Scheduling Flutter App – Design Document

By Hannah Jeffers, Matthew Latta, Patrick McManus, Michael Connors, and Pedro Gomez

# Table of Contents

[Table of Contents 2](#_Toc2056180534)

[1). Introduction 2](#_Toc1707118335)

[1.1 Purpose of the Document 2](#_Toc1866535835)

[1.2 Intended Audience 2](#_Toc1923069679)

[2). General Description 3](#_Toc443920532)

[2.1 Product Goal 3](#_Toc323617190)

[2.2 Product Profile 3](#_Toc1810587831)

[2.3 Assumptions and Dependencies 4](#_Toc358291509)

[3). Functional Design 5](#_Toc1287489213)

[3.1 User Class 5](#_Toc1216362887)

[3.1.1 getUsernameFromEmail(String email) 6](#_Toc953208578)

[3.1.2 getUserIDFromEmail(String email) 6](#_Toc803291748)

[3.1.3 addClub(Club club) 6](#_Toc347958209)

[3.1.4 removeClub(Club club) 6](#_Toc1079032401)

[3.1.5 addPreference(String preference) 7](#_Toc329608893)

[3.1.6 removePreference(String preference) 7](#_Toc1595078577)

[3.1.7 resetPassword(String email, String newPassword) 7](#_Toc1361425061)

[3.1.8 removeUser(int user\_id) 7](#_Toc19771644)

[3.1.9 isEmailWithUser(String email) 8](#_Toc2067676769)

[3.1.10 isUsernameWithUser(String username) 8](#_Toc1091380623)

[3.2 Coordinator Class 8](#_Toc490760839)

[3.2.1 setCoordinator() 8](#_Toc2090116862)

[3.2.2 setClub(Club club) 9](#_Toc417462010)

[3.2.3 hostEvent(Event event) 10](#_Toc1933424332)

[3.2.4 endEvent(Event event) 10](#_Toc1198688149)

[3.2.4 demote() 11](#_Toc1486307019)

[3.2.5 getManagedEvents() 11](#_Toc1011303460)

[3.2.6 changeEventName(Event event, String name) 11](#_Toc1372598235)

[3.2.7 changeEventDesc(Event event, String desc) 11](#_Toc688265068)

[3.2.8 changeEventTime(Event event, String time, String date) 12](#_Toc218695684)

[3.2.9 getClub() 12](#_Toc852510894)

[3.2.10 leaveClub() 12](#_Toc1737914189)

[3.3 Calendar Class 12](#_Toc953926881)

[3.3.1 createCalendar(String title) 13](#_Toc272030329)

[3.3.2 deleteCalendar() 13](#_Toc565826248)

[3.3.3 addEventToCalendar(Event event) 13](#_Toc1910267056)

[3.3.4 removeEventFromCalendar(String eventID) 13](#_Toc1772525433)

[3.3.5 uploadCalendar(String filePath) 14](#_Toc900379287)

[3.3.6 shareCalendar(String userID) 14](#_Toc1464385287)

[3.3.7 combineCalendars(Calendar calendar, Calendar otherCalendar) 14](#_Toc265003114)

[3.3.8 getEventList(Calander calendar) 15](#_Toc951915021)

[3.3.9 getSharedWithList() 15](#_Toc462076161)

[3.3.10 getCombinedCalendarsList() 15](#_Toc1824634994)

[3.3.11 isSharedWith(String userID) 16](#_Toc1870284681)

[3.3.12 getEventCount(Calander calendar) 16](#_Toc306288013)

[3.4 Event Class 16](#_Toc1262724951)

[3.4.1 function 16](#_Toc754684067)

[3.5 Club Class 17](#_Toc12932724)

[This class holds all the information of a created club. 17](#_Toc238898688)

[String clubName; // Name of club 17](#_Toc511597429)

[String desc; // Description of club 17](#_Toc45459140)

[List memberIDs; // or Map 17](#_Toc2136863124)

[List adminIDs; // Or bool isAdmin in a Map of userIDs 17](#_Toc1900475311)

[List events; // list of events holding event objects 17](#_Toc1170834781)

[3.5.1 addAdmin(String userID); 17](#_Toc372414759)

[3.5.2 removeAdmin(String userID); 17](#_Toc1950430895)

[3.5.3 addMember(String userID); 17](#_Toc1524108636)

[3.5.4 removeMember(String userID); 18](#_Toc910837214)

[3.5.5 changeDesc(String newDesc); 18](#_Toc1951468835)

[3.5.6 addEvent(Event event); 18](#_Toc1816372949)

[3.5.7 removeEvent(String eventID); 18](#_Toc113539730)

[3.5.8 isAdmin(String userID); 18](#_Toc1035201056)

[3.5.9 isMember(String userID); 19](#_Toc855498189)

[3.5.10 getAdminCount(); 19](#_Toc280365892)

[3.5.11 getMemberCount(); 19](#_Toc36820563)

[3.5.12 getEventCount(); 19](#_Toc1988847748)

[3.5.13 getAdminList(); 19](#_Toc1508288015)

[3.5.14 getMemberList(); 19](#_Toc1227768380)

[3.5.15 getEventList(); 19](#_Toc550538342)

[Appendix A – Definitions 20](#_Toc414264344)

[Appendix B – Abbreviations 20](#_Toc927159909)

[Appendix C – References 20](#_Toc1068907593)

# 1). Introduction

## 1.1 Purpose of the Document

This document identifies technical requirements related to a scheduling and event coordinating application.

## 1.2 Intended Audience

The intended audience of this document is primarily college students, faculty, & staff.

Testers will also utilize this document when creating test case plan.

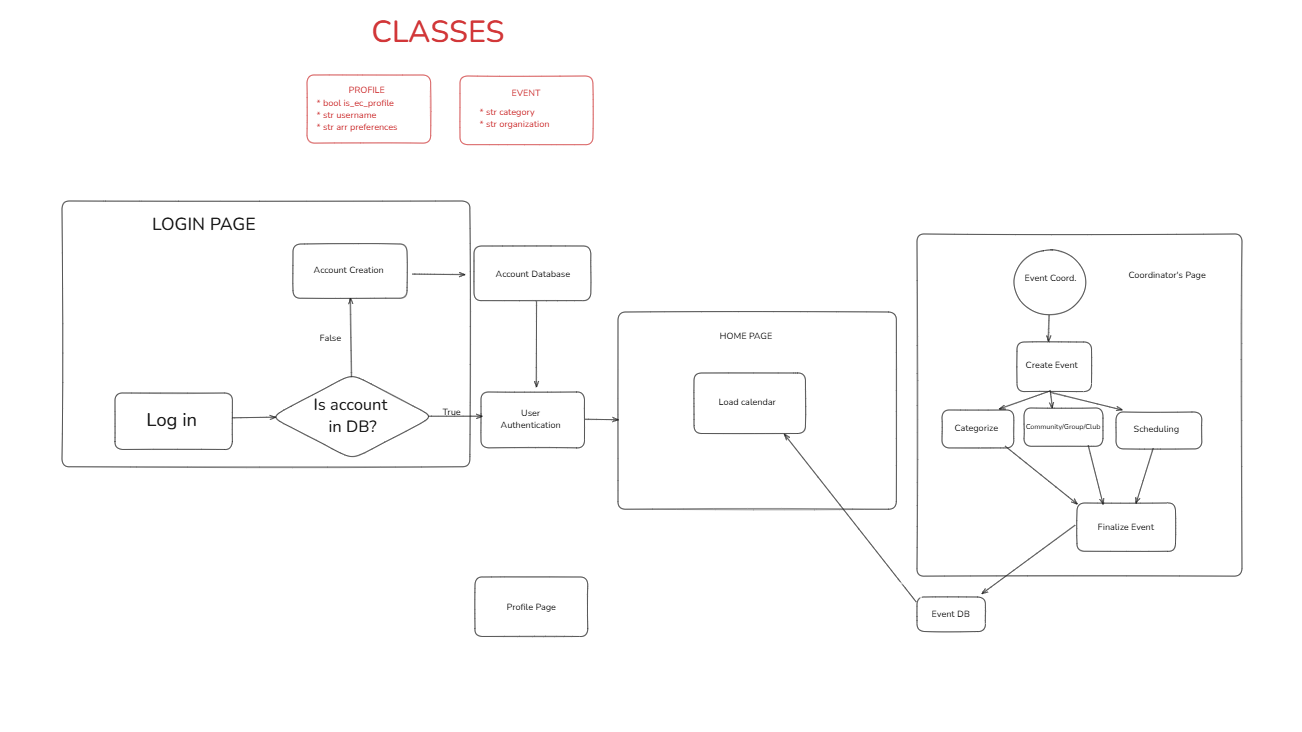
# 2). General Description

## 2.1 Product Goal

The scheduling app aims to resolve issues revolving around organization/club searches, event planning and student engagement outreach. It will spread information about events just like the current system in Outlook, except it will be in a place that makes it much easier to manage schedules and does not keep you from your important emails. It is similar to the project of another group in CS-375, but it focuses more on easing the burden of event coordinators rather than to make scheduling easier for students. To our scheduling app, the help for students is more of a side effect of easily managed schedules for event hosts.

## 2.2 Product Profile

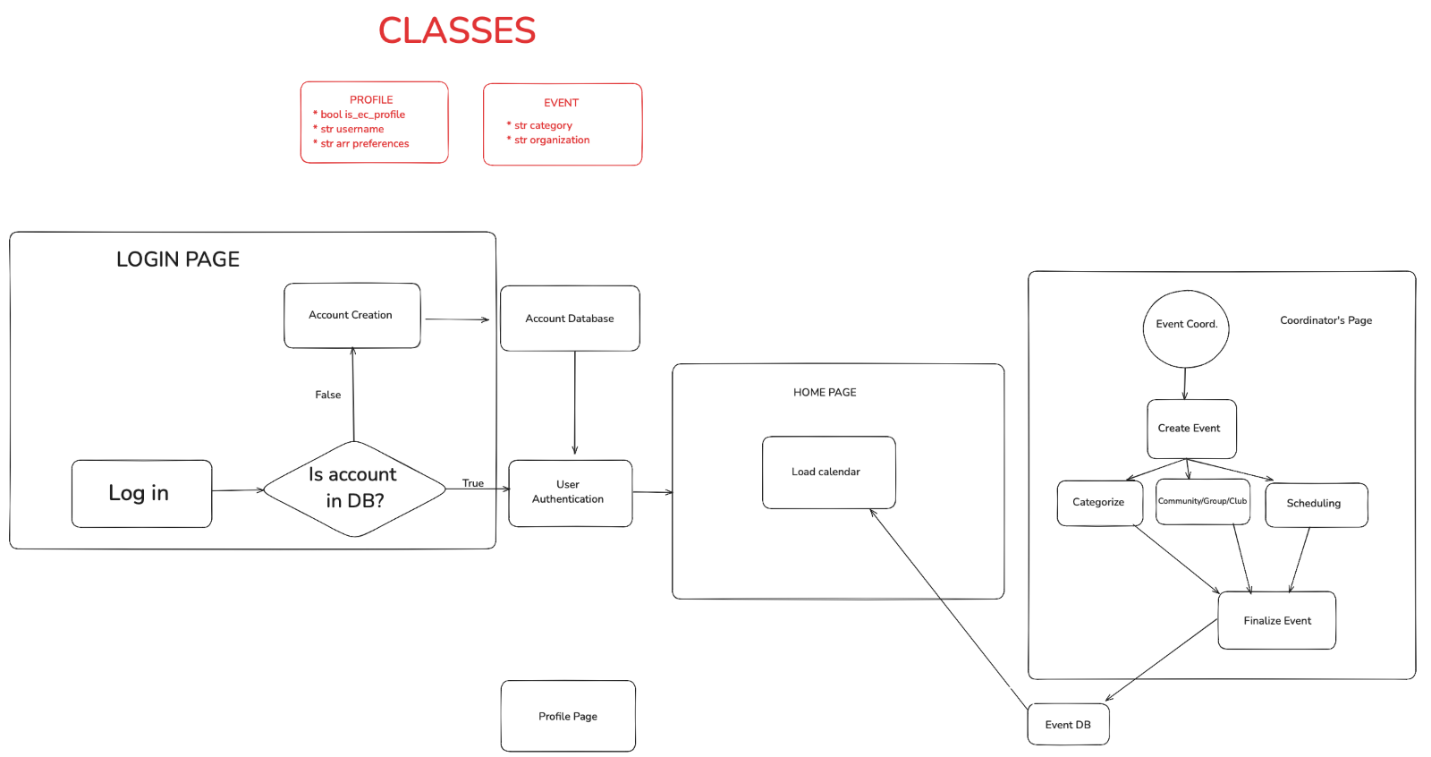
The following diagram gives an overview of the functionality of the scheduling app. On the login in page, it must verify if the user has an account already stored in the database. If not, then it guides the user through the account creation process and stores the new account for future use. Afterwards, the user must be authenticated before entering the home page. Once accessed, the home page can load the calendar of events and allow the user to partake in various functions to customize their calendar to their pleasing. The events added to their calendar come from a database of events. The event coordinator side of the app allows club leaders to create their own events. They will need to categorize the event and add a schedule for this event to be finalized and added to the event database that the user can use for their calendar customization.



## 2.3 Assumptions and Dependencies

Our scheduling app depends on the user having an iOS or Android mobile device that is capable of running the app and has it installed. This includes enough memory and processing power to run the app on your mobile device.

# 3). Functional Design



## 3.1 User Class

This class holds information for a User who uses the app.

Variables:

int user\_id; // Private, unique id per user

String first\_name;

String last\_name;

String username;

String email;

String password;

bool isCoordinator = False; // If User is a Coordinator (default False; if True, make Coordinator object)

List<Club> clubs; // list of clubs a User follows

List<String> preferences; // list of preferences

### 3.1.1 getUsernameFromEmail(String email)

A String function that returns a username from a given email of a User object.

Parameter(s):

email: A String of an email

The function will search through all Users and match them with the parameter email to find the username attached with it.

### 3.1.2 getUserIDFromEmail(String email)

An int function that returns a user\_id from a given email.

Parameter:

email(s): A String object of an email

The function will search through all Users and match them with the parameter email to find the user\_id attached to it.

### 3.1.3 addClub(Club club)

A void function that adds a Club object (club) to the List clubs in the User object.

Parameter(s):

club: A Club object

The function will add the Club object (club) to the List clubs in the User object.

### 3.1.4 removeClub(Club club)

A void function that removes a Club object (club) from the List clubs in the User object.

Parameter(s):

Club: A Club object

The function will search through the List clubs in the User object and match them with the parameter club. If found, delete the club from the List.

### 3.1.5 addPreference(String preference)

A void function that adds a String object (preference) to the List preferences in the User object.

Parameter(s):

preference: A String object of a preference

The function will add the String object (preference) to the List preferences in the User object.

### 3.1.6 removePreference(String preference)

A void function that removes a String object (preference) from the List preferences in the User object.

Parameter(s):

preference: A String object of a preference

The function will search through the List preferences in the User object and match them with the parameter preference. If found, delete the preference from the List.

### 3.1.7 resetPassword(String email, String newPassword)

A void function that changes the password of a User object from a given email.

Parameter(s):

email: A String object of an email

newPassword: A String object of a new password from user input

The function will search through all Users and match them with the parameter email to find the User object attached to it. If found, change the password variable with the newPassword parameter.

### 3.1.8 removeUser(int user\_id)

A void function that removes a User object from all the Users from a given user\_id.

Parameter(s):

user\_id: An integer of a private, unique user\_id

The function will search through all Users and match them with the parameter user\_id to find the User object attached to it. If found, delete the User object.

### 3.1.9 isEmailWithUser(String email)

A bool function that returns True or False if a given email is already connected with an active, not yet deleted, User object.

Parameter(s):

email: A String object of an email

The function will search through all Users and match them with the parameter email to find the User object attached to it. If found, return True. If not found, return False.

### 3.1.10 isUsernameWithUser(String username)

A bool function that returns True or False if a given username is already connected with an active, not yet deleted, User object.

Parameter(s):

username: A String object of an username

The function will search through all Users and match them with the parameter username to find the User object attached to it. If found, return True. If not found, return False.

## 3.2 Coordinator Class

The Coordinator Class is a subclass of User with more permission on the app.

Coordinator page where the coordinator tools exist.

Variables (in addition to User variables):  
 Club coordinatedClub;

List<Event> managedEvents; // The ongoing events managed by the Coordinator

### 3.2.1 setCoordinator()

Initializes the User as a Coordinator of a certain club that will be set later

this.isCoordinator = true;

//this.isAdmin = true;

### 3.2.2 setClub(Club club)

This function assigns a club for the Coordinator to have permission to coordinate

Parameter(s):

* club: A Club object to determine the club they can access

this.coordinatedClub = club;

club.adminList.append(this);

The function will set the Coordinator’s coordinatedClub to be the club specified in the parameters.

### 3.2.3 hostEvent(Event event)

Adds an event to the club hosted by the coordinator

Parameter(s):

* event: an Event object that will be added to this Coordinator’s club

this.coordinatedClub.addEvent(event) // adding event to Club class using method from club class.

If event !in managedEvents:

this.managedEvents.append(event)

If event !in this.coordinatedClub.getEventList()

this.coordinatedClub.removeEvent(event) // adding event to Club class using method from club class.

### 3.2.4 endEvent(Event event)

After an event is finished, it will be removed from the club and the ongoing events

Parameter(s):

* event: an Event object that will be added to this Coordinator’s club

If event is in managedEvents

this.managedEvents.remove(event) // the actual code will be added later on

If event is in this.coordinatedClub.getEventList()

this.coordinatedClub.removeEvent(event) // removing event from Club class using method from club class.

event.deleteEvent() //deconstruct event using method from event class

### 3.2.4 demote()

Step down from position as Coordinator

this.isCoordinator = false;

The Coordinator object is deconstructed and turned back into a normal User

### 3.2.5 getManagedEvents()

Returns the list of events that are current ongoing and can be modified by the Coordinator

### 3.2.6 changeEventName(Event event, String name)

If the coordinator is managing an event, they can change its name

Parameter(s):

* event: an Event class object to be edited
* name: a string that will be the new name for the event

If event in this.managedEvents

event.name = name;

### 3.2.7 changeEventDesc(Event event, String desc)

Parameter(s):

* event: an Event class object to be edited
* desc: a string that will be the new description for the event

If event in this.managedEvents

event.description = desc;

### 3.2.8 changeEventTime(Event event, String time, String date)

Parameter(s):

* event: an Event class object to be edited
* time: a string that will be the new time for the event
* date: a string that will be the new date for the event

If event in this.managedEvents

event.time = time;

event.date = date;

### 3.2.9 getClub()

Returns the club that the Coordinator is associated with

Return this.coordinatedClub

### 3.2.10 leaveClub()

Unassociates the Cooridnator with their previosuly managed club.

this.coordinatedClub = null;

## 3.3 Calendar Class

// Variables

String calendarTitle;

List eventList;

String uploadedCalendarFile;

List sharedWith;

List combinedCalendars;

### **3.3.1 createCalendar(String title)**

A void function that creates a new calendar with the given title.

**Parameter(s):**

title: A String representing the title of the calendar.

The function initializes a new calendar object with the specified title and an empty list of events.

### **3.3.2 deleteCalendar()**

A void function that deletes the current calendar.

**Parameter(s):**

None

The function removes the calendar from storage and clears all associated data. A confirmation dialog will be displayed before deletion.

### **3.3.3 addEventToCalendar(Event event)**

A void function that adds an event to the calendar.

**Parameter(s):**

event: An Event object representing the event to be added.

The function checks if the event already exists in the calendar and, if not, adds it to the event list.

### **3.3.4 removeEventFromCalendar(String eventID)**

A void function that removes an event from the calendar.

**Parameter(s):**

eventID: A String representing the unique identifier of the event.

The function searches for the event by its ID and removes it from the event list.

### **3.3.5 uploadCalendar(String filePath)**

A void function that uploads a pre-existing calendar.

**Parameter(s):**

filePath: A String representing the file path of the calendar to upload.

The function loads the calendar from the specified file and imports its events into the system.

### **3.3.6 shareCalendar(String userID)**

A void function that shares the calendar with a specified user.

**Parameter(s):**

userID: A String representing the unique identifier of the user to share the calendar with.

The function adds the user to the shared list, granting them access to the calendar.

### **3.3.7 combineCalendars(Calendar calendar, Calendar otherCalendar)**

A void function that merges another calendar with the current calendar.

**Parameter(s):**

otherCalendar: A Calendar object representing the calendar to be combined.

The function merges the events from both calendars into a single calendar while avoiding duplicate events.

### **3.3.8 getEventList(Calander calendar)**

A List function that returns a list of events in the calendar.

**Parameter(s):**

Calander

The function retrieves and returns all events currently stored in the calendar.

### **3.3.9 getSharedWithList()**

A List function that returns a list of users the calendar is shared with.

**Parameter(s):**

None

The function retrieves and returns a list of user IDs that have access to the calendar.

### **3.3.10 getCombinedCalendarsList()**

A List function that returns a list of calendars that have been combined.

**Parameter(s):**

None

The function retrieves and returns a list of all calendars merged with the current one.

### **3.3.11 isSharedWith(String userID)**

A bool function that checks if the calendar is shared with a specific user.

**Parameter(s):**

userID: A String representing the unique identifier of the user.

The function searches the shared list to determine if the specified user has access to the calendar.

### **3.3.12 getEventCount(Calander calendar)**

An int function that returns the number of events in the calendar.

**Parameter(s):**

Calendar: The current calendar that we are checking

The function counts and returns the total number of events in the calendar’s event list.

## 3.4 Event Class

Event Class

The event class is responsible for the creation, and functions of event instances.

### 3.4.1 function

This function

## 3.5 Club Class

### This class holds all the information of a created club.

**Variables:**

### String clubName; // Name of club

### String desc; // Description of club

### List memberIDs; // or Map

### List adminIDs; // Or bool isAdmin in a Map of userIDs

### List events; // list of events holding event objects

### 3.5.1 addAdmin(String userID);

A function that returns nothing.  
Given a userID input:

If userID not in member list:

Add userID to member list

If userID not in admin list:  
 Adds userID to admin list

### 3.5.2 removeAdmin(String userID);

A function that returns nothing.

Given a userID input:  
 If userID is in admin list:  
 Removes userID from admin list

### 3.5.3 addMember(String userID);

A function that returns nothing.

Given a userID input:

If userID is not in member list:

Add userID to member list

### 3.5.4 removeMember(String userID);

A function that returns nothing.

Given a userID input:

If userID is in member list:

Remove userID from member list

### 3.5.5 changeDesc(String newDesc);

A function that returns nothing.

Given a newDesc input:

Update desc to newDesc

### 3.5.6 addEvent(Event event);

A function that returns nothing.

Given an event input:

If event is not in event list:

Add event to event list

### 3.5.7 removeEvent(String eventID);

A function that returns nothing.

Given an eventID input:

If eventID is in event list:

Remove event from event list

### 3.5.8 isAdmin(String userID);

A function that returns bool variable.

Given a userID input:

Check if userID in admin list

### 3.5.9 isMember(String userID);

A function that returns bool variable.

Given a userID input:

Check if userID in member list

### 3.5.10 getAdminCount();

A function that returns int variable of admin count.

### 3.5.11 getMemberCount();

A function that returns int variable of member count.

### 3.5.12 getEventCount();

A function that returns int variable of event count.

### 3.5.13 getAdminList();

A function that returns list of admins.

### 3.5.14 getMemberList();

A function that returns list of members.

### 3.5.15 getEventList();

A function that returns list of events.

# Appendix A – Definitions

Definitions

# Appendix B – Abbreviations

Abbreviations

# Appendix C – References

Bill Mongan *NLiVE Design Document v. 1.0* Microsoft Press, 2005.