

# Internal Design Document - Fat 6

Project Manager: Aidan Clarke Scott

Team Members: John Chapman, Willem van Doorn,  
Christopher Cliff, Matt Wilson, Sasha Maximovitch, Shane  
Labelle

February 8, 2022  
Version 1.1.

## Revision History

Date	Version	Description
02/08/2022	1.0	Creation of Document
03/30/2022	1.1	Removed Outlook API things and updated DynamoDB and endpoint documentation

# 1 Introduction

## 1.1 Overview

---

HSBC's workforce is 80% remote, which poses new challenges for connecting coworkers. With teams constantly in Zoom video meetings, it can be difficult to know where individual team members are. HSBC needs a new solution for managing Zoom meetings to increase productivity online and within the Zoom platform.

This project will help streamline the online collaboration process by mimicking the office environment, bringing more natural office-like interactions into the virtual space. Our plan is to build a dashboard that allows teams to easily manage and show current and upcoming Zoom meetings of team members or colleagues within the organization.

## 1.2 Goals

---

### Primary Goals

- Be able to see what Zoom meetings coworkers are in currently.
- Have the ability to create favorites lists of users and call all the users on that list.
- Easily assemble group meetings in one click.
- View archive of meetings from the past.
- View information of future meetings in calendar view.
- Alert users with browser notifications of upcoming meetings.
- Have a modal UI popup that appears with a link to the meeting when it's time for a scheduled meeting.

### Non Functional Goals

- Ability to handle concurrent usage of up to 20 users.
- Response times for an individual search should be less than 3 seconds.
- The app can accommodate up to the maximum number of Zoom users.

### Stretch Goals

- Integrate with Outlook to get details of recurring meeting times.
- Have links to recorded meetings available in the calendar view.
- Have the recorded meetings playable in the browser.
- Notify the invitees by email if they are not able to be called or are not in the Zoom room after 2 minutes.
- Use Infrastructure as Code (IaC) to provision infrastructure using Terraform.

## 1.3 Assumptions

---

Assumptions	
Type	Description
Architecture	Solution will be hosted and available throughout development.
Budget	We will experience minimal costs for building and testing our application, and any unforeseen project costs we incur will be covered.
Design	Employee database must be maintained as a hierarchy.
Methodology	The project will follow a mix of the Waterfall and Agile model.
Organization	The company has an internal employee hierarchy, with one person at the top, and everyone, besides the CEO, has employees above and below them, and there are no independent trees.
Technical	Calendar view shows only your meetings by default, but will allow users to select another user to view their meeting schedule.
Technical	A user will be able to see the location of all employees of the organization.
Technical	Security and authentication for Zoom recordings is handled by Zoom.

## 2 Technical Details

### 2.1 Programming Environment

---

**Programming Languages:** Typescript (for both frontend and backend), NodeJS (backend)

**IDEs Used:** IntelliJ, Visual Studio Code

**Tools:** Cloudwatch, Lambda, Amazon API Gateway, CloudFormation

**External APIs:** Zoom API

**Databases:** DynamoDB

**Source Control System:** Git and Github

**UML/Diagram Tool:** draw.io

## 2.2 Production & Test Environment

---

- Frontend hosting (Heroku), backend (cloud AWS), middle-end hook handler (Heroku)
  - API Testing: Manually with Postman (9.12.2)
- Backend testing through wscat (5.1.0)

## 2.3 Software Architecture

---

### Summary

The software architecture of our project can be split up into two main structures consisting of a client which runs the web application on a browser and the various cloud services which are used by the application in order to function. The client hosts the frontend of the application, which has a user interface built with ReactJS. The client application depends on the services provided by AWS for authentication, managing the user database, and getting information about Zoom meetings. Within AWS the client interfaces with the Amazon API Gateway which forwards requests to the appropriate AWS Lambda function. These functions include a Data Handler which connects to the database (DynamoDB) for accessing user and group data. An Authenticator which interfaces with Firebase for performing user authentication. As well as Zoom API Adapters which handle calls to the Zoom API respectively.

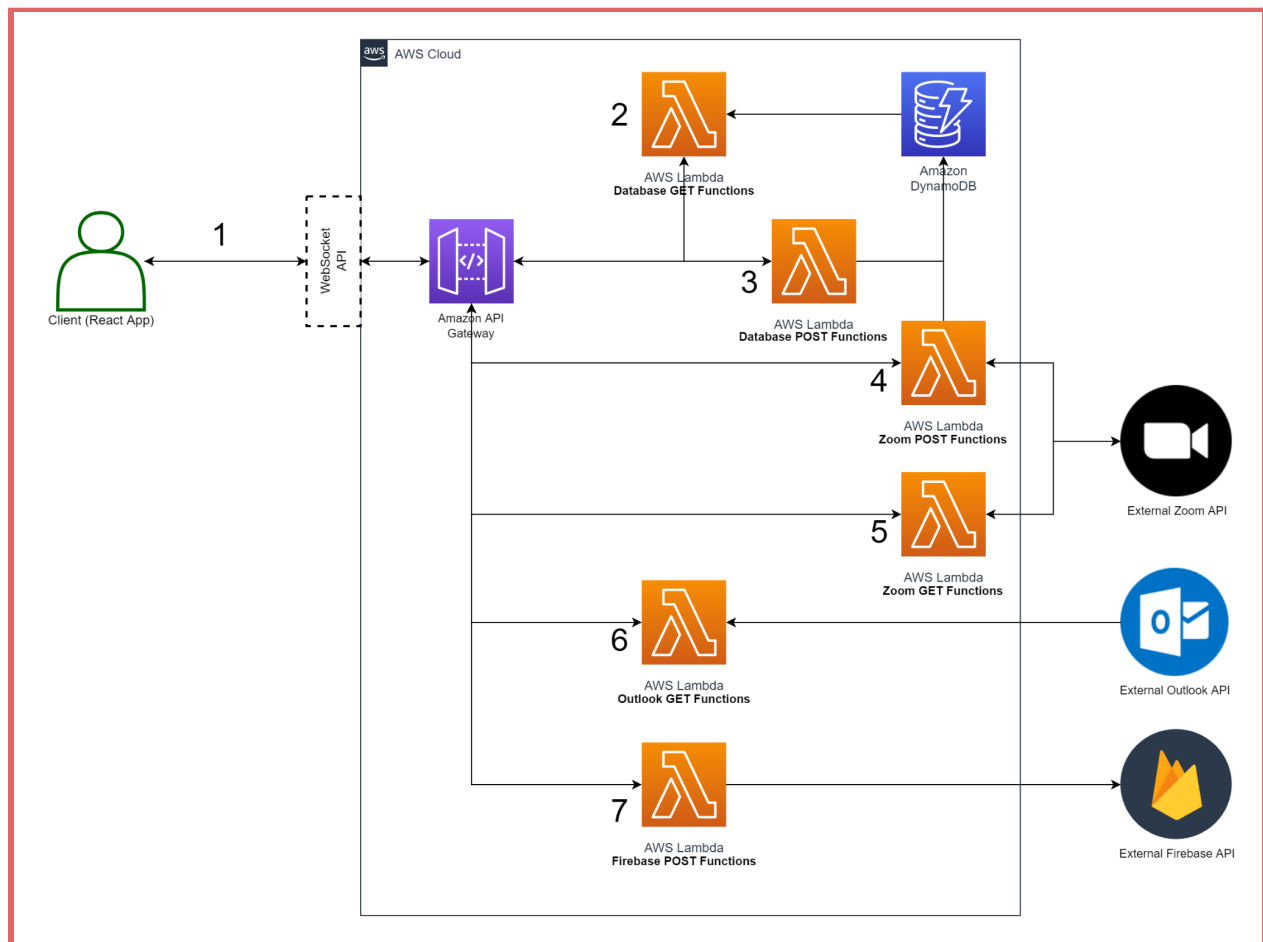
### API

In general, our API has four main stages: the client WebSocket requests the Amazon API Gateway, the Gateway's interactions with our various Lambda functions, and the Lambda functions' interactions with external APIs and other AWS systems. The diagram below shows a simplified version with the general 'groups' of Lambda functions that have similar functionality; more specific endpoints are specified below. In the diagram, there are 7 labels (1-7), which are described as follows:

1. The WebSocket API is included in the Amazon API Gateway, which handles the connections between the react app (client) and our backend (Lambda functions). The WebSocket API is used in order to avoid our client having to poll for updates regarding changes that will need to be represented in the frontend. For example, if the logged-in user is added to a new Zoom meeting, that change would need to be updated and represented in the frontend.
2. There are a subset of Lambda functions that will involve getting information from the DynamoDB database, these are functions that involve displaying information in the frontend based on information that is stored for the logged-in user. Examples of these functions include getting meeting participants, and meetings the user is invited to, getting the user's previously saved group information, and all other functions involving getting information stored in the DynamoDB database.
3. These subset of Lambda functions include those involving adding new information to the database. Examples of these functions involve creating new user groups, adding new users etc.
4. The Zoom Post subset of Lambda functions are involved with all functionality that involves sending information to the external Zoom API such as creating new meetings.

5. The Zoom Get subset of Lambda functions are involved with all functionality that involves getting information from the external Zoom API such as getting meeting information.
6. **Deprecated:** These are the Lambda functions involved with getting information from the external Outlook API. An example of this would be getting recurring meeting information from Outlook. (Outdated, not used)
7. Lastly these Lambda functions are responsible for handling user authentication with Firebase, mainly verifying user credentials and logging in.

*Note: the Outlook section of this diagram is now deprecated*



To view how the backend fits into the whole application, a component diagram is available in section 7.2. (This diagram is out of date, we do not have any outlook functionality in our app currently)

## 3 Data & API Design

### 3.1 DynamoDB Tables & Schemas

---

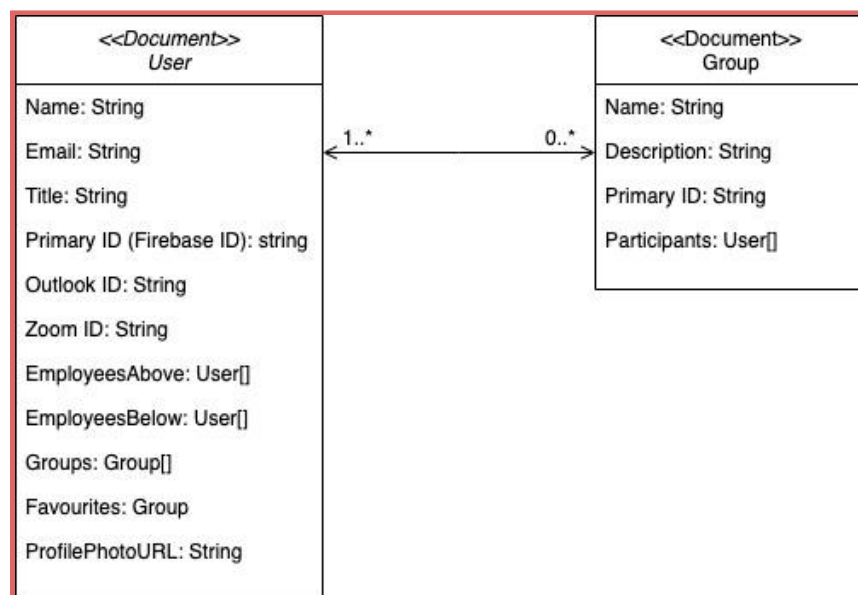
**Users:** The Users DynamoDB table stores information about each of the users, and their various identification for all of the services we incorporate into our app. For identification, we store the user's FirebaseId, ZoomId, and the ConnectionId used by the WebSocket to send information to the correct client (whoever has logged-in as that specific user). We also store information pertaining to the specific user, such as their name, rank, profile picture, and whatever groups of favourites they end up making in our app.

The partition key for the ActiveMeetings table is the *UserID*.

**Groups:** The Groups DynamoDB table stores information about each of the user-created groups. Information here includes, group title, description and members. Each of the groups is defined by a GroupID, which is referenced in the Users table, as part of a given user's list of groups.

The partition key for the ActiveMeetings table is the *GroupID*.

*Note: This UML diagram is now deprecated, with the new design of our Dynamo tables*



**ActiveMeetings:** This DynamoDB table stores all of the currently active meetings. This is required, as we need to store information for our App that the Zoom APIs do not natively supply. This is mainly the current participants of a given meeting, but we also store information such as the host's rank, and the invitedMembers for scheduled 'active' meetings.

The partition key for the ActiveMeetings table is the *MeetingID*.

**ScheduledMeetings:** Similar to the ActiveMeetings table, we store all of the scheduled meetings created through our app in a DynamoDB table. This allows us to keep track of information such as, the invited participants, startTime, endTime, and zoom link, so we can easily display it in the frontend calendar.

The partition key for the ScheduledMeetings table is the *MeetingID*.

## 3.2 Specific Endpoints (WebSocket actions)

---

### ***Database-based WebSocket Actions:***

- createGroup - creates a new group for the provided user
- deleteGroup - deletes a given group for the given user
- editGroup - edits the given group for the given user
- getGroups - gets all of the groups for the provided user
- getActiveMeetings - gets all of the current active meetings, and their users/ host rank
- getFavourites - gets the favourites for the provided user
- updateFavourites - updates the favourites for the provided user
- getOwnCalendarMeetings - gets all of the scheduled meetings that will be displayed on the calendar of the currently authorised (logged-in) user
- getUserCalendarMeetings - gets all of the calendar-formatted scheduled meetings of the provided user (so you can look at other user's scheduled meetings).
- getProfileInfo - gets the profile information of the currently authorised (logged-in) user
- getUserLocations - gets the current locations of all of the provided users (what zoom meeting they are currently in)
- getUsers - gets a list of all of the users in the database

### ***Zoom-based WebSocket Actions (send requests to external Zoom API):***

- createMeeting - creates a new active meeting, and calls the zoom API to start it
- createScheduledMeeting - creates a new scheduled meeting, and calls the zoom API to start it

### ***Zoom-based WebHook Actions (WebHooks that Zoom sends to our backend):***

- meetingCreated - zoom sends this webhook when a meeting is created to our backend, and our backend adds the created meeting to our database for future user
- meetingEnded - zoom sends this webhook when a meeting is ended to our backend, which then deletes the meeting from the database, and removes all other dependencies for that meeting (user locations etc.)
- meetingStarted - zoom sends this webhook when a meeting is started to our backend, and our backend uses this action to determine when to send email notifications for scheduled meetings, and updates the database with any new info from zoom
- userJoinedMeeting - zoom sends this webhook when a user joins a meeting, our backend receives this call, and updates the members of the correct meeting in the database, along with updating the joined user's location

- `userLeftMeeting` - zoom sends this webhook when a user leaves a meeting, our backend receives this call and updates the members of the correct meeting in the database, along with removing the meeting from the user's location

## 4 Algorithms

### 4.1 Search & Sort Active Meetings

---

In the frontend, users will be able to search for active meetings by name, or participant names. They will also be able to sort the meetings by a number of common criteria like alphabetically/reverse. Users can sort meetings/groups by their favourites.

## 5 Notable Trade-Offs & Risks

### 5.1 Tradeoffs

---

**Employee Directory:** One tradeoff is that we will be making our own example employee directory, with our own example information. This is a trade off because it might be difficult to implement with HSBC and their own active directory later because we might not have the same information stored in our directories, so there might need to be code changes made in the future to account for change in the directory structure.

**Authentication:** For authentication, we will be using Google Firebase for sign in. This is a tradeoff we have to make because we won't be using HSBC's SSO system. In the future, to integrate our application into HSBC's system, there will need to be more security changes made including switching from Google Firebase sign on to HSBC's SSO system.

**Meetings:** We are not storing current, future, or past meetings in our own database. This allows more flexibility among employees, as changes made to meetings through sources other than our app will continually be updated. There may be an efficiency trade-off, since we will need to continually fetch external information which is likely more time consuming than storing the information internally. Furthermore, this will save valuable storage space and ensure that our app never carries outdated information.

### 5.2 Risks

---

The main risks that come with our solution is that our database will contain employee information that may raise privacy and security concerns. Since we will be storing employee names, hierarchical information, and also Zoom links to meetings and recordings that contain sensitive information, it is crucial that the information stored in our solution is accessible only by those that should be able to access it. All users will need to initially log in to use the solution, and the login credentials, and corresponding user id will then also determine which meetings a user can view.



Additionally, another risk is that we'll be passing Zoom meetings and recording links from the frontend to the backend, and vice versa, which presents a possible risk in that if Amazon AWS security is breached/hacked, the links and information will be vulnerable.

## 6 Solution Delivery Strategy

### 6.1 Hosting

---

**Application Logistics:** The application will be wholly hosted and delivered on the cloud through Amazon Web Services (AWS). As such, all required resources will be allocated, managed, maintained, and scaled through Amazon. These resources include all database and storage needs, which will be provided through DynamoDB. Lambda will provide all of our backend functionality, fetching all information from the database and external sources and providing it to the frontend. Amazon API Gateway handles all requests from the frontend and connects API requests to the associated Lambda functions.

**Capacity and Performance:** These serverless technologies facilitate varying capacities and scale accordingly to handle varying loads. The application can easily be scaled to support many concurrent users (> 20), but resource scaling limitations can be implemented to keep costs down and cap spend. Performance may be impacted based on server load and resource limits, but can easily be kept to ensure low (< 3 second) response times for searches and information fetch.

**System Availability, Maintenance, and Points of Failure:** Due to the serverless nature of the application, no maintenance will be required from our end as Amazon is responsible for all cloud resources. AWS handles all backups and ensures that all servers stay up and running. This creates a critical point of failure in the event of an AWS outage as we do not have any physical servers to keep the application running. Similarly, a database failure or corruption on the side of DynamoDB may potentially alter data without the possibility of backup on our end.

### 6.2 Main Functions

---

**Ongoing Zoom Meetings:** The front page of the application will display a dashboard with all the current Zoom meetings and the participant list of each meeting. There will be functions responsible for querying the backend to retrieve this meeting information and displaying it for the user. The user will be able to search and filter all current meetings.

**Personalised Groups Lists:** There will be several functions responsible for allowing users to create, edit, and delete personalised groups. The functions will be responsible for displaying input fields and interpreting the provided information.

**Favourites Group/List:** Users will be able to favourite any employee which will cause the employee to be added to the users favourites group. A user's favourites will also show by default on the search page along with their current locations.

**Meetings and Calls:** Buttons and input fields on the frontend will allow users to click to start a call with all members of one of their personalised groups, or to edit/schedule a future meeting with all members of the group invited. The functions will be responsible for allowing and interpreting user input and passing the information on to the backend, where lambda functions will communicate created meeting details to the Zoom API. The meeting will then be created through Zoom and have all group members added as participants, before having meeting details communicated back to the frontend.

**Virtual Location Status:** Functions will be responsible for continually retrieving the current Zoom room whereabouts for individual users. This information will then be displayed as a status next to each user in personalised groups or call lists, and can be easily found through a search page of all employees.

**Meeting Calendar:** Information from the Lambda functions pertaining to current/future/past meetings in Zoom for a particular user are ported into a calendar display. If recordings for past meetings are available, these will also appear as a link in the calendar entry which will open and play in-browser. Current and future meetings will appear with a join link that will be accessible at the scheduled time.

## 6.3 External Interfaces

---

The backend of our application will continually be interfacing with the Zoom and relies heavily on the information fetched from these external sources. In the event of a Zoom failure, the app will run, but will not display desired, accurate or up-to-date data. Zoom will be used to gather information about current, past and future meetings, the participants, and creating or modifying meetings. Zoom information will be integrated together in-app to show future meeting information and its participants. The app will also have a connection with Firebase for authentication. In the event of a Firebase failure, users will not be able to authenticate - disallowing use of the application.

## 6.4 Security

---

Ensuring our solution is secure is crucial as it will facilitate both Zoom meetings and recording links. Using [Google Firebase authentication](#), we will be able to securely save the users data and provide a similar experience across devices. This will mainly act as the outer layer of security, which stops any non-HSBC workers from accessing our solution. We also want to ensure individuals cannot join meetings they shouldn't have access to, and also cannot watch recorded meetings they shouldn't have access to. We will still display all of these meetings, but ensuring that only the right people have access to meetings and recordings will be handled through Zoom by employees of HSBC. To protect meetings, hosts will be able to create waiting rooms through Zoom as they usually would to prevent unwanted people from immediately entering the call. The existence of recordings should be visible, but clicking on links for recordings a user was not invited to at the time should prevent them from watching the actual recording.

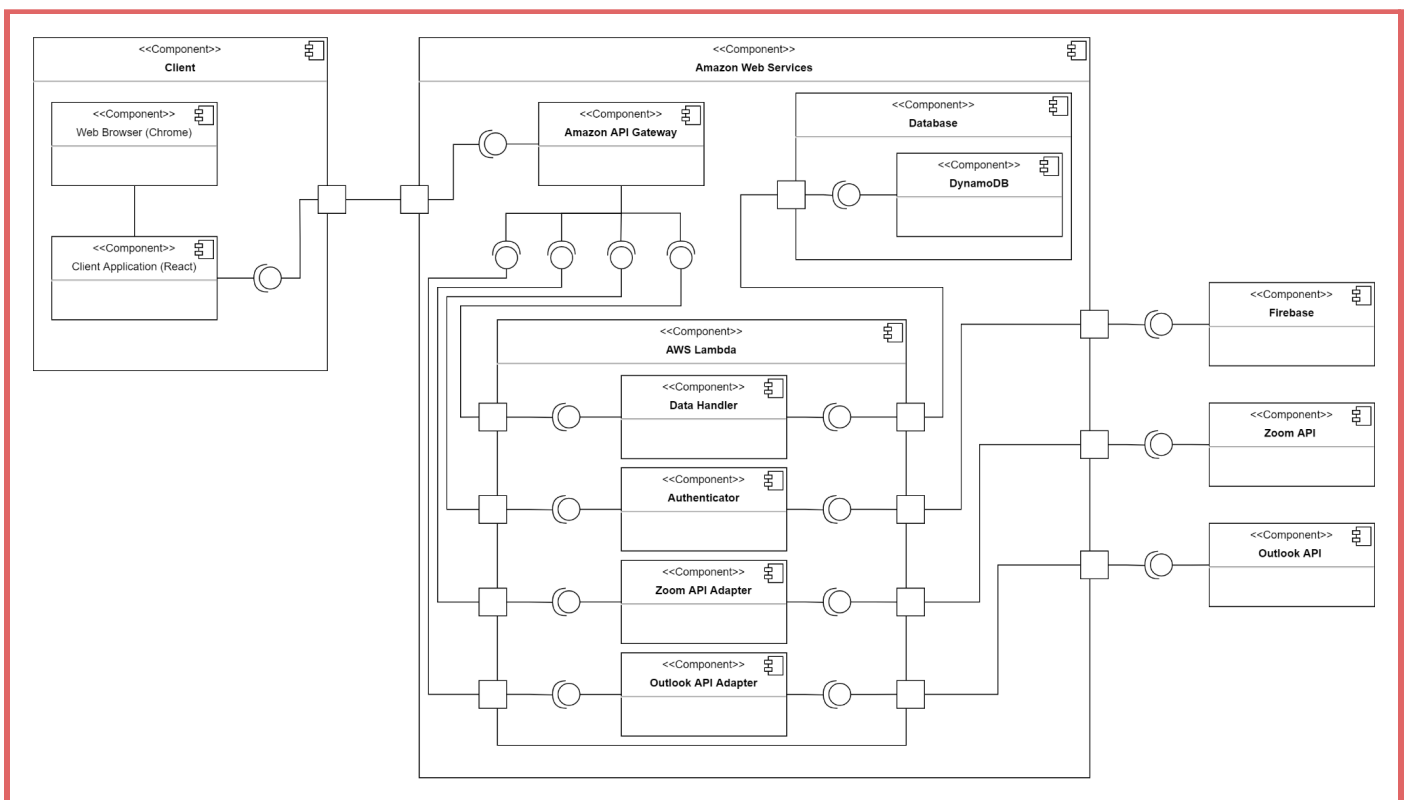
# 7 Appendix

## 7.1 Github Practices

We will have a main branch, development branch, and feature branches for every feature we are working on. We will make pull requests for any changes that need to be merged into the development and main branches which need to be approved by at least one other member of the team.

## 7.2 Component Diagram

*Note: the Outlook section of this diagram is now deprecated*

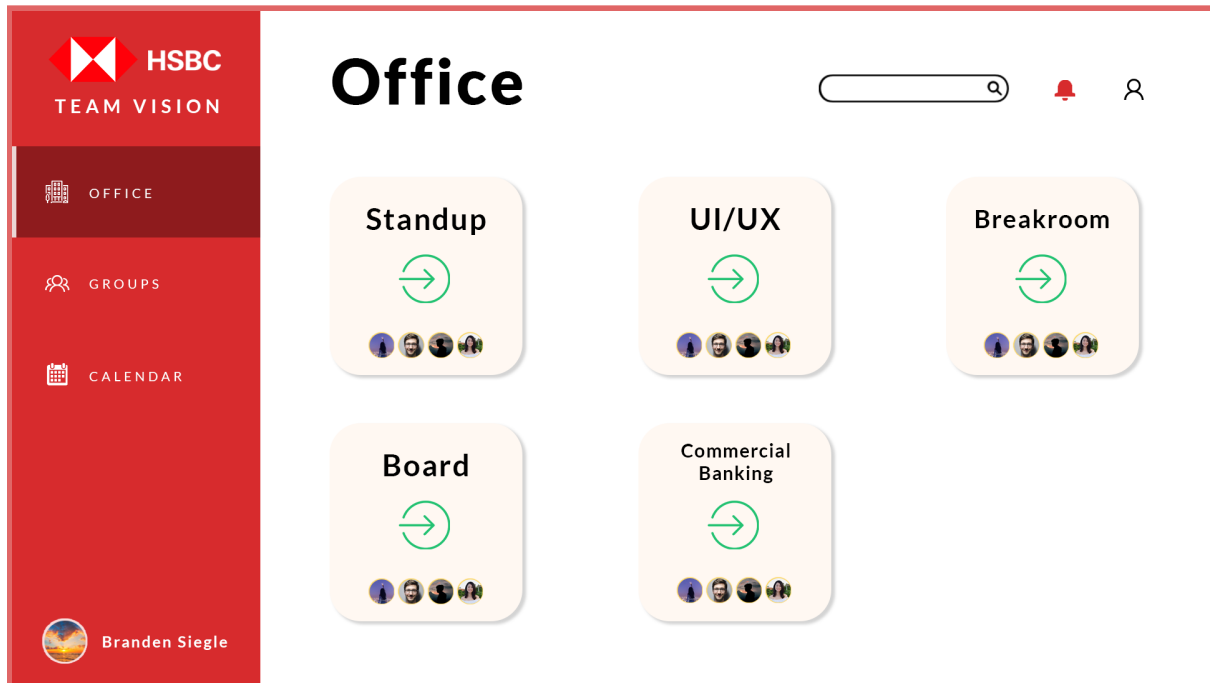


Refer to section 3.2 for a more detailed view of the AWS backend component. [Component Diagram Higher Resolution](#)

## 7.3 Mockups (Outdated)

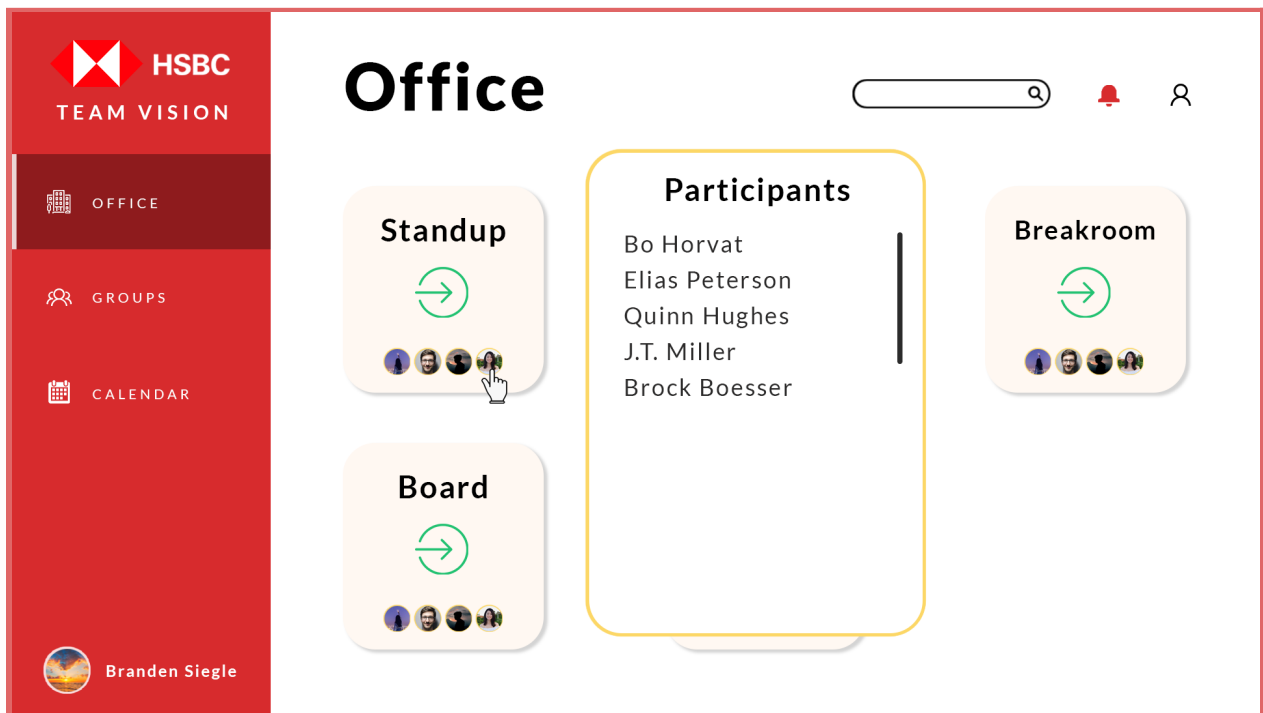
Office View

This is the main view of the application which shows all the active meetings. The user will be able to join any meeting, and sort/search by a number of criteria.



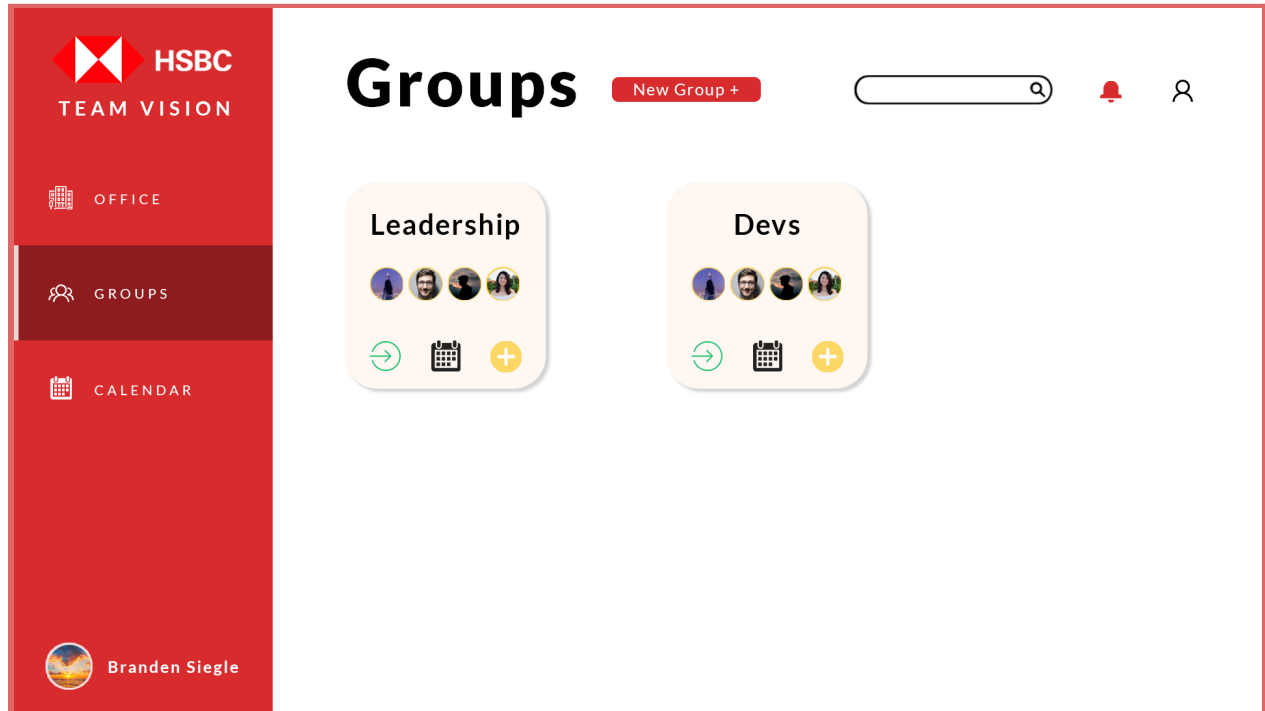
### Office View (Participants Modal)

When a user clicks on the participants of an active meeting, they will see a modal popup that shows the names of all the participants of the given meeting.



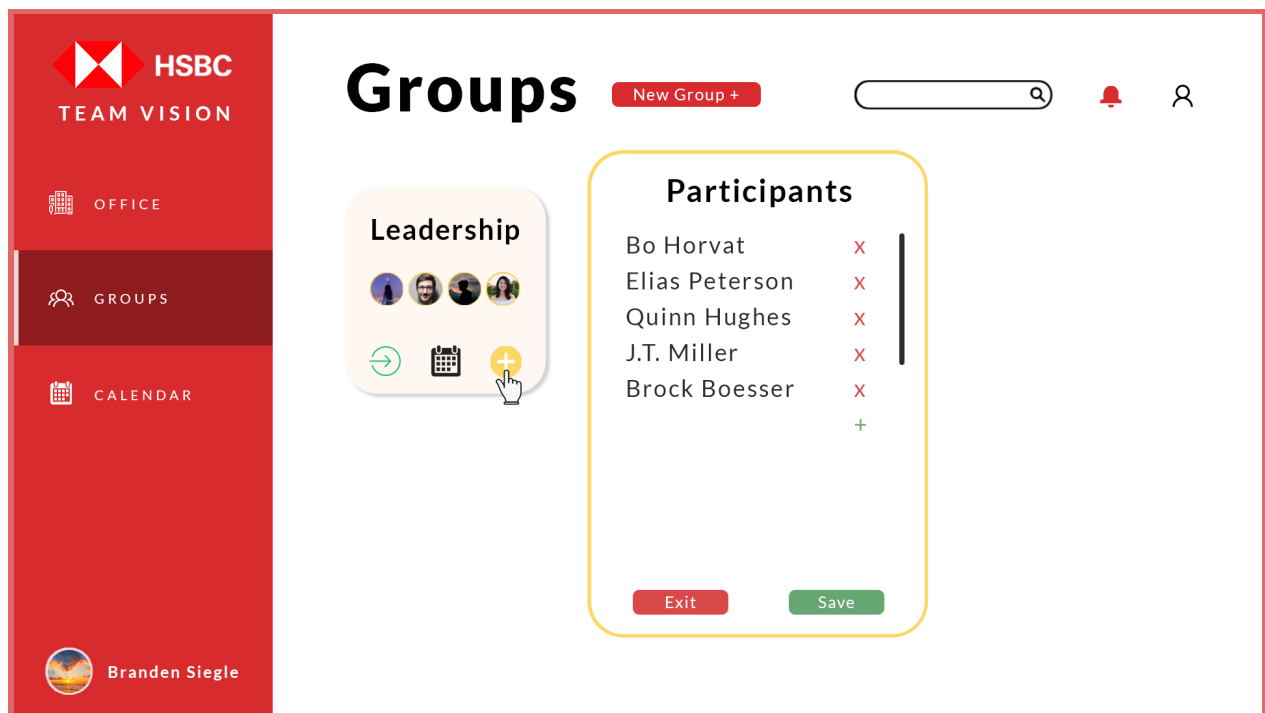
### Groups View

This page will display all the groups the user has created. They will be able to view and edit participants, hold a meeting immediately, or schedule a meeting with this group for the future. Users will also be able to create new groups from this page, and search/sort through groups.




### Groups View (Participants Modal)


When a user clicks on the participants of an active meeting, they will see a modal popup that shows the names of all the participants of the given meeting. They will be able to add/remove participants from the selected group.





## Calendar View


This view allows users to view all their meetings for a given day/week/month. They will also be able to search the calendar for meetings in the past/future. By inputting the name of another employee in the organization, users will also be able to see other employees' schedules here.

**HSBC**  
TEAM VISION



 OFFICE

 GROUPS

 **CALENDAR**

 **Branden Siegle**

# Calendar




**Branden Siegle**


DAYWEEKMONTH


	Sun 24	Mon 25	Tue 26	Wed 27	Thu 28	Fri 29	Sat 30
9:30 AM		Website Re-Design ... 9:30 AM - 11:30 AM					
10:00 AM			Approve Personal C... 10:00 AM - 11:00 AM	Install New Database 9:45 AM - 11:15 AM		Create Icons for We... 10:00 AM - 11:30 AM	
10:30 AM							
11:00 AM					Prepare... 11:00 AM - 1:30 PM	Custom... 11:00 AM - 12:00 PM	
11:30 AM							
12:00 PM		Book Flights to San ... 12:00 PM - 1:00 PM	Final Budget Review 12:00 PM - 1:35 PM	Approve New Online... 12:00 PM - 2:00 PM			
12:30 PM						Launch New Website 12:20 PM - 2:00 PM	
1:00 PM							
1:30 PM							
2:00 PM					Brochure Design Re... 2:00 PM - 3:30 PM		
2:30 PM		Install New Router L... 2:30 PM - 3:30 PM	New Brochures 2:30 PM - 3:45 PM			Upgrade Server Har... 2:30 PM - 4:00 PM	


## Profile View


This page will allow users to view and update their information (including login information) as well as manage notification settings and change profile picture.

**HSBC**  
TEAM VISION



 OFFICE


 GROUPS

 **CALENDAR**

 **Branden Siegle**

# Profile





Edit Image

## Details

Name: **Branden Siegle**

Email: **branden.k.siegle@hsbc.ca**

Title: **Senior Cloud Engineer**