Eat App – Analysis

Background

The eat app will be a cross-platform mobile application written in Dart, using the Flutter SDK by Google.

The eat app is an app for the foodie, the student, the penny pincher and the everyday person. The eat app lets you save money on food and it’s as simple as that.

Restaurants are quiet between the hours of 3-6 and sometimes even lunch. The eat app will give you, as a consumer, a chance to visit these restaurants during pre-selected hours and save 30-50% on your food bill. Example: you’re a student in Exeter, you want to eat out, however you’re conscious about money. You log into the app and find all the restaurants offering deals near you. To access the deal you pay a time fee of £1 per person. That £1 voucher will give you access to 30-50% of your food bill at whichever restaurant you choose to eat out at.

I’ve decided to create this app simply because of two things:

1. I thoroughly enjoy eating out.
2. I’m broke.

I think this situation not only applies to myself but many, many people across the UK and therefore I think this app would be extremely useful to a large demographic.

There are a few components and problems however, that need to be addressed. First of all, it’s very hard to know where you want to eat out, if you’ve never really been to any of the restaurants. There are hundreds of styles of restaurants that all have their own take on classic meals but also have their own new and inventive meals. Therefore, the app will need to help the user understand what sort of food each restaurant serves and maybe help them decide where they might like to eat out. This could be done by assigning categories to every restaurant, e.g. italian, and then allowing the user to search through these categories to find a restaurant that serves a particular food type that they are a fan of. It could also be achieved by having a profile page for every restaurant. A profile page would include pictures of the restaurant and their meals, a copy of their menu, ratings and reviews, social media activity and general updates from the restaurant. A user should also be able to favourite restaurants and that data could be used to start to suggest new restaurants to a user based on their previous eating habits.

Furthermore, finding restaurants near you when you are away from home can also be very difficult. Especially when you can only search for ‘restaurants near me’ and you end up having to sift through a hundred different chicken restaurants only to find that one burger bar that looks like it would suit you to a tee. The app will give users a personalised feel no matter where they are by allowing them to search restaurants near them with many combinations of filters and keywords and then also order those search results.

Another problem is - how does it benefit restaurants? Well, the most obvious benefit is that they get customers with cash filled pockets, ready to empty them, through their doors. Also, restaurants will be able to understand what sort of people they get through their doors at certain times. E.g. from data collected they’ll be able to see that students often like to go out on Wednesday night so they may decide to offer extra deals on drinks or snacks every Wednesday, to people who book through the app.

In general, this is a win-win situation for the restaurants and customers.

Client

The app’s intended user will be people who eat out a lot, particularly in the demographic of age 30 and under also people in relationships. This is because people of this age eat out the most (Forbes reported millenials spend 44% of their money on food) and many are looking to eat out for as cheap as possible. Also, couples typically eat out together often so are always looking for the cheapest place possible - but they also want quality. This is a key selling point of the app - you aren’t losing any quality by going to cheaper restaurants, you are just getting a great deal.

The typical millennial is actually very competent when it comes to technology and mobile apps, therefore the user interface can be slightly more creative which will allow me to display more complex content to improve the user’s experience. If the target client was slightly older then the user interface would have to be simplified so that they are able to use the app, stress free.

Interface

Why a mobile app? Everyone has a phone, especially in my target demographic. It also means that people are able to use the app to book a meal at any time they fancy – having a catch up with someone on your lunch break and fancy meeting up later, no worries as you can open the app and book it there and then. This means that the product is always accessible. On the other hand, if the app were a website it requires a lot more effort from the client to navigate to your webpage and sign in etc - even if your site is mobile friendly!

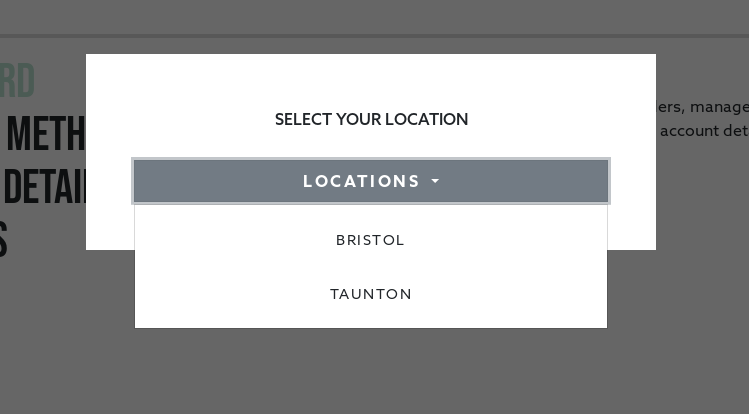
Technologies

The app will be written in a programming language called Dart. Dart is an object-oriented programming language that is fast, portable, and has a large ecosystem of libraries and contributors to its name. It’s also open source! I am also using Flutter. Flutter is Google’s mobile app SDK for crafting high-quality native interfaces on iOS and Android in record time. Flutter is also open source! I will be using Firebase – a NOSQL cloud database system along with a hoard of other capabilities such as authentication and cloud functions. The payment gateway I’ll be using is Braintree by PayPal. This will allow payment via many card types and also PayPal.

Existing Solution: EatEarly.co.uk

There is an existing product of a very similar style called EatEarly. It is a website that enables you to scroll through a list of restaurants based on the location that you select, and then buy a voucher for a certain restaurant. There are, however, many improvements that I will make in my app.

Examples of current system:

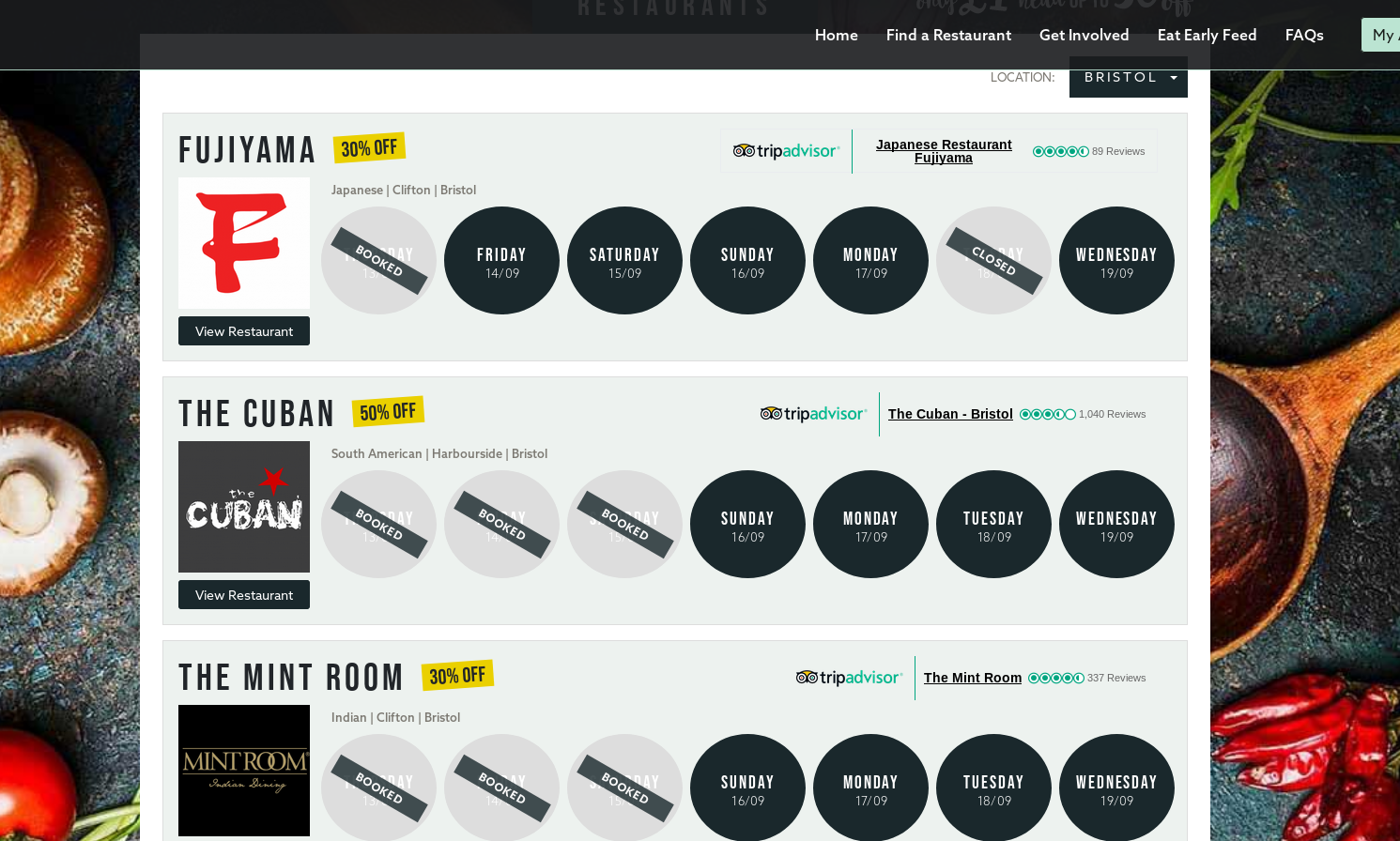


Location dropdown:

This dropdown list is the only way you are able to filter through restaurants

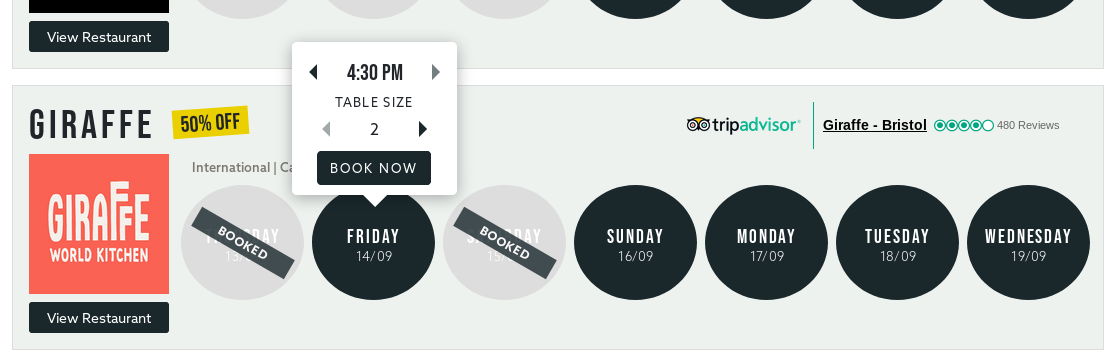
List of restaurants:

Once you’ve selected your location you are presented with a list of all the restaurants in that location. You have to scroll through this entire list looking at each one to find the restaurant that you want to eat at. If you want to find out more about the restaurant you have to click on their logo which will take you to their profile page. This can take a long time to load all of the information about the restaurant. If you are looking at many restaurants this means it can take a very long time to read all the info about each one and decide on your favourite.



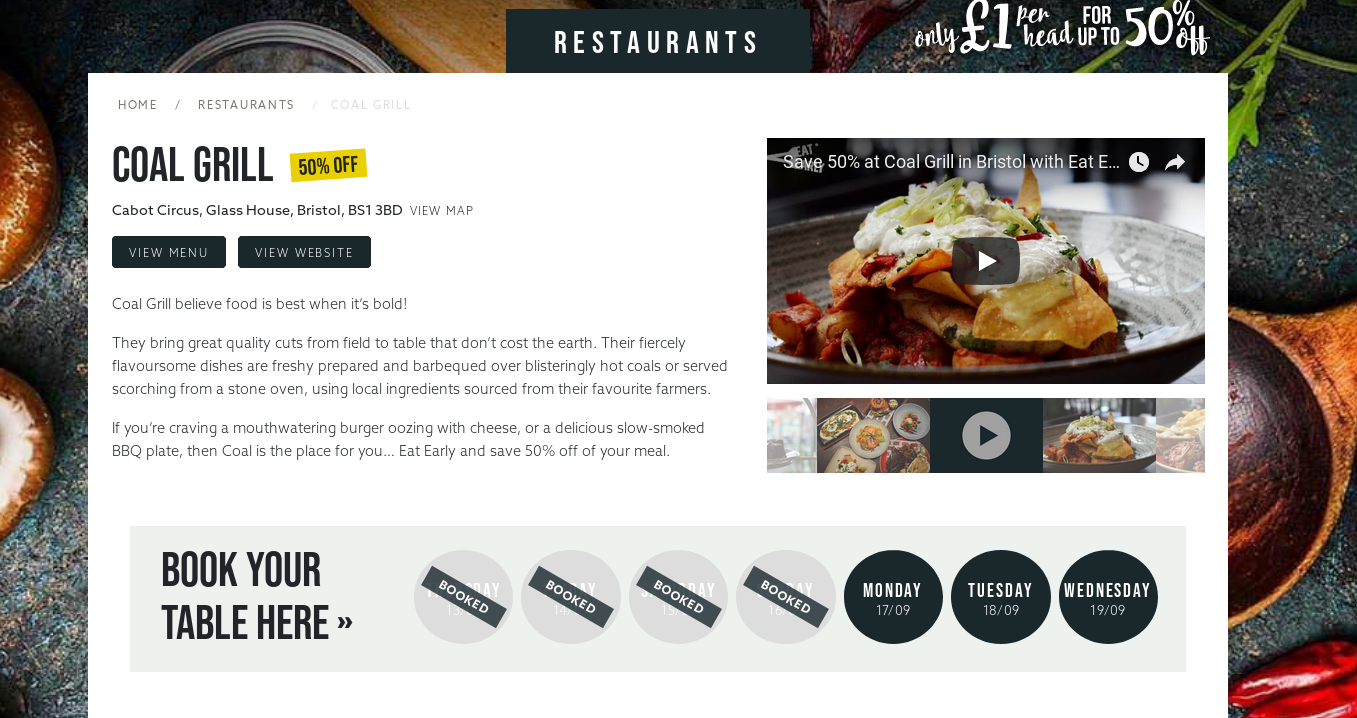
Booking interface:

The booking interface is also quite difficult to get your head round if you are wanting to book a meal quickly, however some of its design is actually quite intuitive. You have to click the arrow repeatedly to get to a time that is later than you are currently on and the same goes for the table size.



Restaurant view:

The restaurant view is very well laid out and provides a lot of easily-accessible information, quickly. You can book a table with a few clicks, view their menu, visit their website, watch their promotional videos and also read their description. I will use this page as inspiration for the profile page within the app as it has a lot of content, but is very easily digested.



Improvements on EatEarly

* Being mobile; having a mobile app means that people are able to easily book through their phone whenever they like.
* User experience; the existing solution doesn’t have a very friendly user experience. The UI is in fact very beautiful, however, there aren’t many features that really appeal to a user specifically. For example, there is no suggestion for restaurant types, no way to view your voucher as a code/barcode and not a central view of restaurants based on your location or any preferences. These will all be at the forefront of the app design.
* Maps; with EatEarly you are not able to see restaurants near your current location. You are only able to scroll through a list of all the restaurants in a certain location, that you select from a drop down list.
* Filtering; there are no filtering options with EatEarly. You select your location and then have to scroll through all the restaurants, clicking on each one to find out more, to book your selected restaurant. The app will allow you to filter on things such as food type (e.g. Italian), restaurant size, distance from your location, discount size, rating and opening times etc. so that you are able to find your perfect restaurant at the perfect time, for cheap!
* Speed; it can take sometimes over 10 seconds for EatEarly to load a list of restaurants, the mobile app will be much quicker.

Data needed

There is a certain amount of data I will need from the restaurants and data that will also be stored from customers and generated data from the app.

Data from the restaurants: opening days of the week, times that they are willing to offer a discount for, the amount of discount they are offering, number of places available for voucher customers, the menu, logo, description, website and social media links and contact details.

All of this information will be used throughout the app when allowing users to book a table, view the restaurants profile and also via the backend when contacting a restaurant about a reservation.

Objectives

* 1 - Login page and authentication.
  + 1.1 - Login page content.
    - 1.1.1 - Splash screen with app logo and background.
    - 1.1.2 - A form for existing users to log in with their email and password.
      * 1.1.2.1 - If the email entered is not recognised a dialog box should appear notifying the user.
      * 1.1.2.2 - If the password entered is incorrect a dialog box should appear notifying the user of their mistake.
      * 1.1.2.3 - A button that when pressed makes a loading dialog appear while their account is authenticated. On success they will be taken to the home page.
      * 1.1.2.4 - On a failurer other than listed above, the user should be notified via a dialog box.
      * 1.1.2.5 - A forgot password button will be underneath the form that if pressed will allow the user to enter their email. If this email is linked to an account they will be sent a password reset email.
    - 1.1.3 - A form for new users to sign up and create an account with their email and password.
      * 1.1.3.1 - Required information will be: email, full name, password, repeated password, terms and conditions accepted.
      * 1.1.3.2 - If the email entered is in an incorrect format, a dialog box will appear notifying the user.
      * 1.1.3.3 - The password should be of a minimum length of 8 characters.
      * 1.1.3.4 - The passwords must match for the user to successfully sign up.
      * 1.1.3.5 - There will be a sign up button which when pressed will present a loading dialog. In the background a verification email will be sent to the user. If they do not verify their account, certain features (like booking a reservation) will be unavailable to them until they do so. Once successfully signed up they will be sent to the home page.
      * 1.1.3.6 - If any of the required info is not filled in when the user tries to sign up, then a dialog box will be displayed informing the user that they must complete the whole form.
    - 1.1.4 - These will be provisioned using Firebase to create, store and authenticate users.
  + 1.2 - Login page backend.
    - 1.2.1 - The backend will be handled via Firebase. This will involve including the plugin in my login page code and calling the plugin functions which will authenticate and create users, updating the Firebase database at the same time.
    - 1.2.2 - When a user is created/logs in I will store some data in the database such as last login time.
    - 1.2.3 - I will create my own user class that will store data about the currently logged in user (e.g. name and voucher info etc).
    - 1.2.4 - The information about the user will need to be accessed all across the app. This is called state management (how the data inside my app is passed around to different pages). There are a few options for state management. I will experiment with ScopedModel, Redux and BLoC.
* 2 - Home page.
  + 2.1 - Home page content.
    - 2.1.1 - A splash screen will be displayed containing offers of the day and any special promotions that are currently happening.
    - 2.1.2 - You will be able to select any of the current offers and you will be taken straight to the information page for that voucher so you can book it.
    - 2.1.3 - If there are no available offers or promotions that are special or limited time, suggested restaurants will be displayed.
  + 2.2 - Home page backend
    - 2.2.1 - The home page will need to access state about the currently available offers. I will need to create a custom class for offers which will store details about the offer that I will display such as the name, location and price.
    - 2.2.2 - A lot of widgets in the home page and other pages will be reused across many different pages. As a result of this I will create custom Widget classes by inheriting existing Widgets from Flutter and customizing them.
* 3 - Find a restaurant page.
  + 3.1 - Find a restaurant page content.
    - 3.1.1 - The page will first load with restaurants based on location or, if not available, will load a placeholder asking them to search for a location or restaurant name.
      * 3.1.1.1 - If the user has not yet allowed/disallowed the app to use their location then this will be requested first.
      * 3.1.1.2 - If the user hasn’t allowed the app to use their location then the map features and sorting by distance won’t be available. This is unlikely.
    - 3.1.2 - A list of restaurants will be displayed based on the current filter and search criteria.
      * 3.1.2.1 - If a restaurant is clicked on then you will be taken to their profile page where you can find out more about the restaurant and book a table.
      * 3.1.2.2 - Each list item will also have a book now table. When clicked a the booking popup will display where you can select the date, time and number of people that you wish to book for.
    - 3.1.3 - At the top of the page will be an expandable selection which the user can press on. When pressed it will toggle into/out of its expanded state and will show options to filter the restaurant.
      * 3.1.3.1 - Options will consist of food types, days of the week to eat, time ranges to eat and for the number of people you wish to book for. The list of restaurants will then change to only show restaurants that meet this criteria.
    - 3.1.4 - In the top right of the page there will be an icon that you can press to only display restaurants that you have favourited. You can favourite restaurants via the restaurant’s profile page.
    - 3.1.5 - In the top left will be a button that you can press that will display the map view version of the restaurants.
  + 3.2 - Restaurant page backend.
    - 3.2.1 - I will create a custom restaurant class that will store all of the details about the restaurants. This will also need to be provisioned to the rest of the app via my selected state management method.
    - 3.2.2 - Data about the restaurants will be stored in the database and will often be updated so therefore my state management method will need to react to changes in the database and make sure that all clients are updated.
    - 3.2.3 - The restaurant class is likely to include composition by aggregation, as it will contain the currently booked vouchers for the restaurants etc. This is the same with the user class.
    - 3.2.4 - I will need to perform complex filtering and searching on the restaurants based on the search criteria the user makes.
      * 3.2.4.1 - When the user searches for a restaurant name or location I will need to perform a search on all of the restaurants to find similar named places, and update the view in realtime (after each character typed). [Algorithm undecided on at the moment]
      * 3.2.4.2 - When the user sorts by location I will use a merge sort method to order the distance from the user’s current location and the restaurant. I may use an API like the google maps API to find the distance via roads, rather than a direct distance based on GPS coordinates alone.
* 4 - Map view.
  + 4.1 - The map view will be a full screen map that as you pan around will display markers of restaurants within the current view of the map.
  + 4.2 - The user will be able to click on a restaurant marker and view that restaurant’s profile.
  + 4.3 - The user can also press a button that will take them to their current location, so they are able to see whereabouts, and how far away the restaurants that they are looking at are.
  + 4.4 - The backend of the map view will consist of my own custom list structure that will hold the locations of the restaurant markers. I will then use this list structure to update the map as the user pans around.
* 5 - Restaurant profile page.
  + 5.1 - Restaurant profile page content.
    - 5.1.1 - The page should contain the restaurant’s name, logo, description, food type and location.
    - 5.1.2 - At the bottom of the profile should be links to their website and also a link to view their menu.
    - 5.1.3 - There will be a way to view the available days and times that the restaurant you are viewing has available. If you select a time/day then the booking popup widget should display already filled out with these selections.
      * 5.1.3.1 - This view should show when the restaurant has no free voucher spaces left or is closed (will be done from reading from database).
    - 5.1.4 - There will also be a book now button which will just display the booking popup widget with the default contents.
    - 5.1.5 - A favourite button will be in the top right hand corner of the page which when pressed will add this restaurant to their list of favourites that they can then go to later.
  + 5.2 - Restaurant page backend.
    - 5.2.1 - This page will have to access the restaurants state that will contain the custom restaurants class. Inside each restaurant class will contain data such as the available times that the restaurant has to book so that the UI can show the correct data.
    - 5.2.2 - It is very important that this page is updated live (most of the pages will be) as multiple people could be trying to book the last remaining vouchers of a restaurant. If this is the case then the UI should be updated as often as possible.
* 6 - Vouchers page.
  + 6.1 - Current vouchers.
    - 6.1.1 - This tab will display a list of all your current vouchers that you haven’t used yet and are in the future. This will be the main place that you’ll visit when you show up to a restaurant that you’ve booked a voucher with.
    - 6.1.2 - When you click on the voucher it will show you some more details about the voucher. E.g. the restaurant it’s booked at, for how many people and what time and will display a QR code version of the voucher code.
    - 6.1.2 - You’ll be able to click on the restaurant’s profile from the vouchers page to find their contact details just in case you need to contact them.
    - 6.1.3 - Each list item will display some information about that voucher in brief: where its for, what time and how many people.
    - 6.1.4 - If a user has no current vouchers, a small message will be displayed in the center of the page saying ‘You currently have no vouchers’.
  + 6.2 - Expired vouchers.
    - 6.2.1 - This tab will display a list of all your vouchers that you either used, or expired without you using them (hopefully not the latter way!).
    - 6.2.2 - Each list item will show some content about that voucher: whether it expired or was used, where it was for, what time and for how many people.
    - 6.2.3 - You will also be able to navigate the restaurant’s profile and view more info by clicking on the list item as with the current vouchers tab.
    - 6.2.4 - If a user has no expired or past vouchers, a small message will be displayed in the center of the page saying ‘You have no expired or used vouchers’.
  + 6.3 - Vouchers backend.
    - 6.3.1 - I will create a custom voucher class that will contain all of the information about a voucher such as location, voucher id and date of purchases/use.
    - 6.3.2 - The state management for this page will need to only contain the vouchers of the currently logged in user. Therefore I will only need the users state, and inside of this class will contain the state for their vouchers.
* 7 - Settings page.
  + 7.1 - User account details page.
    - 7.1.1 - This page will allow the user to change the details of their account.
    - 7.1.2 - There will be input fields for a new password so that the user can change this.
      * 7.1.2.1 - This password must also meet the minimum length, if it does not a dialog box will appear notifying the user of their error.
    - 7.1.3 - There will be an input field so that the user can change their name.
    - 7.1.4 - This page will need to update the currently logged in user’s class and state. It is likely that this will be done when the database is updated. The stream of data being read from the database will cause the state of the user to update.
    - 7.1.5 - If the user hasn’t authenticated themself in a while, they will need to re-authenticate in order to change their password. A message box will appear notifying them of this with an input for their password.
  + 7.2 - Payment details page.
    - 7.2.1 - This is where the user has the option of entering in a card that will they can select to use for future payments. This means that they won’t have to enter in the entire card number every time they want to purchase a voucher.
    - 7.2.2 - There will first be a list of the user’s cards and the default card will be indicated with a small icon.
    - 7.2.3 - A button will be present on the page which will allow the user to add a new card.
      * 7.2.3.1 - When pressed a form will appear where the user can enter in their card details.
        + 7.2.3.1.1 - The card details will be verified when the user tries to save them. If they aren’t valid then a message will appear, notifying the user that the details they’ve provided aren’t valid.
        + 7.2.3.1.2 - If the details are valid a success message will appear and their card will be visible back on the payment details page.
    - 7.2.4 - On each card item, the user will be able to swipe the card away to initiate a delete function.
      * 7.2.4.1 - When the user swipes across a popup message will display, asking if the user is sure that they want to delete the card.
        + 7.2.4.1.1 - If the user confirms they wish to delete the card then that card will be completely removed from their account and a new default will be selected (if they have any other cards, else no default is selected).
    - 7.2.5 - The user can also tap on the list item to edit that card’s details. The inputted data will also be checked like it was when a new card was added.
    - 7.2.6 - I will need to create a custom card class to hold data about the user’s card when they are adding it. This card class will be used within the user class when interacting with the payment API.
* 8 - Menu widget.
  + 8.1 - The menu widget is a key component of the app. It will be present on almost every single page once the user has logged in. The menu widget will be a small circular button in the bottom right hand corner of the app. It is placed here for ergonomic reasons (most people are right handed, but even if they aren’t it is much easier to reach the bottom of the phone than the very top these days).
  + 8.2 - When clicked on it will expand to take up most of the available screen, like a side drawer, and will blur and darken the page in the background that the user was on.
  + 8.3 - When expanded there will be a small list of all the different pages that the user can navigate to: home, find a restaurant, vouchers and settings.
  + 8.4 - When one of these list items is clicked on, the app will navigate to this page and the menu widget will retract back down into the bottom right hand corner.
* 9 - Booking popup widget
  + 9.1 - Booking popup widget display and content
    - 9.1.1 - When it is displayed, it won’t take up the whole screen but will sit on top of, and darken and blur, the current page that the user in on.
    - 9.1.2 - There will be a button allowing the user to close the booking popup.
    - 9.1.3 - The popup will contain a few selections.
      * 9.1.3.1 - A selection for the date that you wish to attend.
      * 9.1.3.2 - A selection for the time that you wish to book for.
      * 9.1.3.3 - The number of people that will be attending.
    - 9.1.4 - Once the user has selected these values, they will then be able to press a book now button. On press, the button will cause a payment widget to be displayed and the user will either select a saved card or fill in one to be used for this payment.
      * 9.1.4.1 - If the user hasn’t selected a date or time then a small error message will appear letting them know that they must select some values.
      * 9.1.4.2 - When the user books a voucher, that voucher will appear in their current vouchers page and they will also get a confirmatory email.
  + 9.2 - Booking popup backend.
    - 9.2.1 - I will create a custom booking class. This class will compose of the currently logged in user object and a newly created voucher object.
    - 9.2.2 - When the user confirms and pays for a booking, this class will be used to communicate information to the payment API and also data from this class will be written into the database.

Information about the Firebase service: It is a NOSQL database, structured like a massive JSON tree or like sets of collections that contain many documents. This means my app will have a complex data model and structure with the database and queries will stretch across different collections and branches of the tree.

Throughout my models and other parts of my code I’ll also be able to include custom data structures such as hash tables and linked lists where I will maintain and code all the operations from scratch.

Data flow

Below is a diagram of some of the data flows that will happen within my app. As you can see, the Firebase store will be the most frequently used part of my app. This is because all information about the restaurants, vouchers and users is stored, fetched and continuously updated in the database.

As you can see from below, when a user purchases a voucher… CONTINUE HERE

