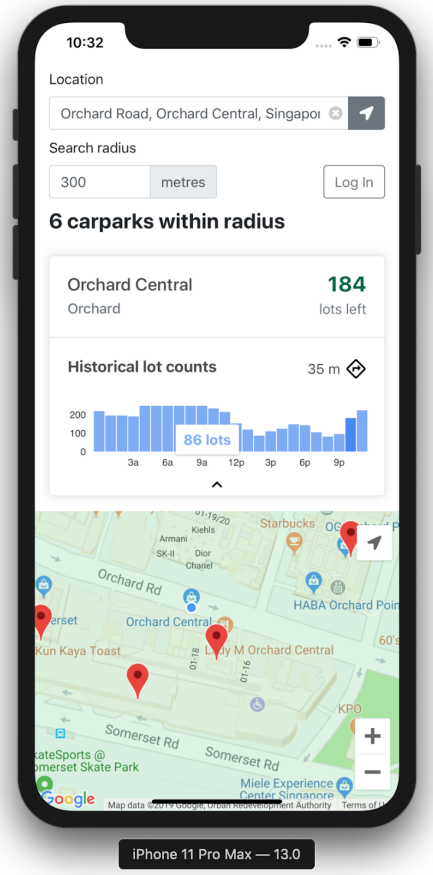


Carina

CS3216 Software Product Engineering for Digital Markets

Assignment 3 Milestones

Group Members: Lau Kar Rui

Jeremy Tan Kai Qun

Ng Wei Jie, Brandon

Table of Contents

Milestone 0: Problem Description

Problem Statement

Driving around in circles just to find a parking lot? Waiting for more than 30 minutes before finding a parking lot? Apologising to friends and family for being late because you can't find a parking lot? These experiences should be familiar to Singapore drivers when they are finding parking lots, especially during the peak hours at urban districts.

Finding parking lots in Singapore can be difficult and frustrating during peak hours in urban areas. Often, crowded carparks cause drivers to end up driving around in circles for long periods of time to search for available parking lots. For instance, in order to find a parking lot, drivers may have to either drive to another carpark or visit other applications that show available lots for only a particular carpark. This is time consuming and tedious.

Furthermore, existing platforms such as Google Maps may help drivers to find their way to the nearby carparks but it does not indicate the number of empty lots available in those carparks. To make matters worse, drivers would feel even more frustrated when they have driven all the way to the carpark only to find out that the carpark is full.

As such, having real-time information on carparks can potentially be useful for Singapore drivers to minimise the time needed to find available parking lots. Imagine having real-time data of all the carparks being made available on a single platform where drivers can easily access them, check which carparks have available parking lots and easily locate them...

Milestone 1: About Carina

Application Description

This is exactly what Carina is built for. Carina is a progressive web application that aims to help drivers search for parking lots seamlessly and conveniently. The core feature of this application is to allow drivers/users to view the number of available lots for the carparks within the vicinity so that they can better find a parking lot. This data is real-time and is accessible from any web browser on desktops or mobile devices. Additionally, Carina also has additional features such as:

1. Show the proximity of the carparks so that drivers can find out which carpark is nearer.
2. Display a histogram of the number of available lots for a particular carpark across the day for drivers to know which time period is the least/most crowded. This information could facilitate drivers to plan ahead if they should avoid/go to the carpark during that period.
3. Allow drivers to save/favourite their frequently-visited carparks so that they can easily look up the number of available lots in these favourite carparks.

Flexibility

Carina allows users to save/favourite their frequently-visited carparks anywhere. To allow users to access their favourites seamlessly, Carina leverages on the flexibility of mobile cloud computing so that the users can access the application and mobile data from any device easily. The data of a user would not be stored on the current mobile device that requires manual transfer if the user wants to switch to another mobile device.

Multiple Platform Support

Drivers looking for carparks can use Carina to search for available parking lots irrespective of the target platform, such as IOS device, Android, or Desktop Browsers. To find the available parking lots easily, the users can easily access the software and data stored in the cloud. Therefore, Carina leverages on multiple platform support of mobile cloud computing to allow the user experience to be hassle-free.

Cost Efficient

The services that Carina provides are currently free of charge. However, there is a recurring cost to host Carina. Hence, using cloud computing is one of the most cost-efficient methods to lower the upfront cost of Carina. Additionally, unlike IOS and Android applications, there are no hefty fees charged for upgrades and licensing.

Real-Time Data Availability

Carina provides real-time data of the nearest carpark lots, and it also provides a histogram of the number of available lots for a particular carpark across the day. Therefore, mobile cloud computing allows Carina to provide real-time carpark lots to the users as all the data is managed externally, and users can access and update real-time data on their mobile devices.

Milestone 2: Carina's Users

Target Users

The first group of Carina's users are the drivers who want to bring their families for a family outing on the weekends or for festive seasons.

- **Profile**:
  - The driver brings his family out by car.
  - The family owns more than one car.
  - The entire family cannot fit into a car.
- **Behaviour**:
  - The driver brings the family to shopping malls and other developments.
  - The driver brings the family to visit friends and relatives.
  - The driver search for a parking lot at multiple areas beforehand.
- **Goals**:
  - The driver wants to reach their destination quickly.
  - The family and the driver wants to start their outing soon.
  - The driver wants to find the nearest carpark that guarantees a parking lot.
- **Frustrations**:
  - Carparks are full at peak hours at shopping malls and other districts.
  - The driver wait for a long time before finding parking lots.
  - The parking lot is occupied by the time the driver reaches it.
  - The driver needs to drive to another carpark to find out the available lots.

The second group of Carina's users are the workers/employees who go to work urban districts.

- **Profile**:
  - The driver often drive to the company located at an urban district.
  - The family owns more than one car.
  - Each family member drives a car to work.
- **Behaviour**:
  - The driver wants to reach the company on time.
  - The driver wants to avoid parking in the company because it can be costly.
  - The driver have breakfast nearby in the urban district.
  - The driver search for a parking lot at multiple areas beforehand.
- **Goals**:
  - The driver wants to save time on finding carpark.
  - The driver does not want to worry with find carpark.
  - The driver wants to find the nearest carpark that guarantees a parking lot.

- **Frustrations**:
  - Carparks are full at peak hours at shopping malls and other districts.
  - The driver wait for a long time before finding parking lots.
  - The parking lot is occupied by the time the driver reaches it.
  - The driver needs to drive to another carpark to find out the available lots.

From the user profiles above, the users can be summarized into two categories. The first category are families who want to find carparks for family outings or to visit relatives and friends during festive seasons. To dive further, the carparks for visiting relatives and friends are mostly at HDB based on the assumption that four-fifths of Singapore families reside in HDB. The carparks for family outings are mostly in shopping malls and other popular districts. The second category are the employees who want to find carparks at their companies which are mostly location at urban areas. The similarities among these drivers are that they have a set of favourite carpark lots they will search for, and that they will search for the available lots before driving.

Therefore, the target users are drivers who want to search seamlessly for real-time data of parking lots at urban areas, HDB, and shopping malls.

Promote Application

Given that the driver is likely to search for available lots onsite when the current carpark he is at is full, the driver needs to search for Carina on Google easily. Therefore, the application needs to appear as one of the top hits on Google search results. This means that Search Engine Optimization (SEO) for the application is required and optimized. The search keywords and meta tags for SEO are cars, available, parking lots, nearest, and search. This is done using Google Webmaster with no cost to implement.

Additionally, the application can be promoted online to rapidly increase the customer pool. Singaporeans are generally digitally savvy, and most Singaporeans should have utilized some form of social media, such as Facebook and Instagram. Promoting the application on these platforms would easily reach out to many customers easily. Most of the older generation would be using Facebook and the younger generation would be using Instagram. Advertising Carina to users with on these platforms with tags, such as shopping, festivals, and carparks, would reach a wide potential consumer base. This method is still relatively cheap, but the level of aggressiveness in promoting the application depends on the budget size.

Moreover, an important part of the online marketing effort is the full-length promotion videos. These videos will be posted on YouTube. The videos start off showing common frustrations and pain in searching for carparks and how the application is used to solve the problem. The emphasis is on how easy it is to use Carina to find the nearest carparks, and that they can also save their favourite carparks to check on the parking lots available in future seamlessly. This is very effective in showing the user step-by-step process of using the application, with some cost.

Furthermore, billboards and advertising boards near carparks at HDB, shopping malls and urban areas can be used to promote the application. The advertisements will contain a big text “Having problems finding carparks?” or “Want a parking lot now?”, followed by simple steps to use the application. Drivers who face too many difficulties in securing parking lots would be immediately interested and want to know how to use this application immediately. However, the promoting method may be expensive, particularly in urban areas, because many other vendors are using such platforms to promote their products.

Milestone 3: Entity-Relationship Diagram

ER Diagram

Milestone 4: REST API

API Description

Documentation Link: https://www.nwjbrandon.com/swagger-ui.html

The project used Swagger Docs to document all the APIs. The reason for using Swagger Docs is that it is automatically build and serve whenever a new endpoint is added. Furthermore, the front-end developers can test the endpoint by filling in the information in the request parameters and request body, including the authentication token on their local machine.

The endpoints are grouped into /api/public and /api/private. This is for convenience to the front-end developers to know which endpoints requires authentication to fetch data.

Version Endpoint

GET /api/public/version

This endpoint simply receives the version of the backend api. No request parameters and request body is needed. This endpoint is created at the very first step to ensure that the endpoint setup is working. The subsequent uses of this endpoint is to track whether the backend is deployed successfully with the correct version.

Authentication Endpoint

POST /api/public/auth/login

```
{
        email: “test@example.com”,
        password: “password”
}
```

The endpoint logs in the user. The post request is done with the above request body. The data is put into the request body instead of the request parameters, so that the data is not so exposed in the endpoint url. Calling this endpoint fetches a jwt token to be added into the headers of subsequent api calls. If authentication fails, a 401 response will be returned.

POST /api/public/auth/signup

```
{
        email: “test@example.com”,
        name: “test”,
        password: “password”
}
```

The endpoint create an account for the user. The post request is done with the above request body. The data is put into the request again so that the data is not so exposed in the endpoint url. Calling this endpoint only creates a user account so that the user can store their favourite carparks.

Carpark Availability Endpoint

GET /api/public/carpark-availability/carpark-id

?carparkIds=A0007,A0012&lotTypes=C,Y

This endpoint will return a response of the latest parking lots available in the carpark by carpark id and lot type. The above request parameters are provided with the endpoint url. As the data is obtained from LTA DataMall, there are parking lots for cars, motorcycles, and heavy vehicles. Currently, Carina provides information for cars only, but by implementing the GET request to accept a list of lot types in the request param, Carina can be configured to display other types of parking lots easily in future with minimum changes on the backend. If the carpark does not exist, an empty object is returned.

GET /api/public/carpark-availability/latest

This endpoint will return a response of the parking lots for all the carparks. This difference between this endpoint and the previous one is that this endpoint will provide data for all the carparks when Carina is first loaded. If a selected few carparks is required to be updated, this endpoint will not be fired, but the previous endpoint is. This is to save time on the response and the unnecessary computation to update all the carparks.

GET /api/public/carpark-availability/nearest/queries

?latitude=1.4450904&longitude=103.7975843&lotTypes=C&radius=300

The endpoint will return a list of all the carparks within the vicinity of the user. The client can specify his current location and the distance he is willing to travel to find available parking lots. If no carparks are found, an empty list is returned. The latitude, longitude, and radius are in the request params because semantically, it is easier to read these customizable data as the request parameters. Besides, having the decimal point in the endpoint url does not make sense.

GET /api/public/carpark-availability/statistics

?carpark_id=A0007&days=1,2&lotTypes=C

The endpoint will return the number of available parking lots for a specified carpark over the past week. The data are sorted by time and day. This statistical data will be plotted on a histogram on the client side. If the carpark cannot be found, an empty list is

returned. If the parking lot is missing for a particular hour, the data for that hour and day will simply be missing JSON response. The carpark id is inside the request parameters and not a path variable in this endpoint, even though the endpoint is fetching the statistics for a single carpark only. This is because the other endpoints use the request params for a list of carpark ids, and to standardize the endpoints, the carpark id is set as the request param.

User Endpoint

GET /api/private/user/favourites

This endpoint will return all the user information except for the password hash. There is no request params or request body required for this endpoint. Instead, the primary key of the user is obtained in the jwt token attached in the headers of each request. The jwt is decoded to get the primary key of the user, and the user data is obtained with the primary key. If jwt is invalid or expired, a 401 response is returned.

PUT /api/private/user/favourites

```
{
        carparkId: “A0007”
}
```

This endpoint will store the user favourite carpark. Only the carpark id is required in the request body for this endpoint, and the jwt token provided in the headers of this PUT request. The jwt is decoded to get the primary key of the user, and the user id is obtained from the user table. The user id, the foreign key, is used to insert the user favourite stored in the favourites table. A new list of the user favourites is returned. If jwt is invalid or expired, a 401 response is returned.

DELETE /api/private/user/favourites

```
{
        carparkId: “A0007”
}
```

This endpoint will delete the user favourite carpark. Only the carpark id is required in the request body for this endpoint, and the jwt token provided in the headers of this DELETE request. The jwt is decoded to get the primary key of the user, and the user id is obtained from the user table. The user id, the foreign key, is used to delete the user favourite stored in the favourites table. If jwt is invalid or expired, a 401 response is returned.

GET /api/private/user/settings

This endpoint will return all the user information except for the password hash. There is no request params or request body required for this endpoint. Instead, the primary key of the user is obtained in the jwt token attached in the headers of each request. The jwt is decoded to get the primary key of the user, and the user data is obtained with the primary key. If jwt is invalid or expired, a 401 response is returned.

Currently, even though the endpoints in the user controller, including the user authentication for social logins in Milestone 15, are functioning, the endpoints are not in used. The user data request and authentication, including the social login, are done with firebase. This was a last minute change because firebase also provides additional features such as realtime crash reports, application monitoring, and data analytics that are useful for understanding customer behaviour. All these data are not recorded yet, but these implementation set foot for future developments of Carina.

Milestone 5: Database

SQL Queries

INSERT INTO favourites (carpark_id, user_id) VALUES (:carpark_id, :user_id)
When the user save his favourite carpark, this SQL query will be executed to store the user's favourite carpark. The user_id is a foreign key tied to the user table.

SELECT * FROM carpark_availability_monday where lot_type IN (:lotTypes)
To display the available parking lots as a histogram on the client hourly, the user can specify the type of parking lots he wants to view. There are three types of parking lots to choose - cars, heavy vehicles, and motorcycles. Therefore, this SQL query will fetch all the data that contains the type of parking lots that the user requested.

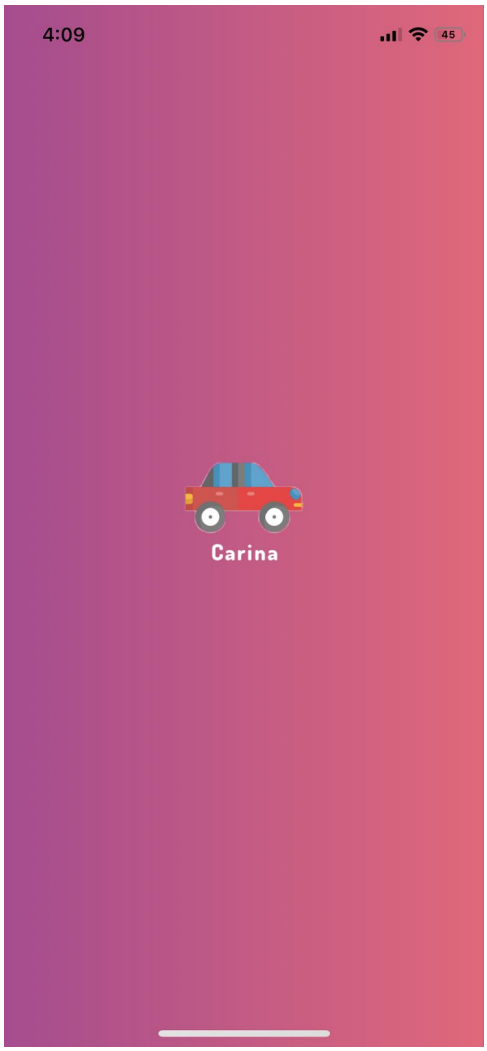
DELETE FROM favourites WHERE carpark_id = :carpark_id and user_id = :user_id
When the user deletes his favourite carpark, this SQL query will be executed to delete the user's favourite carpark. The user_id is a foreign key tied to the user table.

UPDATE carpark_availability_monday SET available_lots = :available_lots WHERE carpark_id = :carpark_id AND area = :area AND development = :development AND lot_type = :lot_type AND hour = :hour
To display the available parking lots as a histogram on the client hourly over the entire week, the data needs to be updated from LTA DataMall on an hourly basis. An AWS Lambda is running with Cloud Watch Event used to trigger hourly updates. The carpark_id, area, development, lot_type and hour are used together as composite key to update the available lots.

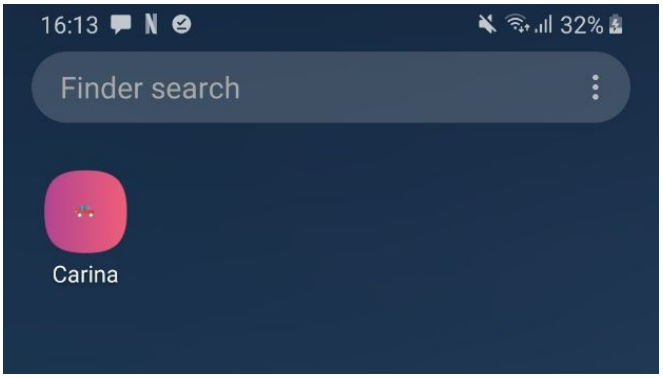
Milestone 6: Home Screen

Attractive Icon and Splash Screen

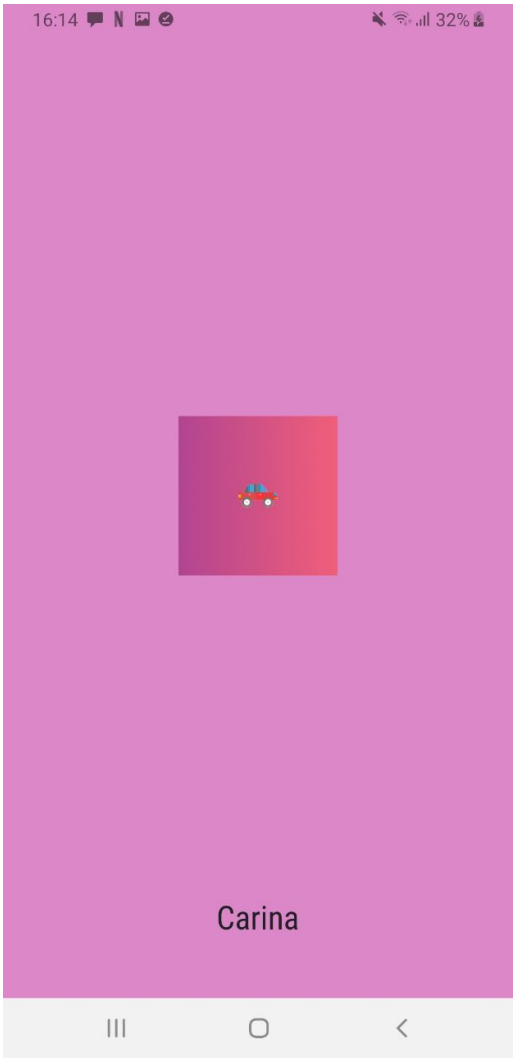
Icon on iPhone



Splash screen on iPhone



Icon on Android



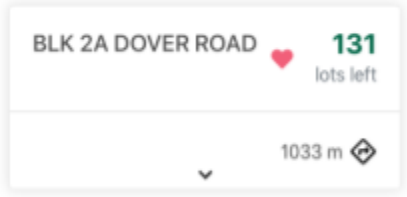
Splash screen on Android

Milestone 7: UI Components

CSS Methodologies

The CSS methodology used in this project is mostly Object Oriented CSS (OOCSS). Since we use SCSS as a CSS preprocessor due to its ease of writing nested syntax, OOCSS is the best methodology to pair with our preprocessor.
In addition, the CSS library Bootstrap 4 is also used in this project, and their CSS methodology used is also OOCSS or close to that equivalent, keeping the CSS Methodology consistent.

Some examples of how this methodology is applied can be seen in the CarparkInfo component below:



CarparkInfo component

```
<div class="info-wrapper">
    <div class="info card">
        <div class="card-body d-flex no-gutters
                    justify-content-between">
            <div class="carpark-addresses">
                <h5 class="card-title">...</h5>
            </div>
            <div class="carpark-lots">
                <div class="lot-count high">131</div>
                <div class="card-subtitle text-muted">...</div>
            </div>
        </div>
        <div class="card-body d-flex no-gutters justify-content-end
                    elab-wrapper">
            <div class="d-flex distance-info">...</div>
        </div>
        ...
    </div>
</div>
```

We can see that the component CSS is object oriented, starting with a "`info-wrapper`" class, then slowly becoming more specific in the styling, such as having a "`info`" class which contains "`card-body`" classes, which in turn contains specific classes such as "`carpark-addresses`", "`carpark-lots`", and "`distance-info`". Styling the CSS this way follows the OOCSS principle of separating containers with content, giving us the assurance that: (1) all unclassed container units will look the same; (2) all elements with the category class will look the same; and 3) we won't need to create an override style for the case when you actually do want e.g. `.myObject` h2 to look like the normal <h2>.

An abridged version of the CSS styles used for the above component can be seen below, which also demonstrates the power of SCSS over CSS:

```scss
.info-wrapper {
  padding: 1em 0;
  margin-left: 8px;
```

```
margin-right: 8px;

.info {
  position: relative;
  color: rgb(72, 72, 72);
  margin-bottom: 4px;
  border-width: 1px;
  border-style: solid;
  border-color: rgb(235, 235, 235);
  border-image: initial;
  border-radius: 4px;

  &:hover {
    position: relative !important;
    color: rgb(72, 72, 72) !important;
    height: 100% !important;
    margin-bottom: 4px !important;
    box-shadow: rgba(0, 0, 0, 0.12) 0px 0px 12px !important;
    border-width: 1px !important;
    border-style: solid !important;
    border-color: rgb(235, 235, 235) !important;
    border-image: initial !important;
    border-radius: 4px !important;
  }

  &.selected {
    position: relative !important;
    color: rgb(72, 72, 72) !important;
    height: 100% !important;
    margin-bottom: 4px !important;
    box-shadow: rgba(228, 105, 35, 0.12) 0px 0px 12px !important;
    border-width: 1px !important;
```

```
    border-style: solid !important;
    border-color: #ffc107 !important;
    border-image: initial !important;
    border-radius: 4px !important;
  }
  .carpark-lots {
    display: flex;
    justify-content: center;
    flex-direction: column;
    align-items: flex-end;
    text-align: end;

    .lot-count {
      font-size: 1.5em;
      line-height: 1em;
      margin-bottom: 0.4em;
      font-weight: bold;
      &.high {
        color: #006647;
      }
      &.med {
        color: #cc860c;
      }
      &.low {
        color: #b50600;
      }
    }
  }

  .distance-info {
    align-items: flex-end;
  }
```

```
    .redirection-icon {
      margin-left: 6px;
    }
  }
}
```

Milestone 8: HTTPS

Application URL

Frontend: https://carina-parking.netlify.com/
Backend: https://nwjbrandon.com/swagger-ui.html/

Best Practices

One of the best practice is to use robust security certificate when enabling HTTPS for the website. The certificate is issued by a certificate authority (CA), and it will take steps to verify that the website address actually belongs to the correct organization. This is to help protect users from man-in-the-middle attacks. Another step in setting up the certificate to ensure a high level of security is by choosing a 2048-key instead of 1024-key. According to the General Number Field Sieve factoring algorithm, the 1024 bit key has around 80 bits of strength while a 2048 bits key has around 112 bits. This means that it takes a billion times longer to factor a 2048 key, making this method more secure.

The second best practice is to use a server-side 301 redirects. The 301 status code means that a page has moved to a new location permanently. The 301 redirect helps to make the transition from http to https seamless. Moreover, users can access the application through different URLs, and will always redirect to the correct URL with https prefixed. This ensures that users are always using the application over secure connections.

The third best practice is to verify that the website pages that are secured over HTTPS can be crawled and indexed by Google. The first step is not to block HTTPS pages in the robots.txt files. The second is not to include meta noindex tags in the HTTPS pages. The third step is to use the URL inspection tool in Google Webmaster to test whether Googlebot can crawl and index the pages. The Google Webmaster can provide a report to show whether the pages are secured or have any other potential issues. Besides, this practice also ensures that the SEO of the HTTPS website is still optimized.

The fourth best practice is to enabled HTTPS sites support for HSTS (HTTP Strict Transport Security). HSTS tells the browsers to request HTTPS pages automatically, even though the user fetch content using HTTP in the browser. Besides, it also tells Google to serve the secure HTTPS pages in Google search results. This ensures that users are using HTTPS to receive and send data.

Certificate Pinning

Certificate pinning is the process of associating a host with the expected public key and trusting the server's public certificate. Because the server-side and client side code are owned by the organization, the organization can configure the client to accept a specific certificate. This means that when the client returns back to the server, the client trust the server, with the specific certificate only, as the client remembers the server with that certificate.

One advantage of certificate pinning is that it ensure authenticity and confidentiality. Using certificate pinning means that it reduces the tendency of rely on trusting third parties when making security decisions with regards to identity. In the event where the certificate issuer is malicious, the issuer would need to steal the organization's private key to perform man-in-the-middle attack, but that is quite impossible because the attacker does not have access to the server private keys and certificates.

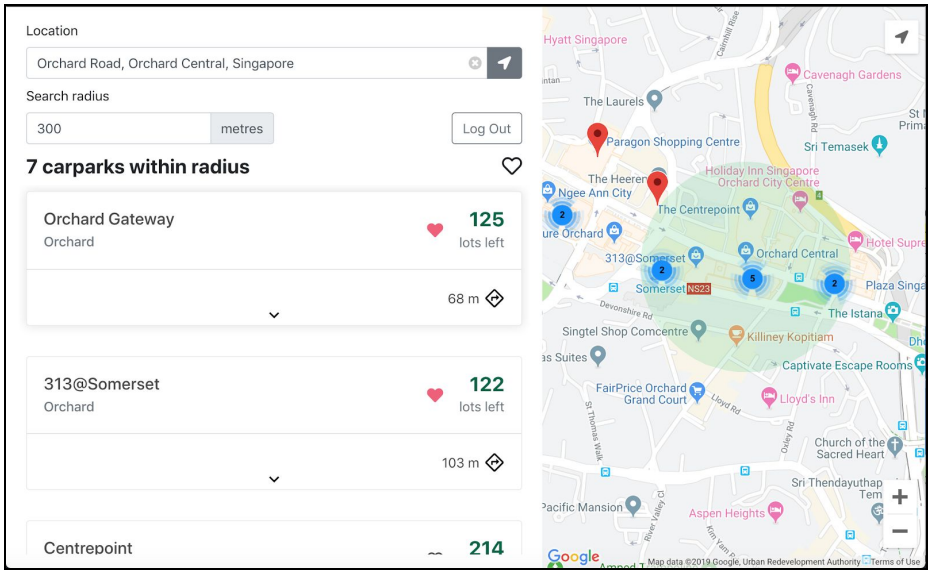
However, the disadvantage of using certificate pinning is that if the server change its certificate regularly, then the application would also need to be updated regularly. For example, if Google rotates its certificates, the application needs to need to updates as well. Currently, even though Google rotates its certificates, the underlying public keys remain static and the users do not need regularly updates the applications.

Certificate pinning will not be use in Carina. Most users generally do not want the hassle of regularly updating their application when they need to find an alternative carpark with available lots. This is because they are already frustrated with the fact that the carparks are full, and if the application were to irritate the users with updates, the users would not have a good user experience and may not rely on the application in future.

Milestone 9: Offline Functionality

Description

In offline mode, our app will retain a list of the available lots from all the carparks that can be found in LTA DataMall. This data is cached previously when the app was last connected to the internet.



Online mode

Offline mode

In other words, users will be able to use keywords (such as location name or carpark name) to search for carparks information. The app will then return a list of carparks that matches the keywords. The main information users will get out of this is the number of available lots during the period when the app was last online. Other supplementary information include the addresses of the carparks in that search region.

Additionally, if the user was previously logged in when the app was connected to the internet, our app will also retain the log-in status and cached the list of favourite carparks from that user. In offline mode, the user will then have access to the same list of favourite carparks. This provides convenience and allows the user to quickly look up on the last cached available lots for their favourite carparks.

Showing favourites in offline mode

However, the key selling point of our app is its online functionality. As such, there will be a notification at the bottom of the app when it is in offline mode so as to warn users that the data shown will be outdated. In short, this app is not suitable to be used offline for prolonged periods.

Implementation

To help the app decide on which page (online/offline) to render and show to the users, we add listeners to the main window to keep track of any change in the (network) state. Any changes detected will be handled according and the appropriate page will be rendered and shown to the users.

Furthermore, as the app is connected to the network, we will evaluate the cached data of the carparks in the local storage of the browser and update accordingly to ensure that the cached data is up-to-date in that device. This same principle is applied to the favourite carparks data every time there is a change in favourite carparks.

Scenarios and cases for offline functionality

1. Network connectivity is cut off while using the app
    a. Offline mode will be toggled and users can continue to view the offline data which was cached during the online session
2. Start up the app without network connectivity
    a. Offline mode will be toggled.
        i. If users have previously used the app in online mode, the cached data will be accurate as of the last online session
        ii. If users have **never** used the app in online mode before, then there will be no carpark data available to the user (very unlikely)
        iii. If users have previously used the app in online mode but they have cleared the cache/browser local storage, then there will also be no carpark data available to the user
3. Previous online session was logged-in
    a. Users can continue to view the list of favourite carparks, if any, in offline mode
    b. If cache is cleared, then there will be no data on the favourite carparks

Milestone 10: Authentication

Token-based vs Session-based

In token based authentication, the user state is stored on the client. The user state is encrypted into a JSON Web Token (JWT) with a secret and sent to the client. The JWT is stored in the localStorage and sent as a header for every request. The server validates the JWT before processing the request.

In session based authentication, the server will create a session for the user after logging in. The session ID is stored as a cookie on the client. The cookie would be sent per request. The server compares the cookie's session id against the memory's session information to verify the user's identity before processing the request.

Stateless, Scalable

The advantage of using token-based over session-based is that the token-based is stateless and scalable. Unlike session-based where session is stored in memory, the back-end does not need to keep a record of tokens in token-based. Each token is self-contained, and the server's only job is to sign or validate tokens on successful login request and verify incoming tokens are valid. Storing sessions in memory may become unscalable when there is a huge number of users using the system simultaneously.

Cross Domain and CORS

Another advantage of using token-based over session-based is that the token-based works well across different domains. In session-based, cookies and the session id are scoped to a specific domain; therefore bound to having all the servers under one domain. This is because the session id will only make sense to those servers who have access to the server side session management. On the other hand, the token-based with CORS enabled allows easy exposure of APIs to different services and domains. This is because as long as there is a valid token, the requests can be processed.

Performance

An additional advantage of using token-based over session-based is that token-based performs the authentication faster and more efficiently. In session-based, the back-end needs to lookup the database for the user information before validating the user's identity. This authentication approach takes longer compared to decoding a token. Additionally, since JWT is flexible and can store additional data, such as the user 2FA enabled status, and level of permission, the back-end does not need to perform additional query to process the requested data.

CSRF Security

Another important of using token-based over session-based is that token-based is slightly more secure than session-based. In terms of CSRF, cookies are dangerous because they are added the request when triggered, regardless of whether the request came from a rogue code. The requests are passed to the server, and CSRF attacks are easily carried out. In token-based, it is still possible to make CSRF with tokens, however, the rogue code must get hold of the token and explicitly add into the header of each request every time the request is triggered. Therefore, the token-based approach is more CSRF-safe than the session-based approach.

Token and Session ID Size

However, one disadvantage of using token-based authentication is that the size of JWT is much bigger than the session id stored in cookie. This is because JWT contains more user information. Therefore, when the user data is encoded, the JWT becomes longer and that is stored on the client side.

Token-based for Carina

Carina's authentication is based on token-based approach. The user base for Carina is expected to be large as there are many drivers in Singapore. Therefore, to reduce the amount of memory usage and to increase performance in processing request, Carina used token-based authentication to accommodate the large number of users using Carina's services. Moreover, to secure the user information, such as their favourite carparks, a token-based authentication is preferred over session-based so that hackers cannot find out the whereabouts of the users.

Milestone 11: Choice of Framework

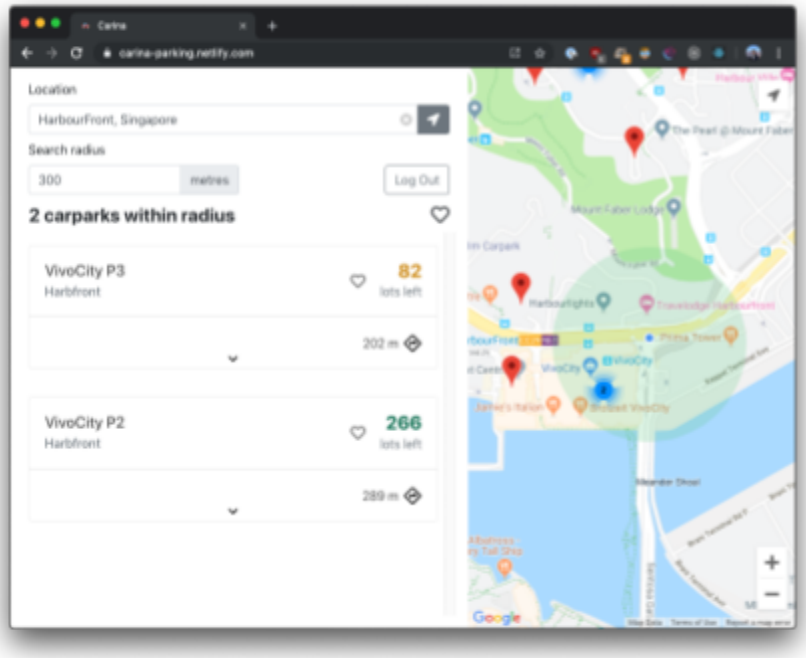
Explanation for Framework.

React.js was used along with TypeScript. React.js was used over other web frameworks such as Vue or Angular due to the relative familiarity the group had with React.js over the other frameworks. Angular was not even considered due to it being overkill for a simple application like Carina. React also had a few important pros over Vue, such as the ease of creating a Progressive Web App (PWA) as needed for this assignment due to `create-react-app` library, better syntax for writing TypeScript as compared to Vue.js, and the much richer package ecosystem as compared to Vue.js.

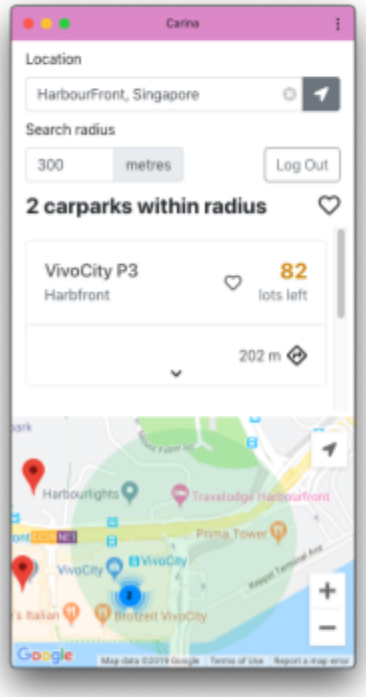
TypeScript was chosen over JavaScript due to its static typing feature, allowing us to catch errors during compile time, reducing massively the amount of runtime bugs that may occur.

Optimize your entire site for mobile

The website is optimized for mobile usage, changing the layouts to better suit the limited size of the screen.



Desktop view



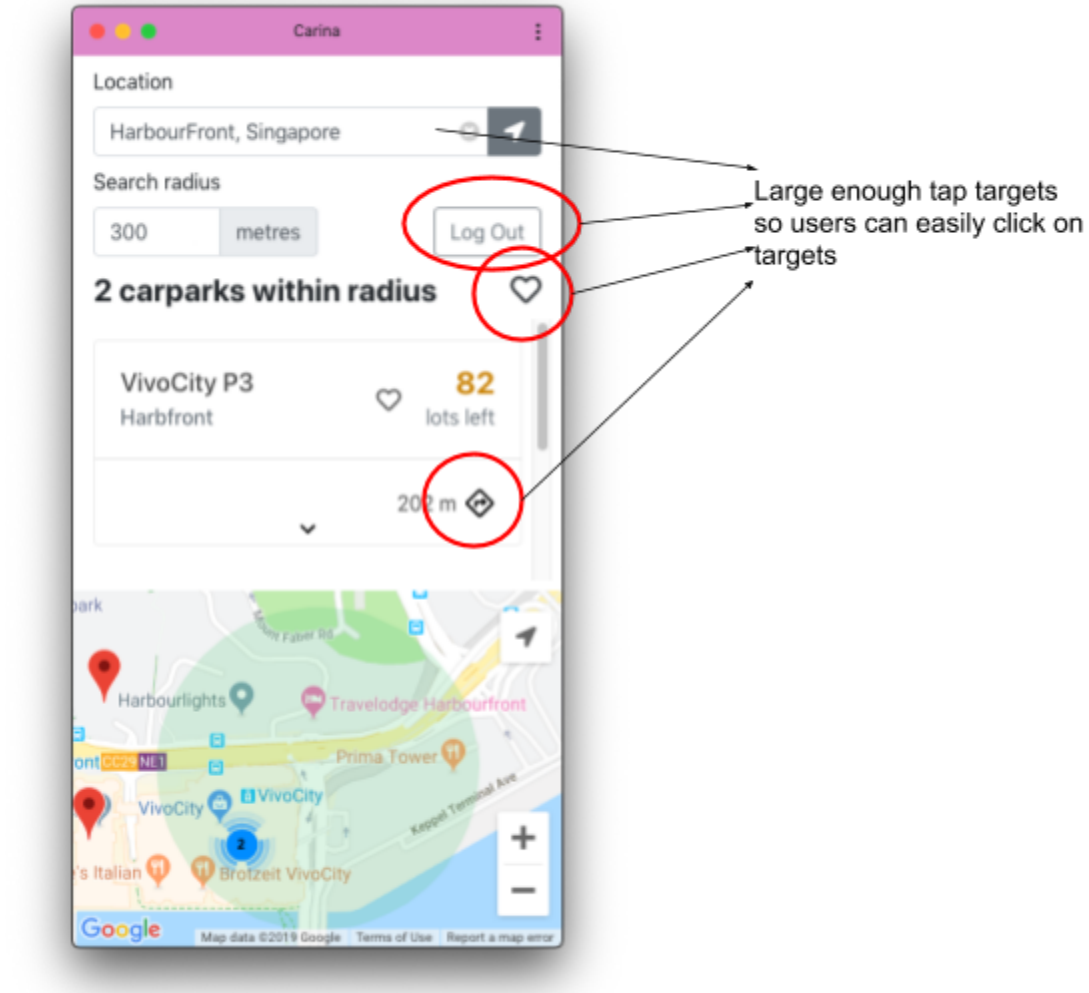
Mobile view

Don't make users pinch to zoom

Being mobile optimized also means that users do not have to pinch to zoom to look at details just because the font is too small to read.

Finger friendly touch targets

Our action buttons are also sized appropriately to allow users to easily click on targets with their fingers.

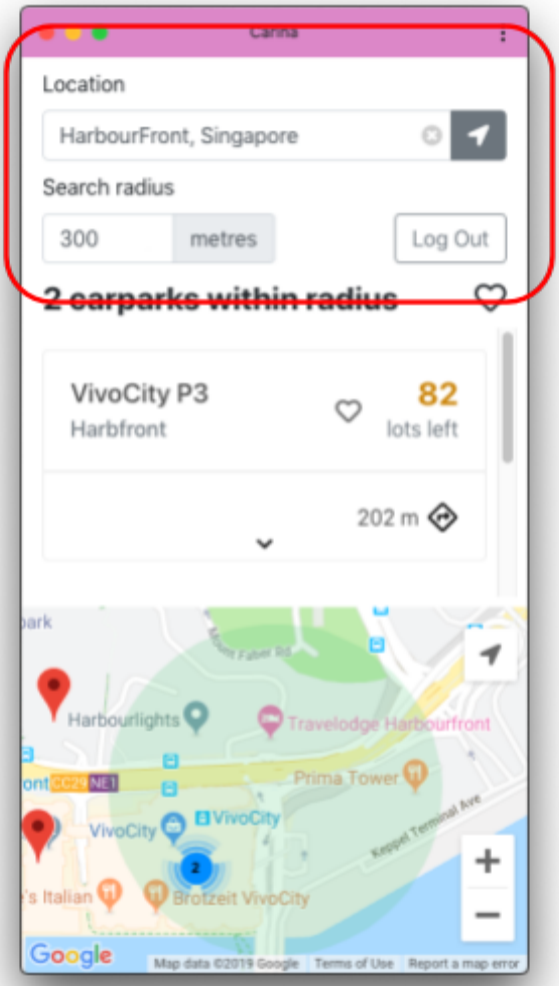


Keep your user in a single browser window

Users do not have to face cumbersome white screens from navigating away from the application (besides logging in or getting directions from our application using Google Maps.).

Keep calls-to-action front and center

The whole point of our application is to search for the best carparks given a location. Thus, a significant portion of the screen's real estate is dedicated to that very action.


Location
HarbourFront, Singapore
Search radius
300
metres
Log Out
2 carparks within radius
VivoCity P3
Harbfront
82
lots left
202 m
Harbourlights
Travelodge Harbourfront
Prima Tower
VivoCity
VivoCity
Brotzeit VivoCity
Google
Map data ©2019 Google
Terms of Use
Report a map error

Milestone 12: Workflows

Common Workflows

Running the application on the same page

Carina is designed such that all the features are implemented on the same page. In other words, the users do not need to navigate to another url or visit another page for another service.

From the user perspective, the user will most likely use the application the instant he is unable to find a parking lot. Therefore, the user is only concerned with finding the next available parking lots when using the application. Therefore, when the user first loads the application, the user expects to be able to immediately start searching for the nearest parking lot with available lots. The map on the side page shows the user's current location and markers of all nearby carparks within his vicinity. The map also provides a visual representation of the carparks allowing the user to select the carpark to get more information about it.

All these services are integrated into a single page; therefore, the users can seamlessly check for available parking lots without having to navigate to another page for the information.

Scrollable list of nearest carparks

When the user searches for nearest carpark lots in his vicinity, the user will receive a list of carparks at the bottom of the search bar. The user will have to scroll down for more carparks instead of swiping horizontally for more carpark.

Originally, the list of nearest carparks were implemented such that the user have to swipe horizontally to look at the next carpark. However, the first few users who tested Carina always swipe downwards instead of doing it vertically. The reason that they provided was that they naturally thought they had to swipe down to view more carparks. Besides that, from the user perspective, the user wants to quickly see the name of all the nearest carpark lots and the number of available lots next to it. A vertically-scrollable list would help the user to scroll through the entire list of carparks to find the name of carparks that they are familiar with.

Therefore, when searching for the nearest carpark lots in the list, the user can scroll through the list quickly to look at the names and the number of parking lots to select his alternative choice of carpark.

Auto-Scrolling upon mark clicked

After the users search for the search carparks, there could be a long list of carparks to choose from. On the map, there are markers to indicate the various locations of the nearest carparks. If the user click on the marker, the application will automatically scroll to the selected carpark.

When the first few users tested our application, they often click on the marker to identify the name of the carpark lot and the available lots. After which, they would scroll through the list of available lots to find more information about the carpark. That took them some time to find the correct carpark. To improve the user experience, the application was designed such that when the user clicked on the marker, the application will automatically scroll to that carpark lot to help the user find more information quickly about the parking lot.

Accounting for the user behaviour in our testing phase and to facilitate easy search of the carpark information, Carina includes an auto-scroll feature bring the information of the carpark to the user. The user does not have to scroll through the list of the nearest carpark which would have taken some time.

Searching for carparks in a region

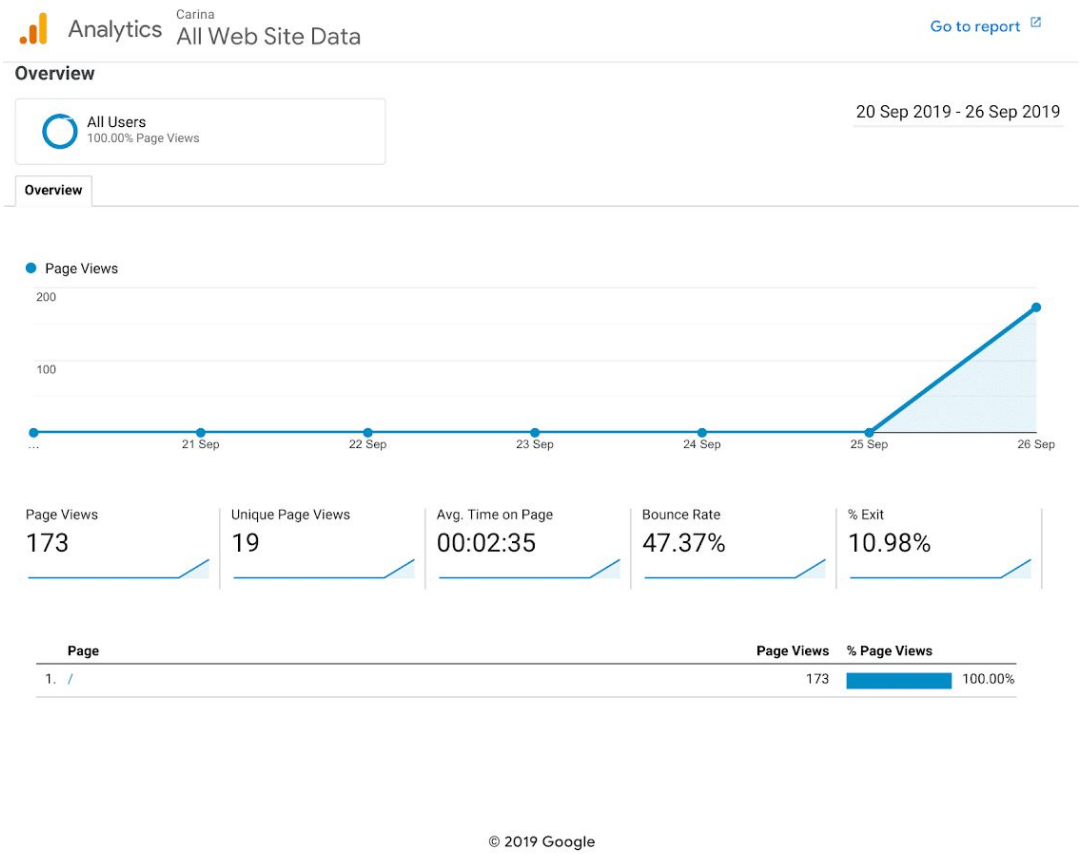
Carina not only provides the ability for users to find the nearest carparks at his current location, but it also provides the ability for users to find the nearest carparks at other locations as well. But instead of typing in the latitude and longitude, the user only needs to specify the radius and the name of the location in the search bar at the top of the screen.

One of the users who tested our application mentioned how he wanted to search for the nearest carpark lot in a certain area. He gave the example in which he is going for an interview, and he would like to find out information about the carparks around his interview sites. Carina was later further developed where the users can also search for the carparks by the area's name. Any other users who needs to attend events and functions, and needs to search for carparks near these areas would find this functionality useful.

To search for carparks in a region, the users can provide the area name to fetch a list of nearest carparks at their destination.

Milestone 13: Google Analytics

Google Analytics



Analytics Carina All Web Site Data

Go to report

Audience Overview

All Users
100.00% Users

20 Sep 2019 - 26 Sep 2019

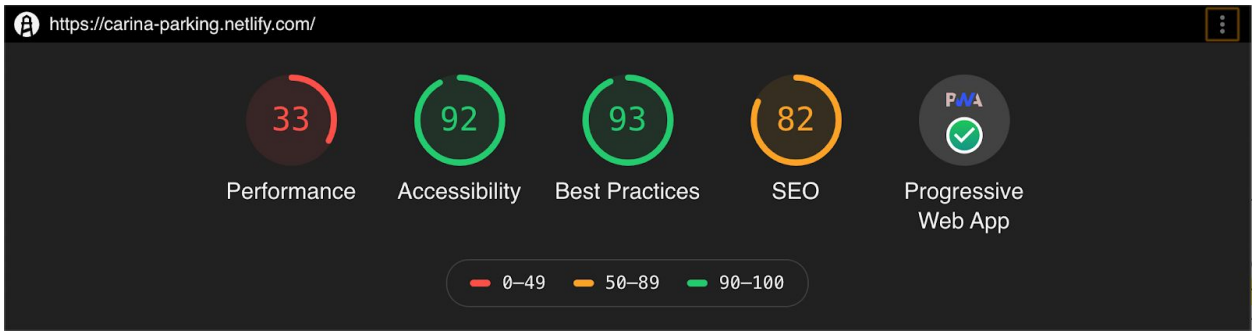
Overview

Users 14	New Users 14	Sessions 19
Number of Sessions per User 1.36	Page Views 173	Pages/Session 9.11
Avg. Session Duration 00:20:52	Bounce Rate 47.37%	

	Language	Users	% Users
1.	en-gb	9	64.29%
2.	en-sg	3	21.43%
3.	en-us	2	14.29%

Milestone 14: Lighthouse

Lighthouse Report



(Some points were docked in Accessibility and SEO due to using Google Maps in the application, where the rendered map had instances of accessibility violations.)

Milestone 15: Social Network

Social Plugins

Carina supports user signup and user login on gmail. The reason for enabling gmail login on Carina is to facilitate seamless user experience of the application. The purpose of the user info is to store the user's favourite carparks so that the user can check on the number of available lots easily.

Gmail was chosen because almost everyone in Singapore has a gmail account. Therefore, enabling gmail login facilitate most users on Carina to login with their gmail account. Naturally, most users will sign in with Gmail even when other providers are enabled. Hence, Gmail is the main mode of user login.

Milestone 16: Geolocation API

Geolocation

Carina allows users to find available parking lots based on the user's current location. Therefore, Carina uses the Geolocation API in order to obtain the user latitude and longitude to fire a GET request to the backend. The user will receive a list of all the nearest carpark lots within the vicinity.