

American Heart Screening Management System

User/Developer Manual

Carlo Campanini ccampanini2018@my.fit.edu

Chris Newberry cnewberry2018@my.fit.edu

Jack Dewey jdewey2018@my.fit.edu

Noah Wilson wilsonn2018@my.fit.edu

Led By:

Drew Dunkelberger ddunkelberge2018@my.fit.edu

Sponsored By:

Dr. Eraldo Ribeiro eribeiro@fit.edu

Client:

Evan Ernst, CEO - Who We Play For

Klynton Holmes, Tech Advisor - Who We Play For

Table of Contents

User/Developer Manual	1
Table of Contents	2
User Manual	5
Director	5
Event Creation	5
Event Type	5
Payment Option	5
Time Slots	5
...	5
Event Statuses	5
InProgress	6
Published	6
Unpublished	6
Finalized *	6
InSession	6
Happening	6
Occurred	6
Event Actions	6
Edit	6
Downloads	7
Sign-in Sheet	7
Cardea Sheet	7
Registration Link	7
Copy Registration Link	7
Get QR Code	7
Cardea Link	7
Finalize	7
Delete (Request)	7
Participant	7

Registration	7
Public Page	7
Link / QR Code	7
Cancellation	7
Policy	7
Refunds	7
Accountant	7
Navigation	7
Yesterday's Payments	7
Exporting to Spreadsheet	7
Search Payments	7
Filters	7
Behavior	7
Payment/Participant Actions	7
Refund	7
Cancel	8
Mark as No Show	8
Payeezy Merchant Account (First Data)	8
Developer Manual	9
System Architecture	9
State Diagrams	11
Event State Diagram - Controlled by Director	11
Participant State Diagram - Controlled by Account	11
Source Code	12
Project Structure	13
Logical Organization	14
Packages	14
Libraries	14
Database	14
Frontend	15
Views	15
Components	15

Store	15
Backend	15
APIs	15
Core	15
Service	16
Close Registration	16
Update Event Status	17
Template Emails	17
Terraform	18
Components	18
Main	18
Shared	18
Makefile	18
Config	19
Terraform Default Vars	19
Tools and Configuration	19
Node Package Manager (NPM)	19
NPM Install Order	19
Common Issues	21
Uploading Changes	21
AWS	21
Project Configuration	21
AWS CLI v2	21
Terraform	21
Running the System	21

User Manual

1. Director

1.1. Event Creation

In order to create an event, a director must click a “CREATE EVENT” button at the top of the director UI. This brings up a form in which the director inputs the event and participant details.

1.1.1. Event Type

Directors must outline whether the event created is a school, community, or private event. School and community events are given a public type and are viewable for registration by anyone, but private events are given a private status and are only available for registration if a participant has the registration link distributed by the director.

1.1.2. Payment Option

Directors must decide what the payment options for the event are. There are three payment options: required, optional, and not required. Required means that all participants must pay for the event on registration, optional means that students can register without paying or decide to donate, and not required means that a student can register without making a payment. **If the participant is an adult, they must pay to register regardless of the payment option.**

1.1.3. Time Slots

When directors create an event, the timeslots for the event are generated using the duration of the event and the number of slots per window. While events have an **InProgress** status, participants must select a timeslot to register, however after an event is **Finalized** the registration link for the event will skip timeslot selection.

1.2. Event Statuses

There are various statuses throughout the lifecycle of an event. Furthermore, each status is comprised of a status and a sub-status in the form *status#sub-status*.

There are two statuses, **InProgress** and **Finalized**, and each status has its own set of sub-statuses. Each status is outlined in more detail below.

1.2.1. InProgress

The **InProgress** status is reserved for events that have not reached their start date yet. Directors can edit these events and request deletion by an admin. They can also choose to finalize the event early if desired. **InProgress** events can have one of two sub-statuses, **Published** or **Unpublished**. Directors can switch an event between these two sub-statuses by clicking a “Published” switch next to the event’s actions menu.

1.2.1.1. Published

Published events are available for registration from participants.

1.2.1.2. Unpublished

Unpublished events are not shown to participants for registration.

1.2.2. Finalized *

A **Finalized** status denotes that an event is no longer modifiable; these events can not be edited or deleted. There are three sub-statuses for a **Finalized** status: **InSession**, **Happening**, and **Occurred**.

1.2.2.1. InSession

InSession denotes that an event is a day before its scheduled start time at some cutoff time (currently 5pm EST), or that the director of the event had finalized the event prematurely, but the time for the event has not yet been reached.

1.2.2.2. Happening

Happening denotes that an event is happening on the current day.

1.2.2.3. Occurred

Occurred denotes that the event is past the day of the event.

1.3. Event Actions

There are two tables, “In-Progress Events” and “Finalized Events”. Each table contains a row for each event in its given table, and each event row contains a blue “Events” button. Clicking this button will open a dropdown menu with various actions you can take for your event. Each item in the actions bar is outlined in more detail below.

1.3.1. Edit

The edit button allows the director to edit various details about an event such as the name, time, location, type, etc. **This option is only available to events with an *InProgress* status.**

1.3.2. Downloads

There are two download options for events, downloading a sign-in sheet and downloading a Cardea Sheet.

1.3.2.1. Sign-in Sheet

The Download Sign-In Sheet button allows a director to download an excel spreadsheet containing all the participants who signed into the given event.

1.3.2.2. Cardea Sheet

The Download Cardea Sheet button allows a director to download an excel spreadsheet containing the cardea information of the participants of the given event.

1.3.3. Registration Link

There are two options for obtaining the registration link of an event, the Copy Registration Link button and the Get QR Code button.

1.3.3.1. Copy Registration Link

The Copy Registration Link button allows a director to copy the url of an event’s registration page. When clicked, a popup will be presented to the user containing the url and the url will be automatically copied to the user’s clipboard.

1.3.3.2. Get QR Code

The Get QR Code button allows the director to get a QR code for the url of the event's registration page. When clicked a third party site will pull up and the event url will be in the input section to create the URL, so the director is able to easily click generate QR and obtain their QR code.

1.3.4. Cardea Link

The Cardea Link button provides the director with a link that can be input into excel to get automatic updates for Cardea Results. When clicked a popup displays the URL of the link and the link is copied to the director's clipboard. **The Cardea Link button is only available for events with a finalized status.**

1.3.5. Finalize

The Finalize button allows the director to change an event's status to **Finalized**. This will make an event uneditable and undeletable. **The Finalize button is only available to InProgress events.**

1.3.6. Delete (Request)

The delete button sends a request for event deletion to an administrator, and "finalizes" the event so that no more participants can register. **This action is only available for InProgress events.**

Currently there is no administrator role / UI to approve deletion; however, in the future an administrator will examine and approve a deletion request, followed by accountants refunding all involved participants.

2. Participant

2.1. Registration

The main feature of the participant UI is the ability to sign up for events created by directors.

2.1.1. Public Page

The public registration page contains all public events available for registration. Participants are able to scroll through the list of available events to find the one they

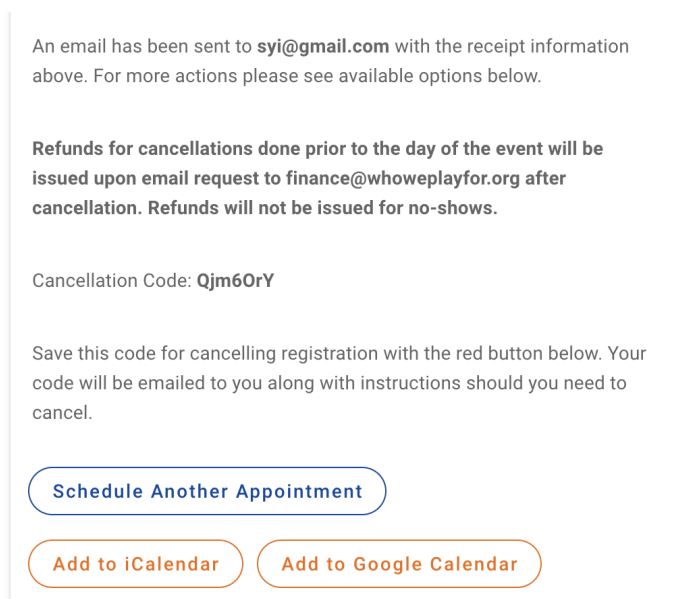
want to sign up for and click an available time slot which will bring up a registration page for the given event. **Events can only appear on the public page if they are a public event, the director has set the event to published, and the event has an InProgress status.**

2.1.2. Link / QR Code

If a participant has a link to the registration for an event via a URL or QR code, they will be able to follow the link to a registration page for the given event. Links allow participants to sign up for events of all types and sign up for events up until the event ends to handle walk-ins.

2.2. Cancellation

Participants are able to cancel their registration for an event using a unique cancellation code that they receive immediately after registration. Participant's are redirected to a confirmation after registration, containing cancellation details. Participant's also receive the code in their registration confirmation email. Below is a snippet of the UI confirmation:



An email has been sent to syi@gmail.com with the receipt information above. For more actions please see available options below.

Refunds for cancellations done prior to the day of the event will be issued upon email request to finance@whoweplayfor.org after cancellation. Refunds will not be issued for no-shows.

Cancellation Code: Qjm6OrY

Save this code for cancelling registration with the red button below. Your code will be emailed to you along with instructions should you need to cancel.

[Schedule Another Appointment](#)

[Add to iCalendar](#) [Add to Google Calendar](#)

2.2.1. Policy

No shows will not be considered for cancellation.

2.2.2. Refunds

Participants are able to request refunds by sending an email to the accountant after canceling their registration. Refunds are only available to participants that have paid for an event and have already canceled their registration. It is preferred that all refunds and cancellations occur through the web application (as opposed to the Payeezy Merchant Account) to keep the database consistent.

3. Accountant

3.1. Navigation

The accountant UI contains two tabs to navigate to pages with different functionality. There are two tabs, **Yesterday's Payments** and **Search Payments**. The functionality of each tab is outlined in more detail below.

3.1.1. Yesterday's Payments

The **Yesterday's Payments** tab allows an accountant to view all payments made on the day before the current day. Furthermore, there are two tables on the page, the "Completed Events" table and the "Events to Review" table. Each of these will contain rows for every participant of relevant events as well as a dropdown button on the left side of the row that will expand the row to show all of the participants' responses for the registration questions.

3.1.1.1. Exporting to Spreadsheet

On the accountant UI there is an "Export Payments" button in the Yesterday's Payments tab. When clicked this button will export all of the data in the Yesterday's Payments participant tables to an excel spreadsheet that the participant can download. Here is an example spreadsheet:

	A	B	C	D	E	F	G	H	I
1	Payment Date	Total Collected	Card Brand	PAN Suffix	Description	Email	Cardholder Name	Transaction Type	Transaction #
2	2022-03-20 15:02	35.00	Visa	1111	Drew Dunk on Sat Mar 19 2022, (some client)	roger@gmail.com	Drew Dunk	purchase	ET190754
3	2022-03-20 19:49	20.00	Visa	1111	Jack Dew on Sat Mar 19 2022, 06:10am (some client)	jdew@gmail.com	John Dewey	purchase	ET195393
4	2022-03-20 20:26	0.00			Kid Man on Sat Mar 19 2022, (some client)	guy@hotmail.com		N/A	
5	2022-03-23 02:57	0.00			Noah Wil on Wed Mar 23 2022, 06:50am (another client)	jwil@gmail.com		N/A	
6	2022-03-23 03:01	20.00	Visa	1111	Chris New on Wed Mar 23 2022, (another client)	mnew@hotmail.com	John Doe	purchase	ET140333
7	2022-03-23 12:17	-35.00	Visa	1111	Drew Dunk on Sat Mar 19 2022, (some client)	roger@gmail.com	Drew Dunk	refund	RETURN
8	2022-03-24 02:55	35.00	Visa	1111	Carlo Camp on Wed Mar 23 2022, (another client)	carlocamp@fit.edu	John Doe	purchase	ET122202
9	2022-03-25 03:33	-35.00	Visa	1111	Carlo Camp on Wed Mar 23 2022, (another client)	carlocamp@fit.edu	John Doe	refund	RETURN
10	2022-03-26 11:26	35.00	Visa	1111	Test Testerson on Fri Mar 25 2022, (another client)	test@test.com	Joe Test	purchase	ET108639
11	2022-03-26 11:45	-35.00	Visa	1111	Test Testerson on Fri Mar 25 2022, (another client)	test@test.com	Joe Test	refund	RETURN
12	2022-03-27 12:30	35.00	Visa	1111	Drew Dunkelberger on Sat Mar 26 2022, 06:30am (Embry Riddle)	ddunkelberge2018@my.fit.edu	Drew Dunk	purchase	ET168127
13	2022-03-28 10:48 pm	35.00	Visa	1111	Evan Dunk on Mon Mar 28 2022, 06:40am (Embry Riddle)	ddunk123@gmail.com	John Snow	purchase	ET159028
14	2022-04-10 10:26 pm	35.00	Visa	1111	Robert Bob on Wed Jan 30 2030, (some client)	robbob@gmail.com	Robert Bob	purchase	ET140083
15	2022-04-10 10:31 pm	-35.00	Visa	1111	Robert Bob on Wed Jan 30 2030, (some client)	robbob@gmail.com	Robert Bob	refund	RETURN
16	2022-04-10 11:18 pm	35.00	Visa	1111	Jane Doe on Wed Jan 30 2030, (some client)	janedoe@gmail.com	Jane Doe	purchase	ET199880
17	2022-04-10 11:26 pm	0.00			Sam Smith on Wed Jan 30 2030, (some client)	ss@gmail.com		N/A	
18	2022-04-10 11:30 pm	0.00			Sherri Yi on Wed Jan 30 2030, (some client)	syi@gmail.com		N/A	
19	2022-04-12 03:45 pm	35.00	Visa	1111	Derick Jones on Wed Jan 30 2030, (some client)	djones@gmail.com	Derick Jones	purchase	ET169071
20	2022-04-12 08:37 pm	35.00	Visa	1111	John Baker on Wed Jan 30 2030, (some client)	jbaker@gmail.com	John Baker	purchase	ET168267
21									
22	Net Total	215.00							

3.1.2. Search Payment

On the “Search Payments tab, accountants will be able to search for specific payments based on a variety of inputs. Accountants are able to use any combination of the given input fields in their search. When the blue “Search” button is clicked, each of the non-empty fields will be used to filter the payments returned to the accountant. Each returned payment is put into a row in a table on the “Search Payments” tab.

3.1.2.1. Filters

There are various filters an accountant can use to find payments. Accountants can search by participant name, guardian name, phone number, email, event name, and date. Each of these filters can be used individually or in combination with the others.

3.2. Payment/Participant Actions

Each table, like the director’s event tables, contains an actions menu button for each payment. Currently, the two supported actions are refunding a payment, marking a participant as a no-show, and canceling a payment.

3.2.1. Refund

The refund button allows an accountant to refund a given payment. When clicked this button will refund a participant’s payment and update the payment’s status to refunded. **Refunds are only available to payments that are greater than \$0.00**

where the participant canceled registration or did not show up to the event they registered for.

3.2.2. Cancel

The cancel button allows an accountant to cancel a participant's payment/registration. This will cancel the participant's payment, but not refund it. This action is needed for when participants request cancellation in registration. **Cancellation is available to all participants until an event is conducted.**

3.2.3. Mark as No Show

The mark as no show button will note if a participant does not attend an event they signed up for. This action is reserved for after an event. **No shows will not be refunded.**

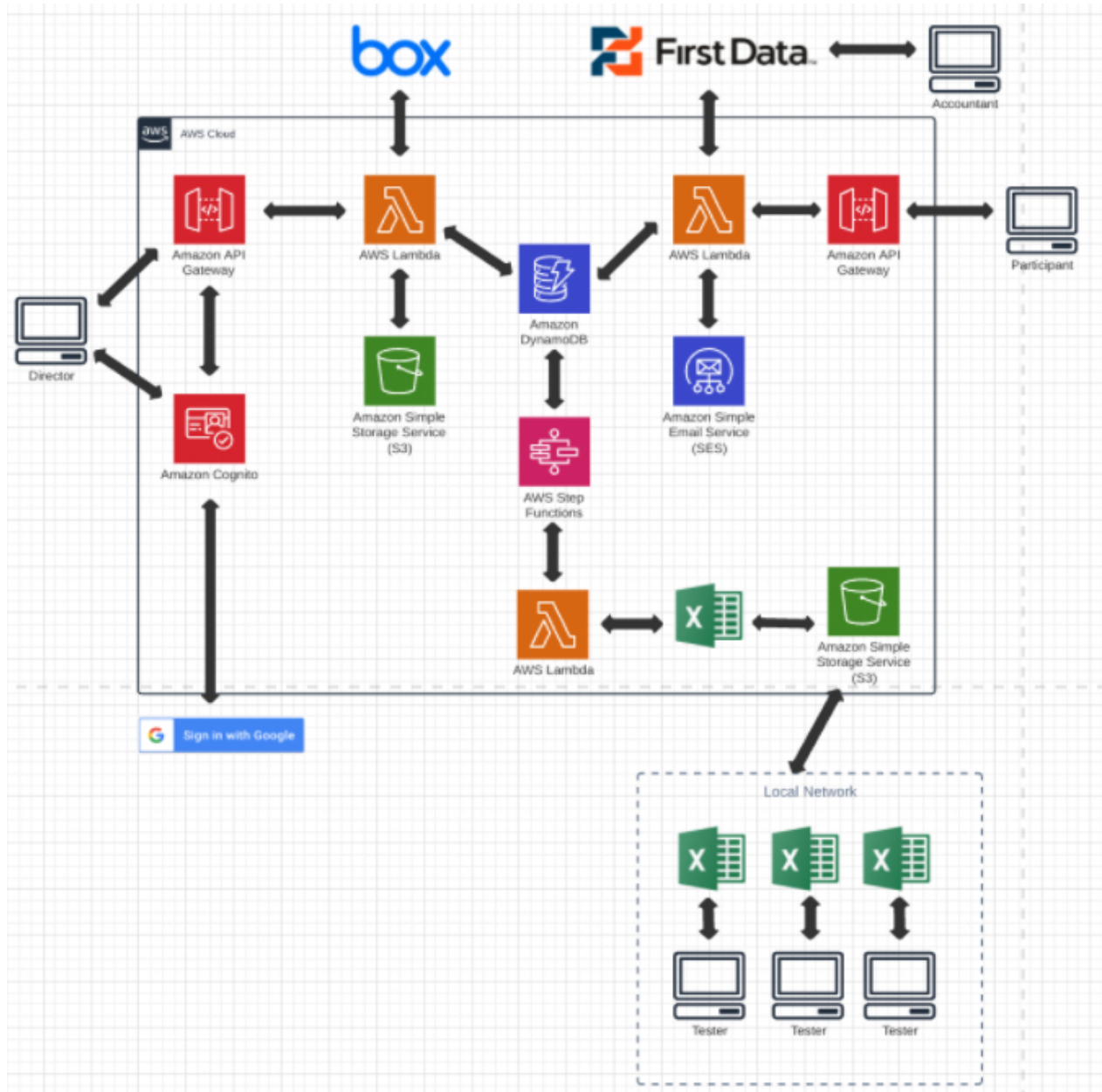
3.3. Payeezy Merchant Account (First Data)

Each payment made on the website will also be recorded by a Payeezy merchant account. This can be used to check the accuracy of the site's information.

Note: Refunds should be made through the website rather than the merchant account

Developer Manual

1. System Architecture



Our system follows a **serverless** architecture by utilizing various **AWS** services. All user requests from the frontend applications are handled by API Gateway via HTTP API calls. These calls are handled by Lambda handler functions. Any dependent code that the Lambda handler needs has been packaged and uploaded to the AWS console with the function itself.

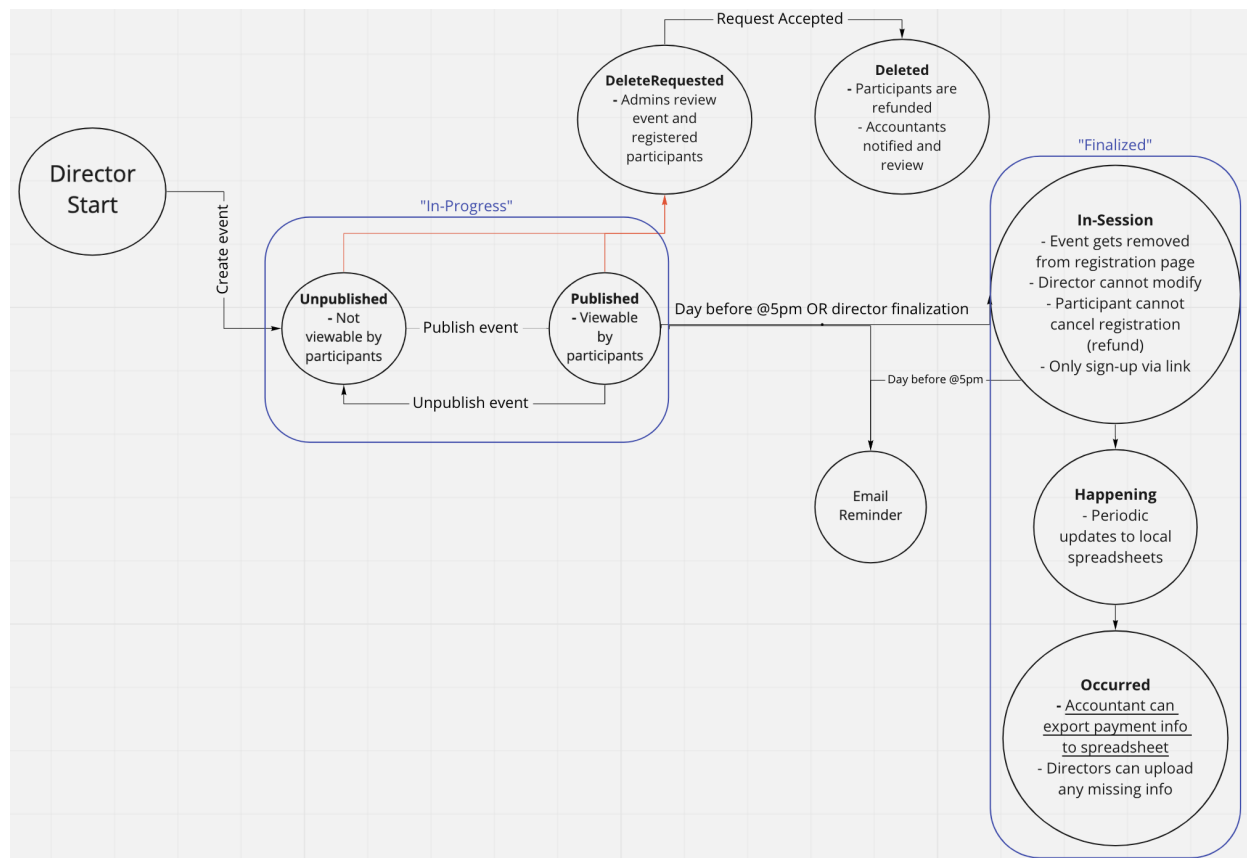
After receiving a request, the Lambda handler will make the corresponding request to the service/business layer functions which can interact with the data access object (DAO) to query, update, or remove from the database (DynamoDB). DynamoDB is a scalable NoSQL database, which contains entries for both events and registered participants.

To automate/manage the transition between states of an event, AWS step functions are utilized to call system Lambda functions based on the start and end of an event to update its status and perform various actions.

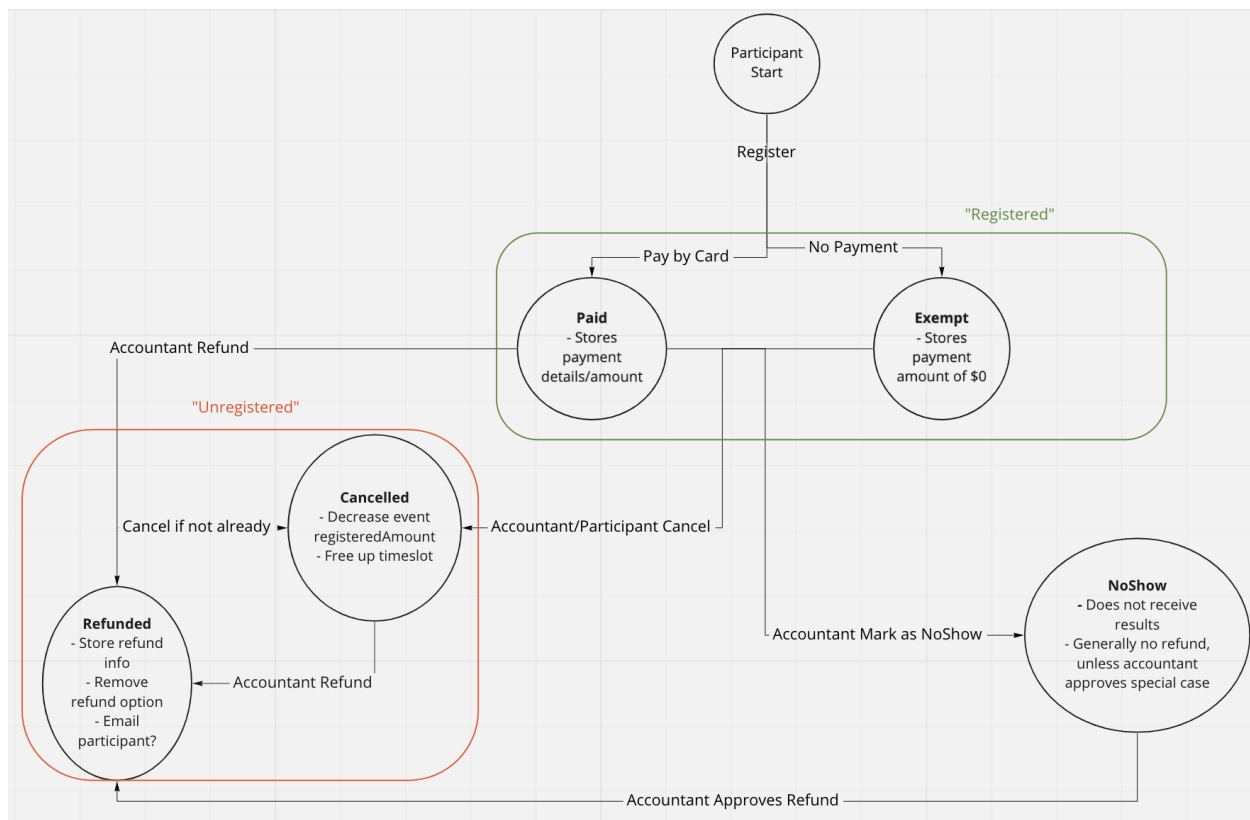
For further details of each module and interactions with AWS, see the [Logical Organization](#) section.

2. State Diagrams

2.1. Event State Diagram - Controlled by Director



2.2. Participant State Diagram - Controlled by Account



3. Source Code

3.1. Project Structure

```
drew_dunk@drew_dunkelberger ems.nosync % tree --gitignore -L 3
.
├── Makefile
├── README.md
├── lerna.json
├── package.json
├── packages
│   ├── backend
│   │   ├── api-accountant
│   │   ├── api-director
│   │   ├── api-participant
│   │   ├── core-accountant
│   │   ├── core-director
│   │   └── core-participant
│   ├── database
│   │   └── dao-ems
│   ├── frontend
│   │   ├── ui-webapp-accountant
│   │   ├── ui-webapp-director
│   │   └── ui-webapp-participant
│   ├── library
│   │   ├── lib-gsuite_authorizer
│   │   ├── lib-logger
│   │   ├── lib-payeezy
│   │   └── lib-response_handler
│   └── service
│       ├── svc-close-registration
│       ├── svc-tmpl-emails
│       └── svc-update-event-status
├── readme_attachments
│   ├── event_state_diagram.png
│   └── participant_payment_state_diagram.png
├── terraform
│   ├── Makefile
│   ├── components
│   │   ├── main
│   │   └── shared
│   ├── config.json
│   └── vars.default.tf
└── 28 directories, 9 files
```

3.2. Logical Organization

Special notes:

- Each user type of the system (Director, Accountant, Participant, etc.) has a corresponding user interface (UI), API, core logic file, and Terraform configurations.
- All *packages* generally contain a **test** folder, in which any unit tests, integration tests, or test configurations are added. The only exception is the frontend, which does not have any written tests.
- Any directory that contains a **package.json** will generally contain a **node_modules** directory (ignored by git), containing all dependencies specified and a **package-lock.json**, containing the latest record of what packages and versions were installed. For more information, see the [NPM](#) section below.

3.2.1. Packages

3.2.1.1. Libraries

Common utilities used throughout the codebase. Contains modules for Google Suite authentication, API calls to Payeezy (for processing and refunding payments), HTTP response/CORS configuration, and debug logging.

3.2.1.2. Database

Isolates all interactions with the database (DynamoDB) to a single file, **dao-ems.js**, which is logically organized by which user type(s) utilize that interaction in their core logic file. Interactions with DynamoDB are made using the [AWS SDK for JavaScript](#). Use this documentation as a source of truth for implementing new queries, or modifying existing ones.

The **test.config.json** file contains the environment variables for many of the unit tests across the different packages. For ease of use, any variables that are dependent on the client/environment are listed at the top, and contain “<env>-<client>-<product>-<service>” in their value. These client-dependent variables need to be changed when running tests, since as a developer you are a client for that environment. Otherwise, you may be populating another developer’s database, or using another one of their resources, making it difficult to debug.

3.2.1.3. Frontend

Contains a [Vue.JS \(v2\)](#) web application for each user type. To reduce time spent on styling the UI, this project makes use of the Vue design library, [Vuetify](#). The *UI Components* section of this link provides an extensive list of reusable components and examples to ease frontend development with Vue.

3.2.1.3.1. Views

An aggregate, user-defined UI component that is composed of one or more different user-defined components. Can also be nested within other views. Some top-level views are referenced for routing purposes.

3.2.1.3.2. Components

Atomic, reusable, user-defined UI components that are only used within views.

3.2.1.3.3. Store

Contains the methods to make API calls to the backend, including authentication, configuration variables, and HTTP requests with [Axios](#).

3.2.1.4. Backend

3.2.1.4.1. APIs

Contains the API gateways for making calls from the frontend to the backend for a specific user type. Parses the supplied route/path and data from Axios to make the correct business logic call in the core file, as explained below. A visual of the gateway and its configurations can be found on [AWS API Gateway](#).

Each API directory contains a **Makefile** which is used for uploading changes to AWS. Make sure you understand the [NPM install order](#) and when re-installing is necessary before uploading changes, since this module is dependent on several others.

3.2.1.4.2. Core

Contains the core business logic for each user type. Methods defined here are generally called by the corresponding API and utilize the queries defined in the database module to perform calculations regarding events, registration, payments, etc.

3.2.1.5. Service

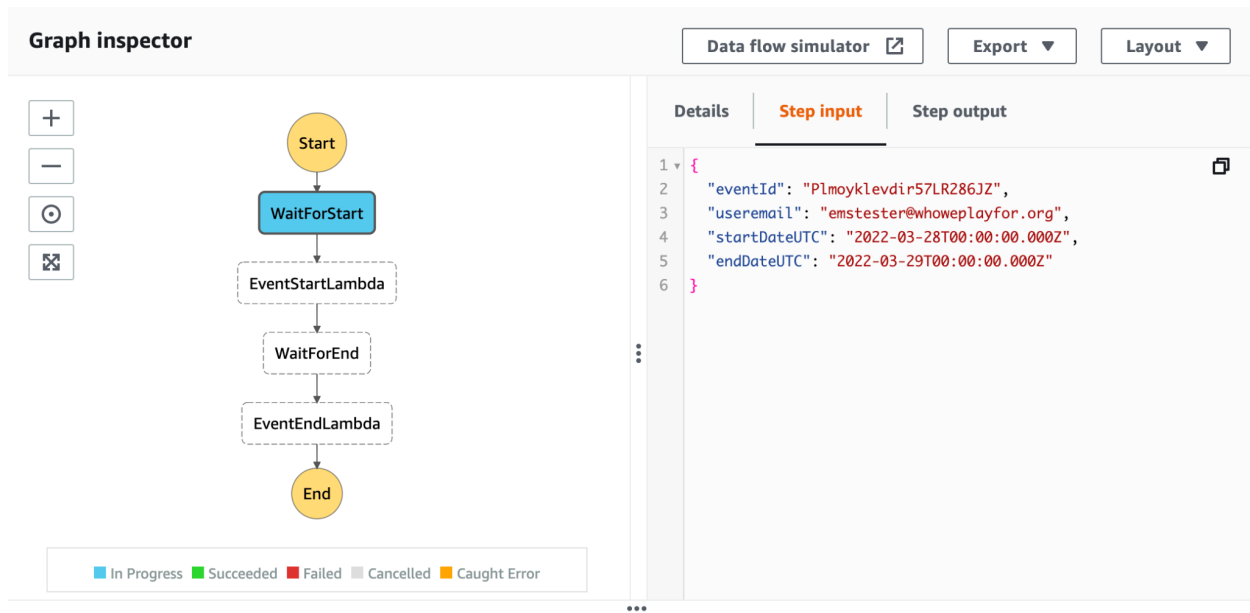
The service module is responsible for carrying out system actions that are not related to any specific user type. As a result, there is no separation between the core logic and API here, since each sub-module has a very specific purpose.

The sub-modules are listed below, and are generally handlers that listen for a specific event generated by AWS to carry out a service function. The only exception is the template emails module, which is used solely for updating, testing, and uploading various email templates used in the project.

3.2.1.5.1. Close Registration

This handler is activated daily at some specified time (currently 5-6pm EDT) by [AWS EventBridge](#) to automatically close public registration (“Finalize”) for events occurring the next day and send reminder emails to all registered participants. Additionally, a spreadsheet containing all participant’s registration information is uploaded temporarily to an [AWS S3](#) bucket (cloud storage container), so that it can be accessed by volunteers during the course of an event (see [Update Event Status](#)).

Finally, a state machine for each event is started, which controls the automatic event state transitions from InSession -> Happening -> Occurred. This state machine calls the **update event status** service function at the specified start and end times of an event to achieve this. A visual of each state machine execution, their current status, and logs are viewable on [AWS Step Functions](#):



3.2.1.5.2. Update Event Status

The main purpose of this handler is to modify the state of an associated event to the state specified by the caller. The caller is either the **close registration** service function, specifying to either start or end a *single* event, or a periodic EventBridge rule that specifies an update to *all Happening* events after each call.

When called with the update status, all Happening events are fetched from the database and have their cardea sheet re-uploaded to the S3 bucket. This remote spreadsheet is available via a **pre-signed URL**, allowing access to the spreadsheet for anyone with the link. Event volunteers are able to connect excel sheets on local machines to the one stored in S3, allowing them to get live updates for any on-site registrations.

3.2.1.5.3. Template Emails

Unlike the other service modules, this one does not contain source code. The **Makefile** here allows for uploading email templates to [AWS Simple Email Service](#) (SES). By default, uploading and testing the template with a sample email are commented out. To use the script, run 'make tmpl=<template-name>'. Currently, the two available templates are **reminder** and **receipt**. The script fetches the template name for

AWS using the **database/dao-ems/test/test.config.json** file. Make sure to change the client name to yours before uploading the template.

Note: anytime the service infrastructure is re-deployed with Terraform, the templates are reset to a default template due to the nature of Terraform. Simply re-run the Makefile with the correct test.config.json environment variables to upload them again.

3.2.2. Terraform

3.2.2.1. Components

3.2.2.1.1. *Main*

Contains all the infrastructure as code (IaC) for AWS involving **storage** (DynamoDB and S3), **backend**, **frontend**, and **service**. These configurations can repeatedly be deployed for various environments/clients pairs.

3.2.2.1.2. *Shared*

Contains IaC for user pool authentication and certificates. Generally, changes will not need to be made here.

3.2.2.2. Makefile

Contains commands that utilize Terraform to setup, destroy, and update infrastructure using a specified configuration file and other arguments. The vast majority of the time, you will be using the **setupClientCmpnts** command (corresponding to **main** components) to deploy infrastructure changes to AWS. When running, you must specify the following arguments:

- **cfg**: local path to config file to use
- **env**: environment you are developing on
- **client**: your developer client name

After setting up components for the first time, a **live_deploys** directory is generated (hidden to git), containing the local state of terraform/AWS infrastructure for any environment/client pairs that you deploy to AWS. Terraform will update these local states each subsequent time you deploy infrastructure to save time.

Important: Do not run commands involving **shared** components unless absolutely necessary and with guidance from the project maintainer

3.2.2.3. Config

JSON object of all input variables into AWS infrastructure configuration (via Terraform). We recommend that you leave this file as is since you do not want to commit environment variables to source control. Instead, create a **test/test.config.json** and use it for any Terraform/Makefile commands. This path is ignored by git and is only used by you locally. Ask the project maintainer for the values to populate it with.

3.2.2.4. Terraform Default Vars

Default configuration variables for Terraform. The provider in our case is AWS. The configuration in the **Provider** object needs to reference your developer configuration stored in your local hidden ~/.aws directory (created for setting up the AWS CLI v2) any time you use terraform to deploy infrastructure.

4. Tools and Configuration

4.1. Node Package Manager (NPM)

4.1.1. Packages and Installation

Each module contains a **package.json** file, specifying 3rd-party package dependencies as well as local module dependencies (starting with **@liodas/**). In order to use these dependencies when code is uploaded to AWS, you need to *install* these dependencies using 'npm install'. The order in which you install dependencies is dictated by which modules depend on other modules and is outlined in the next section.

4.1.2. NPM Install Order

When cloning the project for the first time, you should first run 'npm install' in the root **ems** directory. This will take the longest, as it installs all 3rd party dependencies.

Frontend

The general order for installing dependencies is as follows:

1. Root:
ems
2. Director & Accountant Web App
ems/packages/frontend/ui-web-app-director
ems/packages/frontend/ui-web-app-accountant
3. Participant Web App

ems/packages/frontend/ui-web-app-participant**Backend**

The general order for installing dependencies is as follows:

1. Root:

ems *(Does not need to be run again if done above)*

2. Library:

ems/packages/library/lib-logger

ems/packages/library/* *(remaining modules)*

3. Database:

ems/packages/database/dao-ems

5. Backend Core:

ems/packages/backend/core-director

ems/packages/backend/core-participant

ems/packages/backend/core-accountant

6. Backend API:

ems/packages/backend/api-director

ems/packages/backend/api-participant

ems/packages/backend/api-accountant

- **Note:** If you are uploading code to AWS (first time, or for a change). run **make client={name} env={version}** instead. This command will run npm install before uploading the changes to AWS.

7. Service:

ems/packages/service/svc-close-registration

ems/packages/service/svc-update-event-status

- **Note:** If you are uploading code to AWS (first time, or for a change). run **make client={name} env={version}** instead. This command will run npm install before uploading the changes to AWS.

Note: This is a general guideline for the order of installation, and may change as more modules are added to the project or previous ones change. When in doubt, follow the

dependencies listed in **package.json**. Module dependencies are prepended with **@liodas/**

Installation Tips:

Once this install order is done, you only ever NEED to run 'npm install' when:

- A backend module depends on another that has changed, and you need to upload code to AWS (to see changes reflected with the frontend)
- You switch to another branch where there are backend changes, and would like to see those changes reflected with the frontend
- You merge in new changes from the **main** branch, and would like to see those changes reflected with the frontend
- You re-clone the repository
- The **node_modules** folder gets corrupted in a module, which usually can be resolved by removing it with "rm -r node_modules" followed by "npm install"

You DO NOT need to run 'npm install' when:

- You make changes to the frontend. These changes are reflected immediately in browser or after a refresh
- You are making backend changes, but are only running tests locally. You will only need to install once you are ready to upload and test the changes with the frontend
- You are switching between branches, but do not need to see the changes reflected with the frontend

4.2. AWS

4.3. AWS CLI v2 Configuration

- Purpose
 - Used to run AWS API commands via command line
 - These commands are used in the project Makefiles to upload code to AWS
- Installation

- <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>
- Configuration
 - Once installed, use the command, **aws configure** to step through several config options to specify:

Option	Value to Set
AWS Access Key ID	<Get value from account maintainer, corresponding to your user account>
AWS Secret Access Key	<Get value from account maintainer, corresponding to your user account>
Default region name	us-east-1
Default output format	<none>(hit enter)

- To verify configuration, you can check the newly created, hidden **.aws** directory located in your home directory
 - You should see your specified config options in the **config** and **credentials** files here
 - You can easily get to this directory by using **cd ~/.aws** in your terminal
- To test that you have the correct configurations, try to upload changes to own of your Lambda functions, as outlined below

4.4. Deploying/Uploading Changes to AWS

In order for changes to take place when testing through the UI, you need to deploy any backend changes to AWS. This is done by using the *Makefile* present in several of the modules. These include:

- api-director
- api-participant
- api-accountant
- svc-close-registration
- svc-update-event-status

Following the general rule, **you only need to run npm install in modules that depend on files that you changed.**

- For example, if you were testing the director UI and made a change to **database/dao-ems**, you would need to run npm install in **core-director** and then use the makefile to upload in **api-director** to ensure that it is referencing the updated database code.

In order to upload changes, you must have [AWS CLI v2 installed and configured](#).

4.5. Mocha & Chai Test Frameworks

The Mocha & Chai javascript packages are used to write unit/integration tests for the project (backend). There are currently no automated tests for the frontend. All tests are contained within **test/** directories to distinguish them from source code. For more information, read the documentation at the links below:

- Mocha: <https://mochajs.org/>
- Chai: <https://www.chaijs.com/>

Some of the **test/** directories contain test resources for a fake participant or event. The **packages/database/dao-ems/test/test.config.json** contains configurations that are used in all tests throughout the project. Any configurations that should be changed for your environment are listed at the top of this file.

4.6. Terraform

- Terraform allows you to create infrastructure programmatically rather than through the AWS console/GUI
 - You can create, destroy, and update resources using terminal commands (and the Makefile under the **terraform/** directory), and Terraform handles all dependencies for you
 - This saves time as you make future changes to resources and need changes to take effect across everyone's environments without breaking the system due to dependency conflicts
- Documentation for Terraform for AWS:
<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>
 - Contains syntax and examples for creating all AWS resources
- When creating AWS resources that you are not familiar with, it is often beneficial to create a hello world resource using the AWS console first
 - This allows you to see all of the available configuration options at once and get a better feel for the service
 - Some services let you export the configurations as a JSON from the console, which can sometimes be used in the Terraform code to create an identical resource
 - Example: AWS Step Functions allows you to design state machines graphically in the console. You can then export it as a JSON, which can be pasted into Terraform to create an identical state machine resource programmatically

4.7. API Configuration

- Whenever you checkout a branch from **main**, you should configure the API Gateway Deploy URLs to match the ones corresponding to your developer account
 - This ensure that you are using your own resources and not someone else's, which can make debugging difficult and interfere with their environment
- URL Locations
 - There is an API deploy URL for each role's UI:

- **packages/frontend/ui-webapp-<role>/source/store/apis/api.config.js**
- Update the **<ROLE>_APIGATEWAY_DEPLOY_URL** to the URL provided by the project maintainer
 - These values can also be obtained using **tfoutput** command in the Terraform Makefile specifying your client name as a parameter
- Stashing your URLs
 - Since you don't really want to commit your URL to history, you can instead stash the changes using git
 - This allows you to quickly apply the changes with a single command when checking out a new branch
 - Steps:
 - Checkout a clean branch without any changes (**main** will work fine)
 - Update the API deploy URLs to your own
 - Run **git stash -m "Configure API URLs"**
 - Saves your changes with given message
 - Verify that changes were stashed, by running **git stash list**
 - You should see your changes at some index
 - To apply them, use **git stash apply <index>**
 - This applies the changes to whatever branch you are on without removing them from the stash
 - Stashing changes is useful in general when you want to switch between different branches without committing changes to source control yet

5. Running the System

Once you configure AWS CLI v2, set up your API Gateway URLs, and upload code to your AWS resources, then you are ready to run the system:

- Navigate to the desired web application that you'd like to run at **packages/frontend/ui-webapp-<role>**
- Serve the frontend:
 - Run **npm run serve**

- After completion, open **localhost:8080**
- Any subsequent UI will be served on **localhost:8081**
- Follow user manual to gain general understanding of the frontend and available actions for each user

Notes:

- If running participant UI and the director or accountant UI, run the participant UI last, as the development environment is configured to use port **8081** for registration links by default
- Currently, the director and accountant cannot be run simultaneously
 - This is because they currently share the same Google Suite account and further setup is required with AWS Cognito

6. Useful Links

AWS Terraform Registry	https://registry.terraform.io/providers/hashicorp/aws/latest/docs
AWS JS SDK Documentation	https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html
AWS Management Console	https://us-east-1.console.aws.amazon.com/console/home?region=us-east-1#
Mocha Test Framework	https://mochajs.org/
Chai Assertion Library	https://www.chaijs.com/
Vue v2 Guide	https://v2.vuejs.org/v2/guide/
Vuetify Component Library	https://vuetifyjs.com/en/components/
MDI Icon Search	https://materialdesignicons.com/