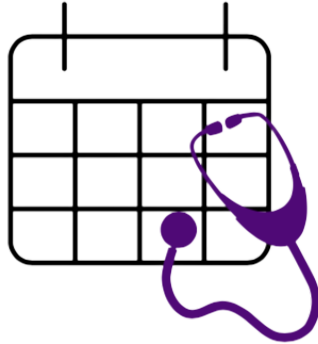

TCU & UNTHSC Volunteer Scheduler



Developer Manual
Version 2.4

Revision History

Date	Version	Description	Author
2020	1.0	Started “How to Run” and “Accessing Database” sections	not Lydia Pape
Early 2021	1.1	Started API document	Lydia Pape
2021	1.2	Updated API document	Lydia Pape
04/18/2021	2.0	Updated “How to Run” instructions	Lydia Pape
04/18/2021	2.1	Added instructions on how to run the front end (Section 2.2)	Maria Amoros
04/26/2021	2.2	Removed some redundant instructions from how to run back end section (already covered in how to run front end section)	Lydia Pape
04/26/2021	2.2	Updated API document with small changes to request methods for Bulk Delete and Find by Email	Lydia Pape
04/27/2021	2.3	Started section on PostgreSQL Heroku deployment	Lydia Pape
05/04/2021	2.4	Finished PostgreSQL section (3.2), added sections on Firebase and Deployment to Heroku	Riley Durbin
05/04/2021	2.4	Filled in Section 7: Domain Name Management	Jeshua Suarez-Lugo

Table of Contents

1. Introduction	4
2. Getting Started	4
2.1 How to run backend	4
2.2 How to run frontend	6
3 Accessing Database	8
3.1 H2 Console in Browser	8
3.2 PostgreSQL Heroku Server	8
4 Communication Between Front end and Back end	9
4.1 API Document	9
5. Firebase Management	30
6. Deployment to Heroku	30
7. Domain name management	31

1. Introduction

1.1 Purpose

The purpose of this document is to provide current and future developers of this project an explanation on how the software operates and how to use it.

1.2 Project Overview

The TCU & UNTHSC Volunteer Scheduler is a web application built using Springboot for its backend, and Vue.js for its frontend. It consists of the major components of:

- **Vue.js front end:** progressive framework for building user interfaces.
- **Spring Boot back end:** Using Java 11, it does all the business logic and handles requests sent from the front-end.
- **Firebase Authentication:** Users are being authenticated and their accounts' passwords being managed.
- **SQL Database:** relational database that stores all data in the system

2. Getting Started

The project's front end and back end are in separate Github repositories:

- Volunteer-App for the front end
- Server-Volunteer-App for the back end

The latter of these is deployed using Heroku, and includes a place for a front end build. Each time a commit is pushed to Server-Volunteer-App's main branch, the entire project is redeployed with any new changes applied.

2.1 How to run backend

First, you must install all dependencies using Maven. Go to the `/server` directory and type:

```
./mvnw clean install
```

Then, to run the server type:

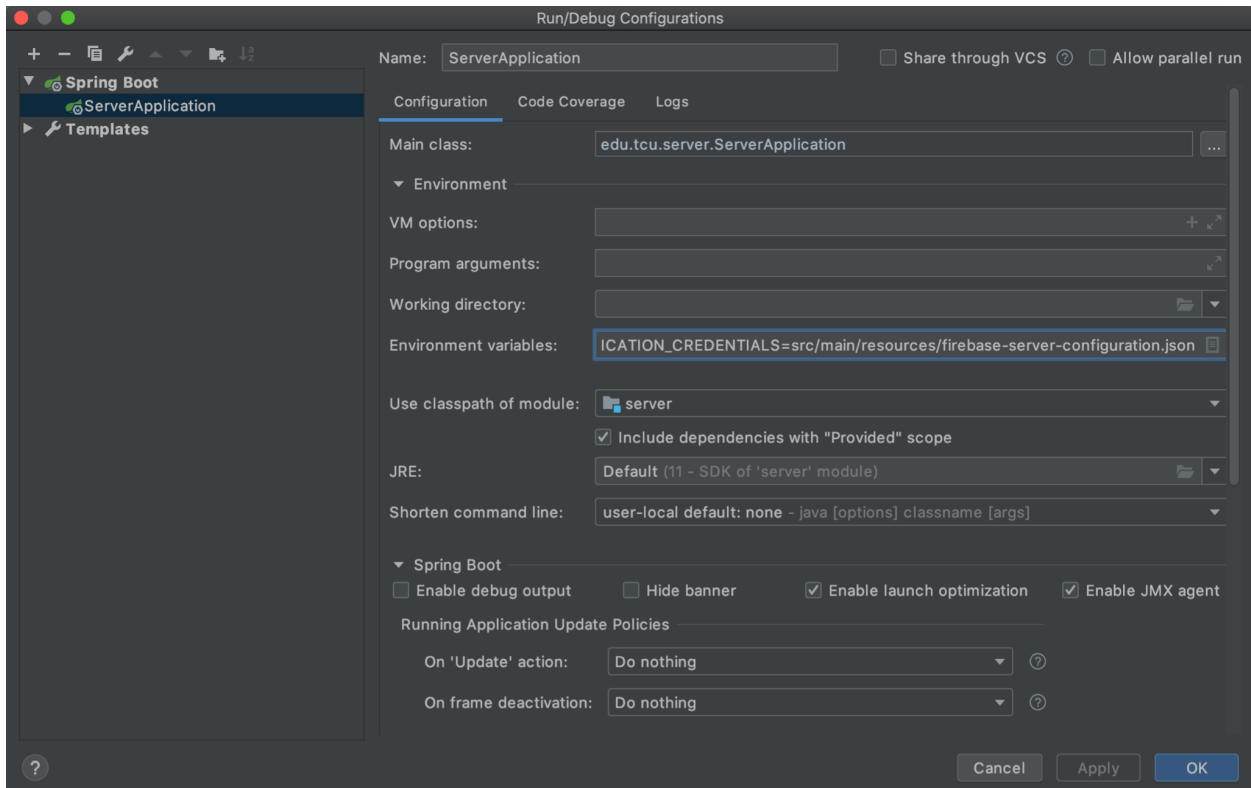
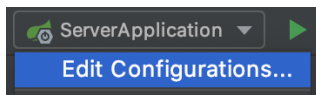
```
./mvnw spring-boot:run
```

The above command is the same thing as running the project in IntelliJ.

Further setup may be necessary to develop and run the back end using IntelliJ:

- Right-click pom.xml and select "make this a Maven project" (or similar)

- Add this Environment variable to ServerApplication’s configuration (see screenshots):
GOOGLE_APPLICATION_CREDENTIALS=src/main/resources/firebase-server-configuration.json (don’t forget to “Apply”)



- Find the Maven tab on the right and run “clean” then “install”

NOTE on testing the project locally:

Due to a complication with the deployment with Heroku, dates/times will be displayed with the wrong time zone when running the project locally. There is a fix for this in the back end’s code, commented out, in two places in CalendarEvent.java and two more in Attendance.java. Applying this fix allows the times to display correctly when running the project locally, but will mess them up in the deployed version. Here is what it looks like, for example, in CalendarEvent.java:

```

81
82 // @JsonFormat(pattern = "dd MMMMMMMM", 'yyyy' at 'h:mm a', timezone = "America/Chicago", shape = JsonFormat.Shape.STRING) // nice String format
83 // @JsonFormat(pattern = "yyyy-MM-dd" 'h:mm a', timezone = "America/Chicago", shape = JsonFormat.Shape.STRING) // best format for calendar (unused)
84 @JsonFormat(pattern = "dd MMMMMMMM", 'yyyy' at 'h:mm a', shape = JsonFormat.Shape.STRING) // nice String format, no timezone alteration
85 private Date startDateTime;
86 // @JsonFormat(pattern = "dd MMMMMMMM", 'yyyy' at 'h:mm a', timezone = "America/Chicago", shape = JsonFormat.Shape.STRING) // nice String format
87 // @JsonFormat(pattern = "yyyy-MM-dd" 'h:mm a', timezone = "America/Chicago", shape = JsonFormat.Shape.STRING) // best format for calendar (unused)
88 @JsonFormat(pattern = "dd MMMMMMMM", 'yyyy' at 'h:mm a', shape = JsonFormat.Shape.STRING) // nice String format, no timezone alteration
89 private Date endDateTime;
90

```

To apply the time zone fix for running locally, uncomment the first JsonFormat annotation for each of these two variables (and, similarly, in Attendance.java) and comment out the last one.

2.2 How to run frontend

Node.js:

It is an open source server environment that is used to run and host the front end of the project while developing. The version that we are using is the 14.16.1 LTS, and it can be found for installation at <https://nodejs.org>.

Npm: We use the node package manager to install several different packages to be used while developing in the front end, and it is installed when Node.js is installed.

Vue.js:

This project uses for the front end the progressive JavaScript framework vue.js. The version that we are using is 3, and it is not compatible with vue 2. Per the vue.js website, this framework is built by design to be incrementally adoptable. This means that it can be integrated into a project multiple ways depending on the requirements. For this project, it needs to be installed using npm by running the following command:

```
npm install vue@next
```

For Vue 3, you should use Vue CLI v4.5 available on npm as @vue/cli and can be installed as following:

```
npm install -g @vue/cli
```

In case it is too far in the future, and you already have the vue CLI installed, you may want to simply upgrade by running

```
vue upgrade --next
```

Additionally, axios is a lightweight HTTP client based on the XMLHttpRequests service used to perform HTTP requests, and it can be installed by running

```
npm install --save axios vue-axios
```

In addition, there are many other dependencies in the front end that currently are as follows and may need to be updated depending on how these dependencies have been updated.

- Vuex
- Vue-router
- Vue-full-calendar
- Primevue
- Primeicons

- Firebase
- Eventservice
- Element-ui
- Core-js
- @types/webpack-env
- @fullcalendar/timegrid
- @fullcalendar/interaction
- @fullcalendar/daygrid
- @fullcalendar/core

The dependencies for the development in the front end are the following

- Babel CLI
- Babel core
- vue/cli-plugin-router
- vue/cli-service
- vue/cli-plugin-babel
- Sass-loader
- Vue-loader

For now all of these can be found and seen at the package.json.

In order to run the project and be able to see it in localhost:8080, you will need to run in the terminal the following:

```
npm run serve
```

However, if trying to run the backend as well to test the website, it will not allow you to run since it will attempt to run it as well in port 8080. This is why where the front end runs can be changed by having a vue config file. Currently, you can find one in our project and it is called vue.config.js and only has the following lines of code:

```
Module.exports = {  
  devServer: {  
    Port: 3000  
  }  
}
```

This port can be changed in case anything else is running in port 3000.

In addition, since there are two repositories as mentioned above (one for the front end and one for the backend), you need to compile and minify the front end for production and this can be done by running:

```
npm run build
```

And since the deployment is done in the server repository, what needs to be done to have all of your front end changes, is to

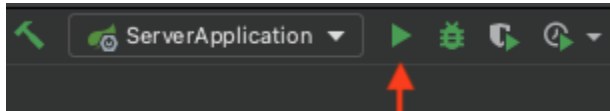
1. Delete everything in `/Server-Volunteer-App/src/main/resources/static/`
2. In the Volunteer-App frontend, run `npm run build`
3. Copy everything from `/Volunteer-App/dist` into `/Server-Volunteer-App/src/main/resources/static/`
4. Run the spring boot project

3 Accessing Database

3.1 H2 Console in Browser

The data that has been persisted through the database or has been added in `DBDataInitializer.java` and can be accessed by doing the following:

1. Run the spring boot application

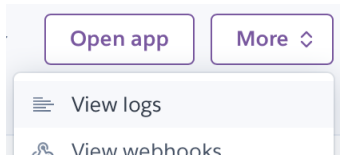


2. Open your browser and navigate to `localhost:8080/h2-console`
3. Press "Connect" and you can now enter queries to the database.

3.2 PostgreSQL Heroku Server

1. Go to <https://id.heroku.com/login>
2. Log in
3. Select `volunteer-server-springproject`

You can now access the back end console by going to More > View logs



This can be used to find any console messages printed by the back end, which might be useful information when someone using the deployed website experiences an error or a bug.

To browse the Database:

One way to browse the PostgreSQL database is through this tool. It is initially a 7-day free trial but could be useful if you want to manually edit the database using a UI in the future :

1. Navigate to: <https://datazenit.com/heroku-data-explorer.html>
2. Sign in with your Heroku account
3. Select “volunteer-server-springproject”
4. Run a query or click one of the tables on the left-hand side of the screen.

You may also connect to the database using the terminal:

1. Make sure you have heroku-cli installed on your machine
 - a. <https://devcenter.heroku.com/articles/heroku-cli>
2. Make sure you have the psql command installed on your machine
 - a. <https://devcenter.heroku.com/articles/heroku-postgresql#local-setup>
3. In the terminal, run the command:
 - a. `heroku pg:psql --app volunteer-server-springproject`
 - b. (Note that there are two dashes before “app” in the command above.)

4 Communication Between Front end and Back end

4.1 API Document

All responses (the contents of `response.data` in the front end) (except with Reports) will include a “flag” which will be **true** for success and **false** for failure, and a “code” which will be 200 for success or an error code for failure. The “message” and “data” fields will vary depending on the request. **All requests must include a header “Authorization” containing the encoded Firebase token String.**

The backend will send date/time information to the frontend as a String:

- "dd MMMMMMMMM", 'yyyy' at 'h:mm a"
- example "24 February, 2021 at 9:25 PM"
- Exception: Waitlisted timestamp field will be a long value instead of a String.

The frontend will send date/time information to the backend using four fields (see relevant examples for specifics on what to name the fields):

- Year, as a number (such as 2021)
- Month, as a string (such as “February”) (capitalization does not matter, but must be spelled right)
- Day, as a number (such as 12)
- Time, as a string (such as “12:10 p.m.”) (AM or PM must be present, but the exact form it takes does not matter; there doesn’t even have to be a space)
 - Other time examples that should parse just fine (this is not all of them)
 - “12:10 PM”
 - “12:10pm”
 - “12:10p.m.”
 - “12:10 P”

Where necessary, fields can be left blank: String fields can be an empty String, null, or any invalid String, and numbers can be null or 0. It also works to leave the fields out altogether. The backend will send back empty number fields as 0, and empty String, Date, and Enum fields as null.

[Some of the examples below are outdated](#): since they were captured, the “link” field has been added to Events and the “status” field has been removed from Users.

Rows marked **“admins only”** will only work if the person responsible for the request is logged in as an admin.
 Rows that deal with the **current** User rely completely on authentication with the Firebase token.

Request Purpose	Request Type	URL Endpoint	Request Body	Example	Response (in case of success)	Example
Find all Event categories	GET	/calendar /categories”	none	N/A	“message”: “Find All Categories Success”, “data”: [list of the names of all Event categories]	<pre> { "flag": true, "code": 200, "message": "Find All Categories Success", "data": ["TEST1", "TEST2", "TEST3"] } </pre> <p>The values being returned for this request come from the properties instead of the database-- one can change them by editing application.yml.</p>
Find all Events the current User is eligible for	GET	/calendar /eligible”	none	N/A	“message”: “Find Eligible Events Success”, “data”: [list of all Events the current User is eligible for]	<pre> { "flag": true, "code": 200, "message": "Find All Success", "data": [{ "eventId": 1, "title": "Friday meeting", "category": null, "sex": null, "description": null, "address": null, "startDateTime": "05 March, 2021 at 11:00 AM", "endDateTime": "05 March, 2021 at 12:00 PM", "nonFacultyCapacity": 6, "facultyCapacity": 2, "gradYear": 0, "comment": null, "host": { "partnerId": 1, "name": "Pet Clinic", "email": "example@pets4ever.org" }, "active": false, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 3, "numFacultySignedUp": 0, "numAttended": 3 }, { "eventId": 2, "title": "Spring break", "category": null, </pre>

<p>Find all Events (admins only)</p>	<p>GET</p>	<p>"/calendar"</p>	<p>none</p>	<p>N/A</p>	<p>"message": "Find All Success", "data": [list of all events; empty list if there are no events]</p>	<pre> { "flag": true, "code": 200, "message": "Find All Success", "data": [{ "eventId": 1, "title": "Friday meeting", "category": null, "sex": null, "description": null, "address": null, "startDateTime": "05 March, 2021 at 11:00 AM", "endDateTime": "05 March, 2021 at 12:00 PM", "nonFacultyCapacity": 6, "facultyCapacity": 2, "gradYear": 0, "comment": null, "host": { "partnerId": 1, "name": "Pet Clinic", "email": "example@pets4ever.org" }, "active": false, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 3, "numFacultySignedUp": 0, "numAttended": 3 }, { "eventId": 2, "title": "Spring break", "category": null, }] } </pre>
<p>Find an Event by ID</p>	<p>GET</p>	<p>"/calendar/{eventId}"</p>	<p>none</p>	<p>URL: localhost:8080/calendar/4</p>	<p>"message": "Find One Success", "data": [event's data]</p>	<pre> { "flag": true, "code": 200, "message": "Find One Success", "data": { "eventId": 4, "title": "Event 1", "category": "TEST1", "sex": null, "description": null, "address": null, "startDateTime": "25 February, 2021 at 8:00 AM", "endDateTime": "25 February, 2021 at 12:00 PM", "nonFacultyCapacity": 50, "facultyCapacity": 25, "gradYear": 0, "comment": null, "host": { "partnerId": 3, "name": "TCU", "email": "blank@tcu.edu" }, "active": false, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 1, "numFacultySignedUp": 0, "numAttended": 1 } } </pre>

<p>Save a new Event (admins only)</p>	<p>POST</p>	<p>"/calendar"</p>	<p>Data for event, but don't worry about ID</p> <p>New field added: String "link"</p> <p>NOTE: to assign a host organization, the "organization" field must contain the organization's ID (would consider renaming to "partnerId")</p>	<pre>{ "title": "new", "organization": 1, "description": "new description", "address": "Yoshi's Island", "category": null, "startingYear": 2021, "startingMonth": "June", "startingDay": 7, "startingTime": "7:25 AM", "endingYear": 2021, "endingMonth": "June", "endingDay": 7, "endingTime": "7:30 AM", "gradYear": 2021, "sex": "", "comment": "Contact Dr. Bonnell if you have", "nonFacultyCapacity": 14, "facultyCapacity": 6 }</pre> <p>(the whole comment isn't shown; it will likely be a longer string)</p> <p>Default sex: ALL</p>	<p>"message": "Save Success", "data": [new Event]</p>	<pre>{ "flag": true, "code": 200, "message": "Save Success", "data": { "eventId": 12, "title": "new", "category": null, "sex": "ALL", "description": "new description", "address": "Yoshi's Island", "startDateTime": "07 June, 2021 at 7:25 AM", "endDateTime": "07 June, 2021 at 7:30 AM", "nonFacultyCapacity": 14, "facultyCapacity": 6, "gradYear": 2021, "comment": "Contact Dr. Bonnell if you have t", "host": { "partnerId": 1, "name": "Pet Clinic", "email": "example@pets4ever.org" }, "active": true, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 0, "numFacultySignedUp": 0, "numAttended": 0 } }</pre>
<p>Update an Event (admins only)</p>	<p>PUT</p>	<p>"/calendar /{eventId}"</p>	<p>Same as for saving a new event; the data will be applied to the event with the given ID</p> <p>Fields left blank on update will NOT be overwritten.</p>	<p>URL: localhost:8080/calendar/4</p> <p>Body:</p> <pre>{ "title": "Event 1", "organization": 1, "description": "this event has a description now!", "address": "", "category": "TEST1", "startingYear": 2021, "startingMonth": "June", "startingDay": 7, "startingTime": "7:25 AM", "endingYear": 2021, "endingMonth": "June", "endingDay": 7, "endingTime": "7:30 AM", "gradYear": 2021, "sex": "Male", "comment": "and now there is a comment", "nonFacultyCapacity": 20, "facultyCapacity": 10 }</pre>	<p>"message": "Update Success", "data": [updated Event]</p>	<pre>{ "flag": true, "code": 200, "message": "Update Success", "data": { "eventId": 4, "title": "Event 1", "category": "TEST1", "sex": "MALE", "description": "this event has a description now!", "address": null, "startDateTime": "07 June, 2021 at 7:25 AM", "endDateTime": "07 June, 2021 at 7:30 AM", "nonFacultyCapacity": 20, "facultyCapacity": 10, "gradYear": 2021, "comment": "and now there is a comment", "host": { "partnerId": 1, "name": "Pet Clinic", "email": "example@pets4ever.org" }, "active": true, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 1, "numFacultySignedUp": 0, "numAttended": 1 } }</pre>
<p>Delete an Event (admins only)</p>	<p>DELETE</p>	<p>"/calendar /{eventId}"</p>	<p>none</p>	<p>URL: localhost:8080/calendar/9</p>	<p>"message": "Delete Success", "data": null</p>	<p>N/A</p>

<p>Find an Event's Waitlist or Signup list (admins only)</p>	<p>GET</p>	<p>"/calendar/{xy}/{eventId}"</p> <p>{x}: n for <i>nonfaculty</i>, or f for <i>faculty</i></p> <p>{y}: w for <i>waitlist</i> or s for <i>signup</i> list</p>	<p>none</p>	<p>URL Examples</p> <p>Nonfaculty Waitlist: localhost:8080/calendar/nw/7</p> <p>Faculty Waitlist: localhost:8080/calendar/fw/7</p> <p>Non Faculty Sign Up list: localhost:8080/calendar/ns/7</p> <p>Faculty Signup list: localhost:8080/calendar/fs/7</p>	<p>"message": "Find [Waitlist/ Signup list] Success", "data": [list of Waitlisted objects/ Users signed up]</p> <p>(a waitlist is <i>not</i> just a list of Users)</p> <p>Note: this waitlist example shows a list with only one item, but it's still a list instead of just the item (also these are both <i>nonfaculty</i> lists)</p>	<p>Waitlist:</p> <pre> "flag": true, "code": 200, "message": "Find Waitlist Success", "data": [{ "relationId": { "user": { "userId": 7, "email": "marjavanoros@gmail.com", "type": "STUDENT", "status": null, "admin": true, "firstName": "Marjan", "lastName": "Amoros", "gradYear": 2021, "totalHours": 53.0, "hoursLeft": 22.0 } }, "calendarEvent": { "eventId": 7, "title": "Academy Cono (Apr 16)", "category": null, "sex": null, "description": null, "address": "Zoom only", "startTime": "16 April, 2021 at 12:45 PM", "endTime": "16 April, 2021 at 2:30 PM", "nonFacultyCapacity": 3, "facultyCapacity": 4, "gradYear": 2021, "comment": null, "host": { "partnerId": 4, "name": "Mercy Clinic of Fort Worth", "email": null } }, "active": true, "numNonFacultyWaitlisted": 1, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 3, "numFacultySignedUp": 2, "numAttended": 0 }], "timestamp": 161773588972 } </pre> <p>Signup list:</p> <pre> "flag": true, "code": 200, "message": "Find Signup list Success", "data": [{ "userId": 1, "email": "riley.durbin@tcu.edu", "type": "STUDENT", "status": null, "admin": false, "firstName": "Riley", "lastName": "Durbin", "gradYear": 2021, "totalHours": 0.0, "hoursLeft": 75.0 }, { "userId": 2, "email": "jin.croce@tcu.edu", "type": "STUDENT", </pre>
---	------------	--	-------------	---	---	---

<p>Find an Event's Attendance records (admins only)</p>	<p>GET</p>	<p>"/calendar /a/{eventId}"</p>	<p>none</p>	<p>URL: localhost:8080/calendar/a/1</p>	<p>"message": "Find Attendance Success", "data": [list of Attendance objects] (not just a list of Users)</p>	<pre> { "flag": true, "code": 200, "message": "Find Attendance Success", "data": { "relationId": { "user": { "userId": 3, "email": "lydia.pope@tcu.edu", "type": "STUDENT", "status": null, "admin": true, "firstName": "Lydia", "lastName": "Pope", "gradYear": 2021, "totalHours": 48.0, "hoursLeft": 27.0 } }, "calendarEvent": { "eventId": 1, "title": "Friday meeting", "category": null, "sex": null, "description": null, "address": null, "startDateTime": "05 March, 2021 at 11:00 AM", "endDateTime": "05 March, 2021 at 12:00 PM", "nonFacultyCapacity": 0, "facultyCapacity": 0, "gradYear": 0, "comment": null, "host": { "partnerId": 1, "name": "Pet Clinic", "email": "example@pets4ever.org" }, "active": false, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 0, "numFacultySignedUp": 0, "numAttended": 3 } }, "checkin": "05 March, 2021 at 11:00 AM", "checkout": "05 March, 2021 at 12:00 PM", "hours": 1.0 }, { "relationId": { "user": { </pre>
<p>Add the current User to an Event's Waitlist or Signup list</p>	<p>PUT</p>	<p>"/calendar /{x} /{eventId}" {x}: w for waitlist or s for signup list</p>	<p>none</p>	<p>URL Examples: Waitlist: localhost:8080/calendar/w/8 Signup: localhost:8080/calendar/s/8</p>	<p>"message": "[Waitlist/Signup] Success", "data": null</p>	<p>Alternate responses: Waitlist full on waitlist attempt:</p> <pre> { "flag": false, "code": 400, "message": "Could not waitlist (Event waitlist full)", "data": null } </pre> <p>Event full, but waitlist open, on signup attempt:</p> <pre> { "flag": true, "code": 200, "message": "Could not sign up (Event full); waitlisted instead", "data": null } </pre> <p>Event and waitlist both full on signup attempt:</p>

						<pre> "flag": false, "code": 404, "message": "Could not sign up— Event signup and waitlist both full", "data": null </pre>
<p>Remove the current User's Waitlist or Signup status</p>	DELETE	<p>"/calendar /{x} /{eventId}"</p> <p>{x}: same as above</p>	none	<p>URL Examples:</p> <p>Waitlist: localhost:8080/calendar/w/8</p> <p>Signup: localhost:8080/calendar/s/8</p>	<p>"message": "[Removal from Waitlist/Cancel Signup] Success",</p> <p>"data": [null for waitlist; the User who was signed up instead, or null if no one was on the waitlist]</p>	<p>For signup, when the corresponding waitlist wasn't empty:</p> <pre> "flag": true, "code": 200, "message": "Cancel Signup Success", "data": { "userId": 7, "email": "example2@tcu.edu", "type": "STUDENT", "status": null, "admin": false, "firstname": "Maria", "lastname": "Amoros", "gradYear": 2021, "totalHours": 0.0, "hoursLeft": 75.0 } </pre> <p>(In this example, someone who is not faculty cancelled their signup. Maria was on the non faculty waitlist; this indicates that she has now been signed up as a result of the cancellation.)</p>

<p>Check the current User In to or Out of an Event</p>	<p>POST</p>	<p>"/check /{eventId}"</p>	<p>Checkin date/time and/or Checkout date/time, in the same form as for events</p> <p>If only one time is given and the other time has already been stored in the DB record via a previous request, the old information will be kept for that time.</p>	<p>URL: localhost:8080/check/8</p> <p>Body:</p> <pre> { "checkoutYear": 2021, "checkoutMonth": "February", "checkoutDay": 17, "checkoutTime": "10:00 AM" } </pre> <p>If the Attendance record already exists in the DB, it will be updated with the time(s); if it doesn't exist yet then it will be created.</p>	<p>"message": "Check in / Check out Success", "data": [the Attendance record that was created or updated, including hours]</p> <p>An Attendance record may be considered valid if the hours field is nonzero.</p>	<pre> { "flag": true, "code": 200, "message": "Check in / Check out Success", "data": { "relationId": { "user": { "userId": 4, "email": "ric.bonnell@tcu.edu", "type": "FACULTY", "status": null, "admin": true, "firstName": "Ric", "lastName": "Bonnell", "gradYear": 0, "totalHours": 24.0, "hoursLeft": 0.0 } }, "calendarEvent": { "eventId": 8, "title": "Academy4 Como (May 7)", "category": null, "sex": null, "description": null, "address": "Zoom only", "startDateTime": "07 May, 2021 at 12:45 PM", "endDateTime": "07 May, 2021 at 2:30 PM", "nonFacultyCapacity": 10, "facultyCapacity": 5, "gradYear": 2024, "comment": null, "host": null, "active": true, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedup": 0, "numFacultySignedup": 0, "numAttended": 1 } } }, { "checkin": "16 February, 2021 at 10:00 AM", "checkout": "17 February, 2021 at 10:00 AM", "hours": 24.0 } } </pre> <p>In this example, Dr. Bonnell's total of 24 hours <i>includes</i> (is entirely) the 24 hours of work from which he is checking out.</p>
<p>Add a User to an Event's Waitlist or Signup list (admins only)</p>	<p>PUT</p>	<p>"/calendar /{x} /{eventId} /{userId}"</p> <p>{x}: w for <i>waitlist</i> or s for <i>signup</i> list</p>	<p>none</p>	<p>URL Examples:</p> <p>Waitlist: localhost:8080/calendar/w/8/4</p> <p>Signup: localhost:8080/calendar/s/8/4</p>	<p>"message": "[Waitlist/ Signup Success", "data": null</p>	<p>Alternate responses:</p> <p>Waitlist full on waitlist attempt:</p> <pre> { "flag": false, "code": 400, "message": "Could not waitlist (Event waitlist full)", "data": null } </pre> <p>Event full, but waitlist open, on signup attempt:</p>

		NOTE: the <i>Event's</i> ID comes first				<pre> "flag": true, "code": 289, "message": "Could not sign up (Event full); waitlisted instead", "data": null } </pre> <p>Event and waitlist both full on signup attempt:</p> <pre> "flag": false, "code": 488, "message": "Could not sign up— Event signup and waitlist both full", "data": null } </pre>
Remove a User's Waitlist or Signup status (admins only)	DELETE	<p>"/calendar /{x} /{eventId} /{userId}"</p> <p>{x}: same as above</p> <p>NOTE: the <i>Event's</i> ID comes first</p>	none	<p>URL Examples:</p> <p>Waitlist: localhost:8080/calendar/w/8/4</p> <p>Signup: localhost:8080/calendar/s/7/3</p>	<p>"message": "[Removal from Waitlist/ Cancel Signup] Success",</p> <p>"data": [null for waitlist; for signup, the User who was signed up instead, or null if no one was on the waitlist]</p>	<p>For signup, when the corresponding waitlist wasn't empty:</p> <pre> "flag": true, "code": 200, "message": "Cancel Signup Success", "data": { "userId": 7, "email": "example2@tcu.edu", "type": "STUDENT", "status": null, "admin": false, "firstname": "Maria", "lastname": "Amoros", "gradYear": 2021, "totalHours": 0.0, "hoursLeft": 75.0 } </pre> <p>(In this example, someone who is not faculty cancelled their signup. Maria was on the non faculty waitlist; this indicates that she has now been signed up as a result of the cancellation.)</p>

<p>Check a User In to or Out of an Event (admins only)</p>	<p>POST</p>	<p>"/check /{eventId} /{userId}"</p> <p>NOTE: the <i>Event's</i> ID comes first</p>	<p>Checkin date/time and/or Checkout date/time, in the same form as for events</p> <p>If only one time is given and the other time has already been stored in the DB record via a previous request, the old information will be kept for that time.</p>	<p>URL: localhost:8080/check/8/4</p> <p>Body: {"checkoutYear": 2021, "checkoutMonth": "February", "checkoutDay": 17, "checkoutTime": "10:00 AM"}</p> <p>If the Attendance record already exists in the DB, it will be updated with the time(s), but if it doesn't exist yet then it will be created.</p>	<p>"message": "Check in / Check out Success", "data": [the Attendance record that was created or updated, including hours]</p> <p>An Attendance record may be considered valid if the hours field is nonzero.</p>	<pre> "flag": true, "code": 200, "message": "Check in / Check out Success", "data": { "relationId": { "user": { "userId": 4, "email": "ric.bonnell@tcu.edu", "type": "FACULTY", "status": null, "admin": true, "firstname": "Ric", "lastname": "Bonnell", "gradYear": 0, "totalHours": 25.0, "hoursLeft": 0.0 }, "calendarEvent": { "eventId": 0, "title": "MATH", "category": "TEST1", "sex": null, "description": null, "address": null, "startDateTime": "25 March, 2021 at 4:00 AM", "endDateTime": "25 March, 2021 at 4:00 PM", "nonFacultyCapacity": 4, "facultyCapacity": 3, "gradYear": 0, "comment": null, "host": null, "active": false, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 0, "numFacultySignedUp": 1 } }, "checkin": "16 February, 2021 at 10:00 AM", "checkout": "17 February, 2021 at 10:00 AM", "hours": 24.0 } </pre> <p>In this example, Dr. Bonnells's total of 25 hours <i>includes</i> the 24 hours of work from which he is checking out.</p>
---	-------------	---	---	--	---	---

<p>Find an Attendance by ID (admins only)</p>	<p>GET</p>	<p>"/check {eventId} {userId}"</p> <p>NOTE: the <i>Event's</i> ID comes first</p>	<p>none</p>	<p>URL: localhost:8080/check/8/4</p>	<p>"message": "Find One Success", "data": [the Attendance data]</p>	<pre> { "flag": true, "code": 200, "message": "Find One Success", "data": { "relationId": { "user": { "userId": 4, "email": "ric.bonnell@tcu.edu", "type": "FACULTY", "status": null, "roleId": true, "firstname": "Ric", "lastname": "Bonnell", "gradYear": 0, "totalHours": 25.0, "hoursLeft": 0.0 }, "calendarEvent": { "eventId": 8, "title": "MATH", "category": "TEST1", "sex": null, "description": null, "address": null, "startDateTime": "25 March, 2021 at 4:00 AM", "endDateTime": "25 March, 2021 at 4:00 PM", "nonFacultyCapacity": 4, "facultyCapacity": 5, "gradYear": 0, "comment": null, "host": null, "active": false, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 0, "numFacultySignedUp": 1 } }, "checkIn": "10 February, 2021 at 10:00 AM", "checkOut": "17 February, 2021 at 10:00 AM", "hours": 24.0 } } </pre>
<p>Find Total Hours required of all students</p>	<p>GET</p>	<p>"/users /totalHours"</p>	<p>none</p>	<p>N/A</p>	<p>"Message": "Find Total Required Hours Success", "data": [the decimal number of hours required of each student]</p>	<pre> { "flag": true, "code": 200, "message": "Find Total Required Hours Success", "data": 75.0 } </pre> <p>The value being returned for this request comes from the properties instead of the database-- one can change it by editing application.yml.</p>
<p>Find all Users who are admins</p>	<p>GET</p>	<p>"/users /admins"</p>	<p>none</p>	<p>N/A</p>	<p>"Message": "Find All Admins Success", "data": [list</p>	<p>Similar to below vvv</p>

					of all admin Users; empty list if there are no admins]	
Find all Users (admins only)	GET	"/users"	none	N/A	"message": "Find All Success", "data": [list of all Users; empty list if there are no Users]	<pre> "flag": true, "code": 200, "message": "Find All Success", "data": [{ "userId": 1, "email": "riley.durbin@tcu.edu", "type": "STUDENT", "status": null, "admin": false, "firstname": "Riley", "lastname": "Durbin", "gradYear": 2021, "totalHours": 0.0, "hoursLeft": 75.0 }, { "userId": 2, "email": "jim.croce@tcu.edu", "type": "STUDENT", </pre>
Find info on the current User	GET	"/users /current"	none	N/A	"message": "Find One Success", "data": [User's info]	<pre> "flag": true, "code": 200, "message": "Find One Success", "data": { "userId": 7, "email": "example2@tcu.edu", "type": "STUDENT", "status": null, "admin": false, "firstname": "Maria", "lastname": "Amoros", "gradYear": 2021, "totalHours": 0.0, "hoursLeft": 75.0 } </pre>

Find a User by ID (admins only)	GET	"/users /{userId}"	none	URL: localhost:8080/users/7	"message": "Find One Success", "data": [User's info]	<pre> { "flag": true, "code": 200, "message": "Find One Success", "data": { "userId": 7, "email": "example2@tcu.edu", "type": "STUDENT", "status": null, "admin": false, "firstname": "Maria", "lastname": "Amoros", "gradYear": 2021, "totalHours": 0.0, "hoursLeft": 75.0 } } </pre>
Find a User by email (admins only)	POST	"/users /email"	Just one field "email" with the email	<pre> { "email": "example2@tcu.edu" } </pre>	"message": "Find One Success", "data": [User's info]	<pre> { "flag": true, "code": 200, "message": "Find One Success", "data": { "userId": 7, "email": "example2@tcu.edu", "type": "STUDENT", "status": null, "admin": false, "firstname": "Maria", "lastname": "Amoros", "gradYear": 2021, "totalHours": 0.0, "hoursLeft": 75.0 } } </pre>
Save a new User (admins only)	POST	"/users"	User's info (don't worry about the ID) Old field removed: String "status" "gradYear" is optional (should only be filled in for students). "admin" is optional; false by default.	<pre> { "firstname": "Steve", "lastname": "?", "email": "steve@minecraft.net", "type": "other", "gradYear": 2012, "admin": false, "status": "disabled" } </pre> <p>Default "type": OTHER</p>	"message": "Save Success", "data": [new User's info]	<pre> { "flag": true, "code": 200, "message": "Save Success", "data": { "userId": 8, "email": "steve@minecraft.net", "type": "OTHER", "status": "DISABLED", "admin": false, "firstname": "Steve", "lastname": "?", "gradYear": 2012, "totalHours": 0.0, "hoursLeft": 0.0 } } </pre> <p>(Steve is not a student, so his hoursLeft will</p>

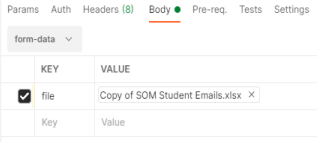
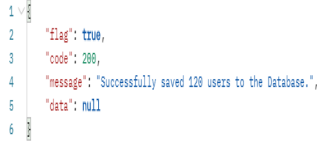
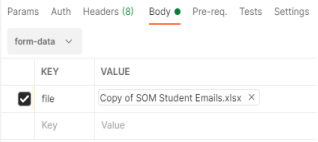

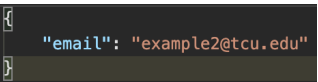
						always be zero.)
Update a User (admins only)	PUT	"/users /{userId}"	Same as above Fields left blank on update will NOT be overwritten. (The exception is "admin" which will default to false if it is left blank. This will always be updated according to the request body.)	URL: localhost:8080/users/8 Body: <pre>"firstname": "Alex", "lastname": "?", "email": "alex@minecraft.net", "status": "approved"</pre>	"message": "Update Success", "data": [updated User's info]	<pre>"flag": true, "code": 200, "message": "Update Success", "data": { "userId": 8, "email": "alex@minecraft.net", "type": "OTHER", "status": "APPROVED", "admin": false, "firstname": "Alex", "lastname": "?", "gradYear": 2012, "totalHours": 0.0, "hoursLeft": 0.0 }</pre>
Delete a User (admins only)	DELETE	"/users /{userId}"	none	URL: localhost:8080/users/3	"message": "Delete Success", "data": null	N/A
Find the current User's Waitlist info, Events signed up for, or past Attendance	GET	"/users /{x}" {x}: w for waitlist, s for signup list, or a for list of attendance	none	URL examples Waitlisted Info: localhost:8080/users/w Events Signed Up For: localhost:8080/users/s Past Attendance: localhost:8080/users/a	"message": "Find [Waitlist/ Signup list/ Attendance] Success", "data": [list of Waitlisted objects/ Events signed up for/ Attendance objects]	Similar to below vvv

<p>Find a User's Waitlist info, Events signed up for, or past Attendance (admins only)</p>	<p>GET</p>	<p>"/users /{x} /{userId}"</p> <p>{x}: w for <i>waitlist</i>, s for <i>signup</i> list, or a for list of <i>attendance</i></p>	<p>none</p>	<p>URL examples</p> <p>Waitlisted Info: localhost:8080/users/w/7</p> <p>Events Signed Up For: localhost:8080/users/s/2</p> <p>Past Attendance: localhost:8080/users/a/4</p>	<p>"message": "Find [Waitlist/ Signup list/ Attendance] Success", "data": [list of Waitlisted objects/ Events signed up for/ Attendance objects]</p> <p>(<i>not</i> just a list of Events, except for signup)</p> <p>Note: these examples -> each show a list with only one item, but it's still a list instead of just the item</p>	<p>Waitlist:</p> <pre> { "flag": true, "code": 200, "message": "Find Waitlist Success", "data": [{ "relationId": { "user": { "userId": 7, "email": "mariavamoros@gmail.com", "type": "STUDENT", "status": null, "admin": true, "firstName": "Maria", "lastName": "Amoros", "gradYear": 2021, "totalHours": 53.0, "hoursLeft": 22.0 } }, "calendarEvent": { "eventId": 7, "title": "Academy4 Como (Apr 16)", "category": null, "sex": null, "description": null, "address": "Zoom only", "startDateTime": "16 April, 2021 at 1", "endDateTime": "16 April, 2021 at 2:3", "nonFacultyCapacity": 3, "facultyCapacity": 4, "gradYear": 2024, "comment": null, "host": { "partnerId": 4, "name": "Mercy Clinic of Fort Wor", "email": null } }, "active": true, "numNonFacultyWaitlisted": 1, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 3, "numFacultySignedUp": 2, "numAttended": 0 }], "timestamp": 1617735088972 } </pre> <p>Signup list:</p>
---	------------	--	-------------	--	---	---

						<pre> "flag": true, "code": 200, "message": "Find Signup list Success", "data": { { "eventId": 7, "title": "Academy4 Como (Apr 16)", "category": null, "sex": null, "description": null, "address": "Zoom only", "startDateTime": "16 April, 2021 at 12:45 PM", "endDateTime": "16 April, 2021 at 2:30 PM", "nonFacultyCapacity": 3, "facultyCapacity": 4, "gradYear": 2024, "comment": null, "host": { "partnerId": 4, "name": "Mercy Clinic of Fort Worth", "email": null }, "active": true, "numNonFacultyWaitlisted": 1, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 3, "numFacultySignedUp": 2, "numAttended": 0 } } </pre> <p>Attendance:</p> <pre> "flag": true, "code": 200, "message": "Find Attendance Success", "data": { { "relationId": { "user": { "userId": 4, "email": "ric.bonnell@tcu.edu", "type": "FACULTY", "status": null, "admin": true, "firstname": "Ric", "lastname": "Bonnell", "gradYear": 0, "totalHours": 24.0, "hoursLeft": 0.0 }, "calendarEvent": { "eventId": 0, "title": "Academy4 Como (May 7)", "category": null, "sex": null, "description": null, "address": "Zoom only", "startDateTime": "07 May, 2021 at 12:45 PM", "endDateTime": "07 May, 2021 at 2:30 PM", "nonFacultyCapacity": 10, "facultyCapacity": 5, "gradYear": 2024, "comment": null, "host": null, "active": true, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 0, "numFacultySignedUp": 0, "numAttended": 1 } }, "checkin": "16 February, 2021 at 10:00 AM", "checkout": "17 February, 2021 at 10:00 AM", "hours": 24.0 } } </pre>
--	--	--	--	--	--	---

Find all Partner Orgs.	GET	"/orgs"	none	N/A	"message": "Find All Success", "data": [list of all Partner Orgs.; empty list if there are none]	<pre> { "flag": true, "code": 200, "message": "Find All Success", "data": [{ "partnerId": 1, "name": "Super Mario Bros", "email": "mario.mario@kingdom.org" }, { "partnerId": 2, "name": "Able Sisters", "email": "mableandsable@crossing.org" }, { "partnerId": 3, "name": "TCU", "email": "blank@tcu.edu" }, { "partnerId": 4, "name": "Aaa", "email": "no" }] } </pre>
Find a Partner Org. by ID	GET	"/orgs /{partnerId}"	none	URL: localhost:8080/orgs/2	"message": "Find One Success", "data": [Partner Org's info]	<pre> { "flag": true, "code": 200, "message": "Find One Success", "data": { "partnerId": 2, "name": "Able Sisters", "email": "mableandsable@crossing.org" } } </pre>
Find a Partner Org. by email	POST	"/orgs /email"	Just one field "email" with the email	<pre> { "email": "blank@tcu.edu" } </pre>	"message": "Find One Success", "data": [Partner Org's info]	<pre> { "flag": true, "code": 200, "message": "Find One Success", "data": { "partnerId": 3, "name": "TCU", "email": "blank@tcu.edu" } } </pre>
Save a new Partner Org. (admins only)	POST	"/orgs"	The organization's info (don't worry about the ID)	<pre> { "name": "New Org", "email": "new.org@neworg.com" } </pre>	"message": "Save Success", "data": [new Org]	<pre> { "flag": true, "code": 200, "message": "Save Success", "data": { "partnerId": 5, "name": "New Org", "email": "new.org@neworg.com" } } </pre>

<p>Update a Partner Org. (admins only)</p>	<p>PUT</p>	<p>"/orgs /{partnerId}"</p>	<p>Same as above</p> <p>Note: for other objects, blank fields are not overwritten on update; but for Partner Orgs, both fields have to be sent each time, even if they are unchanged-- in this case blank fields <i>will</i> be overwritten.</p>	<p>URL: localhost:8080/orgs/3</p> <p>Body:</p> <pre>{ "name": "Texas Chicken University", "email": "blank@tcu.edu" }</pre>	<p>"message": "Update Success", "data": [updated Org]</p>	<pre>{ "flag": true, "code": 200, "message": "Update Success", "data": { "partnerId": 3, "name": "Texas Chicken University", "email": "blank@tcu.edu" } }</pre>
<p>Delete a Partner Org. (admins only)</p>	<p>DELETE</p>	<p>"/orgs /{partnerId}"</p>	<p>none</p>	<p>URL: localhost:8080/orgs/2</p>	<p>"message": "Delete Success", "data": null</p>	<p>N/A</p>
<p>Find Events hosted by a Partner Org. (admins only)</p>	<p>GET</p>	<p>"/orgs /h /{partnerId}"</p>	<p>none</p>	<p>URL: localhost:8080/orgs/h/1</p>	<p>"message": "Find Hosted Events Success", "data": [list of Events; empty list if the org. hosted no Events]</p>	<pre>{ "flag": true, "code": 200, "message": "Find Hosted Events Success", "data": [{ "eventId": 1, "title": "Friday meeting", "category": null, "sex": null, "description": null, "address": null, "startDateTime": "05 March, 2021 at 11:00 AM", "endDateTime": "05 March, 2021 at 12:00 PM", "nonFacultyCapacity": 0, "facultyCapacity": 2, "gradYear": 0, "comment": null, "host": { "partnerId": 1, "name": "Pet Clinic", "email": "example@pets4ever.org" }, "active": false, "numNonFacultyWaitlisted": 0, "numFacultyWaitlisted": 0, "numNonFacultySignedUp": 3, "numFacultySignedUp": 0, "numAttended": 3 }, { "eventId": 2, "title": "Spring break", "category": null, </pre>

Bulk Insertion of Users (admins only)	POST	"/upload"	Excel file to be processed		"message" : "Successfull y saved x users to the Database."	
Bulk Deletion of Users (admins only)	POST	"/bulkDelete"	Excel file to be processed		"message" : "Successfull y deleted x users from the Database."	
Create a Report for a single User (admins only)	GET	"/report /{userId}"	none	URL: [url]/report/1	Contains the Excel file with inline Content Disposition	N/A
Create a Report for a single User via Email (admins only)	POST	"/report/email"	Email passed in the request body as a String email Turns out, it doesn't work to send a GET request with a body... so, POST!		Contains the Excel file with inline Content Disposition	N/A
Create a Report for an Event (admins only)	GET	"/report /event /{eventID}"	none	URL: [url]/report/event/5	See above	N/A

5. Firebase Management

The users are managed in the app using the admin-only “User Management” tab in the navigation bar. Adding/removing a user should only be done using this feature because we need to add them or remove them from both the backend database and Firebase authentication directory. Adding or removing a user from one, but not the other can cause unexpected errors in the application for that user. If for whatever reason you may need to log in to the Firebase Console, Dr. Bonnell will have the login credentials. From there, you can navigate to the “Authentication” dashboard and view the users in the system.

6. Deployment to Heroku

Heroku has a cool feature in that you may link it to a GitHub repository branch and it will detect when changes are made to that branch and automatically deploy them.

- **Automatic deployments:** Handled automatically by Heroku based on the linked GitHub repository.
- **Manual deployments:**
 1. Navigate to volunteer-server-springproject
 2. Deploy
 3. Manual deploy
 4. Deploy Branch

To change the linked GitHub account:

1. Navigate to volunteer-server-springproject
2. Click the “Deploy” tab
3. Disconnect the currently linked account
4. Choose your account and associated branch to automatically deploy

If you want to view the Spring Boot console:

1. Navigate to volunteer-server-springproject
2. “More” tab in the top right-hand corner
3. View logs

Billing information is currently managed by Dr. Bonnell’s account. If billing information needs to be changed, you can do the following:

1. Click your user icon in the top right-hand corner of the screen
2. Click “Account Settings”
3. Click the “Billing” tab

7. Domain name management

All functions relating to the domain can be found under the Manage tab found in the Main Dashboard found at www1.domain.com.

7.1. Purchasing a domain name

Within this site, information regarding:

- Domain lease renewal
- DNS and Name server pointer configuration

can be found throughout the various tabs.

7.1.1. Domain Lease Renewal

The domain at the time of this document will auto-renew on March 28, 2023. If the administrator chooses to move to a different DNS management system (such as an on-premises solution or a different registrar), transfers can be initiated after June 12, 2021 under the Transfers tab in the left hand side menu.

7.2. Configuring DNS and Nameserver settings

The DNS and Name servers tab will display the configuration for the global DNS pointers and the DNS records for the tcusomservice.com domain listing.

Main Requirements:

- The DNS SOA record must match the Name servers listed or the domain will not point towards the records, breaking the connection to the Heroku project (found as NS records).
- The CNAME record is the single point of connection between Heroku DNS and Domain.com DNS. If another service is used in the future (ex. AWS), this record will need to be replaced with the new deployed location.

7.3. Enabling SSL

Heroku DNS currently provides free SSL encryption if using paid dynos, which are currently being used in the deployment configuration. If this changes in the future or the project moves to another service that does not provide SSL, Domain.com offers LetsEncrypt Free SSL which should be enabled to ensure the connection remains secure.

For Heroku:

- Click on volunteer-server-springproject > Settings
- Scroll to the SSL Certificates section
- Click Configure SSL and select Automated Certificate Management

For Domain.com

- Select Manage and Select LetsEncrypt Free SSL and Enforce SSL