TREECKLE

CS3219 Software Engineering Principles and Patterns

Project Report

Source code: https://github.com/CS3219-SE-Principles-and-Patterns/cs3219-ay2021-s1-project-2020-s1-g12

Deployed web app: https://v2.treeckle.com

Member	Hong Shao Yi	Ng Yi Long, Kester	Ooi Ming Sheng	Tan Kai Qun, Jeremy
Student Number	A0183289N	A0136125N	A00984780W	A0136134N

1. Individual Contributions

All group members contributed to the deployment of the application. See below for further details.

Hong Shao Yi	**Report** - [3] Requirements Specification - Appendix: Use Cases
	Code - [F1.3] User management UI and integration - [F2.2, F2.3] Front-end design and UI for bookings page - [F2.4] UI for comments and integration - Integration of [F2] and [F1] - Configure frontend environment for deployment
Ng Yi Long, Kester	Report - [3] Requirement Specification - [4] Software Development Process - [5.8] Application Flow
	Code - [F1.1] Mass invite emails and send email invitation to invited users - [F2.2, F2.3] Booking models, CRUD, auto updating of status upon an approved booking - [F2.4] Added Comment models, CRUD.

	- [F2.5] Auto email sending for creation of booking, updates to booking and addition of comments - Integration of [F2] and [F1] . - Configure backend environment for deployment
Ooi Ming Sheng	**Report** - [5.5] Back-end architecture design - [5.6] Database design
	Code - [F1.1, F1.2] Implemented CRUD functionalities for users and added login implementation. - [F2.1] Added CRUD for venues - [F3] Added event CRUD, recommendations, subscriptions, registration - Integration of [F3] - Configure deployment emailing service
Tan Kai Qun, Jeremy	**Report** - [2] Introduction - [5.1, 5.2, 5.3, 5.4, 5.7] Application Design - [6] Other Design Considerations - [7] Future Plans
	Code - [F2.1] Venues Creation UI

	- [F3.1] Events UI - [F3.2] Event registration and QR code UI - [F3.4] Event subscription UI - Integration of [F3] - Set up travis CI - Dockerise app for deployment - Setup DigitalOcean virtual machine for deployment

2. Introduction

2.1 Background

The NUS Residential Colleges (RCs) serve as an integrated space for its residents to interact with one another and to enjoy an enriching communal living experience. Every semester, each of the colleges have close to a 1000 facilities bookings being made and around 100+ house/college events being held. Yet, all of the colleges face a common issue and that is the lack of an efficient system to effectively manage the facilities bookings and event signups. For instance, in Tembusu, one must contact the Enrichment Director of the College Students Committee (CSC) directly through private messaging in order to book a facility. On the other hand, in CAPT, one has to repeatedly fill up the same signup forms with the same fields for every single event they want to attend.

With these pressing problems affecting thousands of residents across all RCs every semester, project Treeckle, was started in 2019 to help resolve these inconveniences. Treeckle aims to be a one-stop platform for all our residential needs. With Treeckle, residents will be able to book facilities and sign up for events with ease. For the admins, they will be able to manage the bookings, event sign-ups and many more... All these on one centralized platform.

2.2 Treeckle 2.0

Treeckle 2.0 is built upon the concept and motivations of Treeckle but with a complete overhaul of the codebase, tech stack and app architecture. Treeckle 2.0 is designed to be highly maintainable, extensible and scalable, and is developed to support both mobile and

desktop view. Additionally, a wide range of features were carefully curated and implemented in Treeckle 2.0 to further support our residential needs, and they will be further elaborated in the subsequent sections.

3. Requirements Specification

3.1 Organisations

In Treeckle 2.0, we term a Residential College as an organisation. Organisations serve to partition users, venues and events according to the Residential College they belong to. Thus, this gives rise to a multi-tenancy mode of operation, which will be elaborated in Section 6.3.

3.2 User Roles and Permissions

The following table delineates the 3 user roles which Treeckle 2.0 caters for. Their respective access rights are also broadly listed here. Detailed access rights for each user role can be found in the Functional Requirements while an explanation of each feature mentioned here can be found in the User Requirements.

User Role	General Access Rights
Resident	They correspond to all residents living in a RC. They are granted basic access to Treeckle 2.0's features such as: ● Event Registration, Recommendation & Subscription ● Venue Booking & Booking Comment ● Email Notifications
Organiser	They correspond to residents holding executive positions in Interest Groups and/or College Students' Committee in a RC. In addition to the basic access, they are also full granted access to Treeckle 2.0's event management features such as: ● Events Management
Admin	They correspond to residents holding upper management positions in a RC. They are granted full access to Treeckle 2.0, including

	administrative features like: • User Management • Venue Management

3.3 Features

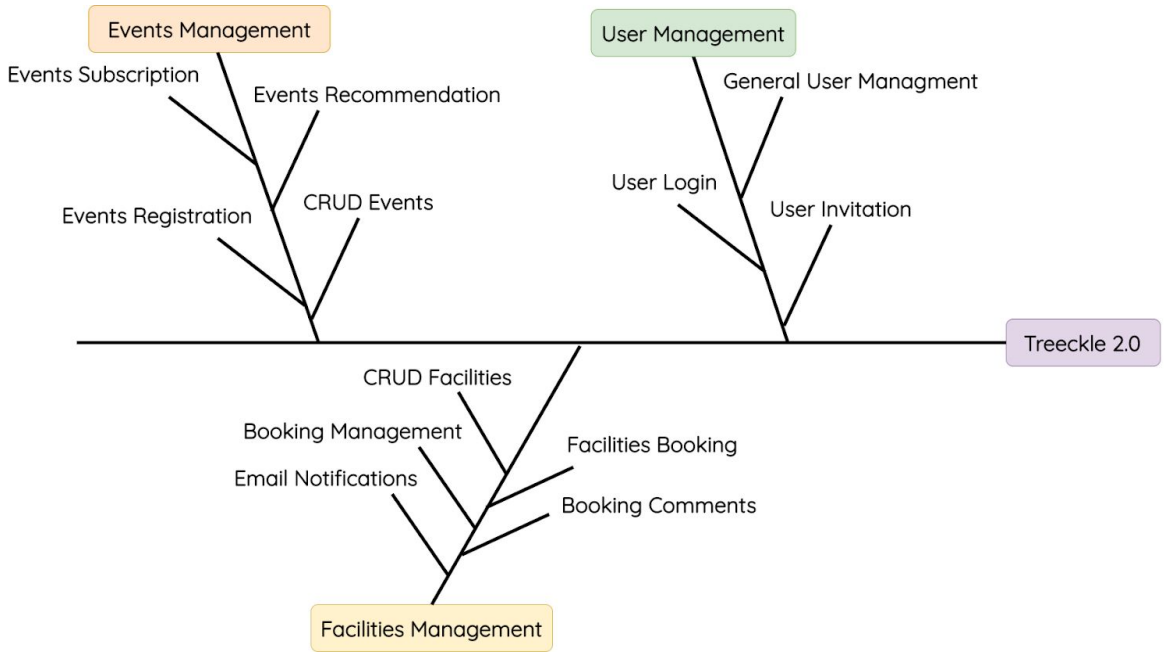
Treeckle 2.0 provides 3 main feature domains, broadly explained below.

Feature Domain	
Feature	**Description**
Venue Management	
CRUD Venue	• Venue in Treeckle 2.0 are conceptualised as an entity, each with its unique name, booking form and set of associated bookings. Currently, the venue is the only facility that can be booked. • An admin is able to create, edit and delete venues belonging to his/her organisation. • Each facility in Treeckle 2.0 is tagged with a custom form to be filled in by interested bookers. This form is customizable by the admin, and is generated using a form builder when creating a facility. This allows each booking request to capture a unique set of additional information, as per the requirements of each facility.
Venue Booking	• Booking requests are another entity in Treeckle 2.0. • All users are able to book venue available in their organisation, limited to the date and time slots with no approved bookings. • They are required to submit a set of basic information relevant to the booking (facility-of-interest and date time slots), as well as a set of additional information unique to each facility, as stipulated by the creator of this facility.

	• Users are also able to perform mass booking, replicating the request across different dates available for booking.
Booking Management	• All booking requests made will have to be approved by their organisational admins before they are approved. Once approved, all pending bookings with clashing time slots will automatically be rejected.
Booking Comments	• Users are able to add comments for each booking request made by them, serving as a side communication channel with admins. • All admins can also comment on all booking requests in their organisation, thus establishing bilateral communication.
Email Notifications	• When a booking request is created, updated or cancelled, an email notification will be sent to all interested parties i.e. the user who created the booking and the approver (if relevant). • Similarly, whenever a comment is created, an email notification will be sent to all interested parties which includes the creator of the booking and anyone who has also commented on that particular booking. • Additionally, an admin may stipulate an additional organisational email for which these email notifications will be forwarded to, thus allowing all booking requests to be tracked.
Events Management	
CRUD Events	• Events are another entity in Treeckle 2.0 • Organisers and admins are able to create, edit and delete events for their respective organisations. • Some notable attributes of events: a. A poster image that serves as attractive graphical medium for publicizing the event b. Categories that characterize the event (eg. sports, technology, etc) • Events must also specify if they are signup-able or not. If they are, the event must also specify if the signup needs to be approved by the organiser. Hence, users will have to register for the event and be approved in order to participate in the event.

Events Registration	• All users can register for a signup-able event. • Organisers are able to generate a unique QR code that allows participants to indicate their attendance on the day of the event.
Events Recommendation	• Users view recommendations of events similar to those that they have attended. This recommendation is performed by a recommender system, powered using the Cosine Similarity algorithm.
Events Subscription	• Users can indicate event categories that they are interested in and view a feed of all events that fall under these categories.
User Management	
User Invitation	• Admins can add users into Treeckle 2.0 by submitting the residents' Google emails or NUSNET emails into the system. Only residents with their email in the invitation log can login as a user. • Admins can also upload a CSV file containing the emails, which will be automatically parsed by the system for submission.
User Login	• Users can login into Treeckle 2.0 by establishing an authentication with either their Google account or NUSNET account. • User registration happens under the hood when the user login for the first time.
General User Management	• Admins can view all users in their own organisations. • Admins can also change the role of Residents and Organisers, but not the role of other Admins.

The following diagram illustrates a brief overview of the features in Treeckle 2.0 via feature tree.



Feature Tree for Treeckle 2.0

3.4 Functional Requirements

The following sections specify the functional requirements (FRs) of Treeckle 2.0. All the planned FRs have been implemented and deployed to production. Each table corresponds to a feature domain, as seen in the section above, and each functional requirement is prioritized according to the priority legend below.

Priority Legend	
High	Mandatory for the success of the project
Medium	Important for the full user experience
Low	Implement if sufficient time and resources

In addition, each functional requirement is linked to a use case, which can be found in the Appendix.

3.4.1 User Management

S/N	Description	Trace to Use Case	Priority	Effort
F1	**User Management**			
F1.1	**User Invitation**			
F1.1.1	The application should allow admins to invite new users to their organisation by submitting their Google account email.	UC5	High	1 hour
F1.1.2	The application should allow admins to invite new users to their organisation by submitting their NUSNET email.		High	1 hour
F1.1.3	The application should allow admins to mass invite users by uploading a CSV file containing the emails.	UC4	Medium	1 day
F1.1.4	The application should send an email invitation to all successfully invited emails. The email should prompt the user towards the application link, where they can login into the application.	UC4 & UC5	High	3 hours
F1.2	**User Login**			
F1.2.1	The application should allow users to log in using their NUSNET email.	UC1	High	1 hour
F1.2.2	The application should allow users to log in using their Google account email.	UC2	High	1 hour
F1.3	**General User Management**			
F1.3.1	The application should allow admins to view all users in their organisation.	UC3	High	1 day
F1.3.2	The application should allow admins to search for users in their organisation.		Medium	1 day

F1.3.3	The application should allow admins to sort the listing of users in their organisation.		Medium	1 day
F1.3.4	The application should allow admins to change the role of residents and organisers in their organisation.	UC6	High	2 hours
F1.3.5	The application should not allow admins to change the role of other admins in their organisation.		High	2 hours

3.4.2 Venue Management

S/N	Description	Trace to Use Case	Priority	Effort
F2	**Venue Management**			
F2.1	**CRUD venues**			
F2.1.1	The application should allow users to view all venues found in the organisation.	UC7	High	1 day
F2.1.2	The application should allow admins to create new venues in their organisation. The basic details required to create each venue consist of: • Facility name • Category: Seminar Room / Multipurpose Hall	UC8	High	1 day
F2.1.3	The application should allow admins to create a custom booking form for each facility that is created. Each field in this booking form can be customized according to their field type, field label, placeholder and whether it is a required field. The types of field input that can be created are: • Text • Textarea		High	3 days

	• Number • Boolean			
F2.1.4	The application should allow admins to remove existing venues in their organisation.	UC9	High	2 hours
F2.1.5	The application should allow admins to edit the basic details of existing venues, as well as their custom booking forms, in their organisation.	UC10	High	3 hours
F2.2	**Venue Booking**			
F2.2.1	The application should allow users to book venues in available date time slots. A given time period is considered available for booking when there are no approved bookings for that facility within that period. Each booking request should minimally contain: • Name of the facility to be booked • Start date and time • End date and time	UC12	High	2 days
F2.2.2	The application should require users to fill in an additional custom booking form for each facility, on top of the basic information stated above.		High	1 day
F2.2.3	The application could allow users to mass book venue. This is performed via a recurring fashion, where the same time slot and facility is booked across multiple dates.		Medium	1 day
F2.2.4	The application should allow users to view their booking requests as well as their booking requests.	UC13	High	1 day
F2.2.5	The application should allow users to cancel any booking requests that were made by them.	UC14	Medium	1 hour
F2.3	**Booking Management**			

F2.3.1	The application should allow admins to view all booking requests in their assigned organisation.	UC11	High	1 day
F2.3.2	The application should allow admins to approve or reject booking requests for their own organisation only.	UC17	High	2 hours
F2.3.3	When a booking request for a facility is approved, the application should automatically reject all pending booking requests for that particular facility that fall within the date time range of the approved request.		High	1 day
F2.4	**Booking Comments**			
F2.4.1	The application should allow the respective booker to add comments to his/her booking requests.	UC16	Medium	1 day
F2.4.2	The application should allow admins or organisers to add comments to all booking requests within their organisation.	UC15	Medium	2 hours
F2.4.3	The application should allow users to view all comments made to a booking request.	UC11	Medium	3 hours
F2.5	**Email Notifications**			
F2.5.1	The application should send all email notifications to users' NUSNET email or Google mail, depending on which authentication system the user has signed up with.	UC12 & UC14-17	Medium	1 day
F2.5.2	The application should send automated email notifications to users and organisation listeners when their booking request status is changed.	UC14 & UC17	Medium	1 day
F2.5.3	The application should send automated email notifications to users when they create a booking request.	UC12	Low	1 day
F2.5.5	The application should send automated email notifications to the booker and all other commenters of a booking request when a new comment is created on it.	UC15 & UC16	Low	1 day

F2.5.6	The application should allow admins to configure a list of emails to receive booking request related notifications	UC28	Medium	6 hours

3.4.3 Events Management

S/N	Description	Trace to Use Case	Priority	Effort
F3	**Events Management**			
F3.1	**CRUD Events**			
F3.1.1	The application should allow users to view all college events in their organisations.	UC18	High	1 day
F3.1.2	The application should allow organisers to create and publish new college events. Event details include their • Title • Description • Date time • Recommended Capacity • Poster image.	UC21	High	2 days
F3.1.3	The application should allow organisers to indicate whether an event is a signup-able event or not.		High	1 hour
F3.1.4	The application should allow organisers to indicate whether the event requires the organisers' approval for successful sign up.		High	1 hour
F3.1.5	The application should allow organisers to upload a poster image for an event.		Medium	1 day
F3.1.6	The application should allow organisers to label an event under several categories that characterize it.		Medium	3 hours

F3.1.7	The application should allow organisers to create new categories to group an event under.		Low	1 day
F3.1.8	The application should allow organisers to update existing events details.	UC22	High	4 hours
F3.1.9	The application should allow organisers to remove created events.	UC23	High	2 hours
F3.1.10	When an event is removed, if the event is still open for registration and has sign ups, the application should remove these sign up entries well.		High	1 day
F3.1.11	The application should allow organisers to indicate whether an event is published or not.	UC21	High	2 hours
F3.1.12	The application should allow all published events to be viewable to users and vice versa.	UC18	High	1 day
F3.2	**Event Registration**			
F3.2.1	The application should allow users to sign up for events in their organisation.	UC20	High	1 day
F3.2.2	If an event has a maximum capacity specified and the capacity is reached, the application should prevent users from signing up for this event.		High	3 hours
F3.2.3	The application should allow residents to view the events they have signed up for and their sign up status. The sign up status is applicable only if the event is a registrable event.	UC18	High	3 hours
F3.2.4	The application should be able to generate a unique QR code for each event, allowing participants to indicate their attendance by scanning this code.	UC26	High	2 day
F3.2.5	The application should allow all participants of an event to indicate their event attendance by scanning a QR code.		High	1 day
F3.2.6	The application should allow organisers to view event sign ups and attendance.	UC24	High	1 day

F3.2.7	The application should allow organisers to approve/disapprove event sign-ups. This is only applicable for signup-able events that require approval.	UC25	High	1 day
F3.3	**Events Recommendation**			
F3.3.1	The application should be able to generate a list of recommended upcoming events for users. This list of events is based on the events' similarity with the title and description of past events that the user has attended, via the Cosine Similarity algorithm.	UC18	Medium	4 days
F3.3.2	The application should allow users to view events that are recommended to them.		Medium	1 day
F3.4	**Events Subscription**			
F3.4.1	The application should allow users to indicate event categories that they are interested in.	UC19	Medium	1 day
F3.4.2	The application should allow users to view a feed of upcoming events that matches the event categories that they are interested in.	UC18	Medium	1 day

3.5 Non-Functional Requirements

S/N	Description
NF1	**System Requirement**
NF1.1	Treeckle 2.0 should be mobile-friendly and responsive.
NF1.2	Treeckle 2.0 should run on all LTS versions of major browsers, such as: • Chrome v85 • Firefox v80

	• Safari v14
NF2	**Constraints**
NF2.1	Treeckle 2.0 only supports comma separated values (CSV) files for mass user invitation.
NF2.2	Treeckle 2.0 requires internet connectivity to function.
NF2.3	Treeckle 2.0 can only accept event poster images with a fixed dimension of A4 size, a maximum image size of 2MB and limited to png and jpeg format.
NF2	**Performance**
NF2.1	The webpage must provide 3 seconds or less First Contentful Paint (FCP) time, including the rendering of text and images, over an LTE connection.
NF2.2	Each request should be processed within 5 seconds.
NF2.3	The system should be able to support at least 200 concurrent users at any instance.
NF3	**Security**
NF3.1	Approval of booking requests and invitation of users to the organisation can only be made by an admin.
NF3.2	All API calls, with the exception of login, must be authenticated with an access token, so that only authorized users can make these calls.
NF3.3	Users must be registered by the system before being able to log in into the platform.
NF4	**Usability**
NF4.1	An untrained user must be able to submit a booking request in an average of 4 minutes.
NF5	**Availability**

NF5.1	The system should have an uptime of 99% at all times.
NF5.2	All system errors leading to significant downtime should be resolved within an hour.
NF6	**Scalability**
NF6.1	All backend containers should maintain 70% capacity at all times. An auto scaler shall spin up additional instances when this condition is being breached due to an increase in number of users.
NF7	**Reliability**
NF7.1	During mass account invitation, the application must be able to send to at least 500 different email recipients with an overall 95% success rate.
NF8	**Extensibility**
NF8.1	A developer that is familiar with the codebase will be able to add new features with 3 hours or less development effort.
NF9	**Integrity**
NF9.1	The system shall protect against the unauthorized addition, deletion or modification of data.

3.6 Quality Attributes Prioritization Matrix

The following table illustrates the prioritization of quality attributes as specified in the section above.

Attribute	Score	Performance	Security	Usability	Availability	Scalability	Reliability	Extensibility	Integrity
Performance	3		^	^	<	<	<	^	^
Security	5			^	<	<	<	^	<
Usability	5				^	<	<	^	<
Availability	2					<	^	^	^
Scalability	2						<	^	<
Reliability	2							<	^
Extensibility	5								^
Integrity	4								

Following the prioritization matrix, our team decided to focus on the top 3 attributes, namely: extensibility, usability, and security.

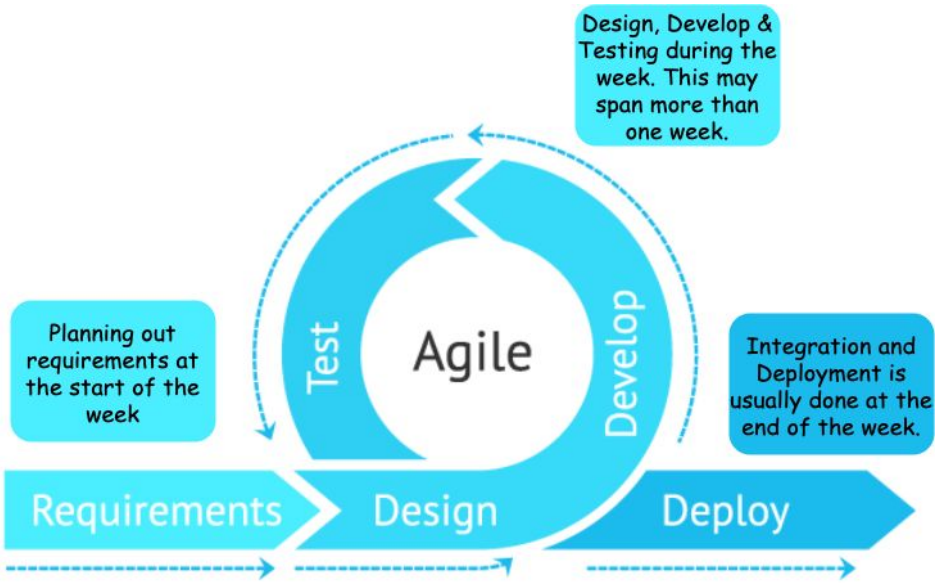
A focus on extensibility meant that much of our components should follow design principles that promote looser coupling and reusability. Doing so facilitates greater understandability of the components, allowing future developers to easily assimilate into the codebase and make early, quality contributions. For instance, one approach to this is to write reusable UI components, defined only by its presentation styles and generic logic in the application. This promotes consistency between styles of UI components and allows decoupling between presentation styles and the business logic. A developer would only have to pass these components the data to be rendered and callback functions to be invoked, with minimal concern over the styling of these components.

A focus on usability meant that our web application should take minimal effort to understand and operate. Besides ease of use, a greater emphasis is placed on memorability as well as error handling and avoidance. For instance, all form inputs should be validated. In the scenario of an erroneous input, the application would provide a user-friendly and informative message that enables the user to correct the input, leading to a positive user journey. Thus, our application should provide positive feedback to the user, which will promote active interaction with the application as well.

Lastly, a focus on security meant unauthorized access to our application API and data should be blocked. As our application handles personal data, these information are sensitive and should not be easily accessed by unwanted parties. An approach utilized by our team to ensure security can be found on section 6.2.

4. Software Development Process

Our team used an agile development process for this project. In the beginning, we discussed what features to develop and write down the FRs so that we can assign each FR with a priority as well as an estimated amount of effort (in hours) needed. Then, we made a general timeline to note down the implementation deadline of each of the high-level features and who is responsible for implementing the associated FRs. Naturally, FRs that are of higher priority will be developed first.



Agile workflow

Our weekly workflow follows the diagram above which is typical for an agile team. We will usually meet at the start of the week, typically on Mondays, to update what each of us will be implementing in this week. During the meeting, Jeremy will also highlight on the current overall progress, what has been developed last week as well as what has been deployed to production (only for the last 1 month).

Subsequently, we will then spend the next 4-5 days designing, developing and self-testing the features. Usually, we will stop development by Saturday and over the weekend, we will review and test each other's PR to ensure that the FRs are well implemented before merging to master. In the last 1 month, weekly deployment to production is also being executed so that we can avoid a large scale deployment at the end which is extremely risky.

In our development process, we also used Travis CI to help us with continuous integration to check for compile/build errors so that we can ensure the product build on master is deployable. Additionally, we also employed end-to-end testing to ensure proper application workflow and integration. Finally, in the last one month, we performed user

testing with our fellow RC residents to ensure that the application satisfies the functional requirements, user goals and provides a good UX.

5. Application Design

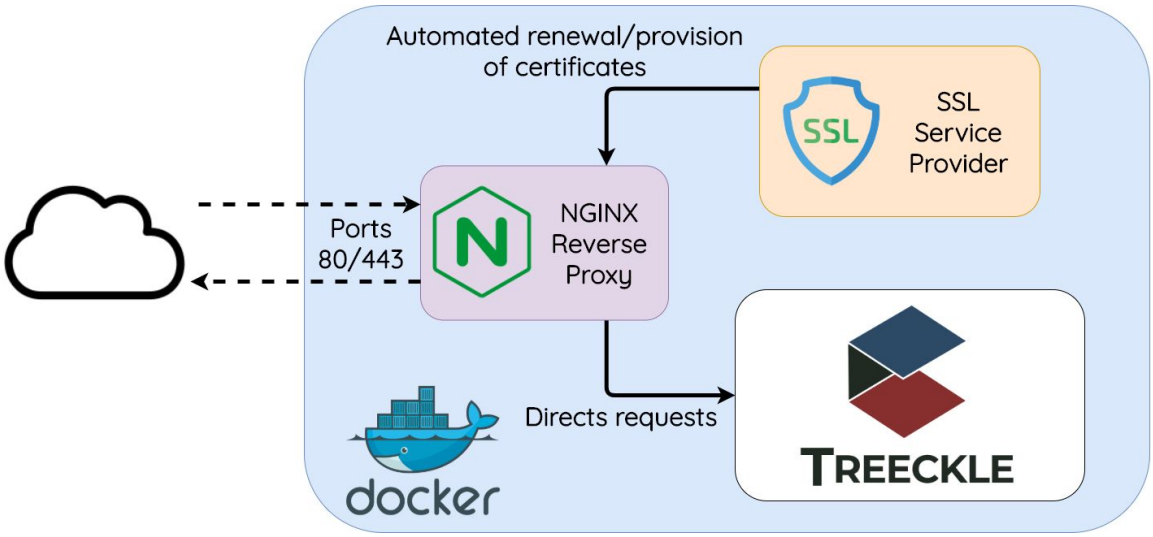
5.1 Tech Stack

Choice	Rationale
Frontend: React	- Fast render because of virtual DOM. - A lot of support from the community. - Component-based programming with clean abstraction. This facilitates creation of highly reusable components. - Fast learning curve.
Backend: Django	- Highly opinionated framework which promotes a structured and maintainable codebase and encourages developers to conform to certain best practices. - Allows rapid development and easy testing. - A lot of support from the community. - Includes a very powerful Django admin feature that can be used for internal management of site content and database.
Database: PostgreSQL	- Popular relational database with a lot of support from the community. - App content is highly relational so it makes sense to store the content in a relational database.
CI Tool: Travis CI	- Enables automatic testing of builds on github branches.
Deployment Tool: Docker	- Manages complexity of large-scale/multiple app deployments.

	- Version control app containers on Docker Hub.
Deployment Platform: DigitalOcean	- Cheap ($5/month). - Highly scalable.
Image CDN: ImageKit.io	- Free. - Easy-to-use SDK.
Email Service: Mailgun	- Free 5000 emails/month. - Easy-to-use SDK.

5.2 Deployed Architecture in Docker

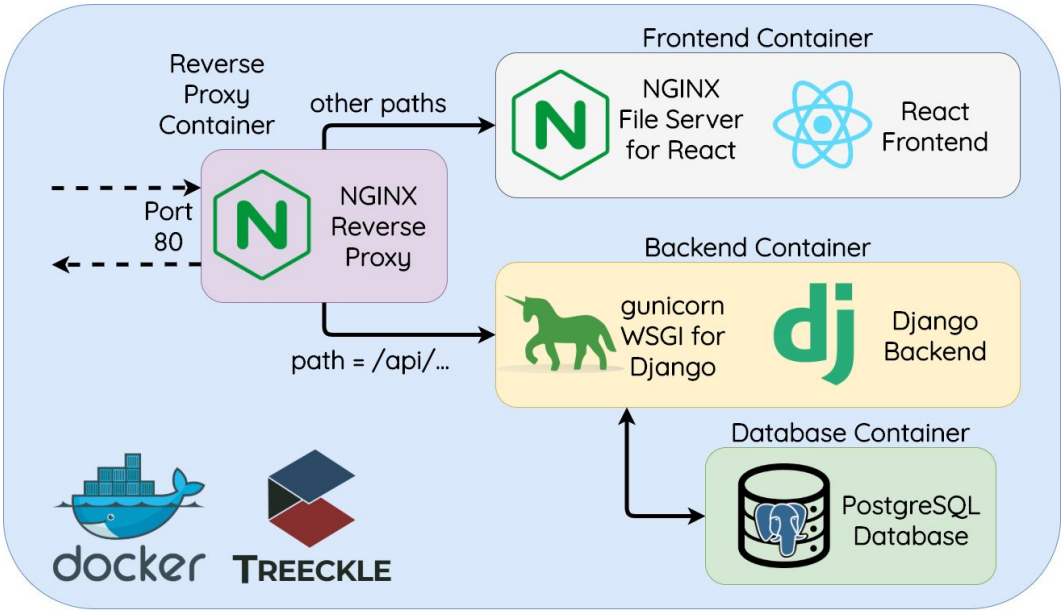
This app is designed to be highly extensible and scalable as our team envision the app to continue to evolve even after this module, when we allow the different RCs to contribute to the development of the app. As such, our team has decided to dockerise the app to facilitate easy scaling and deployment.



Top layer architecture in docker

This top layer reverse proxy container manages requests across the different apps under the same domain. Since currently there is only one instance of Treeckle 2.0 running, all requests will be directed to it. However, if in future the different RCs want to deploy their own version of the Treeckle 2.0 app, they can easily link up their own variant of the app

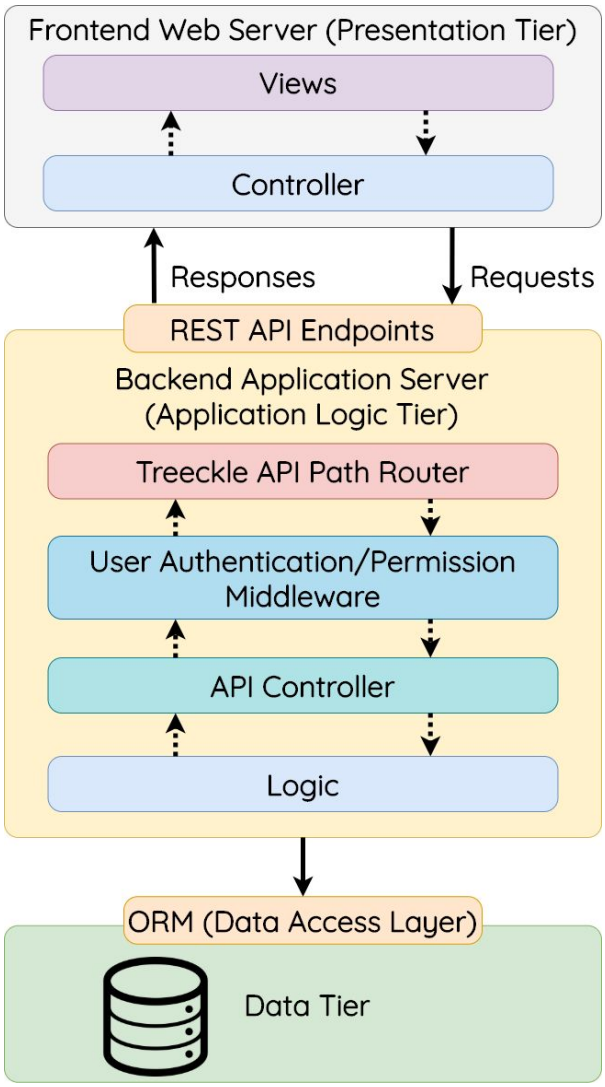
container with the top layer reverse proxy under a different subdomain or path and the deployment is done. There is no need for the developer to setup additional configurations such as HTTPS as this will be automatically provided whenever a new app is discovered by the SSL service provider.



Treeckle 2.0 app architecture in docker

The frontend, backend and database components of Treeckle 2.0 are also containerised to facilitate better portability, easy testing and deployment.

5.3 Overall Application Architecture



3-tier client-server architecture

Treeckle 2.0 is a single-page-application (SPA) which implements a 3-tier client-server architecture. For the 1st tier, we have the presentation tier that renders the web pages to the user and manages the local app state and logic. Next, the application logic tier provides REST API endpoints for the client to interact with as well as handles the core app logic and processing. Within the application logic tier, there are several layers stacked onto one another that processes the received requests sequentially from top to bottom. Once the requests are processed, the responses (if any) will be propagated up the layers and sent back to the client. Finally, we have the data tier which is responsible for providing and managing persistence.

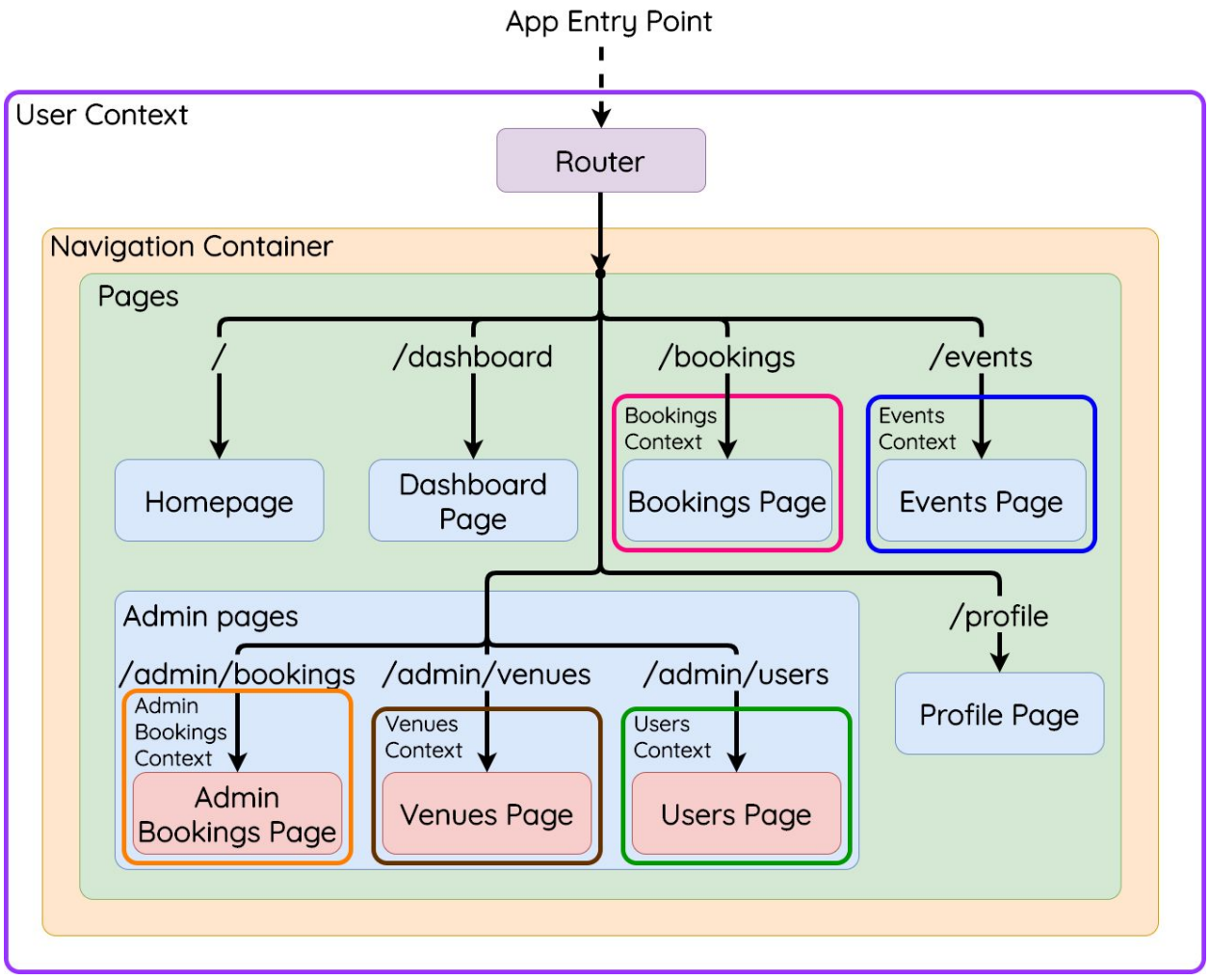
A multi-tier architecture was chosen for this app as it provides the following advantages:

1. **Improve modularity of the web app.** E.g. the key components (presentation, application logic and data) of the web app are clearly segregated and encapsulated. Each of the tiers has a distinct and specific role and they can be developed in parallel which speeds up the development process.

2. **Minimal coupling between the tiers.** The application logic and data tiers provide well-defined interfaces (REST API endpoints and ORM) that serve as a means for the other tiers to communicate. Since only the interfaces are exposed, the underlying implementations of the tiers are abstracted and independent of each other.

3. Following on the previous point, **the adaptability of this app is greatly improved** as a result. With minimal coupling, this architecture allows any of the 3 tiers to be upgraded or replaced independently in response to changes in requirements or technology. E.g. if this app were to support native platforms (E.g. iOS and Android), we can simply develop another presentation tier with native view and it can interact with the existing backend application server through REST API. No other modification is required for the other tiers.

5.4 Frontend Architecture



High-level frontend component layout

Since our app is a SPA, there will only be a one-time load of the webpage file at the start from the server and subsequently, frontend routing will be used to decide the pages to be rendered. While this improves the speed of transitioning between pages, the frontend has to do most of the heavy lifting and thus the complexity of the frontend also increases. As such, our team has decided to design the frontend architecture to follow a hierarchical structure that closely models after the Document Object Model (DOM) tree of the browser. It is worthy to note that in this architecture, composition is preferred over inheritance and components are repeatedly composed to form sophisticated UI.

The router component is the first point of entry to the app and depending on the current path and login state of the session, it will render the associated page to the user.

The navigation container component is responsible for listening to the viewport dimensions and rendering the correct view (mobile or desktop view). As such, it wraps all the other pages/components.

The user context contains information about the current user session such as if the user is logged in, the access token that will be used for making API calls to the server, etc. From the above diagram, all the components which are nested within the user context will have access to the information. This form of data sharing design is actually an extension of the flux pattern where instead of only having a single view component listening to the store, we now can have any number of components subscribing to the single context store. The following are various advantages of using context:

1. **Single source of truth for subscribed components.** When we need multiple components to share the same data, it can be extremely difficult to sync and make sure the components have the latest most updated data. Using context helps to overcome this problem as now the information stored in the context will be shared among all the subscribed components simultaneously. Even when an action has triggered a change in the context data, we can be sure that this new data will be propagated to the components and these components will have the most updated version of the data. For the case of the user context, whenever the access token is expired or refreshed, the components will all see the updated access token and they can continue to execute the appropriate action (e.g. make API calls or log the user out).

2. **Limit access to which components can view or change its data.** Unlike other interaction patterns such as the publish–subscribe pattern where any components can subscribe to the central messaging system to receive updates, this context pattern is more restrictive in the way it only allows those components which are nested within itself to have access to its data store. This allows developers to define clear boundaries as to which set of components can access the context. This greatly improves code readability as well as prevent developers from accidentally accessing unrelated context from other components during development. As shown in the above diagram, the feature pages each have their own context and cross feature access of context is not allowed.

3. **Versatility of context.** If we want to allow cross feature access of context, we can simply shift the context boundary to include more than 1 feature (imagine expanding a feature context box in the diagram above to include more features).

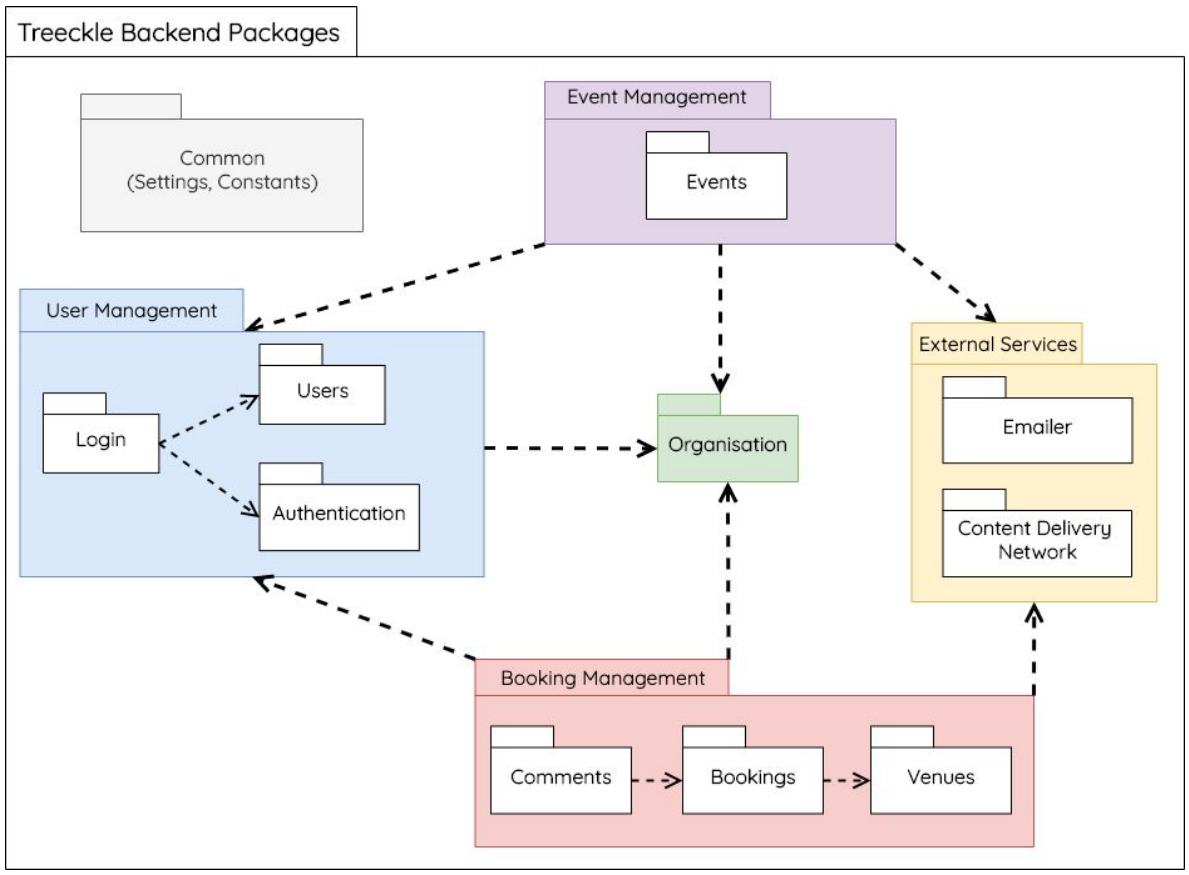
While context is useful in helping to manage the complexity of our app, there are some trade-offs to consider:

1. **Code overhead.** The creation of context requires quite a fair amount of boilerplate code to initialise and define the store shape as well as the actions available.
2. Following on the previous point, **overusing of context could slow down development and also negatively affect code readability**.

Considering the trade-offs, our team decided to use context sparingly and only for sharing feature/page-level data. For the smaller individual components, using React's builtin flux pattern (which requires no additional setup) is sufficient.

With this frontend design, adding a new feature UI, such as a forum page as described in section 7.1.2, will be simply defining the respective forum page components and perhaps a forum context, and adding them as a new route from the router.

5.5 Backend Architecture



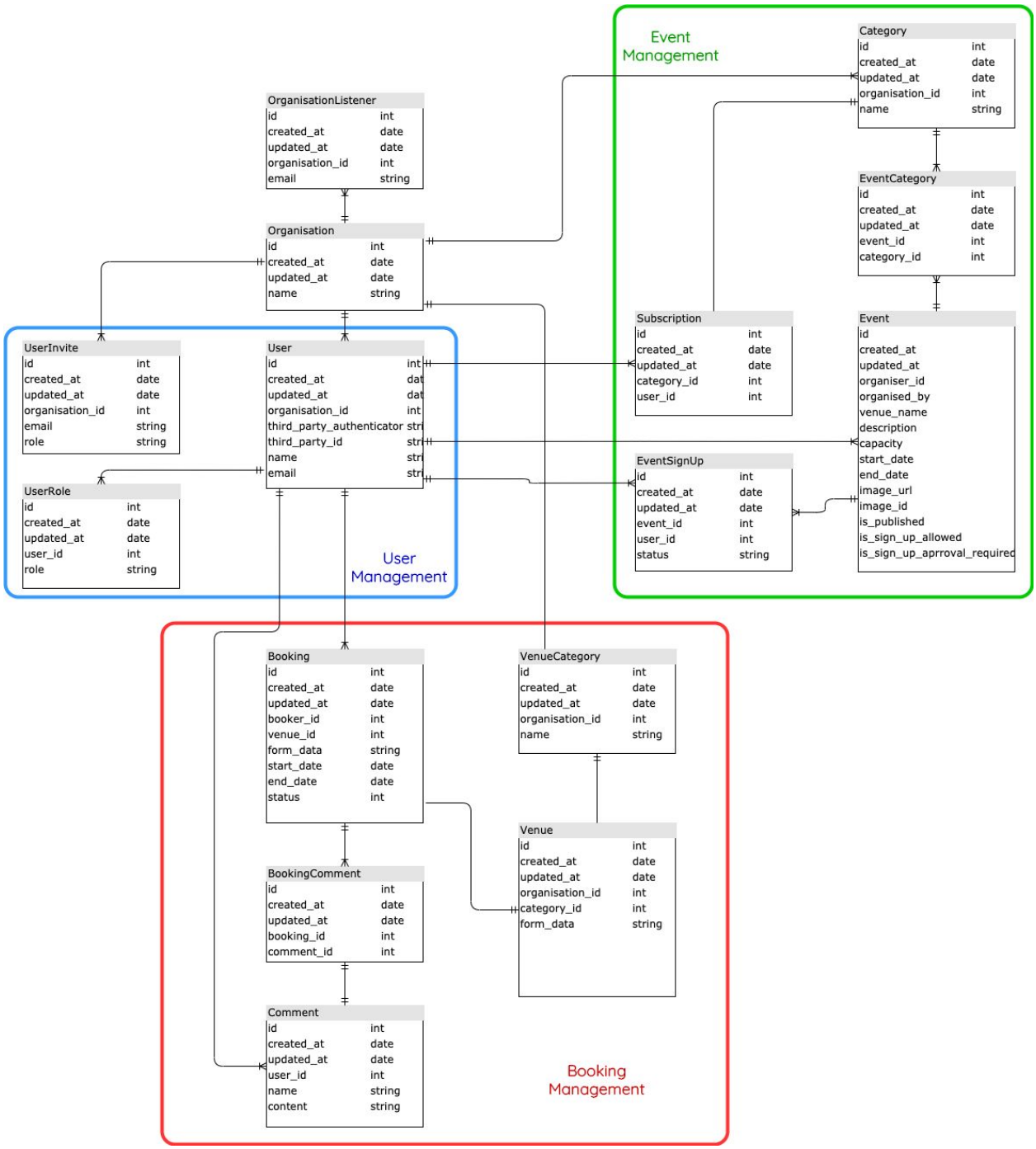
Backend architecture

The backend architecture style is inspired by Domain-driven design (DDD). Consequently, in each package, the structure and language of software code matches the business domain. Overall, there are 2 common packages (users management and organisations) that will generally be expected to be used by any new packages created. Beyond that, one can observe that there is very low coupling among the other business domains (e.g. events and booking management). Furthermore, the packages with many dependents (e.g. users and organisations) are expected to be relatively stable throughout the lifetime of the application. This design is intentional to allow domains to develop independently as much as possible, improving the rate of development for future versions of the application.

Each package may have its own API endpoint exposed. In such a case, the internal software structure of each package is decomposed into layers (e.g. router, middleware, controller, logic, data access layer) as outlined in section 5.3. As can be observed from the diagrams in section 5.3 and section 5.5, there was much intentionality taken to decouple the software both vertically by layers and horizontally by business domains.

Consequently, with this design, it would be a simple addition of another package if one would like to add a new business domain. This is highly maintainable as the architecture encourages proper separation of concerns and the single responsibility principle, making the code readable, incredibly flexible and easy to extend.

5.6 Database Design



Entity-Relationship diagram

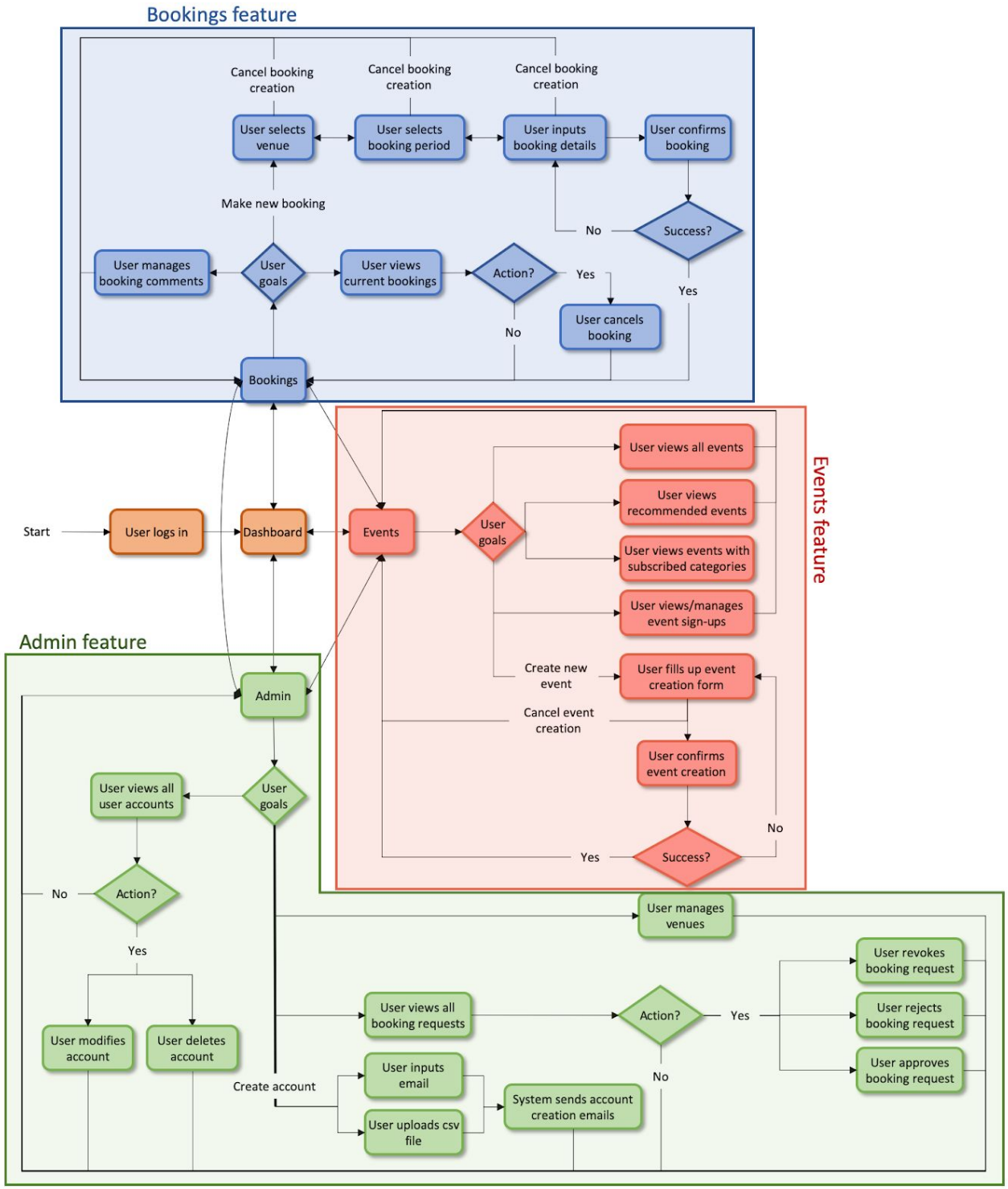
A relational database is selected for this app as our models are easily representable by relationships and we can maximise the use of these relationships to gain meaningful

insights by joining several tables together. This allows us to perform powerful and fast queries based on relations with involved entities. E.g. finding which categories of events are the most popular (e.g. top 3 highest sign-ups) or finding which time period has the most number of venue bookings. These insights could then be used for a future extension of the project which will be discussed in section 7.1.3.

With relational databases, we can also ensure that the state of the entire system is consistent at any point in time and using PostgreSQL allows us to easily export and back up our data. This is crucial as data consistency and integrity is key for our web app.

Finally, to minimise data redundancy, our team also attempted to normalize our tables during our database designing process.

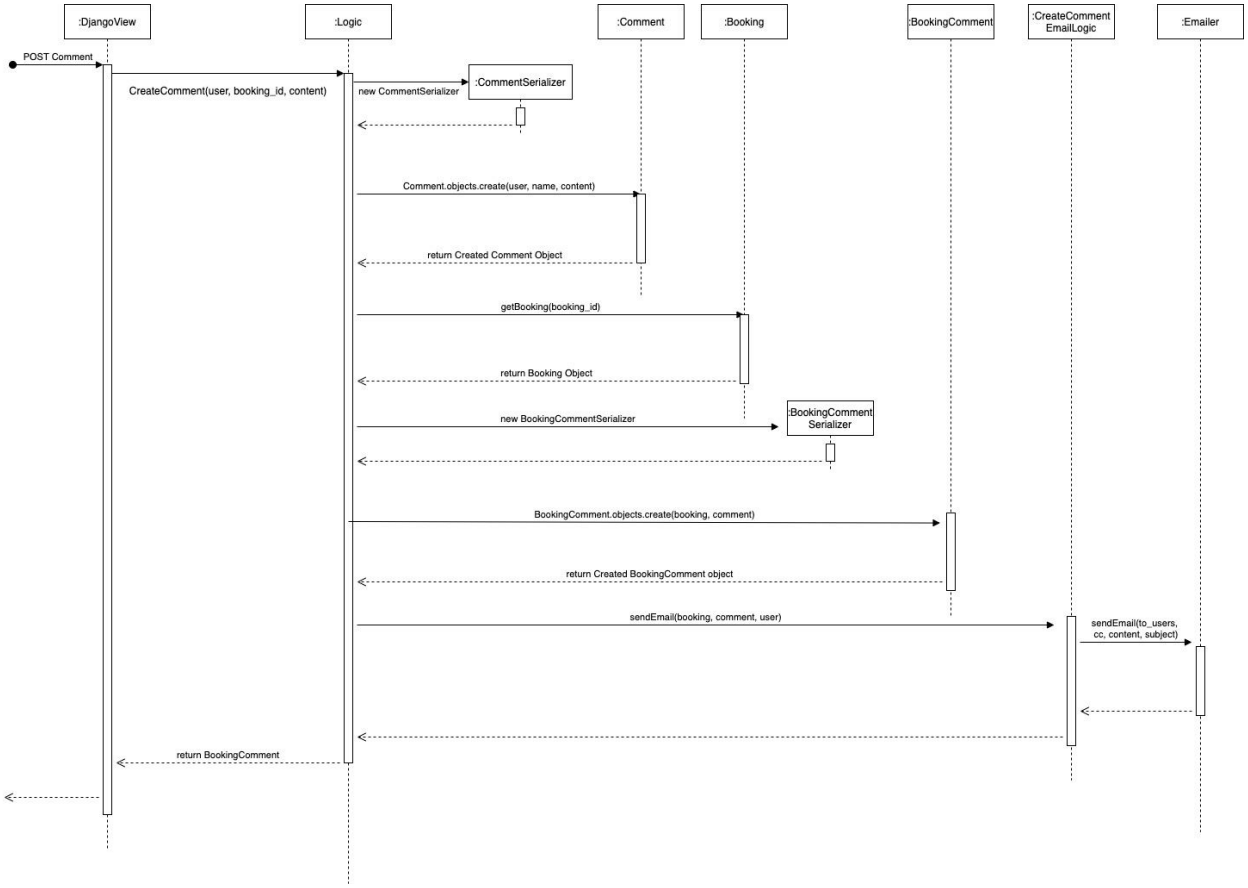
5.7 User Flow



User flow diagram

5.8 Application Flow

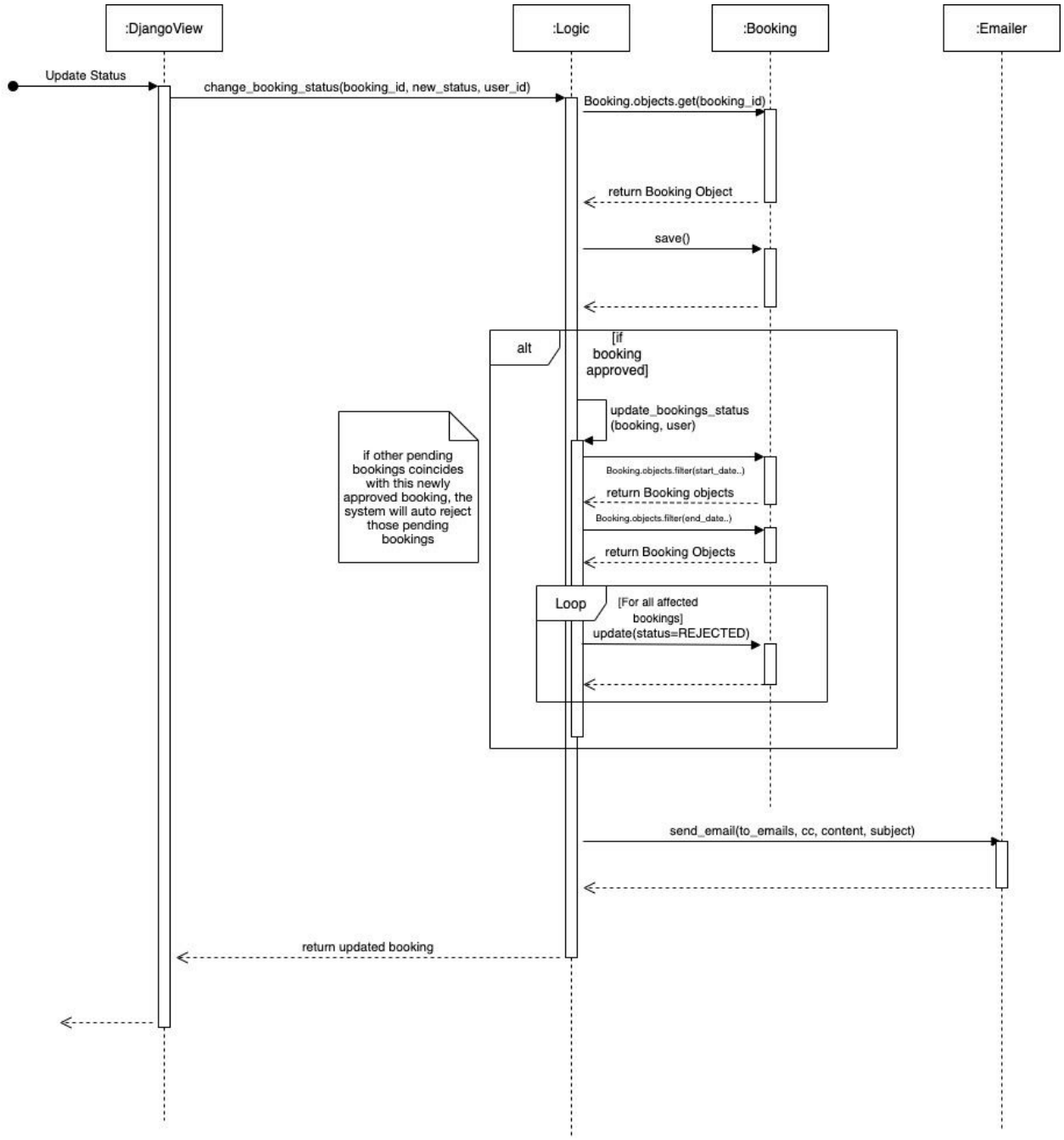
5.8.1 Booking Comment Creation



Sequence diagram for creating a booking comment

The above sequence diagram illustrates the application flow when a user creates a new comment for a particular booking. Firstly, the system creates a new Comment object to encapsulate the user's response. Next, it retrieves the associated Booking object and forms a relation between the Booking and Comment object in the BookingComment table. Finally, a notification email will be sent to the relevant parties which can either be the other users who have commented or the organisation listeners. This ensures that our users are kept notified of the latest comments so that they can respond timely.

5.8.2 Update Booking Status



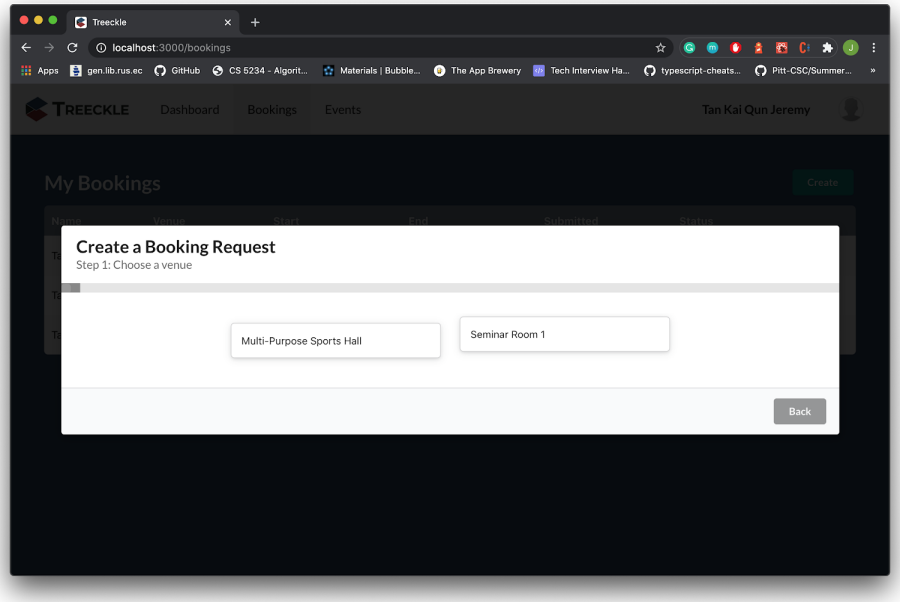
Sequence diagram for updating status of a booking request

The sequence diagram depicts the event when an admin updates the status of a booking to approved or rejected or whenever a resident cancels his or her booking. Firstly, when a user sends a request to update the status of a booking, the logic component will grab the relevant booking. Then it will update its status and save it into the database. If a booking is

approved, then there are additional steps required which is to reject all those other pending bookings that clashes with the current approved booking. Subsequently, once all the statuses of the affected bookings are updated, an email receipt that indicates the change in the booking status will be sent to the relevant parties.

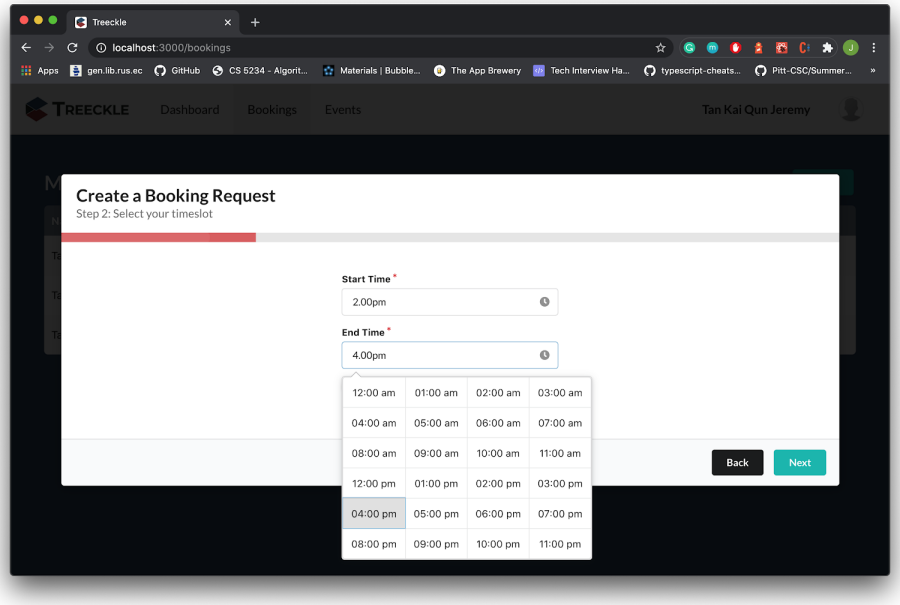
5.8.3 Booking Creation UI

This subsection describes the flow to create a booking request.



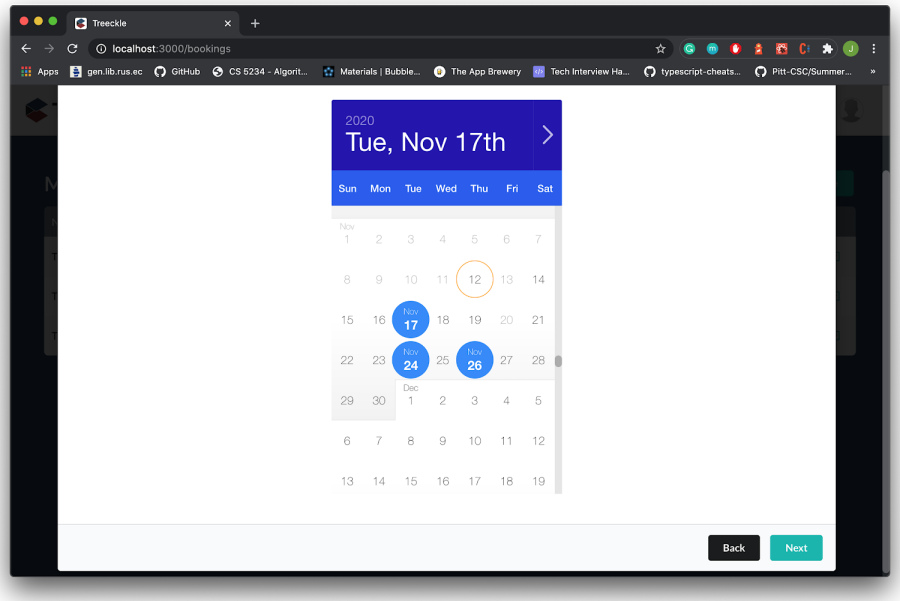
Venue selection

Firstly, the user is required to select a venue to book. From the above image, the user can either select Multi-Purpose Sports Hall or Seminar Room 1.

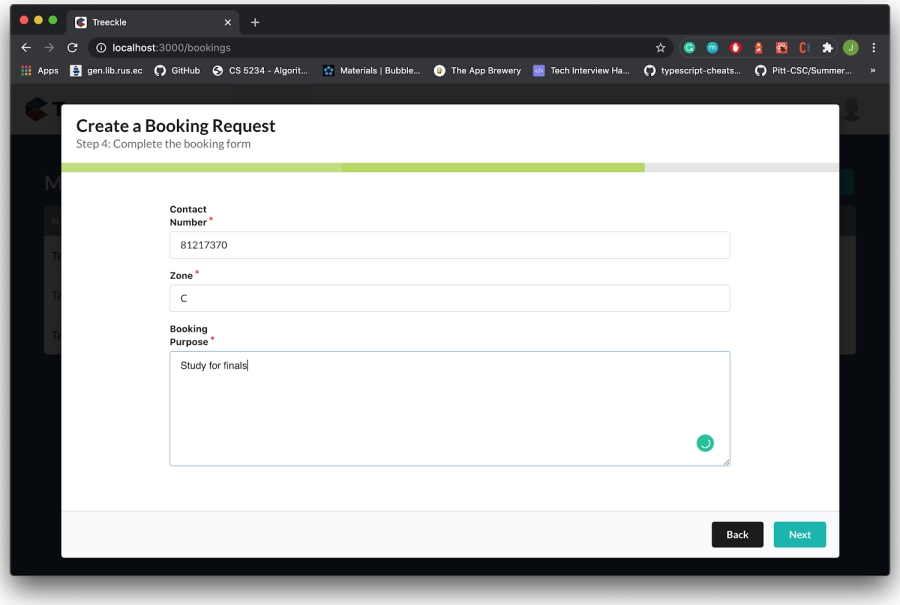


Selection of start/end time

Next, the user indicates the timing as well as the date(s) for the requested booking. It is worthy to note that the user can select multiple dates for the selected time period.

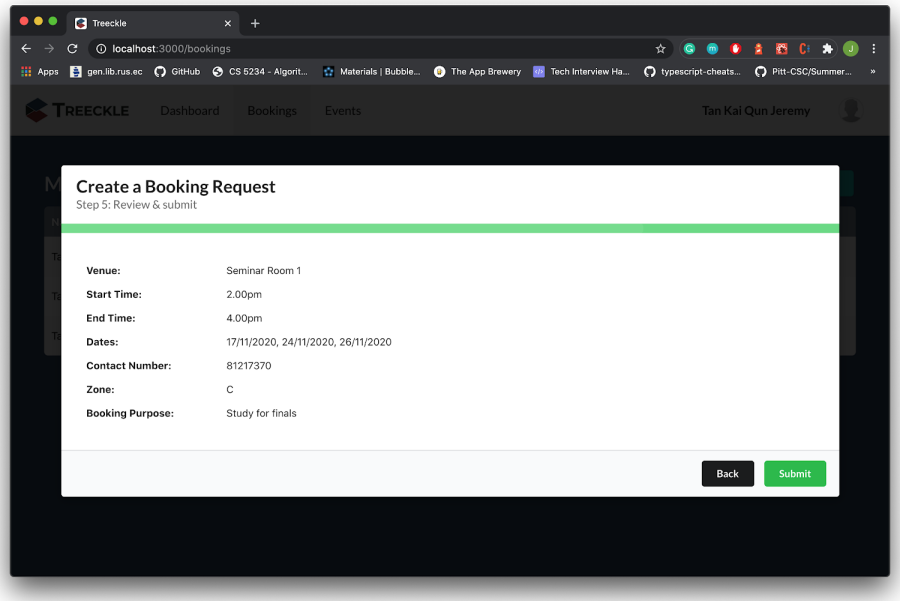


Selection of date(s)



Additional information for booking

After selecting the date/time, there may be other additional information needed from the user. From the above image, the contact number, zoning details and purpose are required.

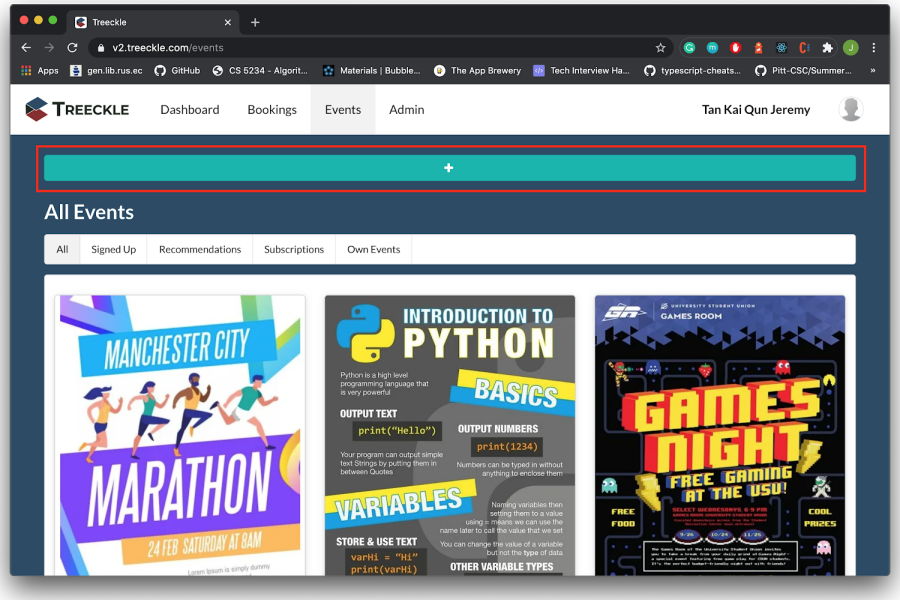


Confirmation of booking

Finally, the user will review their booking request before submitting. Once satisfied, he/she can click on submit to create a new booking request. Upon successful submission, an email receipt containing the booking details will be sent to both the requestor and the organisation listeners. This step-by-step booking creation UI/UX is designed to smoothly guide the user to complete the entire venue booking flow.

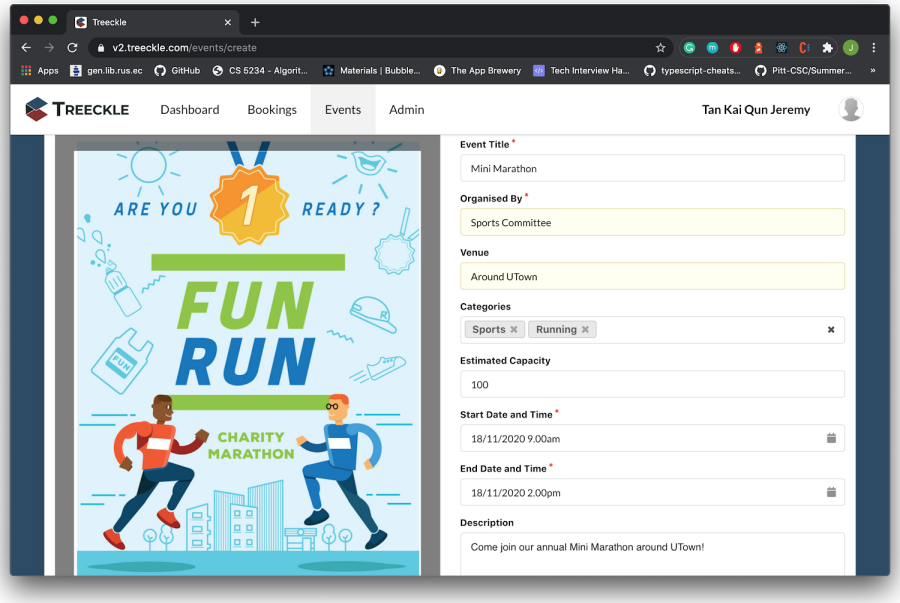
5.8.4 Event Creation UI

This subsection describes the flow to create a new event.



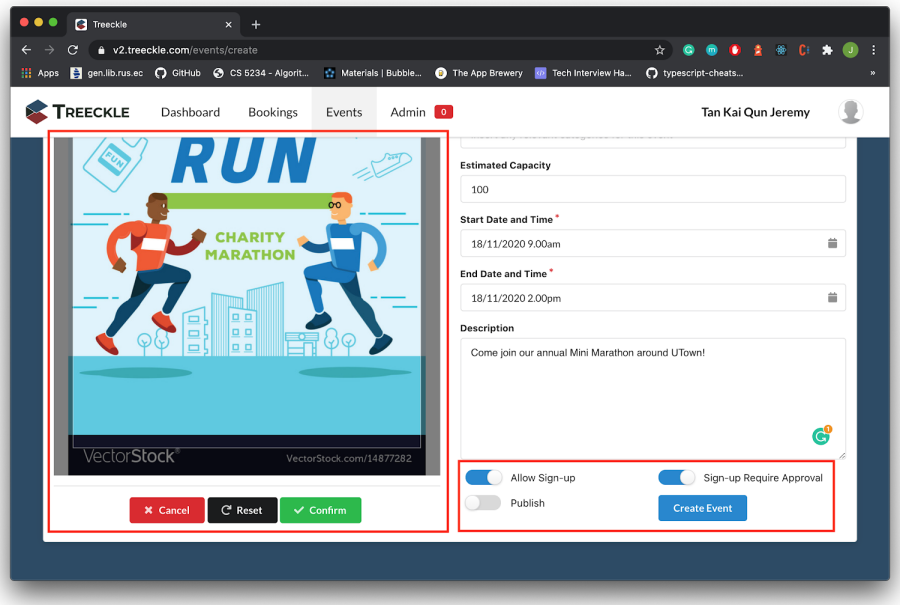
Events Page

Firstly, the event creator will click on the add button located at the top of the events page to enter into the event creation page.



Event Creation Form

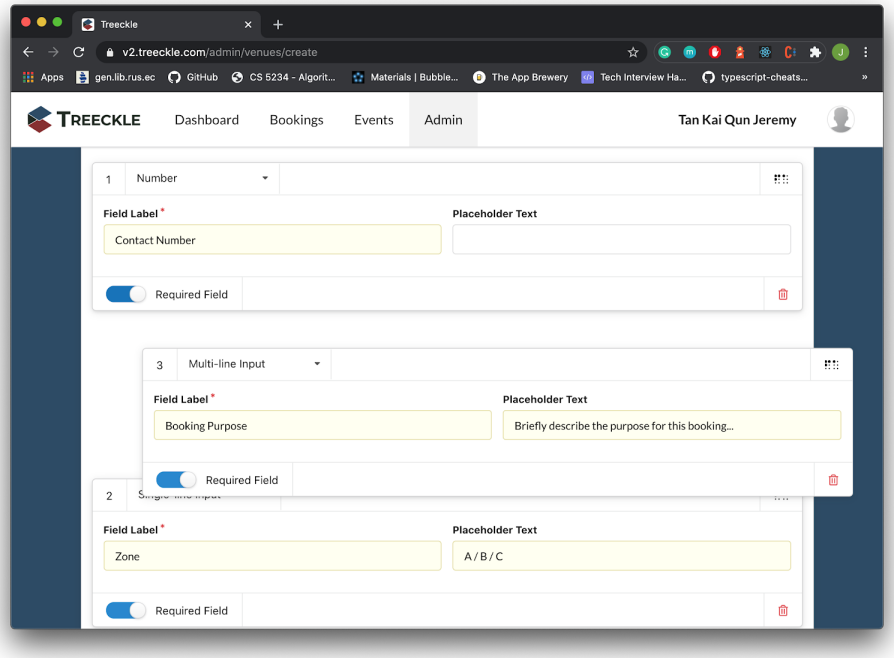
Next, he/she will specify the relevant details for the new event. Other than inputting textual information, the creator can also optionally choose to upload an event poster that will be displayed together with the event information.



Different event customisations

Since poster images must be A4 size, a cropper will be provided for the user to crop the uploaded image to the fixed size. Additionally, the creator can also choose if he/she wants the event to be sign up-able and can even choose if user sign-ups will require the creator's approval. Finally, the creator can decide to publish the event immediately or choose to create the event first and publish at a later time.

5.8.5 Venue Creation UI



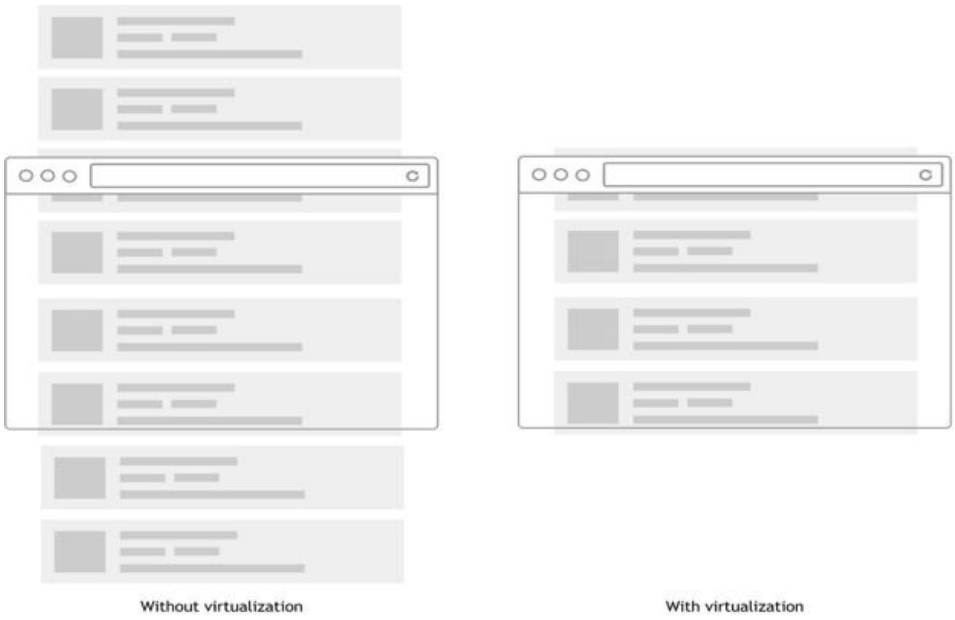
Highly customisable booking form fields

The key highlight of the venue creation UI is the ability for the admins to create highly customisable venue booking form fields for each of the venues. The motivation behind this feature is that different venues require users to provide different sets of information during booking. For instance, booking the sports hall requires users to state if they need to loan sports equipment while booking of the common kitchen requires users to state if halal utensils is required. During this Covid 19 period, the admins now require users to also state the zone they belong to. As such, our team realised that the field requirements of the booking forms changes between different venues and periods. Hence, our team took inspiration from google forms and decided to implement this customisable form builder that supports 4 different types of fields - Single-line input, multi-line input, number input and yes/no response.

6. Other Design Considerations

6.1 UI Performance

One of the key aspects of a scalable web app is that it is able to scale to handle and render a large amount of data without compromising the UX. As the number of users grows and the number of booking requests made increases, the tables would be required to render a large number of rows of data. This becomes a problem when the browser DOM is filled with too many elements and the browser/scrolling would start to lag, causing a decline in performance. This is where DOM virtualization (not to be confused with virtual DOM) comes into play and can vastly improve rendering performance.



Comparison between with and without DOM virtualization

The idea of DOM virtualization is actually quite simple and that is to only mount elements which are within the visible scrollable viewport. From the above diagram, we can see that without virtualization, all the elements which are out of the viewport are actually mounted onto the DOM even when the users cannot see them. This incurs extra wasted memory and mounting time, and hence when the tables are huge, the decline in performance becomes noticeable. On the other hand, with virtualization, we can see that

only viewable elements are mounted onto the DOM, thus UI performance becomes independent of the amount of data.

While the idea of DOM virtualization is simple, the actual implementation is non-trivial. As such, our team utilised a popular DOM virtualization package to achieve our desired outcome.

6.2 Data Privacy and Management

The nature of the application necessitates the need to deal with a certain amount of personal data, and we had to comply with the PDPA as well as residential college-specific rules in order to get this application deployed. Additionally, as Software Engineers working towards the greater good, we prevent malicious use and instead build defensively such that users can have the peace of mind that the application does not compromise data that they would not want others to see.

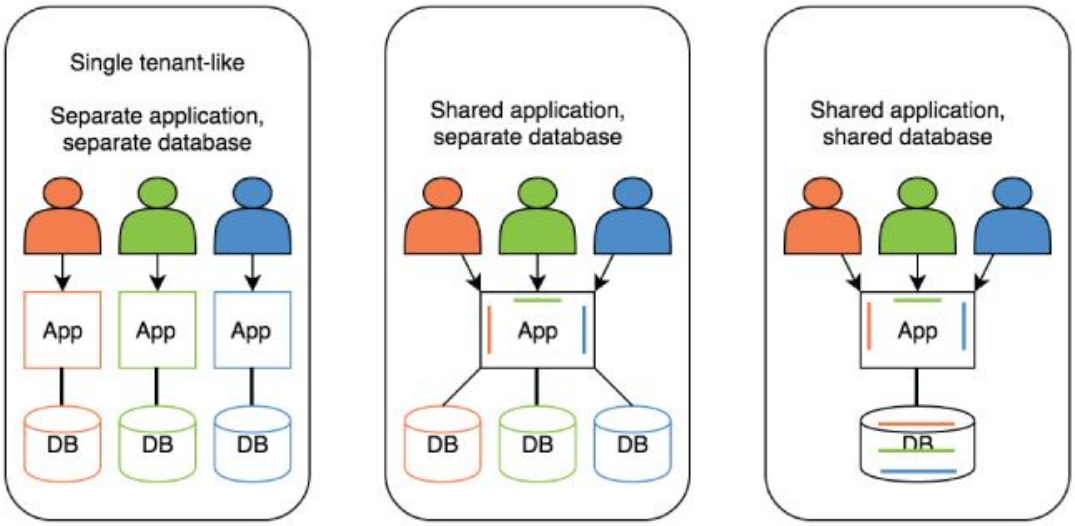
To protect resources, we divide users by roles and provide them access to a set of API calls on a need to know basis. To facilitate this in a scalable manner, we have an access control matrix embedded in our backend, which is checked against using the decrypted JWT provided as part of every protected API call to validate the user's ability to access the relevant information. The API calls are logically tiered and provide incremental amounts of information, with typically the most amount of information revealed to the owners of the resources themselves. E.g. a resident will have access to the fields within their own booking request but no access to anyone else's except the presence or absence of a room booking at the specified times, as they will need to know about booking conflicts before making one.

A concern with such a system is that if the application goes down, a lot of data will be lost and users will be left in the dark. Treeckle 2.0 provides data redundancy with our integration of status update emails, which sends an email for every request made and updated to the relevant parties. It also plays a secondary role of a reminder/notification system.

6.3 Multi-Tenancy

As this app was built simultaneously while working out the details of how ownership of the application will be established and distributed, we had to build in a manner that makes no unnecessary assumptions about how the application will be deployed. E.g. with each

RC hosting its own instance, or all RCs treating it as a common platform, contributing towards its development and upkeep.



Comparison among the different types of deployed architecture

A multi-tenanted database (diagram on the right) is the most versatile option available. This is because it can 1) work within a central server with co-existence of multiple organisations, each having access only to their own data, 2) or effectively be run as a single-tenant database if the RCs decide to proceed with having their own instance of Treeckle 2.0 hosted by them.

7. Future Plans

7.1 Features

7.1.1 Announcement

Currently, with the exception of the embedded NUSMods timetable, the dashboard does not show any other useful information. This page can be better utilised with the announcement feature where upon login, the first thing users will see is the list of announcements posted by the admins. With this feature, the admins, CSC or other parties can easily post and share announcements on Treeckle 2.0. Additionally, we can also

integrate announcements with the existing emailing service so that whenever a new announcement is posted, an email will be sent to everyone in the organisation.

7.1.2 Forum

Another extension to this app is to build a forum page. Every RC will have their own forum page and its users will be able to create or comment on the forum posts. This forum feature can prove to be very useful as it is very versatile and can support a wide range of use-cases. Users can:

1. Create daily supper jio posts on the forum to ask around if other residents want to order supper. Those interested residents can then reply to the post with their choice of food as well as unit number. In this way, the minimum order amount can be easily reached and maximising the number of people ordering can help minimise the delivery cost per pax when shared equally among the residents.

2. Create posts on unofficial events/activities to encourage residents to participate. The current Events feature is used for official events such as college-wide events or CCA sessions. As such, it will be nice to have an unofficial platform for residents to promote their own hosted events/activities such as board games sessions at the lounge, night cycling events, jam sessions, etc.

3. Share and discuss almost anything under the sun. E.g. latest news, latest viral trends, fun facts, latest movies, etc.

7.1.3 Bookings and Events Statistics

Finally, with the type and projected amount of data (600+ residents per college) collected in this app, a bookings and events statistics viewer feature can be implemented for the admins. The admins will be able to discover interesting trends within the college such as which categories of events are popular among residents. With this information, the admins can plan more events of those categories to attract more residents to participate.

Additionally, having bookings statistics can help the admin to find out which venues and time slots have the most number of bookings. The admin can then feedback to the college office to perhaps open up more venues or available time slots during those popular time slots to cater to the demand.

7.2 Deployment

7.2.1 Deployment of Treeckle 2.0 in CAPT (starting from AY20/21 Semester 2)

Jeremy has met up with CAPT's CSC to introduce, demo and gather feedback for Treeckle 2.0. The CSC said that they like this app and have given the green light to start using this app from next semester. In the meantime, Jeremy will continue to liaise with the CSC to work on administrative tasks such as briefing and producing a user guide for the residents.

7.2.2 Meeting with Tembusu College and RC4 CSCs

In the middle of this semester, the CSCs of Tembusu College and RC4 heard of this project and they have expressed interest to implement this app in their own colleges. As such, over the winter break, we will be meeting up with the respective parties to discuss more in detail regarding how to go about making this app available to all 3 RCs. E.g. how to go about sharing the domain fee, mailgun subscription fee, server fee, etc.

Appendix

A. Glossary

Term	Definition
CSC	College Students Committee. The student committee in charge of all college affairs.
Facility	Synonymous with venue.
Organisation	An entity that defines the group a user belongs to. E.g. each of the residential colleges is an organisation by itself.
Organisation Listeners	They are users that will be notified through email whenever a booking or comment is created or updated.
Role	The permission level of a user.

B. Use Cases

UC1	User login using NUSNET ID
Actor	All users
MSS	1. User navigates to login page. 2. User attempts to login. 3. User is directed to NUS OpenId site. 4. User enters username and password. 5. User is redirected back to Treeckle. 6. Treeckle logs user in using the redirection data given by OpenId authentication. Use case ends.
Extensions	3a. User is already logged in. 3a1. User is redirected back to Treeckle. 3a2. Treeckle logs user in. Use case ends. 4a. User enters wrong username or password. 4a1. NUS OpenId informs user username or password is incorrect. Use case resumes from step 4.

UC2	User login using Google email
Actor	All users

MSS	1. User navigates to login page. 2. User attempts to login using Google account. 3. User is prompted for authentication. 4. User enters username and password. 5. User is redirected back to Treeckle. 6. Treeckle logs user in using the redirection data given by OpenId authentication. Use case ends.
Extensions	3a. User is already logged in. 3a1. User is redirected back to Treeckle. 3a2. Treeckle logs user in. Use case ends. 4a. User enters wrong username or password. 4a1. NUS OpenId informs user username or password is incorrect. Use case resumes from step 4.

UC3	View users
Actor	Admin
Precondition	Admin is logged in.
MSS	1. Admin navigates to “Users” page. 2. Treeckle displays all users, except for the admin himself. Use case ends.

UC4	Mass invitation of users
Actor	Admin
Precondition	Admin is logged in.
MSS	1. Admin views users (UC3). 2. Admin requests for user invitation. 3. Treeckle requests for the emails to be submitted. 4. Admin uploads the csv. 5. Treeckle receives the file, parses it for the emails and displays the parsed emails to the admin. 6. Admin confirms submission. 7. Treeckle invites all users by sending an invitation mail to all specified emails Use case ends.
Extensions	5a. Treeckle receives an invalid file. 5a1. Treeckle alerts the admin of this error and prompts the admin to re-upload the correct csv file. Use case resumes from step 4.

UC5	Invitation of users
Actor	Admin
Precondition	Admin is logged in.
MSS	1. Admin views users (UC3).

	2. Admin requests for user invitation. 3. Treeckle requests for the emails to be submitted. 4. Admin types, enters an email, and submits. 5. Treeckle invites the users by sending an invitation mail to the email. Use case ends.

UC6	Change role of user
Actor	Admin
Precondition	Admin is logged in.
MSS	1. Admin views users (UC3). 2. Admin requests to change the role of a user. 3. Treeckle prompts for what role it should be changed to. 4. Admin chooses the role to change to. 5. Treeckle changes the role and updates the change in the user listing. Use case ends.

UC7	View Availability of Venue
Actor	Resident
Precondition	User is logged in.
MSS	1. Resident navigates to the venue booking page. 2. Treeckle displays all venues. 3. Resident can choose any of the venues he is interested in to view its availability.

	4. Treeckle displays the availability of the selected facility in a calendar. Use case ends.

UC8	Create venues booking form
Actor	Admin
Precondition	Admin is logged in.
MSS	1. Admin views availabilities of venues (UC7). 2. Admin requests to create a new facility. 3. Treeckle prompts admin to provide the basic details of the facility. 4. Admin inputs the basic details and submits. 5. Treeckle prompts admin for the details (fieldname, type of field, placeholder) of a field to be added to the form. 6. Admin inputs the details of this form. Step 5-6 is repeated until the admin has no more fields to add. 7. Admin finalizes the form creation. 8. Treeckle displays the consolidated form to be created and requests for confirmation. 9. Admin confirms creation. 10. Treeckle creates the form and the new facility and form are displayed in the venues availability listing. Use case ends.
Extensions	*a. Admin may abort form creation at any stage. *a1. Treeckle requests for confirmation to abort. *a2. Admin confirms for abort. Use case ends.

	4a. Treeckle encounters an error in the basic details of the facility. 4a1. Treeckle alerts admin of this error. Use case resumes from step 3.

UC9	Remove an existing facility
Actor	Admin
Precondition	Admin is logged in.
MSS	1. Admin views availabilities of venues (UC7). 2. Admin requests to delete an existing facility 3. Treeckle requests for confirmation to delete. 4. Admin confirms deletion. 5. Treeckle deletes the facility and the facility is no longer displayed in the venue listing. Use case ends.

UC10	Edit an existing venues booking form
Actor	Admin
Precondition	Admin is logged in.
MSS	1. Admin views availabilities of venues (UC7). 2. Admin requests to edit an existing venues booking form. 3. Treeckle displays the basic details of the facility as well as all fields of the form and their settings.

	4. Admin edits the current fields or facility details, or add new fields to the form. 5. Admin finalizes the changes. 6. Treeckle requests for confirmation to edit. 7. Admin confirms edit. 8. Treeckle updates the form and the updated fields are displayed in the form. Use case ends.
Extensions	4a. Treeckle encounters an error while validating the changes. 4a1. Treeckle alerts admin of this error. Use case resumes from step 3.

UC11	View all bookings requests
Actor	User
Precondition	Admin is logged in.
MSS	1. Admin navigates to "Bookings" from the main landing page/dashboard 2. Treeckle displays all booking requests made. 3. Admin views all booking requests. Use case ends.
Extensions	2a. If no booking requests are made, Treeckle displays a message informing the admin that there are no booking requests made. Use case ends. 3b. Admin can select a booking request to view more information or

	comments about it. 3b1. Treeckle displays all information about the request. Use case ends.

UC12	Venues Booking
Actor	User
Precondition	User is logged in.
MSS	1. User views the availability of the venues (UC7). 2. User chooses the facility he wishes to book. 3. Treeckle displays a calendar for the user to choose the booking period and asks whether the user is attempting mass booking. 4. User chooses the start time, end time and date that he wishes to book. 5. User fills up additional questions provided by the booking form. 6. User submits the form. 7. Treeckle displays the booking data to be submitted and requests for confirmation to submit. 8. User confirms submission. 9. Treeckle informs the user that the form is submitted successfully. Treeckle sends an automated email to the user and admins about this booking request. Use case ends.
Extensions	4a. User may opt for mass booking. 4a1. User chooses the start time and end time that he wishes to book. 4a2. User then selects the dates he wishes to book.

	Use case resumes from step 5. 7a. Treeckle informs user that the form is incorrectly filled. Use case resumes from step 5. 7b. Treeckle informs user that the time period chosen is unavailable. Use case resumes from step 4.

UC13	View resident's bookings
Actor	Resident
Precondition	User is logged in.
MSS	1. Resident navigates to "My Bookings" from the main landing page/dashboard 2. Treeckle displays all bookings made by the resident himself. 3. Resident views all his bookings. Use case ends.
Extensions	2a. If resident has no bookings made, Treeckle displays a message informing the user that there are no bookings made. Use case ends.

UC14	Cancel bookings
Actor	User

Precondition	User is logged in.
MSS	1. User views his bookings (UC13). 2. User requests to cancel any one of his bookings. 3. Treeckle requests for confirmation. 4. Treeckle informs user that the booking is successfully cancelled. Treeckle sends an automated email to the booker and admins. Use case ends.

UC15	Add comments to a booking request
Actor	Admin
Precondition	Admin is logged in.
MSS	1. Admin views all bookings requests made (UC11). 2. Admin requests to add comments to the booking request. 3. Treeckle prompts the admin for the comment. 4. Admin inputs the comment and submits. 5. Treeckle adds the comment to the booking requests, which can be viewed by the resident who made the request. Treacle sends an automated email to the booker and admin about this comment. Steps 2-5 are repeated until the admin has no more comments to make. Use case ends.

UC16	Add comments to own booking request
Actor	User

Precondition	User is logged in.
MSS	1. User views the booking requests made by him(UC13). 2. User requests to add comments to the booking request. 3. Treeckle prompts the user for the comment. 4. User inputs the comment and submits. 5. Treeckle adds the comment to the booking requests, which can be viewed by the resident who made the request. Treacle sends an automated email to the booker and admin about this comment. Steps 2-5 are repeated until the user has no more comments to make. Use case ends.

UC17	Change status of a booking request
Actor	Admin
Precondition	Admin is logged in.
MSS	1. Admin views all bookings requests made (UC11). 2. Admin changes the status of a booking request. 3. Treeckle displays the new status of the booking request and sends an automated email notification to the admin, booker and the admins. Use case ends.

UC18	View college events

Actor	User
Precondition	User is logged in.
MSS	1. User navigates to “College Events” from the main landing page/dashboard. 2. Treeckle displays all relevant events. 3. User views these events. Use case ends.
Extensions	1a. User may navigate to the “Recommended” tab that displays all events recommended to the user. Use case resumes from step 2. 1b. User may navigate to the “Subscriptions” tab that displays all events subscribed by the user. Use case resumes from step 2. 1c. User may navigate to the “My events” tab that displays all events created by the user (organiser). Use case resumes from step 2. 2a. If no college events are created, Treeckle displays an empty events listing message. Use case ends. 3a. User can click on an event to view more information about the event. 3a1. Treeckle displays all information related to the clicked event. Use case ends.

UC19	Subscribe to event categories
Actor	User
Precondition	User is logged in.
MSS	1. User navigates to “College Events” from the main landing page/dashboard. 2. User navigates to “Subscriptions” tab . 3. User selects a category he/she is interested in and subscribes to events matching it. 4. Treeckle registers this subscription for the user and displays events that match this category . Use case ends.
Extensions	3a. If the category is already subscribed, selecting the category would unsubscribe the user from it. Use case ends.

UC20	Sign up for college events
Actor	User
Precondition	User is logged in.
MSS	1. User views college events (UC18). 2. User selects and signs up for an event he is interested in. 3. Treeckle requests for sign up confirmation. 4. User confirms sign up. 5. Treeckle successfully registers the resident for the event and

	displays a success message. Use case ends.
Extensions	2a. Treeckle may display additional cautionary message/reminders regarding the event when the resident signs up for it. Use case resumes from step 3.

UC21	Create and publish new events
Actor	Organiser
Precondition	User is logged in.
MSS	1. Organiser navigates to "College Events" from the main landing page/dashboard. 2. Organiser requests to create a new event. 3. Treeckle displays an event creation form, requesting for details about the event. 4. Organiser provides these details in the form. 5. Organiser submits the form. 6. Treeckle displays the details to be submitted and requests for confirmation to submit the form. 7. Organiser confirms submission. 8. Treeckle creates the new event with the given details and displays this event in the events listing. Use case ends.
Extensions	*a. Organiser may abort event creation. *a1. Treeckle requests for confirmation to abort.

	*a2. Organiser confirms to abort. Use case ends. 6a. Treeckle detects mandatory fields that are unfilled. 6a1. Treeckle alerts the organiser to these fields and prompts the organiser to fill them in. Use case resumes from step 4.

UC22	Update a created event
Actor	Organiser
Precondition	User is logged in.
MSS	1. Organiser views "My Events" (UC18). 2. Organiser selects the event he wishes to update. 3. Treeckle displays all details about the event. 4. Organiser requests to update the event details. 5. Treeckle displays a form populated with editable fields that are filled with the current details about the event. 6. Organiser updates the details in the form. 7. Organiser submits the form. 8. Treeckle displays the details to be submitted and requests for confirmation to submit the form. 9. Organiser confirms submission. 10. Treeckle updates the event with the changed details and displays this updated event in the events listing. Use case ends.

Extensions	*a. Organiser may abort event creation. *a1. Treeckle requests for confirmation to abort. *a2. Organiser confirms to abort. Use case ends. 8a. Treeckle detects mandatory fields that are unfilled. 8a1. Treeckle alerts the organiser to these fields and prompts the organiser to fill them in. Use case resumes from step 6.

UC23	Remove a created event
Actor	Organiser
Precondition	User is logged in.
MSS	1. Organiser views "My Events" (UC18). 2. Organiser selects the event he wishes to remove. 3. Treeckle requests for confirmation to delete. 4. Organiser confirms submission. 5. Treeckle deletes the event and removes this event from the events listing. Use case ends.
Extension	4a. Organiser cancels confirmation. Use case ends.

UC24	View event sign ups and attendance
Actor	Organiser
Precondition	User is logged in.
MSS	1. Organiser views "My Events" (UC18). 2. Organiser selects the event he wishes to approve sign ups for. 3. Treeckle displays all details about the event. 4. Organiser requests to view all sign ups for the event. 5. Treeckle displays all sign ups and attendance for the event, sorted according to their sign up status and attendance. Use case ends.
Extension	5a. If there are no sign ups for the event, Treeckle displays a message informing the user that there are no sign ups. Use case ends.

UC25	Approve or disapprove event sign ups
Actor	Organiser
Precondition	User is logged in.
MSS	1. Organiser views event signups and attendance (UC24). 2. Organiser approves or disapproves a sign up application. Step 2 is repeated until the organiser finishes all approvals. Use case ends.

UC26	Log attendance for events
Actor	Resident and Organiser
Precondition	Organiser is logged in.
MSS	1. Organiser views event signups and attendance (UC24). 2. Organiser requests to generate the QR code for event participants to log their attendance. 3. Treeckle displays the QR code to the organiser. 4. Organiser physically shows this QR code to the participating residents. 5. Resident scans the QR code. 6. Treeckle logs the resident's attendance and displays this in the event's attendance details. 7. Treeckle informs the resident that the attendance is logged successfully. Steps 5-7 are repeated until all participating residents have logged their attendance. Use case ends.
Extensions	6a. If resident is not logged in, Treeckle will redirect the user to the login page and prompt the user to login (UC1 or UC2). Use case resumes from step 6.

UC27	View college settings
Actor	Admin
Precondition	Admin is logged in.

MSS	1. Admin navigates to “Settings” from the main landing page/dashboard. 2. Treeckle displays all current settings of the college. Use case ends.

UC28	Update external emails
Actor	Admin
Precondition	Admin is logged in.
MSS	1. Admin views college settings (UC27). 2. Admin inputs an email that will be sent email notifications upon booking-related changes. 3. Admin requests to save the current settings. 4. Treeckle saves the current settings and the email is displayed as the CC email. Use case ends.
Extension	2a. Treeckle validates the email, and finds it invalid. 2a1. Treeckle alerts the admin that the email is invalid. Use case resumes from step 2.