Slotify

Cynthia Lee Weng Yan (A0190363H)
Liu Yongliang (A0214023B)
Ng Shi Wei (A0185450E)
Tian Fang (A0194568L)

Slotify **1**

Milestone 0 **3**
Problem Statement 3

Milestone 1 **3**
Application Description 3
Mobile Cloud Computing 4

Milestone 2 **5**
Target Users 5
User stories 5
Promotion Plan 5

Milestone 3 **6**

Milestone 4 **8**
Alternative to REST: gRPC 8
Explanation of choosing REST 9

Milestone 5 **10**
API Documentation 10
Conforming to REST Principles 10
Explanation for Deviations 10

Milestone 6 **11**

Milestone 7 **12**
Splash Screen Justification 12

Milestone 8 **12**
Introduction 12
Justification of methodologies 12
Productivity 13
Maintainability 13
Exceptions 13

Milestone 9 **13**

Milestone 10 **14**

Milestone 11 **14**

Comparison between session and token-based authentication 14

Justification for JWT 15

Milestone 12 **15**

Mobile Site Design Principles 16

Keep calls to action front and center 16

Keep menus short and sweet 17

Make site search visible 17

Implement filters to narrow results 18

Use click-to-call buttons for complex tasks 19

Milestone 13 **19**

Common Workflows 19

Create an Event 20

Sign up for an event 20

View event sign-ups and take attendance (for admins) 21

Milestone 14 **21**

Google Analytics 21

Milestone 16 **22**

Google Social Integration 22

Telegram Integration 22

Milestone 0

Problem Statement

Student leaders have a hard time scheduling sessions and events for their members. Currently, many of them use Google sheets, NUSync, or telegram bots to allow registration and scheduling of weekly sessions. However, there are limitations to the existing solutions:

- Unable to easily account for different experience levels of participants.
- Members have to repeatedly input their details into different google sheets.
- Everyone can see each other's personal details, resulting in privacy issues!
- Members might accidentally overwrite/remove another member's slot.
- Google sheets are not user friendly on mobile.
- Admins have a hard time configuring certain advanced options to filter or view participant information easily.
- Admins have to create new google sheets for each new week/session.
- Hard to read the google sheets as each sheet could contain multiple sessions.

Milestone 1

Application Description

Our application aims to help university student leaders to view and organize groups and events as well as manage attendance. Application features are listed as follows:

- Login/register
    - Google login/register
    - Normal login/register using email
- User profile
    - User Name
    - Email
    - Student Number
    - NUSNET ID
    - Telegram Handle (optional)
- Explore/Search/Filter
    - View events
    - View groups
- Create groups
    - Group profile
        - Group name
        - Description
        - Category (e.g. Sports)
        - Group banner (optional)
- Join groups
    - Request to join the group
- Manage groups (Admins of groups)
    - Approve request to join the group
    - Remove members

    - Assign roles to members (Admin or Member)
    - Assign Member tags (No tag, Junior, Senior)
- Create events (Admins of groups)
    - Add event details
        - Title
        - Description
        - Start and End Time
        - Location
        - Available Participant Slots
            - Junior (only members with Junior tag can join)
            - Senior (only members with Senior tag can join)
            - Member (all members can join regardless of tag)
            - Public (anyone can join, including people who are not in the group)
- Sign up for events
    - View all available slots of an event
    - Sign up for a slot if the user is eligible
    - View all events that they have signed up/on the waiting list
- Take attendance (Admins)
    - View all the sign-ups
    - Confirm attendee's attendance

Mobile Cloud Computing

We explain the suitability of building our product as a mobile cloud application by examining the following characteristics of mobile cloud computing:

1. Availability
    a. Our users do not have to download the app to use it. Hence, it's easier for them to switch to another device if they want to. This is especially important for student groups that constantly switch event locations.
    b. Additionally, event attendees do not have to download an app just to view events' details or sign up for an event.
    c. The availability trait is also important to event hosts as they can create and fill in details of events on the computer and just take a phone with him/her to take attendance.
2. Productivity
    a. Easy for participants to sign up for events on the go as signing up is a quick task so having to open up applications like google sheets on a laptop is too cumbersome.
    b. Easy for both admins and participants to view on mobile and mark attendance onsite/during the event where it will not be convenient to use laptops.
3. Device-independent
    a. Web app so that all members can access the required data, regardless of the device they use. This also reduces any discrimination for anybody who might have to use certain scheduling applications that are only available in iOS or Android, in order to book a slot for certain clubs.
4. Low hardware requirements
    a. As Google sheets and other applications may take up much space and memory, it may not be convenient for everybody. In contrast, our application will require minimal system resources and have fast load time.

Milestone 2

Target Users

Our target users are university students who are actively involved in CCAs and student life activities. In particular, we aim to target group leaders (such as CCA executive committee (EXCO) members) who wish to conveniently schedule events and manage sign ups for their members.

User stories

- As a CCA EXCO, I want to create a group to manage all CCA related activities
- As a CCA EXCO, I want to fill in relevant details to customize the group
- As a CCA EXCO, I want to update group details.
- As a CCA EXCO, I want to members to be able to join the group
- As a CCA EXCO, I want to view and approve join requests
- As a CCA EXCO, I want to be able to remove members
- As a CCA EXCO, I want to be able to find existing members easily
- As a CCA EXCO, I want to delegate admin roles
- As a CCA EXCO, I want to group members according to their experience level
- As a CCA EXCO, I want to create members-only events/sessions
- As a CCA EXCO, I want to create public events for outreach
- As a CCA EXCO, I want to see all on-going events/sessions
- As a CCA EXCO, I want to easily view the latest event/session
- As a CCA EXCO, I want to be able to reserve slots for members of a particular experience level
- As a CCA EXCO, I want my CCA members to be in a waiting list if there are no slots available
- As a CCA EXCO, I want CCA members in the waiting list to be allocated the slot if another member withdraws
- As a CCA EXCO, I want to have a link to share an event to
- As a CCA EXCO, I want to see the details of the members who signed up for the event
- As a CCA EXCO, I want to be able to take attendance of participants of an event
- As a CCA Member, I want to be able to join the CCA group I am in
- As a CCA Member, I want to see the upcoming CCA events
- As a CCA Member, I want to sign up for applicable events
- As a CCA Member, I want to be able to withdraw from an event
- As a student, I want to find ongoing school events I am interested in
- As a student, I want to sign up for public events without joining the group

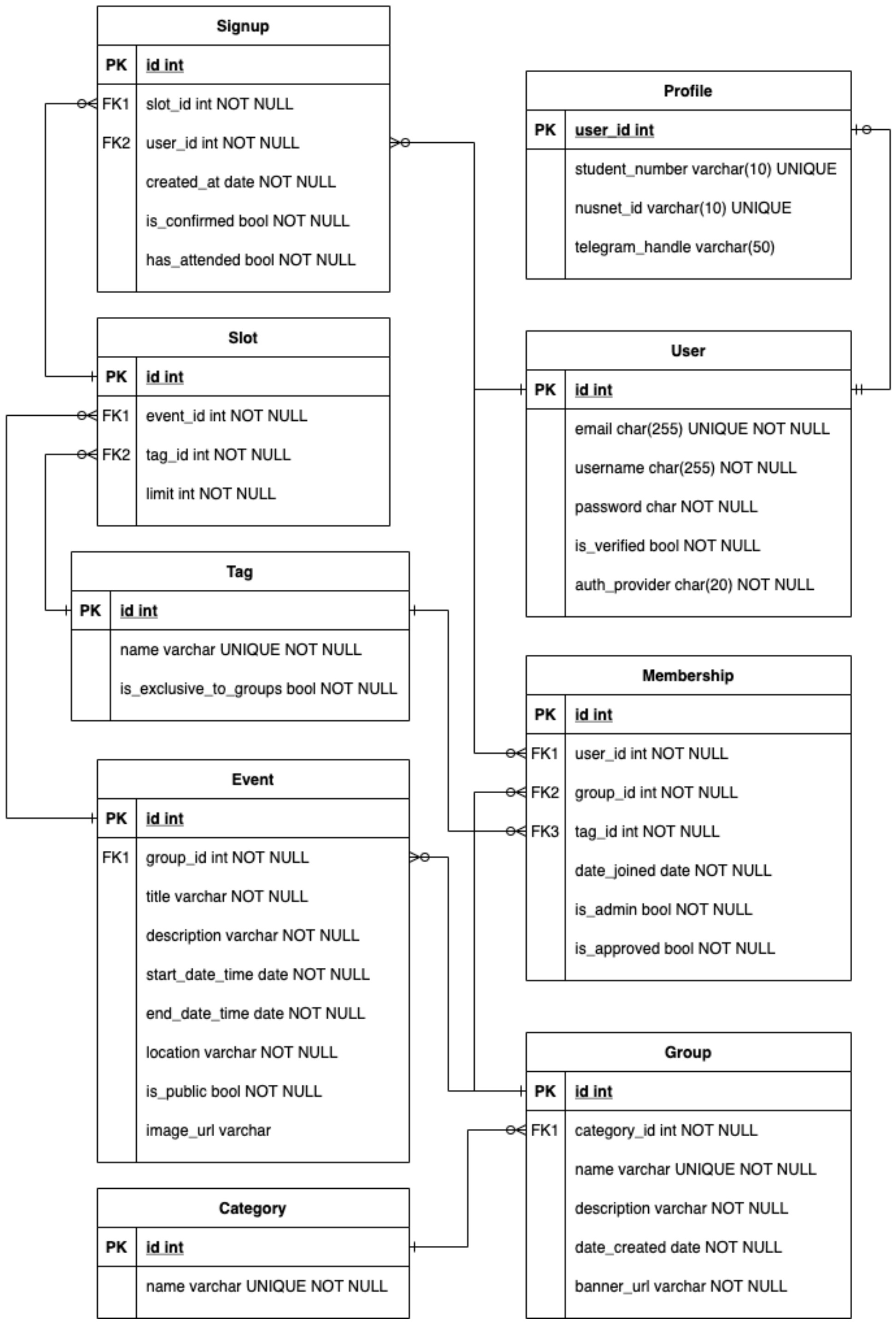
Promotion Plan

- Booth and promotion campaign at the Student Life Fair (at the start of the semester).
    - As many CCAs and their members will be promoting their respective groups during this time, it would be a good chance to raise publicity and attract both group leaders and members.

- Posters around faculties/CCA facilities, Utown, UCC, USC, Utown Gym, MPSH and other commonly used sports facilities.
    - These facilities are regularly used by CCA members and have high traffic since they are shared by all students.
- Welfare pack flyer.
- Video demonstration of the key features of the application, which can be shared to target users via multiple channels.
    - University students have limited time and short attention spans. A succinct video will help students to quickly and easily understand the app’s functionalities.
- Emailing the CCA Excos/Promote to CCA leaders.
- Social media postings/Instagram ads.
- Publicity on Varsity Telegram channels such as https://t.me/nus_sg.

Milestone 3

Entity Relationship Diagram for the application


Signup
PK id int
FK1 slot_id int NOT NULL
FK2 user_id int NOT NULL
created_at date NOT NULL
is_confirmed bool NOT NULL
has_attended bool NOT NULL
Profile
PK user_id int
student_number varchar(10) UNIQUE
nusnet_id varchar(10) UNIQUE
telegram_handle varchar(50)
Slot
PK id int
FK1 event_id int NOT NULL
FK2 tag_id int NOT NULL
limit int NOT NULL
User
PK id int
email char(255) UNIQUE NOT NULL
username char(255) NOT NULL
password char NOT NULL
is_verified bool NOT NULL
auth_provider char(20) NOT NULL
Tag
PK id int
name varchar UNIQUE NOT NULL
is_exclusive_to_groups bool NOT NULL
Membership
PK id int
FK1 user_id int NOT NULL
FK2 group_id int NOT NULL
FK3 tag_id int NOT NULL
date_joined date NOT NULL
is_admin bool NOT NULL
is_approved bool NOT NULL
Event
PK id int
FK1 group_id int NOT NULL
title varchar NOT NULL
description varchar NOT NULL
start_date_time date NOT NULL
end_date_time date NOT NULL
location varchar NOT NULL
is_public bool NOT NULL
image_url varchar
Group
PK id int
FK1 category_id int NOT NULL
name varchar UNIQUE NOT NULL
description varchar NOT NULL
date_created date NOT NULL
banner_url varchar NOT NULL
Category
PK id int
name varchar UNIQUE NOT NULL

The following table contains further elaboration on the purpose of each Entity in our application. Explanations for specific notable fields of each entity are also provided for clarification.

Entity	**Purpose**	**Notable Fields**
User	Stores the key authentication fields of the users.	is_verified: users need to verify their emails before being authorized to use the app. This is because notifications are sent mostly through email.
Profile	Stores information required for users to sign up for events. Each user only has 1 profile (one-to-one relationship with User table).	
Group	Stores the fields describing a group.	
Category	Stores the information on available group categories. Categories are currently managed by application administrators.	
Membership	Represents a user's relationship with a specific group.	is_approved: Indicates if a group admin has approved of the user joining the group. A user is only considered a group member if this is true. is_admin: Indicates if a user is a group admin.
Event	Represents an event organised by a group.	
Tag	Can be assigned to a member in a group, or to a slot, for permission purposes. The Tags are currently managed by the application administrators. There are 4 main tags: **Public, Member, Senior** and **Junior**, with more possibly being added in the future.	is_exclusive_to_group: Indicates if this tag is exclusive to members of a group. This is false for the **Public** tag. This is used to allow non-group members to sign up for a public slot.
Slot	Represents a slot that a user can sign up for. Each slot is assigned a Tag, which determines who can sign up for an event slot. **Public:** All users regardless of membership, including non-group members.	limit: Indicates the maximum number of people that are guaranteed a place for the slot. Beyond the limit, other signups will be waitlisted.

	Member: Only group members, regardless of membership tag. **Junior/Senior:** Only members with the specific, respective tag.	
Signup	Represents a user signup for a particular slot.	is_confirmed: Indicates if a signup is confirmed, or if the user is waitlisted.

Milestone 4

Alternative to REST: gRPC

gRPC is an open-source Remote Procedure Call (RPC) framework developed by Google. It uses language-agnostic protocol buffers (also known as protobuf) to define an API contract in the form of messages and services, representing the expected request and response formats between the server and client. gRPC also provides built-in, out-of-the-box client/server code generation with its protobuf compiler, which allows for easy and convenient use across different languages and frameworks. After generating the required client/server code, the developer would simply have to implement the desired logic, and the framework will handle the data serialization and communication between client and server.

The following table highlights the comparisons between REST and gRPC:

	REST	**gRPC**
HTTP Protocol that it is built on	HTTP/1.1: supports request-response communication. Unable to handle multiple requests at once, thus restricting it to support a single request and response at any one time.	HTTP/2: supports client-response communication. Able to support receiving multiple requests simultaneously - thus able to support different types of streaming (where one entity sends a stream of multiple messages to the other): client, server or bidirectional, on top of unary requests and responses.
Payload data structure	Text-based data formats such as JSON and XML.	Protocol buffers are used to serialize and deserialize payload data into binary packages.
	Human readable and allows for higher flexibility in structure, which allows for dynamic data structures.	Strongly-typed and allows for strict type validation, providing better type-safety.
	Data requires serialization, which would increase transmission time and	The lightweight data format allows for faster transmission - resulting in lower

	potentially hurt performance.	latency.
Browser support	Fully supported by all browsers, and follows the standard HTTP protocol.	Poor browser support, requires gRPC-web and a proxy layer to interface between HTTP/1.1 and HTTP/2. Not as suitable for client-facing applications, but have high applicability for internal systems (eg. an ecosystem of microservices communicating with each other).

Given these considerations, we believe that gPRC would be an optimal choice in the following contexts:

- Microservice-architecture systems where there are multiple individual microservices (potentially implemented using different languages or frameworks) which need to communicate with each other.
- Applications which require low latency, high throughput inter-server communications - such as applications with real-time requirements.
- For communication between internal applications and services which are non-client facing.

Explanation of choosing REST

RESTful API design has been the standard for general API development. We choose REST over other alternatives based on practical considerations as follows:

Application requirements

- For our application, we do not expect high traffic, and there are no real-time requirements. In general, the volume of data retrieved is not high, and there is no need for bi-directional streaming or communication between client and server. As such, we do not find the additional functionalities and performance benefits that gRPC offers to be sufficient incentive for us to adopt it over traditional REST.
- Our application is a PWA with a simple architecture: a monolithic Django backend application, and a frontend application which requires support on browsers. REST is universally supported by default on all browsers. On the other hand, gRPC will require the implementation of an adapter layer to be supported on client-facing frontend applications. This would incur additional development time and delay us from achieving our MVP.

Familiarity with REST

- As most of the web development and APIs out there involve REST, our team is most familiar with REST. This is both from the perspective of an API consumer and an API developer.
- As gRPC is relatively new and young compared to REST, it still has limited developer and community support, and the amount of information regarding troubleshooting and best practices is still comparatively limited.

Ease of setup in our backend server of choice

- We decided to build our backend in Django, which has an excellent toolkit (Django Rest Framework) for us to build Web APIs fast. This choice also means that going with REST allows us to build on top of the existing API View and Serializer classes provided by the framework.

Milestone 5

API Documentation

The API documentation has been published here.

Conforming to REST Principles

- Resources are named according to nouns (eg. events, groups)
- To retrieve/update/delete individual resources, we specify the identifier in the url (ie. `groups/1`), and send the request with the corresponding method - `GET, PUT/PATCH` or `DELETE`.
- Versioning of API is used to ensure that future changes can be done according to semantic version and prevent disruption to existing clients. We added `api/v1/` at the start of the API path.
- To allow ease of browsing, we included filtering and pagination for some of our API paths that could potentially return a large list of items.
    - Filter by category
        - `GET /api/v1/groups/?category=1`
    - Pagination via concept of limit & offset
        - `GET /api/v1/groups/?limit=1&offset=1`
- We try to design our APIs in a way that would clearly show the relationships between the resource entities:
    - To retrieve all slots for an event
        - `GET /api/v1/events/<event_id>/slots`
    - To get members of a group
        - `GET /api/v1/groups/members/`

Explanation for Deviations

For creation of certain objects, we decided to use a separate API endpoint that ends with `/new` and handles POST requests. Examples are as follows:

- `POST /api/v1/groups/new`
- `POST /api/v1/groups/<int:group_id>/events/new`

The reasoning is as follows (using the example of group creation endpoints):

- We reserved `/api/v1/groups` for listing of all the groups. We decided against using `/api/v1/groups` for creation because the creation logic requires upload of a group banner image. This means we have the frontend send over the group creation data, together with the image, in a multipart-form. This however, is different from the JSON format that we use for the group listing endpoint mentioned above. Thus to avoid confusion, we decided to use `api/v1/groups/new` for group creation.

- Following from the above details, the backend logic for handling listing of groups and creation is quite different. In gist, it involves two different serializers and two different views. For better separation of concerns, we decided to have separate classes to handle creation and listing. Thus, the compromise is to have a separate API endpoint.

Milestone 6

Purpose	**SQL Query**

Fetch all the events that a user has signed up for.	`SELECT * FROM events_event`
	`WHERE id IN (`
	`      SELECT event_id FROM events_slot`
	`      WHERE id IN (`
	`            SELECT slot_id FROM events_signup`
	`            WHERE user_id = user_id`
	`      )`
	`)`

List members of a group and their membership details (whether they are admins, their specific tags in the group)	`SELECT`
	`      username,`
	`      email,`
	`      student_number,`
	`      telegram_handle,`
	`      is_admin,`
	`      is_approved,`
	`      tag."name" AS "tag_name"`
	`FROM`
	`      authentication_user AS user_table`
	`      INNER JOIN authentication_profile AS profile ON user_table.id = profile.user_id`
	`      INNER JOIN groups_membership AS membership ON membership.user_id = user_table.id`
	`      LEFT JOIN groups_tag AS tag ON tag_id = membership.tag_id`
	`WHERE`
	`      membership.group_id = group_id;`
	
	Note: `LEFT JOIN` is used because of the possibility that a user's tag in a particular group is `NULL`.

(For admins) Get user signup details for a particular slot for an event, sorted by time of sign up	`SELECT`
	`      username,`
	`      email,`
	`      student_number,`
	`      telegram_handle,`
	`      signup.has_attended,`

	`signup.created_at`
	`FROM`
	`authentication_user AS user_table`
	`INNER JOIN authentication_profile AS profile ON user_table.id = profile.user_id`
	`INNER JOIN events_signup AS signup ON signup.user_id = user_table.id`
	`WHERE`
	`signup.slot_id = slot_id`
	`ORDER BY`
	`signup.created_at`

Milestone 7

Splash Screen Justification

We decided not to use a splash screen to allow users to have a quick launch of the application. In the case where the user might want to quickly view the information required such as the time and location of an upcoming event, not having a splash screen will be a smoother experience for the user.

Milestone 8

Introduction

Good CSS is not easy to write.
Our user interface is designed in Figma, implemented via heavy usage of pre-designed (and good looking) Ionic UI components, with style adjustments made possible by Tailwind CSS utilities.

Justification of methodologies

As mentioned, our application relies on UI components that are either pre-designed or tweaked via CSS utility classes. The combination of the two (Ionic UI Components + Tailwind CSS, in our case) makes for extremely fast and effective styling. As our UI is made up of components, the CSS that we wrote are co-located within each component. Using utility-first CSS libraries can be somewhat counter intuitive and many developers are critical about it's eventual maintainability. However, we made this decision to use the utility first CSS approach because we believe the productivity and maintainability achieved far outweigh traditional CSS architectures. The following justifications will refer to Tailwind CSS but it applies to the other libraries that adopt the same methodology. (For other professional and developer opinions out there, please feel free to read more about utility first approach at https://johnpolacek.github.io/the-case-for-atomic-css/)

Productivity

1. The “batteries included” feature of Tailwind CSS means we need not spend time redefining certain standard CSS properties to be used in the project. For example, we will be able to use a set of sizing classes that help to achieve a consistent font size with utilities such as `text-sm` , `text-md`, `text-lg`. Given that these classes are also well documented, we technically have Tailwind CSS’s documentation as our guide to develop a consistent look and feel with these predefined utilities.
2. The developer toolings for such approaches are fantastic. The intellisense available via IDE plugins can help developers refer to the CSS utilities that they want to use and immediately view the corresponding underlying CSS code. This means less reconstruction of CSS code to achieve the same element style.

Maintainability

1. The idea of co-locating CSS with the javascript/html code can be intimidating. However, the shift towards such approaches (commonly known as CSS-in-JS etc) is a great step towards improving the overall toolchain in the front-end. With proper scoping rules, the components will be styled at the right place and it will be obvious what is being applied in a glance.
2. On top of using pre-existing utilities, there are also ways to compose and construct new utilities. Thus, there are many ways to further consolidate CSS styles into succinct utilities that are unique to the application. In our case, we defined a set of extensions to existing CSS utilities in `tailwind.config.js`.

Exceptions

Other than Tailwind CSS methodology, there are also some global CSS changes made for Ionic components in `index.css`. As Ionic components contain shadow DOM, Tailwind CSS styles do not work well for some cases, so the intended style was defined in the global `index.css` for application wide styling. With this work-around, we were able to define a set of consistent styles to be used across our application.

Milestone 9

Our application is separated into two components:

- The React front-end is hosted on the Vercel platform and connected to our canonical domain at slotify.club. HTTPS is enabled by default.
- The Django Rest back-end is hosted on a DigitalOcean droplet and connected to our API endpoint at api.slotify.club. The application is served with Gunicorn and has Nginx in front to pass respective traffic to the correct processes. To enable HTTPS, we utilized the free Let’s Encrypt service to obtain a free TLS/SSL certificate. We automated the configuration of Nginx to use encrypted HTTPS with the help of Certbot.We also implemented a Permanent Server-Side 301 Redirects of all variants to HTTPS such that Nginx will always route HTTP to HTTPS.

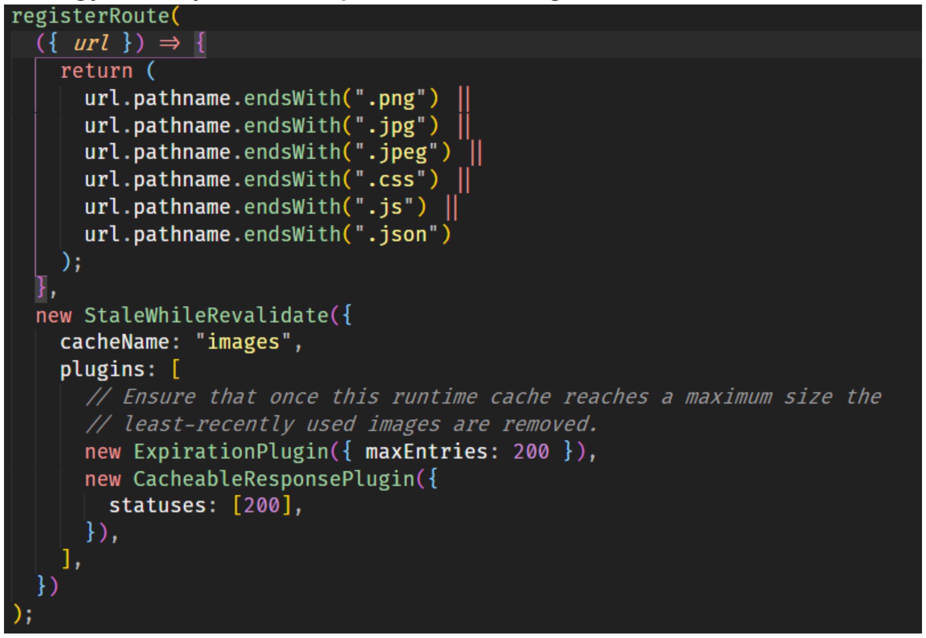
The 3 best practices for adopting HTTPS for our application:

- Automated configuration

- Redirection of HTTP to HTTPS
- Utilize secure defaults provided by the platforms

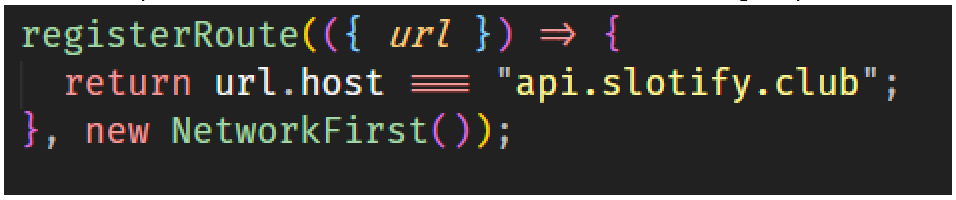
Milestone 10

Static assets such as css files, images, json files are cached using the StaleWhileRevalidate strategy as they do not require much changes.



```
registerRoute(
  ({ url }) => {
    return (
      url.pathname.endsWith(".png") ||
      url.pathname.endsWith(".jpg") ||
      url.pathname.endsWith(".jpeg") ||
      url.pathname.endsWith(".css") ||
      url.pathname.endsWith(".js") ||
      url.pathname.endsWith(".json")
    );
  },
  new StaleWhileRevalidate({
    cacheName: "images",
    plugins: [
      // Ensure that once this runtime cache reaches a maximum size the
      // least-recently used images are removed.
      new ExpirationPlugin({ maxEntries: 200 }),
      new CacheableResponsePlugin({
        statuses: [200],
      }),
    ],
  })
);
```

Pages that the user visits will be cached with a NetworkFirst strategy. This is to ensure the latest data fetched from the server is displayed. However, if the user is offline, previously cached data will still be presented to allow the user to view event and group details.



```
registerRoute(({ url }) => {
  return url.host === "api.slotify.club";
}, new NetworkFirst());
```

Milestone 11

Comparison between session and token-based authentication

The main difference between session-based authentication and token-based authentication is the locations where the user's state is stored. In session-based authentication, after the user is authenticated, the user receives a cookie containing the session id, and attaches this to subsequent requests. The session data is stored in the server's memory, and the server will have to deserialize the cookies it receives and make a database lookup to check against the stored data each time a request is made. When a user logs out, we will also need to make another request to the server to delete the stored session data. Due to this dependency on the server's memory, it is more difficult to scale if many users are using the app at the same time.

On the other hand, the token-based approach stores JWT on the client's computer, and no data is persisted on the server-side. Without this dependency on the server, the token-based approach scales better in comparison to the session-based approach. When a user logs out, we can simply delete a token on the client side with no further action from the server required. Although this could be insecure in higher-risk applications where attackers may try to obtain access tokens, it can be overcome with more advanced techniques such as blacklisting tokens. The generation and verification of tokens are also decoupled, thus allowing tokens to be generated across multiple servers.

The token itself contains all the information that may be required for authentication, and allows more fine-grained and customized access control if we choose to store more information (such as user roles). However, this also results in a generally larger token payload size compared with session-based authentication.

Justification for JWT

We decided to use JWT for a few reasons:

- We utilize the JWT token to keep track of the user who is sending the request. By decoding the JWT token in the server (without accessing the database), we can easily retrieve the identity of the requester. Since most of our API endpoints involve checking the identity of requesters, having a JWT token allows us to effortlessly handle various requests with minimal server load. This is especially important as we expect many people to be making authenticated requests simultaneously (for example, while some users are viewing slots for events, some admins might be taking attendance).
- One drawback of the token based approach is that the JWT might be bigger than the corresponding cookie. Since Slotify doesn't store much user information within the token, the header size is acceptable.
- There are existing libraries such as Simple JWT and Django Rest Framework JWT which are well documented and supported by the community, and easy to use.

Milestone 12

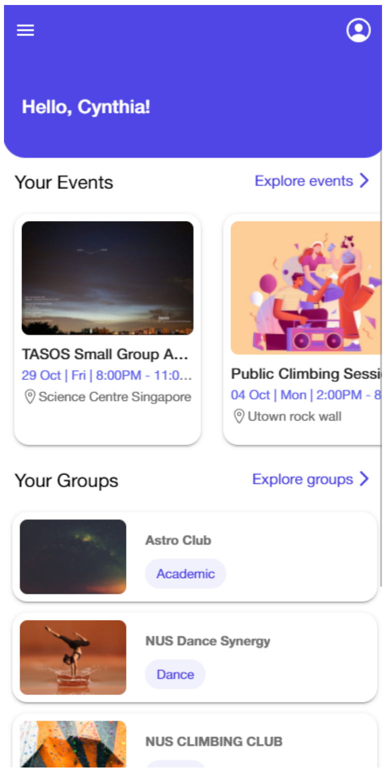
Our UI framework of choice: **Ionic**

Ionic framework was chosen as the framework as it provides the design and look that is most similar to our intended application design. It is more rounded as compared to other frameworks like Ratchet UI and Framework 7. It is also more stylised as compared to others wherein they provide a set of components but are more plain-looking. Ionic framework provides a more modern and native look for both ios and android mode. There are also a set of components that we found more useful like the horizontal slides and cards that our home page requires, and the slide-up date-pickers that are more user-friendly on phones.

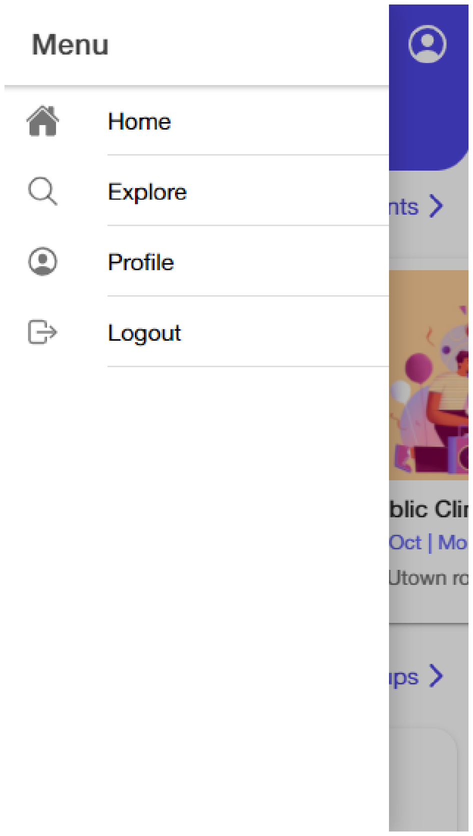
Mobile Site Design Principles

Keep calls to action front and center

The home screen shows the events the user is signed up for and the groups the user is part of, which are common actions that the user will perform. The user can perform more actions like explore events and groups with the provided buttons for each segment in the home page as well.

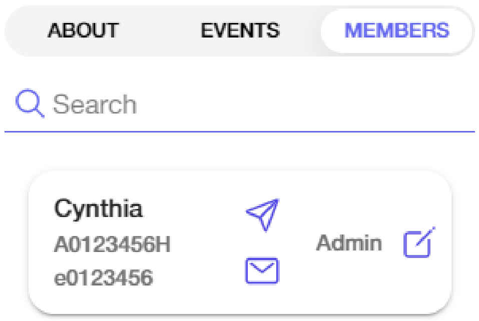


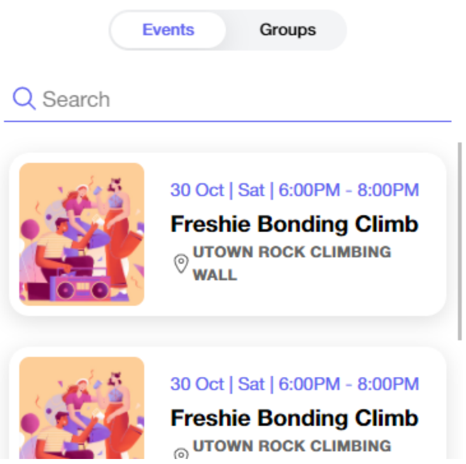
Keep menus short and sweet



Make site search visible

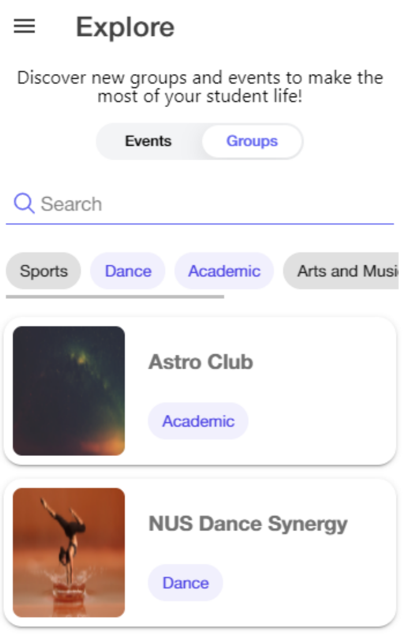
The search bars are not hidden whenever it is available. For example, in the explore page or the members tab under the group page.





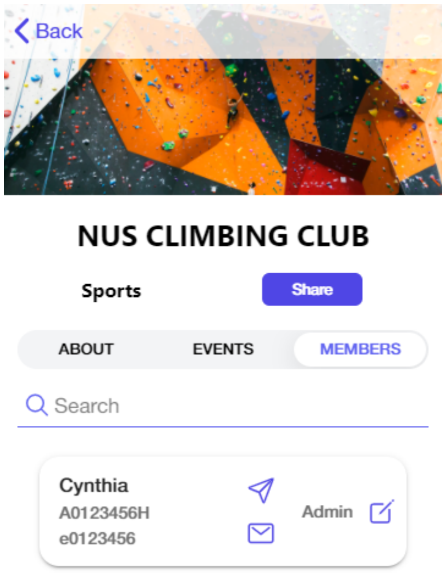
Implement filters to narrow results

The explore page allows search and filter of groups by their category.



Use click-to-call buttons for complex tasks

The share button in the group page opens the user's telegram to allow the user to select a recipient. Admins can also quickly contact a member by tapping on the telegram or email icon on the member's card to launch the respective application.



Milestone 13

Common Workflows

The three common workflows we have are:

- Create an event (for admins)
- Sign up for an event
- View event sign ups and take attendance (for admins)

We'll be mainly comparing our app with event sign up using google sheet as it is the most widely used approach in various CCAs.

Create an Event

Admins of a group can create events for the group. To do that, they can enter the group, go to the "EVENTS" segment and click on the "Create an Event" button at the top of the event list (the button won't appear if the user is not an admin of that group).

After clicking the "Create an Event". The admins will be brought to the event creation page, where they will fill in event details such as time, location and the maximum number of various slots.

After filling in event details, admins can press the "Create" button and go to the event page of the newly created event.

Then admins can press the "share" button to share the link to the event page on telegram with group members.

If the admins were to create the event with Google Sheets, they would first create a google sheet, try to format the sheet with decent coloring and spacing, and send the links to group members to let them fill up their information. Even with all these efforts, the google sheet might still not be intuitive as group members are flooded with overwhelming information. In comparison, it is very easy to create an event using Slotify: just fill in the event details and share the event with group members. Group members can then click on the link and view the event details in a neat way.

Sign up for an event

Users can sign up for events if they are eligible. To find an event to join, they can either go to explore events and search events based on keywords or directly go to a group and search for group-related events.

If users find the event they want to join, they can click on the event to go to the corresponding event page and available slots will be shown at the bottom.

If users are not eligible for a slot (e.g. the slot is for members, but the users are not members of the group), that slot will be greyed out. If the users click on a slot that is not full yet, they can join by clicking the "+" button, otherwise (the slot is full) they will be put on the waiting list.

If Google Sheets was used for event sign-up, a problem would immediately emerge: people can see each others' information which is a privacy concern. Moreover, it is much easier to view all the upcoming events in a list compared to clicking various Google Sheets one by one to check the time and location.

Furthermore, some CCAs have experience requirements for different slots, it is very tedious for admins to check one by one whether members signed up for certain slots have fulfilled the requirement. Also, in rare cases, some members might accidentally/intentionally delete other members' names. Whereas in Slotify, people can only sign up for eligible slots, and they don't have to worry about their sign-ups being tempered with.

Lastly, many CCA events have the waiting list mechanism: when people who have already signed up cannot come to the event, the first person on the waiting list will attend the event. This process requires admins' manual operations in Google Sheets. However, in Slotify, the first person on the waiting list will be automatically moved from the "waiting list" to the "confirmed up" list if someone cannot make it to the event.

View event sign-ups and take attendance (for admins)

Admins of the group can view all the sign-ups and take attendance. To do this, they can click on the "VIEW SIGNUPS" button below the group banner (the button is only visible if users are admins).

After clicking the button, a modal will show up, showing all the signups and remaining slots for various kinds of slots. People who have already signed up and those who are on the waiting list are shown in two separate sections.

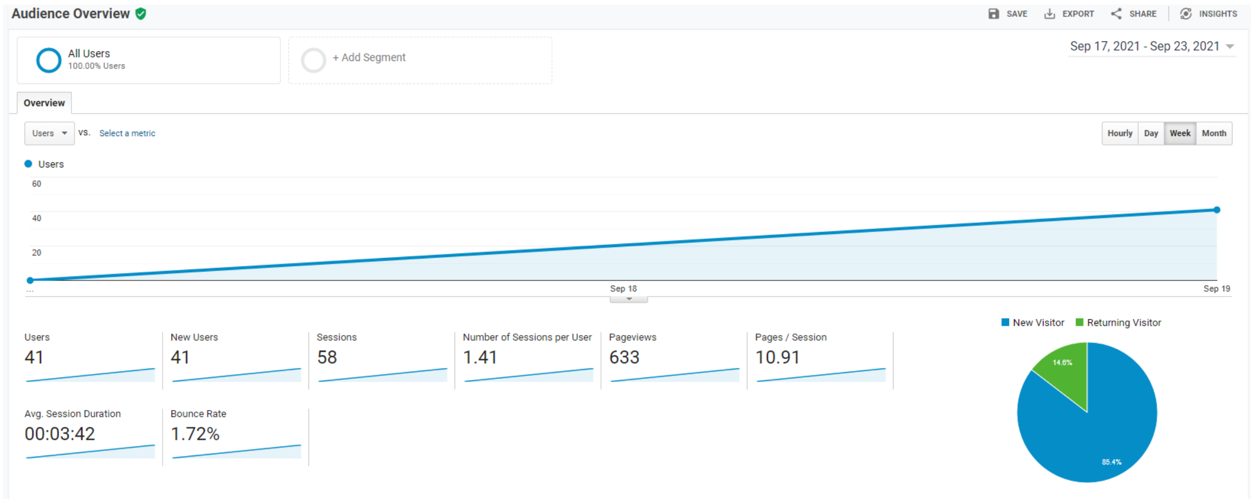
Admins can click the check box at the right-hand side of the sign-up card to confirm the attendee's attendance.

If admins used Google Sheets for event sign-ups, they usually have to extract the sign-up information and print out a piece of paper to take attendance. Often it is the attendee's job to draw a checkmark to indicate their attendance. However, this approach makes it very easy to fake each other's attendance. Additionally, admins will have a hard time compiling the attendance data later on if they need to. In comparison, all sign-up data are digital on Slotify, and admins can just click checkboxes to effortlessly take attendance.

Milestone 14

Google Analytics

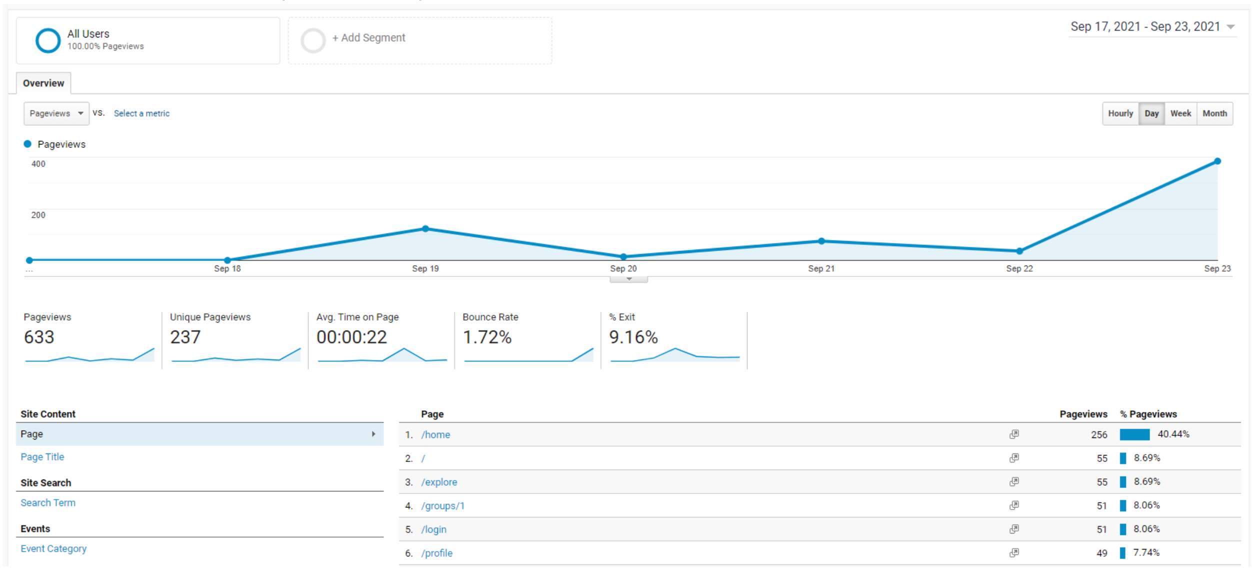
- Screenshot 1 (Audience)



Users	New Users	Sessions	Number of Sessions per User	Pageviews	Pages / Session
41	41	58	1.41	633	10.91

Avg. Session Duration	Bounce Rate
00:03:42	1.72%

- Screenshot 2 (Behaviour)



Pageviews	Unique Pageviews	Avg. Time on Page	Bounce Rate	% Exit
633	237	00:00:22	1.72%	9.16%

	Page	Pageviews	% Pageviews
1.	/home	256	40.44%
2.	/	55	8.69%
3.	/explore	55	8.69%
4.	/groups/1	51	8.06%
5.	/login	51	8.06%
6.	/profile	49	7.74%

Milestone 16

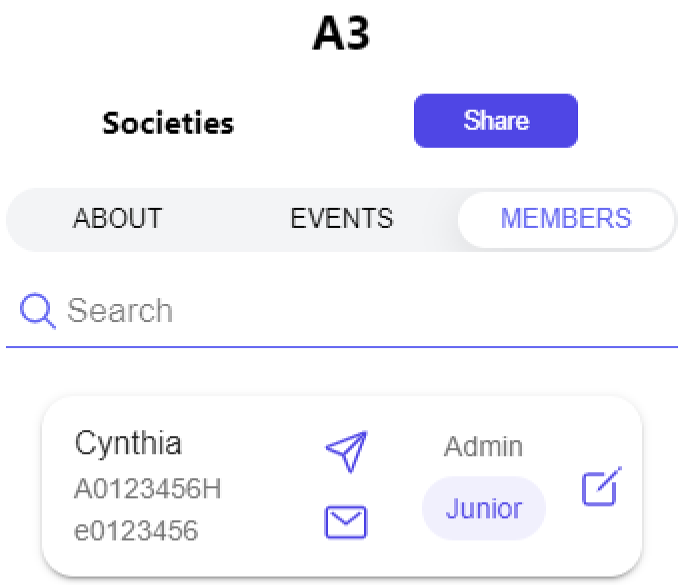
Google Social Integration

One of the reasons why we decided to build this application is due to the inefficient use of Google Spreadsheet to solve the issue of managing event attendance. We decided to include the ability to login via Google to allow ease of access. Users will be able to login and fill in their profile details.

Telegram Integration

As the popularity of Telegram is on the rise, we feel that having an easy way to link and integrate with Telegram will prove to be very convenient for our potential users. In our application, we have a few ways to interact with Telegram:

- The share button on individual group pages will open a web page that prompts you to open your Telegram application. Thereafter, you can choose a recipient and a message that includes the link to the group will be sent to that recipient.
- The members of a group will be able to see two icons for each member. The aeroplane icon will trigger the user to open the Telegram chat with the chosen member (if the telegram handle is provided). The mail icon will open the email client. Screenshot as follows:



-

END