**Green Gardens ****

***GoSchool Run5 Group3***

**Idea Overview**

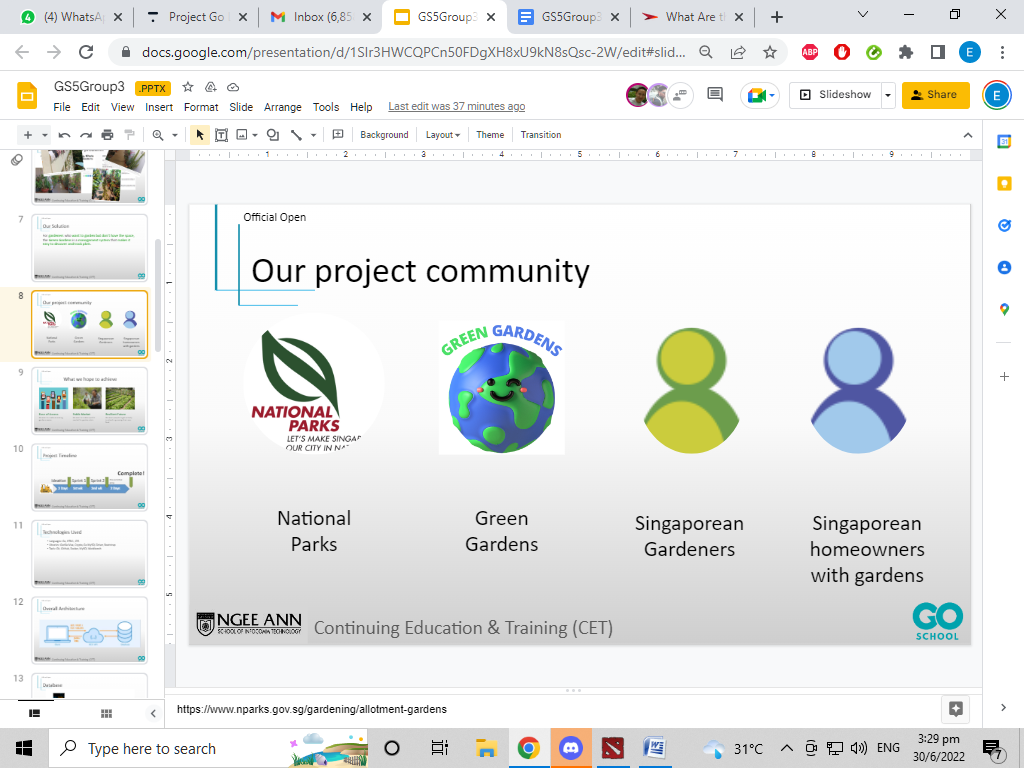
*Problem Brief*

* Singapore has a key target to hit by 2030: 30% of our food consumption should be grown locally.
* While there is a growing demand for farming and gardening, Singapore does not have enough gardening land to meet the supply of budding gardeners.
* The current system for booking plots in community gardens is inefficient and inflexible.

*Project Brief*

* For gardeners who want to garden but don’t have the space, the Green Gardens is a management system that makes it easy to discover and track plots.

*Project Community*

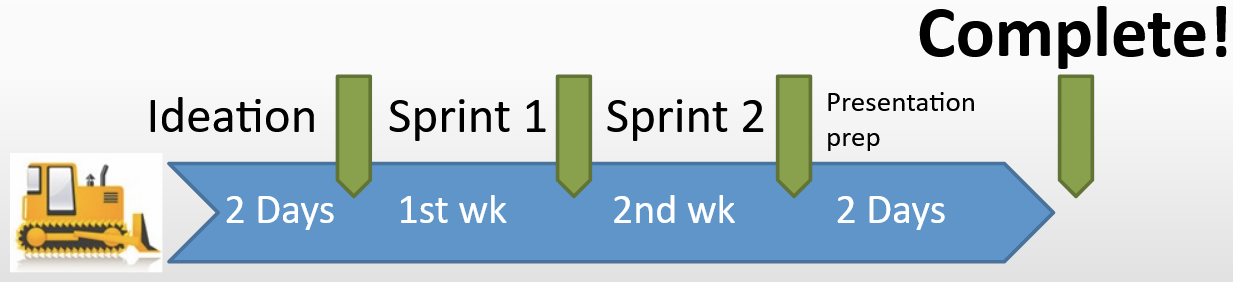
****

*Project Goals*

* Ease of accessing gardens
* Public market for garden plots
* Building a resilient future for Singapore

**Project Overview**

*Project Timeline*

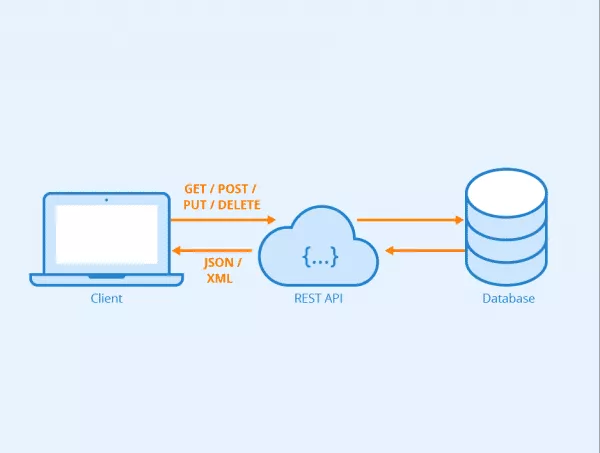
**

* Sprint 1 was focused on setting up our database and API.
* Sprint 2 was focused on getting the client ready.

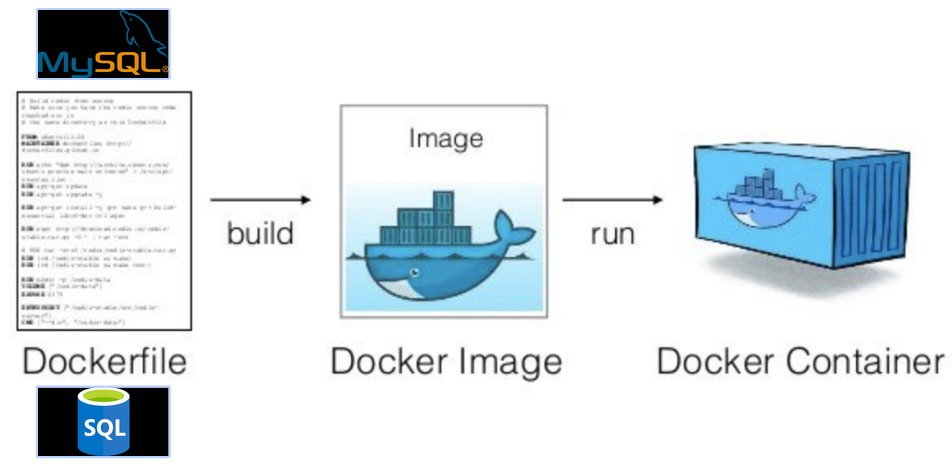
*Technologies Used*

* Languages: Go, HTML, CSS, SQL
* Libraries: Gorilla Mux, Crypto, Go MySQL Driver, Bootstrap
* Tools: Git, GitHub, Docker, MySQL Workbench

*Overall Architecture*

**

**Database**

****

We started with a MySQL Docker Image and used a Dockerfile to copy our SQL Scripts into the image. Once built, the resulting image initialized with all the tables and initial data that we needed for development. Containerizing our database in this way allowed us to have identical databases while developing our project.

**Users Section**

*Routes*

* /signup ,SignUp
* /loginauth, LoginAuth
* /updateresults/, UpdateResults
* /allusers, AllUsers
* /delete, DeleteRecord
* /logout, Logout

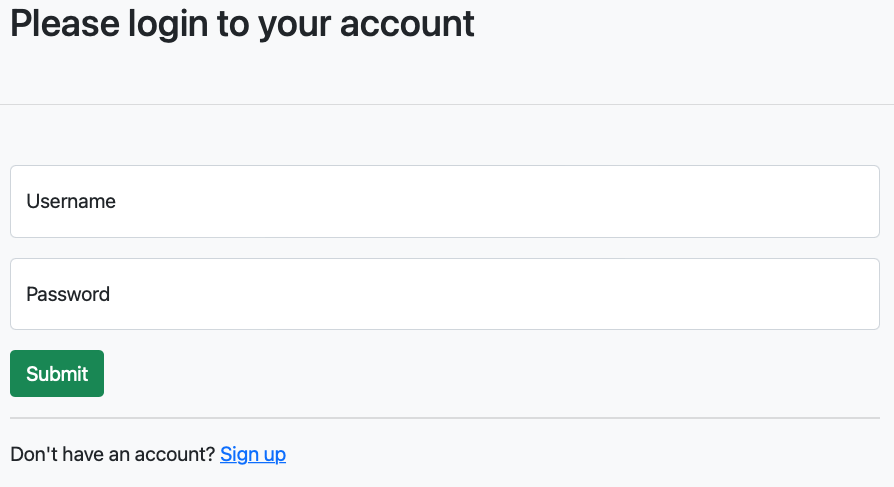
*Handlers*

* SignUp handles either a GET or POST request.
* GET request returns a blank form
* PUT request will request a display name,username, password and email values which will be then stored in the user database.
* LoginAuth handles the authentication of the user details in the user database and logs the user in. Upon successful login, a user session is created using a cookie.
* UpdateResults handles the updating of the users values
* AllUsers handles showing all users in the database.
* DeleteRecord handles deleting a selected user in the database.
* alreadyLoggedIn returns a bool value. It checks for the cookie in the user session and returns true if there is one, allowing the user to have access to the appropriate handler.
* Logout handles the logging out of a user session. It deletes the cookie in the session immediately.

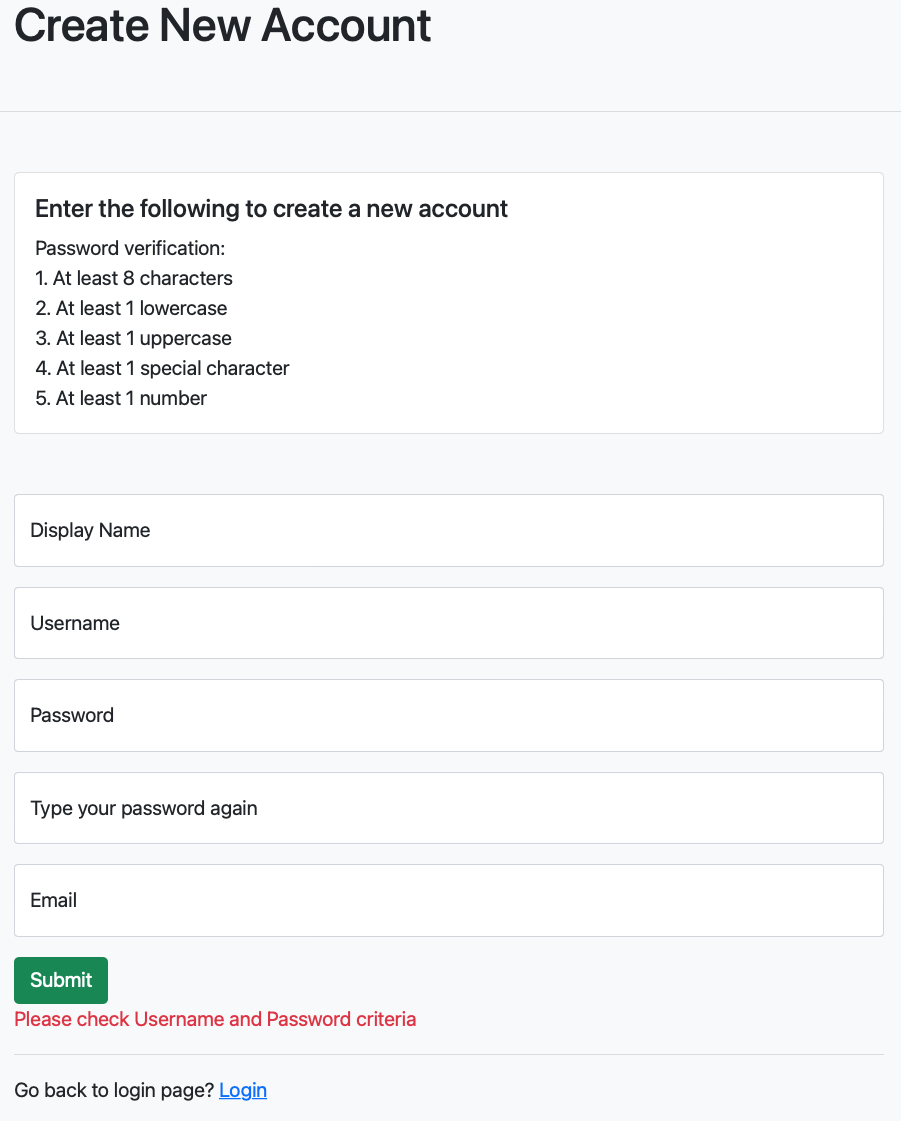
*Security*

* Password has verification criteria to prevent cracking of password
* Password is stored as a hash in our database
* Upon logging in, a cookie is created. Cookies are checked when a request is made, so it prevents users from accessing pages if they are not logged in.

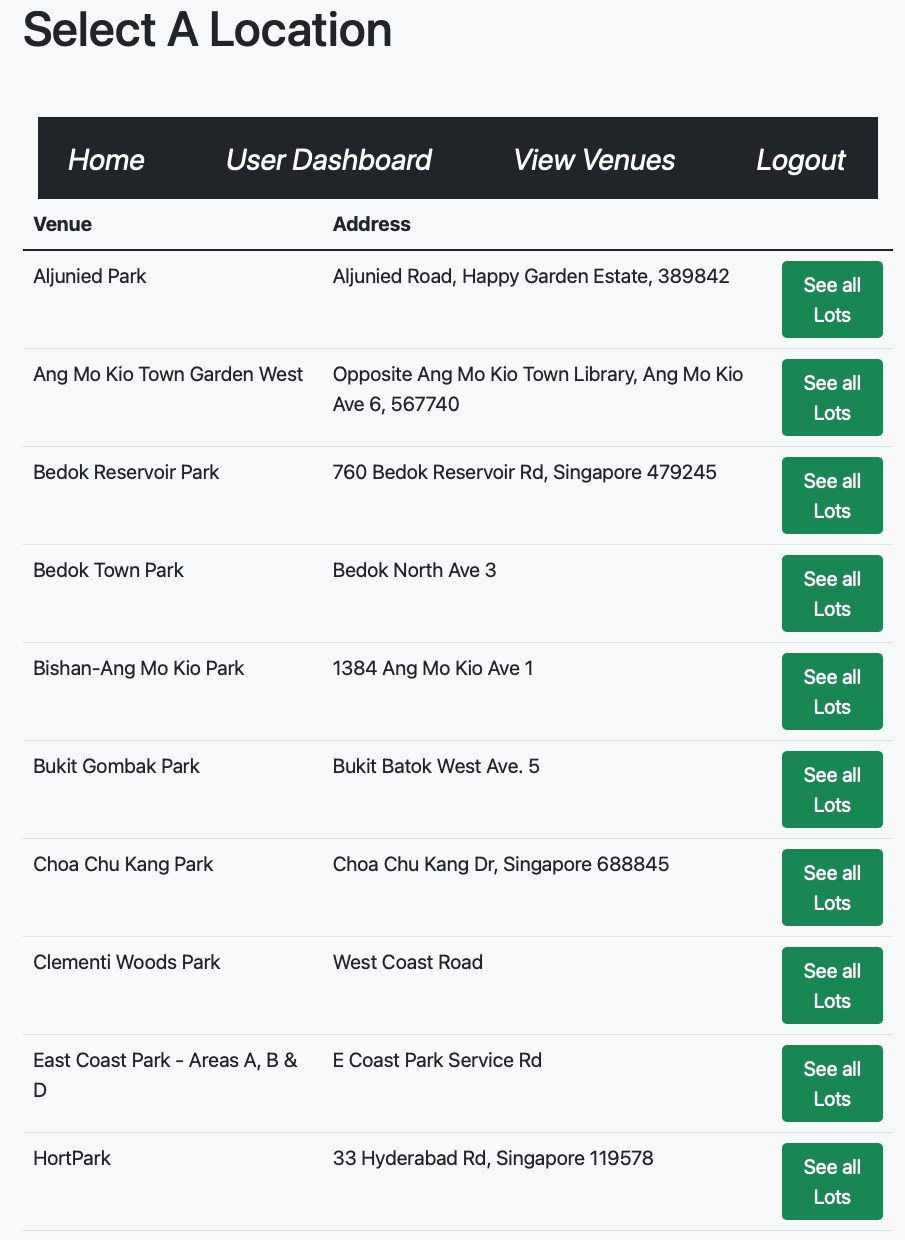
*Client*



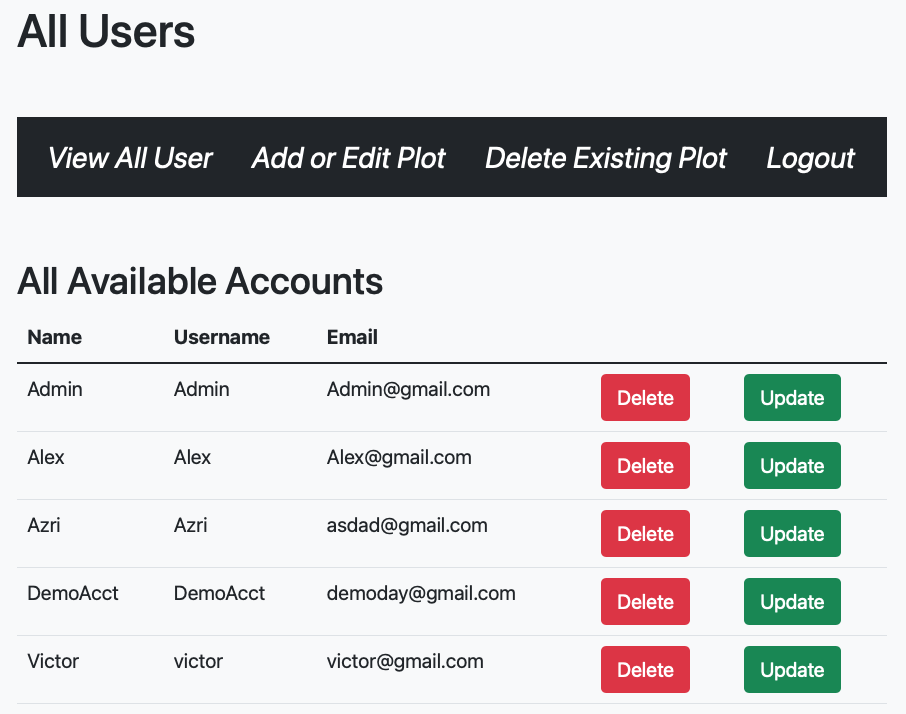
The start of the web application, users will be prompted to either Login if they have an existing account or sign up a new account.



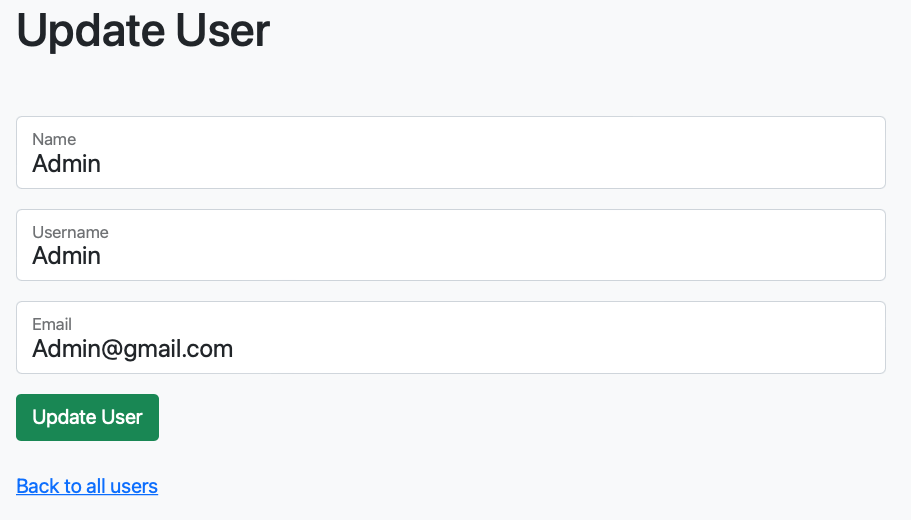
In the SignUp page it displays a form that requires the user to fill. Input verifications are in place so as to allow the appropriate data to be parsed to the database hence allowing a proper creation of a user account. Upon successful creation of the user account, users will be routed back to the login page which they can login with their new accounts.



A successful login will direct the users to the users’ homepage which displays all the available venues sorted by alphabetical order.



Admins accounts are available, these accounts are able to view all users and perform Delete or Update operations.



When updating a profile a form will prompt the users for their name, display name and email address. All these fields are set to required before sending a Put request to the users database.

**Plots Section**

*API*

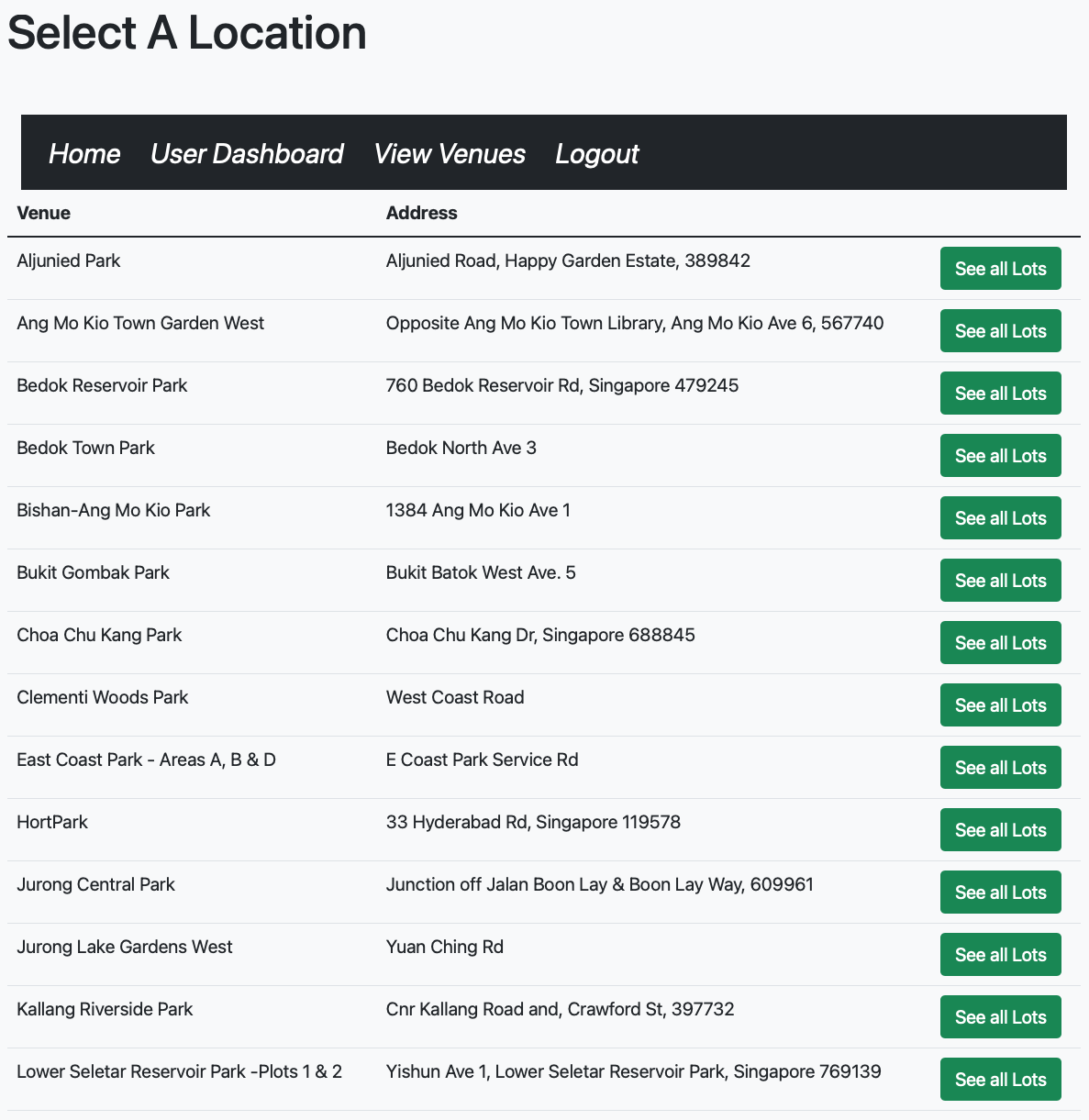
Routes

* /api/v1/plots, GetAllPlots
* /api/v1/plots/{plotId}, PlotHandler
* /api/v1/plots/venue, venueHandler
* /api/v1/plots/venue/{VenueName}, viewVenuePlots

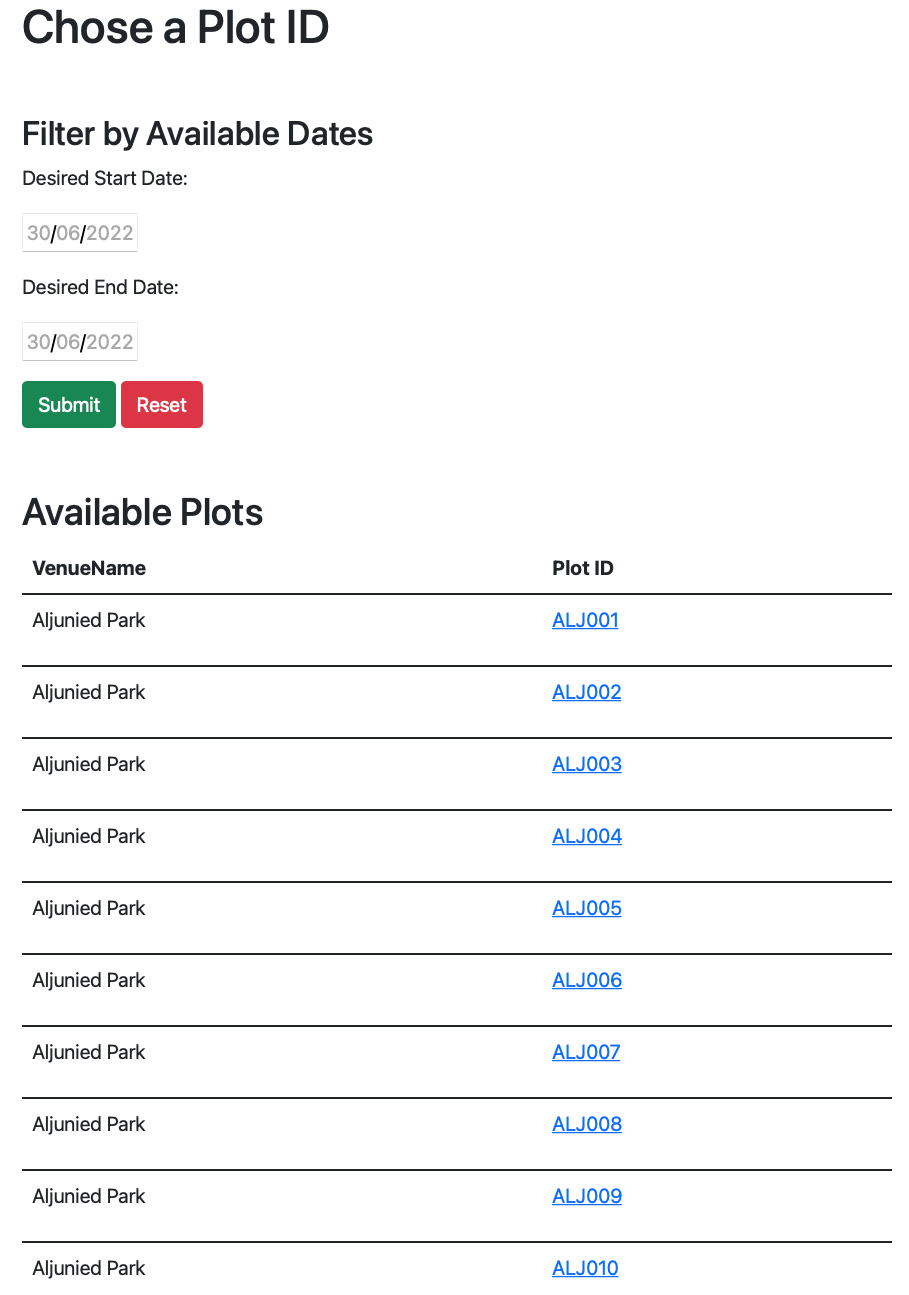
Handlers

* GetAllPlots handles a GET request from the client. It returns all the Plots from the database and stores the information into a Map that is used to be sorted later when needed.
* PlotHandler handles GET, DELETE, POST and PUT for individual PlotIDs:
  + GET returns a json {PlotID:{PlotID, VenueName, Address}} based off what PlotID was given.
  + DELETE will delete the respective PlotID given to the handler if it exists.
  + POST will add a new PlotID, VenueName and Address depending on what plotid was given to it.
  + PUT will either add a new PlotID if it does not exist, or alter the VenueName and Address depending on what the administrator fills it with.
* venueHandler returns json {VenueName:Address} depending on how many venues there are.
* viewVenuePlots will return a json list of PlotIDs based on what VenueName was given to it.

*Client*

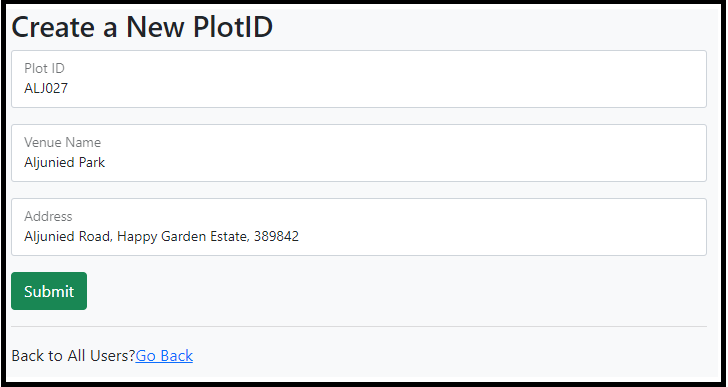


View venues page allows users to look at the available garden spaces located all over Singapore. The venues are sorted by ascending alphabetical order. Users can select the venues of their choice and look at all the available plots within the selected venue.

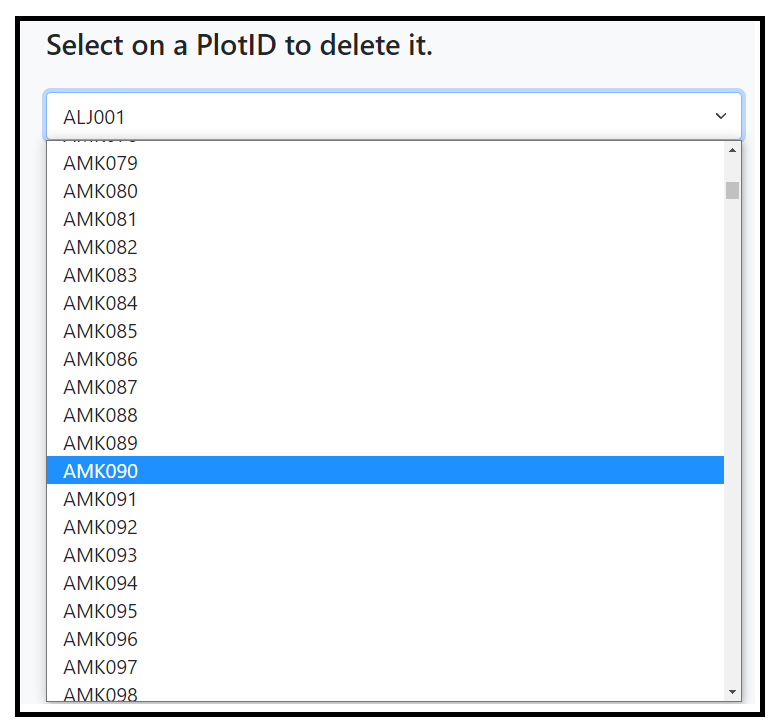


View venue plots enable users to filter plots based on date of availability. Selection of a plot allows users to book it. This lead to the booking page that will be further explained below

*Administrator*



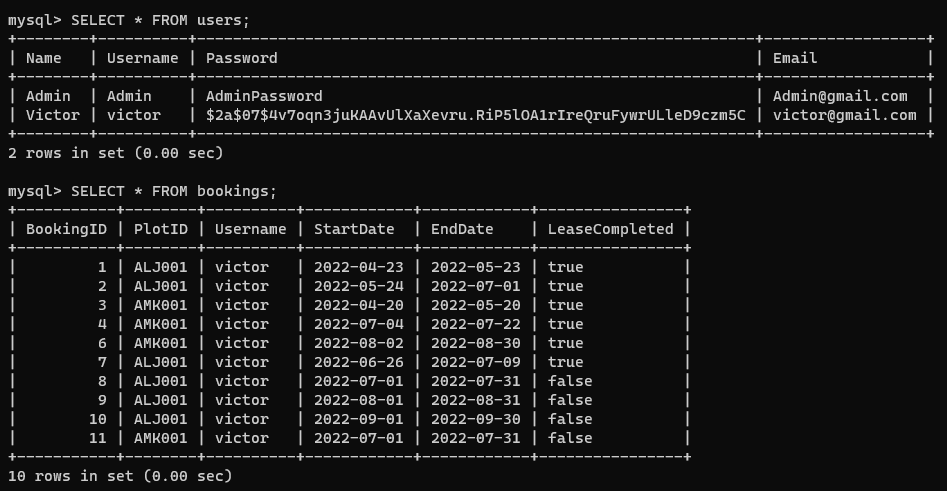
As an administrator, creating a new plot can be done by selecting the “Add or Edit Plot“ tab found in the admin dashboard. The form fields check if the entered PlotId does not exist and append it to the database.



The “Delete Existing Plots” tab in the admin dashboard allows admins to delete existing Plots using PlotID as reference. Upon deletion, The information that is tied to the PlotID will be deleted.

**Bookings Section**

*Database*

****

* *BookingID* (This is a primary key that auto-increments as booking entries are created. It does not need to be specifically entered by the user.)
* *PlotID* (Unique ID for plots)
* *Username* (Unique ID for users)
* *StartDate*
* *EndDate*
* *LeaseCompleted* (Boolean noting whether the lease has been completed. It defaults to *false* when the booking is created and can be changed to *true* by the user when the lease is completed.

*API*

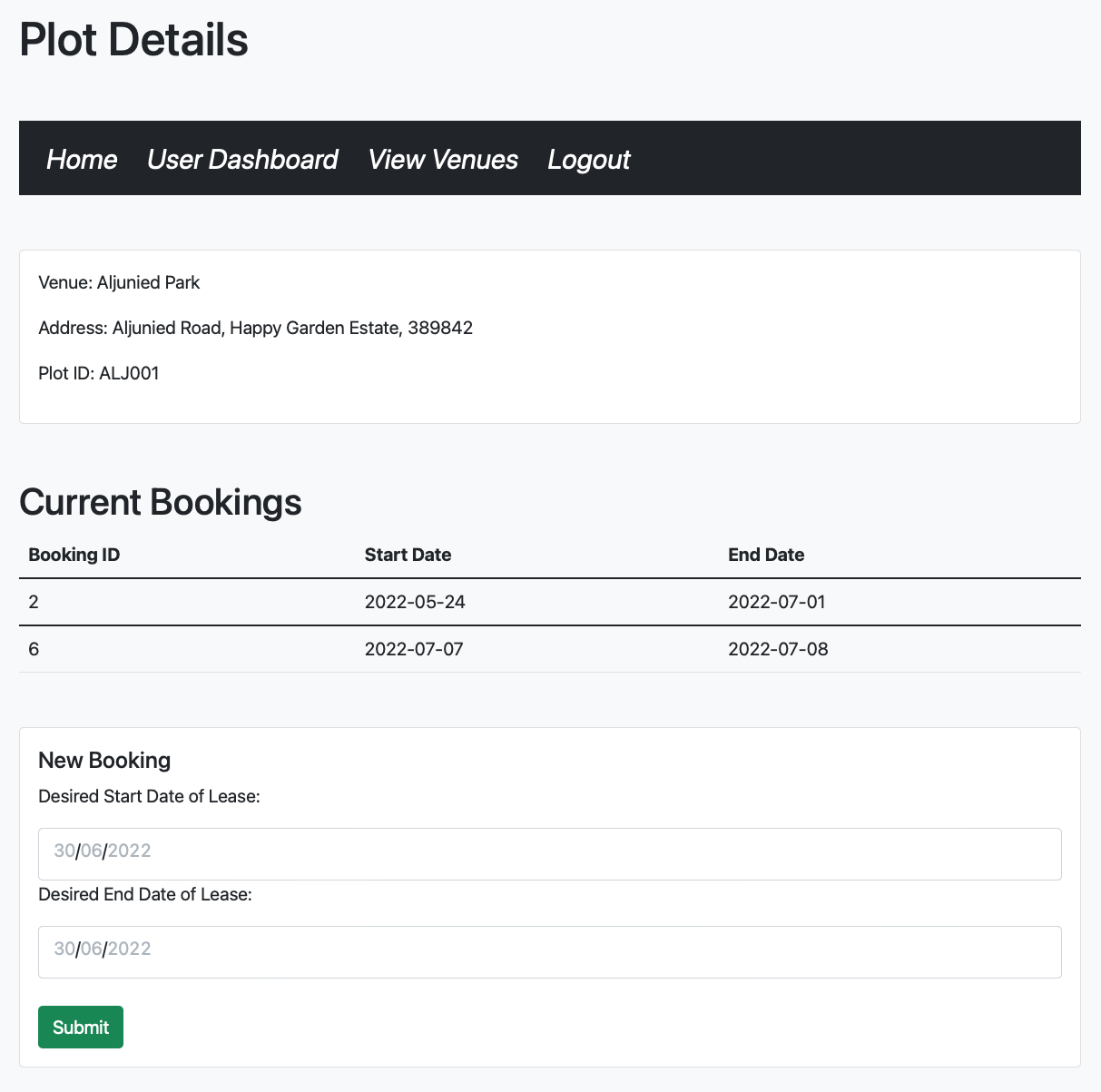
Routes

* /api/v1/bookings, getHandler
* /api/v1/bookings/user/{Username}, getHandler
* /api/v1/bookings/plot/{PlotID}, getHandler
* /api/v1/bookings/booking/{BookingID}, bookingHandler

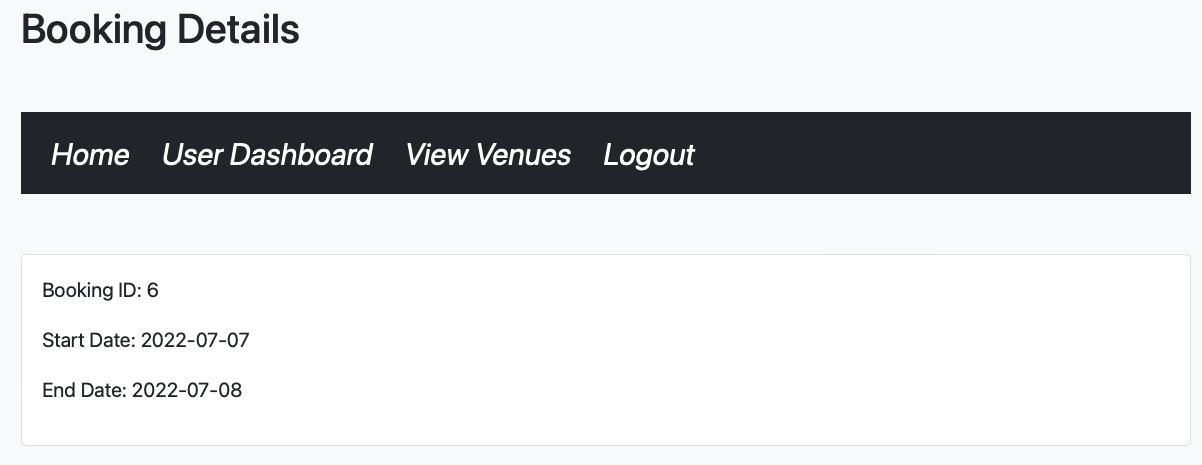
Handlers

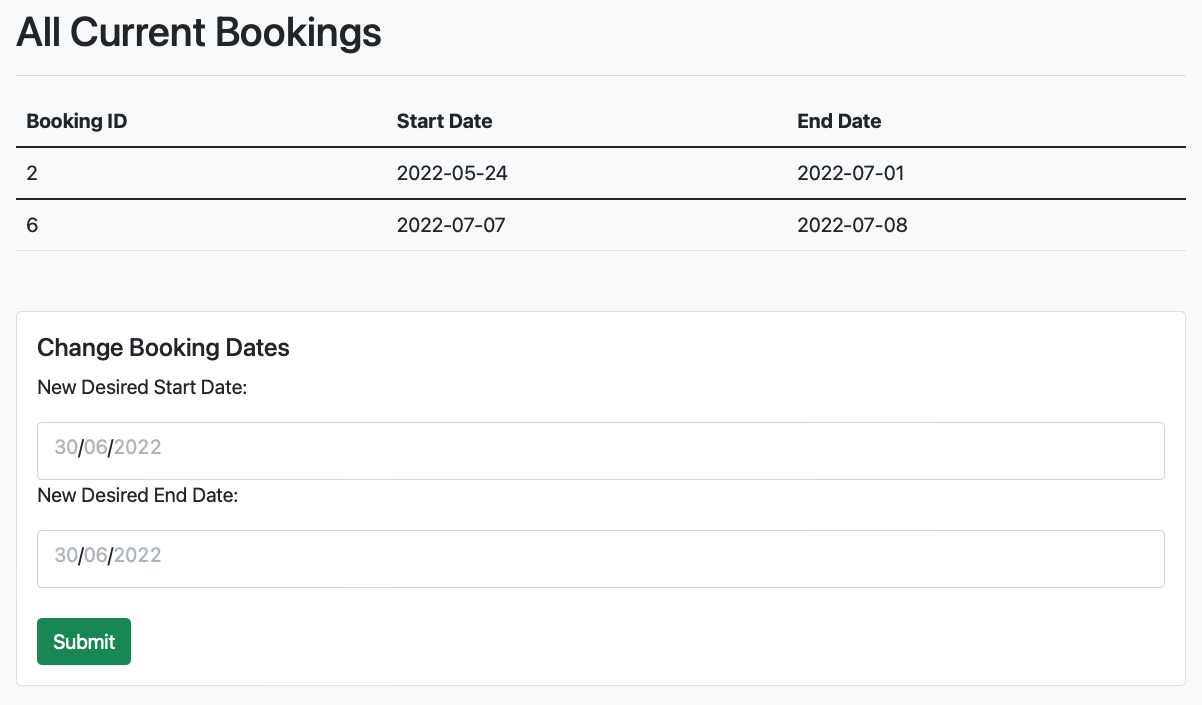
* getHandler handles GET requests for (potentially) multiple bookings. Depending on URL parameters, it will either return all bookings, all bookings for one *Username*, or all bookings for one *PlotID*.
  + GET returns information for bookings by specified parameter. If the specified parameter (*Username* or *PlotID*) is not found, it will return a 404 error.
* bookingHandler handles GET, POST, PUT, PATCH, and DELETE requests for one specific *BookingID*
  + GET returns information for one booking. If the booking is not found, it will return a 404 error.
  + POST creates a new booking. If the desired date range overlaps with another booking on the plot, it will return an error. The *StartDate* entered also has to be before the *EndDate*.
  + PUT modifies an existing booking. If the desired date range overlaps with another booking on the plot, it will return an error. The *StartDate* entered also has to be before the *EndDate*.
  + PATCH marks a booking as completed. Only on-going bookings with *LeaseCompleted* set to *false* can be marked as completed.
  + DELETE will cancel a booking. Only on-going bookings with *LeaseCompleted* set to *false* can be deleted. After a booking is marked as completed, it becomes history and cannot be deleted anymore.

*Client*

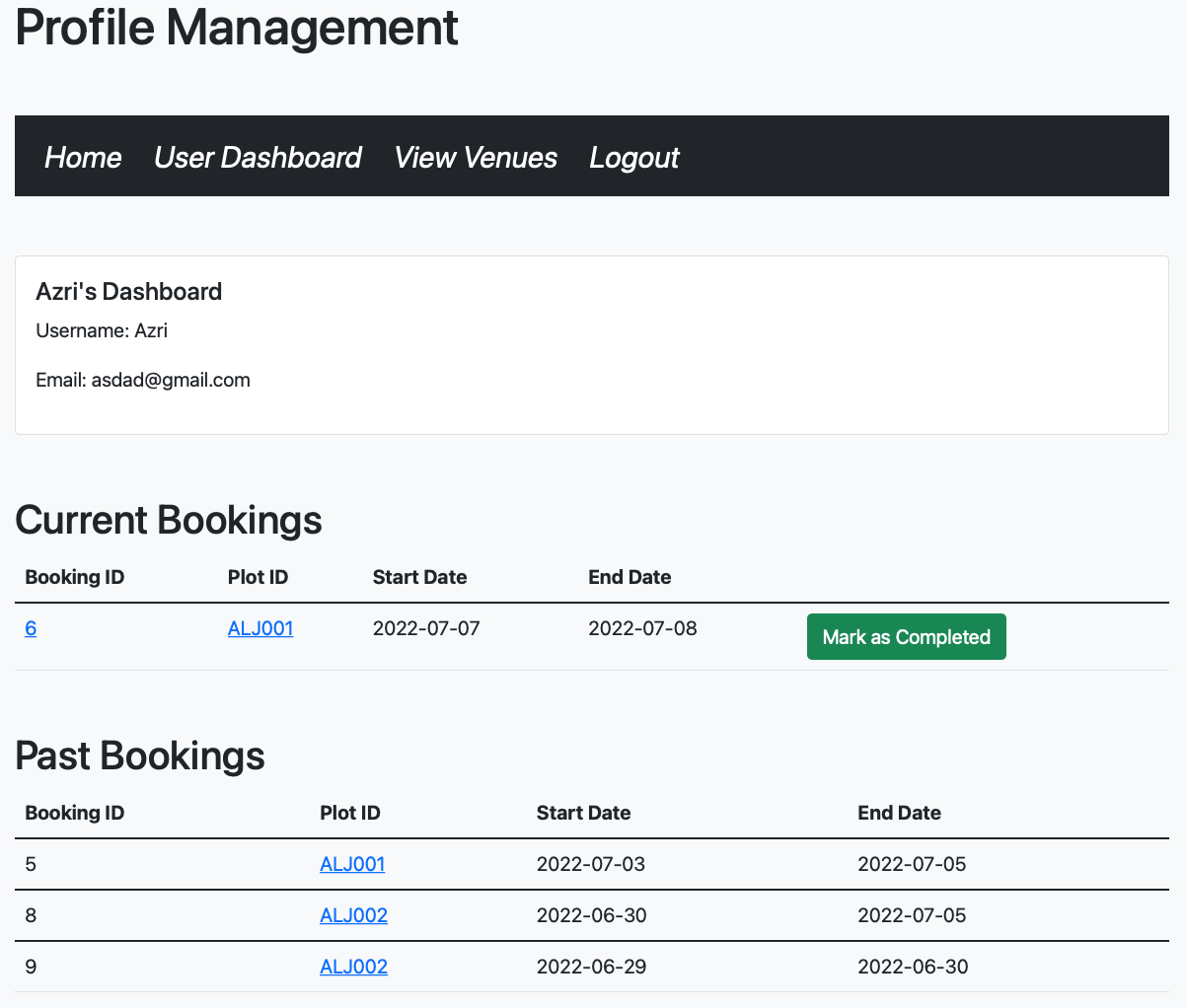
**

New Booking Template allows the user to create a new booking. The template also doubles as a general plot details page that has all of the current bookings on the plot. The template has to pull user, booking, and plot information and does all three concurrently using goroutines.

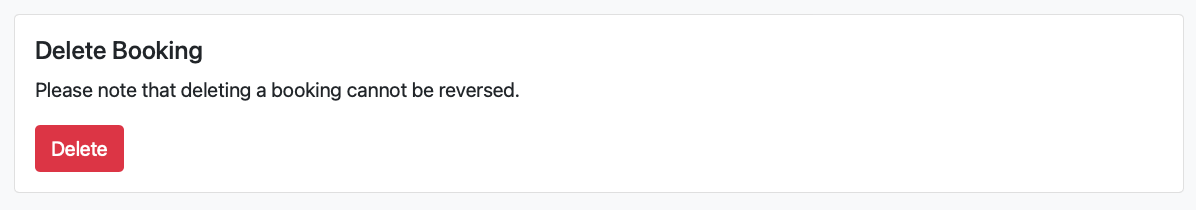




Edit Booking Template allows the user to edit an existing booking. The template also doubles as a general booking details page. User information is pulled concurrently in a goroutine.



User Dashboard Template shows the user information on their account as well as current and past bookings.



DeleteBooking and CompleteBooking functions can be accessed on the Edit Booking and User Dashboard templates, respectively.

**Limitations and Future Improvements**

Limitations of this project

* This is a community driven initiative
* Confirmation of newly added plots needs to be manually done

Future improvements

* Allow users to lease out private plots
* Secure API with an API key for either user/administrative purposes
* Search and filter by Region/ Postal code when viewing venues
* Using HTTPS to secure transfer of data from client side to database
* Using Singpass and myInfo verification to control unique new user traffic
* Deleting a Venue Plot Id will trigger to cancel all existing bookings and notify the affected users

**After MVP**

* Leasing of other space could reach hydroponics, insect farming, aquaculture, aquaponic etc
* Educational content can be included in the client
* Enable produce or gardening materials to be sold on the client to users

**Setup Guidelines**

Database

1. From the database directory, build a customized mysql image by running

$ docker build -t database .

(Please take note of the period at the end of the command.)

1. Run the new image with this command

$ docker run --name database -p 32769:3306 -e MYSQL\_ROOT\_PASSWORD=password -d database

1. You should now be able to access the database on the 32769 port

### 

### Check to see if database container is working

1. $ mysql -P 32769 --protocol=tcp -u root -p
2. Enter password, it should be ‘password’ if you followed the instructions above. If there's an error or you don't know what to do:

$ docker start database

1. You are now in the mysql CLI
2. mysql> use database
3. mysql> select \* from bookings;
4. mysql> select \* from users;
5. mysql> select \* from plots;
6. You should see a table with the preloaded data.

Client and API

1. In the root of the directory:

$ go run .

1. In your browser, go to localhost:8080/signup to begin testing