# Marching Masters
# Software Design Specification

| | |
|---|---|
| **Group Members** | Brandin Bulicki, Adam Luong, Aparna Mishra, Siddharth Srinivasan, Tumaris Yalikun, Jeffer Zhang |
| **Faculty Advisor** | Dr. Filippos Vokolos, Ph. D. |
| **Project Stakeholder** | Marching Masters |

# Revision History

| Name | Date | Reason for Change | Revision |
|---|---|---|---|
| Brandin Bulicki, Adam Luong, Aparna Mishra, Siddharth Srinivasan, Tumaris Yalikun, Jeffer Zhang | 1/12/2021 | First Draft: Initial Outline | 1.0 |
| Brandin Bulicki, Adam Luong, Aparna Mishra, Siddharth Srinivasan, Tumaris Yalikun, Jeffer Zhang | 1/22/2021 | Completion of Introduction and Description Sections | 1.2 |
| Brandin Bulicki, Adam Luong, Aparna Mishra, Siddharth Srinivasan, Tumaris Yalikun, Jeffer Zhang | 1/25/2021 | Completion of Design Overview Section | 1.4 |
| Brandin Bulicki, Adam Luong, Aparna Mishra, Siddharth Srinivasan, Tumaris Yalikun, Jeffer Zhang | 2/1/2021 | Initial Composition of UML Design Sections | 1.6 |
| Brandin Bulicki, Adam Luong, Aparna Mishra, Siddharth Srinivasan, Tumaris Yalikun, Jeffer Zhang | 2/7/2021 | Completion of UML Design Sections | 1.8 |
| Brandin Bulicki, Adam Luong, Aparna Mishra, Siddharth Srinivasan, Tumaris Yalikun, Jeffer Zhang | 2/9/2021 | Completion of Software Design Document | 2.0 |

# Table of Contents

# 1. Introduction

## 1.1. Purpose

This is the Software Design Specifications for the Marching Masters Software. The purpose of this document is to describe the implementation of the Marching Masters Software described in the requirements document. Marching Masters is designed to track and improve performer accuracy within the marching arts. This document will outline software design specifications for Marching Masters in addition to system architecture and system components.

## 1.2. Scope

This document describes the implementation details of the Marching Masters Product. The software will consist of two major functions and several secondary functions. Firstly, to allow for the tracking of a performer's accuracy when performing a marching drill and the creation of two users (Instructors and Performers) which will monitor the progress made. Secondary functions will include the sharing of documents, events, and assignments, an editable drill book, and communication between Instructor and Performer. This document will not specify the testing of the software.

## 1.3. Definitions

### 1.3.1. Technologies Definitions

Flutter: A Google UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase. [1]

REST: Representational State Transfer, is an architectural style for providing standards between computer systems on the web. This style allows for code to be edited independently on the client and server. [2]

AWS Lambda: a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes. [3]

AWS Lambda: a key-value and document database that delivers single-digit millisecond performance. [4]

### 1.3.2. Marching Arts Definitions [5]

Coordinate/Dot: An individual position on the field designed by the number of steps away from the hashes or sidelines, and yard lines.
Step(s): The measurement of marching performers from landmarks on the field.
8-to-5: The standard step-size for marching meaning 8 steps per every 5 yards. (22.5" Step)

# 2. Design Overview

## 2.1. Description of Problem

Every summer, marching bands and drum corps all around the world learn drills for their season's performance. Then, in the winter, indoor percussion, guard, and wind ensembles prepare their drill for their seasons. As it stands, the only way to effectively learn the coordinates (aka dots) for the show is to go 'set-by-set' and check every performer's positioning for every drill move. These outdated methods demand lengthy amounts of time taken during rehearsals spent learning the drill. Marching Masters works to decrease the amount of time needed to spend learning the drill while also increasing the accuracy of the performers.

## 2.2. Technologies Used

The Marching Masters Software will utilize Flutter, an open-source UI software development kit, for the front-end. Flutter allows native deployment to both Android and iOS which simplifies cross-platform development.

We aim to build a lightweight application and rely on REST calls to synchronize end user inputs and outputs. Therefore, we are aiming to build out the backend infrastructure using AWS, specifically AWS lambdas for serverless computing. AWS DynamoDB is a great option for a NoSQL key-value database. And AWS S3 for document storage.

## 2.3. System Architecture

Figure 1 shows a high-level overview of the Marching Masters.
The System for this game comprises of the following elements:

- Front-End Interface

This interface will be used by the user to experience, interact, and access items stored in the database.

- Integration System

  This system will be used as a way for the Front-End Interface and Back-End Interface to communicate with one another.

- Back-End Interface

  This interface will be responsible for supporting the front-end interface, specifications of back-end routes for requests, and to manage socket communication.

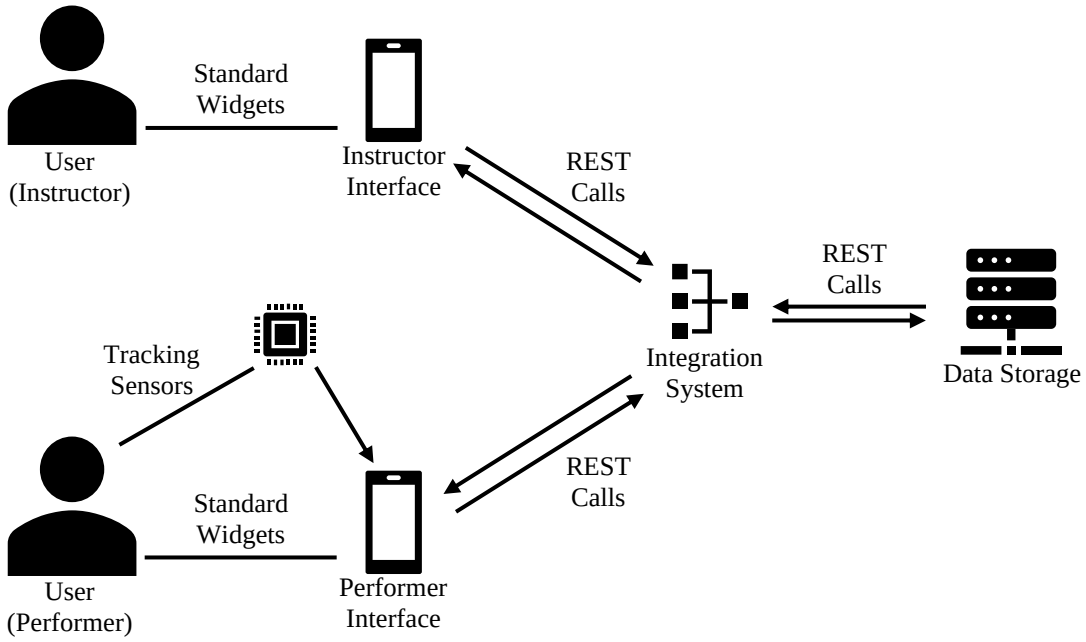


*Figure 1: High-Level overview of user and server interaction for the Marching Masters System Interface*

## 2.4. System Operation

The following figures show the typical sequence of events that takes place for main actions taken by users. The communication arrows shown between the user and the back-end system below are propagated via REST calls.

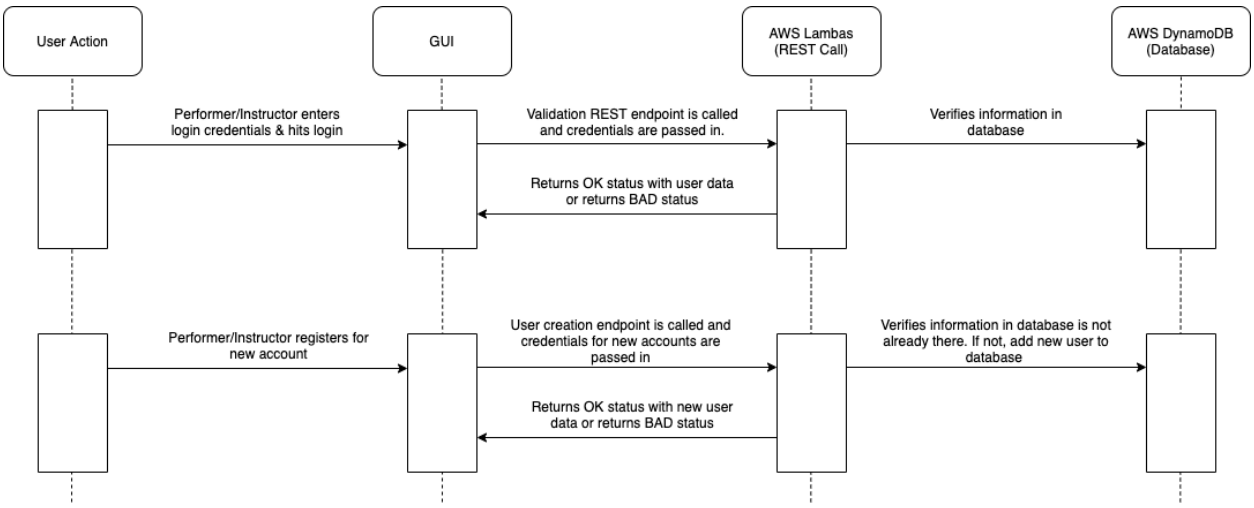


*Figure 2: Sequence Diagram of the User Login Feature*

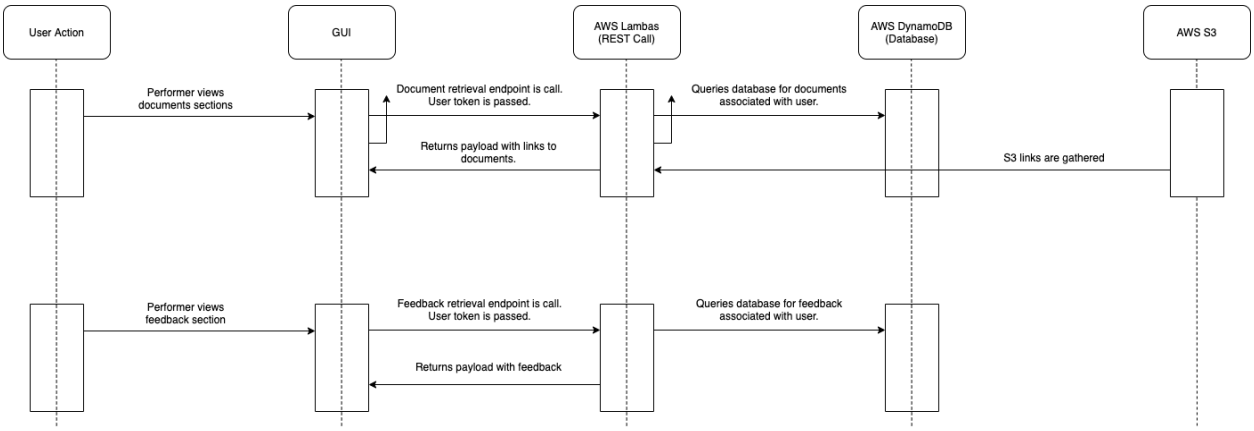


*Figure 3: Sequence Diagram of Document/Feedback Feature*

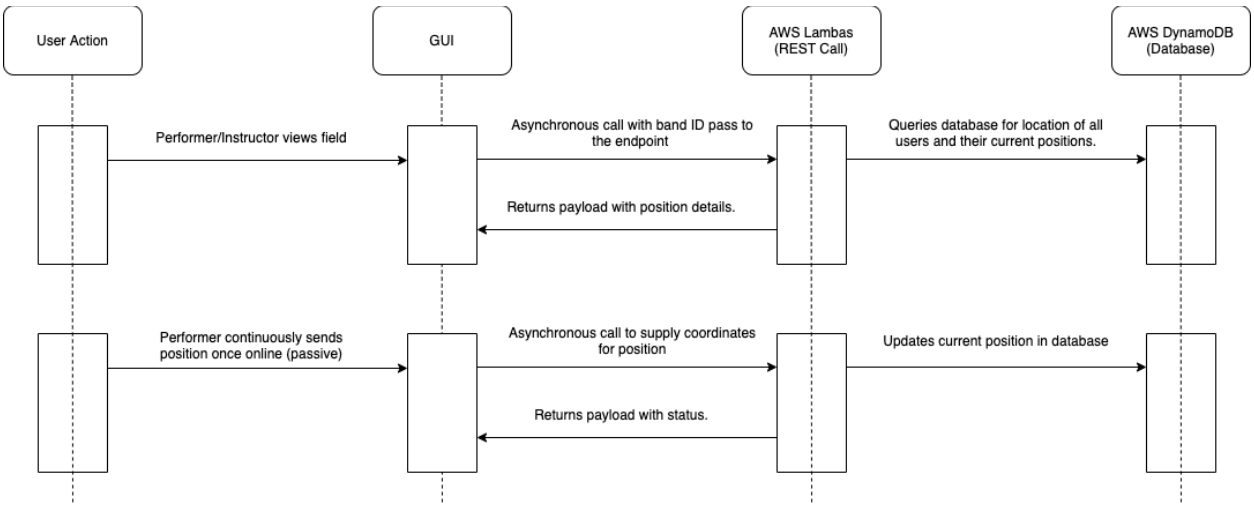


*Figure 4: Sequence Diagram of the Field View Feature*

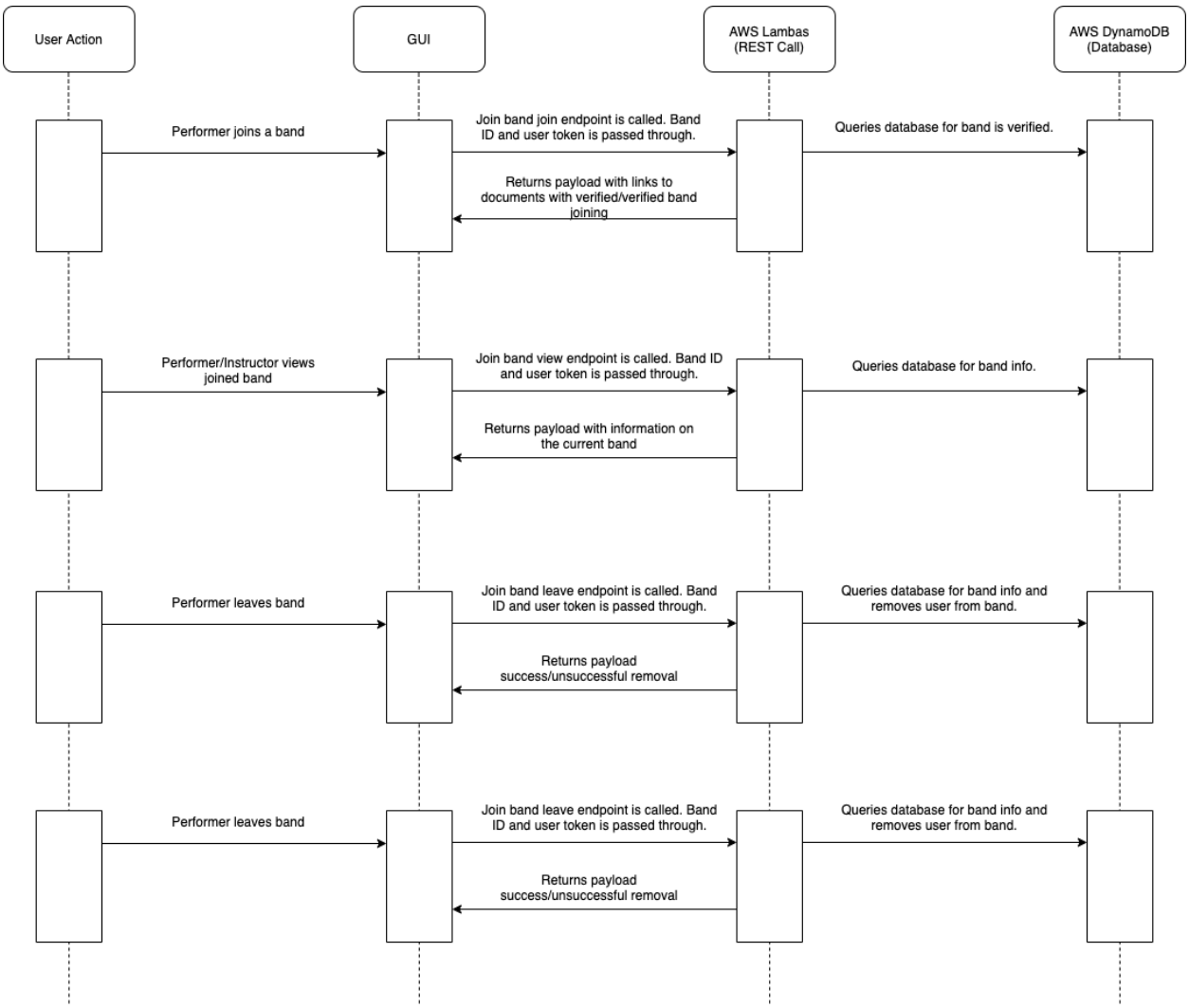


*Figure 5: Sequence Diagram of the Joining/Leaving of Band Feature*

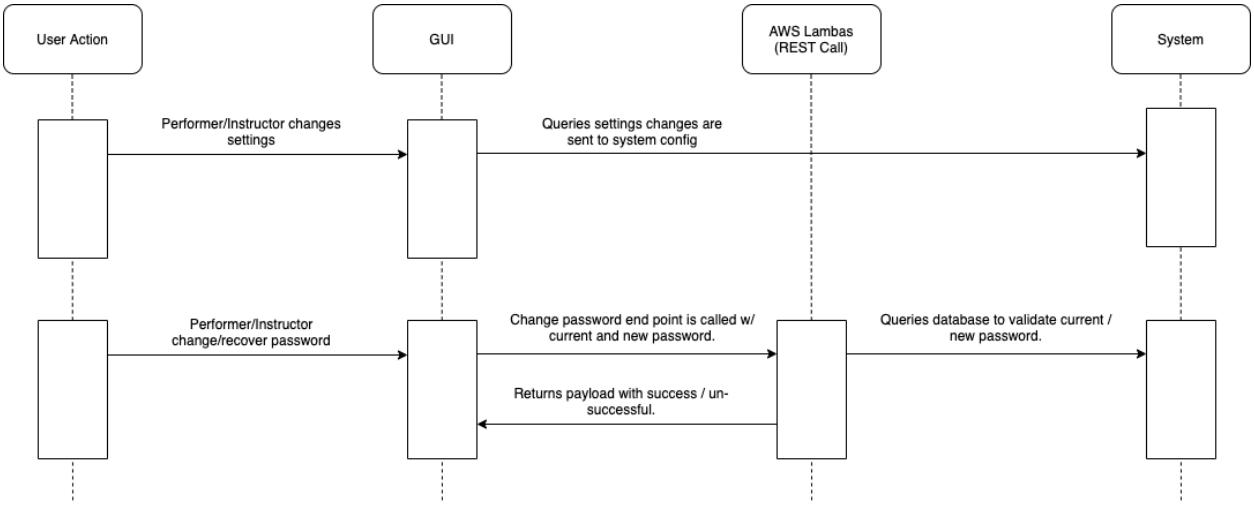


*Figure 6: Sequence Diagram of the Settings Update Feature*

# 3. Requirements Traceability

| Requirement | Description | Design Reference |
|---|---|---|
| R3.1.1.* | Login | §4, §6 |
| R3.1.2.* | Registration | §4, §6 |
| R3.1.3.* | Home Screen | §4, §6 |
| R3.1.4.* | Field View | §4, §6 |
| R3.1.5.* | Band Information – Instructor | §4, §6 |
| R3.1.6.* | Band Information – Performer | §4, §6 |
| R3.1.7.* | Document/Events/Assignment Handling | §4, §6 |
| R3.1.8.* | Feedback Communication – Instructor | §4, §6 |
| R3.1.9.* | Feedback Communication – Performer | §4, §6 |
| R3.1.10.* | Settings | §4, §5, §6 |
| R3.2.1.* | Tracking Speed (Performance) | §4, §5, §6 |
| R3.2.2.* | Tracking Accuracy | §4, §5, §6 |

# 4. Front-End Interface

## 4.1. Overview

The front-end interface of the application will communicate with the back-end interface via the integration system via REST calls. It will primarily handle user interactions with the system, using the Screen component.

## 4.2. Attributes/Methods

FieldView:

Attributes:

| Name | Type | Description |
|---|---|---|
| Trackingelement | | Used to contain Tracking information |

Methods:

| getFieldView(field: Field) | |
|---|---|
| **Input** | FieldObject |
| **Output** | None |
| **Description** | The Method to display the current fieldview |

Bands:

Attributes:

| Name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| bandID | UUID | Attribute Containing the Band ID |

Methods:

| **joinBand(bandID: UUID)** | |
|---|---|
| **Input** | IIOD Object to identify the band |
| **Output** | Void |
| **Description** | Method used to join a new band |

Bands (Instructor):

Methods:

| **createBand(bandID: UUID)** | |
|---|---|
| **Input** | IIOD Object to identify the band |
| **Output** | Void |
| **Description** | Method used to create a new band |

Display:

Attributes:

| **Name** | **Type** | **Description** |
|---|---|---|
| currentScreen | Screen | Variable for the current screen |
| localUser | User | User object to know the current user. |

Methods:

| **Method** |
|---|
| **Input** |
| **Output** |
| **Description** |

Screen:

Attributes:

| **Name** | **Type** | **Description** |
|---|---|---|
| Type | Screentype | Type of Screen that is being displayed |

Methods:

| **changeScreen(type: Screentype)** | |
|---|---|
| **Input** | Screentype Object to change the displayScreen |
| **Output** | Void |
| **Description** | Method to change current screen |

| getScreen() | |
|---|---|
| **Input** | None |
| **Output** | Screentype |
| **Description** | Method used to retrieve the ScreenType Information |

Navigation:

Attributes:

| Name | Type | Description |
|---|---|---|
| NavType | Screen | Type of Navigation Screen to Display |

Login:

Attributes:

| Name | Type | Description |
|---|---|---|
| User | String | String for the user's username |
| Pass | String | String for the user's password |

Methods:

| enterSystem(user:String, pass: String) | |
|---|---|
| **Input** | String for the username and password |
| **Output** | Void |
| **Description** | Method used for a user to enter the application |

User:

Attributes:

| Name | Type | Description |
|---|---|---|
| IsAdmin | Bool | Used to check if user is administrator |
| Name | String | Variable for name of the user |

Methods:

| User(name: String): User | |
|---|---|
| **Input** | String Containing the Username |
| **Output** | Object containing the initialized user |
| **Description** | Method to initialize the user object |

| getName():string | |
|---|---|
| **Input** | Button Push |
| **Output** | String Containing the username |
| **Description** | Method to get the user data from back-end |

Interface:

Attributes:

| Name | Type | Description |
|---|---|---|
| currentScreen | Screen | Current Screen to be displayed |
| LocalUser | User | User object containing User Data about current user |

Methods:

| showDisplay(screen: Screen): Display | |
|---|---|
| **Input** | Screen Object |
| **Output** | Display object will display the current screen object |
| **Description** | Method to update the screen display |

Documents:

Attributes:

| Name | Type | Description |
|---|---|---|
| Type | DocType | Contains the type of Document Screen |
| File | Filetype | Contains the file to be sent |

Methods:

| viewDocuments(type: DocType) | |
|---|---|
| **Input** | Doctype object to view |
| **Output** | Void |
| **Description** | Method used to view documents |

| sendDocuments(file: filetype) | |
|---|---|
| **Input** | file object to send |
| **Output** | Void |
| **Description** | Method used to send documents |

Settings:

Attributes:

| Name | Type | Description |
|---|---|---|
| Curpass | String | Current password of the user |

Methods:

| **setUsername(newUser: String)** | |
|---|---|
| **Input** | String containing the new username |
| **Output** | Void |
| **Description** | Method used to change the value of the username variable |

| **setPassword(newPass: String)** | |
|---|---|
| **Input** | String Containing the new password |
| **Output** | Void |
| **Description** | Method used to change the value of the password variable |

Help:

Methods:

| **getHelp()** | |
|---|---|
| **Input** | Button Push |
| **Output** | Void |
| **Description** | Method Called to allow the user to get further help with navigating the application |

Tracking:

Attributes:

| **Name** | **Type** | **Description** |
|---|---|---|
| TrackingStatus | Tracking | Registers whether the tracking is online or offline |

# 4.3. Front-End UML Diagram

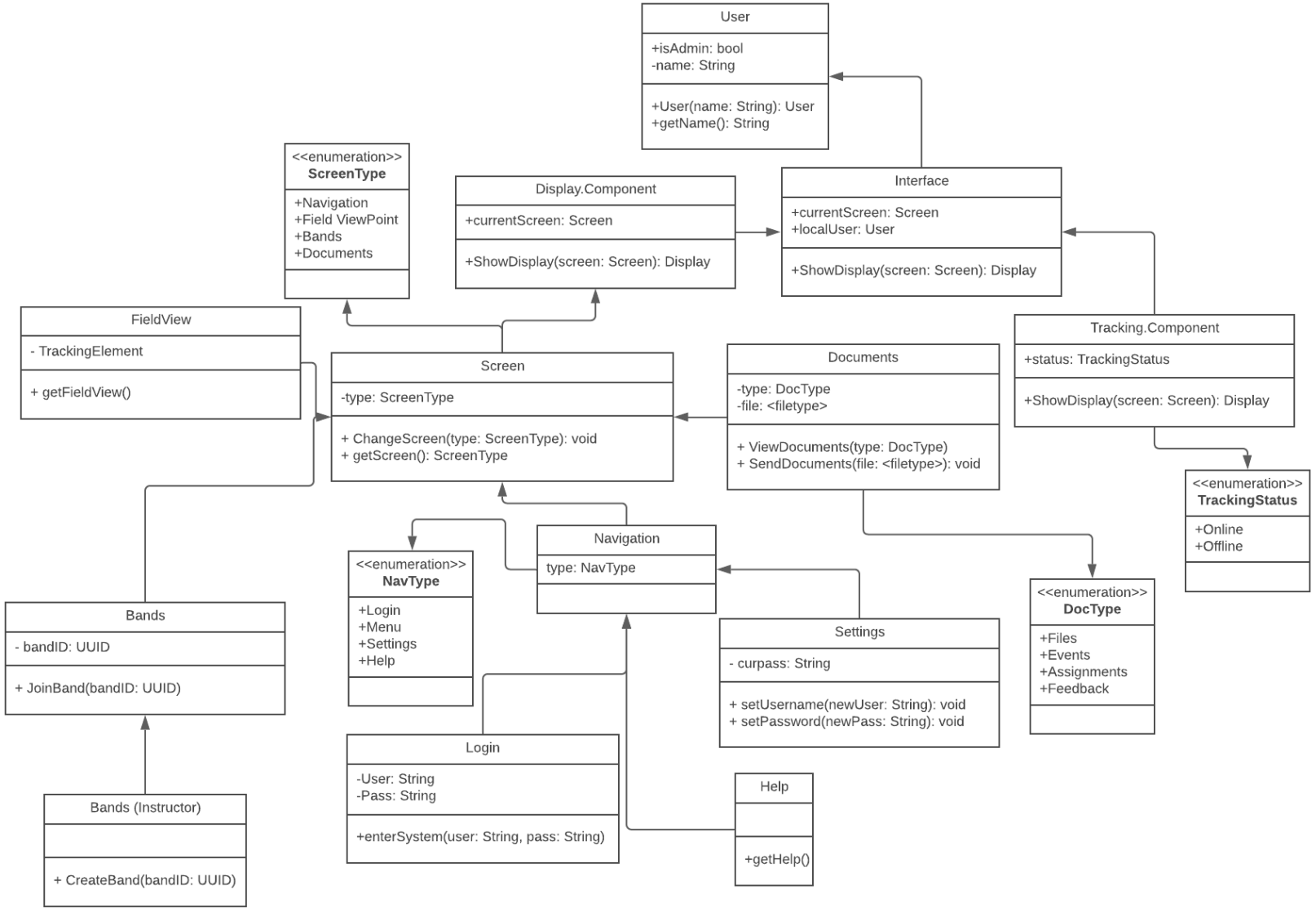


*Figure 7: UML Diagram of the Front-End Interface for the Marching Masters System*

# 5. Back-End Interface

## 5.1. Overview

The back-end interface of the application will support the front-end interface. It will primarily handle information storage and requests, communicating with the client using AWS Lambda.

## 5.2. Attributes/Methods

Profile Manager:

Attributes:

| Name | Type | Description |
|---|---|---|
| Name | String | Contains a string that indicates the name of the performer, the instructor, or the task. |
| DataPath | String | Contains the main directory for the storage of Profiles of Instructors and Performers. |
| Sessions | String | Contains a string that indicates the session details. |

Methods:

| **Vector <String> getLoginDetails()** | |
|---|---|
| **Input** | A string that contains information about login details. |
| **Output** | A vector of objects from the login details class that is loaded from user data. |
| **Description** | Method returns a vector of login details and cross checks with AWS Dynamo DB. |

| **Vector <String> getLoginDetails()** | |
|---|---|
| **Input** | A string that contains information about signup details. |
| **Output** | A vector of objects from the details class that is loaded from user data. |
| **Description** | Method returns a vector of sign up details and cross checks with AWS Dynamo DB. |

| **Void viewDocument( )** | |
|---|---|
| **Input** | Void |
| **Output** | A String that contains a query for documents associated with the user token. |
| **Description** | Method returns the document and payloads associated with links to the document. |

| **Bool CreateBand ( )** | |
|---|---|
| **Input** | A string containing Band ID and user token from the GUI. |
| **Output** | Bool |
| **Description** | Method returns payload of whether the creation was successful or not. |

| **Bool LeaveBand ( )** | |
|---|---|
| **Input** | A string containing Band ID and user token from the GUI. |
| **Output** | Bool |
| **Description** | Method returns payload of whether the removal was successful or not. |

| **Void viewFeedback ( )** | |
|---|---|
| **Input** | Void |
| **Output** | A String that contains a query for feedback associated with the user token. |
| **Description** | Method returns the feedback and payloads associated with links to the document. |

Session Validation:

Attributes:

| **Name** | **Type** | **Description** |
|---|---|---|
| DataPath | String | Contains the main directory for the storage of Profiles of Instructors and Performers. |
| Tasks | TaskData[] | Contains Tasks that are to be associated with. |
| Performer | String | Contains a string that exhibits performer view. |

Methods:

| **Void viewAccount ( )** | |
|---|---|
| **Input** | Void |
| **Output** | Strings representing account data for Performers and Instructors. |
| **Description** | Method returns strings representing the accounts data for Performers and Instructors. |

| **Void viewSettings ( )** | |
|---|---|
| **Input** | Void |
| **Output** | Strings representing settings data for Performers and Instructors. |
| **Description** | Method returns strings representing the data for Performers and Instructors. |

| Void getTasks ( ) (vector <String> fields) | |
|---|---|
| **Input** | A vector of strings that represent the names of all the fields that end up making a task. |
| **Output** | Void |
| **Description** | Method is responsible for the vector of strings that represent the Task creating fields. |

Task Data:

Attributes:

| Name | Type | Description |
|---|---|---|
| Name | String | Contains a string that indicates the name of the performer, the instructor, or the task. |
| Data | File | Contains data associated with the Profiles of Instructors and Performers. |
| Metric | String | Contains a string that indicates associated with the data points for Instructors and Performers. |

Methods:

| String getName ( ) | |
|---|---|
| **Input** | String |
| **Output** | A string that indicates the name of the performer, the instructor or the task. |
| **Description** | Method returns a string that indicates the name of the performer, the instructor or the task. |

| File getData ( ) | |
|---|---|
| **Input** | File |
| **Output** | A string that contains the main data for profiles of performers and instructors. |
| **Description** | Method returns main data points for performers and instructors . |

## 5.3. Back-End UML Diagram

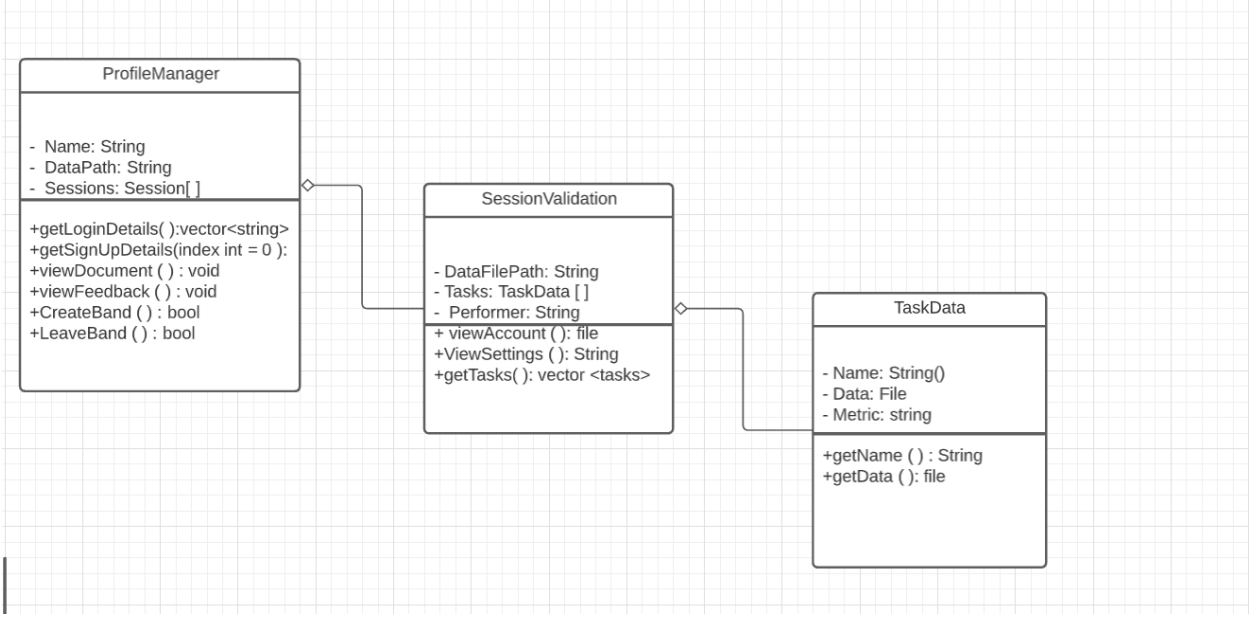


*Figure 8: UML Diagram of the Back-End Interface for the Marching Masters System*

*Figure 8: UML Diagram of the Back-End Interface for the Marching Masters System*

# 6. REST Interface

## 6.1. Overview

The REST interface of the application will connect the client and server. Using REST calls, the front-end and back-end will communicate with one another.

## 6.2. AWS Lambda Methods

Method: validateCredentials()

Take in passed credential information from GUI to verify if it exists in AWS DynamoDB. If it exists, returns “OK” status with user data, else returns “BAD” status.

Method: createUser()

Take in passed created credential information from GUI to verify that it does not exist in AWS DynamoDB. If it exists, returns “BAD” status, else returns “OK” status.

Method: retrieveDocument()

Take in passed user token from GUI to create a query for documents associated with the user token. Returns payload with links to the documents.

Method: retrieveFeedback()

Take in passed user token from GUI to create a query for feedback associated with the user token. Returns payload with feedback.

Method: createBand()

Take in both Band ID and user token from GUI to create a query for band creation. Returns payload of whether the creation was successful or unsuccessful.

Method: joinBand()

Take in both Band ID and user token from GUI to create a query for band verification. Returns payload with links to documents with verified band joining.

Method: viewBand()

Take in both Band ID and user token from GUI to create a query for band information. Returns payload with information of current band.

Method: leaveBand()

Take in both Band ID and user token from GUI to create a query for removal of the user from the band. Returns payload of whether the removal was successful or unsuccessful.

Method: disbandBand()

Take in both Band ID and user token from GUI to create a query for removal of band. Returns payload of whether the removal was successful or unsuccessful.

Method: viewPosition()

Take in Band ID from GUI to create a query for the location of all users and their current positions. Returns payload with position details.

Method: loadPosition()

Takes in current position and user token to update the current position in the database for a query.

Method: changePassword()

Takes in both current and new password from GUI to create a query to validate current password and update it to the new password. Returns payload with whether the password change is successful or unsuccessful.

# 7. References

[1] https://flutter.dev

[2] https://www.codecademy.com/articles/what-is-rest

[3] https://aws.amazon.com/lambda/

[4] https://aws.amazon.com/dynamodb/

[5] https://www.kennettmarchingband.com/marching-band-terms.html