IGotNxt Design Document

IGotNxt
May 3rd 2024

Team Members:
Jake Konrad
Dylan Cardoza
Charlie Hattas
Jose Estrada
Alex Alferez
Ross Baldwin

General SDD Information

1. **Online Project Notebook**
    a. https://drive.google.com/drive/folders/1LhW9oZ6ZAupK3Lm5bCqDvWYEe2UBAs4J?usp=drive_link
2. **Business Plan**
    a. https://docs.google.com/document/d/1Da39vwYn7acPMa5szihR3LkT32On5xhG6acJ1pst6M4/edit?usp=sharing
3. **Project Management Plan**
    a. https://github.com/iGotNxt/alpha
    b. https://trello.com/b/cNc3Jr7J/agile-board-template-trello (mostly used discord though)
4. **Testing Approach**
    a. Our testing strategy will entail that we run our code through whatever emulator available whether it be Android SDK or XCode. This will ensure that all the visual elements of the app actually work within the dimensions of the average phone and are simple to use. We will also be testing the load and scalability of our database by filling it with dummy data and ensuring it doesn't struggle with the load. Firebase is historically easy to scale (up to a certain point), which should be enough to support this app upon startup. We will also be using the emulator to make sure that all the information in the backend that we need displayed on the frontend is being processed correctly as we connect them to ensure that they are not working against each other. Having the emulator open as you work and save changes makes it very simple to continually see how the updates are affecting the app whether it leads to errors or just strange formatting decisions. This approach will make sure that no bugs or defects go undetected for too long and halt the development process.
5. **Sprint Presentation Links**
    a. Sprint 1:
        i. https://drive.google.com/file/d/1dHuPlxBIYoI7ieJDxI090omw5QMRpHSd/view?usp=sharing
    b. Sprint 2:
        i. https://docs.google.com/presentation/d/1k2K_A7ysc6mAdWLIzbqet8PGUMXLvCUkie90YVEQYho/edit?usp=sharing
    c. Sprint 3:
        i. https://docs.google.com/presentation/d/1AsVWxn51jmCAn1CCq3OZOqlzPt1B34FTNA9hDvTOYwE/edit?usp=sharing
    d. Sprint 4:
        i. https://docs.google.com/presentation/d/1wAjFiXdTZMg2Q6ppi225C58nZO8oDa0ibYCuL9hk_IU/edit?usp=sharing
    e. Sprint 5:

i. https://docs.google.com/presentation/d/1M5QYK2kid2mFgDOiIEcStupa7o2jLnkEoGEimhXK6hE/edit?usp=sharing

f. Sprint 6:

i. https://docs.google.com/presentation/d/1it2Y-2ji6WluaJam8rtZonHk8int48P76_w_jHFeOpc/edit?usp=sharing

Design Overview

IGotNxt is a phone application where users can browse (based on location) spaces and see openings to make reservations. It was originally limited to gyms but has expanded to more varied spaces like coffee shops and restaurants. You can see openings and book directly through the app. You can save spaces of interest to a personal list. If you are an owner of the app you can update your business info and will update on the user side. Users of the app looking to rent can create an account and immediately start looking for places to look either searching or using some of our custom filters. From there you can book spaces. Business owners can do the same and immediately add information to their profile for users to browse. This model allows the business owner to communicate information to renters of spaces and those potential renters to make decisions on if they want to have an event there or save it for later. This arrangement is meant to encourage users to interact with local businesses and be able to foster community events or just a night out with a large party.

Design Components

The major components of the app on the renter user side would be the search/filter features on the spaces page. Here, the information for the page is connected to a Firebase database that includes all the important information including: photos, address, name of business, capacity, filter information etc. This allows users to be recommended businesses based on their location as well as be able to directly search for businesses they already have some information about. This same database information is important for the receipts/saved spaces page as well because any of the saved spaces will require the same information to be shown to the user. The upcoming obligations will also need to be based on reservation data read from the database after the booking information is written to the database.

The major components of the business owner's users side is the business details page and the analytics page. The business details page allows for input from the user to specify qualities of the business that will be reflected on the spaces page on the user side. This means that it is connected to the database in a similar way but with the capacity to write data and not just read it. The analytics page only reads data from the user side like views, amount of reservations, and the amount of money they are making which is reflected in a bar graph for the owner to view. This will only be reading information based on the interaction from the renter side of the app.

We have navigation components for both bottom tab navigation for both options (renters/owners) as well as location based functionality based on some API calls. The information above is merely covering our most important screens for the apps as well as their functionality.

Software Flow

There are many custom inputs for the users and business owners to use and customize their experience. Obviously, both can sign up/sign in with email and specify which type of user they are. Beyond that the business owner can edit their business info and see analytics. The renter side can more manipulate filters and searches. These filters can account for distance, cost capacity and other relevant details that a renter might take into account when deciding upon a location. All of the locations will also be based on the location of the user after they give the app permission to track location. This helps foster that sense of community that the app is looking to build. From these business pages they can also book a time directly from that page for whatever event they are interested in booking. Beyond that, there is a spaces page on the renters side of the app where they can save businesses that seem interesting for them to rent and view them later. This will operate in a similar way to the search page where it will be a list of business pages that will expand upon clicking, letting renters book directly from there as well. On a tab on that screen, renters can also see upcoming obligations and relevant information like date and time, party size, and the cost of the event. Both renters and owners share a similar settings screen where in the future they can personalize the app based on their needs and wants.

On the owners side, there is an analytics screen where owners can track views, revenue, and other helpful statistics to see how the app is benefiting them. This will help reflect the app's ability to draw people in and build a sense of community around a business. On the profile screen, businesses can input all important information that they want output to renters like name, address, capacity, and images to promote the restaurant. This will give the owners control over what kind of events they would like to hold at their business and provide simple promotion as well. Additionally, this is where business's create their spaces and manage their times. Business's also have a reservations tab that allows them to see all active or pending reservations made at each of their spaces. They can also switch over to a camera view in order to scan in users.

Figure : IGotNxt User Flow Chart

Additional Design Considerations

1. **Database Design**

The database is designed in a way that intends to make it easy to pull specific data from while still supporting deep interaction. It is a "noSql" database, instead utilizing a collections/documents system. A collection is simply a container that can hold documents. These documents can have any kind of field, from simple strings to complex maps or even more collections. Since each relevant set of data is a high level collection, the way they interact is through the references found in their document fields. For example, a "booking", as it is called in the backend, contains not only start and end timestamp data, but references to the user and space that the booking is for. There are pros and cons to this system, a major pro being ease of understanding and use. For our circumstance we decided that it would be best to prioritize simplicity.

Aside from the aforementioned booking collection, the other relevant collections are for the users, businesses, and spaces. When creating an account, depending on the account type we assign its data to either a user or business. The user side is quite simple at the moment as we used it primarily for authentication/testing purposes on the user side. The business collection is quite complicated and contains a lot of moving parts. Most importantly, the way images are stored in the database is through Firebase's cloud storage system. When uploading an image via the business profile editor, first the image gets uploaded to the storage and then a reference link to that location in the storage is created as a field in the business's "images" array field.

Another important thing to understand is the nature of "spaces". A business can create any number of spaces, and doing so delegates them to their own collection. The most important thing to understand about spaces is the way space availability is handled. At the moment, there is no automated response to reset space availability and the bookings that are associated with it. Instead, we have prototype buttons in place that allow for this reset and the deletion of spaces as well. Availability is its own collection nested within every single space document. Inside, the individual documents represent blocks of available time. When a user makes a reservation with inputted times, that block of time gets "carved" out of the availability block it is pulled from, and two new blocks get made to represent the remaining availability left.

One final important thing to mention is the way in which users check in. We generate QR-codes based on the document ID that corresponds to a user's bookings, so when they go to check in, businesses can simply scan this code and register that this particular booking is "activated".

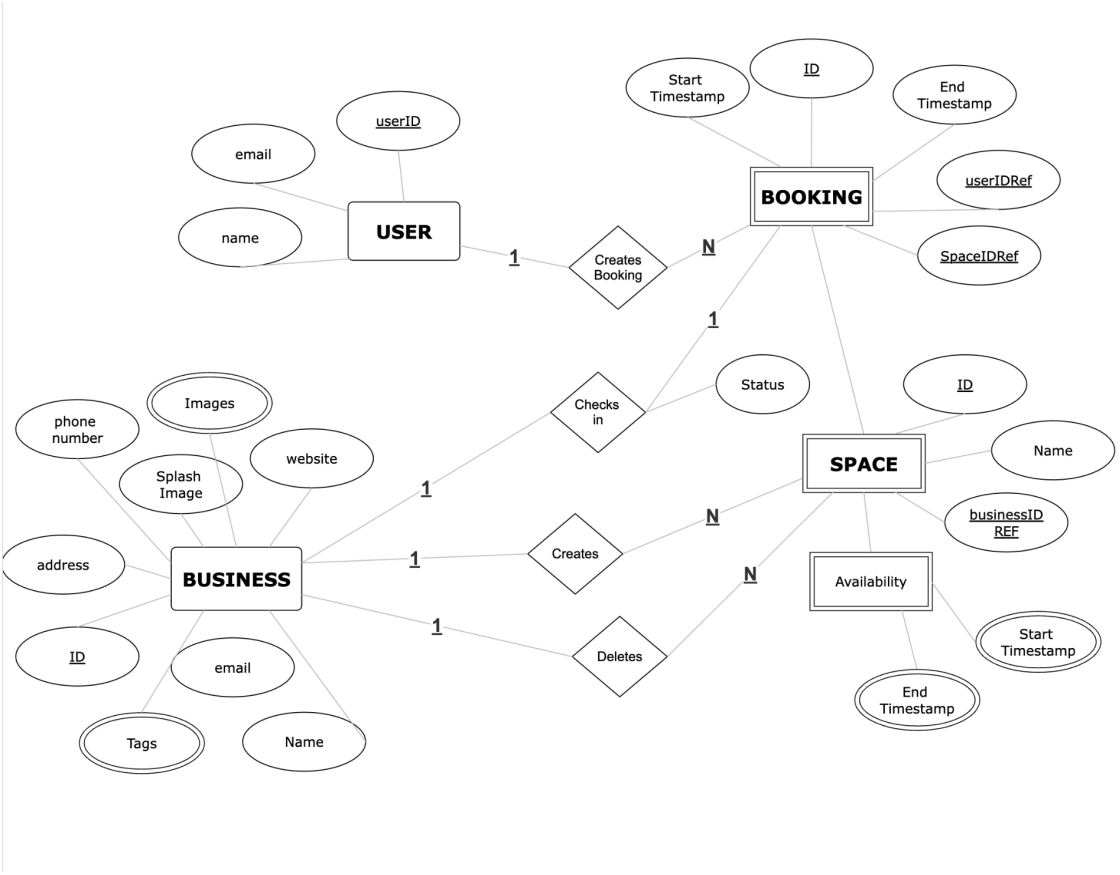


Figure: Database Design

2. User Interface Design

The user interface is tailored individually for the renter side and the owner side. They both have bottom tab navigation to navigate between their respective pages. Both have a page with a tab at the top of the screen to toggle between pages as well.

For renters, the search screen UI has a search bar and an interactive list of businesses in your area. There are filters above the list to further narrow down the options as well. For the spaces page there is the top tab to swap between the upcoming events screen and the saved spaces screen. The saved spaces screen will operate very similarly to the list on the search page. The receipts page is just a list of the basic information one will need for their reservation. The settings Page is a clean list of features that react to touch but currently don't offer any functional changes for the app.

For owners, there is a bottom tab navigator to get between pages. The analytics page offers a scrollview of the saved data like profits per month in a bar graph, squared with data, and reviews at the very bottom. For the business info page, it offers a bunch of custom inputs for the business owner to enter in text information and even images at the bottom. The settings page for the business side operates the same way as the user side as of now.

We wanted the app to feel reactive to user input whether it be tapping a button or entering text and we think we accomplished that fairly well.

3. Input/Output Specifications

For both renters and owners, they can input their account info to sign into the app as well as when creating an account. Also, if they have forgotten their password there is a screen where they can input their email and they will receive an email for them to change it.

The input for renters is defined by them interacting with business pages and viewing or deciding to book them. They can also save businesses for themselves but this doesn't do much for the business. When booking, the information is connected to the business side of the app so they can track it in analytics and keep track of upcoming reservations. Also, on the search screen, they can input names into the search bar and apply filters that affect the businesses that are outputted to them on that screen.

For the owner side, they can input information about their business and images which will be output onto the user side if the renter is in their area. This will help renters make decisions on booking and the business can track the outputs from the analytics page and make changes accordingly. Both renters and owners have a settings page which at this point is only purposed for signing out of the app.

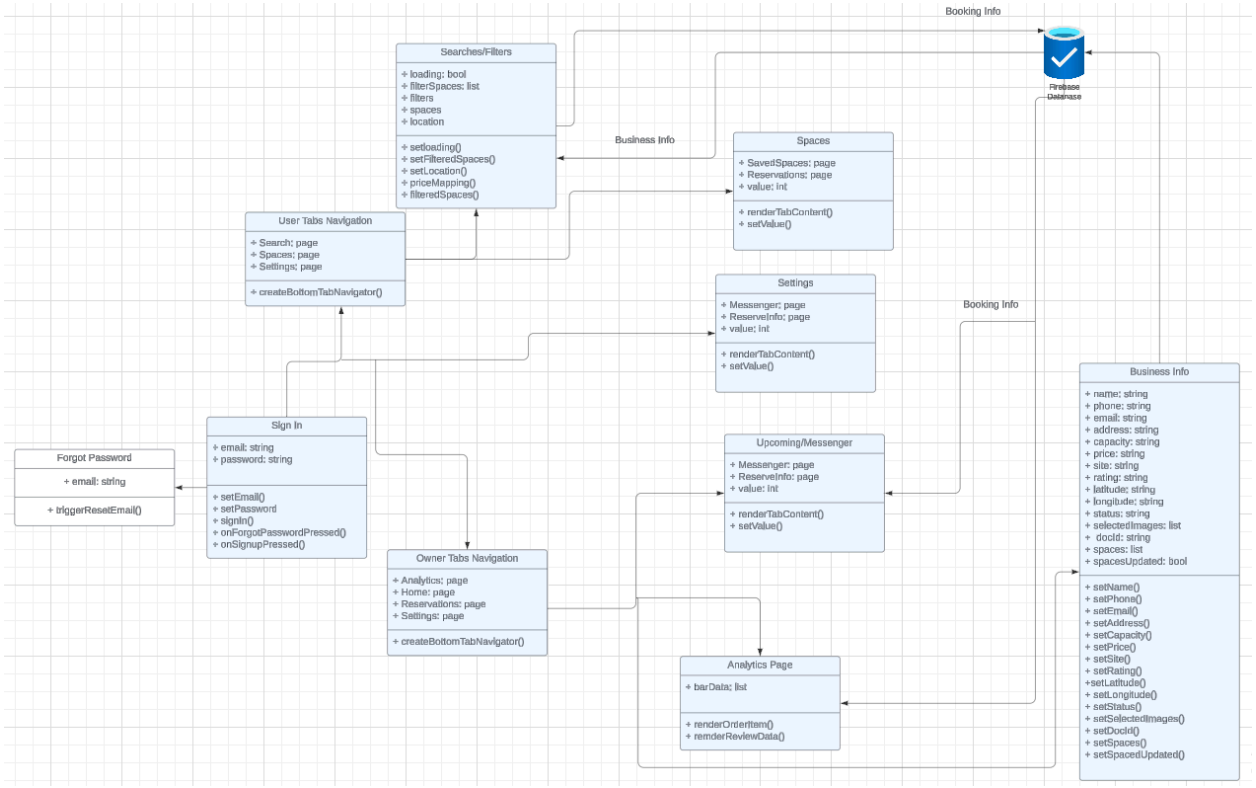
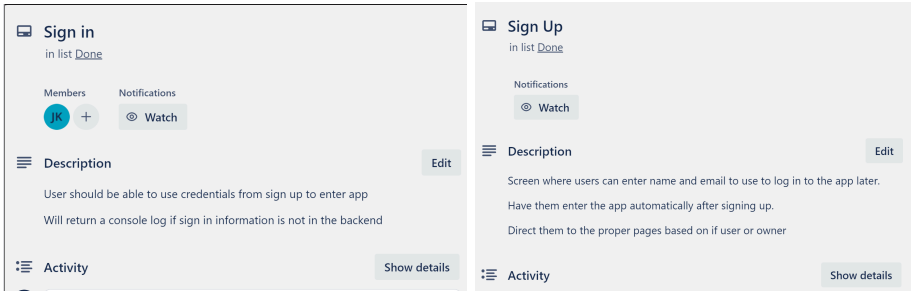
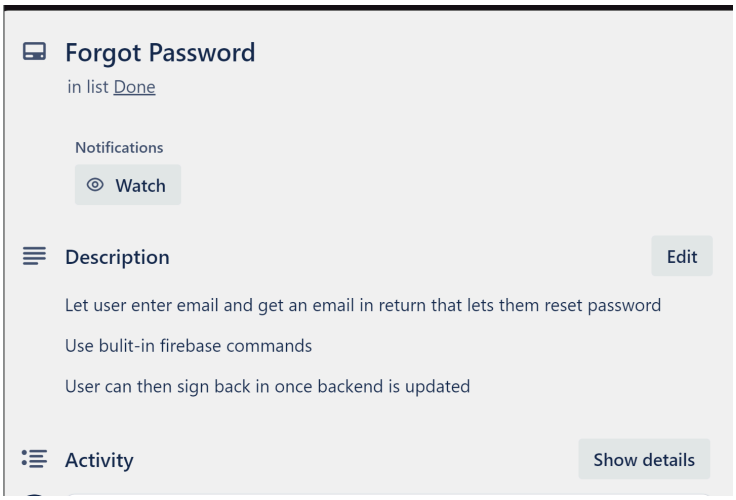


Figure: UML Diagram for UI and Input/Output

4. Features/Modules

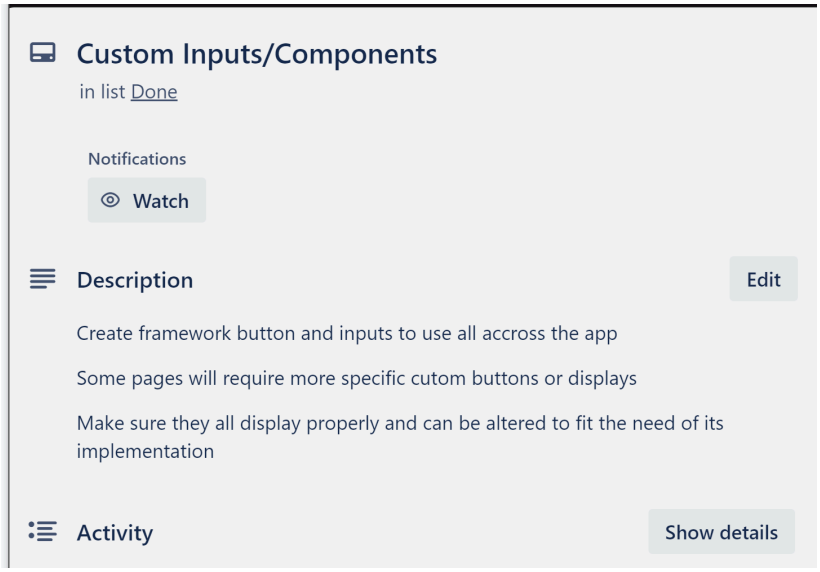
a. Sign In/Sign up

i. Sign in and Sign up is a screen where users can use pre-made credentials to log into the app as a renter or owner and be validated before entering the app.
ii. https://github.com/iGotNxt/alpha/tree/MapView/HelloWorld/src/Screens/AuthScreens
iii. 

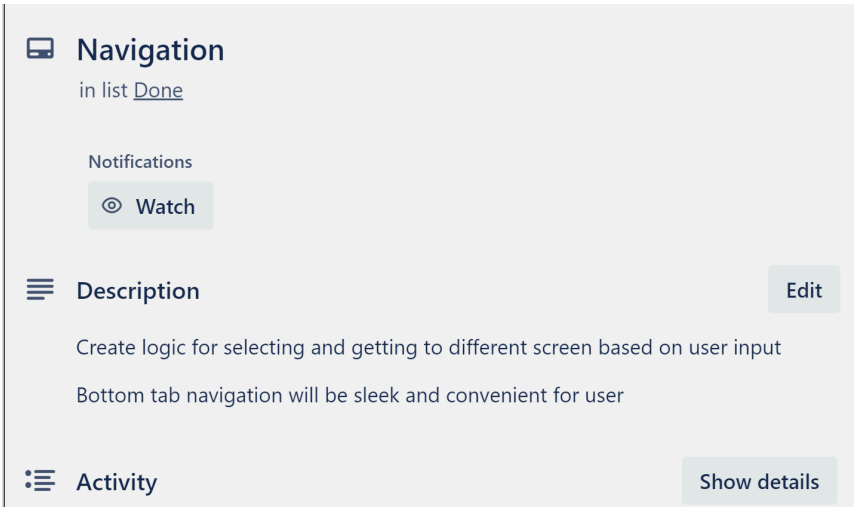
b. Forgot Password Screen
i. Users can enter their email and get sent an email to reset their password.
ii. https://github.com/iGotNxt/alpha/blob/marq6/HelloWorld/src/Screens/AuthScreens/ForgotPassword.js
iii. 

iv. Testing:
1. Enter our personal emails into the system after already having an account and use the link sent to us by email to update our password. Once our password is updated, attempt to login in with the new credentials. If successful we can move on.

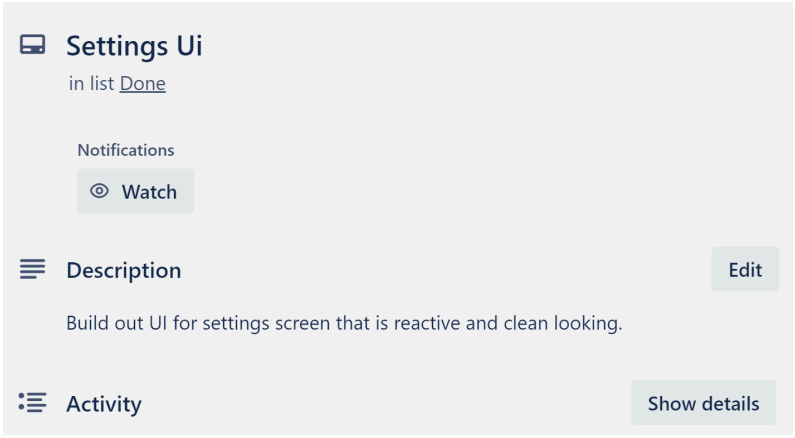
c. Custom Inputs
i. Buttons, text inputs, and etc. Meant for the users to interact with to navigate or read/write data to the database.
ii. https://github.com/iGotNxt/alpha/tree/MapView/HelloWorld/src/Components

iii. 

iv. Testing:
    1. Make sure that all are displayed properly in terms of frontend. If information is connected to the backend, make sure that the inputs are updating the backend and displaying properly where necessary.

d. Navigation
    i. Create navigation logic to use for all our pages, a bottom tab navigator being the main source of selecting different pages.
    ii. https://github.com/iGotNxt/alpha/tree/MapView/HelloWorld/src/Navigations
    iii. 

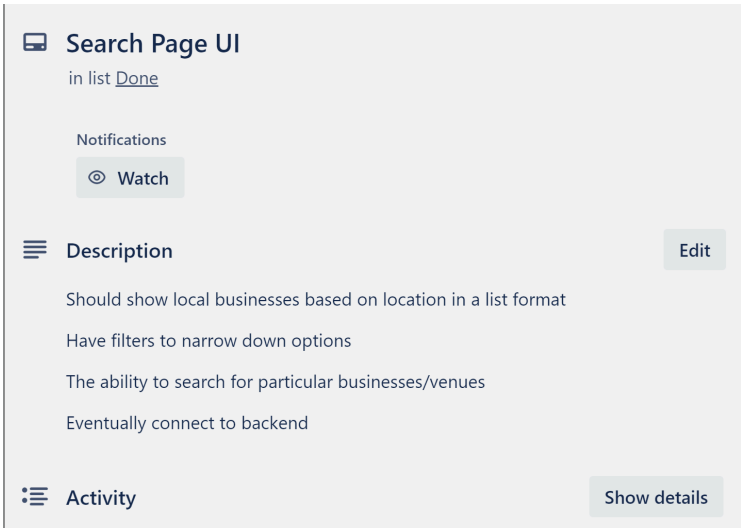
    iv. Testing:
        1. Make sure that navigation between screens is working correctly in the emulator. Ensure that the user is being directed to the right screen based on their inputs and that is not confusing in terms of operating with the navigation.

e. Settings Page
    i. Frontend for a basic settings screen that looks and operates the same way for renter and owner. This is the screen they can use to sign out.
    ii. https://github.com/iGotNxt/alpha/blob/marq6/HelloWorld/src/Screens/Account.js
    iii. 

    iv. Testing:
        1. Make sure that the page feels reactive. Not looking for much functionality so beyond that just make sure all the elements react to interaction in a satisfying way. Make sure that the sign out buttons at the bottom of the page send the user back to the login screen in the emulator.

RENTERS

f. Search/Filters
    i. Page where businesses appear in a list where they can be clicked to get more information on a business and even book a time right from that page. Filters and search bar pull information from the user based on location and using filters can narrow down the options.
    ii. https://github.com/iGotNxt/alpha/blob/MapView/HelloWorld/src/Screens/Search.js

iii. 

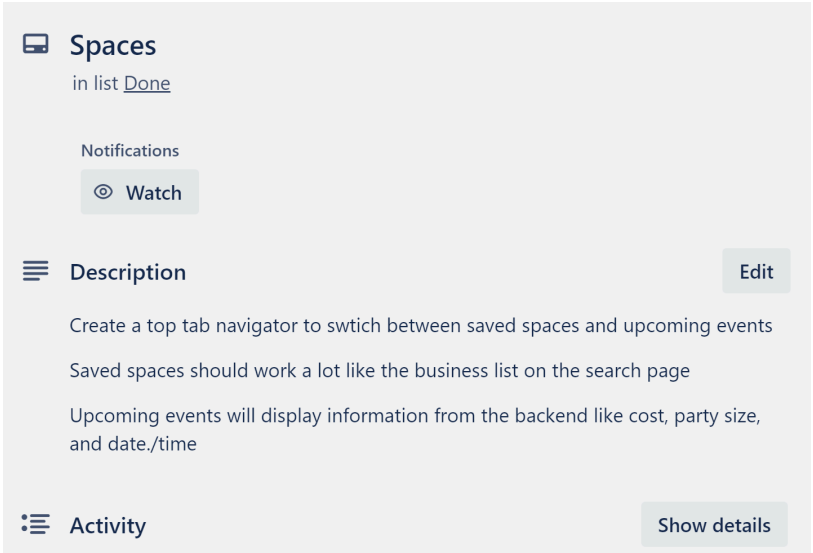
iv. Testing:

1. Make sure that the location functionality is working and is giving the user businesses in their area in the original list. From there, apply filter(s) to ensure that they are all interacting together properly and not creating any bugs. After that, use the search function and make sure that it is filtering businesses in a similar way but based on the user input.

g. Spaces Page

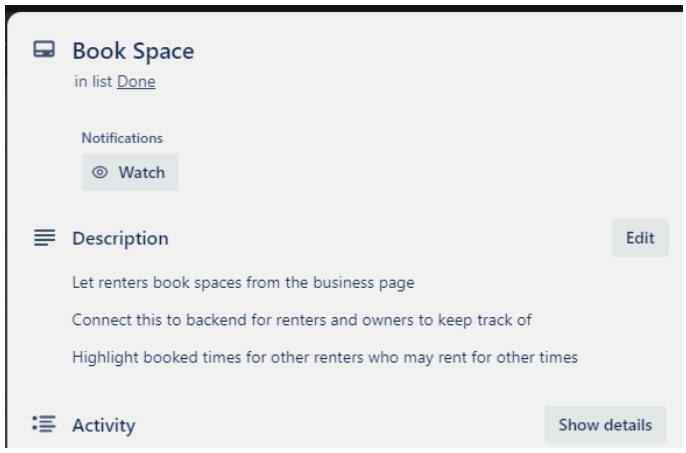
i. Here renters can see saved businesses for potential later rental as well as use a tab at the top of the page to look at future obligations and important details.

ii. https://github.com/iGotNxt/alpha/blob/marq6/HelloWorld/src/Screens/Spaces.js

iii. 

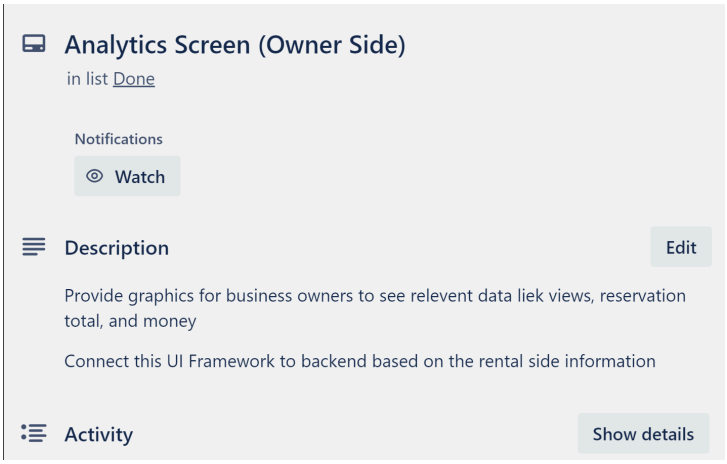
iv. Testing:

1. Make sure the proper pages are being displayed when using the top tab navigator. After that, make sure that saved spaces functions in a similar way to the business list from the search page, just without filters and searching. Being able to click into businesses and find more information as well as booking should all be possible. After that make sure that the information being displayed for upcoming events is coming from the backend and being displayed properly within the UI.

h. Book Space
   i. Allows the users, once in the business details screen, to book times for events and other reservations. This will be reflected in the backend and will be visible to the owners
   ii. https://github.com/iGotNxt/alpha/blob/MapView/HelloWorld/src/Screens/BookSpace.js
   iii. 

OWNERS

i. Analytics
   i. Owner screen that tracks relevant data collected from the renter side like views, reservations, etc.
   ii. https://github.com/iGotNxt/alpha/blob/marq6/HelloWorld/src/Screens/OwnerScreens/AnalysticsScreen.js

iii. 

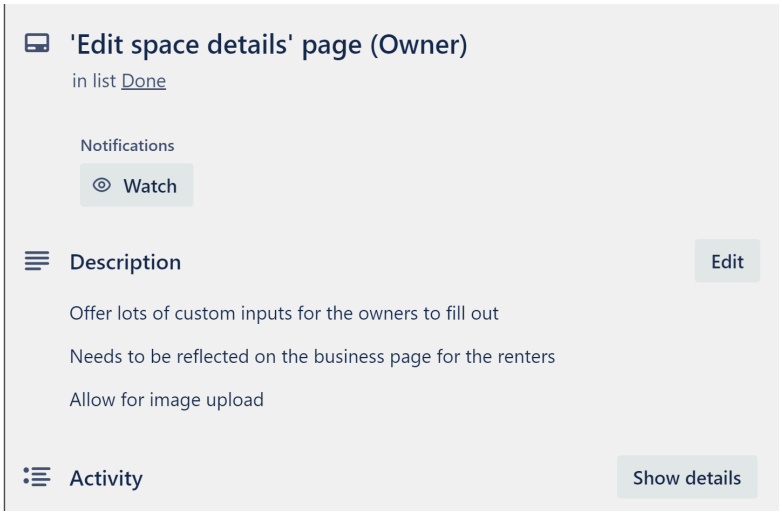
iv. Testing:

1. Make sure that the UI is working properly with the scrolling and different elements involved. Beyond that, once connecting the backend, ensure that the information is updated properly from the database and reflected accurately for the owner.

j. Business Details Page

i. Page where the owner can input information about the business that they want to be shown to potential renters on the business pages.

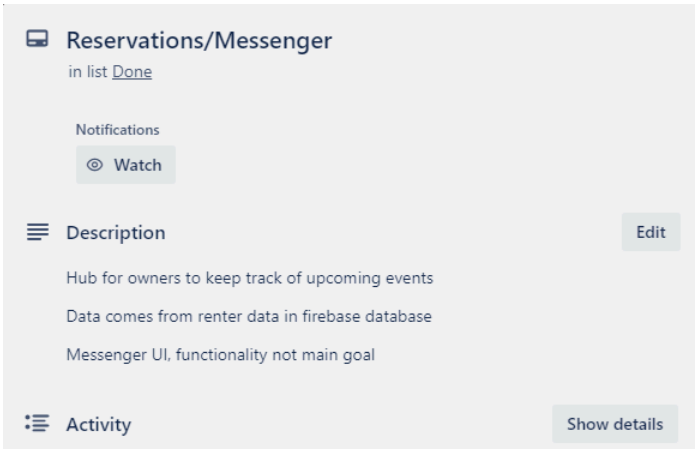
ii. https://github.com/iGotNxt/alpha/blob/MapView/HelloWorld/src/Screens/OwnerScreens/OwnerHomeScreen.js

iii. 

iv. Testing:

1. Through the emulator ensure that once the information is input by the business that it is updating into the firebase system. You can then check from the renters pages to see if it is updating properly but if it's all being called from the proper points of the database it should be working correctly.

k. Reservations/Messenger

i. This is where owners can track reservations and the frontend for a messenger is built out. Uses a top tab to navigate between the two.
ii. https://github.com/iGotNxt/alpha/blob/marq6/HelloWorld/src/Screens/OwnerScreens/OwnerReservations.js
iii. 

iv. Testing:
    1. Through the emulator, ensure that the UI for the tabs, messenger, and upcoming events is clear and fits the page. Make sure that all the UI is reactive and that the information from the database is formatted correctly for the upcoming page.

(Note: Our discussion about features were usually on discord voice call so we don't have recorded messages for every feature.)

Going Live Plan

We are going to hand off permissions to our database to Monte's partner Omelio so he can start to understand the structure of our backend. He has been a part of our github repo throughout the process so he already has access to the work already completed in terms of our frontend. We will stay in contact and ensure that any questions or concerns are adequately responded to and can keep up to date on the building of the app. After our final presentation, we will work on getting the app posted to testflight or at least giving Omelio all the information he needs to get the app posted to Testflight or another applicable testing application like the one built into the Expo platform. There Monte can develop useful data on what his prospective customers will want from the app and develop further requirements from there.

There are relevant links required for understanding the nature of the project currently. This resource [https://docs.expo.dev/build/introduction/] outlines the steps necessary to deploy. It merely requires the necessary credentials as well as a "native project". Because we have been operating in a managed workflow form of Expo React Native, it has made deployment a lot harder. First, what needs to happen is for a development build to be created. What this entails is that a lot of the lower level processes become exposed. It requires a more to get working which is why we stuck to the managed workflow. Our goals were to fulfill the intended core "loop" which involves business's creating/managing spaces along with the bookings that users can

search for and create. Here is a relevant link to begin this development build [https://docs.expo.dev/workflow/overview/#android-and-ios-native-projects]