Further efforts were taken to collect the physical records from the Highland Park administration through the help of park volunteers. The information pertaining to the individual (deceased person) and their personal information were not provided by the park administration as they are not properly maintained. The usual process of erecting a memorial in a public park run by the government involves a donation to the park's administration and submission of an application. Information is recorded and put up on special plaques and displayed in park's notice board, only if the donation exceeds a certain threshold (which differs from park to park).

Title	Type	Lat	Long
Bonnie Jean O'Connor	Chair	43.131545	-77.601942
Cole Kelley Melcher	Chair	43.131553	-77.602652
Dr. Donald W. Frank	Chair	43.131712	-77.602099
June Peer	Chair	43.131276	-77.603322
Angelo August West	Tree	43.129683	-77.607171
Angelo August West	Tree	43.129584	-77.607435
Ann Kremer Reinhardt	Tree	43.129581	-77.607162
Bill & Margaret Finucane	Tree	43.129498	-77.607186
Winifred Ann Margaret Richardson	Tree	43.129823	-77.607303

Figure 2 Preliminary Data

Alternative approaches like searching local obituary websites (Rochester Democrat And Chronicle Obituaries and Guestbooks, 2018) and news articles by cross-referencing the name from the memorials, helped to retrieve few

personal details up to some extent. Other sources include websites (Public Records Directory, 2018) that specializes in extracting information about the person by name were also used to collect the data. All the data collected through these websites, although not complete, provide the groundwork to build deceased person profile, their memorial, and its location.

5.1.2 Discussion of Design Features

The data collection process revealed the necessary artifacts needed to model the data. All the intelligence obtained about the deceased person scavenged from various sources are collated together for building the database at the later stage. The artifacts that are collected during the data collection process for the deceased person are as follows

- Deceased person's essential details like name, date of birth, date of death, address, profile picture, description about the person like their characteristics, lifetime achievements (some personal information about the deceased person's life)
- Details about the memorial like contributor's name and erection date, type of memorial (tree, bench, garden or other unique contribution), location (latitude and longitude coordinates) and a photo of the memorial if available.

The above observations are enough to capture all the details about the deceased person and their memorials. The particulars that were collected from online resources may or may not provide concrete and conclusive proof that the information belongs to the deceased person. This has to be confirmed by an acquaintance or someone who knows the departed person. So there is a need for verification which is required to be incorporated along with the person's information. If there are modifications, changes to the existing data must be carried out, and such modifications have to be segregated from the rest of the verified content. Also, the changes have to be reviewed to justify the modification. Moreover, not everybody can edit

the database, only authorized users (moderators) are allowed to change the content in order to mitigate entry of bogus facts. The moderators should be easily identified from regular users and their personal social media profile has to be recorded in order to make sure that the authenticated user is a legit person.

Since the server directly exposes the database through GraphQL, the endpoint must be secured by wrapping it with scopes (*section 5.2.5*), so that the client methods in GraphQL can be separated from admin commands like editing the database.

5.1.3 MongoDB

The design features discussed in *section 5.1.2* was modeled with MongoDB database in mind. It is classified as a No-SQL database because it does not follow any schema. MongoDB is superior to SQL database as the structure can be changed on the fly without affecting any data. It can store a large chunk of data without requiring any additional plugins and easily be manipulated through JavaScript. Also, geospatial queries with the help of spatial index, distance calculation between the locations, can be easily performed with MongoDB. Since the application has the potential to grow into a bigger platform, flexibility and scalability are needed. MongoDB provides these functionalities through features like replication and sharding. Sharding is the process of distributing the database across multiple machines. Each instance is a subset (called shards) of the original database. Sharding enables the distribution of the workload across multiple machines by directing operations to required shard. This increases the performance of both read and write operations. Replication is the process of duplicating the database/shards or having multiple instances (called replica sets) of the same database/shards. This provides recovery from failure by switching to another node that runs the copy of the failed shard/database.

5.1.4 Database Model and Relationships

Based on the analysis from *section 5.1.2*, database scheme is developed. The E-R model kind of representation, although not accurate for the no-SQL database is provided in *Figure 3* and *Figure 4* for better understanding the relationship between the collections. All the relationship constraint and business rules are enforced logically in server design through validation which can be found in *section 5.2.6*.

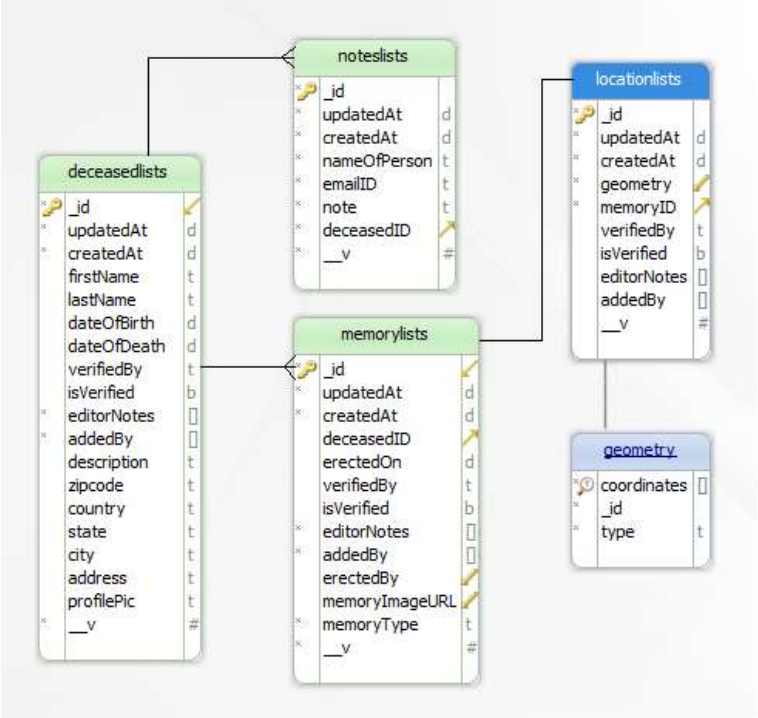


Figure 3 Schema – I

The data dictionary of each database is given in tables below. The *deceasedLists collection* (refer *Table 1*) encapsulates the personal information of a deceased person. The *memoryLists collection* (refer *Table 2*) records the memorial detail contributed to the deceased person. The *locationlists* (refer *Table 3*) contains the location information of the memorial. The coordinates are represented as Geo JSON objects which are indexed as 2dsphere for geospatial queries.

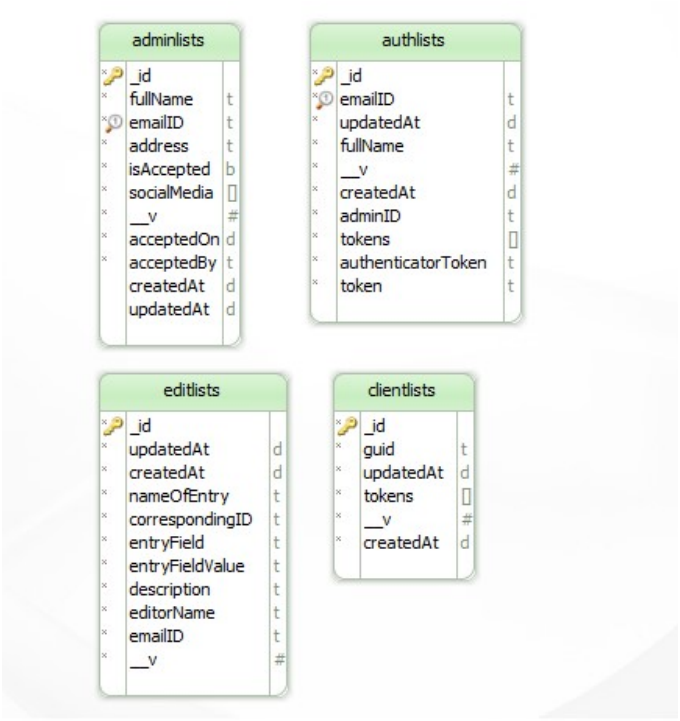


Figure 4 Schema – II

The *notesList* (refer *Table 4*) encapsulates the testimony for deceased person entry. User information like email and name is collected to ensure the legitimacy of the entry.

The *editsLists* (refer *Table 5*) records all the edits that are needed to be made to the entries in other collections. Some additional data about the user is collected to ensure the legitimacy of the user.

The *adminLists* (refer

Table 6) contain profile information of the moderators. Both pending requests to become a moderator and current moderators are listed in the same collection. In addition to the above, separate collections are required to store token information of client and moderator for authentication and accessing the endpoints. The *clientLists* (refer *Table 8*) contains registered clients and their token status. Client devices are identified by a unique identifier stored in *guid* (Globally Unique Identifier). For now, it is just a random 24 character wide generated from ID generator package. In future, it can be swapped to

implement ID based on device information combined with a secret to restrict the automated creation of clients by anyone who is accessing the API.

Table 1 deceasedLists Collection

Field Name	***Data type***	***Business Rules***	***Description***
*firstName**	String	Alphabets and symbols are permitted. Length: 2-15	First Name of the deceased person
*lastName**	String	Alphabets and symbols are permitted. Length: 2-15	Last Name of the deceased person
profilePic	String	Valid Cloudinary URL	URL of an image
dateOfBirth	Date	Date in format "YYYY-MM-DD". This should be older than *dateOfDeath*	Date of Birth of a deceased person
dateOfDeath	Date	Date in format "YYYY-MM-DD". This should be newer than *dateOfBirth*	Date of Death of a deceased person
address	String	Alphabets, numbers and symbols are allowed	Street address of deceased person
city	String	Only alphabets are permitted	City - address of the deceased person
state	String	Abbreviated form of state (only 2 characters are allowed)	State - address of the deceased person
country	String	Only alphabets are permitted	Country - address of the deceased person
zipcode	String	Only valid postal codes are allowed	Postal code - address of the deceased person
description	String	Alphabets and symbols are permitted.	Description about the deceased person
*addedBy**	List of Strings	Valid name and email id in format "<contributor name> - <contributor email ID>"	Details of the person who contributed modification to the entry. New contributions are added to the top of the list.
*editorNotes**	List of Strings	Alphabets and symbols are permitted.	Notes for the edits. New notes are added to the top of the list.
isVerified	Boolean	Always set to false for new entry	This flag is set if the entry is reviewed and verified by an administrator
verifiedBy	String	String representation of MongoDB object id of valid administrator, if *isVerified* is truth-y	Points to the profile of reviewer
editsList	List of Strings	String representation of MongoDB object id of valid edit entry	Points to list of edits that was performed on the entry

* The variable is required to create a document in the collection.

Table 2 memoryLists Collection

Field Name	***Data type***	***Business Rules***	***Description***
memoryType[*]	Enum (default: OTHER)	It can be one among the following 'TREE', 'BENCH', 'GARDEN', 'OTHER'.	Category of memorial
memoryImageURL	List of Strings	Valid Cloudinary URL	List of URL's containing pictures of memorial
deceasedID[*]	String	String representation of MongoDB object id of valid deceased entry	Points to the deceased person profile
erectedOn	Date	Date in format "YYYY-MM-DD". This should be newer than *dateOfDeath*	Established Date
erectedBy	List of Strings	Alphabets and symbols are permitted. Each of length: 2-30	List of memorial contributors
addedBy[*]	List of Strings	Valid name and email id in format "<contributor name> - <contributor email ID>"	Details of the person who contributed modification to the entry. New contributions are added to the top of the list.
editorNotes[*]	List of Strings	Alphabets and symbols are permitted.	Notes for the edits. New notes are added to the top of the list.
isVerified	Boolean	Always set to false for new entry	This flag is set if the entry is reviewed and verified by an administrator
verifiedBy	String	String representation of MongoDB object id of valid administrator, if *isVerified* is truth-y	Points to the profile of reviewer
editsList	List of Strings	String representation of MongoDB object id of valid edit entry	Points to list of edits that was performed on the entry

[*] The variable is required to create a document in the collection.

Table 3 locationLists Collection

Field Name	***Data type***	***Business Rules***	***Description***
*latitude** *(geometry collection)*	String	Valid latitude coordinate	Latitude coordinate of the memorial
*longitude** *(geometry collection)*	String	Valid longitude coordinate	Longitude coordinate of the memorial
*memoryID**	String	String representation of MongoDB object id of valid memory entry	Points to the memorial entry
*addedBy**	List of Strings	Valid name and email id in format "<contributor name> - <contributor email ID>"	Details of the person who contributed modification to the entry. New contributions are added to the top of the list.
*editorNotes**	List of Strings	Alphabets and symbols are permitted.	Notes for the edits. New notes are added to the top of the list.
isVerified	Boolean	Always set to false for new entry	This flag is set if the entry is reviewed and verified by an administrator
verifiedBy	String	String representation of MongoDB object id of valid administrator, if *isVerified* is truth-y	Points to the profile of reviewer
editsList	List of Strings	String representation of MongoDB object id of valid edit entry	Points to list of edits that was performed on the entry

Table 4 notesLists Collection

Field Name	***Data type***	***Business Rules***	***Description***
*nameOfPerson**	String	Alphabets and symbols are permitted. Length: 2-30	User who writes the notes
*emailID**	String	Should be a valid email ID	Email id of user
*note**	String	Alphabets and symbols are permitted.	Testimonial written by user to the deceased person
*deceasedID**	String	String representation of MongoDB object id of valid deceased entry	Points to the deceased person profile

* The variable is required to create a document in the collection.

Table 5 editsLists Collection

Field Name	***Data type***	***Business Rules***	***Description***
*nameOfEntry**	Enum	It can be one among the following 'DECEASED_PERSON', 'MEMORY', 'LOCATION', 'DECEASED_PERSON_IMAGE', 'MEMORY_IMAGE'.	Category of Edit
*correspondingID**	String	String representation of MongoDB object id of valid entry in database	Points to the corresponding entry in collection based on *nameOfEntry*
*entryField**	String	Alphabets and symbols are permitted.	Suggest problem
*entryFieldValue**	String	Alphabets and symbols are permitted.	Suggest modification
*description**	String	Alphabets and symbols are permitted.	Justification for modification
*editorName**	String	Alphabets and symbols are permitted. Length: 2-30	Name of the user who suggests edit
*emailID**	String	Should be a valid email ID	Email id of user
*archive**	Boolean	Always set to false for new edit	This flag is set if the suggested edit is applied

Table 6 adminLists Collection

Field Name	***Data type***	***Business Rules***	***Description***
*fullName**	Enum	Alphabets and symbols are permitted. Length: 2-30	Name of the administrator
*emailID**	String	Should be a valid email ID	Email id of the administrator
socialMedia	List of Strings	Valid URL	Social media profiles that belongs to the administrator
isAccepted	Boolean	Always set to false for new entry	This flag is set if the entry is reviewed and accepted by an administrator
acceptedBy	String	String representation of MongoDB object id of valid administrator, if *isAccepted* is truth-y	Points to the profile of reviewer
acceptedOn	Date	Should be set if the *isAccepted* flag is truth-y	Timestamp the moderator got accepted

* The variable is required to create a document in the collection.

Table 7 authLists Collection

Field Name	*Data type*	*Business Rules*	*Description*
emailID[*]	String	Should be a valid email ID	Email id of the administrator
tokens[*]	List of Strings	Token Id of JWT	Access and Refresh token id that is generated for the administrator
authenticatorToken[*]	String	Google user id	Google user id
adminId[*]	String	String representation of MongoDB object id of valid administrator entry	Points to the profile of administrator

Table 8 clientLists Collection

Field Name	*Data type*	*Business Rules*	*Description*
guid[*]	String	Globally Unique Identifier (guid)	Unique identifier of client
tokens[*]	List of Strings	Token Id of JWT	Access and Refresh token id that is generated for the entry
Quota[†]	String	Quota for image upload to limit user upload	Quota for client to upload images

The relationship between the collections and the ground rules on who can modify the collection are as follows:

- Each deceased person can have n memorials contributed to them but each memorial should only one location entry.
- Each deceased person can have many testimonials written by acquaints.
- Anyone can create an entry (users and moderators), but only a moderator can edit an already existing entry.
- A record in the collection can be verified only by authorized users (moderators).

5.2 Server Implementation

The server abstracts the data from the database and implements business logic like authentication of clients, validation of fields, and handling photo upload.

[*] The variable is required to create a document in the collection.
[†] Experimental feature (not implemented)

The architectural stack of the server is provided in *Figure 5*, and each layer is explained in the below subsections.

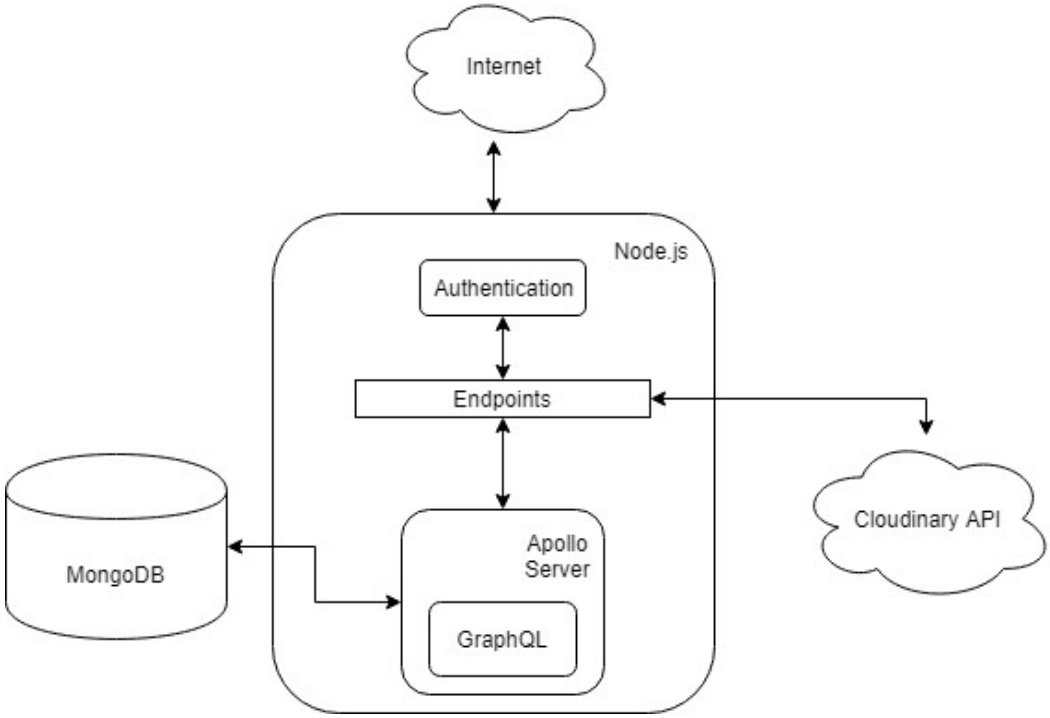


Figure 5 Server Architecture

5.2.1 Node.js

Node.js enables JavaScript to be executed on the server-side. It uses Google's V8 JavaScript runtime to execute JavaScript code on the server. Since it is an event-driven loop, the connection between the client and the server need not be maintained at all times. It also has powerful plugin support and builds tools that can be used to develop, maintain and test the functionality quickly. The node.js server is split into different endpoints to access GraphQL server, REST API for maintaining images in Cloudinary data-store (*section 5.2.4*) and user authentication.

5.2.2 Apollo Server and GraphQL

Apollo server is an implementation of GraphQL server that has good library support, ease of use and works well with Apollo client (*section 5.3.2*) providing additional functionalities. GraphQL is a data query language that is used to

query application servers with specific data requirements. The language allows a client to define the exact structure of data required and receive the same structure of data back from the server. Unlike REST API which requires multiple requests to retrieve resources at different endpoints, GraphQL can acquire all the data needed by the application in a single request. This leads to fewer network hops. In addition, GraphQL is designed in terms of fields and "types", not endpoints. A GraphQL server is queried for the type it supports, ensuring that the client can only ask for what is possible and provide unambiguous and helpful errors. This avoids manual parsing code at client side. The main advantage of GraphQL is its WYSIWYG model. In other words, the structure of a query's result will map exactly to the structure of the query request, which reduces the number of errors during unmarshalling responses.

Each GraphQL schema is built up of "types" and "resolvers". The "types" defines the structure of data returned along with the operations-queries (similar to read operation) and mutations (similar to write operations) - that can be performed on the database. Each "type" has its own "resolver" which defines how a "type" is mapped to the fields in the database and how a query or mutation is resolved. The fields can be of any primitive data type. Although, it is possible to add object field value in GraphQL-as the document grows with many memorials-it becomes complex and resource intensive to query and modify the entry. So the structure of the schema is separated into individual collections. Another quirk of GraphQL is that it only handles text-based fields, so the images and other files cannot be stored as blobs. However, it is possible to store it locally as a file on the server and expose the files as a public resource. This approach doesn't provide much security as the server can be prone to attacks. The images are stored in a dedicated cloud service and is discussed in *section 5.2.4*.

5.2.3 GraphQL Schema

The GraphQL schema was created to interface with the underlying MongoDB database. The resulting schema is a complex structure made up of different types, enums (enumerated list), inputs, queries, and mutation. A detailed

explanation can be found in the documentation, while a brief overview of the schema is discussed below.

- **Deceased Type** captures all the details about the deceased person and maps to the *deceasedlists* from the database. It contains queries to search a deceased person by *firstName* and *lastName* and by *personID*. It also provides options to create, modify and verify an entry.
- **Memory Type** encapsulates the details such as the type of memory, erected date, and contributor name and maps to the *memoriallists* from the database. Memorials can be queried by memorial type and by ID and can be created, modified and verified using mutations.
- **Location Type** maps to the *locationlists*. Queries like *getNearByLocation* and mutations like *createLocation*, *editLocation* can be performed.
- **Note Type** maps to the *notelists* and contains testimonials written by acquaints of the deceased person. Mutations to create a note and query to get a note by ID can be executed.
- **Admin Type** maps to the *adminlists* database. Mutation to refresh authentication tokens and query to get admins by ID is exposed.
- **Client Type** mutations and queries are exposed to register and refresh authentication tokens to the client application.

The main type "Deceased, Memory, Location" is modeled in such a way to have a cyclic redundancy with other linked types. For example, "Deceased" type has a field called *memories* which returns a list of memorial entries corresponding to the deceased person. The memory contains variable *deceasedPerson* that returns deceased person entry for that memorial entry. So, this enables the user to access any piece of data by invoking any operations corresponding to the type. Due to security concerns, some crucial fields and flags are not allowed to be sent through the input. For example, variable *isVerified* which determines whether a field is verified or not is exposed through a method *verfiedDeceasedPerson(id)* to modify the content. Each "type" has accompanying

queries and mutations which are wrapped by authentication middle-ware (*section 5.2.5*) to restrict unauthorized access.

5.2.4 Cloudinary Endpoint

Due to the reason explained in *section 5.2.2*, a RESTful endpoint is created to handle image uploads. RESTful endpoint follows REST (Representational State Transfer) an architectural style for developing web services. It is a stateless client-server architecture that defines a set of constraints based on HTTP (Hypertext Transfer Protocol). RESTful endpoint exposes some functionality through HTTP methods as described in RFC 2616 (Nielsen, et al., 1999).

Based on the guidelines defined by RFC 2616, a RESTful endpoint is created to store the incoming images in Cloudinary server. Cloudinary is a SaaS (Software as a Service) technology company that provides cloud-based media management solution. The Cloudinary service is chosen because of enterprise features like image transformations on the fly through URL request, auto-scaling of the server, backup, image processing and analysis for nudity detection and its free service for open source projects. The RESTful endpoint that handles uploads also checks for the authorization of incoming request and redirects the request to the Cloudinary API with the help of Cloudinary library (to communicate with the Cloudinary service) and formidable package (support library to process multipart data form). The returned response is a URL with other parameters from the Cloudinary API which is shortened and sent to the requester.

5.2.5 Authentication and Authorization Strategy

The server API's was secured using token-based authentication so that online-bots and other such unauthorized users cannot add bogus data to the database. The authentication is implemented for both administrators and clients. The JSON Web Token (*JWT*) (Bradley, Sakimura, & Jones, 2015) is an open standard to communicate claims (information) between two parties which are used for authentication in the application. The claims in JWT contains email id in case of admin token and Globally Unique Identifier (*guid*) in case of client token. It

also consists of scopes, issuer and expiry date of the token. Since anyone can read the claim from the token, it is signed using RS256 algorithm (Jonsson, Moriarty, Kaliski, & Rusch, 2016). RS256 is an asymmetric algorithm, and it uses private and public key pair. The private key is used to sign the token and public key is used to verify the signature. The RS256 is chosen with scalability in mind if the application grew to have many servers, one instance of the server can be made exclusive to provide signed tokens using the private key and the public key can be shared among others to verify the validity of signed token. The public key can also be shared with the clients to verify the authenticity of the token.

The client access token contains scopes that enable the user to perform query and operation and create an entry in the database. The client access token is not authorized to execute an edit on an existing record. It also has scope to upload an image to the Cloudinary server. The admin access token has the same scope, but in addition to that, they can modify an existing entry and also verify an entry. A special scope *admin:old* is issued to the members who have been a user for more than 100 days. Only moderators with this scope can approve other moderators (pending requests). In addition to the access token which is valid for 24 hours, refresh token is provided with scope *client:refresh admin:refresh* respectively to refresh the access token. The refresh token is valid for 30 days and after that, the user has to login again to get new set of tokens.

There are two possible ways to perform scope check on operations in the schema, one is by using directives (custom functions that are executed before each invocation, which are used for meeting certain criteria) and the other way is to use a middle-ware before each operation in the GraphQL resolver. The support to run custom directives on QUERY and MUTATION were not (and still not) available at the time of development. So a simple middleware function is created to check scopes before the execution of each operation.

The tokens are issued to the clients through *registerClient({guid:...})* mutation. This is the only operation in the schema that doesn't need any authentication.

The admin token is issued through Google OAuth server-side logic (Boyd, 2012) which is represented in *Figure 6*. The moderator login using Google OAuth which return a token object. The access code from the token object is sent to a special REST endpoint to check the integrity of access code. The access token is issued by verification against Google OAuth server and database entry. Mutation like *refreshClientAccessToken*, *refreshAdminToken* mutation can be used to refresh tokens after their initial issue. The scope list is provided in the *Appendix-A*.

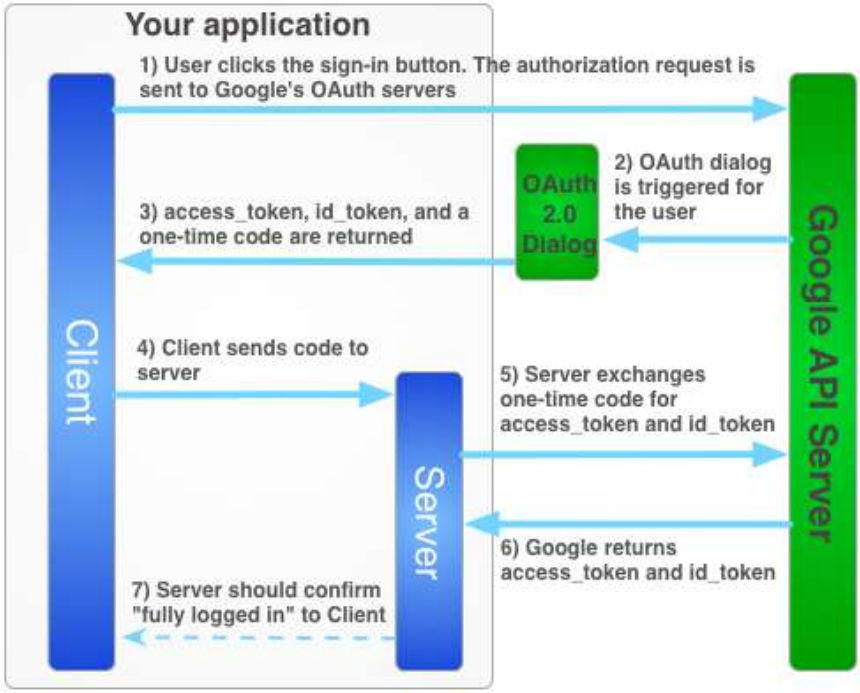


Figure 6 Google OAuth Logic (Google Inc., 2017)

5.2.6 Validation

GraphQL validates the type of data that is being sent from the client. The data type check might not be enough to justify the content of the data. In addition to that, business rules pertaining to the system has to be implemented. Based on the discussion in *section 5.1.4*, custom class methods were written to check the validity of the incoming data object. All the constraints listed in *Table 1* deceasedLists Collection to *Table 8* clientLists Collection were implemented.

The incoming data is passed to the validator class and appropriate error messages are returned if the input doesn't satisfy the business rule.

5.2.7 Error Handling and Logging

All the activity on the server is logged as color-coded JSON objects using Winston package. Winston is a logging library that provides interfaces for recording warnings, errors or custom messages for troubleshooting purposes. The log messages are stored in separate files to segregate between the regular log and error log. All the error handling is done using Apollo error package which provides good integration with the GraphQL response. Error module with custom messages for common error scenarios was assigned to handle errors. Each error message has a unique identifier and a stack trace which are logged in files. This identifier makes it easy to identify the error and troubleshoot them. A complete log of error messages can be found in *Appendix B*.

5.2.8 Testing

Testing of the server revealed a lot of bugs during the initial phase of development. The unit tests were written using Mocha and Chai packages to test the business logic. Mocha is a testing framework for Node.js and Chai is the assertion library to test the scenarios. Unit tests were written to test the underlying database schema and validation logic to make sure it works according to the business logic. The Cloudinary endpoint (endpoint to upload images to the Cloudinary server) is tested using Postman by providing various test images. Postman provides the client interface for testing web services through the HTTP protocol. The GraphQL queries and mutations were tested using GraphiQL. GraphiQL is a graphical interface that enables the user to interact with the GraphQL server, to check if the operations return data or proper Apollo error messages.

5.3 Mobile Client Implementation

The initial plan was to develop the client for both Android and iOS. But, due to the time constraint of the project and difficulties faced during the developmental phase of a mobile client in React Native, only the Android version of the mobile client was possible. The challenges were solely due to the rapid development cycle of React Native and supporting libraries releasing breaking updates. The detailed description scrutinizing the developmental process is discussed in the *section 5.3.4*.

5.3.1 Wireframe and Prototype

Based on the tasks that the application must perform, a wire-frame model was designed, and it is shown in *Figure 7*. Building upon the wire-frame model, prototype of the application was created and listed in *Appendix C.*

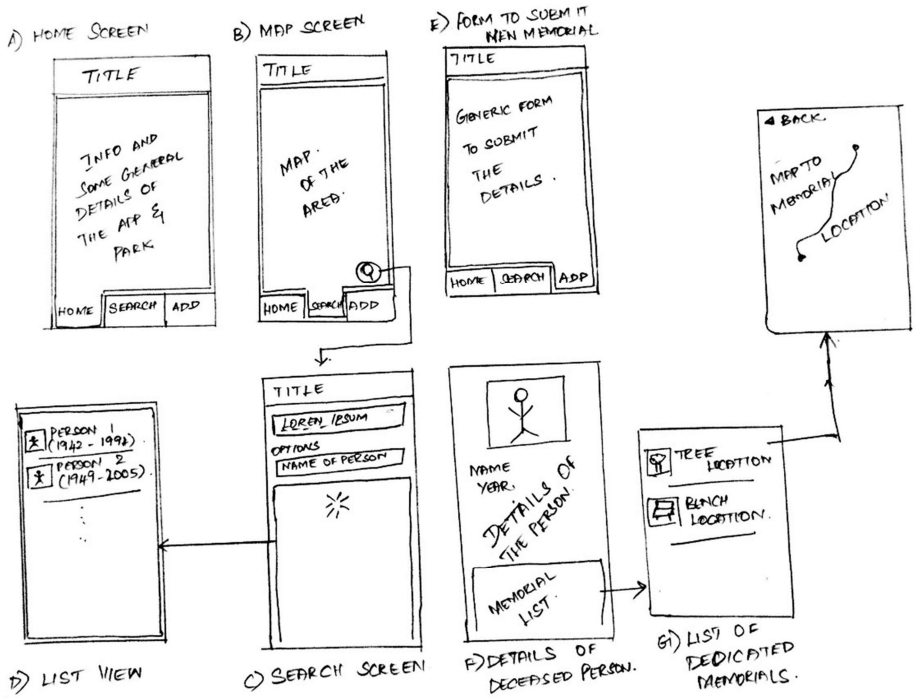


Figure 7 Wireframe diagram

5.3.2 Mobile Client Stack

React Native is an extension of hybrid mobile application. The hybrid mobile application is written in JavaScript and run on mobile devices using a web

container. Instead of running the application in a web container, React Native converts JavaScript code into native API calls. The React Native uses a component called "bridge" that contains native-JavaScript libraries to convert the JavaScript code to native code (refer *figure 5*). The end product despite written in JavaScript will be invoking native commands. This makes the React Native application to leverage the flexibility of JavaScript application and performance benefits of native code

Apollo client is a data management and caching tool for the user interface. It helps for fast loading of data and consistent UI between different views. Apollo client has an inbuilt data store, which stores all the data needed for the UI. If any changes are made to the data store all components using the data from the data store will be notified and changes will be made to the UI. This is necessary for the react application because dependencies and state management are hard to manage, as the data flows only one way. Caching is another advantage in Apollo as only data needed and not present in the cache will be requested. This reduces the payload and increases the efficiency of the system.

Google Maps API is used to integrate Google maps with the developer application. Google map can be used to track user location, create a pin on the map and navigate from one place to another. Google also provides developer tools to create custom maps with user data which will be used to implement custom markers and to create navigation to the memorials.

5.3.3 Client Development

The main feature of the application is to provide directions to the nearby memorials based on the user's location. The memorials spots are highlighted on Google maps interface which was achieved by integrating with react native maps developed by Airbnb. The initial loading animation was implemented using Lottie package (support library to implement custom animations). The loading screen is required to initialize the cache and check/fetch the access token from the server. The location data necessary to get nearby memorials is also fetched at this stage. In addition to the proximity query, the user can also

use the search tab to manually search for the deceased person by name and get memorials by their type.

The user can also create a new entry for the deceased person, memorial entry or a location entry through the interface. The user can suggest edits using the edit option provided for each entry. The user can also apply to become one of the moderators by signing up using the request form present under "About" tab. The direction to the selected memorial is provided by drawing a polygon using the response from Google Directions API. The polygon sketched using Google Directions API may not be able to give precise direction, so a dedicated button to provide directions using Google Maps application is provided in the interface for usability purposes.

5.3.4 Scrutinizing the Support Libraries

The initial base project setup was using the official react-native-app library. Although the setup process was fairly easy and works out of the box, the core packages listed in the "package.json" were outdated. These packages caused a few issues with using libraries like Lottie and React Native Maps. Adding to it, the provided Android API versions, Gradle and Android build tools were out of date which caused issues while integrating with the map library. The documentation related to the issues were very limiting and most of the solution to the problems were found by skimming through the various issue trackers of the corresponding libraries and trying out different solutions provided by other developers.

The Apollo client recently deprecated the native redux (state management framework) support to their own implementation for state management. Even though the transition was smooth, the documentation to incorporate the new change was not clear enough to understand, but it has been improved a lot during the writing of this report. The swapping of access tokens and refreshing of a token on the fly can be performed using a custom function in Apollo context layer. In the Apollo context layer, the response can be read and packaged again to be sent to the UI or the token can be refreshed, if there are

some errors. The request uses new inbuilt Higher Order Component (<Query>, <Mutation>) which binds to the query that is being invoked. Modifying the query in the context layer to intercept the errors corrupts the response. The authentication flow is implemented in such a way that the application checks for the token validity during each boot up. The headers in the Apollo client cannot be reset after the initialization of Apollo object. This makes it difficult to pass additional parameters in the header if required. Some Apollo client version like Apollo-client-2.2.8 crashes the application if the response contains Apollo errors.

The integration of iOS with Google maps is very limited and buggy. Conversely, the permission requiring for accessing the location and camera have a different API which requires the application to be rewritten twice using respective native API. Even the React Native API that interfaces with both Android and iOS support different feature sets. In order to have a consistent experience, various part of the code base has to be forked to support both the environments. Due to the time constraint of the project and various bugs in the underlying library, only Android client application was possible.

5.3.5 Known Bugs in Mobile Client

During the developmental cycle, various bugs were found and fixed but there are still a few bugs. All the bugs are due to the underlying issues with the supporting libraries and it cannot be fixed unless the code-base is fixed.

- The animation in Android is dodgy. The transition between the screens is not smooth. The underlying animation API has gone through a lot of updates recently and in the future, this might be improved.
- The custom markers in the signed-apk version are big and don't scale according to zoom. The use of multiple image sizes for the markers improved the image appearance, but it is too big for the view. This is a known bug in React Native maps and the community is actively working to fix it.

- A scrollable text view inside a scroll view container cannot be scrolled. The scrolling action always results in the scrolling of the parent container. This is a known bug in the React Native package and the only solution is to avoid using it.
- During offline mode, requesting some data from the server through Apollo using the new Higher order component (<Query>, <Mutation>) results in crashing the application if that said request is not cached. The default behavior should be queuing of the request until the network is available and return GraphQL error on timeout.

5.3.6 Administration Console

The administrative console provides the user interface for the moderators to edit the entries in the database. The administrative console is built using React framework with the help of Apollo client. The moderators are provided with a way to login into the interface through Google OAuth feature and access the dashboard to edit an entry, verify an entry in the database and accept pending request. The feature set of the administrative console is relatively basic and doesn't provide the feature to edit an entry with the form submission. The moderator has to use a GraphiQL tool to manually query the server to make a modification.

6 Future work

All the technology that is used to build the application is fairly new and in the middle of aggressive development cycles. The authentication logic can be reworked in the future to apply as directives instead of the current implementing as middle-ware for cleaner code. The refreshing of tokens in the client is currently checked once during the start of the application due to lack of custom method implementation for Apollo GraphQL component. This can be switched to on-the-fly swapping of tokens by intercepting the errors due to an expired token. The administration console is a basic implementation and doesn't have many features. In future, the administrative console can be

improved to provide more features and functionalities. The React Native application has few bugs due to the supporting libraries as discussed in *section 5.3.5*. These can be fixed in future with the improvement in packages. The iOS version implementation of the client can be done. There is a lot of room for improvement in both client application and administration console. The application is fairly basic and a lot of features can be integrated as the underlying core projects mature.

7 Conclusion

The primary goal of the project is to build a scalable open-source platform to identify memorials in the local parks. In spite of being challenging in every way from the collection of data to the building up of the server to the development of client, it was a good learning experience building up the project from an idea to a fully evolved client-server application. The React Native, although relatively new with lack of features, has the potential to become a great platform that leverages the native features. The next step of the project will be to fix the bugs and improve the feature set of the application. As the underlying project matures, the memorial application has a lot of potentials to grow into a formidable open-source platform to assist local parks around the world or transform entirely into a powerful resource that morphs to the demands of the local park.

References

Baker, M. (2017, July 20). *Memorial Tree Dedicated To 9/11 Victim Mistakenly Cut Down In New Jersey*. (C. B. York, Editor) Retrieved from CBS Organization: http://newyork.cbslocal.com/2017/07/20/nj-trees-cut-down/

Barrett, A. G., Pitas, N. A., & Mowen, A. J. (2017). First In Our Hearts but Not in Our Pocket Books: Trends in Local Governmental Financing for Parks and Recreation from 2004 to 2014. *Journal of Park & Recreation Administration, 35*.

Boyd, R. (2012). *Getting started with OAuth 2.0.* " O'Reilly Media, Inc.".

Bradley, J., Sakimura, N., & Jones, M. (2015, 5). *JSON Web Token (JWT)*. RFC, Internet Engineering Task Force. Retrieved from http://tools.ietf.org/html/rfc7519.html

Cloke, P., & Pawson, E. (2008). Memorial trees and treescape memories. *Environment and Planning D: Society and Space, 26*, 107-122.

Dogu, U., & Erkip, F. (2000). Spatial factors affecting wayfinding and orientation: A case study in a shopping mall. *Environment and behavior, 32*, 731-755.

Fussell, P. (2009). *The Great War and modern memory.* Sterling Publishing Company, Inc.

Google Inc. (2017, October 17). *Google Sign-In for Server-Side Apps*. Retrieved from Google Developers: https://developers.google.com/identity/sign-in/web/server-side-flow

Head, D., & Isom, M. (2010). Age effects on wayfinding and route learning skills. *Behavioural brain research, 209*, 49-58.

Jonsson, J., Moriarty, K., Kaliski, B., & Rusch, A. (2016, 11). *PKCS# 1: RSA Cryptography Specifications Version 2.2.* RFC, Internet Engineering Task Force. Retrieved from https://www.rfc-editor.org/rfc/rfc8017.txt

McKelvey, B. (1940). The Flower City: Center of Nurseries and Fruit Orchards. *The Rochester Historical Society Publications, 18*, 121-169.

Meehan, T. (1907). *The Catholic Encycolpedia* (Vol. 2). Robert Appleton Company. Retrieved from http://www.newadvent.org/cathen/02312a.htm

Nielsen, H. F., Mogul, J., Masinter, L. M., Fielding, R. T., Gettys, J., Leach, P. J., & Berners-Lee, T. (1999, 6). *Hypertext Transfer Protocol -- HTTP/1.1.* RFC Editor. doi:10.17487/RFC2616

Passini, R. (1981). Wayfinding: A conceptual framework. *Urban Ecology, 5*, 17-31.

Peck, W. F. (1908). *History of Rochester and Monroe County, New York: From the Earliest Historic Times to the Beginning of 1907* (Vol. 1). Pioneer Publishing Company.

Public Records Directory. (2018, March). Retrieved from https://publicrecords.directory

Qeepr: Free Online Memorial Sites & Tributes. (2018, April). Retrieved from Qeepr: http://www.qeepr.com/

Rochester Democrat And Chronicle Obituaries and Guestbooks. (2018, April). Retrieved from Democrat And Chronicle: http://obits.democratandchronicle.com/obituaries/democratandchronicle/

Services, N. P. (2016, 10). *Digital Map for National Park Services*. Retrieved from https://www.nps.gov/npmap/

Waymarking. (2018, April). Retrieved from Waymarking: http://www.waymarking.com/

WebCemeteries: Memorial Solutions. (2018, April). Retrieved from WebCemeteries: http://www.webcemeteries.com/

Appendix

A. Scopes

The following are the admin scopes

- *admin:read* - access to get content from the server (moderator)
- *admin:create* - access to create an entry (moderator)
- *admin:edit* - access to edit an entry (moderator)
- *image:upload* - access to upload image to the cloud server (moderator)
- *admin:read* - access to get content from the server (moderator)
- *admin:old* - access to accept other moderators to the system. This is provided if the moderator is regular user who is using the system for more than 100 days.
- *admin:refresh* access to refresh the admin access token. Admin refresh token contains this scope.

The following are the client scopes

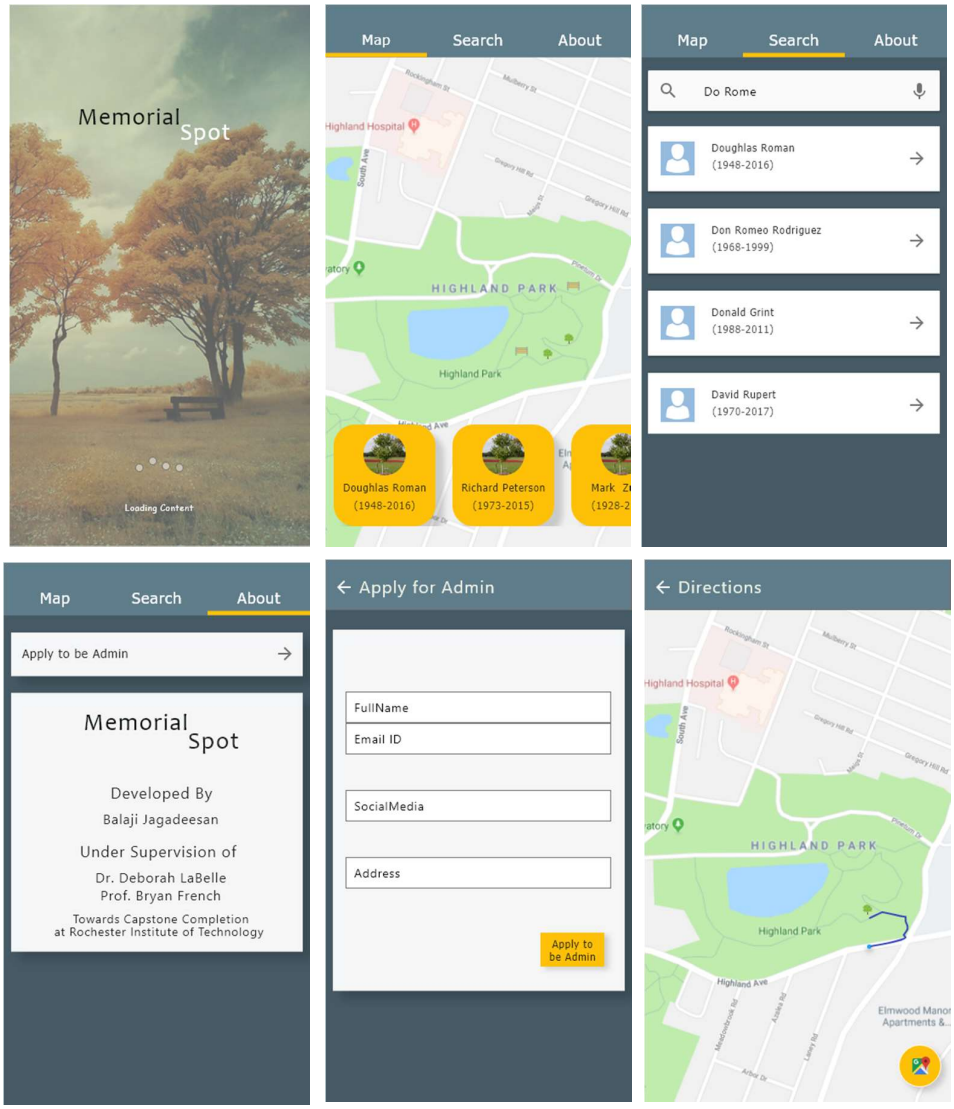
- *client:read* - access to read a resource
- *client:create* - access to create an entry
- *image:upload* - access to upload an image to the cloud server
- *client:refresh* - access to refresh the client access. Client refresh token contains this scope.

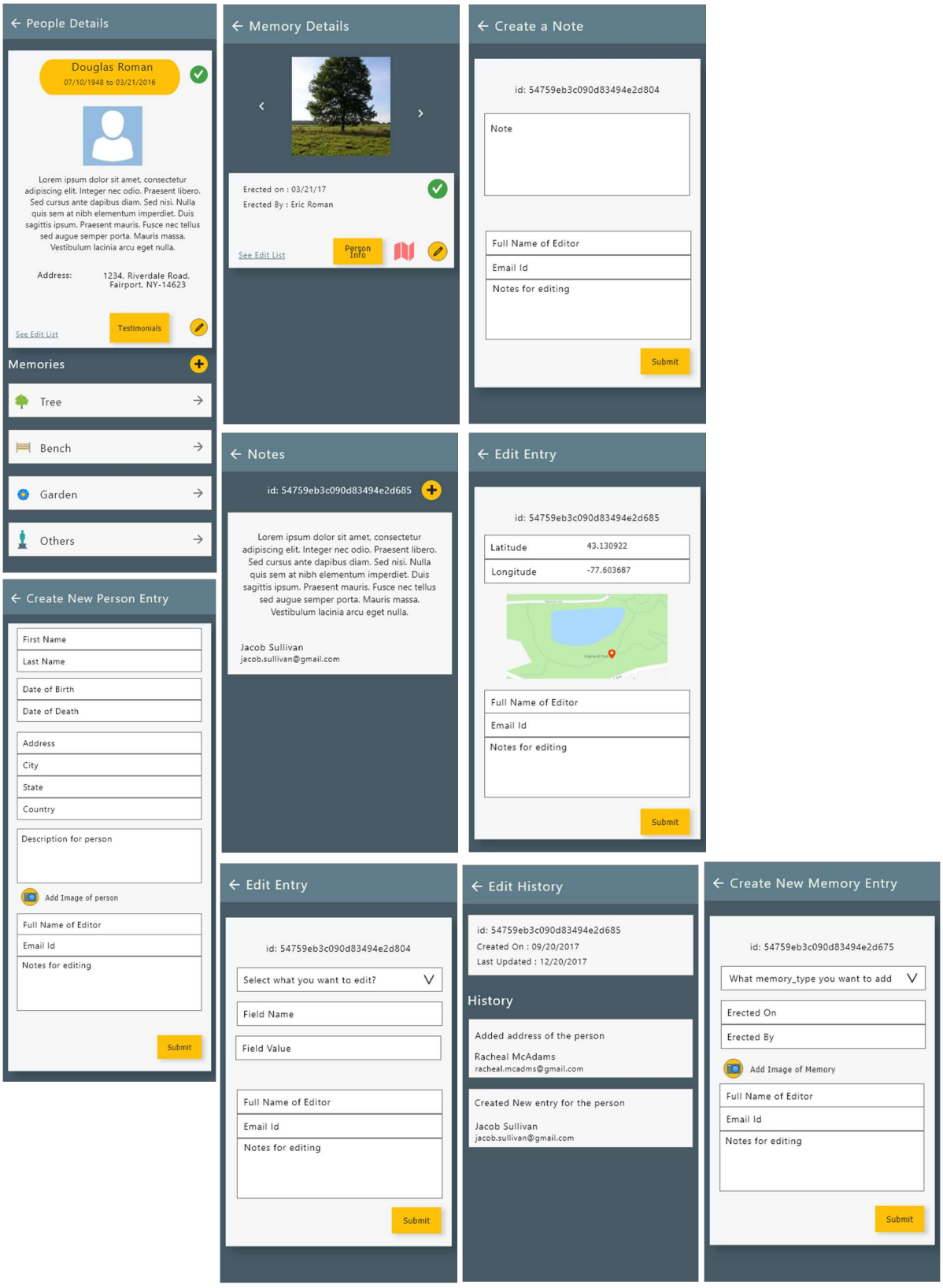
B. Error Codes

- 1500 - Not Authorized - Generalized not authorized error
- 1501 - No token can be found - thrown if token is not present in header
- 1502 - Token creation error
- 1503- expired token - token exp claim (time) is less than the current time
- 1504 - Token verification error, sign of token doesn't match the key

- 1505 - Invalid Token - Generalized token error
- 1511 - Token doesn't have necessary scope
- 1512 - GUID of client doesn't have any match in database
- 1513 - ID provided in input data(in GraphQL mutation/query) is not an object ID
- 1514 - Email ID not found database/ request pending
- 1000 - Some internal MongoDB error
- 2000 - Generalized unexpected server error
- 1550 - Validation error for input (in GraphQL mutation/query)

C. Prototype




People Details
Douglas Roman
07/10/1948 to 03/21/2016
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla.
Address:
1234, Riverdale Road, Fairport, NY-14623
See Edit List
Testimonials
Memories
Tree
Bench
Garden
Others
Memory Details
Erected on : 03/21/17
Erected By : Eric Roman
See Edit List
Person Info
Create a Note
id: 54759eb3c090d83494e2d804
Note
Full Name of Editor
Email Id
Notes for editing
Submit
Notes
id: 54759eb3c090d83494e2d685
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla.
Jacob Sullivan
jacob.sullivan@gmail.com
Edit Entry
id: 54759eb3c090d83494e2d685
Latitude
43.130922
Longitude
-77.603687
Full Name of Editor
Email Id
Notes for editing
Submit
Create New Person Entry
First Name
Last Name
Date of Birth
Date of Death
Address
City
State
Country
Description for person
Add Image of person
Full Name of Editor
Email Id
Notes for editing
Submit
Edit Entry
id: 54759eb3c090d83494e2d804
Select what you want to edit?
Field Name
Field Value
Full Name of Editor
Email Id
Notes for editing
Submit
Edit History
id: 54759eb3c090d83494e2d685
Created On : 09/20/2017
Last Updated : 12/20/2017
History
Added address of the person
Racheal McAdams
racheal.mcadms@gmail.com
Created New entry for the person
Jacob Sullivan
jacob.sullivan@gmail.com
Create New Memory Entry
id: 54759eb3c090d83494e2d675
What memory_type you want to add
Erected On
Erected By
Add Image of Memory
Full Name of Editor
Email Id
Notes for editing
Submit