

CS3216 Software Product Engineering for Digital Markets

AY 2022/23 Semester 1

Assignment 3 G07



Bringing smiles to your cheeks, one clean toilet at a time

<https://clean-cheeks.vercel.app/>

G07 Team Info:

Keane Chan Jun Yu	A0205678W
Wu Peirong	A0199926E
Tan Su Yin	A0205369B
Terence Ho Wei Yang	A0196511E

Table of Contents

Table of Contents	2
Phase 1: Design	3
Milestone 0: Problem	3
Milestone 1: Ideation	3
Milestone 2: Target Users	4
Milestone 3: ER Diagram	8
Milestone 4: Alternatives to REST	9
Milestone 5: API Design	10
Uniform Interface	10
Client-Server	11
Layered Architecture	11
Statelessness	11
Cacheable	11
Code on Demand	12
Good Practices	12
Phase 2: API Server	13
Milestone 6: SQL Queries	13
Query 1: Toilets with User Preferences	13
Query 2: Fetch ranking of toilets	14
Query 3: Deleting User Preference for Toilet	15
Phase 3: Mobile Client	16
Milestone 7: Icon/Splash Screen	16
Milestone 8: CSS	17
UI Design	17
CSS Methodologies	20
Milestone 9: HTTPS	21
Milestone 10: Offline	21
Milestone 11: Authentication	23
Milestone 12: Client Framework	26
Mobile Site Design Principles	27
Milestone 13: User Experience	29
Searching/Filtering Toilets Flow	30
Favoriting/Blacklisting Toilets Flow	31
Rating Toilets Flow	32
Milestone 14: Analytics	33

Milestone 15: Google Lighthouse	34
Phase 4: Coolness Factor	35
Milestone 16 (Optional): Social Integration	35
Milestone 17 (Optional): Geolocation	35
Appendix	35

Phase 1: Design

Milestone 0: Problem

Students do not have an easy way to find out where the nearest toilets on campus are, or view the cleanliness of the toilets. They also have no easy way of finding toilets with the amenities they want (e.g. showers/bidets).

In our preliminary survey, our respondents indicated that they find it hard to locate the toilets in new, unfamiliar areas of NUS. There are also rarely visible signages or directions to assist them to find the toilets. Hence, 92% of our respondents felt that it was helpful to have an application that can help them identify where the toilets are in NUS.

In addition, they also felt that cleanliness was a very important factor for them when using the toilets. All the respondents felt that it would be useful to know the cleanliness of the toilets around them, so that they can choose the cleanest toilets to go to. In addition, they would also like additional information on the toilets (e.g. Toilet capacity such as how many squatting or seating toilets, the facilities in the toilet such as showers, bidets etc.).

Milestone 1: Ideation

Our application, Clean Cheeks, allows users to find the nearest toilets on campus and easily view their amenities (e.g. showers/bidets) and cleanliness levels (clean VS dirty). Users can search for locations within NUS and the app will display clusters of toilets that are near to that location on the map. Users can also filter for toilets that suit their needs (handicap, facilities such as showers or water-coolers).

In order to ensure that the information is updated over time, we added a rating feature to source for community feedback regarding the toilet cleanliness. To prevent spam, we added

a 30-minute cooldown between rating for toilets. In addition, we added customisation features such as favoriting and blacklisting toilets to allow users to track which toilets they prefer or wish to avoid. For rating and customisation features to work, the user has to log in with their Google account.

We find that implementing this application is done best via mobile cloud computing as the application is designed to assist users in quickly locating toilets in a hassle-free fashion. As most of our target users carry a smartphone around and internet connection is widely available in NUS, locating toilets using their mobile phones becomes a natural solution to our identified problem.

Furthermore, most users would be moving about while looking for their toilets, hence using their mobile phones makes the process of searching for a clean toilet convenient and the information is easily accessible at their fingertips (literally).

Milestone 2: Target Users

Our target/primary users are mainly members of the NUS community (students, faculty staff etc.), with the general public as our secondary users. The primary users use the toilets in NUS on a daily basis but are not satisfied with their experience in the toilets. One of the key issues is due to the cleanliness of the toilets and they have no way of knowing beforehand how clean the toilets are.

Additionally, as the users move from one faculty to another, they might not be familiar with their immediate surroundings. Locating toilets hence becomes difficult and especially urgent if they are feeling urgent (if you know what I mean). Hence, having a quick-and-easy solution at just one click away makes it a valuable solution to the NUS community. In order to bring the application to our target users, we have designed the following marketing strategy.

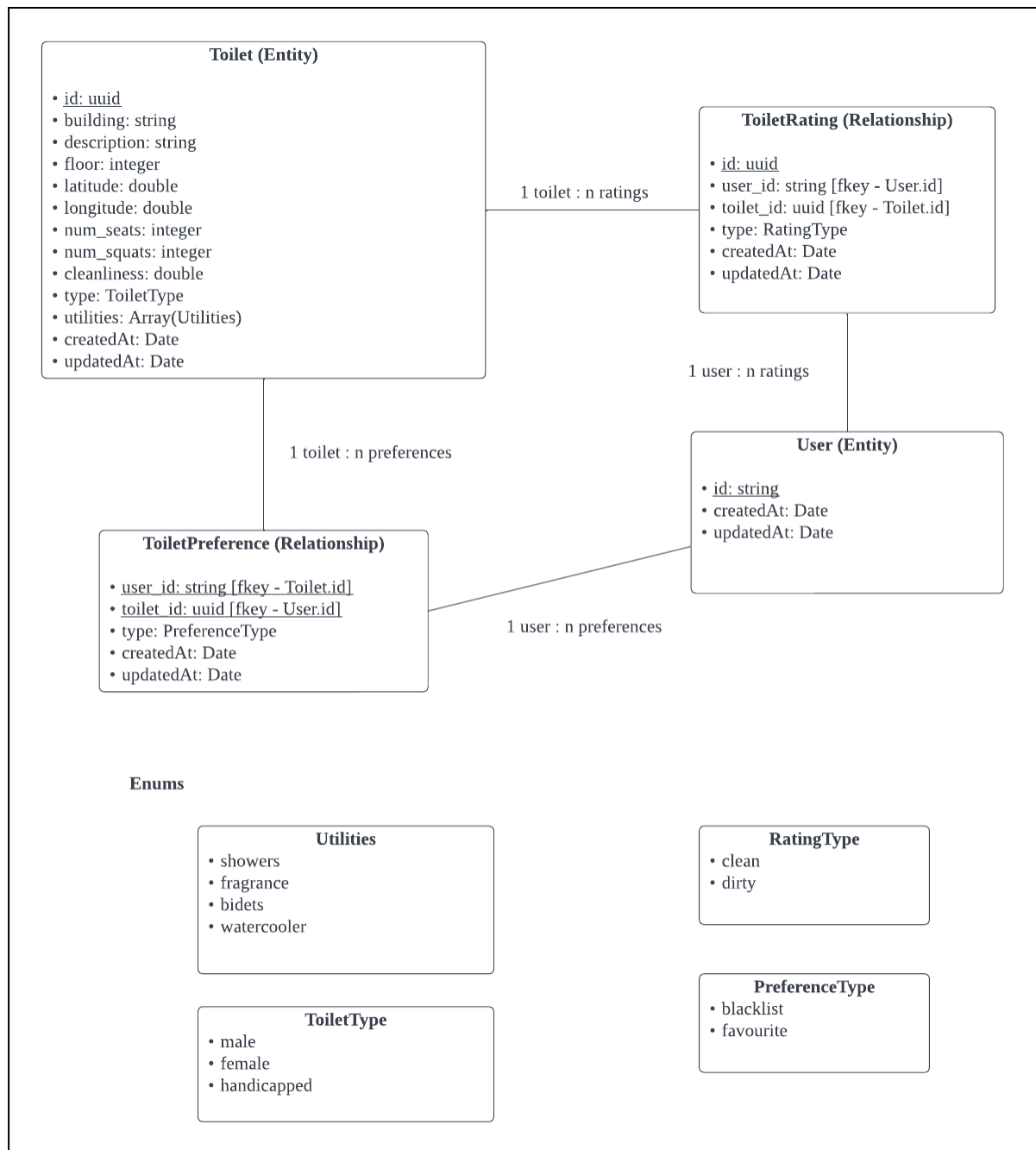
Bringing awareness to Clean Cheeks	
Stakeholders involved	Members of NUS Community, Student-life groups (e.g. NUSSU), NUS communities
Action Plan	<p>Create a series of social media posts that promote the use of Clean Cheeks, showing the idea behind the app and how to use it.</p> <p>These can be in the form of short videos (TikTok, YouTube</p>

	<p>shorts etc.) or images (Instagram stories or posts).</p> <ul style="list-style-type: none"> • We can introduce the main use cases in a satirical or comical nature (e.g. urgently finding a toilet but none are clean or available) <p>In addition, we can reach out to student-life groups or NUS communities to assist in promoting the app. If they are willing, they can promote our app in their newsletters/emails, or showcase our application during events (e.g. Computing Club Welfare events).</p>
Rationale	<p>The ubiquity of social media makes it easier for us to reach our target audience, as many of the NUS community uses some form of social media.</p> <p>By introducing the application and injecting some entertainment to them serves to invoke interest in our target communities. In addition, the posts serve to engage the audience and teach them how to onboard our application, making it easy to understand and in turn assist users in the retention of the content we deliver.</p> <p>By working with the existing student-life/NUS groups, we can tap on their network to reach out to the student body that we might be unable to do so on our own. In turn, our application can serve to improve the welfare of the NUS community, which is in line with the goals of these groups.</p>
Gathering feedback and iterating the product	
Stakeholders involved	Primary and secondary users
Action Plan	<p>Gather feedback from the users that used Clean Cheeks.</p> <ul style="list-style-type: none"> • Adding a link or embedding a feedback form in Clean Cheeks for users to give us feedback anytime • Interviewing users for feedback <p>We can also conduct outreach programmes, such as hosting an AMA (ask me anything) on platforms such as Reddit or Instagram to gather feedback from the users on how our app can be further improved, or to clarify any doubts users have</p>

	about our app.
Rationale	<p>As users start to use our product, we can gather evidence on the effectiveness of our product. In addition, we can also identify pain points that help us to improve how the application works. In the future, we can improve on the features to better serve the community.</p> <ul style="list-style-type: none"> • We can also derive feature requests, potential bugs and user experience improvements via these feedback <p>In addition, the users might also provide valuable data that the developer team might miss out. For example, there might be new toilets being constructed and old toilets being removed and these might not be captured by the development team. The users might be able to provide valuable feedback for us to improve the app and make the information more accurate.</p>
Data analytics and growth monitoring	
Stakeholders involved	Primary and secondary users
Action Plan	<p>As the users make use of the product, we collect analytics via Google Analytics to generate insights about our application. At the same time, we ensure that we do not infringe on our users' privacy.</p> <p>We collect information about what users click on in the application in order to identify what features are more popular and which are less useful to them. In turn, they help us decide what features to iterate on or remove entirely.</p>
Rationale	<p>By collecting feedback and data, we can derive further insights on what works or does not work for users and further improve the product.</p> <p>We are transparent about the type of information collected in order to build trust and confidence with our users about the product and the branding of the team.</p>
Community Collaboration	

Stakeholders involved	Primary and secondary users, NUS management
Action Plan	<p>Make the project open-source and allow the NUS community to contribute in the development and maintenance of the application.</p> <p>Work with NUS management to acquire even more data and insights towards the toilet information in NUS.</p>
Rationale	<p>By making the project open-source, there will be more developers to build features that are desired by the NUS community.</p> <p>By working with NUS management, we can provide more accurate data and information to the NUS community. At the same time, we can share insights to improve NUS infrastructure. For example, there might be a lot of demand for handicapped toilets in COM-3 but there might not be enough accessible toilets there. It might be an indicator for NUS to provide more handicap-accessible toilets in COM-3.</p>

Milestone 3: ER Diagram



Above is the list of the entities and relationships in our database. The toilet has uuid as its primary key, while the user has a string as the primary key, which refers to the unique google login ID of the user. The ToiletPreference table refers to the favourite and blacklisted toilets for each user, which has its primary key as the toilet_id and user_id. For the ToiletRating table, since each user is able to rate a toilet multiple times, we defined a unique id for each record instead. Here we could also have used `pkey(toilet_id, user_id, created_time)`, but it was not doable due to the constraints of sequelize ORM.

Milestone 4: Alternatives to REST

Let us compare gRPC to REST.

Pros of using gRPC compared to REST
Easier to handle gRPC calls on the server as there is a logical mapping between the request (some procedure with arguments supplied) and the function handler on the server.
REST requires the server to parse the incoming request and its parameters and call the right implementation functions.
gRPC allows for streaming communication between the client and server while REST follows a request-response model where every request is handled independently by the server.
gRPC serialises the payload using Protocol Buffers which greatly reduces their size compared to REST which usually uses JSON. Smaller size helps with transmission durations and network load especially during busy periods.

Cons of using gRPC compared to REST
Limited browser support as gRPC maps to HTTP/2 underneath and most browsers only offer full support for HTTP/1.1 while REST has comprehensive browser support.
gRPC uses Protocol Buffers which are not supported natively by browsers. A library and .proto definitions are usually required to decode the protobuf object which adds to the complexity of handling procedure calls on the client and server.
REST deals commonly with JSON which is natively supported by the browser and many programming language runtimes. Much easier developer experience to manipulate JSON objects on both the client and server.
Generating gRPC calls on both the client and server require special software installed.
REST API calls can be easily issued using cURL commands which comes preinstalled with most operating systems or typing a url path in a web browser.

REST will be more appropriate for our mobile PWA. The inability to construct and send gRPC calls natively on the mobile browser since it runs over HTTP/2.0 was a dealbreaker for us. Even though there are frontend libraries like **gRPC web (one by Google and one by Improbable)** that help to convert a gRPC call sent over HTTP/1.1 to the HTTP/2.0 format,

there are some limitations with using it that will affect the user experience of the app. One main advantage of gRPC over REST is that it offers us the ability to client-stream data from the server. However, due to [limitations](#) of the Browser HTTP primitives (fetch and XMLHttpRequest), the HTTP/2-based transports provided by **gRPC web** can only support unary and server-streaming requests. Attempts to invoke client-streaming will result in failure. This, coupled with the overhead of running a proxy server on the browser to convert our HTTP/1.1 calls to HTTP/2 and handle the communication between the browser and gRPC server meant that we will not see huge performance gains from using gRPC to talk to our backend service. It imposes more load on mobile phones and we felt the ecosystem is not ripe yet for using gRPC for web communication at the moment.

With a limited span of 3 weeks to come up with our app, we wanted to focus our time on the architecture, look and implementation of our app. Having to also pickup the protocol use case, construct our own Protocol Buffer definitions and handle their translation to JavaScript objects will add on to our priorities and one that we felt was not worth the performance improvements offered by gRPC.

Milestone 5: API Design

Our API documentation is generated using Swagger. You can find the documentation [here](#). Note that the documentation might take a moment to load because of delays by the deployment server.

We have decided to go with the RESTful API architecture and followed some of the guiding principles identified [here](#). The principles are also mentioned in different sources hence we shall use it as the basis for discussion here.

Uniform Interface

We designed our interfaces by routing them to different endpoints in the application. For example, we have a separate **/toilets/** and **/auth/** routes to handle queries for toilet information and authentication. This allows our frontend to identify the resources they need via unique and dedicated endpoints.

For most APIs that require some form of inputs by the frontend, we defined Data Transfer Objects in the backend to ensure there are uniform representations of what the backend expects for each call of the API. The frontend has to adhere to the representations in order to utilise the endpoints.

We also ensured that our endpoints return a uniform representation of the response from the server. The representation is shown below.

```
{
  "status": "success" or "failure"
  "message": "human-readable message describing response"
  "data" : data_returned_in_json_format
}
```

Client-Server

We are using the client-server design pattern, which naturally helps to enforce the separation of concerns as the frontend is now focusing on displaying the representations it receives from the backend. The backend focuses on handling the requests by the frontend to connect to other services, such as the database or authentication third-parties (e.g. Google auth).

Layered Architecture

Our backend system follows a layered architecture which constraints the different behaviour of the system. The frontend follows the contracts defined by the Data Transfer Objects to deliver input to the backend. After which the backend controllers perform input validation and map them to a representation used by the database layer. This means that the upper layers do not need to know how the database stores and manipulates the data. Likewise, the database does not need to know how information is passed around between the client and the server. They interact through the interface of our controller and only know what is happening in their immediate layer.

Statelessness

Every endpoint in our server is stateless. The server does not store any contextual information for it to process the next request. The request made from the server contains the essential information for the server to perform tasks such as validation and querying the database.

Cacheable

While we did not implement server-side caching (due to time constraints), we implemented caching on the client-side (with local storage and service worker) to improve the performance of the application on the frontend such that they do not need to make API calls and wait for the information to be returned to them which may degrade the user

experience. The information we cache generally has an expiry, to ensure that the frontend retrieves the updated information when necessary.

Code on Demand

We did not follow Code-on-Demand as it was optional. At the same time, we did not really need to provide scripts from the backend as the frontend is rather self-contained and built via frontend frameworks (React-Bootstrap). It does not really require any additional scripts to implement its features as it has all the code it needs to operate smoothly.

Good Practices

We took a few of the practices highlighted in this StackOverflow [blog](#). We used JSON as the main medium to transfer data between our client and server. Both our frontend and backend are using some JavaScript framework, which has built-in support for JSON. Hence it was a natural choice to work with JSON.

We also ensured that our endpoints are intuitively named, such that users will know what the API is meant for. For example, `GET api/v1/toilets/with_user_preferences` denotes that we are getting all the toilets appended with the user's preference of each toilet. We also used nouns in the URLs and let the HTTP request methods define the action that we will take. We also try not to have deep nesting of the endpoints to make the endpoints easier to manage.

We also ensured that we used the standard error codes (such as 400 for bad request and inputs, 401 for unauthorised users). We then handled these errors on the frontend by showing a toast and prompting the user on the next action to take (such as authentication).

We also designed our endpoints to provide filtering options. For example, the users are able to get the neighbouring toilets at a certain geographical location.

During deployment, we also made sure that we utilised SSL/TLS for security reasons.

For extensibility reasons, we kept our API in this project to be `/api/v1/`. Having some form of versioning allows us to introduce new changes and gradually phase out old and unused endpoints.

Phase 2: API Server

Milestone 6: SQL Queries

We used Sequelize as the ORM to connect our backend server to the database (PostgreSQL). The following are SQL queries generated by the ORM.

Query 1: Toilets with User Preferences

Query	
ORM Query	<pre>Toilet.findAll({ include: { model: ToiletPreference, as: 'toiletPreferences', where: { user_id: { [Op.eq]: userId, // ignored if undefined }, }, required: false, // Outer join for all toilets }, });</pre>
SQL Query	<pre>SELECT "Toilet"."id", "Toilet"."building", "Toilet"."description", "Toilet"."floor", "Toilet"."longitude", "Toilet"."latitude", "Toilet"."num_seats", "Toilet"."num_squats", "Toilet"."cleanliness", "Toilet"."num_ratings", "Toilet"."type", "Toilet"."utilities", "Toilet"."createdAt", "Toilet"."updatedAt", "toiletPreferences"."user_id" AS "toiletPreferences.user_id", "toiletPreferences"."toilet_id" AS "toiletPreferences.toilet_id", "toiletPreferences"."type" AS "toiletPreferences.type", "toiletPreferences"."createdAt" AS "toiletPreferences.createdAt", "toiletPreferences"."updatedAt" AS</pre>

	<pre> "toiletPreferences.updatedAt" FROM "Toilets" AS "Toilet" LEFT OUTER JOIN "ToiletPreferences" AS "toiletPreferences" ON "Toilet"."id" = "toiletPreferences"."toilet_id" AND "toiletPreferences"."user_id" = 'user_id'; </pre>
Purpose	<p>In this query, we are acquiring all the toilets from that database, and we join it with the current user's preference of each toilet.</p> <p>If the user is unidentified (user_id is null), it will ignore retrieving the user's preference. This is because we used a LEFT OUTER JOIN which will just append the toilets with an empty preference, which will be ignored by the frontend.</p>

Query 2: Fetch ranking of toilets

Query	
ORM Query	<pre> ToiletRating.findOne({ where: { toilet_id: toilet_id, user_id: user_id, }, order: [['createdAt', 'DESC']], }); </pre>
SQL Query	<pre> SELECT "id", "user_id", "toilet_id", "type", "createdAt", "updatedAt" FROM "ToiletRatings" AS "ToiletRating" WHERE "ToiletRating"."toilet_id" = 'toilet_id' AND "ToiletRating"."user_id" = 'user_id' ORDER BY "ToiletRating"."createdAt" DESC LIMIT 1; </pre>
Purpose	<p>This query finds the latest rating that the user made for that toilet. This is used to find out when the user last rated the toilet, and display on the UI on when he is able to rate the toilet again.</p>

Query 3: Deleting User Preference for Toilet

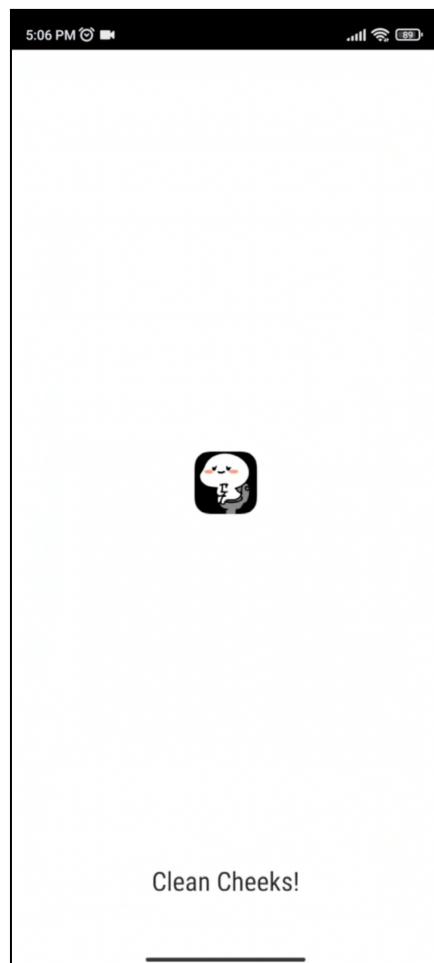
Query	
ORM Query	<pre>ToiletPreference.destroy({ where: { user_id: userId, toilet_id: toiletId, }, });</pre>
SQL Query	<pre>DELETE FROM "ToiletPreferences" WHERE "user_id" = 'user_id' AND "toilet_id" = 'toilet_id'</pre>
Purpose	This query deletes the user's toilet preference, which is executed when the user un-favourite or unblacklists a toilet.

Phase 3: Mobile Client

Milestone 7: Icon/Splash Screen



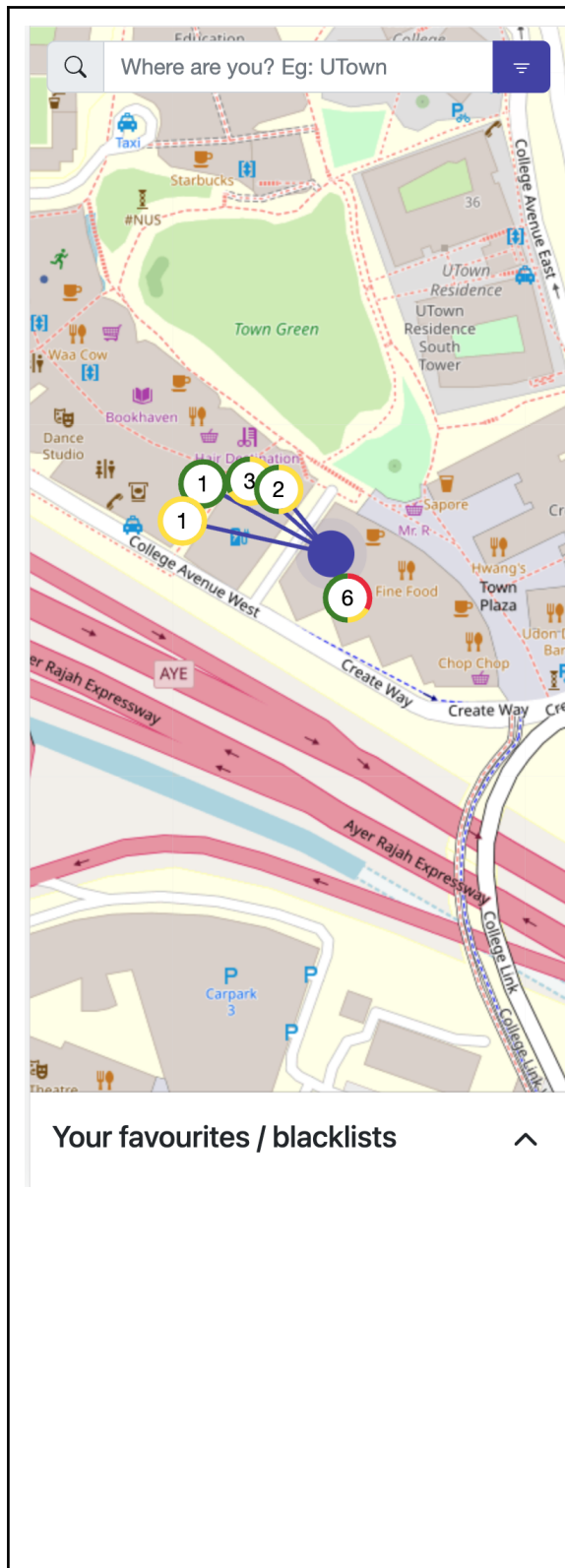
Icon



Splash screen

Milestone 8: CSS

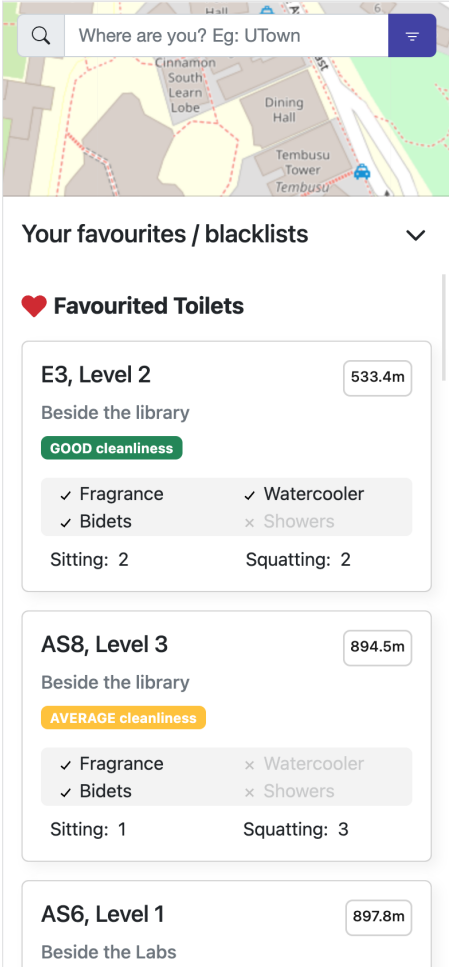
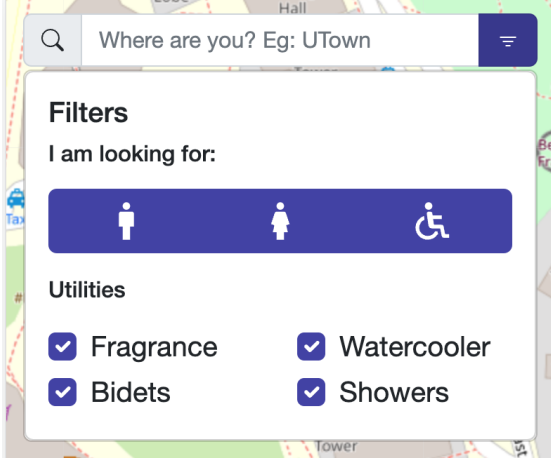
UI Design

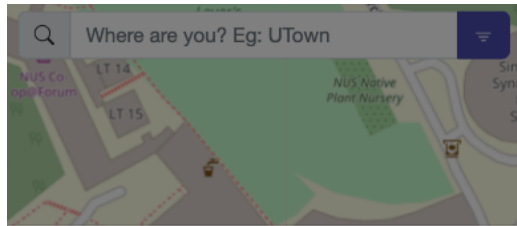


Since our app aims to help users find the closest toilets to their location, our home page shows a large map interface. This makes it easy for users to click on toilet markers on the map and have a more visual representation of how near/far the toilets are from where they are. Since it is possible for multiple toilets to have the same coordinates on the map, the toilet markers are designed to have numbers which correspond to the number of toilets sharing that same set of coordinates. This helps to take away the visual chaos of seeing multiple pins clustered in the same area and the difficulty in clicking on a singular pin. The markers are also coloured green (clean), yellow (average) and red (dirty) so users will know the cleanliness breakdown of toilets in the same area. These are used throughout the app to describe toilet cleanliness, in order to ensure consistency. The proportion of each colour also corresponds to the proportion of clean/average/dirty toilets.

Since one of the core features of the app is allowing our users to find the nearest toilets, searching is an important feature. The home page has a prominent search bar at the top of the screen, making it easily accessible and usable.

Taking inspiration from modern map apps like Apple Maps and Google Maps, our UI design also includes a bottom offcanvas that

	<p>users can easily access from the homepage, which displays their favourite/blacklisted toilets. This makes their saved preferences easily accessible and viewable, which is important to ensure a personalised user experience.</p>
	<p>Our favourites/blacklists offcanvas has a scrollable list, with the favorited toilets positioned at the top and the blacklisted toilets positioned at the bottom. This order of presentation is because users would want to view the toilets they like first rather than the ones they dislike, and they would care more about the status of their favourite toilets (e.g. has the cleanliness of the toilet gone down/up over time?)</p>
	<p>The filter button was designed to be in a prominent position as well, right next to the search bar. This is because with searching also comes a coupled need for filtering. The proximity of the search bar and the filter button prompts users to use both features together to optimise their search experience. The icons used for the filters and the checkboxes are also made to be as intuitive as possible so users will understand them from a glance.</p>



2 toilets are at COM2



56.1m away from your location

COM2, Level 3

Beside staff offices

AVERAGE cleanliness

✓ Fragrance

× Watercooler

✓ Bidets

× Showers

Sitting: 1

Squatting: 1

COM2, Level 1

BLACKLISTED

Opposite the food court

AVERAGE cleanliness

✓ Fragrance

✓ Watercooler

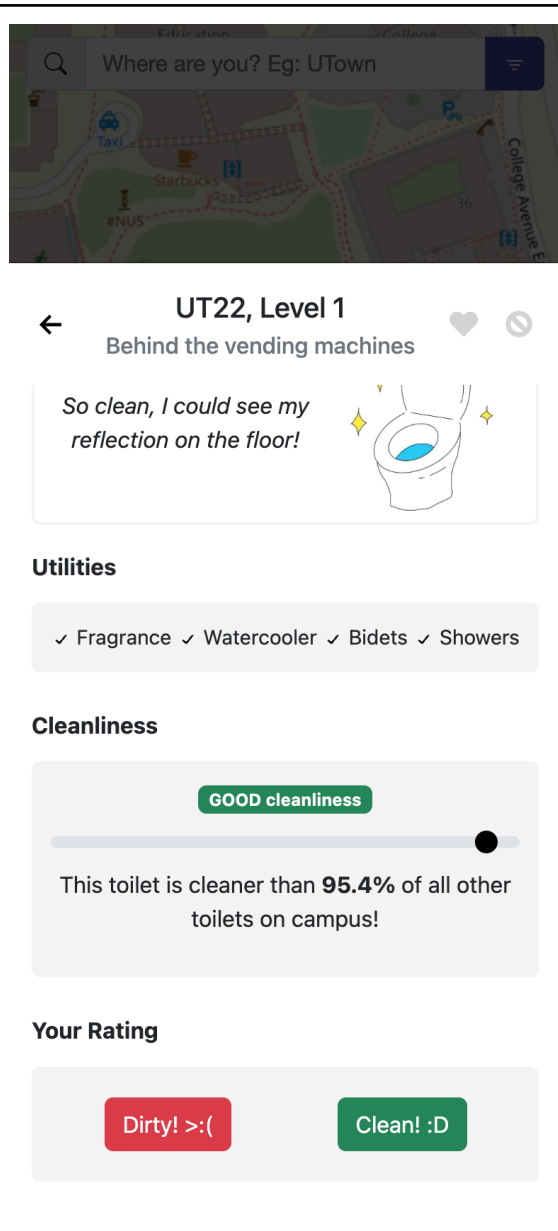
✓ Bidets

× Showers

Sitting: 2

Squatting: 1

When multiple toilets have the same coordinates, clicking on the map marker will open up a list of all the toilets. This list is scrollable and follows a sorting order, with favoured toilets being the first in the list, else the cleanest toilets. This is going by the idea that users will prioritise their favourite toilets. Each toilet is represented as a card on the list, with a drop shadow to create visual hierarchy. The cleanliness rating is represented as a coloured badge that corresponds to one of the 3 colours: green, yellow and red, which are used throughout the app. The toilet's utilities are also displayed clearly on the card, with unavailable utilities being greyed out so the available utilities are more prominent. Lastly, a favourite/blacklisted flag will be shown on the top right of the toilet card for easier viewing.



The toilet detail page provides more functionality for users. In addition to viewing the toilet information (i.e. image (shown as a placeholder in this screenshot), cleanliness rating and utilities) in greater detail, the page also allows users to favourite or blacklist the toilet by clicking on the icons on the top right. These icons were made to be as intuitive as possible so they could be understood by the user. The page also allows users to rate the toilets by clicking on the dirty/clean buttons. These buttons are once again colour coded green/red, which are colours used to indicate cleanliness throughout the app, to ensure consistency.

CSS Methodologies

We used CSS modules to style our application. We chose this methodology as it scales well with the large and complex stylesheets used by the map (**leaflet**) and component libraries (**React-Bootstrap**). Both libraries define a large set of styles and it is easy for our own CSS selectors to conflict with them if they are globally scoped. By using CSS modules, we ensure that each component's custom styling will only be affected by its own stylesheet and minimise the side effect of our CSS classnames conflicting with those provided by **leaflet** and **React-Bootstrap**, leading to unintended side-effects in the look of our app.

Given the limited time frame of the project, we took this approach so that we can style each of our components in parallel without having the fear of conflicting CSS styles. From a scalability standpoint, using CSS modules also allows us to build more features with different CSS styles.

Our app also makes use of Bootstrap CSS, which allows us to use the predefined CSS styles provided by Bootstrap. This set of utility classes allow our team to use a fixed design system, ensuring that our margins, font sizes, colours, typography look mostly the same. This allowed us to use the earlier mentioned CSS modules only when required, reducing the code written for CSS.

```
<Offcanvas.Title className="mb-3 mt-5 fw-bolder">
  <TiCancel
    className={styles['blacklisted-icon']}
    color="#453F41"
    size={28}
  />
  Blacklisted Toilets
</Offcanvas.Title>
```

We believe that we benefited from this approach, allowing us to develop with great speed and correctness, while ensuring readability by writing only custom CSS when necessary in the form of CSS modules, and using utility classes for simpler styles.

Milestone 9: HTTPS

Some of the best practices that we have identified are:

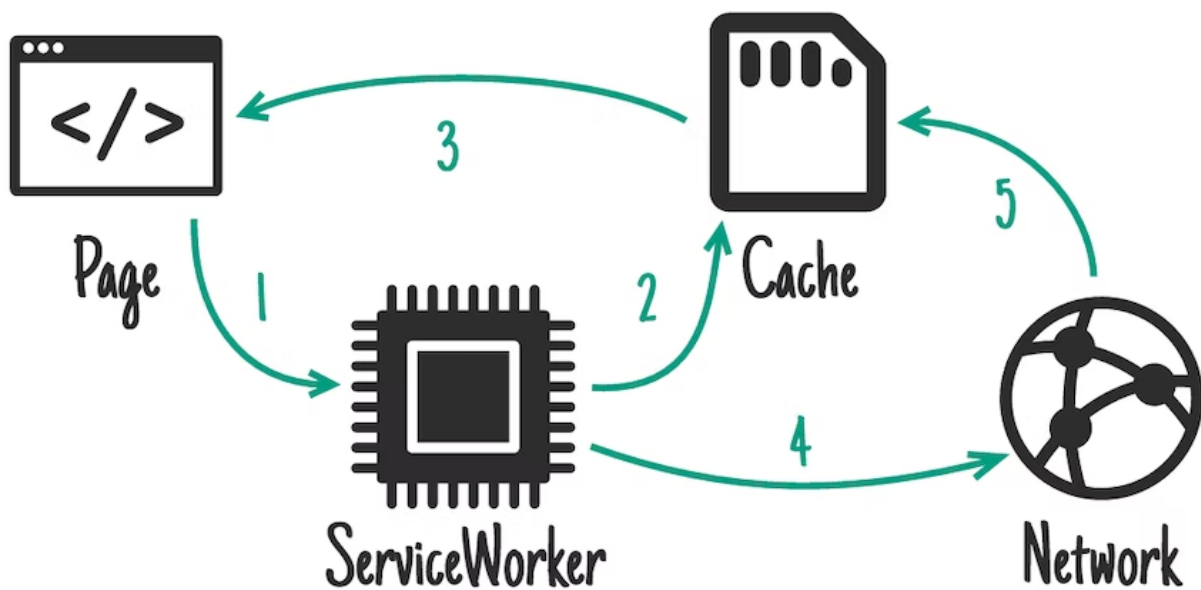
- 1) Choose a reputable Certificate Authority to vouch for your public key. This is to ensure that their certificate is contained within most browsers and HTTPS connections can be set up and launched by clients straight out of the box.
- 2) Redirect users to the secured site when they try to access the HTTP URL.
- 3) Turn on HTTPS support on the web servers.

Our deployments for the frontend and backend uses HTTPS via the deployment sites (Vercel and Render).

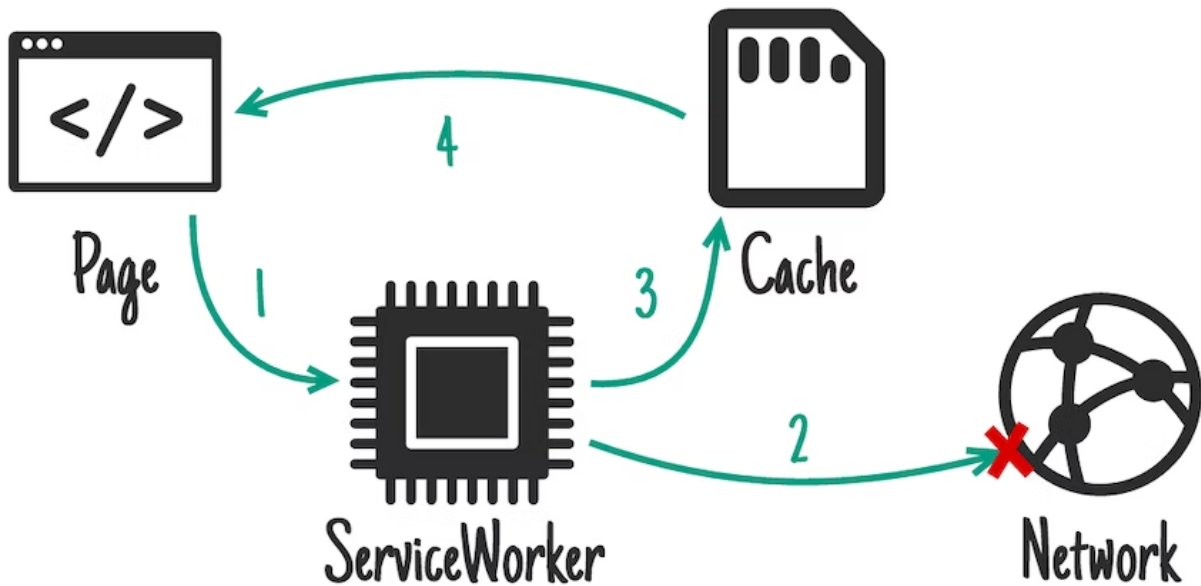
Milestone 10: Offline

When offline, the user will still be able to view the map regions around the location where he went offline as well as the toilet markers near him as we runtime-cache the map images and toilet details using our service worker. His location marker will also be persisted as his last-seen location is stored in localStorage.

We believe it is important for the user to have a sense of continuity when transitioning from the moment network signal is lost to when they are fully offline. That means whatever icons or data they currently see on the map should remain where they are and be viewable. This is why we cache the map images and toilet information so that they can still be used when offline.



We adopted a stale-while-rehydrate caching policy for map images which meant that any request for data, as long as it has been recently seen (one of last 50 requests) by the service worker, will use the result cached to serve the web app. The cached data will then be asynchronously rehydrated by a fresh request. The reason we choose to use cached data instead of using a fresh version from the server is because the map images are unlikely to vary much in between 2 calls. It is quite unlikely a new building will be built or the layout of an area changes in a matter of minutes or hours. This ensures that a response is returned as soon as possible to the user.



We adopted a network-first caching policy for our own api calls. Compared to map images, our own backend-persisted data fluctuates more frequently due to updates in the preferences of users or cleanliness ratings of each toilet. A network-first caching policy guarantees that we use an updated copy of our toilet data from our backend server whenever possible. If that cannot be achieved, most likely when the network is down, we use the cached responses as a better-than-nothing alternative. This minimises the disruption to their offline experience and ensures continuity from their online to offline transition.

We cache the GET requests excluding the call (`/toilets/neighbours`) that fetch the toilets based on user coordinates. This is because the changes in user location which is defined by their latitude and longitude are very minute and it is highly unlikely that their location changes over time will coincide with a previously cached location. Hence, to avoid storing unnecessary responses, we avoid caching this particular endpoint.

Milestone 11: Authentication

In session based authentication, the server stores information regarding the user's session. With each request, the user sends a cookie containing their session ID that the server checks to verify the user's identity. Meanwhile, in token based authentication, the server does not store any information about the user's state. All the server does is verify that the token sent by the user is valid before authenticating the request. The most commonly used token is JSON Web Token (JWT).

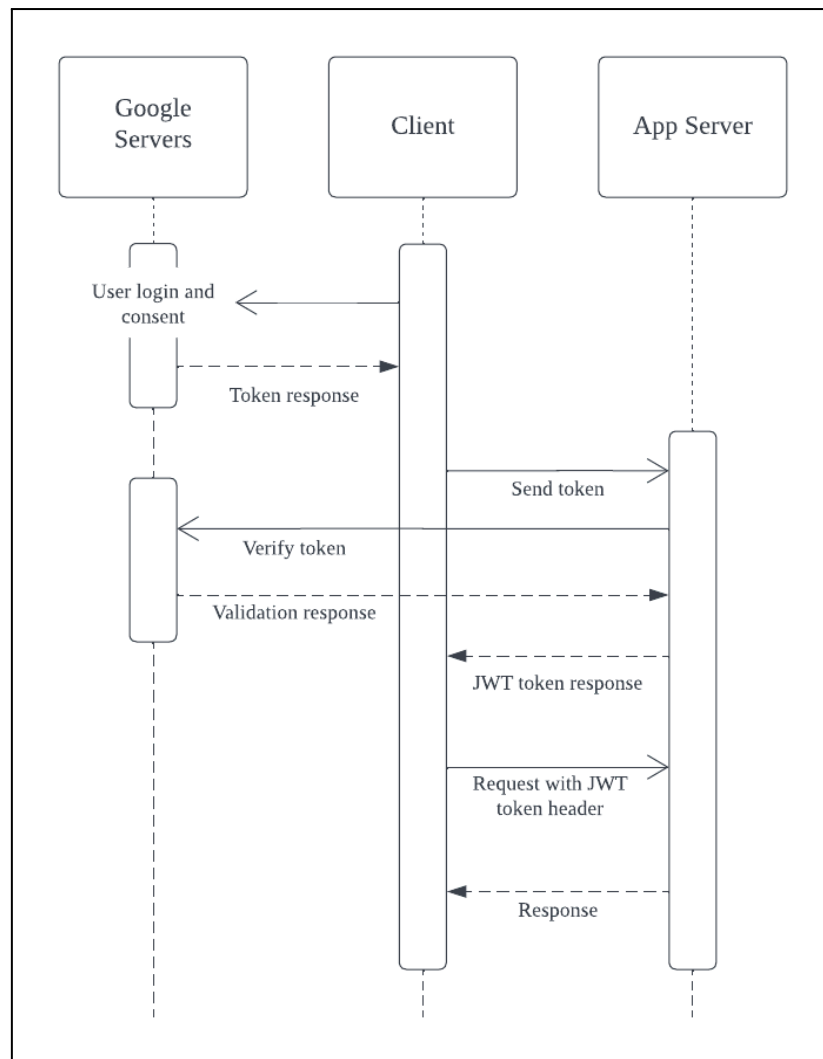
Token based authentication has been gaining popularity over session based authentication in recent years due to reasons such as scalability and performance. On the other hand,

there are also reasons why it might be disadvantageous to use token based authentication as opposed to session based authentication, for reasons including the size of the JWT tokens and security risks. The table below describes in detail the pros and cons of using token based authentication over session based authentication.

Pros of using token based authentication over session based authentication
Scalability With token based authentication, scalability becomes a non-issue since tokens are stored on the client side, allowing applications to handle huge numbers of users. On the contrary, with session based authentication, information about each user's session ID needs to be stored on the server and this takes up space/memory. This makes session based authentication far less scalable than token based authentication.
Performance With session based authentication, the server will have to perform lookups into the database to retrieve the user's session ID, before making a comparison and authenticating the user. This performs worse compared to decoding a JWT token used by token based authentication.

Cons of using token based authentication over session based authentication
Size of JWT token JWT tokens are much larger in size, making it slightly less efficient for storing on the client side. On the other hand, a session cookie is much smaller in size and therefore more efficient for storing.
Security risks With session based authentication, it is possible for a security team to invalidate a user's session ID if they suspect that the account has been breached. This will cause the user to be logged out immediately. However, such actions are not possible for token based authentication since the server does not store any information about the state of the user. If a JWT token is hijacked, it is possible for attackers to use the valid JWT token to make requests with the server.

The diagram below illustrates our team's authentication process.



Our team has adopted token based authentication for our app, while allowing users to log in with their Google account. As our target audience is the whole NUS community, our app will receive a large volume of traffic. Hence, token based authentication is a much more scalable and efficient scheme as compared to session based authentication. We have also set the JWT tokens to expire in 7 days after their generation, so that users can remain logged in for a period of time but also be automatically logged out once the tokens are expired. After receiving the JWT token returned by the server, the client will attach the token in all protected request headers to be validated on the server side.

Milestone 12: Client Framework

Library	React-Bootstrap	Material UI (MUI)	Ant Design
Mobile-Responsive Components	Mobile-responsive components straight out of the box. Allows for layout configuration of grids that vary with the screen widths.		
Documentation	All are well-documented with common use cases and props clearly defined. Tons of help working with customising components from each library are also available on sites like Stackoverflow.		
Support	All are some of the most popular libraries for web frontend and constantly push out new releases with fresh updates.		
CSS styling	Bootstrap's style sheets, which come packaged with React-Bootstrap, give us handy shorthand classnames to style common properties like width or height using just "p-o".	Props have to be passed to each React component. It is a detachment from the underlying CSS stylesheet and incurs some overhead during runtime to convert the props to a CSS style as well.	Need to create own stylesheets and style classes to assign to their components.
Learning Potential	None of us have used React-Bootstrap or Bootstrap before and felt it was a good chance to pick up a new UI library and learn to work with it.	Our team has experience with both of these UI libraries so we felt that the opportunity to gain something new might be limited.	

The different frameworks are all comparable with regards to responsive components, documentation and support. However, we opted to use React-Bootstrap as it allowed us to increase our development speed.

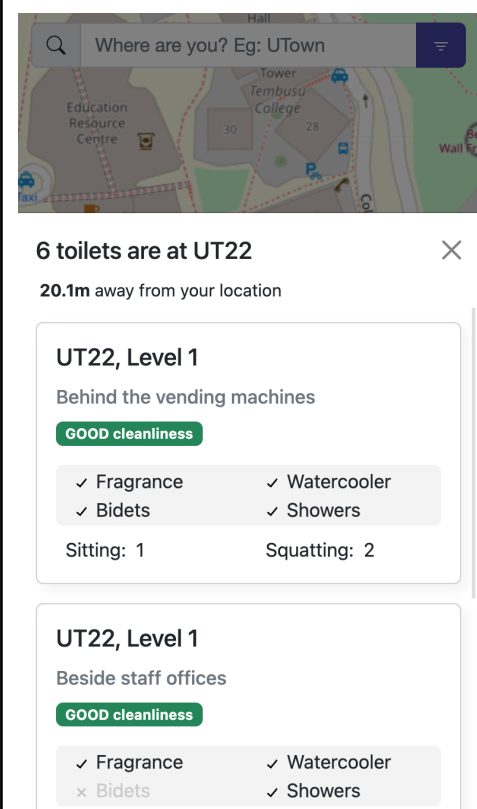
In React-Bootstrap, each component is built as a React component. Moreover, it supports the use of Offcanvas, which are expandable sidebars that we use to show and display our

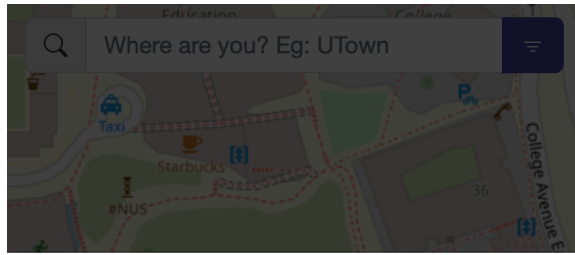
toilet list when a user clicks on a map marker. This built-in functionality allows us to preserve the small space of mobile screens.

React-Bootstrap includes many UI components such as typography, tables, buttons, navigation, labels, notifications, tabs, and so on. It offers enough required elements to create an aesthetically looking interface with minimal effort, allowing developers to focus on the app's usability. Besides that, there are several themes and templates available for download on the internet. As a consequence, development occurs at a rapid pace.

Mobile Site Design Principles

We referenced the design principles from this [site](#). The following are some examples in which we integrated them into our app.

	<p>Firstly, we optimise the UI for mobile sites. There is only vertical scrolling required on the site, and we limit the display to only a single column for our list layouts. This is evident in our toilets cluster display, where all the toilets are displayed in a column for the user to view.</p> <p>The text and fonts are generally larger so that the information is easily seen and understood. Users don't need to zoom in too much or change the sizes to view the content.</p>
--	--



←

UT22, Level 1
♥
ⓘ

Behind the vending machines

So clean, I could see my reflection on the floor!

Utilities

✓ Fragrance
 ✓ Watercooler
 ✓ Bidets
 ✓ Showers

Cleanliness

GOOD cleanliness

This toilet is cleaner than **95.4%** of all other toilets on campus!

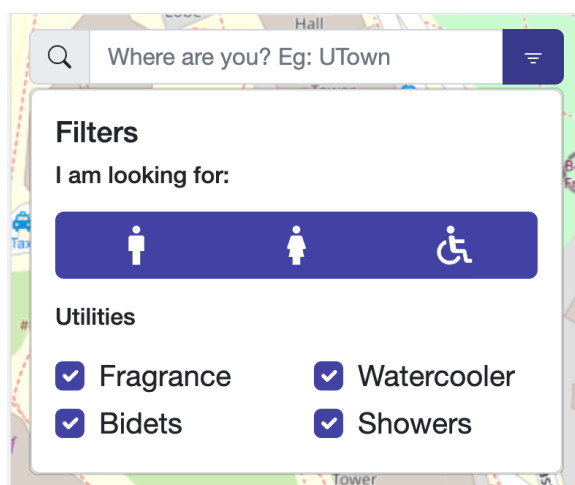
Your Rating

Dirty! >:(

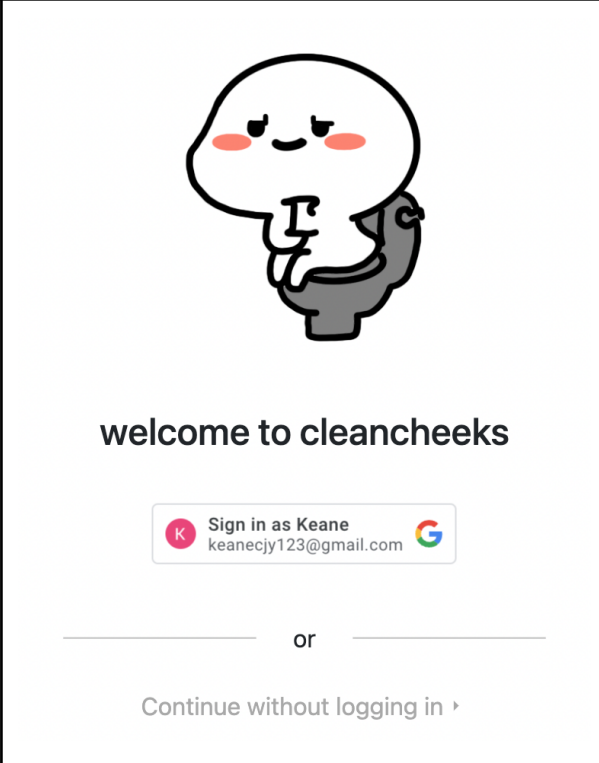
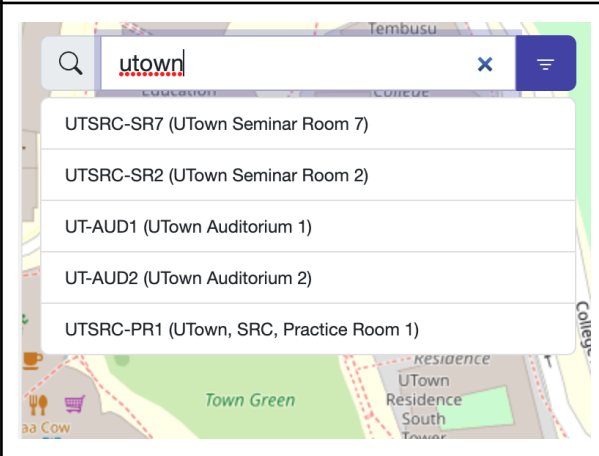
Clean! :D

We also made our Call-to-Action buttons more user friendly. We want to make them prominent to make them easily seen. One such example is our rating buttons. They are larger in size compared to other UI elements and their colours form a contrast with the surrounding text.

The icons are generally large and easy to touch, allowing users to easily tap on the icons to interact with the application.



Next, we also want to make the menus simple but useful and intuitive for our users. If they have too many options and items to select, it might be frustrating for the users and in turn degrade their user experience. Hence, we designed our Toilet filters to be simple, replacing icons with text and making them easily understood.

 <p>The image shows the welcome screen of the 'cleanceeks' app. At the top is a cartoon character with a white face, red cheeks, and a grey body. Below the character, the text 'welcome to cleanceeks' is displayed. Underneath is a login button that says 'Sign in as Keane' with a red 'K' icon and the email 'keanecjy123@gmail.com'. To the right of the email is a Google logo. Below the login button is a horizontal line with the word 'or' in the center. At the bottom is a link that says 'Continue without logging in' followed by a right-pointing arrow.</p>	<p>To give users the flexibility to use Clean Cheeks, we provide them the option to continue to the app without logging in. This gives them the opportunity to explore the application before committing to use our app. Subsequently, they can log in to use additional features such as favoriting toilets or rating them.</p>
 <p>The image shows a search bar in the app. The search bar is white with a magnifying glass icon on the left and a blue 'X' icon on the right. Below the search bar is a list of search results: 'UTSRC-SR7 (UTown Seminar Room 7)', 'UTSRC-SR2 (UTown Seminar Room 2)', 'UT-AUD1 (UTown Auditorium 1)', 'UT-AUD2 (UTown Auditorium 2)', and 'UTSRC-PR1 (UTown, SRC, Practice Room 1)'. The background of the image is a map of a campus area with labels like 'Tembusu', 'Education', 'College', 'Residence', 'UTown', 'Residence South', 'Tower', and 'Town Green'.</p>	<p>To make information easily accessible in our application, we made the Search Bar prominent in the main page of our application. It is the fastest way for users to query where toilets are and it is displayed at the top of the application. This makes it hard to miss and intuitive for users to use.</p>

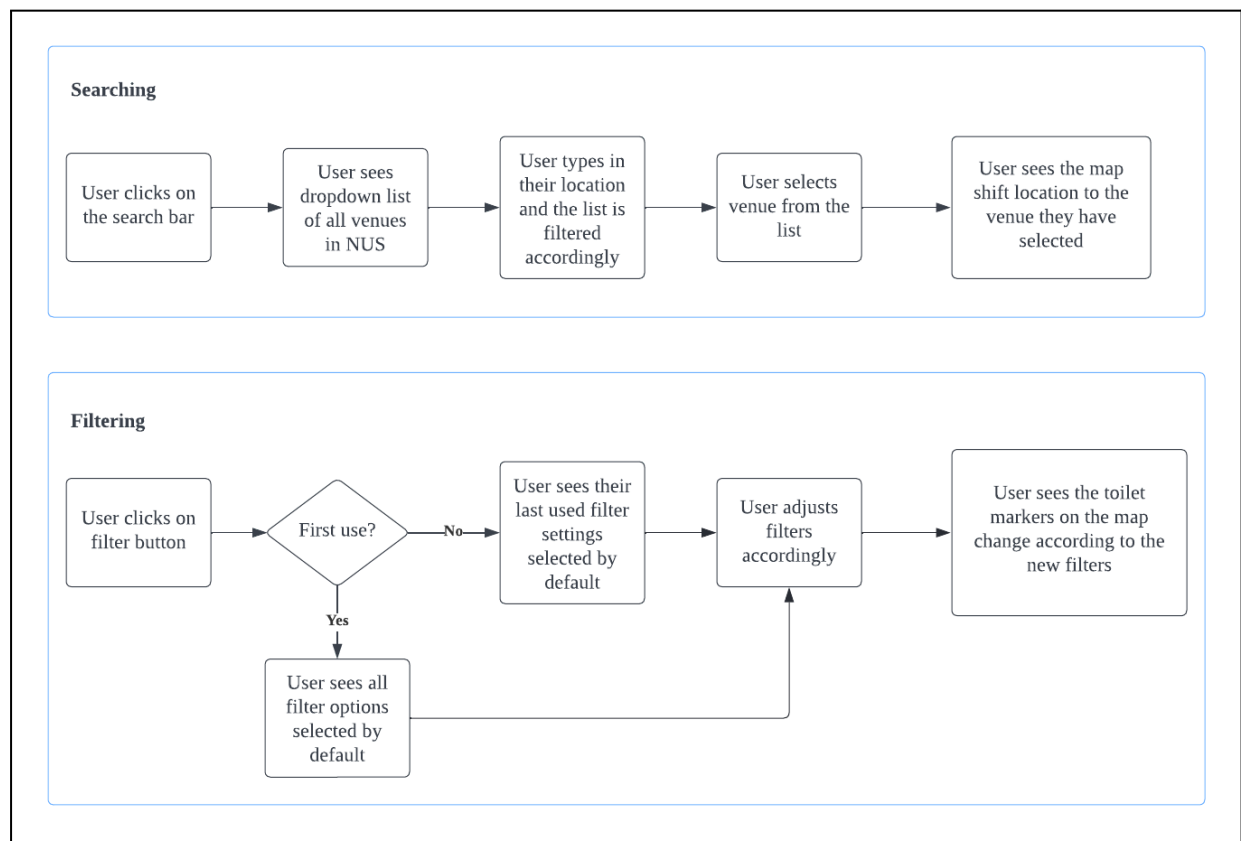
Milestone 13: User Experience

The 3 major workflows in our application are

1. Searching/Filtering toilets flow
2. Favouriting/Blacklisting toilets flow
3. Rating toilets flow.

Below are diagrams showing these flows in detail.

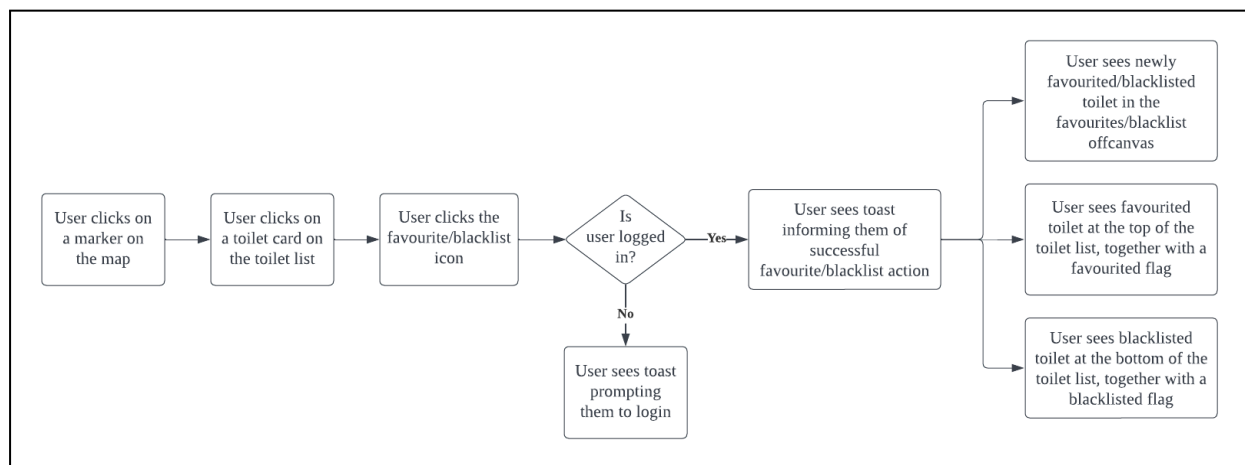
Searching/Filtering Toilets Flow



We chose this flow as a main user flow as it is important for users to be able to search for the toilets nearest to them. It would not be helpful if users had to scroll around the map to find their current location and the toilets near it. As such, a searching feature to let users zoom in directly to their location on the map is extremely vital and a must-have for our application.

Moreover, it is also important to have functionality that lets users only see results that are relevant to them, through the use of a filtering feature. It would not be very helpful if users had to click into every single toilet on the map to see which ones are male/female/handicapped, or which ones have bidets/showers. With the filter feature, users simply have to select the type of toilet and utilities they are looking for and the relevant results are displayed automatically.

Favoriting/Blacklisting Toilets Flow



****Note that a similar flow applies for unfavourite/unblacklist actions***

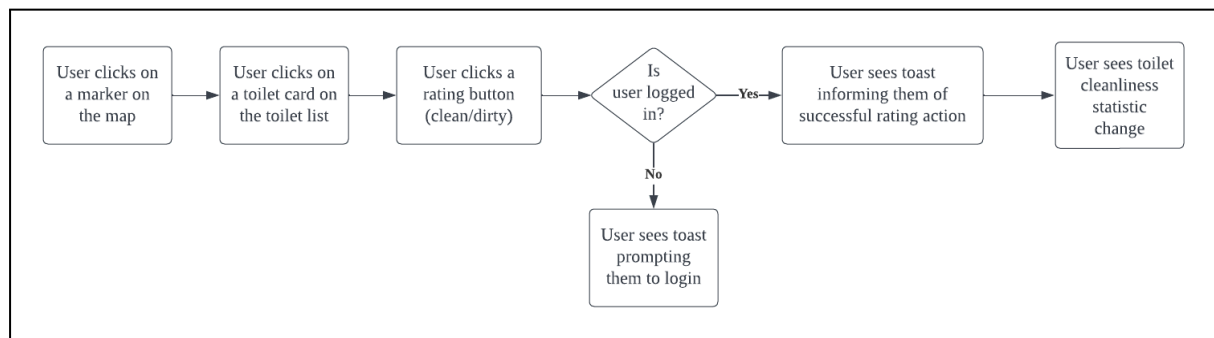
We chose this flow to allow users to specify their toilet preferences. There may be certain toilets that users particularly like, with the utilities (e.g. showers/bidets) that they are looking for. There may also be certain toilets that users particularly dislike and want to actively avoid. As such, the favouriting and blacklisting flow allows users to specify these preferences easily.

When a list of toilets at a particular location is displayed, users will be able to see their favourited toilets at the very top of the list, while the blacklisted toilets are at the very bottom. This helps to facilitate the user experience as they are immediately given the toilets they prefer and need not scroll through the toilets they dislike.

The app also provides an offcanvas on the home screen which displays all of the user's favourited and blacklisted toilets. This helps to facilitate a user's change in preferences. In the event an existing favourited toilet is no longer up to standard, users can easily unfavourite it from the offcanvas without having to manually search for it on the map. The same applies to blacklisted toilets that have improved in standard.

Altogether, the favouriting/blacklisting flow helps to improve and personalise the user experience.

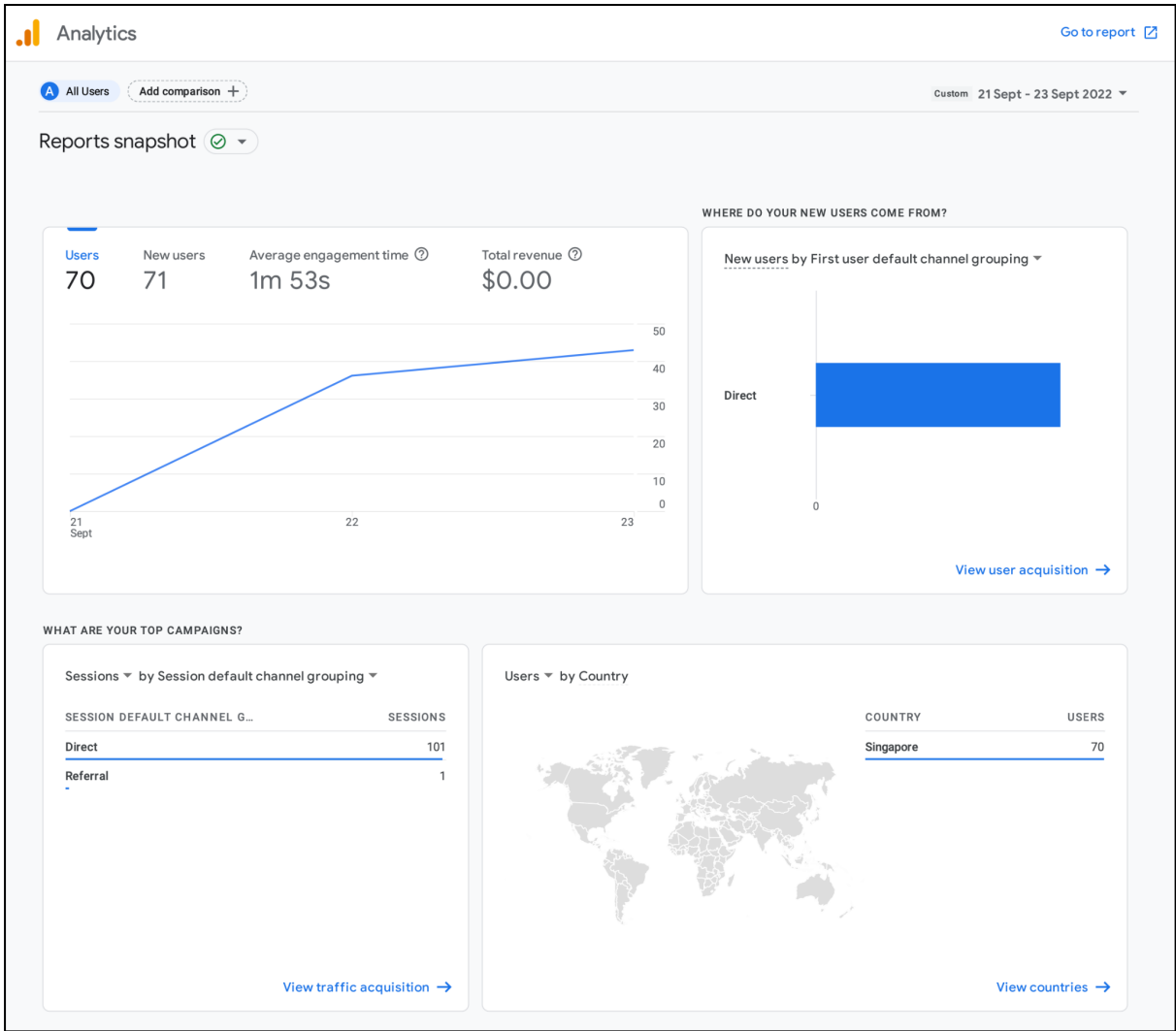
Rating Toilets Flow

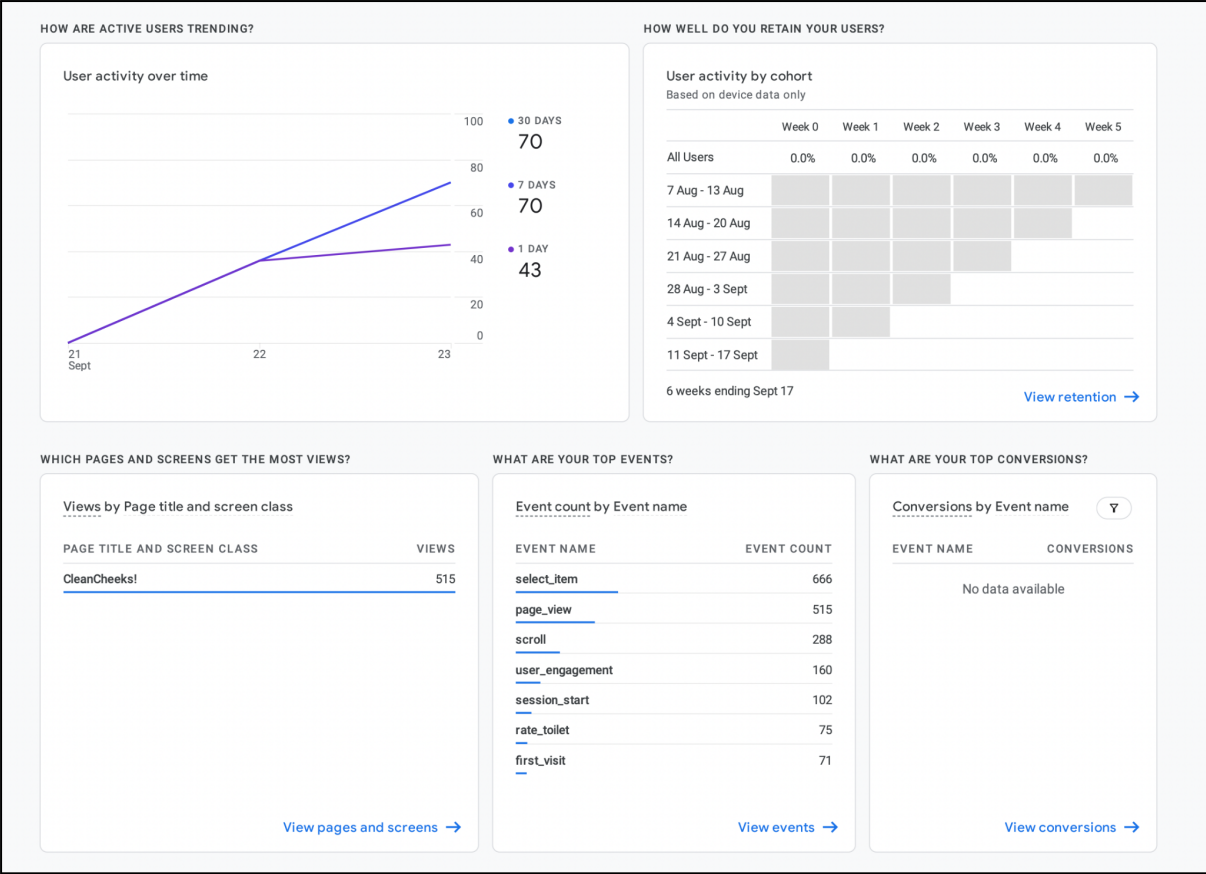


We chose this flow as it is important in keeping toilet cleanliness scores up to date. Since the cleanliness of a toilet fluctuates over time (or even throughout the day), our application needs a feedback channel to get real-time updates regarding the state of the toilets. As such, we included this flow to let users easily rate toilets as clean or dirty, and we use this rating to calculate an updated cleanliness score. This helps to improve the user experience as well, as users have a way to express their satisfaction or dissatisfaction with the toilets, and receive updated, real-time information regarding the cleanliness of the toilets.

Milestone 14: Analytics

We used Google Analytics to keep track of user retention as well as the common events in our application, which includes login, rate_toilet, favourite, blacklist.





Q Search...					Rows per page: 10	Go to: 1	< 1-10 of 11 >
Event name	+	↓ Event count	Total users	Event count per user	Total revenue		
		2,054 100% of total	70 100% of total	29.37 Avg 0%	\$0.00		
1	select_item	666	33	20.18	\$0.00		
2	page_view	515	70	7.36	\$0.00		
3	scroll	288	64	4.50	\$0.00		
4	user_engagement	160	31	5.16	\$0.00		
5	session_start	102	70	1.46	\$0.00		
6	rate_tollit	75	9	8.33	\$0.00		
7	first_visit	71	70	1.01	\$0.00		
8	search	66	28	2.36	\$0.00		
9	FAVOURITE	57	20	2.85	\$0.00		
10	BLACKLIST	38	11	3.45	\$0.00		

Milestone 15: Google Lighthouse

The report can be found in the main folder of our repository.

Phase 4: Coolness Factor

Milestone 16 (Optional): Social Integration

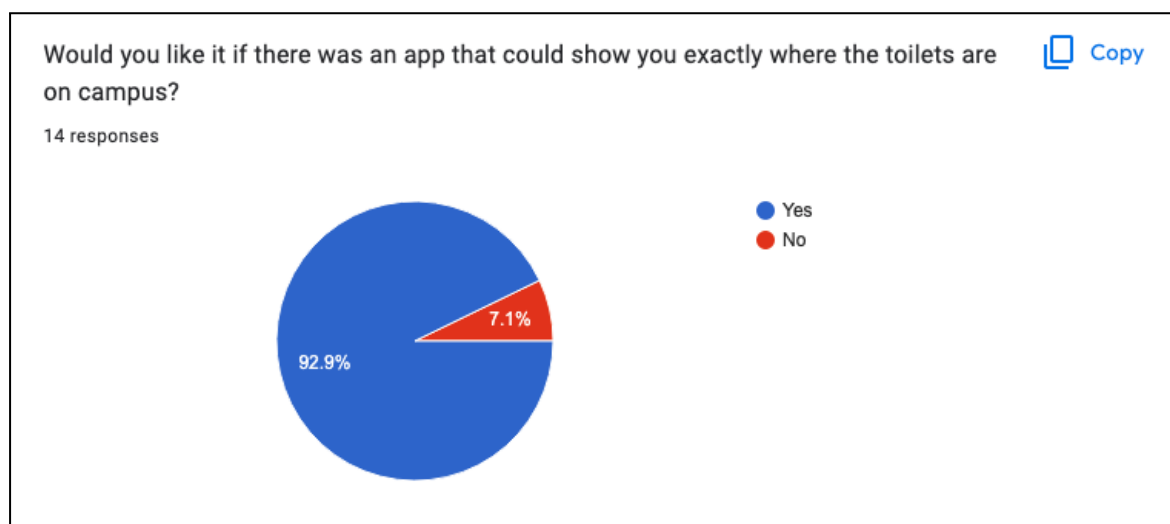
Our app utilises Google Identity Services for logging in. Users can log into the app with just a click of a button. We opted to use this as a large majority of our target audience has a Google account, and using social login makes the login process a lot more seamless. Moreover, users do not have to create new usernames or passwords in order to log into our app, allowing for a hassle free experience.

Milestone 17 (Optional): Geolocation

Our app utilises Geolocation API. Users can enable or disable location services. If enabled, the app will zoom to their location and can show the toilets nearest to them. If disabled, users can still use the search bar to search for the venue that they are located in, in order to view the toilets nearest to them.

Appendix

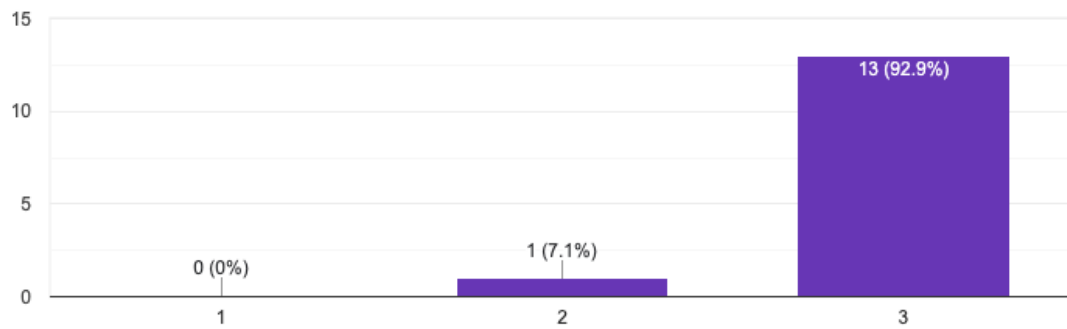
User survey results



How important is the cleanliness of a toilet to you?

 Copy

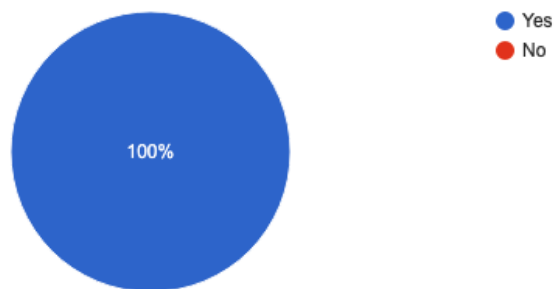
14 responses



Would you like it if there was some way to know the cleanliness of the toilets around you before you pick one to go to?

 Copy

14 responses



With regards to the previous question, how important would a feature like that be to you?

 Copy

14 responses

