YOOBEE COLLEGES

Bachelor of Software Engineering

Diploma in Software Development

CS205

Investigative Studio 2

(NZQF Level 6, 45 credits)

Stage II

System Requirement Specifications

Final Document

Team members:
Anastasiia Karpova

This document complements the Implementation and Research Proposal documents included with this submission

Main updates from the SRS Proposal are tracked or outlined in green

This report belongs to:
Chris (Hui Hui) Chong
210758235@yoobeestudent.ac.nz

Table of Contents

Introduction

Purpose

The purpose of this document is to present a detailed description of our Boostly application . It will explain the purpose and main features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both the stakeholders and the developers of the system and has been approved by the client.

Scope of project

This software system will be a cloud-deployed application that enhances an existing scheduling application. The application is an addon created for small business owners who are already using the Timely Appointment Scheduling application (https://www.gettimely.com/), and would benefit significantly from having an automated waitlist, where clients on the waitlist are able to receive automated notifications of slot availability when another client cancels or changes their appointment booking. This functionality is usually carried out manually by the business owner, who messages each client on their waitlist one at a time, waits for a response, then moves to the next person. Having this waitlist addon would save up to a days' worth of time, and would allow the business owner to focus on their clients and activities that would help to increase or enhance their business.

Overall description

The Boostly application has six main actors – three human actors (the Boostly Team Member, the User, and the User's Client) interacting with a system that comprises of three main service providers (Timely, Google, and AWS). Two main interactions from the system's side are triggers for serverless functions, which set in motion scripts and processes that help to process and store Event data, as well as alert the User or their Clients.

Because we have used multiple services within both the Google and AWS ecosystem, describing the different service functions can be a little complex. The Main Use Case

Diagram aims to provide a broad overview of what each actor is trying to achieve, while the use cases themselves try to provide a clearer breakdown of the implementation details.

Potential limitations

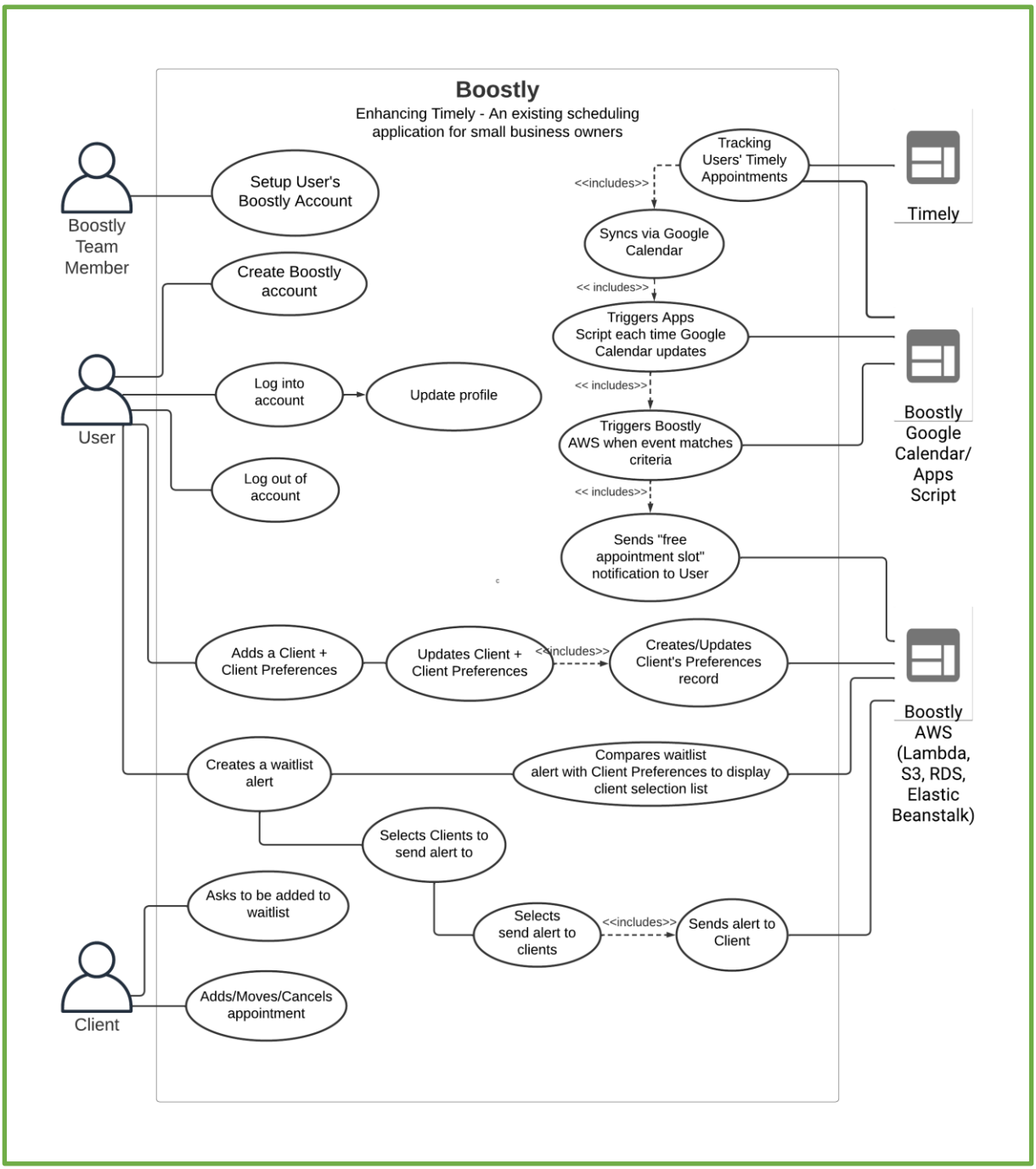
While we have identified and prioritised key functional requirements for our MVP, the entire project hinges on the business owner being able to help form that initial link between Timely (external application) and Boostly (our application). Because Timely does not have an API that we can use to GET and POST requests, the user interaction for the initial setup process has a high possibility of being quite clunky, and possibly not one that we are able to guide users through smoothly. As a worst-case scenario, Boostly will have to offer "professional services" to assist with the initial setup of the Boostly-Timely sync. Due to time constraints, we will only be focusing on the system sending email notifications at this stage. This has been pre-discussed with the client (our tutor) and accepted as a compromise.

Defined User characteristics

The User has been defined as small business owners in the service sector that are time poor and already clients of Timely. End users should therefore have some level of ability to use technology (even if they are technologically challenged). Their gender or ethnicity should not matter, and the language will initially be set as English since that's the only language that Timely is currently offered in.

The User's Clients are likely to come from a large demographics pool due to the varying nature of service businesses. Users might be technologically competent, or fully unable to work with technology. For the MVP, we have chosen to design the process so that the User is the one who adds the clients to the waitlist and manages the clients' preferences.

Functional Requirements Specification



Based on the requirements gathered, research conducted, user scenarios created, and Proof of Concepts (PoCs) conducted (detailed in the main report), we have derived use cases for the User (business owner) and User's Client. We decided to put more emphasis on the initial sync between Boostly and Timely, since that is the trigger and forms the basis of all the data used in our application. As a compromise to meet project timeline constraints, we have chosen to transfer all waitlist management functions to the User, instead of setting up separate client-accessible accounts.

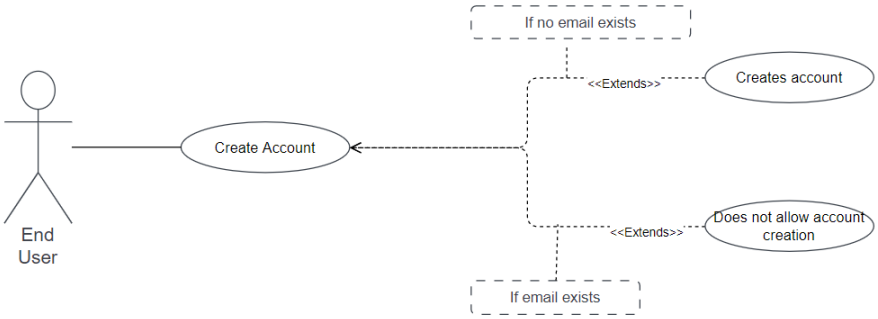
User Use Cases

The User has the following sets of use cases:

1. Access account (create, login)
2. Access account information (view and update)
3. Manage client waitlist
4. Boostly Team Member setup
5. Triggering Waitlist Alerts

Access Account use cases

Use case: Create Account



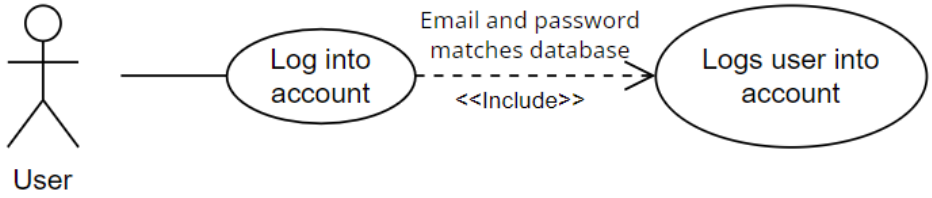
Brief Description
The User creates their personal Boostly account.

Initial Step-By-Step Description
No prerequisites, although the user should have an active Timely account

1. The system presents a choice of logging in or signing up
2. The User selects the 'sign up' link on the login screen
3. The system displays request for new account information – First name, Last name, Email address and Password
4. The End-User enters information and selects 'Create Account'
5. The system checks that the email address is valid and not already in use
6. The system checks that the password meets the requirements (length)
7. If step 5 and/or 6 fails, the system notifies the User and returns to step 3
8. The system creates a hash of the User's password. It then creates a new record in the database and stores the user's account information to that record
9. The End-User is returned to the dashboard, which informs them that a Boostly Team Member will contact them to help setup their Timely-Boostly sync
10. The system sends a notification to the Boostly Setup Team

Explanation:
Account creation should be fast and easy, and only information that is required should be stored. Account verification will not be implemented for the MVP. User password has been hashed for security purposes. The notification to the Boostly Setup Team hasn't been implemented in this MVP since all alpha users have been selected and onboarded manually.

Use case: Log into Account



Brief Description
The User logs into their Boostly account

Initial Step-By-Step Description
Before this use case can be initiated, the User has already created an account.

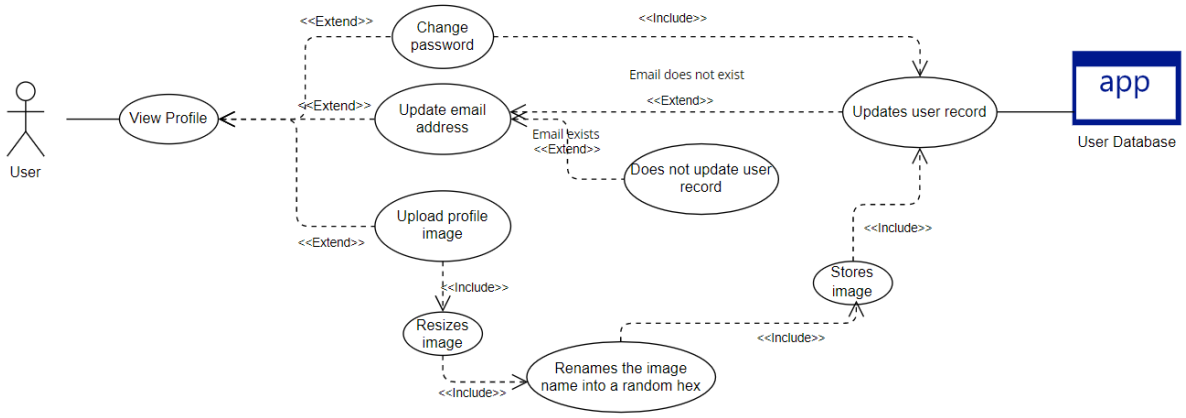
Xref: Create Account

1. The system presents a choice of logging in or signing up
2. The User enters their email address and password, then selects the 'Log In' button
3. The system validates their credentials by checking their email address and password against the database
4. If the email address matches an email address in the database, the system displays the User dashboard

Explanation:
For security purposes, passwords need to be hashed when stored so that, even if the database is compromised, the users' passwords aren't exposed. Because this is Boostly's first addon and the MVP, the logged in dashboard will (at this time) be the Activate Waitlist Addon screen.

View Account information use case

Use case: View/Update Account Details



Brief Description
The User views their account profile and has the option of updating it

Initial Step-By-Step Description
Before this use case can be initiated, the User will have logged into their Boostly account.

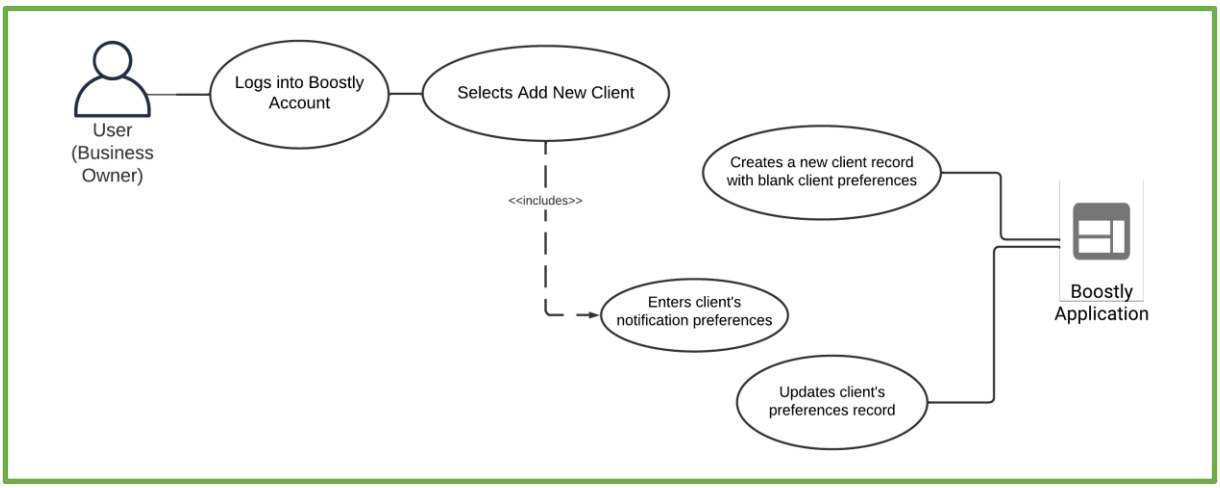
Xref: Log into Account

1. The User selects the “Account” button
2. The system displays the user’s account information (Profile picture, First Name, Last Name, Email address and Password (hidden))
3. If the user enters a new password, the system will update the client’s record to reflect the new password
4. If the user enters a new email address, the system will check to see if the email address exists in the database. If it does, an error message will be shown. Otherwise it will update the client’s record to reflect the new email address
5. If the user uploads a display picture, the system will resize the image to no more than 125px, rename the image name to a random hex value, store the image in a static folder, then update the client record to reflect the new image name

Explanation:
To save space, images should be resized to no more than 125 x 125 px. Because images from different clients might end up having similar names, the system needs to generate and change the name to a random hex value before storing it to the client record.

Client Waitlist Management use case

Use case: Add Client and Client's Preferences to waitlist



Brief Description

The process of the User adding a client and the client's preferences are linked, and have been combined into one use case to reduce repetition.

Initial Step-By-Step Description

Before this use case can be initiated, the User will have created their Boostly account. It is preferable (although not mandatory) for the User to have sync-ed their Timely calendar with Boostly.

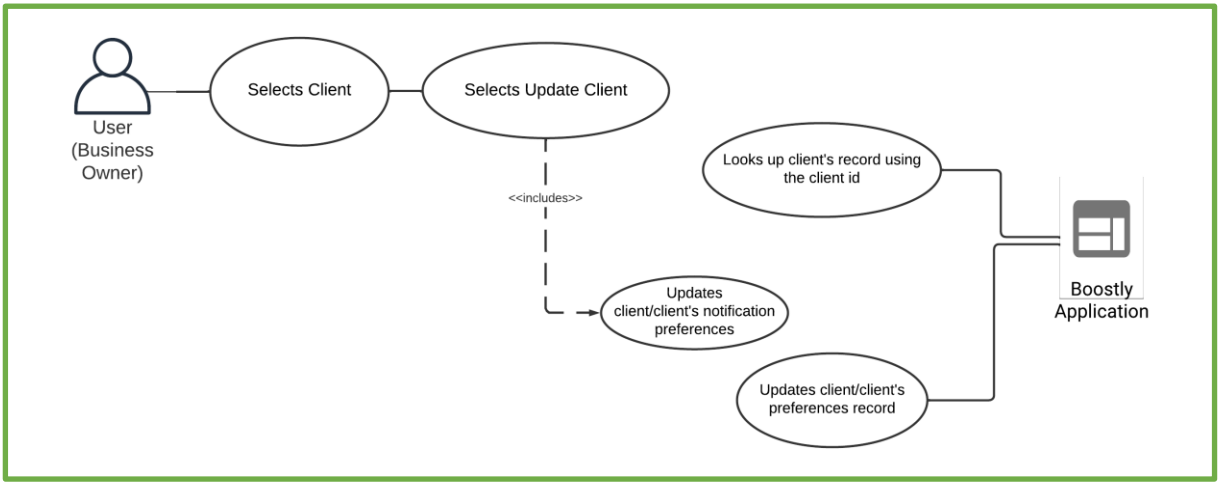
Xref: Create Account, Log into Account

1. The User selects a client (or the client's preferences) in their waitlist, and selects Update Client
2. The system looks up the client's record using the client's id and displays the client's information (or preferences)
3. The User makes changes to their client's record or client's notification preferences
4. The system looks up the client's id and updates their information (or preferences) to their record

Explanation:

We have simplified this process and not included a function that queries the database to check if a client with that information exists.

Use case: Update Client and Client's Preferences to waitlist



Brief Description

The process of the User updating a client and the client's preferences are very similar and have been combined into one use case to reduce repetition.

Initial Step-By-Step Description

Before this use case can be initiated, the User will have created their Boostly account, logged in, and added a client/client preference record. It is preferable (although not mandatory) for the User to have sync-ed their Timely calendar with Boostly.

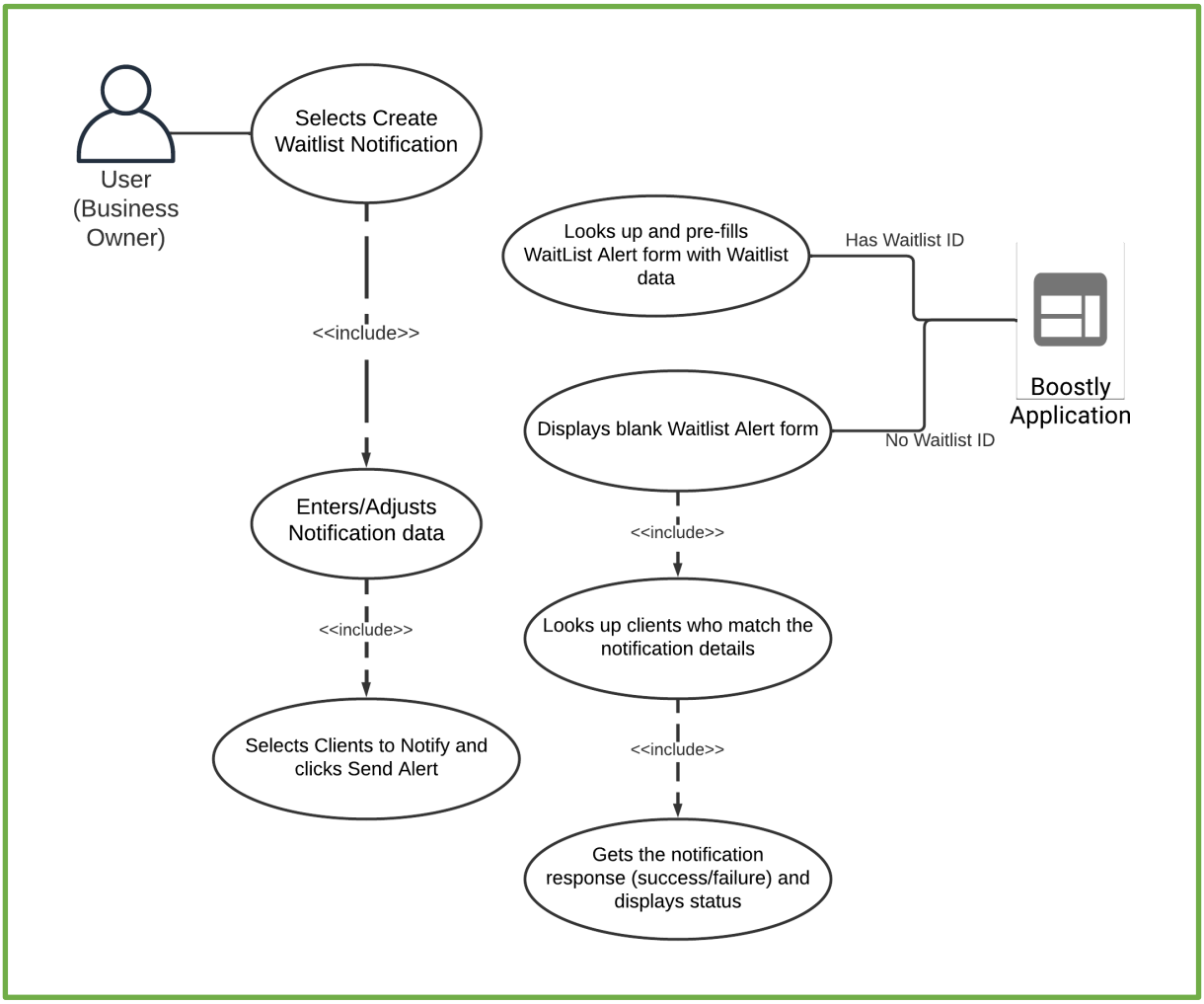
Xref: Create Account, Log into Account, Add Client/Client Preferences

1. The User selects a client (or the client's preferences) in their waitlist, and selects Update Client
2. The system looks up the client's record using the client's id and displays the client's information (or preferences)
3. The User makes changes to their client's record or client's notification preferences
4. The system looks up the client's id and updates their information (or preferences) to their record

Explanation:

We have simplified this process to make user interaction simple, repetitive, and easy to learn

Use Case: Create and Send a Waitlist Notification



Brief Description

A user can either create a new notification alert from scratch, or might get to that notification page via a link from their email, triggered by a client appointment change or cancellation. The process from then on is identical.

Initial Step-By-Step Description

Before this use case can be initiated, the User will have created their Boostly account, logged in, added a client/client preference record, and synced their Timely Calendar with Boostly

Xref: Create Account, Log into Account, Add Client/Client Preferences , Sync Time-Boostly Calendar

1. The User selects a client (or client preferences) in their waitlist, and selects Update Client
2. If the Notification route contains a WaitList id, it looks up the waitlist details and displays it as an editable form. Otherwise, it just presents a blank form.

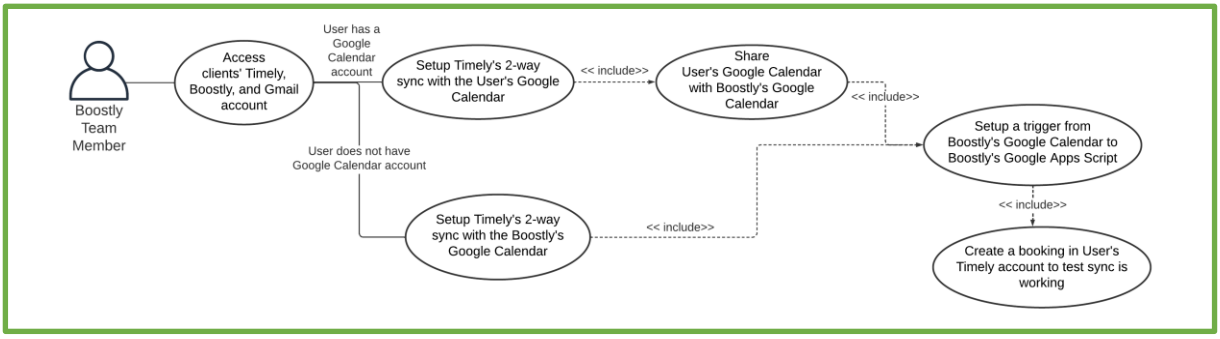
3. The User adds/adjusts the waitlist notification information and presses the “Select Alert Recipient” button.
4. The system makes a query for clients who belong to the User in the system, and whose preferences match the notification information provided by the User and returns a list of Clients that want to be alerted for a slot on that day
5. The User selects the clients that they want to alert and presses the “Send Alert” button
6. The system sends email notifications to the selected Clients, then gets the SMTP response and displays a history of all of the User’s alerts along with the alert status

Explanation:
This process is the main functionality that the User will see, and the interaction therefore needs to be easy and smooth. Pushing the alert id through the route means that the User does not have to transfer the empty slot details over to the Waitlist notification form, saving time and reducing the change of human error. It also helps to ensure that the User’s event details are not exposed

Boostly Team Member Use Case

The Boostly Team Member has the single use case of setting up the Timely-Boostly synchronization.

Use case: Synchronise the User’s Timely account with Boostly



Brief Description
This is a manual step where a Boostly team member will have to help setup the User’s Timely to Boostly calendar synchronisation

Initial Step-By-Step Description
Before this use case can be initiated, the User will have created their Boostly account.

Xref: Create Account

1. The User is contacted about their account setup
2. The Boostly team member requests for access to the User's Timely and Gmail accounts via a remote access and control system (i.e. Teamviewer) so that we never receive/store any external credentials
3. The User logs into accounts, and the Boostly Team Member helps to setup Timely's 2-way sync with the User's Google Calendar
4. The Boostly team member will share the User's Google Calendar to our Boostly Google Calendar. If the User does not have a Google calendar and has no intention of getting one, the Boostly team member will sync the User's Timely account directly with the Boostly Calendar
5. The Boostly Team member then sets up a trigger that executes Boostly's Google Apps Script each time the calendar is updated.
6. The Boostly Team member creates a test booking in the User's Timely account to test if the sync is working.

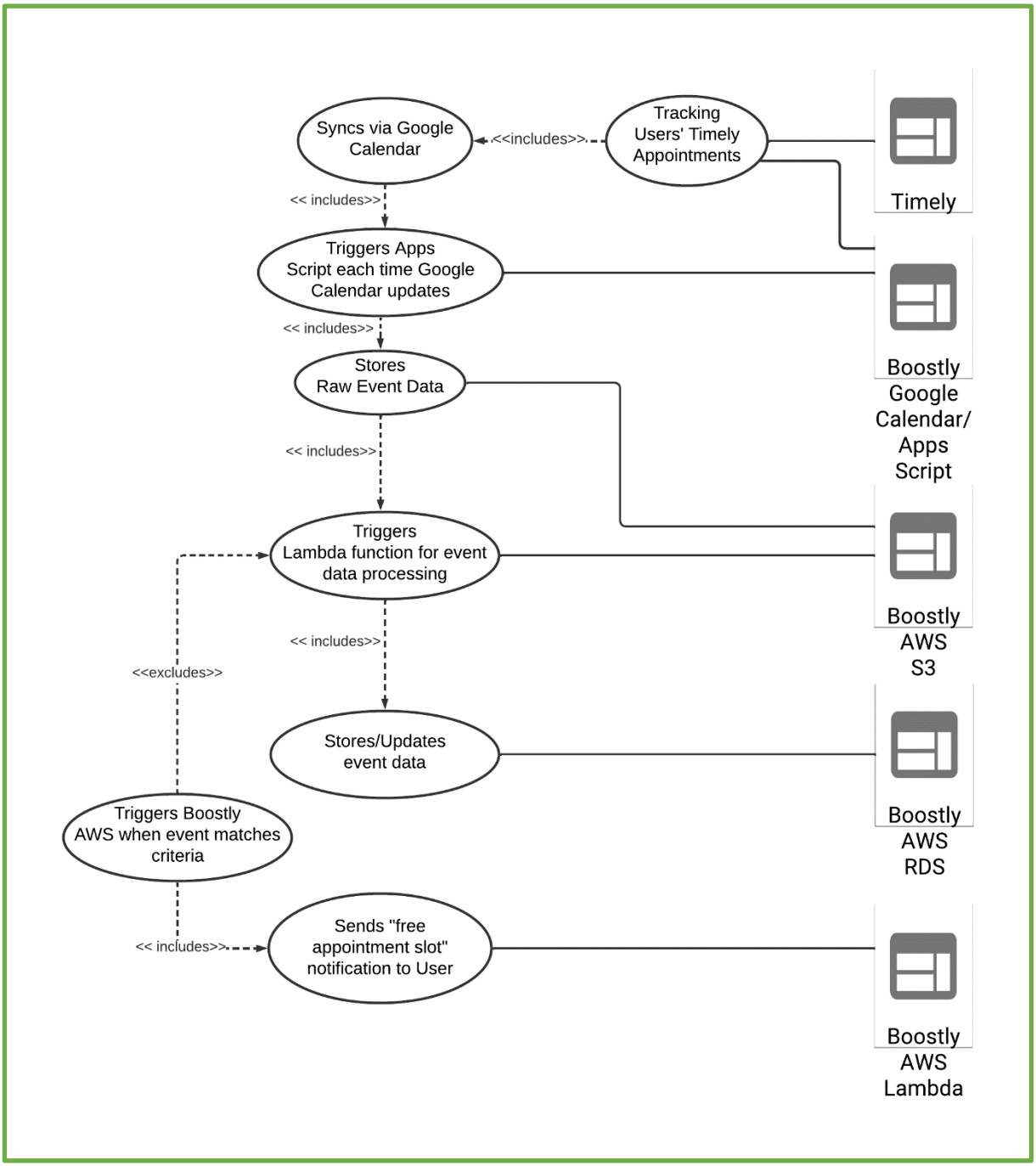
Explanation:
This interaction is quite technically complex for a User to execute and cannot be easily automated as we do not have access to Timely's APIs. We have chosen to keep this onboarding process manual since this is a one-time setup. This way, we can focus on the other key functionality.

Triggering Waitlist Alerts Use cases

In addition to the use cases documented above where the system plays a part, the system also has a separate use case that does not involve any human actors, namely:

1. Tracking, triggering, and sending alerts to User

Use case: Tracking, triggering, and sending alerts to User



Brief Description

Movements/cancellations in the User's Timely calendar triggers a notification to the User

Initial Step-By-Step Description

Before this use case can be initiated, the User will have created their Boostly account, and a

Boostly Team Member would have synced the User's Timely calendar with their Boostly account.

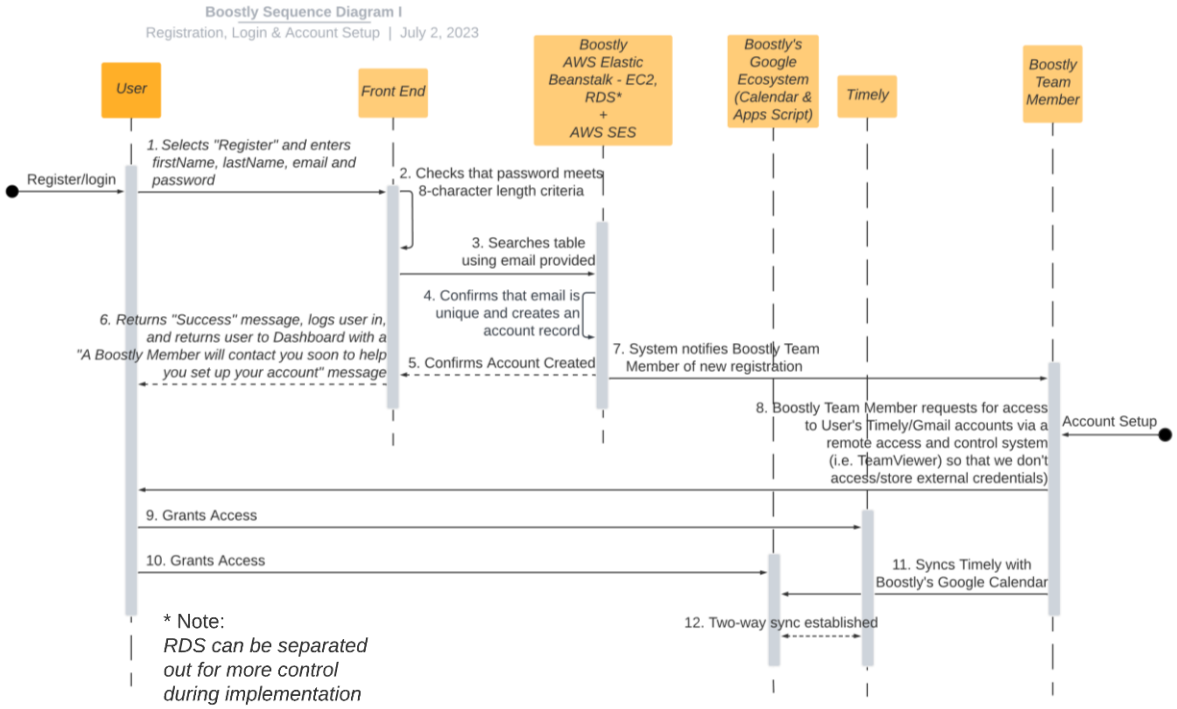
Xref: Create Account, Setup Boostly-Timely Synchronisation

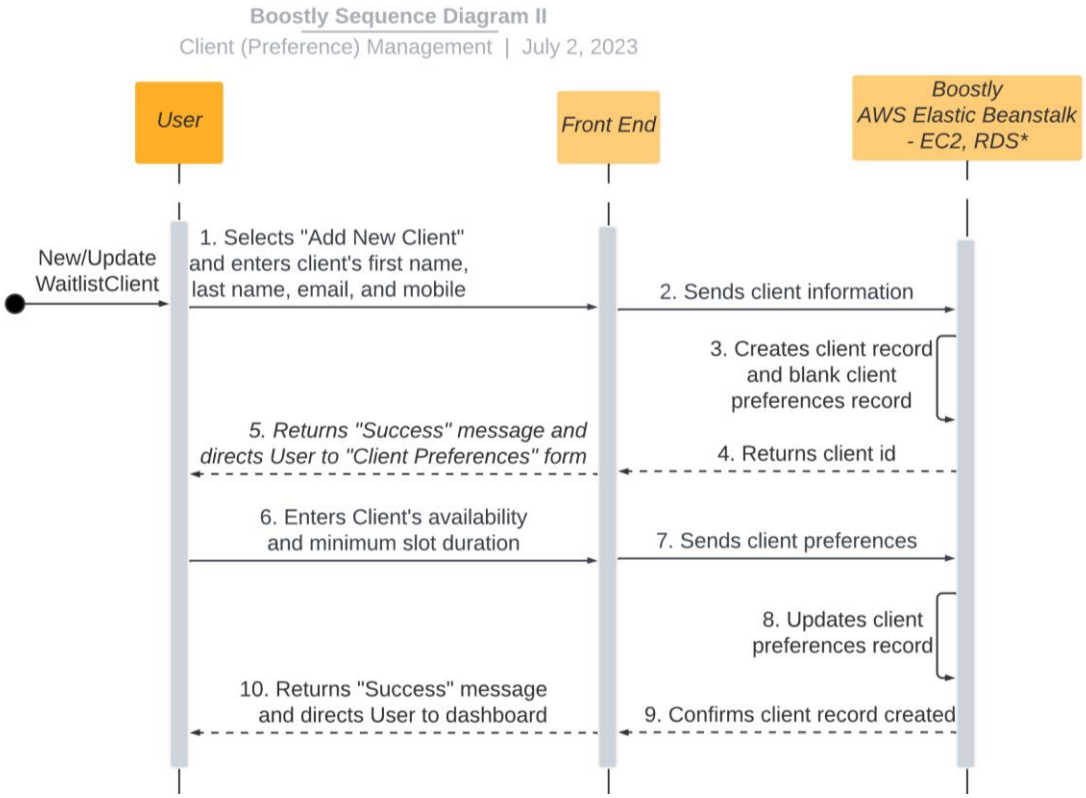
1. The system tracks the User's Timely appointments via their Google calendar sync
2. When an event is added, moved, or cancelled, Google calendar sends a push notification that executes Boostly's Google Apps Script
3. Boostly's Google Apps Script distills the event to key information, then puts it into Boostly's AWS S3 bucket
4. When an object is stored in Boostly's AWS S3 bucket, it triggers a primary AWS Lambda function
5. The Lambda function processes the event information, then stores and updates event data to the Boostly RDS database
6. If the processed event information indicates that a free slot has opened in the User's calendar, the primary Lambda function will trigger a child Lambda function
7. The child Lambda function uses AWS Simple Email Service to send an email notification to the User
8. The User receives the alert, and decides if they want to Create and Send a Waitlist Notification

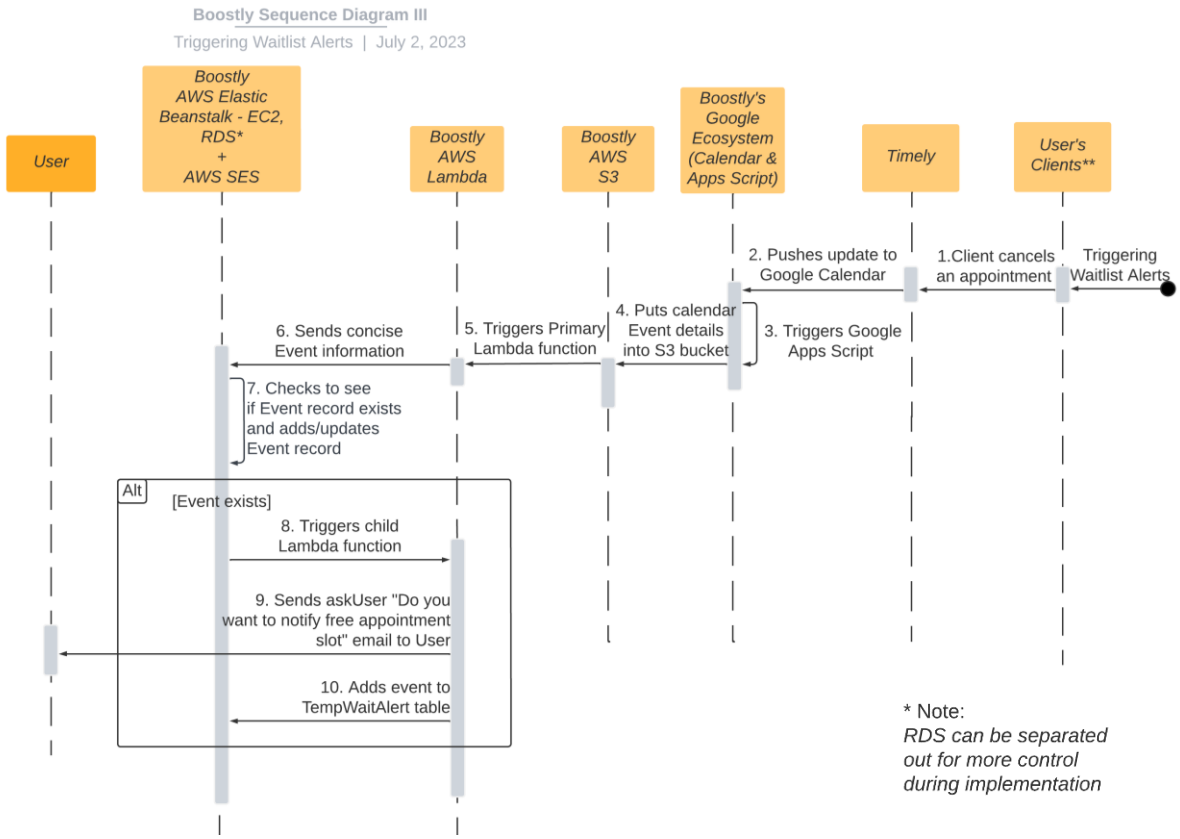
Explanation:
Because Timely does not have a public API, tracking the User's schedule (to determine when a notification can be sent) requires the cooperation of three different service providers, with multiple sub-services in the cloud providers. Establishing a sync with Timely's calender through their two-way sync is not foolproof, but is the most reliable method for tracking that data. Serverless functions have been harnessed to process the data in those events, and determine whether the User should be notified.

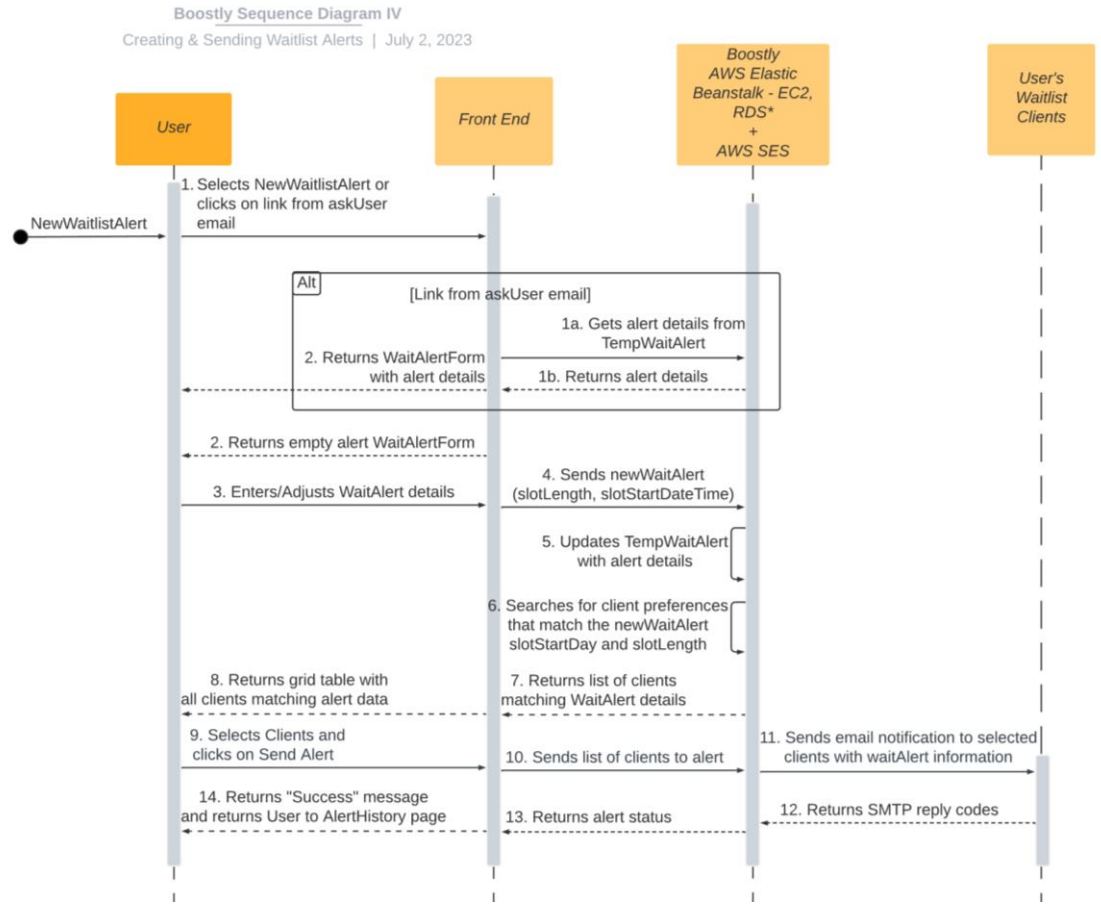
Sequence Diagram

The interactions identified in the use case diagrams have helped us create our sequence diagrams below. These have been split into 4 main diagrams to better portray the sequence of events that take place.



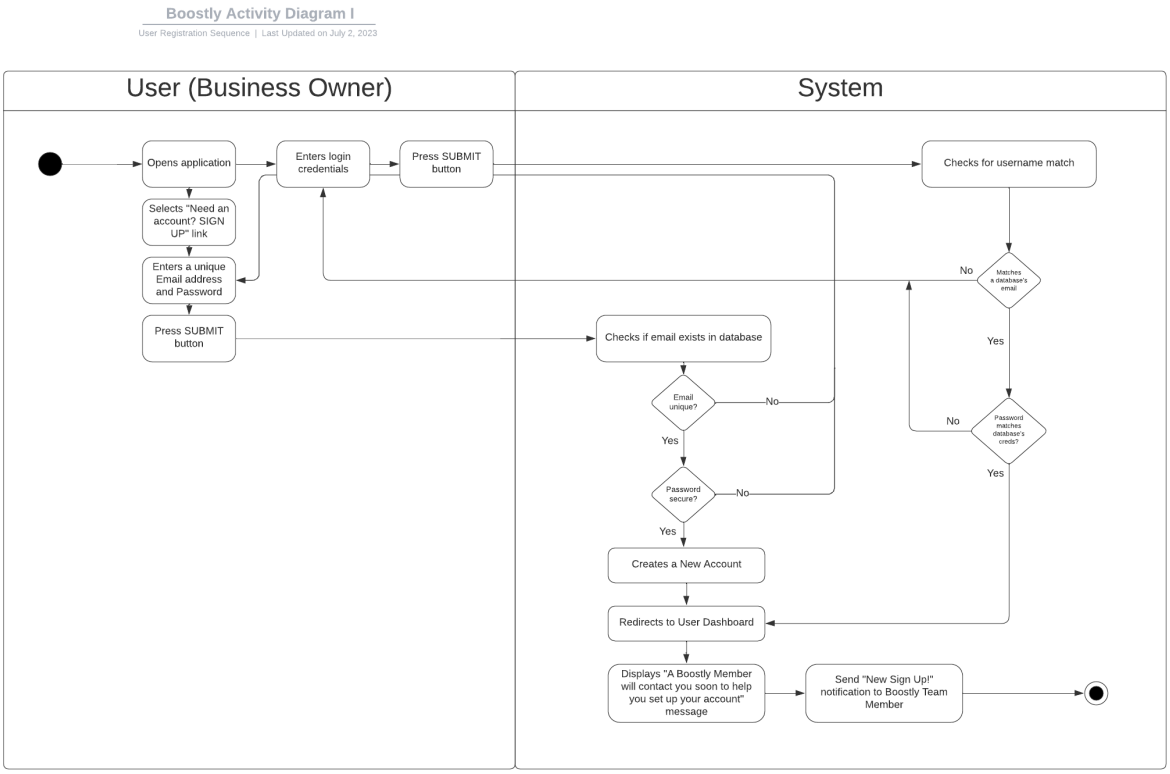



Boostly Sequence Diagram III
Triggering Waitlist Alerts | July 2, 2023
User
Boostly AWS Elastic Beanstalk - EC2, RDS* + AWS SES
Boostly AWS Lambda
Boostly AWS S3
Boostly's Google Ecosystem (Calendar & Apps Script)
Timely
User's Clients**
2. Pushes update to Google Calendar
1.Client cancels an appointment
Triggering Waitlist Alerts
4. Puts calendar Event details into S3 bucket
3. Triggers Google Apps Script
6. Sends concise Event information
5. Triggers Primary Lambda function
7. Checks to see if Event record exists and adds/updates Event record
Alt
[Event exists]
8. Triggers child Lambda function
9. Sends askUser "Do you want to notify free appointment slot" email to User
10. Adds event to TempWaitAlert table
* Note:
RDS can be separated out for more control during implementation

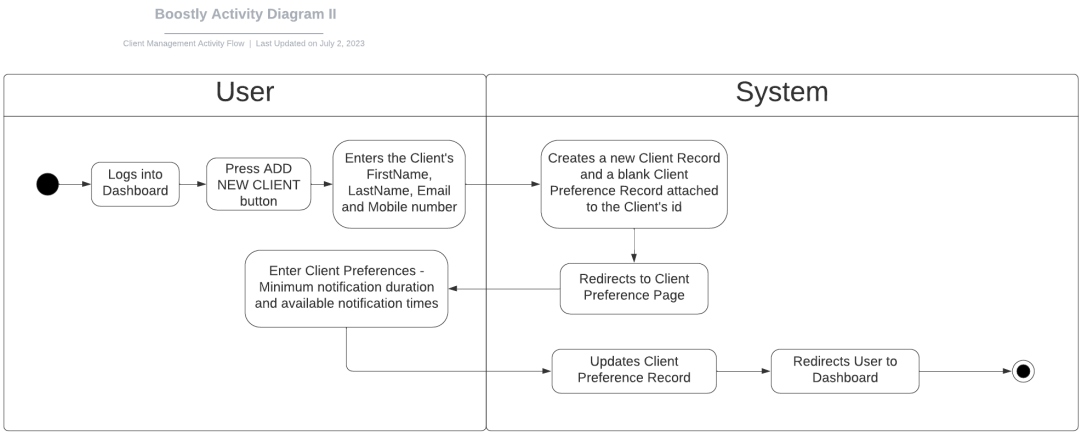

Boostly Sequence Diagram IV
Creating & Sending Waitlist Alerts | July 2, 2023
User
Front End
Boostly AWS Elastic Beanstalk - EC2, RDS* + AWS SES
User's Waitlist Clients
NewWaitlistAlert
1. Selects NewWaitlistAlert or clicks on link from askUser email
Alt
[Link from askUser email]
1a. Gets alert details from TempWaitAlert
2. Returns WaitAlertForm with alert details
1b. Returns alert details
2. Returns empty alert WaitAlertForm
3. Enters/Adjusts WaitAlert details
4. Sends newWaitAlert (slotLength, slotStartDateTime)
5. Updates TempWaitAlert with alert details
6. Searches for client preferences that match the newWaitAlert slotStartDay and slotLength
8. Returns grid table with all clients matching alert data
7. Returns list of clients matching WaitAlert details
9. Selects Clients and clicks on Send Alert
10. Sends list of clients to alert
11. Sends email notification to selected clients with waitAlert information
14. Returns "Success" message and returns User to AlertHistory page
13. Returns alert status
12. Returns SMTP reply codes

Activity Diagrams

Each action and the decision points have been detailed using four different activity diagrams

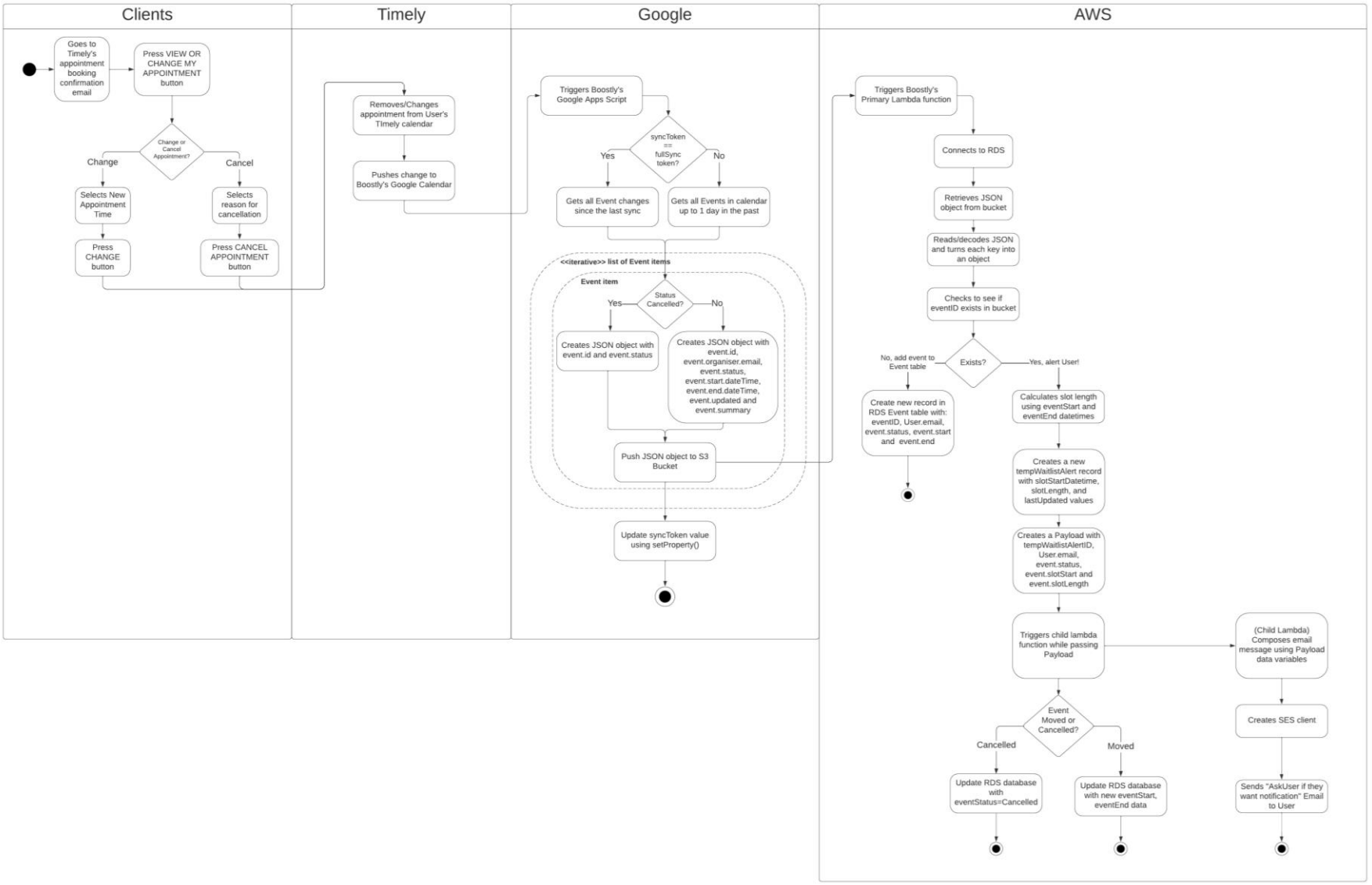


This diagram shows the flow of actions from the User to the System in order to register for a Boostly Account. Having received a “New Sign Up notification, a Boostly Team Member will have to complete the setup process by starting a remote access (i.e. Teamviewer) session with the User and syncing Timely with Boostly’s Google Calendar by asking them to sign into their Timely account and Google Calendar. This is not an ideal process and needs to be rectified before the product can go live. This has been discussed with our tutor and accepted as a compromise for our Boostly MVP so that we can focus on the other parts of the notification process.



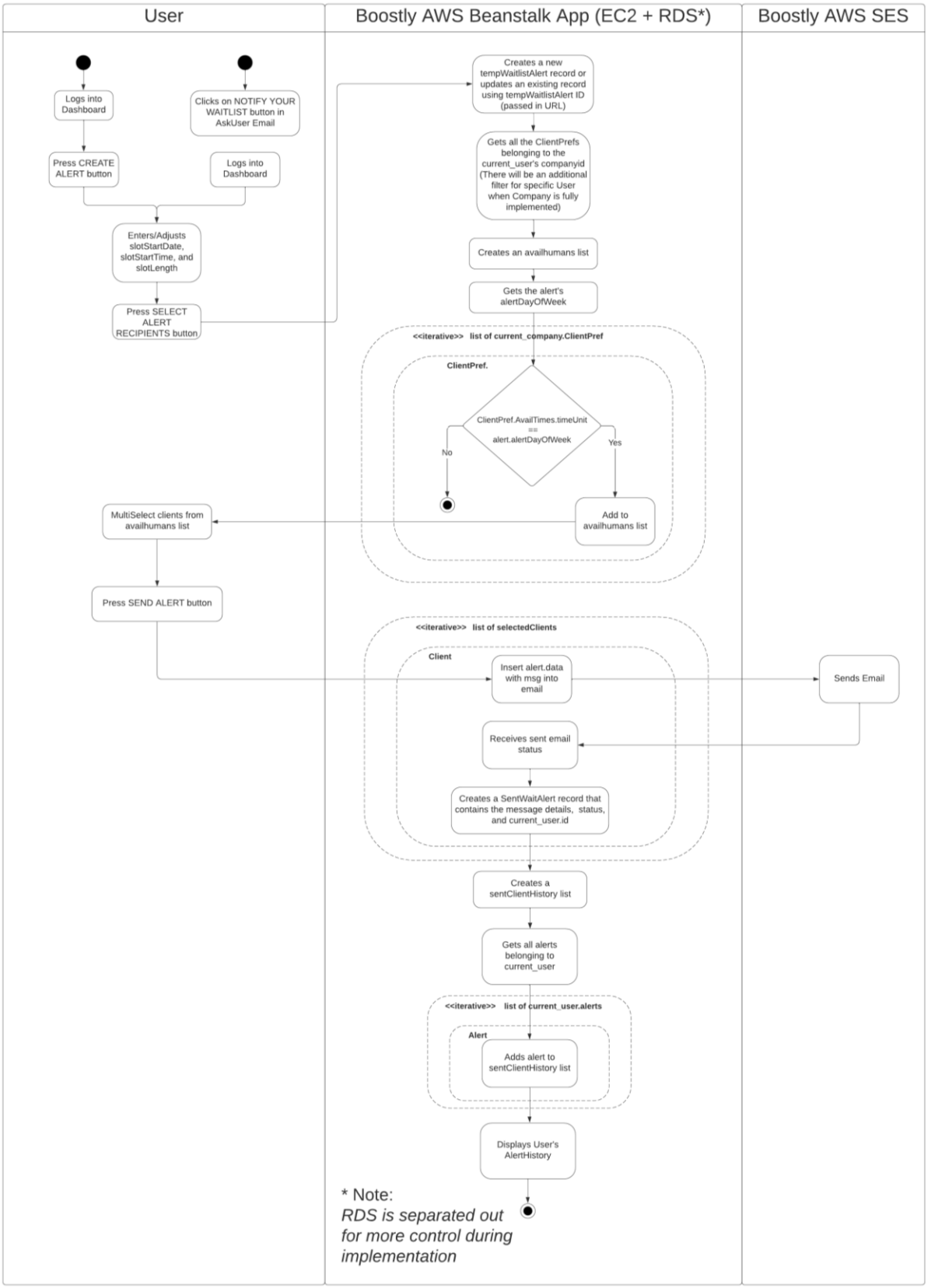
Boostly Activity Diagram III

Triggering Waitlist Alert Activity Flow | Last Updated on July 3, 2023



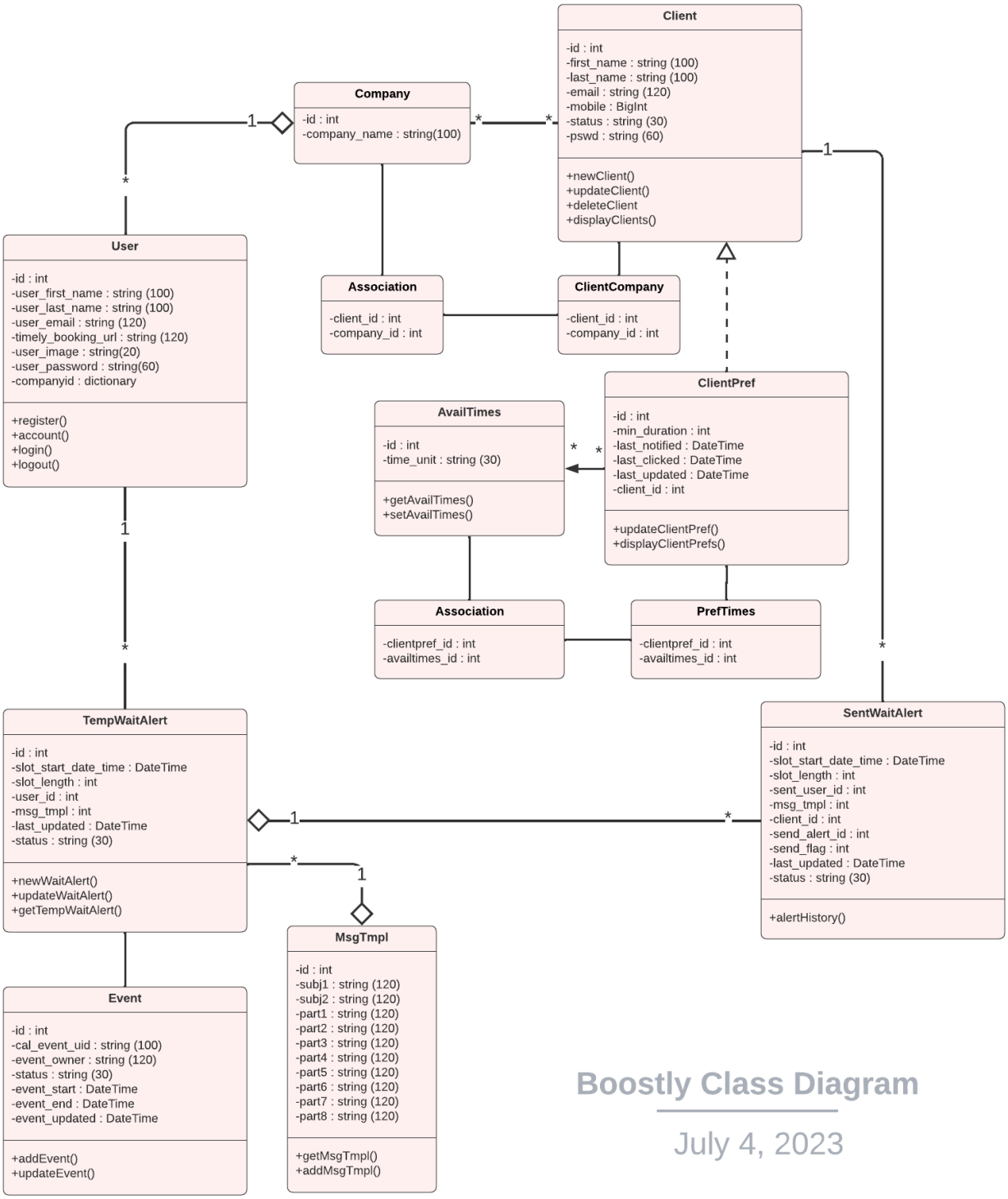
Boostly Activity Diagram IV

Creating & Sending Waitlist Alerts | Last Updated on July 3, 2023



Class Diagram

This class diagram gives an overview of the main classes within Boostly. The diagram shows how the User indirectly relates to the Client through the Company. The Event class stores the least information required, and TempWaitAlert and SentWaitAlert holds no Personal Identifiable Information to provide a direct link between the User and their Events.



I have also decided to separate the Client's PrefTimes from the ClientPref table for two reasons:

- The ClientPref table will get quite large and unwieldy holding the detailed units of time that the clients are available for, especially if the units of time are depicted in string format (i.e., MondayAM, MondayPM etc.)

By making that separation, I can now also have an AvailTimes table that stores the client's preferences in an extremely flexible way. Because of the challenges of presenting a UX/UI with an easy-to-select availability schedule, we have chosen to only break down the time availability units into days. With this implementation, however, it would be extremely easy to change the time units to provide more fine-grained control. For example, the Client could request to not be alerted for Mondays to Fridays from 8AM to 3PM but would be available for any slots that become available after 3PM.

Non-Functional Requirements Specification

This iteration will only address the specific non-functional requirements in this implementation. Our primary requirement here is to save business owners time when compared with their current process.

Because this is a cloud deployed application, our secondary objective is to try our best to create a system that is easy to scale and has no single point of failure . Due to costs and AWS' free tier limitations, we would likely be unable to deploy the application with high availability, but we would want our system design to at least have the ability to run multiple instances easily to handle scale.

However, because we expected to spend most of our time trying to implement the best way to sync with an external application without API endpoints (Timely), this has been placed as a secondary requirement for our MVP. We have, instead, relied on the capabilities of AWS Elastic Beanstalk to manage that high availability and scale on our behalf, which is one of the major benefits to using a cloud provider (as opposed to self-hosting or web hosting providers). There is a lot more to explore in terms of the fine-tuning of the Elastic Beanstalk configurations, but this will not be included in our MVP.

Requirements Validation

These use case diagrams have been vetted by the client (our tutor), and approval has been received to focus on the synchronisation of Timely with our system, Boostly, as opposed to i.e. the onboarding process, or providing the User's Clients self-managing capability. Our goal therefore is to prioritise building core functions before considering areas such as a seamless user setup process, easy scalability, or high availability.