Study Space
Design Document
Version 1.2 - 07/19/2017

1.0 Introduction

1.1 Purpose

This design document describes how the requirements for the Study Space application are achieved. It goes into detail regarding software architecture, packages and module functions.

1.2 Scope

This document guides the software developers in the creation of the product.

1.3 Definitions

BeaconDB - The database table which stores a list of all active beacons.
ChatDB - The database table which logs all chat messages in beacons and between users.
UserDB - The database table which stores all user credentials.
CourseDB – The database table which stores all the course details.

1.4 References

IEEE. IEEE Std 1016-1998 IEEE Recommended Practice for Software Design Descriptions. IEEE Computer Society, 1998

1.5 Overview

This document is written adhering to the guidelines outlined by the IEEE Recommended Practice for Software Design Descriptions.

2.0 Decomposition Description

This section shows the decomposition of the system in 3 ways:

- Module decomposition is the breakdown of each module, the data and functions of each class are intended to be easy to change, reuse and comprehend.
- Concurrent process decomposition explains what the user will be able to see on each page
- Data decomposition explains what is used to store the persistent data.

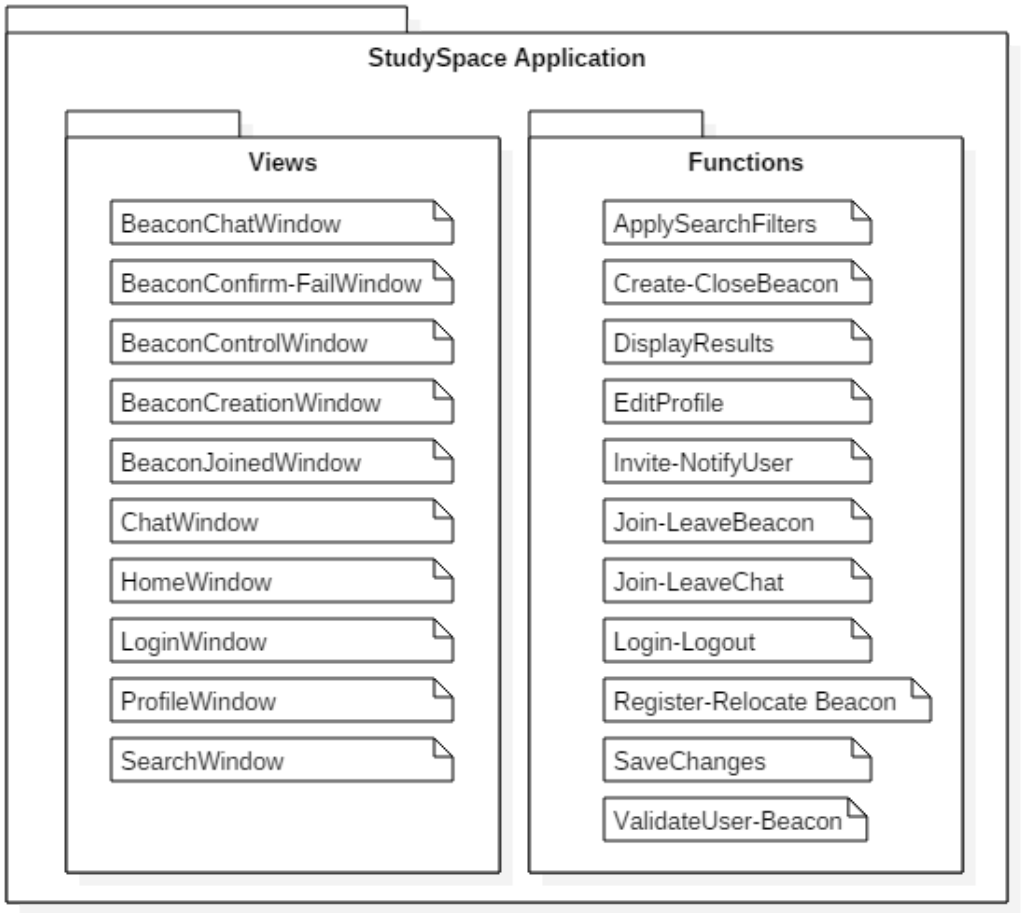
2.1 Module Decomposition

2.1.1 Package Diagram

The design of Study Space separates the front end and back end. The front end is referred to as the Study Space Application and the backend is referred to as the Study Space Backend.

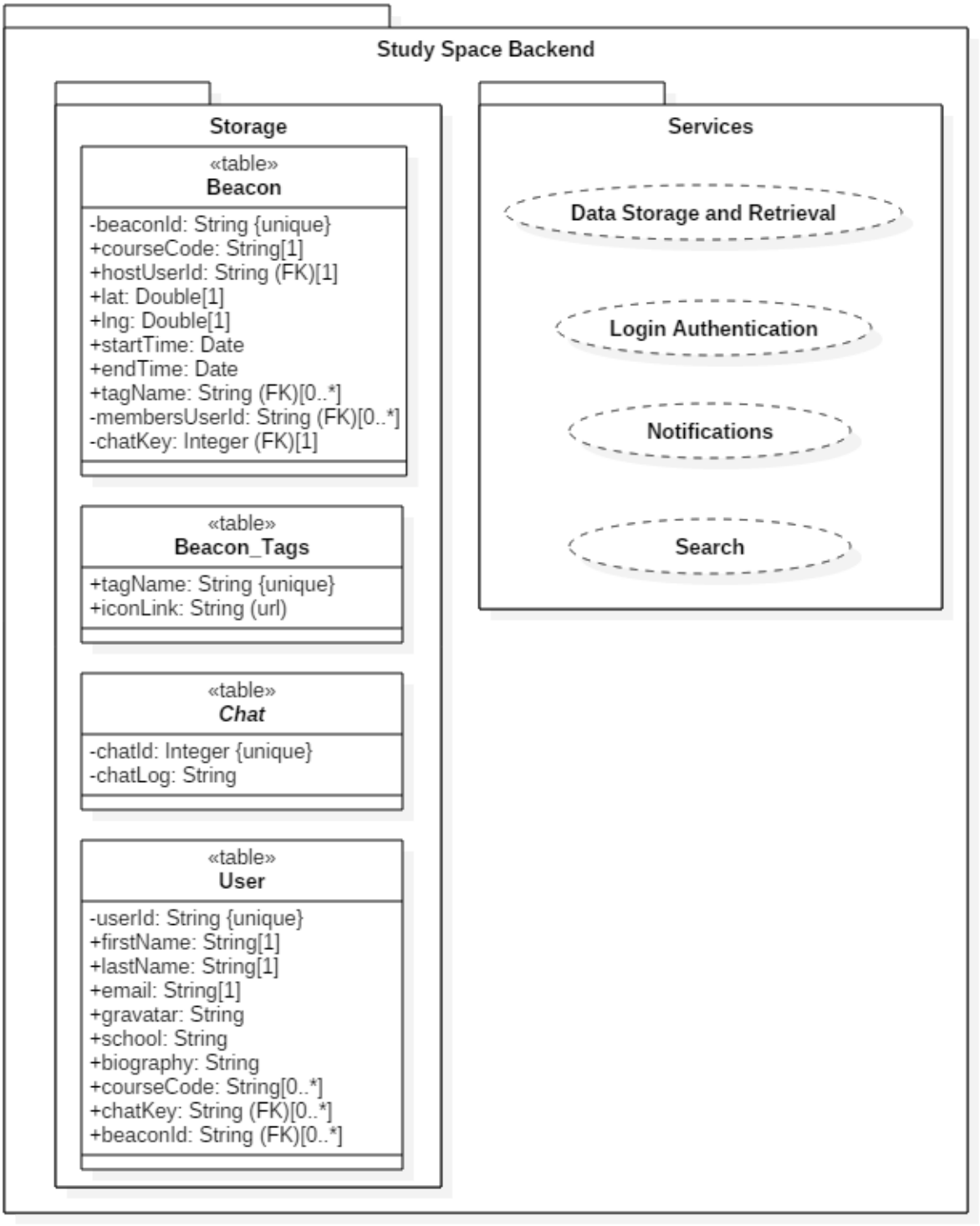
2.1.2 Study Space Application Package Diagram

The Study Space front end application is divided into application views and functions related to those views. A detailed description of all functions can be found in Section 4.0 Detailed Design.

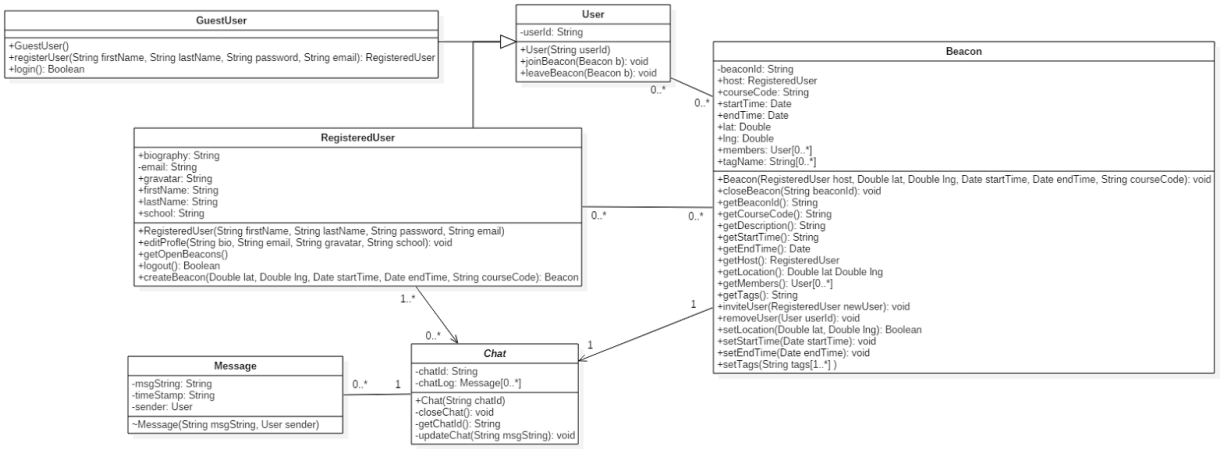


2.1.3 Study Space Backend Package Diagram

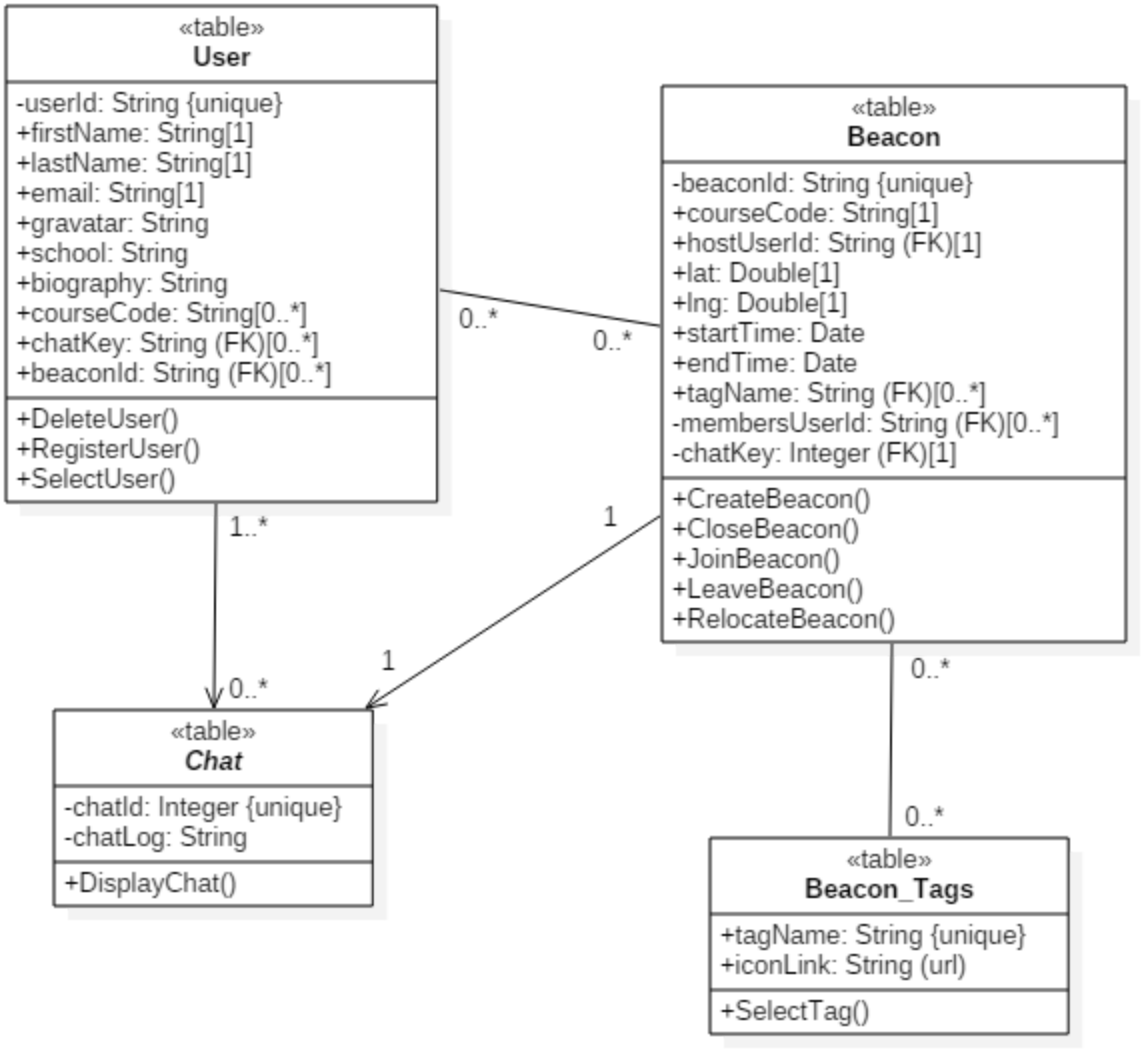
The Study Space backend provides storage and some key services for the frontend.



2.1.4 Study Space Application Class Diagram



2.1.5 Study Space Database Class Diagram



Notes:

- *FK* refers to foreign key
- The *chatlogs* will be stored as a string in a text file containing a line for each message in the format "timeStamp: sender, msgString".

2.2 Concurrent Process Decomposition

2.2.1 HomeWindow:

The main view for the site. It contains a map on the right side which allows for easy selection of beacon locations. It has fields for user to input school, course ID, location, date, start time, end time, and tags for filtering purposes. It will contain a list of active beacons that abide by the filters set which can be joined. Finally, it shall also have a login button at the top right corner.

2.2.2 LoginWindow:

Contains the fields for Registered Users to input an email and password. Also gives users the option to register for an account. The input fields will be: Email, Confirm Email, Password, Confirm Password.

2.2.3 BeaconCreationWindow:

Contains the same view as HomeWindow but the list of beacons to join is gone and the input fields are shifted down to be centered relative to the map. These input fields will be used to set the attributes for the Beacon that is being created. Beacon creation will be finalized by clicking a “Confirm Beacon Creation” button.

2.2.4 ClassDiscoveryWindow:

Contains a list of courses with their respective ratings, quick description and a link to the school page that describes the course.

2.2.5 UserDiscoveryWindow:

Contains a list of Registered Users. Each registered user listed will also show their profile name, the gravitar they set, and a description of themselves that they have also set.

2.2.6 ChatWindow:

A small window at the bottom right corner of the screen that can be used to chat with others.

2.2.7 NavigationBar:

A search bar that will be at the top of each page.

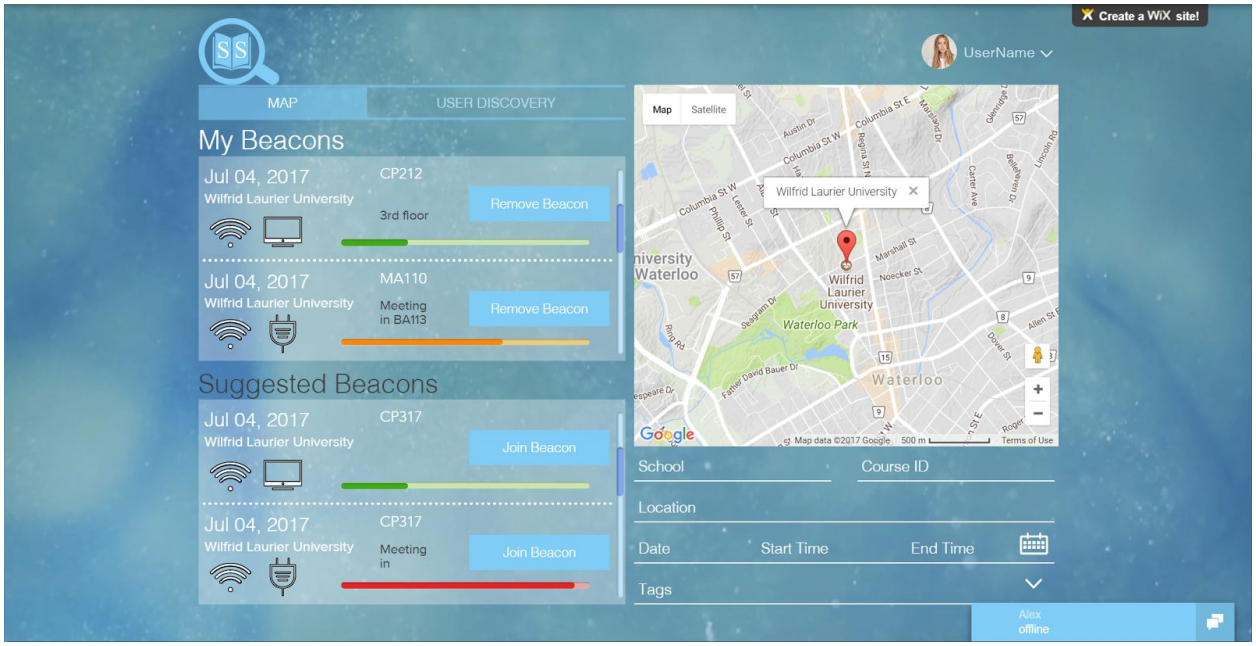
2.3 Data Decomposition

All data is handled by a real-time server hosted using Firebase. Data will be validated locally, and then sent to the server in JSON format. The server will then store the data in the database in JSON format. A unique key will be generated for each piece of data pushed to the server, and this will allow easy referencing of the correct JSON objects.

3.0 Interface Description

3.1 Module Interface

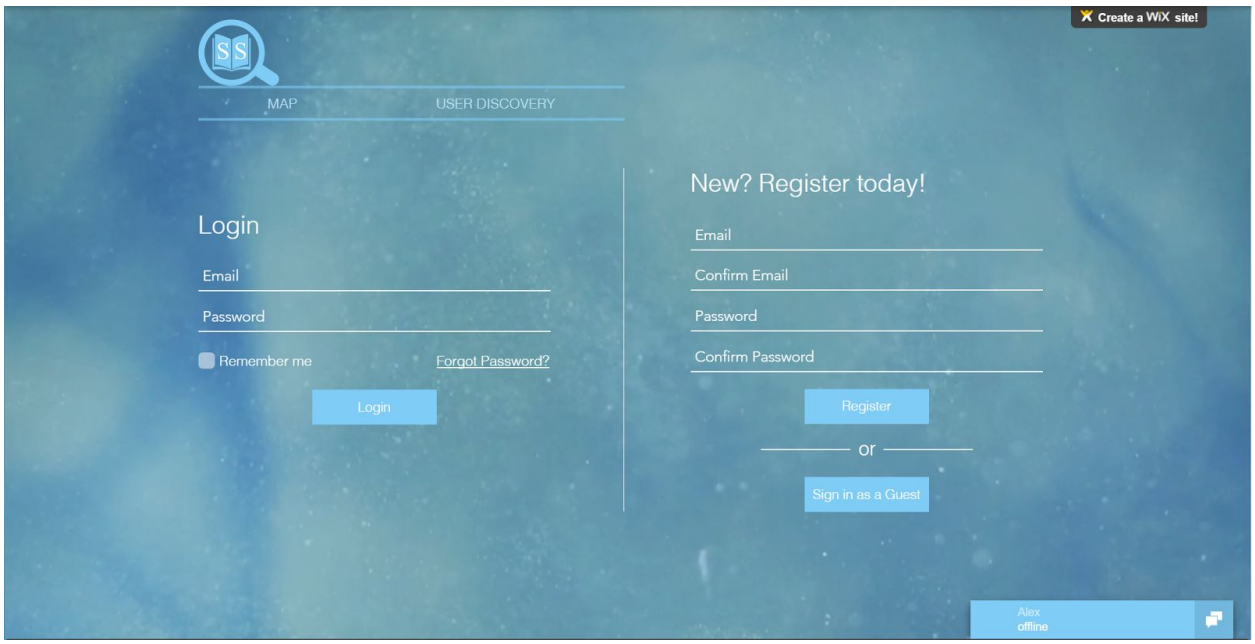
3.1.1 HomeWindow



Field	Type	Description
login-button	Input button	Links user to the RegisteredUserLoginView
search-school	Input text	School name
search-courseID	Input text	ID of the user's course
search-location	Input text	Location the study would take place
search-date	Input text	The day in the week (e.g Monday)

search-startTime	Input text	Time the study group begins (12 hour format)
search-endTime	Input text	Time group finishes studying
search-tags	Input drop-down	Chooses from a multi-check box

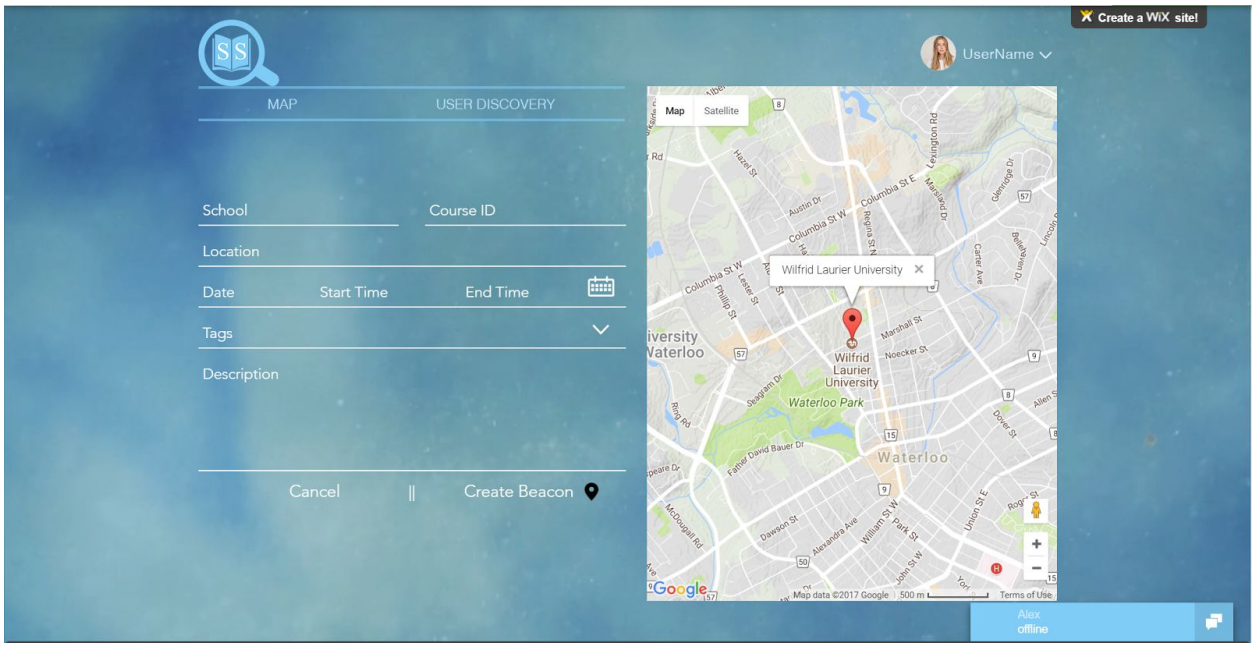
3.1.2 LoginWindow



Field	Type	Description
login-email	Input text	Registered email
login-password	Input text	Password associated with email
login-button	Input button	Control object to send authorization to provide login
register-email	Input text	Input unregistered email
confirm register-email	Input text	Re-enter email entered above

register-password	Input text	Input desired password
confirm register-password	Input text	Reenter desired password
register-button	Input button	Control object to send details to database

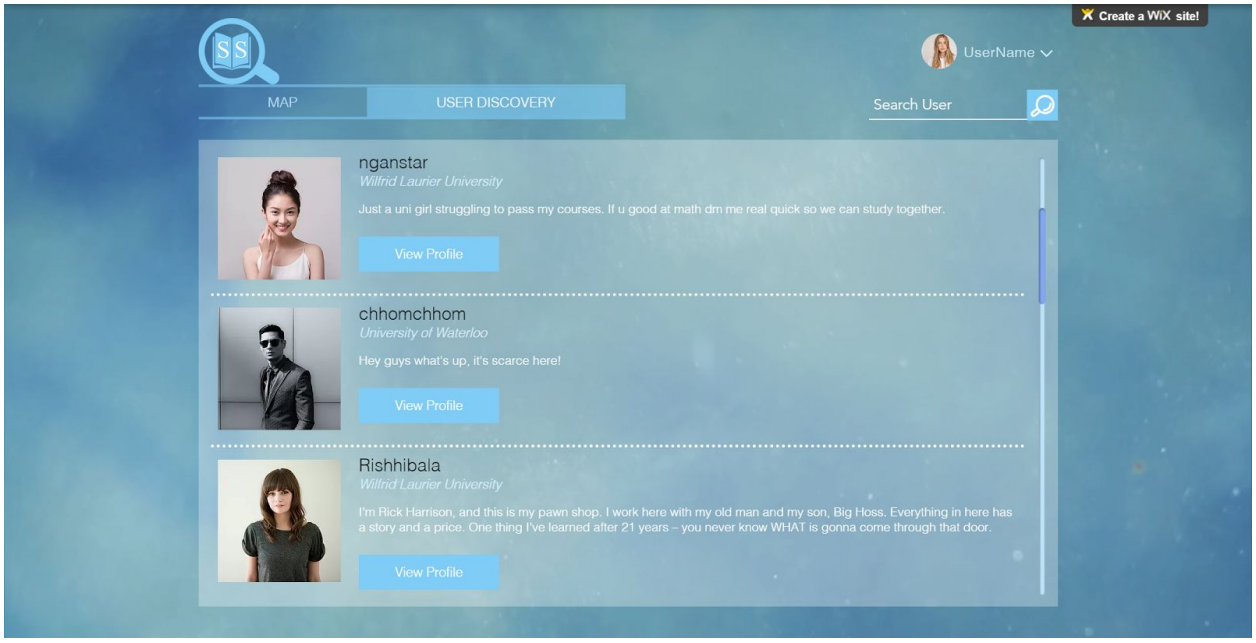
3.1.3 BeaconCreationWindow



Field	Type	Description
beacon-school	Input text	School that the user is studying in
beacon-courseID	Input text	Course that the host/user is studying
beacon-timeframe	Input select	Time of when the beacon is going to start and end
beacon-location	Input text	Location of where the beacon is going to be taking place
host beacon-button	Input button	Registered user attempts to create beacon. Redirects back to

		BeaconCreateWindow if it fails with a failed message

3.1.4 UserDiscoveryWindow



Field	Type	Description
profile-search	Input text	Search through the users by username
profile-view	Input button	Takes the user to the user's profile page

4.0 Detailed Design

4.1 Module Detailed Design

This section provides a detailed look on module design. It provides steps to produce the intended output of each module in pseudocode. This Pseudocode will be useful for implementation as it provides an outline of which can be referenced when developing the product in code.

4.1.1 User Modules

Login
Parameters: None
Description: Logs in a GuestUser to their RegisteredUser account.

1. Call ValidateUser function.
2. If ValidateUser returns true, then log user into their Registered User Account.
3. Notify user “Username or password invalid”.
4. Return to HomeWindow.

Logout
Parameters: None
Description: Logs out a RegisteredUser from their account.

1. Leave any active chats or beacons the user is currently in.
2. Log user out.
3. The user should now be a guest user.

EditProfile
Parameters: None
Description: Opens the ProfileEditWindow so user can apply changes to their RegisteredUser profile.

1. Launch the ProfileEditWindow.

SaveChanges
Parameters: String biography, String userName, String gravatar, String school
Description: Saves changes made to a RegisteredUser ProfileWindow.

1. Update the UserDB database with the changes.
2. Update the local user profile window.
3. Return to ProfileWindow.

CancelChanges
Parameters: None
Description: Cancels changes made to a RegisteredUsers ProfileWindow. (Changes are not applied)

1. Exit the ProfileEditWindow.
2. Return to ProfileWindow.

ValidateUser
Parameters: String userName, String password
Description: Validates a RegisteredUser’s credentials upon login.

1. Access UserDB to compare the stored credentials with the credentials that have been input by the user.
2. If credentials match, then return true.
3. If credentials do not match, return false.

4.1.2 Beacon Modules

CreateBeacon
Parameters: None
Description: Opens the BeaconCreationWindow which will allow the user to set the attributes for the beacon they want to create.

1. Launch the BeaconCreationWindow

EnterInfo
Parameters: RegisteredUser host, String location, Date start, Date end.
Description: Writes RegisteredUser entered info to the Beacon.

1. User enters info.
2. Call validate beacon.

ValidateBeacon
Parameters: RegisteredUser host, String location, Date start, Date end.
Description: Validates Beacon Creation.

1. Check if location is valid.
2. Check if start date is valid.
3. Check if end date is valid.
4. Ensure that end date is after start date, and is therefore valid.
5. If all is valid, call RegisterBeacon. Then launch BeaconControlWindow.
6. If any is not valid call FailedCreation.

FailedCreation
Parameters: None
Description: Notifies the RegisteredUser controlling the Beacon of failed Beacon creation.

1. Notify user that beacon creation failed.
2. Return to BeaconCreationWindow.

RegisterBeacon
Parameters: Beacon beacon
Description: Registers Beacon to BeaconDB.

1. Add the beacon to the BeaconDB.

CloseBeacon

Parameters: String userName

Description: Allows UserBeacon to be closed by the RegisteredUser who is hosting the Beacon.

1. Remove the Beacon from BeaconDB.
2. Close the BeaconControlWindow.
3. Open the BeaconCreationWindow.

RelocateBeacon

Parameters: String userName, String location

Description: Allows RegisteredUser controlling the beacon to relocate their Beacon.

1. Use the key to identify the Beacon.
2. Update the location attribute of the respective beacon in BeaconDB.
3. Update BeaconControlWindow location value.

ExtendBeacon

Parameters: String userName, String date

Description: Allows RegisteredUser controlling the Beacon to extend the time of their Beacon.

1. Use the key to identify the beacon in BeaconDB.
2. Extend the date to the date specified.

JoinBeacon

Parameters: String userName

Description: Joins a GuestUser or RegisteredUser to a Beacon

1. Use the key to identify the beacon in BeaconDB.
2. If Beacon is found (if it still exists)
    a. Add user to the Beacon's members list.
    b. Launch BeaconJoinedWindow.
3. If Beacon is not found (it does not exist anymore)
    a. Launch BeaconFailedJoinWindow.

LeaveBeacon

Parameters: String userName

Description: Removes a GuestUser or RegisteredUser from a Beacon they are in.

1. Use the key to identify the beacon in BeaconDB.
2. Remove userName from the beacon's member list.
3. Return to HomeWindow.

NotifyUser

Parameters: String userName

Description: Notifies GuestUser or RegisteredUser of an invitation to a Beacon.

1. Use userName as a key to identify the beacon in BeaconDB.
2. Notify the user of an invite to the Beacon.

3. User can accept or decline.

ApplySearchFilters
Parameters: String course, String location
Description: Applies Filters to be used while searching BeaconDB.
1. Generate a temporary filtered list from BeaconDB based on filters the user has input. Templist will be a list of userName Strings used as keys to identify beacons which follow the filter requirements.
2. Call DisplayResults

DisplayResults
Parameters: None
Description: Displays results of a search.
1. Pull list of beacons from BeaconDB.
2. Use information pulled from BeaconDB to display results on HomeWindow.
3. Show HomeWindow.

BackToSearch
Parameters: None
Description: Redirects GuestUser or RegisteredUser back to SearchWindow after a failed search.
1. Launch message for failed search.
2. Launch search window.

4.1.3 Chat

SendMsg
Description: Sends a message to a chat.
1. Generate a message object with proper msgString, timeStamp, and sender.
2. Adjust chatLog, and update the chat.
3. Use the chatID to identify the correct chat in ChatDB and append the new message to the chat.

4.2 Documentation

Angular
Angular is an open-source front-end platform used to build web applications. It is maintained by Google, and was developed with the goal of minimizing size and maximizing performance of web applications. It is TypeScript based, meaning applications developed in angular can be coded in JavaScript.

Angular has been chosen for the implementation of Study Space because it is robust and excels in the development of data-driven applications. It provides libraries that will make difficult operations such as data binding much easier. It allows for plenty of code reuse, and applications developed with Angular can be viewed on multiple platforms. Finally, Angular is well documented and easy to learn; There are plenty of resources online that can be accessed, including Tutorials on the Angular website.

MaterializeCSS
Materialize is a CSS framework that allows attractive front end design for webpages. It was created by Google, and allows a unified user experience across many platforms. Materialize will be useful in styling the Study Space web application with a modern look and feel, as well as the implementation of self-evident features.

Google Maps API
The Google Maps API is a service which allows developers to integrate Google Maps into their websites and applications. This API was chosen to implement the map for Study Space because Google is the leading innovator in web mapping technology, and it is a service that all users of the application will be familiar with. The Google Maps API also provides a plethora of documentation which makes learning to work with it much easier.

Node.js
Node.js is an open source, cross-platform javascript runtime environment for executing javascript code server-side. Node.js is particularly useful because the same language can be used on the front end and back end of development, which makes development much less

trivial in general. It is event driven while also supporting asynchronous I/O which suits the needs of the Study Space application nicely.

Firebase
Firebase is a mobile and web application development platform owned and maintained by Google. All data in Firebase databases is stored as JSON objects. For Study Space, Firebase will be used to provide a realtime database to store information on user credentials, chat logs, active beacons, and more. Overall, it is a very functional platform that will allow the Study Space team to be productive during the implementation of the project.

Version 1.1

Adjusted document based on SQA suggestions

1. Calvin Wang
2. Mitch Marino
3. Virack Chhom
4. Ethyl Chan
5. Amandeep Gogia

Thanks to SQA by

1. Jake Loftus
2. Matthew Younatan
3. Garrett Parris