CS3216

Assignment 3

Final Report

Cheferyone

Because everyone can be a chef

Group 1:

1. Christian James Welly (A0188493L)
2. Mario Lorenzo (A0193781U)
3. Otto Alexander Sutianto (A0184556U)
4. Nelson Tan Kok Yi (A0183703H)
5. Yehezkiel Raymundo Theodoroes (A0184595M)

Table of Contents

Milestone 0

Describe the problem that your application solves

During the pandemic, most people stayed at home and some of them lost their jobs due to the recession. In order to cover their losses, a sizable number of them started opening small businesses from home, and food is one of the most popular choices. home chefs usually start their businesses by opening pre-orders, and promoting it through word-of-mouth, or doing social media promotions such as through Facebook, Instagram stories, etc.

However, it is painful to start this first step, since most of the buyers are only their friends or Instagram followers, and thus they would have relatively low outreach. Therefore, Cheferyone attempts to solve home chefs' pain points by providing a centralized platform for them to showcase their cooking and open pre-orders to a wider audience. Using Cheferyone would expand the potential buyers by some margin since now strangers can notice the home chefs' cooking and may want to try their food if they are interested.

Milestone 1

Describe your application and explain how you intend to exploit the characteristics of mobile cloud computing to achieve your application's objectives, i.e. why does it make most sense to implement your application as a mobile cloud application?

Cheferyone allows home chefs to promote their food, and they can also open pre-orders, so that the home chefs can manage the orders easier. Cheferyone is also an app for those who like to taste different kinds of food! Cheferyone provides tagging features for menus and chefs with specialised cuisines so that users can discover them easily. There is little need to worry about the credibility of the chefs and their cooking since Cheferyone allows the buyers to review the menus once their orders are completed.

The first reason why Cheferyone is served better as a mobile cloud application is because it is device-independent. By having it available in multiple platforms at the same time, Cheferyone can have a faster development cycle. More importantly, the web interface can be used by home chefs to easily set up and manage their menus, especially when it gets increasingly large. The mobile aspect makes it enticing for users to use the application as it makes the application more accessible. This is helpful for the home chefs to have access to a wider-range of buyers.

The second reason is regarding the availability and productivity. For mobile cloud applications, information is synced via accounts, allowing frictionless migration of devices, or usage of multiple devices to manage one account at once. The data such as the collection of food after providing proof of payment are served by the central server, rather than being stored in a device locally. Users need not worry about losing their subscription to a particular home chef just because they switched devices.

The final reason why mobile cloud application is suitable for Cheferyone is because mobile cloud application requires low hardware capability, since almost everything is done in the server side. Cheferyone makes use of several computationally-intensive features such as recommendation systems, which are not suitable if done locally. Low hardware capability also makes the application more accessible to a wider range of audience, since users on lower-end devices will not be discriminated against.

Milestone 2

Describe your target users. Explain how you plan to promote your application to attract your target users.

There are two main target users that are covered by Cheferyone. The first target users are home chefs. We have conducted interviews with a few people who are home chefs and occasionally open pre-orders for several reasons, including earning extra money, or simply as a hobby. According to their feedback, they feel that most of their buyers are their friends, and they suspect that the main factor of their friends willing to buy their products is more about supporting the products rather than the fact that they might enjoy the actual taste of their products. Therefore, other than expanding their market to earn more money, they feel that an application like Cheferyone is important, especially when there is a review column which they can use to improve the quality of their products from the past buyers, and having more customers who would order the food for the taste.

The second target users are people who love to try different kinds of food. This can be regional cuisines, or perhaps even fusions. Currently, such people can only scroll through social media to find listings of a possibly-interesting menu, which can be very painful. Cheferyone provides a centralised platform for food hunters to seek for more kinds of non-mainstream foods based on their desires. In particular, the tag feature allows the users to specify what they are craving.

The first and simplest way to promote our app is by directly promoting on social media. For example, we can join Facebook groups with foodies inside, and by promoting the app, we can reach out to groups of people who like to hunt for pre-orders on social media. The promotion can be done through showing videos of what the application is about or even just a description of the application. Cheferyone will particularly be attractive to users who have had experiences of scrolling through social media to discover new menus yet discover nothing.

We could also reach out to influencers to promote and try our application. We can try targeting influencers who have done food reviews as part of their brand. For example, Night

Owl Cinematics which runs the Food King series. An incentive is that they would be able to try out some of the cooking of the home chefs, providing them with some possible content for a new video. Moreover, not only will our application get some publicity, the home chef will get some publicity as well. Asking for their help might come at some cost, and in this scenario it might be worth to make a cost-benefit analysis to gauge whether it is worth it.

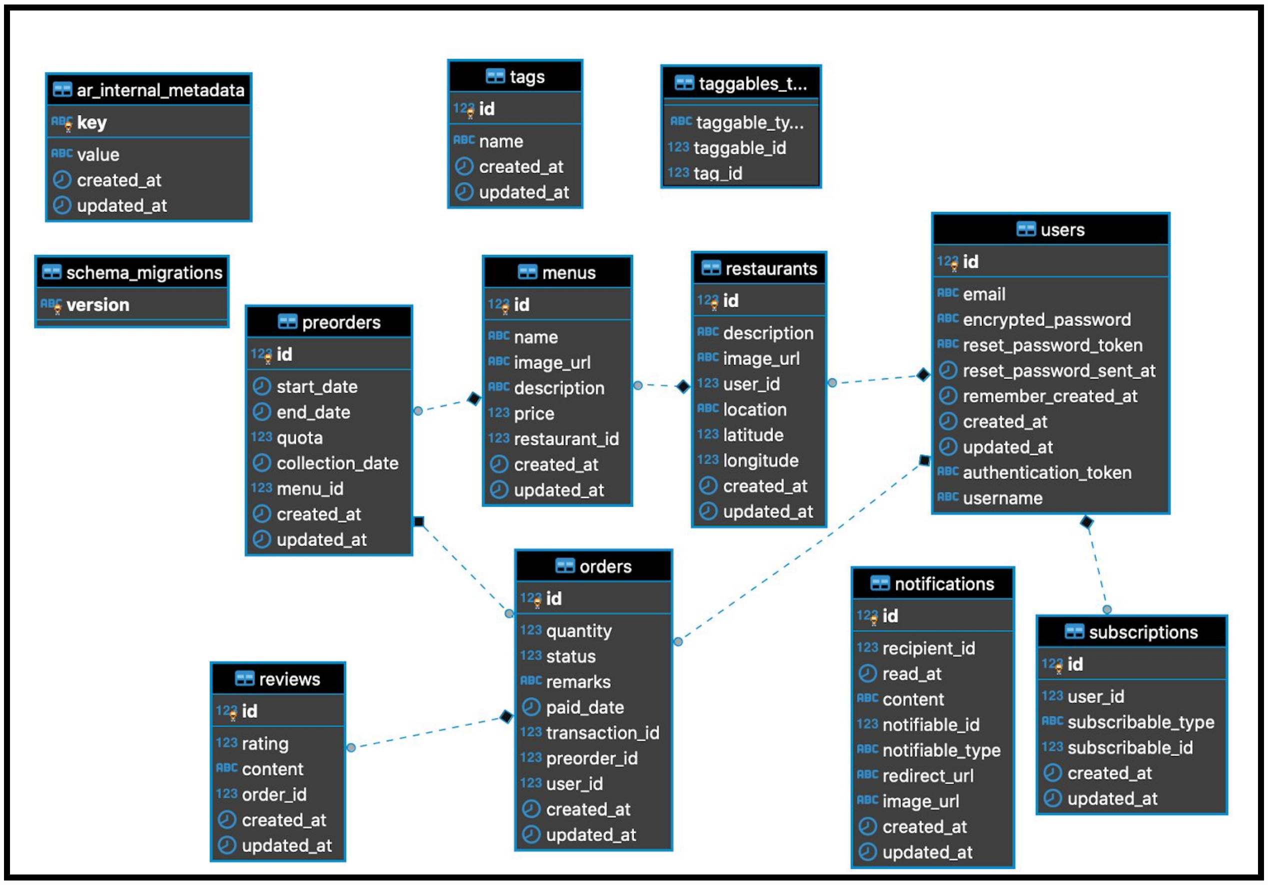
Finally, we think that this application can be promoted naturally by the home chefs as well. We could ask them to try their existing way of promoting in social media, and on top of it they can link their profiles on Cheferyone as well. For example, using the 'Bio' feature in their Instagram profiles. This helps promote the application while at the same time migrating the existing customer following from the social media to use Cheferyone.

Milestone 3

Draw an Entity-Relationship diagram for your database schema.

Overall ER Diagram

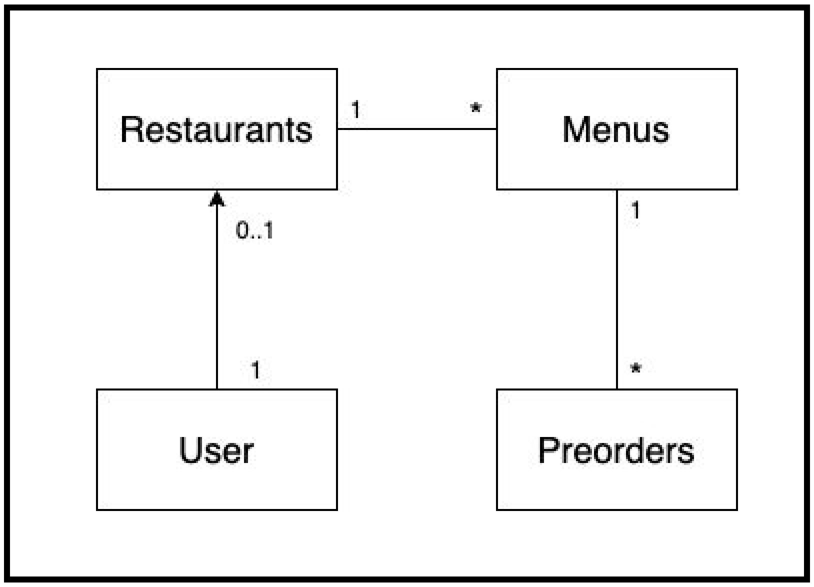
Following is the full ER Diagram, that is auto generated through Dbeaver software.



This Diagram shows us a comprehensive list of all fields and associations from each table along with its association. Most of the tables would have an *id* column as its primary key, except for the associations table, such as *taggables_tags* that associates between taggable records and tag records.

In order to get a better understanding of the relationship, we will separate the diagram into smaller diagrams in the next few sections.

Home Chef and Preorder Associations



Users

In Cheferyone, a user can both be a home chef or a consumer. As a result, all users may (or may not) have an association with a restaurant. Users that are associated with a restaurant will then be recognized as home chefs within Cheferyone.

For the MVP, we put a key constraint between user and restaurant, where a user would only have at most one restaurant. This is in line with our key finding that most of the home chefs only manage a single brand for advertising their foods.

Restaurants

As mentioned before, restaurant tables have a one-to-one relationship with users. In the implementation, the restaurant would have a **foreign key** towards the users table, called the *user_id* column.

Within their own restaurant, a home chef would then be able to manage a number of menus. In other words, a one-to-many relationships with menus tabl.e

Menus

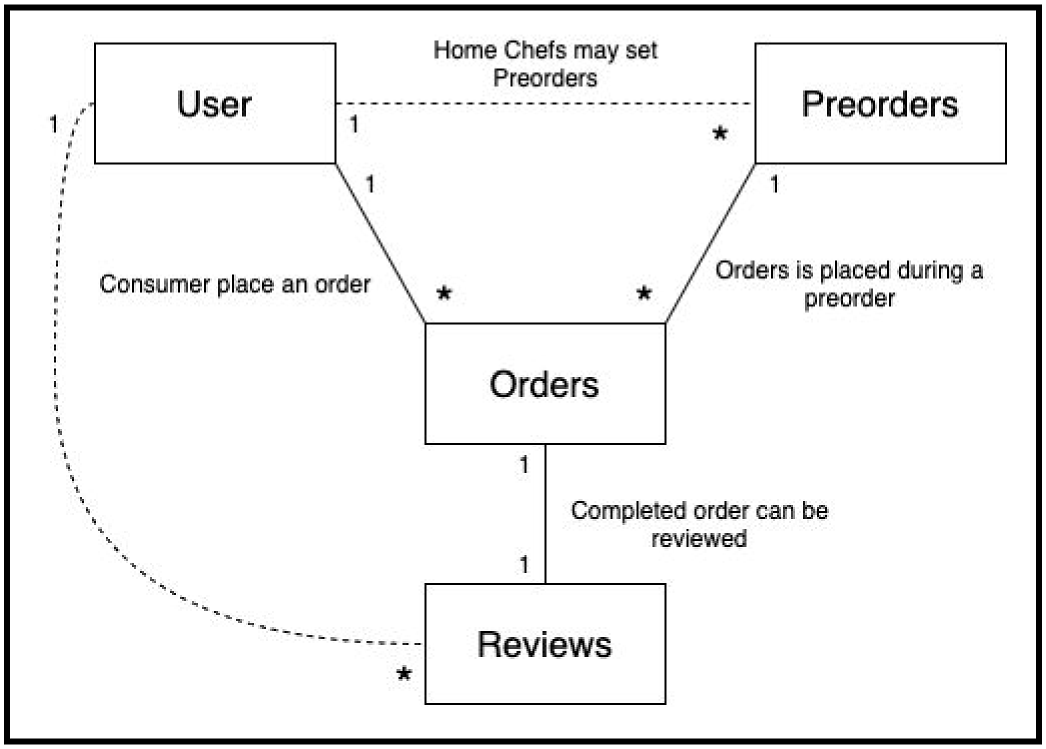
The menu table will be holding the foreign key of restaurants through the restaurant_id column.

Preorders

Before an order can be placed, home chefs have to open a pre-order. This is due to the nature of home chefs where they manage orders in a batch-by-batch manner.

Preorder tables would contain several informations of the menu, including *start_date*, *end_date, quota,* and *collection_date*

Consumer and Order Associations


User
Preorders
Orders
Reviews
Home Chefs may set Preorders
Consumer place an order
Orders is placed during a preorder
Completed order can be reviewed
1
*
1
*
1
*
1
1
1
*

Orders

After a preorder has been scheduled by a home chef, which is also a user from users table, consumers can start making an order towards the preorder.

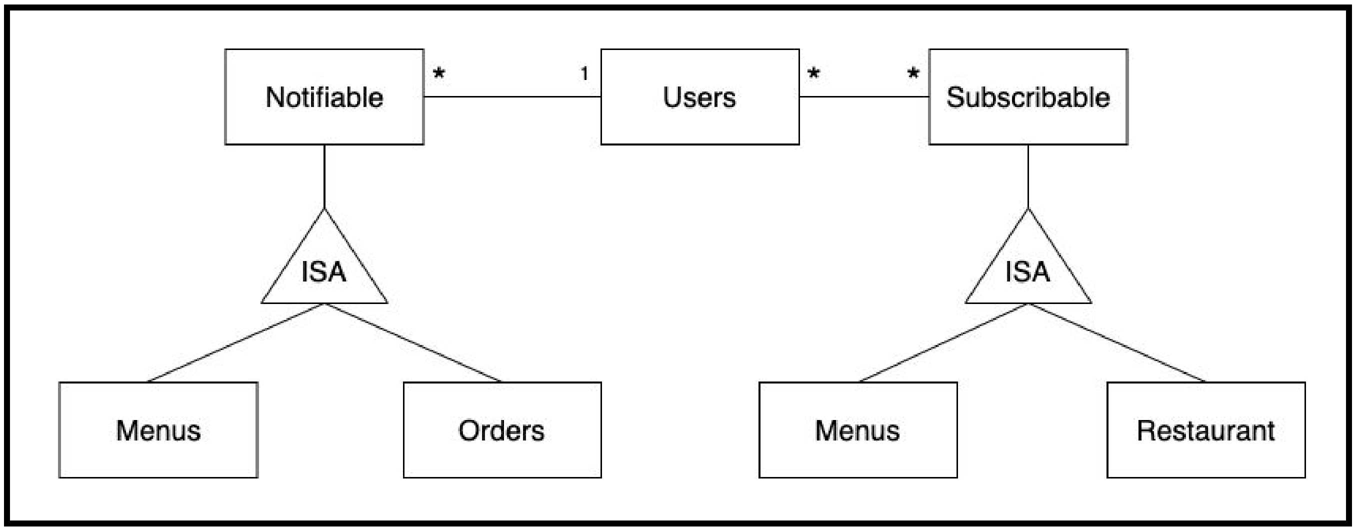
Reviews

Once the order is completed and collected, user can then submit a review, which will be part of our rating systems

Subscribe and Notification Associations

Subscription and notification, is a pair of features that would allow our home chefs to grow their customer base by allowing them to have recurring customers.

Whenever a customer enjoys the home chef menus, they can either subscribe to the menus or the home chefs. By doing so, customers would then be notified with the latest updates of the home chefs including the updates when the next pre-order has just been scheduled.



Subscribable

As Cheferyone allows customers to subscribe to a restaurant or just a specific menu, we implement them as polymorphic associations

Subscriptions table will have *(subscribable_type, subscribable_id)* columns as the foreign key towards the subscribable records. In addition, the table would also have *user_id* as the foreign key towards the users records.

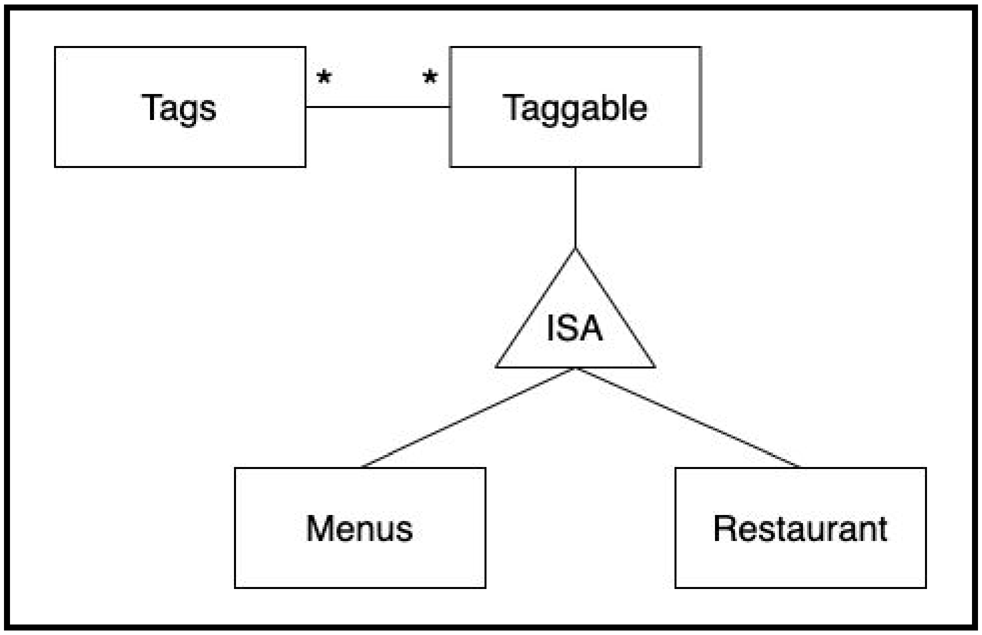
Notifiable

Through subscriptions, users will obtain personalized notifications based on their liking.

Notifications are also implemented similary as polymorphic associations. In MVP, users might get notified because of a menu update or an order update. As a result, both models will be considered as notifiable objects.

Tag Associations

Lastly, the taggings record



Taggable

As our promise to deliver discoverability to the home chefs, we are implementing taggings along with its filtering.

Not only, menus can be tagged but also their restaurant. Therefore, they are again implemented as polymorphic associations with the Tags model.

Milestone 4

Explore one alternative to REST API (may or may not be from the list above). Give a comparison of the chosen alternative against REST (pros and cons, context of use, etc). Between REST and your chosen alternative, identify which might be more appropriate for the application you are building for this project. Explain your choice.

We did some research on REST API and GraphQL as an alternative. REST API works by differentiating requests based on the resource path and its method verb, while GraphQL works by differentiating requests based on its query content. GraphQL organizes the data into a graph, where the objects and their relations are represented as nodes and edges respectively.

The first main advantage of REST API over GraphQL is that REST API has become the industry-standard and adopted by many frameworks, such as Rails which is used by Cheferyone. It is thus widely used, and there is more support for REST API on the web compared to GraphQL.

GraphQL also has a steeper learning curve compared to REST API due to the simplicity of REST API.

Single Responsibility Principle (SRP) is adopted in designing the API because one endpoint is responsible for doing one task, whereas various information can be dumped into a single namespace for GraphQL.

On the other hand, the advantage of GraphQL over REST API is that it can add new fields and types without affecting existing queries, which leads to cleaner and easier-to-maintain server code. Another reason to choose GraphQL over REST API is that the querying process for a specific information is faster compared to REST API since GraphQL only fetches the data that we need, whereas REST API returns us a complete dataset. Using GraphQL also helps us reduce the number of useless endpoints, since there are multiple useless endpoints in REST API, where there are lots of duplications, and thus violating Don't Repeat Yourself (DRY) principle.

Based on the factors above, we decided to go ahead with REST API over GraphQL because it is adopted by the framework that we are using in this project, which is Rails. Moreover, due to the time constraint in this project, REST API helps us to move forward faster since the learning curve is not as steep, and there is more support available on the internet.

Milestone 5

Design and document all your REST API. If you already use Apiary to collaborate within your team, you can simply submit an Apiary link. The documentation should describe the requests in terms of the triplet mentioned above. Do provide us with an explanation on the purpose of each request for reference. Also, explain how your API conforms to the REST principles and why you have chosen to ignore certain practices (if any). You will be penalized if your design violates principles with no good reasons.

The link to the Apiary documentation is available at: https://makan2.docs.apiary.io/.

The following REST principles are written with reference to: https://restfulapi.net/rest-architectural-constraints/

Rails Support

As part of the industry practice, REST itself has been adopted in many frameworks. The good news is, RESTful routing is supported well within Rails Framework[1]. This is indeed our main consideration in using rails as our backend.

[1] RESTful JSON Apis with Rails: https://guides.rubyonrails.org/api_app.html

Uniform Interface

A characteristic statement for this principle is that ‘Any single resource should not be too large and contain each and everything in its representation’. We have tried to conform to this principle for most of our APIs, however there is a small subset which violates this. In particular, the Menus collection (getting a menu item via a GET request). The JSON returned for getting a Menu item also returns the JSON of pre-orders, an entity that the Menu is associated to.

Firstly, Menu has many Preorders. In order to list all the Preorders that a Menu has in the Menu page, we minimally require all the Preorder IDs. A choice we have made for the frontend is that the Menu should be able to update all its Pr-orders together in a single ‘submit’ action. To do this, we are required to load the details of every upcoming Preorders as well as the currently-open Preorder.

This leaves us with the options of:

1. Implementing a separate API to get a list of the future pre-orders (this is not all the pre-orders associated with the menu)
2. Returning the complete JSON objects similar to (1) in a single response.

We chose the second option, as this reduces the number of API calls by two in the frontend (because the menu page needs a list of pre-orders and a single current pre-order). Moreover, we do not need to have a separate page to show a single pre-order nor a separate page to edit the pre-order, as we are performing these changes together with the updates in a Menu. Therefore, we think that implementing an APIs for the Preorder collection might not be a good approach either.

Ultimately, we are making a tradeoff between violating REST principles and user experience. By choosing to have a button that updates both the Menu and all its associated Preorders, we are providing a better User Experience, at the cost of violating some principles. As we are still building a Minimum Viable Product, this JSON response could be optimised further in our future iterations. Currently, we favour some tradeoffs over the other as we are prioritizing a faster development cycle.

Client-Server

The backend of our application is done using the API-only Rails and the frontend is built in ReactJS. They were built independently after defining the interface which was documented in Apiary. We think that whatever we have done conforms to the principles, as the frontend is only required to know the interface and the format without having awareness about the development of the backend

Stateless

Our client-server interactions are all stateless. The only state that the server will ever need is the information about the currently logged in user. Nonetheless, we managed to handle this well by keeping it in our session storage and sending them along with each request. Other than that, no extra information is sent with each request.

Cacheable

Caching of resources is supported well at our front-end through service workers. This is inline with our ideas when we are designing Cheferyone as a Progressive Web App.

Milestone 6

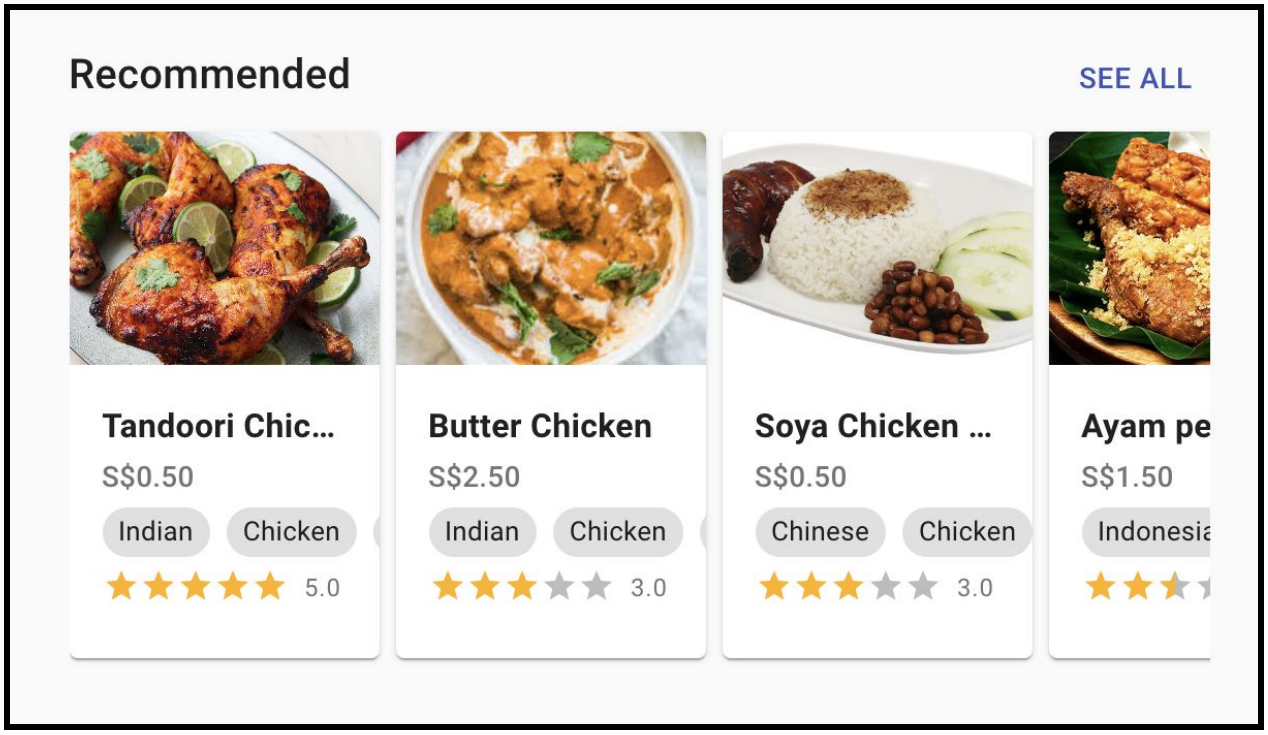
Share with us some queries (at least 3) in your application that require database access. Provide the *actual SQL queries* you use (if you are using an ORM, find out the underlying query and provide both the ORM query and the underlying SQL query). Explain what the query is supposed to be doing.

By default, Rails, which is our choice of backend, uses an ORM instead of a plain SQL query. However, we are mixing the ORM and plain SQL queries in many places within our implementation to optimise for performance.

The following are our top 3 most interesting queries within our application.

Personalized Recommendation API

At the homepage, we are displaying to each user a recommendation list of foods based on their history of orders.



The algorithm works the following way:

1. **Find the top 3 food categories** or tags that this user has ordered before. This is going to be done using a single query, that will be shown in this section. If the user has never made any orders, we will randomly choose their top 3 food categories.

2. **Filter all menus based on these tags** and sort them based on their review ratings in a decreasing order. We are doing this while keeping pagination in mind. In addition, this will take another SQL query and is going to be explained in the next section.

Overall, the recommendation algorithm would only take two SQL queries.

Query Usage

The query aims to find the top 3 favourite tags based on a user history of orders. Following are the examples of tags produced.

```
["Halal", "Chicken", "Indian"]
```

ORM Query

```
User.limit(3).includes(orders: { preorder: { menu: :tags }})
    .where(id: current_user.id, orders: { status: "completed"})
    .group("tags.name")
    .order("sum(orders.quantity) desc")
    .pluck('tags.name as favourite')
```

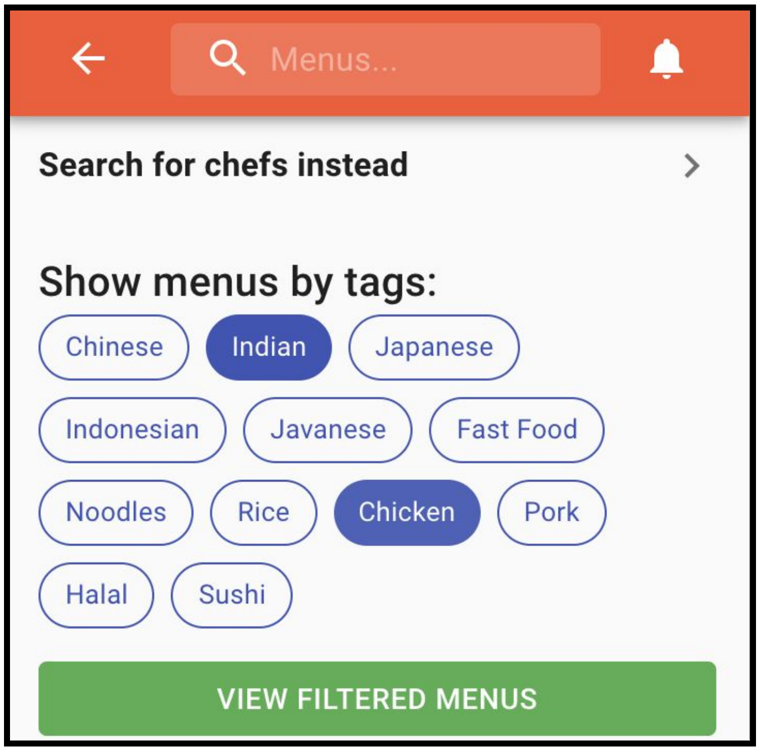
SQL Query

The ORM query will be translated into the following query

```
SELECT tags.name as favourite
FROM "users"
LEFT OUTER JOIN "orders" ON "orders"."user_id" = "users"."id"
LEFT OUTER JOIN "preorders" ON "preorders"."id" = "orders"."preorder_id"
LEFT OUTER JOIN "menus" ON "menus"."id" = "preorders"."menu_id"
LEFT OUTER JOIN "taggables_tags" ON "taggables_tags"."taggable_type" = $1 AND
"taggables_tags"."taggable_id" = "menus"."id"
LEFT OUTER JOIN "tags" ON "tags"."id" = "taggables_tags"."tag_id"
WHERE "users"."id" = $2 AND "orders"."status" = $3
GROUP BY tags.name
ORDER BY sum(orders.quantity) desc LIMIT $4
[["taggable_type", "Menu"], ["id", 8], ["status", 3], ["LIMIT", 3]]
```

Multi-Tag Filtering

Cheferyone utilizes powerful tagging filtering in order to promote discoverability for our Home Chefs.



Both menus and home chef can be filtered with tags and to clarify further, multi-tag filtering will provide you with objects that are categorized in at least one of these tags.

As mentioned before, mult-tag filtering itself is part of the recommendation algorithm. Therefore, we are going to elaborate their use-cases inside the recommendation algorithm.

Query Usage

As part of the recommendation algorithm, we are going to filter a set of menus based on tags and sort them based on their rating in a decreasing order.

ORM Query

```
Menu.joins(:tags).left_outer_joins(:reviews)
    .where(tags: { name: list_of_tags })
    .group(:id).order('avg(rating) desc')
    .limit(@limit).offset(@offset)
```

SQL Query

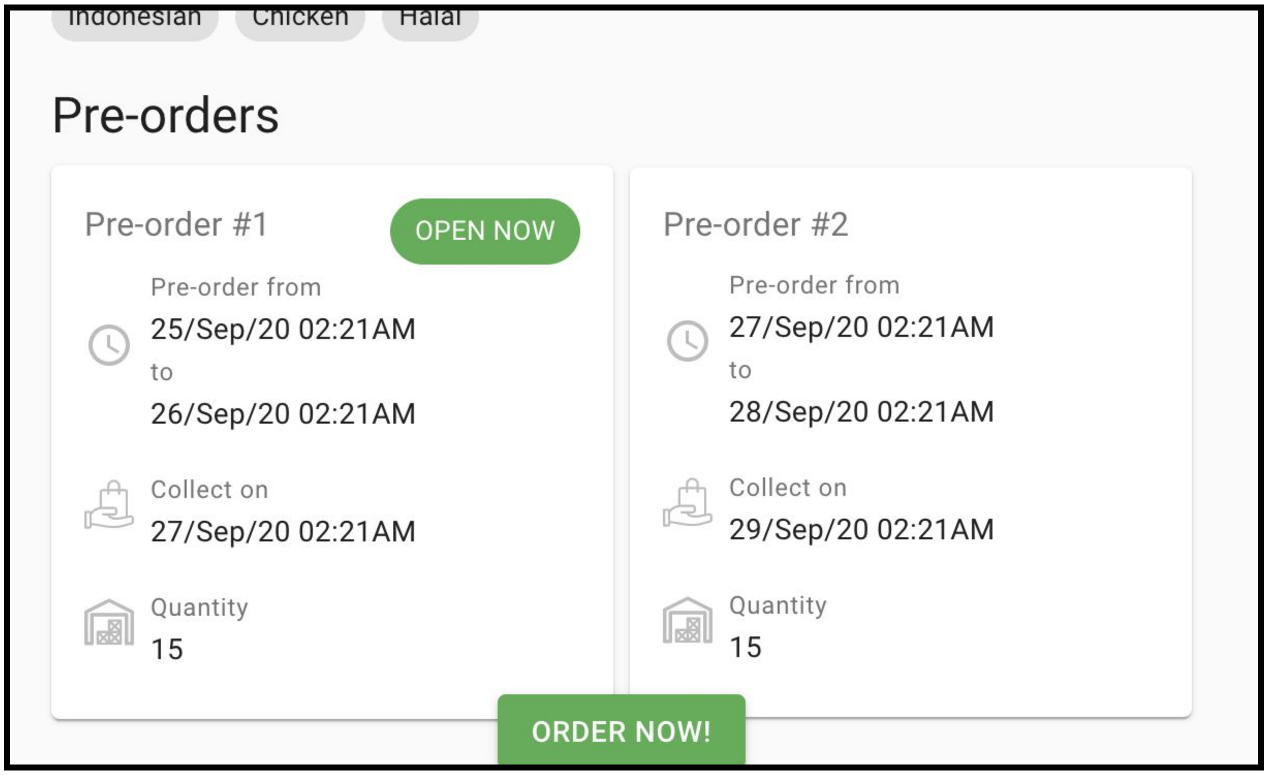
The ORM query will be translated into the following query

```
SELECT "menus".*
```

```
FROM "menus"
INNER JOIN "taggables_tags" ON "taggables_tags"."taggable_type" = $1 AND
"taggables_tags"."taggable_id" = "menus"."id"
INNER JOIN "tags" ON "tags"."id" = "taggables_tags"."tag_id"
LEFT OUTER JOIN "preorders" ON "preorders"."menu_id" = "menus"."id"
LEFT OUTER JOIN "orders" ON "orders"."preorder_id" = "preorders"."id"
LEFT OUTER JOIN "reviews" ON "reviews"."order_id" = "orders"."id"
WHERE "tags"."name" IN ($2, $3, $4)
GROUP BY "menus"."id"
ORDER BY avg(rating) desc LIMIT $5
[["taggable_type", "Menu"], ["name", "Chinese"], ["name", "Indian"],
["name", "Rice"], ["LIMIT", 11]]
```

Current and Future Preorders

Home chefs tend to process orders in batches. To do so, they would open a certain time-period to state their stock and when the customers can make an order. This is what we call ‘pre-orders’. As a customer, I would like to see when the upcoming pre-orders will be scheduled.


Indonesian Chicken Halal

Pre-orders

Pre-order #1 OPEN NOW

Pre-order from
25/Sep/20 02:21AM
to
26/Sep/20 02:21AM

Collect on
27/Sep/20 02:21AM

Quantity
15

Pre-order #2

Pre-order from
27/Sep/20 02:21AM
to
28/Sep/20 02:21AM

Collect on
29/Sep/20 02:21AM

Quantity
15

ORDER NOW!

Query Usage

We are going to query all pre-orders of a certain menu and exclude those that have expired.

ORM Query

```
menu.preorders
    .where("preorders.end_date >= ? ", DateTime.now)
    .order(:start_date)
```

SQL Query

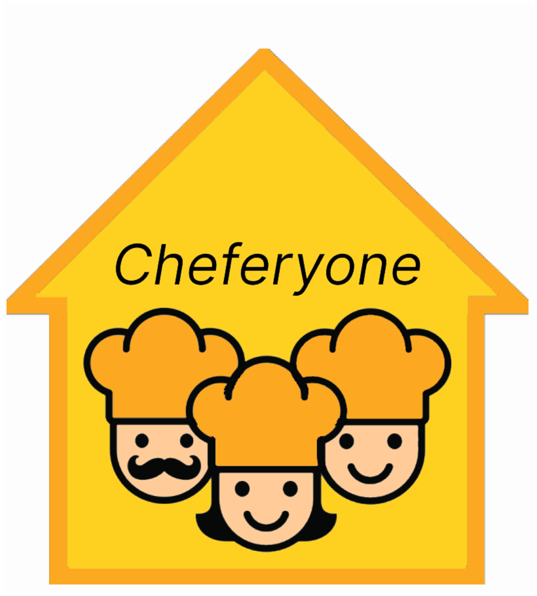
The ORM query will be translated into the following query (Assuming that this query is done on the 24th September 2020 22:39)

```
SELECT "preorders".*
FROM "preorders"
WHERE "preorders"."menu_id" = $1
AND (preorders.end_date >= '2020-09-24 22:39:22.374835' )
ORDER BY "preorders"."start_date" ASC LIMIT $2
[["menu_id", 1], ["LIMIT", 11]]
```

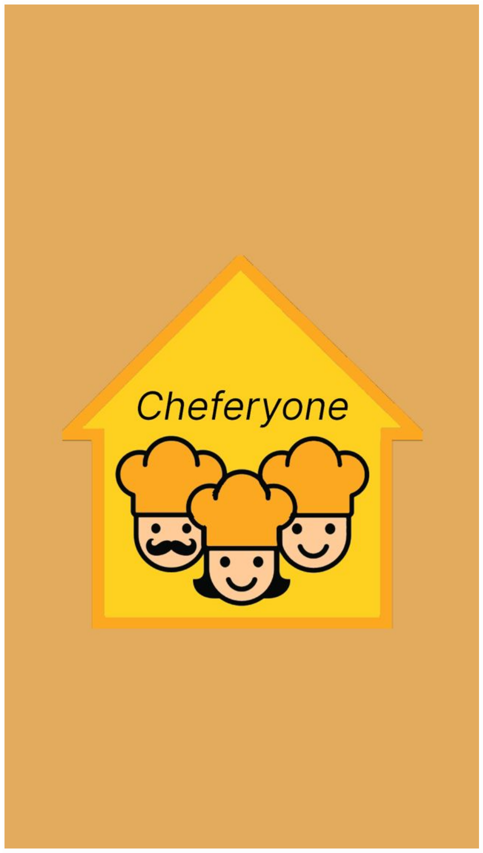
Milestone 7

Create an attractive icon and splash screen for your application. Try adding your application to the home screen to make sure that they are working properly. Include an image of the icon and a screenshot of the splash screen in your writeup. If you did not implement a splash screen, justify your decision with a short paragraph. Add your application to the home screen to make sure that they are working properly. Make sure at least Safari on iOS and Chrome on Android are supported.

Splash icon:



Splash Screen:



The philosophy behind this icon is to emphasize that everyone can be a chef, represented by the three people from different categories: teenagers (boy and girl), as well as middle-aged people (the one with a mustache). This is also reflected in the application name: Cheferyone, which is a portmanteau of ‘Chef’ and ‘Everyone’.

The fact that the three of them are inside a house symbolises that they do not need to go beyond their houses to be a chef.

Milestone 8

Style different UI components within the application using CSS in a structured way (i.e. marks will be deducted if you submit messy code). Explain why your UI design is the best possible UI for your application. Choose one of the CSS methodologies (or others if you know of them) and implement it in your application. Justify your choice of methodology.

UI Consideration

Following are some parts of our UI Design that we would like to highlight.

Easy-to-create Menu

In Cheferyone, we believed that everybody can become a chef. As a result, we are focusing our design in making it seamless to start your own restaurant or your own home cooking business.

Here, we are adopting the concept from popular social media platforms such as Instagram and TikTok. Inside TikTok and Instagram, it is very easy to become a content creator through a single click.

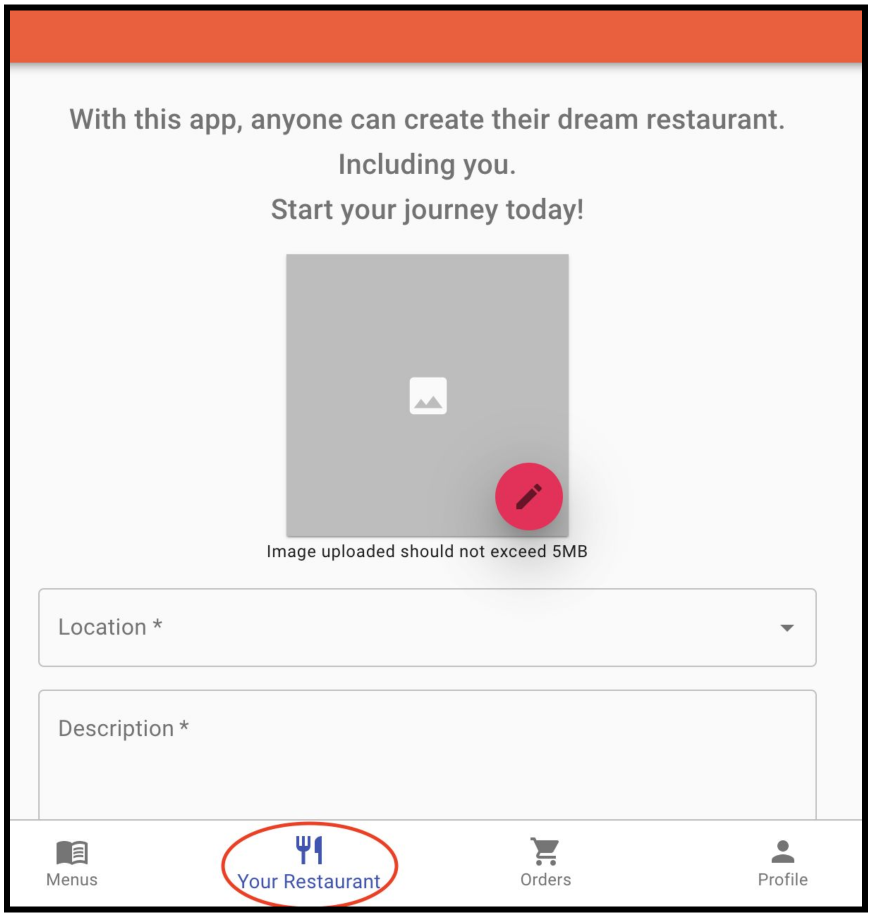

With this app, anyone can create their dream restaurant.
Including you.
Start your journey today!

Image uploaded should not exceed 5MB

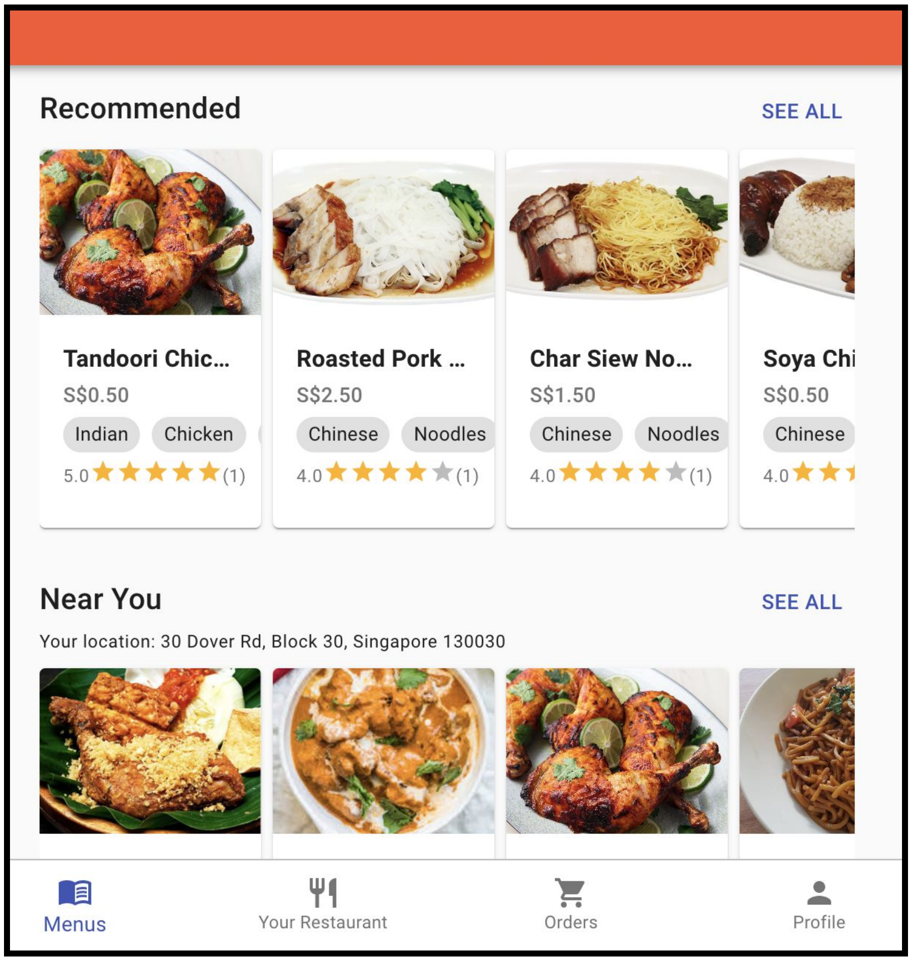
Location *

Description *

Single Click to start becoming a chef

Menu-based design

The application is designed with the individual food items put first. The main screen opens up to a plethora of food options, and users can immediately choose the menu that they prefer. Users no longer have to browse through entire stores to find what they like, and the aggregation of food options in the main page allows the users to make better decisions.



Menu-based design

CSS in JS

CSS in JS was used for the development of the UI. While the methodologies listed above try to enforce a standard to modularize css, CSS in JS takes it to another level and enforces that CSS be inside the React components. CSS in JS organizes CSS by the components that use them, and CSS reuse is minimized. Whenever there are styles that need to be reused, the styles will be abstracted into a separate component and the component will serve as a wrapper to all other components that rely on this style.

Next, CSS in JS is the default and preferred styling option for the framework we are using, Material UI. CSS in JS support is built into Material UI, and most documentation and help topics are written with CSS in JS as the styling option. Thus, using CSS in JS will speed up development time as less time is wasted converting documentation to plain CSS that Material UI does not support very well. Additionally, CSS in JS allows the provisioning of a global theme object that all components have access to, and this allows the standardization of colors and margins to allow the application to have a more uniformed look and feel.

All in all, our team felt that this experience using CSS in JS was a good one, With some members being familiar with BEM, the strengths of CSS in JS shines through, as modularization and scoping of styles is strictly enforced.

Milestone 9

Set up HTTPS for your application, and also redirect users to the https:// version if the user tries to access your site via http://. HTTPS doesn't automatically make your end-to-end communication secure. List 3 best practices for adopting HTTPS for your application.

As part of securing our apps, all communications are encrypted through HTTPS communications.

Following are the 3 best practices in adopting HTTPS within Cheferyone.

1. Deployment on Heroku

Rather than issuing our own SSL Certificates, we have decided to choose our deployment on Heroku. Heroku has a support for HTTPS if we decided not to customized our domain name[2].

2. HSTS Support

As part of the best practices suggested by google[3], we are redirecting all HTTP requests into using HTTPS Protocol. On the backend side, we are implementing this with force_ssl features from rails[4].

3. Third-party integration through HTTPS

Cheferyone has several integrations with third-party apps, such as algolia, google maps geolocation, and firebase. All communications with these integrated services are governed through HTTPS Protocol. Therefore, not only the internal communications, between backend and frontend, is secured and encrypted but also our external communications channel.

[2] Heroku supports HTTPS by default: https://devcenter.heroku.com/changelog-items/1815
[3] Best practices on HTTP: https://support.google.com/webmasters/answer/6073543?hl=en
[4] Force SSL in Rails: https://api.rubyonrails.org/classes/ActionDispatch/SSL.html

Milestone 10

Implement and briefly describe the offline functionality of your application. Explain why the offline functionality of your application fits users' expectations. Implement and explain how you will keep your client synchronised with the server if your application is being used offline. Elaborate on the cases you have taken into consideration and how they will be handled.

Cheferyone allows the users to browse listings they've viewed before offline. Additionally, Cheferyone caches some form submissions offline and will send the requests to the server once the users are online. This offline feature is useful because it provides convenience to the users so that they can decide what to eat anywhere and anytime, because not everyone always stays connected to an internet connection.

Whenever the user submits a form offline, it will be cached in localStorage. Once the "online" event is fired, these network requests will be sent, and a message will show after all requests have been sent. If no data was sent by the user when offline, the app will be repopulated with new data after a reload.

Generally, all static resources like HTML, CSS, and images will be cached via service workers. Since ours is a dynamic application with lots of constant updates from buyers and sellers, cache will only be employed when the network is not available or a GET request fails for some reason. 2 special cases are the Google Analytics and Google Maps Javascript API bundles. These resources have opaque responses, and will seem to take up a lot more space in the cache than they actually do. In our experiments, the bundles seemed to take up a combined 100mb in storage, which is a lot. Additionally, since these 2 bundles can't work offline, we excluded them from caching.

As shared previously, some data submissions will also be queued offline in localStorage. The forms excluded from offline submissions are the ordering form. It does not make sense for someone to pay for an order offline. Next, posting of data involving images is also excluded from offline caching as the localStorage mechanism we use to handle such caching does not have enough space for images. Finally, posting of data including location information will not work offline too as that involves Google Maps.

Milestone 11

Compare the advantages and disadvantages of token-based authentication against session-based authentication. Justify why your choice of authentication scheme is the best for your application.

We have done some research on token-based authentication and session-based authentication. Token-based authentication works by a creation of JSON Web-Token (JWT) with its secret by a server. The JWT is sent to the client and stored. The JWT is included as a header by the client everytime it makes a request, and the server validates the JWT with the secret. For session-based authentication, the server creates a session for the user after he/she logged in. The session id is stored as a cookie, and it will be sent together with the requests. The session verifies the session id inside the cookie and the id in the memory.

Scalability
The advantage of token-based over session-based is regarding the scalability aspect. There is no issue when the number of users scale up for token-based authentication because the token is stored by the client in the local storage, whereas the server is responsible for saving the session id in the session-based authentication, and thus scalability can be a potential issue for session-based authentication when there are numerous users using the app at the same time. Another advantage is that token-based authentication performs signing so that the data can be verified easily that it is indeed coming from the client and has not been modified by unauthorized parties.

Security
The advantage of session-based over token-based is about the securing process. Securing a session id is easier compared to securing an encryption token. However, several frameworks, including Rails, provide signing and encryption for the session, so this aspect is not a serious issue. The next advantage is that the length of token for session-based is shorter than the one used in JWT. And last thing that token-based authentication actually lacks compared to session-based authentication is that we might still need to access the database, even though the JWT is invalid.

Based on the advantages and disadvantages mentioned above, we decided to go ahead with token-based authentication. The advantage of session-based authentication is not really helpful since we use Rails as our backend framework. The scalability is more an issue for our app, and using token-based authentication surely helps to address the memory issue.

Milestone 12

Justify your choice of framework/library by comparing it against others. Explain why the one you have chosen best fulfils your needs. Lastly, list down some (at least 5) of the mobile site design principles and which pages/screens demonstrate them.

Material UI

Our group chose Material UI. Among UI frameworks, Material UI has one of the most stars on Github and thus is one of the most popular UI frameworks. Being a popular UI framework would mean that its documentation is more complete, and that there are more stackoverflow answers to any questions or problems that may arise from its usage. Next, Material UI is also one of the most updated UI frameworks, with 12, 657 commits on Github currently. Most other UI frameworks get only close to 10, 000 and below commits, and this constant update allows Material UI to stay relevant. Thus, the popularity and up to date nature of Material UI makes it a great framework.

Material UI also has a global theme provider, which most other frameworks do not have. This allows an app to have a more consistent look and feel across pages.

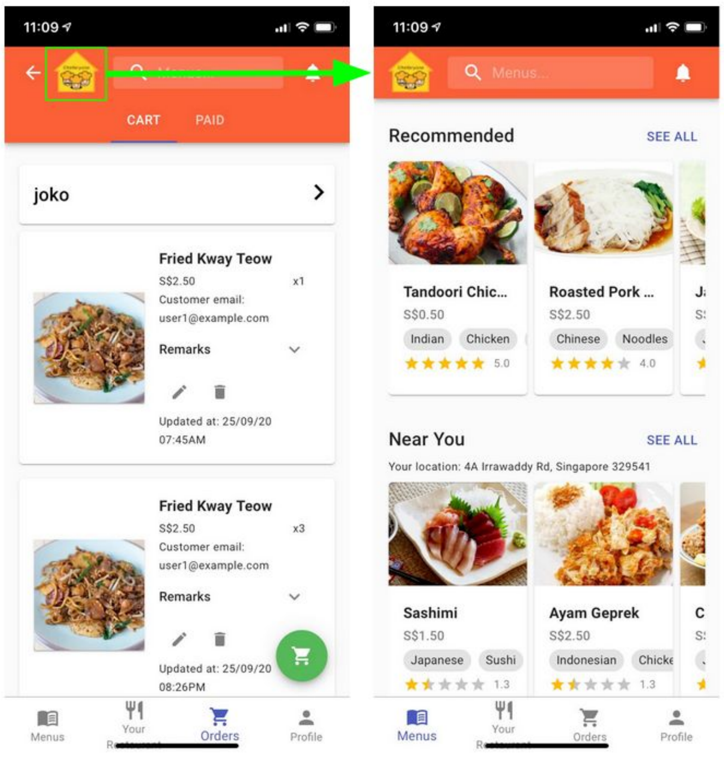
Next, Material UI supports ReactJS, and that is the framework that most of the group members are accustomed to. The familiarity of our group with both ReactJS and Material UI thus makes it a very obvious choice for the group, and this directly helps us fulfil the needs of the project as more time is spent coding and less time is spent reading documentation and learning.

Finally, Material UI offers a familiar and attractive UI for our users. Google uses Material UI design guidelines extensively, and the familiarity of users with Google products will also foster a familiarity with our application, and this is another advantage of Material UI that the other frameworks do not provide.

Mobile Site Design Principles

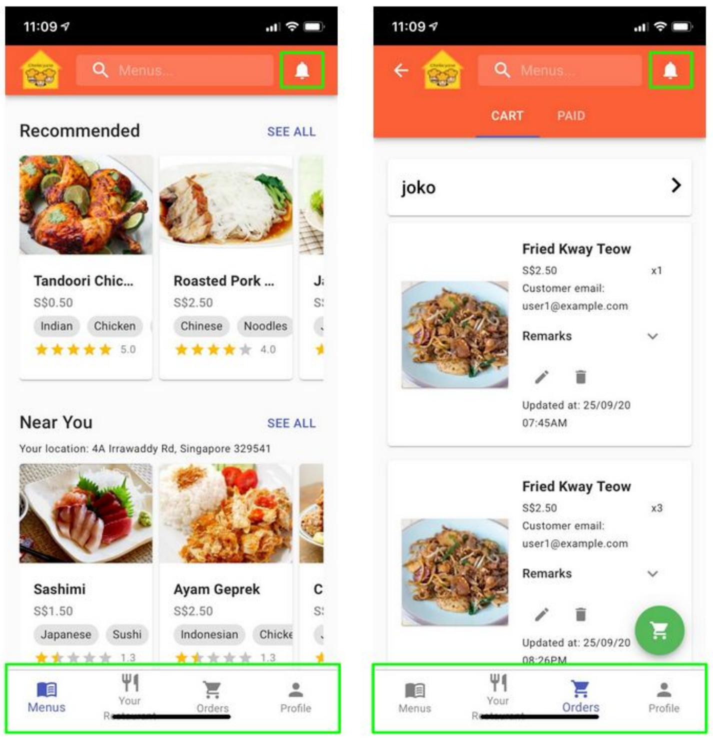
Making it easy to get back to the homepage

The ‘Cheferyone’ logo on the upper left corner is clickable so that the user can return back to the homepage easily. Moreover, the ‘menu’ button on the navbar also allows for an alternative navigation back to the first page which is the ‘menu’ page.



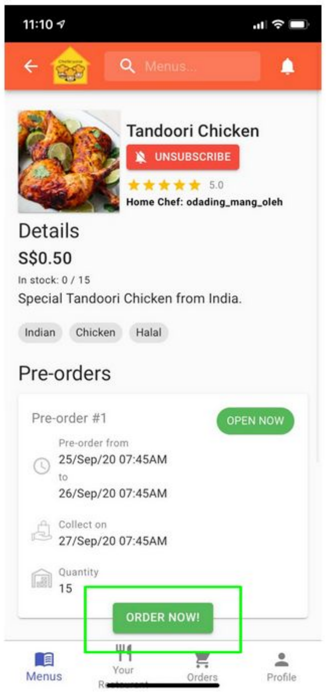
Keeping menus short and sweet

When opening the home (menu) page, the menu is in the form of a bottom navbar with four simple options: ‘Menu’, ‘Your Restaurant’, ‘Orders’, ‘Profile’. The ‘Notification’ button (the bell) is available on the upper-right corner as well in every page.



Keep calls to action front and center

Upon opening the menu page, the user is greeted by cards of the menus which they can make an order from. If the user clicks on it and it is an open pre-order period, the user will be provided with a bright green ‘Order now!’ call-to-action button that is always available regardless of the user’s position in the screen


11:10
Menus...
Tandoori Chicken

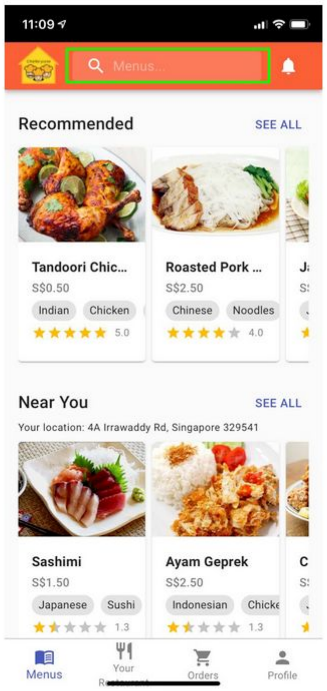
Home Chef: odading_mang_oleh
Details
S$0.50
In stock: 0 / 15
Special Tandoori Chicken from India.

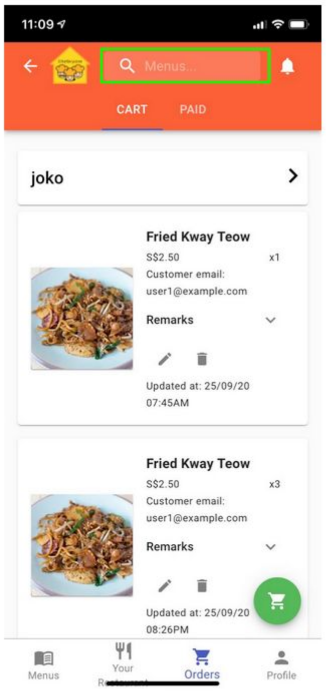
Pre-orders
Pre-order #1

Pre-order from
25/Sep/20 07:45AM
to
26/Sep/20 07:45AM
Collect on
27/Sep/20 07:45AM
Quantity
15

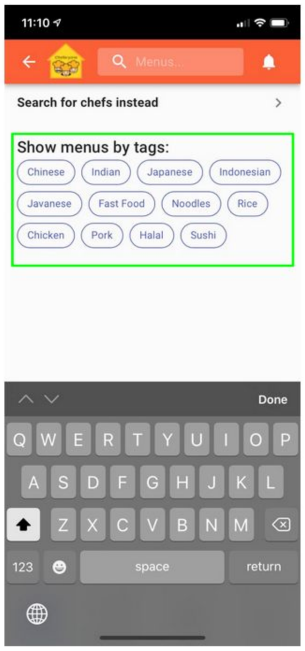
Make site search visible

In Cheferyone, the site search is always visible in the top bar regardless of which page you are in, making navigation accessible from any part of the application

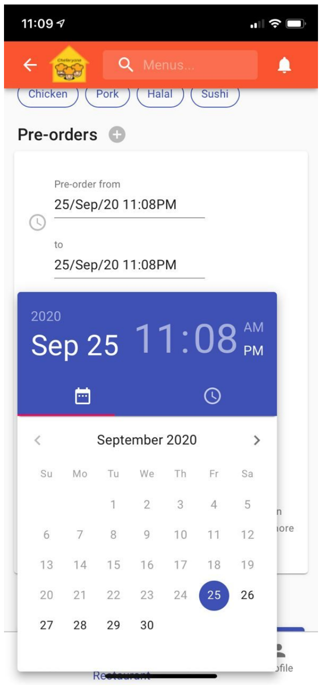




Implement filters to narrow results
We have implemented a set of pre-defined tags which users can use to get a filtered result. It is very easy to filter as the user can simply just select which tag that they want to be included.



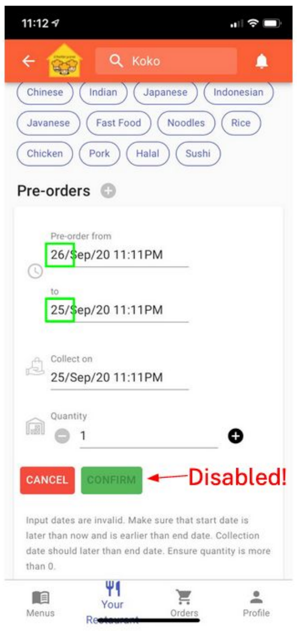
Provide visual calendar for date selection
We use a calendar widget when a restaurant is updating its pre-order dates



Pre-order from
25/Sep/20 11:08PM

to
25/Sep/20 11:08PM

Minimize form errors with labelling and real-time validation
Forms are clearly labelled and if it violates some constraint (start date is after end date), the update button is disabled



Pre-order from
26/Sep/20 11:11PM

to
25/Sep/20 11:11PM

Collect on
25/Sep/20 11:11PM

Quantity
1

Input dates are invalid. Make sure that start date is later than now and is earlier than end date. Collection date should later than end date. Ensure quantity is more than 0.

References:

- https://developers.google.com/web/fundamentals/design-and-ux/principles

Milestone 13

Describe 3 common workflows within your application. Explain why those workflows were chosen over alternatives with regards to improving the user's overall experience with your application.

Users ordering and paying for menus
The first common workflow that Cheferyone provides is about the process of users ordering foods. Starting from the homepage, users are provided two options to find their desired menu. The first option is to scroll through the menus available in homepage, that are divided into three categories: Recommended, Near You, and New. The second option is to search the menus using desired keywords in the search bar. Cheferyone also provides a way to search by menu's name, chef's name, or using tags.

Cheferyone is designed to have separate searches for menus and chefs to prevent mixed and cluttered search results that users may find it difficult to get their desired results, and thus enhancing the user experience. We provide these two ways to cater both people who don't have any cravings for particular foods in his/her mind, and people who already know what he is craving at the moment (thus accessing it via the chef).

In the menu page, users can create an order by simply clicking "Order Now!", decide the quantity, add the remarks, then add it to the cart. Then, the users will be redirected to the cart page. This workflow helps us in enhancing the user experience since the users don't need to go to the cart page every time.

The users can search for more menus, if they want to, and go to the checkout page once they are finished placing orders. The users can provide their credit card information and pay multiple orders from multiple chefs at once (currently it is a mockup and not actually implemented). This is heavily useful compared to making a payment per order, when a user has many orders. After that, the users simply need to wait for confirmations from the chefs

An alternative we considered is that the menu is paid by the restaurant. For example, if I order 2 menus from different restaurants, I pay them separately. However, this was not done in the end because we feel that it is more convenient to pay for everything at once.

Home chefs start a restaurant
The next workflow is from the home chefs' perspectives, where a user wants to start his/her first step as a home chef, creating his/her own restaurant in Cheferyone, then start opening pre-orders.

The first step to be a home chef in Cheferyone is to make our own restaurants in the 'Your Restaurant' tab. The reason for allowing the users to create the restaurants rather than giving every user an initial restaurant is due to some group who only want to be buyers.

Therefore, Cheferyone provides a win-win solution to keep the app features minimalistic to enhance the intuitiveness of the app.

The chefs can update the information of their restaurants in the ‘Your Restaurant/Edit’ Tab. The chefs can simply create menus using the magenta button with plus sign at the bottom right corner. They can fill in the image, name, description, price of the menu, and set pre-orders schedule. If the chefs want to set more pre-order dates, they can simply use the ‘Edit’ button to schedule more pre-orders, together with updating the information of the menu.

The chefs can view other orders’ on their menus by going to the ‘Your Restaurant/Orders’ tab. The orders are categorized based on their status. The reason why Cheferyone segregates the orders based on the status and not merge it into one section and sort them by the order date is because the status of the orders provide more senses of urgency compared to the orders’ date. From this page, the chefs can update the status of the orders e.g. approving or rejecting someone’s orders.

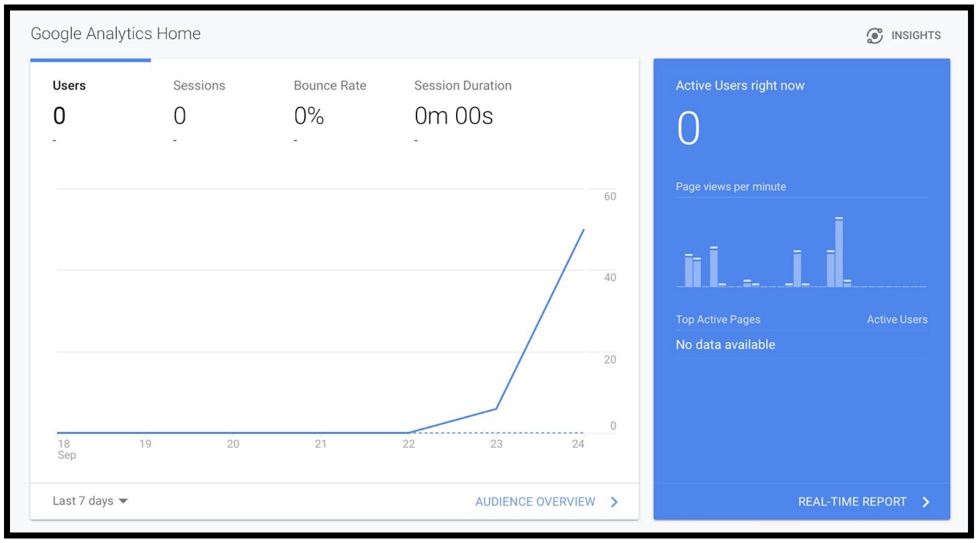
Subscriptions and Notifications
The last workflow that is worth to mention is the subscriptions and notifications. Cheferyone allows the users to subscribe to menus and chefs by going to the profile of the menus or the chefs. There is a notification button provided.

The subscription feature is useful for Cheferyone because the subscribed users will receive notifications whenever new pre-orders of the subscribed menu are opened, or the subscribed chef make a new menu. This notification is available by clicking the ‘bell’ icon on the upper right-hand corner of the screen. Therefore, Cheferyone users should not worry about missing the dates!

Moreover, notifications feature also takes a role in informing the users regarding the status of their orders every time the responsible chefs update them. The chefs also get notifications whenever they get new orders so that they are not missing out any orders to be confirmed or rejected. How cool is that? :)

Milestone 14

Embed Google Analytics in your application and give us a screenshot of the report. Make sure you embed the tracker at least 48 hours before submission deadline as updates are reported once per day.



Google Analytics Home

Users	Sessions	Bounce Rate	Session Duration
0	0	0%	0m 00s
-	-	-	-

Last 7 days

AUDIENCE OVERVIEW

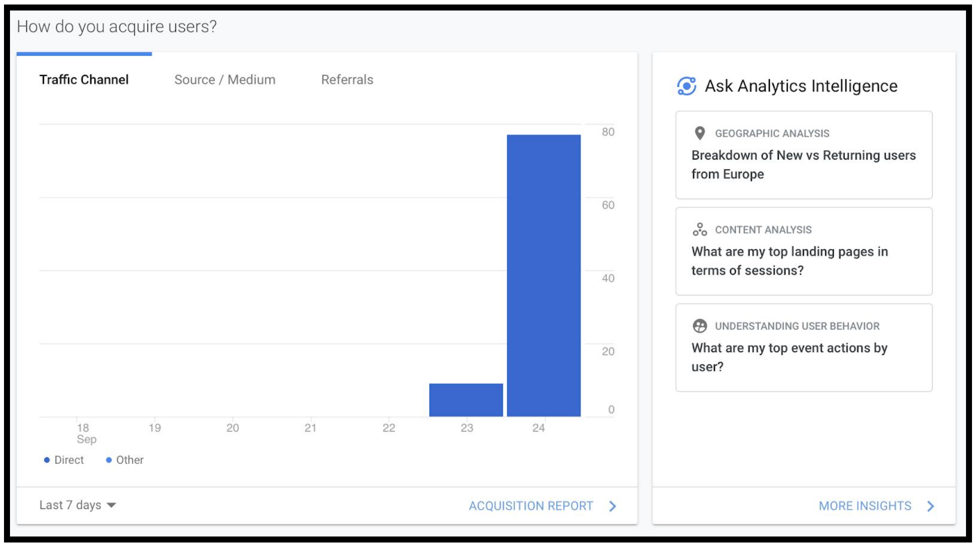
Active Users right now

0

Page views per minute

Top Active Pages	Active Users
No data available	

REAL-TIME REPORT



How do you acquire users?

Traffic Channel   Source / Medium   Referrals

Last 7 days

ACQUISITION REPORT

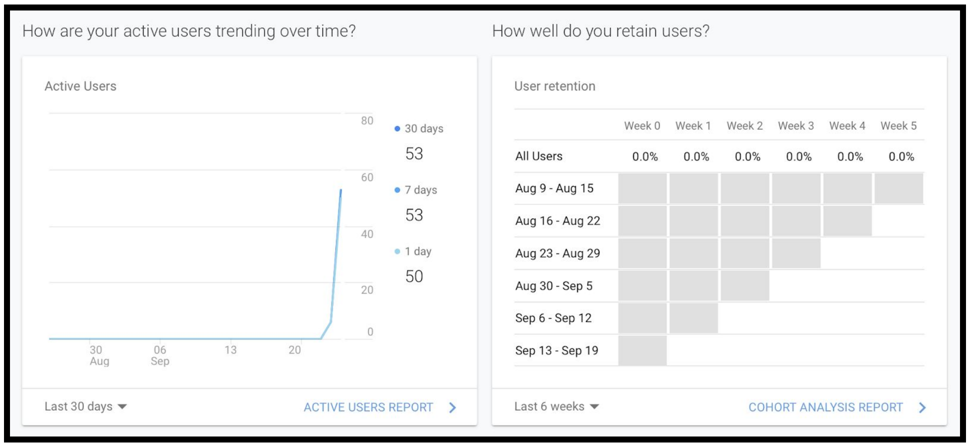
Ask Analytics Intelligence

GEOGRAPHIC ANALYSIS
Breakdown of New vs Returning users from Europe

CONTENT ANALYSIS
What are my top landing pages in terms of sessions?

UNDERSTANDING USER BEHAVIOR
What are my top event actions by user?

MORE INSIGHTS



How are your active users trending over time?

Active Users

● 30 days
53

● 7 days
53

● 1 day
50

Last 30 days

ACTIVE USERS REPORT

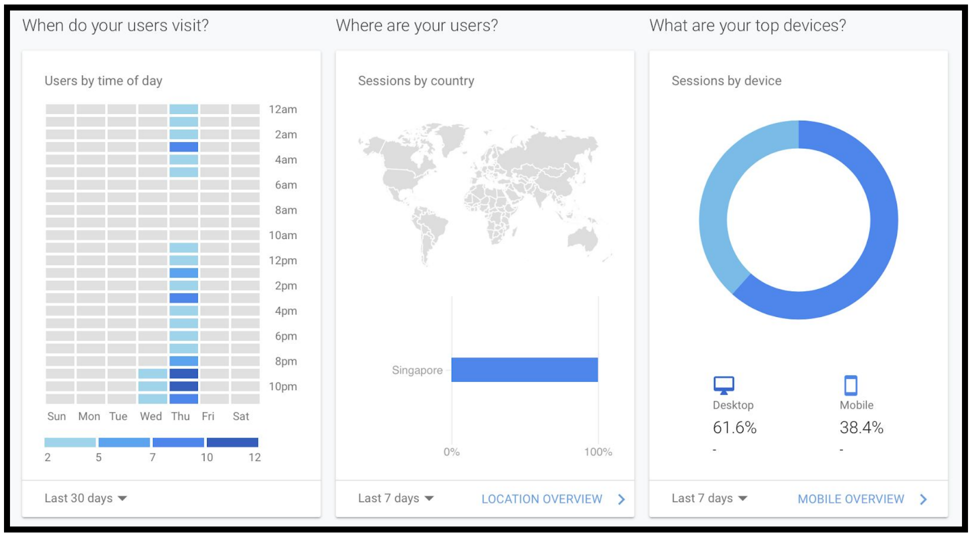
How well do you retain users?

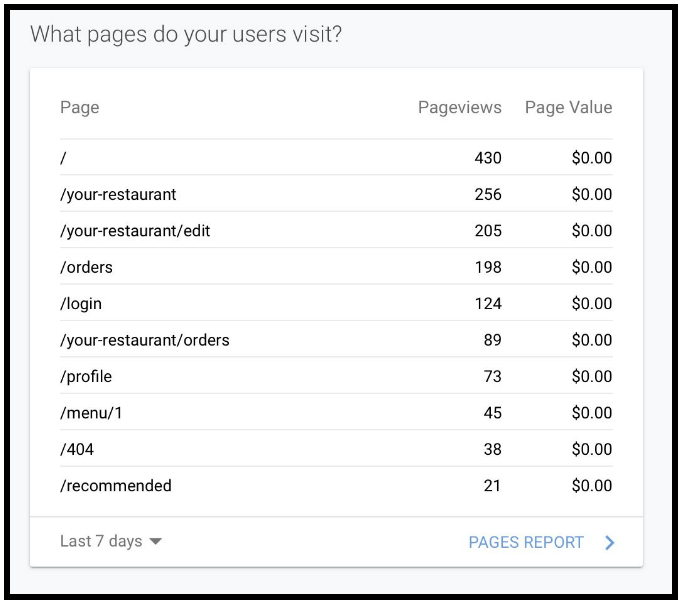
User retention

	Week 0	Week 1	Week 2	Week 3	Week 4	Week 5
All Users	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Aug 9 - Aug 15	▨	▨	▨	▨	▨	▨
Aug 16 - Aug 22	▨	▨	▨	▨	▨	
Aug 23 - Aug 29	▨	▨	▨	▨		
Aug 30 - Sep 5	▨	▨	▨			
Sep 6 - Sep 12	▨	▨				
Sep 13 - Sep 19	▨					

Last 6 weeks

COHORT ANALYSIS REPORT




What pages do your users visit?

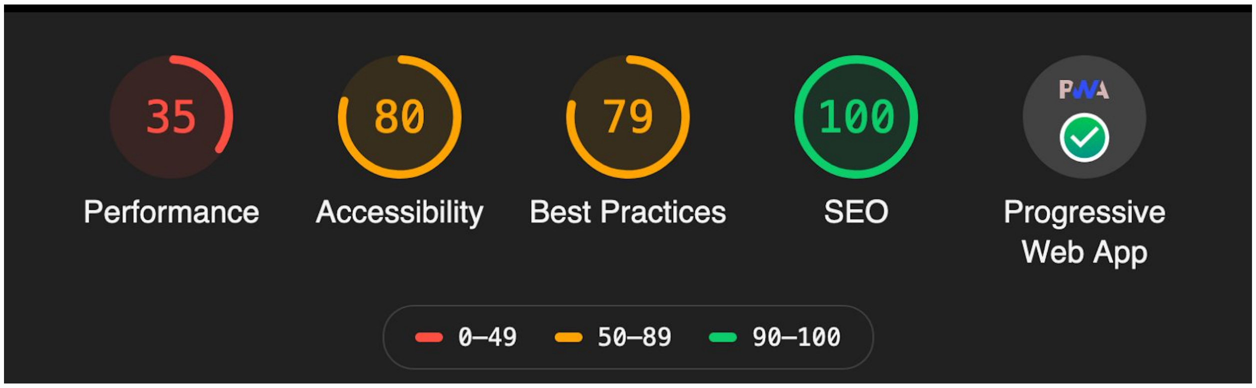
Page	Pageviews	Page Value
/	430	$0.00
/your-restaurant	256	$0.00
/your-restaurant/edit	205	$0.00
/orders	198	$0.00
/login	124	$0.00
/your-restaurant/orders	89	$0.00
/profile	73	$0.00
/menu/1	45	$0.00
/404	38	$0.00
/recommended	21	$0.00

Last 7 days

PAGES REPORT

Milestone 15

Achieve a score of at least 12/14 for the Progressive Web App category and include the Lighthouse html report in your repository.



Milestone 16

Identify and integrate with social network(s) containing users in your target audience. State the social plugins you have used. Explain your choice of social network(s) and plugins. (Optional)

Cheferyone does not provide an implementation to do social media authentication. The existing social media integrated for authentication are mostly Facebook and Google accounts. Meanwhile, our actual target users are mostly from other social media such as Instagram. Therefore, we feel that it is not very relevant to integrate social media authentication to our app and we decide to go ahead spending our time on other features. Don't worry, Cheferyone is still cool, even without social media authentication! ;)

Milestone 17

Make use of the Geolocation API in your application. (Optional)
We use the geolocation API in the Near You menu display in the homepage based on their ascending distance to your current location. We use Geokit Rails API to perform the distance calculation and this API will gather the information regarding the latitude and the longitude of the user in addition to the information of every menu location posted by home chefs. Additionally, we have a location search bar that interacts with the Google Places API to come up with suggested locations.

Deployment

Our app is published on heroku and can be accessed at here
https://cheferyone.herokuapp.com/