Milestone 0: Describe the problem that your application solves. *(Not graded)*

‘MitySG: Am I There Yet?’ is a localised public transport notification web app that allows commuters to be glued to their screens and yet be aware about their transit arrival. Doing away with simply listing bus arrival timings, ‘MitySG’ pushes notifications the user when their user’s bus is arriving and when they are arriving at their destination (both Bus and MRT).

As mobile device usage becomes more prevalent and invasive, commuters tend not notice their surroundings. The distraction posed by mobile devices have led to many relatable instances of commuters missing their stop or not noticing that their bus has arrived. That being said, we would much rather spend our attention on our devices; why look up when you got emails to send and/or are obliterating your opponents’ castles?

In the UK, a study was conducted in 2012 that up to 20 million instances across the year of passengers missing their bus or their stop due to ‘digital distraction’[1]. Singapore is no different.

Rather than siding with the Luddites and discourage individuals from using their electronic devices, we developed a Web App that will notify you of your transit on the very screen you will be staring at.

Our group has sought to address the daily struggle of missing your commute as you get carried away with your devices.

[1] http://www.dailymail.co.uk/sciencetech/article-2341068/Digital-distraction-causes-20-million-passengers-miss-bus-train-stop-year.html

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?

‘MitySG’ is a transit notification web application that uses a combination of Google Maps and LTA APIs to ensure users are informed about their transit arrival. The goal of ‘MitySG’ is to ensure that that users have a ‘worry-free journey’ while being unintrusive to their mobile experience as long as they set up an ‘Am I There Yet’ (MITY) notification.

1. Functionality

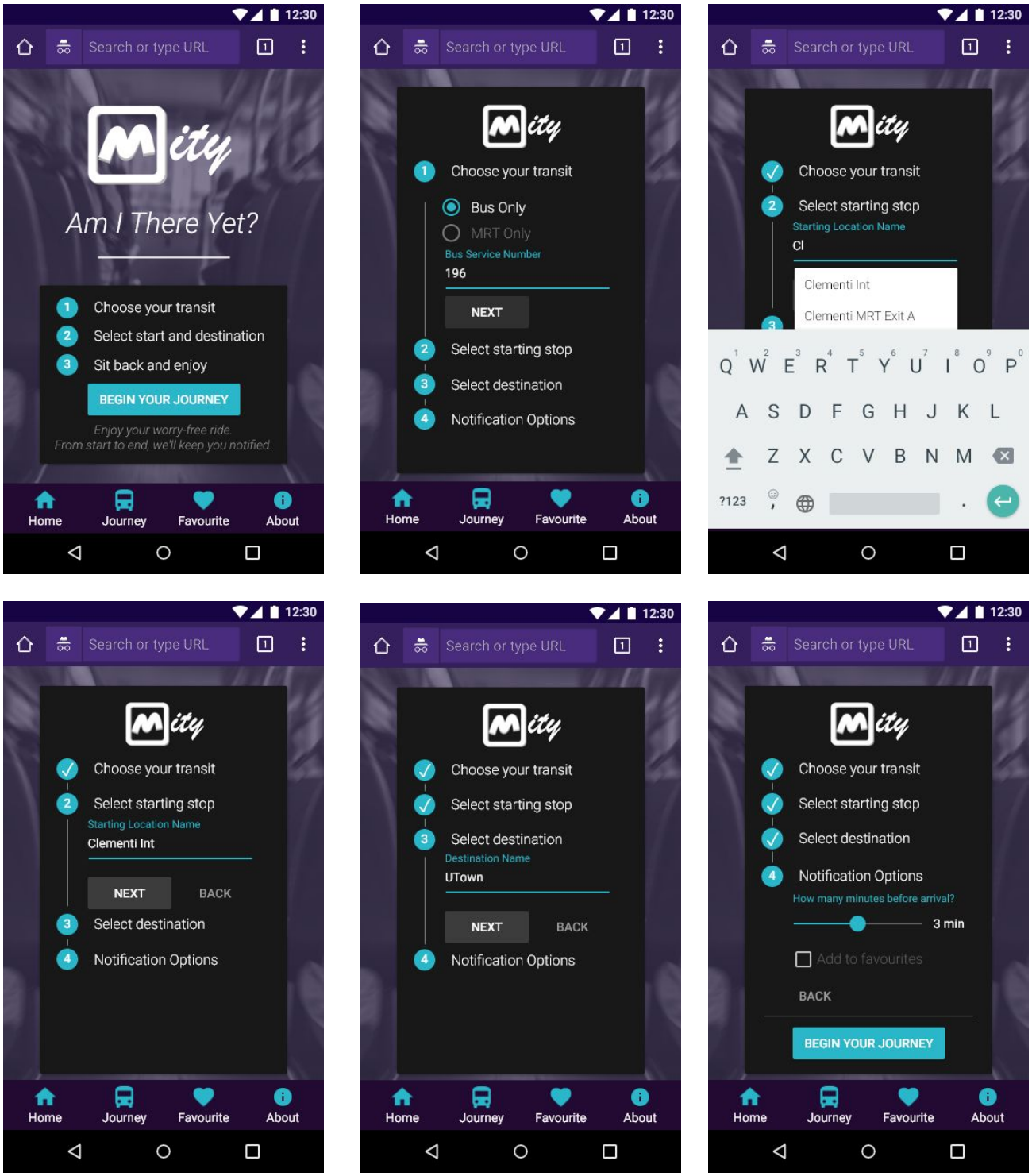
‘MitySG’ pushes notifications the user when their user’s bus is arriving and when they are arriving at their destination (both Bus and MRT). This is achieved by a combination of Google Maps and LTA APIs, that allows us to get around a limitation of Progressive Web Apps (i.e. the lack of GPS functionality when the app is inactive). Using various service worker functions, ‘MitySG’ pushes notifications to your device.

By using 4 simple and straightforward inputs from the user [i.e. Transit Type, Bus Service No (if applicable), Starting Point (fixed list), and Destination (fixed list)], our server will read data (specifically arrival timings of bus no. at bus stop from LTA, AND Google Maps Transit API for ETA of public transport) from both APIs to determine when to push notifications to our user.

When the buffer crosses, our ‘Mity’ will push a notification to inform you that your bus is arriving (encouraging you to ‘Get off your phone and Get onboard!’). LTA’s bus arrival API provides the arrival time and geolocation of the next 3 buses arriving at any particular bus stop. We then make use of this to map the incoming bus at next stop to the bus at previous stop through the geolocation of the arriving buses, thereby enabling us to track the geolocation

of the exact bus that the user has boarded throughout the journey, and subsequently prompt the user before arriving at the destination.

For the MRT stations, we make use of Google Maps API to estimate the travel time and prompt user before the train arrives. As trains are not susceptible to varying traffic conditions, Google Maps API estimate is accurate enough for our purpose.



Setting up a 'MITY' notification is simple and fast (text fields have auto-complete). You have the option of saving the trip for convenient re-use too!

2. As a Mobile Cloud App

Firstly, by being a mobile cloud app, MitySG takes advantage of the wealth of external REST APIs on the internet and uses our backend servers to track which bus you are on, follow the bus as it travels and then send a request to push a notification to the client. Hence, the majority of computation and external API requests are done on the server side, thereby fully exploiting cloud computing.

Secondly, it makes the most sense for MitySG to be a mobile cloud app for ease of use to users. It is readily accessible and does not require any downloads in order to use, plus it does not drain your device's battery as the calculations are done on the server-side instead.

Milestone 2: Describe your target users. Explain how you plan to promote your application to attract your target users.

‘MitySG’ targets your typical Singaporean commuter whose mobile devices have become an essential part of transit.

Our application emphasises on the experience of a ‘worry-free journey’; no more anxiety of missing your commute. With ‘MitySG’, we hope to put an end to panic of looking up from your screen and/or constantly checking public transport applications. We aim to attract users through functionality, accessibility and implications. Below are 4 main attractions of our app that sets it apart from any other.

1. Solving Societal Issues Using Technology

The biggest selling point of ‘MitySG’ is that it deals with the modern day-to-day issue of catching your commute while not interrupting your mobile device usage experience. We cannot deny that we are constantly glued to our phones. Rather than change our ways, technology should adapt to our lifestyle. The experience of using ‘MitySG’ is intended to be seamless. No need to constantly exit change applications. No need to look away from your phone.

2. No Need to Download or Register for an Account

As a web application, ‘MitySG’ has the major selling point of not requiring any downloading or an account to use our features. Recent consumer trend studies have exposed an increasing reluctance to install applications (some have said we have reached ‘peak app’). There is a perceived installation-barrier amongst consumers preventing them from using applications.

‘MitySG’ is intended to serve as a tool. Users of such applications demand fast delivery and service. By removing the barriers of installation and making of accounts, users dive straight to our features, providing instant gratification.

In addition, by making ‘MitySG’ a web application, it is easily accessible and easily discovered. Typing our URL is all it that is needed to use our application. Sharing this application with your friends is also as simple. If you want to re-use ‘MitySG’ you can simply add a shortcut to our app on your homescreen.

A possible expansion of ‘MitySG’ is to have QR codes at major bus stops and MRT stations to have quick access to ‘MitySG’ to make the experience even more seamless.

3. Save Space and Battery Usage

Due to limited phone storage (gone are the days of ejectable micro SD cards), phone users are pressured by the lack of space on their phones. ‘MitySG’ does not eat into your storage as a mobile cloud application. Data is stored on the server rather than on your device. Our backend will to the ETA calculations instead of your phone too.

Unlike many geolocation-based applications, ‘MitySG’ does not drain your battery as you do not need to constantly update your geolocation while using ‘MitySG’. Moreover, the application doesn’t require running in the background to function. There is also no need to open our application for ‘MITY’ notifications to be pushed.

‘MitySG’ requires minimal processing power and almost no internal storage capacity.

4. Easy to Use

‘MitySG’ is very easy to use. No need for massive amounts of textual instructions to teach you how to use the application. Every function is designed to be self-explanatory and obvious. Step-by-step input fields are minimal yet impactful with sufficient information.

Bottom navigation bar makes the application easy to navigate within and for users to be clear what functions are available. All buttons are designed to be straightforward with clear purpose. All input texts come with auto-complete so you need not type a whole field out or fumble with wrong names.

We remember your saved trips! If you tend to always make repeated journeys on the same route and want a ‘MITY’ notification for it, simply save your trips to start the trip again in one button rather than repeating the whole process.

Promotion to attract users:

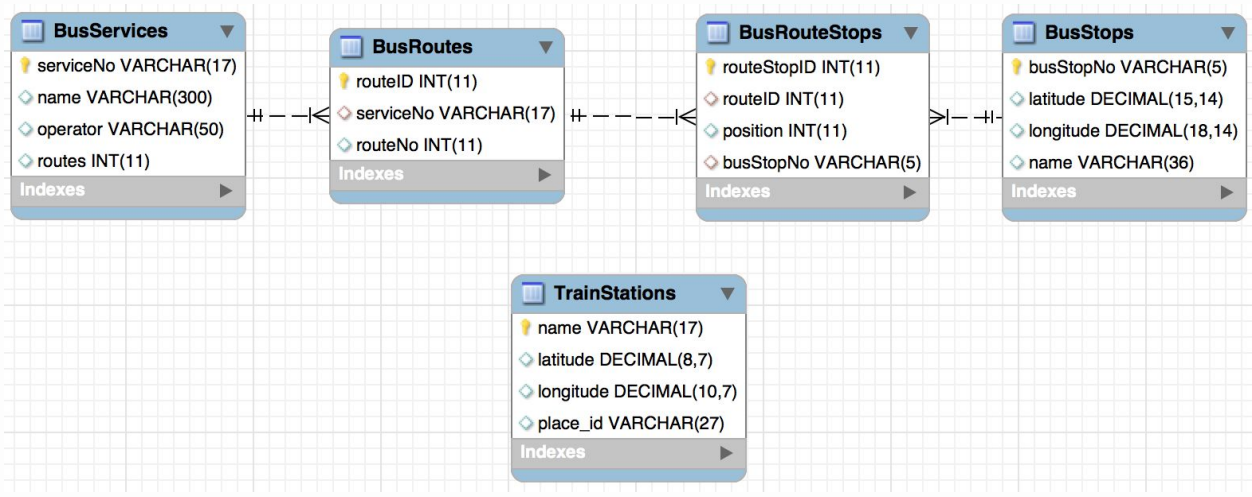
1. Word-of-Mouth

With a large focus on producing a better user experience and with functionalities that mitigates modern day-to-day problems, we push for organic growth. Putting in effort into your product is half the marketing challenge.

2. QR Code at Bus Stops / Promotional Campaigns

For a major publicity, the best campaign for ‘MitySG’ will be placing QR code stickers at major transportation points. Users can immediately scan the QR code to link them to ‘MitySG’ and we can onboard them as they will be using the application immediately. This would be the most effective way to promote our product.

Milestone 3: Draw an Entity-Relationship diagram for your database schema.



Milestone 4: Design and document all your REST API. The documentation should describe the requests in terms of the triplet mentioned above. Do provide us with a brief 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.)

Server URL: https://mityserver.tk/

Request Method + Relative URL	Request Parameters	Response Value(s)	Description
GET /busStops	None	JSON Array of all Bus Stop ID, Description, Latitude and Longitude. {"02161":{"description":"Opp Millenia Twr","longitude":"103.86093156188176","latitude":"1.29268553508737"}, … "76391":{"description":"Blk 418","longitude":"103.94754113603500","latitude":"1.35844376751138"}}	Returns all bus stops and their data
GET /busStops/{busStopNo}	None	JSON of a single Bus Stop ID, Description, Latitude and Longitude {"76391":{"description":"Blk 418","longitude":"103.94754113603500","latitude":"1.35844376751138"}	Returns the data for the bus stop
GET /busServices	None	JSON Array of all Bus Services with an Array of all their Stop IDs {"325":["64009","64019","63349","63359","64399","64389","64421","64039","64369","64431","64441","64461","66499","66489","64119","64491","64081","64091","66491","64469","64449","64439","64361","64031","64429","64381","64391","63351","63341","63061","64011","64009"], …	Returns all busServices and their routes

		"324":["64009","64041","64051","62271","62251","62261","64059","64049","64009"]}	
GET /busServices/{serviceNo}	None	JSON of a single Bus Service with an Array of all its Stop IDs {"4":["75009","76191","76201","76211","76221","76231","76241","98301","98311","97049","97039","97141","97301","97311","97331","97321","97059","98319","98309","76249","76239","76229","76219","76209","76199","75009"]}	Returns the route for bus with serviceNo
GET /busArrival/{serviceNo}?stop={stopNo}	None	JSON of the Bus Arrival Data with the data for the next 3 buses that includes the bus' origin code, destination code, estimated latitude, estimated longitude {"serviceNo":"183","nextBus":{"originCode":"28009","destinationCode":"28009","estimatedArrival":{"offset":{"totalSeconds":28800,"id":"+08:00","rules":{"fixedOffset":true,"transitions":[],"transitionRules":[]}},"zone":{"totalSeconds":28800,"id":"+08:00","rules":{"fixedOffset":true,"transitions":[],"transitionRules":[]}},"dayOfYear":272,"dayOfWeek":"FRIDAY","month":"SEPTEMBER","dayOfMonth":29,"year":2017,"monthValue":9,"hour":11,"minute":29,"second":27,"nano":0,"chronology":{"calendarType":"iso8601","id":"ISO"}},"latitude":1.29001784324646,"longitude":103.77471160888672,"estimatedWait":6,"location":{"lat":1.29001784324646,"lon":103.77471160888672}},"nextBus2":{"originCode":"28009","destinationCode":"28009","estimatedArrival":{"offset":{"totalSeconds":28800,"id":"+08:00","rules":{"fixedOffset":true,"transitions":[],"transitionRules":[]}},"zone":{"totalSeconds":28800,"id":"+08:00","rules":{"fixedOffset":true,"transitions":[],"transitionRules":[]}},"dayOfYear":272,"dayOfWeek":"FRIDAY","month":"SEPTE	Returns the bus arrival data for the service no at the stop no

		MBER","dayOfMonth":29,"year":2017,"monthValue":9,"hour":11,"minute":48,"second":41,"nano":0,"chronology":{"calendarType":"iso8601","id":"ISO"}},"latitude":1.3121027946472168,"longitude":103.77162170410156,"estimatedWait":25,"location":{"lat":1.3121027946472168,"lon":103.77162170410156}},"nextBus3":{"originCode":"28009","destinationCode":"28009","estimatedArrival":{"offset":{"totalSeconds":28800,"id":"+08:00","rules":{"fixedOffset":true,"transitions":[],"transitionRules":[]}},"zone":{"totalSeconds":28800,"id":"+08:00","rules":{"fixedOffset":true,"transitions":[],"transitionRules":[]}},"dayOfYear":272,"dayOfWeek":"FRIDAY","month":"SEPTEMBER","dayOfMonth":29,"year":2017,"monthValue":9,"hour":12,"minute":0,"second":40,"nano":0,"chronology":{"calendarType":"iso8601","id":"ISO"}},"latitude":1.3312275409698486,"longitude":103.75811004638672,"estimatedWait":37,"location":{"lat":1.3312275409698486,"lon":103.75811004638672}}}	
GET /busTiming/{startLat,startLng}/{endLat,endLng}/	None	Int of the Bus Arrival Time in minutes 3	Returns the estimated duration it takes via bus from the start loc to end loc (take note that you must include the / at the end to prevent your endLng from being truncated) Also note that this is not necessarily the exact estimation for the bus, but the shortest amount of time it takes for any bus to go from start to end
POST /busTracker/{serviceNo}?start={startStopNo}&end={endStopNo}&alert=	Pass the subscription object in the request body	JSON containing the latitude and longitude of the bus as of the time that the user requests to be alerted at	Tracks the service no from start stop no to end stop no. Send push noti to user when first

{noOfMins}		{“latitude”:1.34123456, “longitude”: 135.1230688}	bus arrives at start stop no, and no of mins before it arrives at end stop no. Takes in the subscription json as a request body
GET /trainStations	None	JSON containing all train stations including their name, google place id, latitude and longitude {"Eunos":{"id":"ChIJA08Q0gQY2jERrq60-mOgmxQ","latitude":1.3197777,"longitude":103.9028587}, … "Labrador Park":{"id":"ChIJszi8l8Eb2jERMYr0lHRAEBo","latitude":1.2722052,"longitude":103.8024245},"Yishun":{"id":"ChIJc6Z8jG8U2jERv8M81RFvr7Q","latitude":1.4293199,"longitude":103.8350278}}	Returns all train stations and their data
GET /trainStations/{name}	None	JSON containing a single train stations including its name, google place id, latitude and longitude {"id":"ChIJc6Z8jG8U2jERv8M81RFvr7Q","latitude":1.4293199,"longitude":103.8350278}	Returns data for a trains station according to the exact name (from Google Maps)
GET /trainTiming/{startPlaceID}/{endPlaceID}	None	Int of the Train Arrival Time in minutes 5	Returns the number of minutes to get from start to end station
POST /trainTracker/{startPlaceID}/{endPlaceID}?alert={noOfMins}	Pass the subscription object in the request body	Int of how many more minutes till the train should arrive 3	Calculates travel time from start to end station. Sends push noti to user noOf Mins before he will reach. Completely timer based. Takes in the subscription json as a request body

Our application follows client-server architecture, as there are proper separation of concerns between the server and the browser. The client-side takes in user input(bus/train service, bus stops, etc) and sends it to the server through http request. After receiving the requests, the server takes care of the main logic of the application such as bus tracking and calculating the estimated time of arrival.

Then the server sends the response to the client, and the browser displays necessary information to the user.

Our server is stateless. The server does not store any authentication data from the user. Each session is defined by each trip the user makes, using the push notification subscription token as the identification. When the trip is over, the authentication token is discarded.

The client side cache html/css data and some of the GET response data to speed up the loading process for the subsequent sessions.
The server can also cache the MRT / bus stops and services requests easily if necessary.

Our REST API provides uniform interface as resources the client is looking for is uniquely identified by the URL.

We follow the layered system principle as at any time client cannot tell if it is connected to the end server or to an intermediate.

Milestone 5: 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 explain how it works.

- select * from BusStops
- select * from BusStops stops WHERE busStopNo='{no}'
- select * from TrainStations
- select * from TrainStations WHERE name=”{name}’
- select bs.serviceNo, brs.busStopNo, br.routeNo, bs.routes FROM BusRouteStops brs, BusRoutes br, BusServices bs WHERE bs.serviceNo = br.serviceNo AND br.routeID = brs.routeID ORDER BY brs.position;
- select brs.busStopNo, br.routeNo FROM BusRouteStops brs, BusRoutes br, BusServices bs WHERE bs.serviceNo="{no}" AND bs.serviceNo = br.serviceNo AND br.routeID = brs.routeID ORDER BY brs.position;

The first two queries, extract out information about bus stops from the database, and narrows it down with a WHERE clause to only a single bus stop based on its unique stop no. The next two queries do a similar action, but extracts the data for train stations instead.

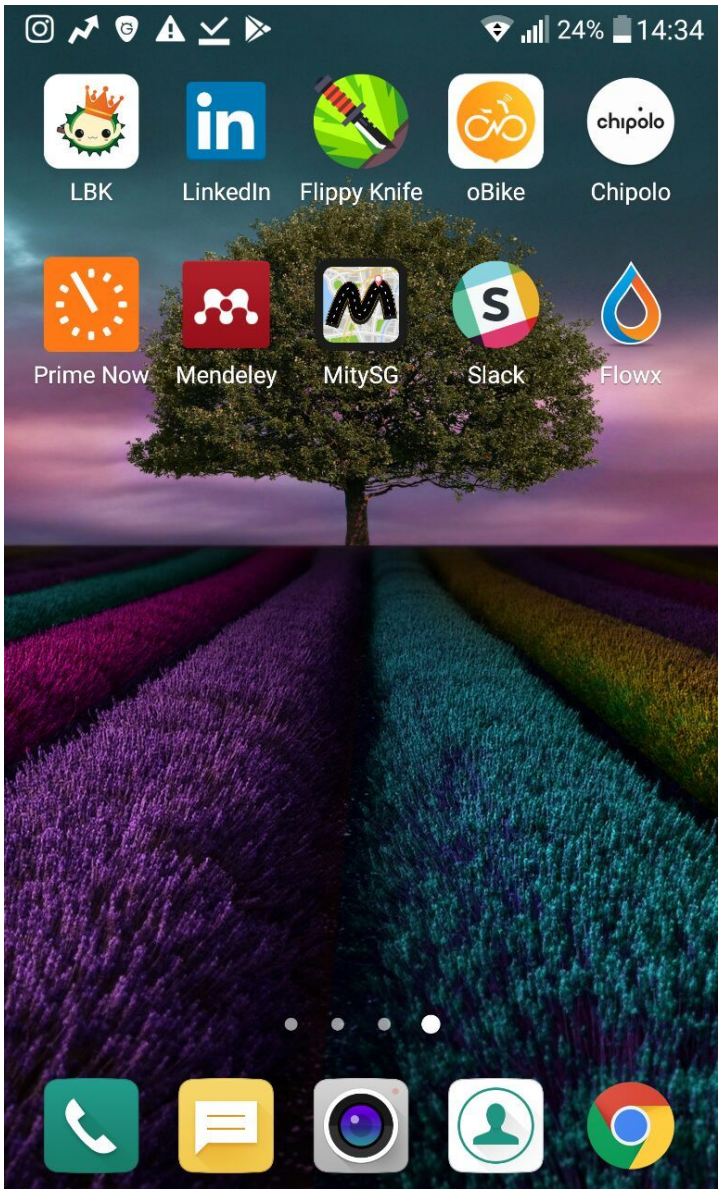
The next query extracts out all the service numbers, their bus stop numbers for the routes, and whether it’s the first or second route in order to populate the data for the entire system. The last query extracts out the route information (bus stop numbers) for a specific bus service.

Milestone 6: 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 Screen

Logo for Web Pages

Logo for Homescreen



Adding to home screen working as intended!

Milestone 7: 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.

CSS has long plagued web developers with its cascading nature and globally scoped class names. In a large application with multiple developers, it becomes difficult to ensure that the styles that one declares is only applied to the intended components, especially if each of the developers use their own naming conventions. Developers often ended up using deeply nested selectors to mitigate global scope issues, but it comes with its own problems such as performance issues, lower reusability etc.

Block Element Modified (BEM) is a popular CSS methodology that attempts to solve the problem by establishing standard naming conventions for className to ensure that it is unique, thereby reducing the risk of specificity clash.

However, this only works if all the developers actually adhere to the naming convention.Its verbosity meant lower productivity and higher chance of making typos.

CSS Modules is a postprocessor that solves the global scope issue by automatically scoping all declared styles locally. When necessary, a :global tag can be used to specifically declare the style as globally scoped.

Moreover, the `composes` keyword allows one class to “inherit” styles from another class, thereby promoting reuse through composition.

However, it comes with 2 problems:

1. Performance penalty

   According to https://github.com/gajus/babel-plugin-react-css-modules#performance, react-css-modules is 61% slower than default CSS. To mitigate this, we ended up with choosing **babel-plugin-react-css-modules**, a more or less drop-in replacement (despite the doc stating otherwise) with a much lower performance penalty of 4-8%.

2. Overriding global css from external libraries

   There are no perfect solution to this, but a couple of workaround has been developed, including @import, inline styles, leveraging on webpack loaders etc. As we choose material-ui v0.19, we had to override the theme with inline styling, and make use of CSS Modules for our own component styles.

Milestone 8: 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. Name 3 best practices for adopting HTTPS for your application. Explain the term "certificate pinning" and discuss the pros and cons of adopting it.

---

1. Ensure Sufficient Hostname Coverage

When registering the SSL certificate, ensure that the certificates cover all the domain names of the site. For example, the certificate should work with and without the *www* prefix.

2. Use Session Resumption for Better Performance

Session resumption is a performance-optimization technique that makes it possible to save the results of costly cryptographic operations and to reuse them for a period of time. A disabled or nonfunctional session resumption mechanism may introduce a significant performance penalty.

3. Set-up an Automated HTTPS Monitoring Report

Use SEMrush integrated with Google Search Console to set-up an HTTPs report and get a complete overview about the state of the HTTPS certificate. The report shows all possible issues and how to fix them. The report also tells if any subdomain that doesn't support HSTS.

With certificate pinning, when the server establishes a connection with a client for the first time, the server provides the client with a SSL certificate inside the application that is pinned by the client. The pinned certificate will be used for verification of the server's identity. Afterwards, every time the browser encounters the same host, the browser will check the pinned key to verify if it matches the host's key. If they do not match, the browser terminates the connection. The Pros is that it prevents a forged certificate from passing off as the real one, if let's say your Certificate Authority Store was compromised. The cons however, is that you assume the first time you visit the site, that it is legitimate. If you visit the compromised site on the first time, certificate pinning cannot help you.

Milestone 9: 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.

In offline, the user still can access most of Mity's UI resources, as they are cached by ServiceWorker. The train and bus data are also stored in the Redux store and subsequently cached to local storage via redux-persist. User will therefore be able to view train stations and bus stops and add/remove favourites so that they can quickly start a journey when they go online.

However, the user is not able to receive notifications from Mity server while offline, as web push notification can only be received with Internet connection.

As user is likely only going to use the application on his mobile, we simply store the favourites in local cache, hence there is no need for synchronization with server when application is used offline. Backup of favourites can also be done through importing/exporting rather than adding the complication of a login to synchronize with server.

However, if server synchronization is desirable, it can simply be implemented by comparing the last modified time of the favourites list on the server vs the one in the local cache when the user goes online, and subsequently replace the resources with the more recently modified version. More complex conflict resolution (like that of Dropbox) is not critical to this app.

Milestone 10: 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.

Session-based Authentication

- On authentication, client receives a session_id which is stored in a cookie and attached to subsequent http requests.
- In this method, the server is stateful. The server identifies the user with the given string of session_id, and the server has to keep the list of session_ids for future authentications.
- Session-based Authentication could be not scalable because the server has to store session_id for each of the user identification. In a large user pool, it can drain the server's storage.
- Users are susceptible to CSRF attacks since they can already be authenticated with one website site and the session stored in the cookie could be compromised when visiting other sites.

Token-based Authentication

- Every single request creates a new token for authentication, created and signed by the application itself. The token is attached to the request header.
- The server is stateless, as it is not storing any data for user identification. Better scalability.
- The token is not stored anywhere, not in cookie or localstorage. Hence it is safer from CSRF attacks.
- Token-based authentication has better extendability, since the token can be created and passed among different services to identify the user.

Our application uses Token-based Authentication provided by ServiceWorker's Push API, as our application does not require any log in to the user. The token is created for each trip, that only lasts for the trip itself. In other words, the user is identified by each trip, and the server uses the token to identify where to send the push notification.

Milestone 11: 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.

We have chosen Material-UI v0.19 as the framework for the following reasons:

1. Unlike frameworks like Ionic, Vuetify, it uses React.js, of which our front-end developer is very familiar with. Furthermore, React.js is very suitable for performant PWAs as it make use of virtual DOM comparison to avoid unnecessary manipulation of the actual DOM.
2. Unlike frameworks like Framework7, it follows material design by Google, hence the UI/UX will be very intuitive to most users.
3. Since all browsers on iOS do not support service workers and the core functionality of our app relies on it, we expect our users to be mainly Android users. Hence the material design adopted by material-ui will feel right at home to our users.
4. It is well documented, with huge community support, and the learning curve is gentle.

Furthermore, We looked through material-ui components and found it to fulfill all our needs for the application.

The core principles of UI/UX are well documented and have become industry standard -- applications that do not meet the principles are discarded by users. Mobile site design requires more specialised tenets that are the result of core UI/UX principles. Ultimately, mobile sites are intended to be goal-oriented. Actions have to be clear and objectives should be met succinctly without intervention.

In the table below, we have listed the principles we have adopted, catergorised by the overarching UI/UX principles.

UI/UX Design Principles	Mobile Site Design Principles
Clarity	Clear Navigation [All]
	Self-Explanatory Design [All]
Flexibility	Optimise Site For Mobile [All]
Familiarity	Material Design [All]
Efficiency	Calls-to-Action Front and Center [1 - 5]
	Easy to Get Back to Homepage [All]
Consistency and Structure	Keep User in a Single Window [All]
Delight	Minimise Text to Read [1 - 7]
	Minimal User Input [2 - 5]

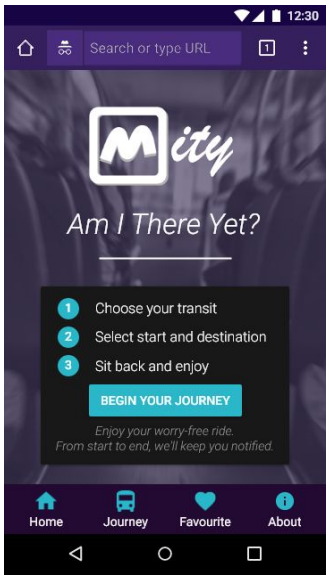


Fig. 1

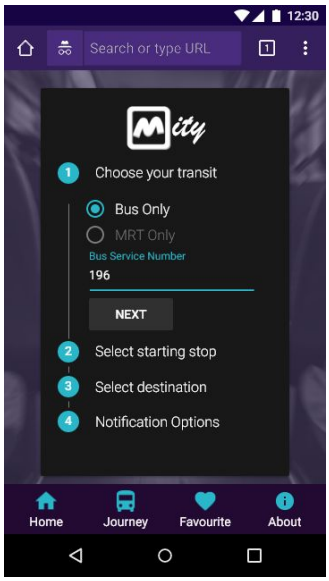


Fig. 2

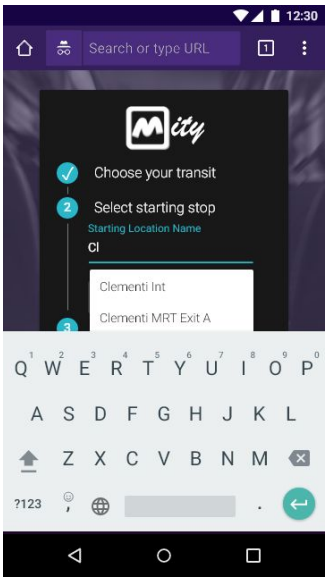


Fig. 3

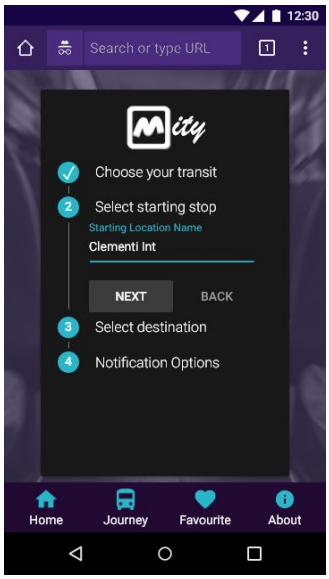


Fig. 4

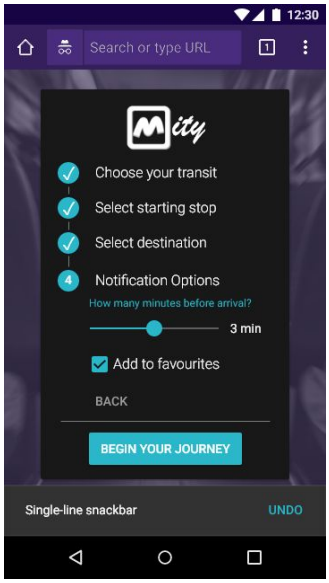


Fig. 5

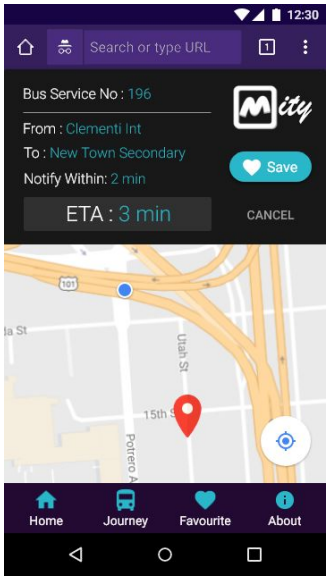


Fig. 6

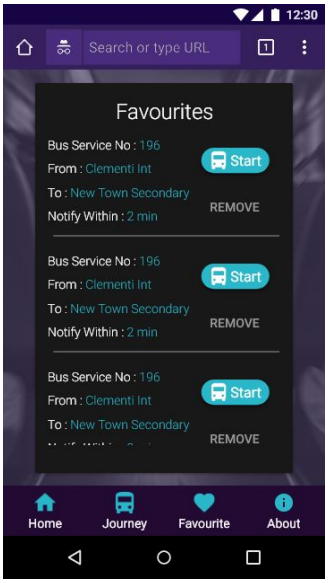


Fig. 7

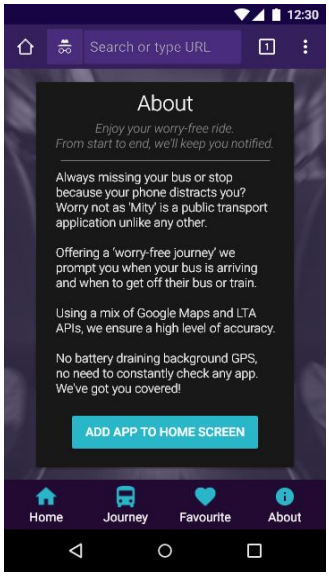


Fig. 8

1. Clarity - Clear Navigation [All]

At any single moment you are on 'MitySG' (except splash screen), there is consistently a bottom navigation bar (only current tab will have blue icon;

inactive tabs are greyed). Firstly, it provides very clear indication of which section of our app you are on. Next, it is easy to jump to other tabs depending on the user's demands (set up a new journey or go straight to favourites).

2. Clarity - Self-Explanatory Design [All]

Although we do not have a landing page that explicitly describes everything that 'MitySG', our application keeps every action simple and obvious. A principle in mobile site design is to ensure users can quickly work their way around your page. This is apparent in all our screens.

3. Flexibility - Optimise Site for Mobile [All]

As we develop 'MitySG' as a Progressive Web Application, a priority was to make the app responsive. Essentially, all screens should be optimised to be viewed across all mobile devices and even on desktop.

4. Familiarity - Material Design [All]

Using Material UI, our application is based on 'Material Design', a design style popularised by Google and now ubiquitous across the most popular apps. Our application embodies this familiarity in all of our screens.

5. Efficiency - Calls-to-Action Front and Center [1 - 5]

From figures 1 to 5, every action required from the user is always made clear and obvious to the user.

6. Efficiency - Easy to Get Back to Homepage [All]

As mentioned above the bottom navigation bar, going back to the home page is simple. No need to hunt for a way back; no need to clear back many times.

7. Consistency and Structure - Keep User in A Single Window [All]

Apart from after setting your ‘MITY’ notification, where you can just exit your application, every action within ‘MitySG’ happens in one window, separated by content tabs. No fumbling into other pages or windows.

8. Delight - Minimise Text to Read [1 - 7]

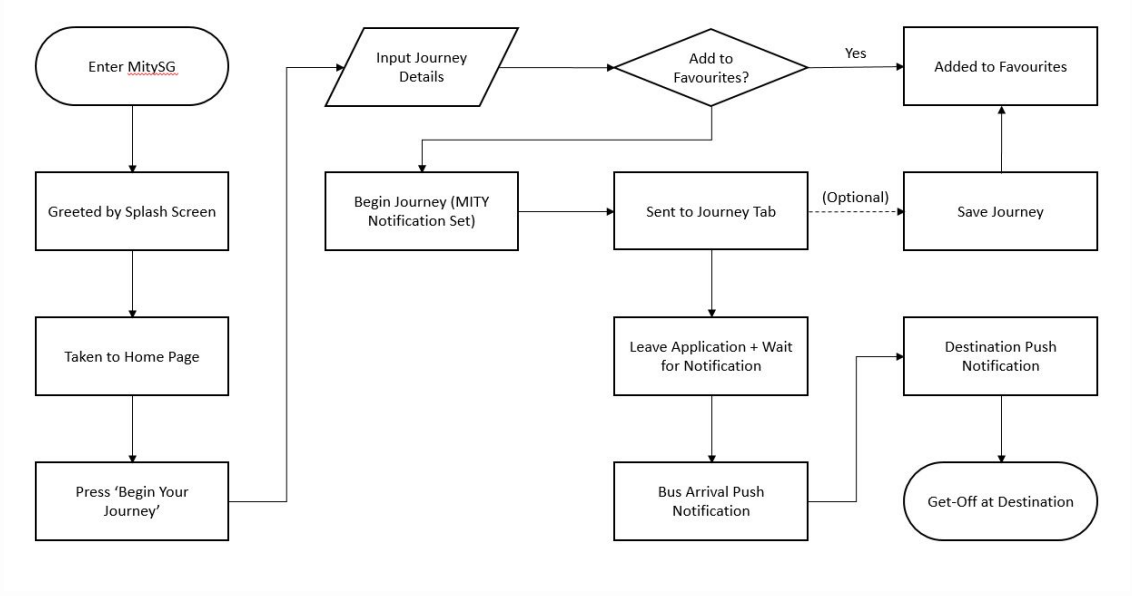
About from the ‘About’ tab in Fig. 8, all text is kept to a minimal. Users do not want to be greeted by large chunks of text. The amount of text on our app is just enough to know what needs to be done and what will happen.

9. Delight - Minimal User Input [2 - 5]

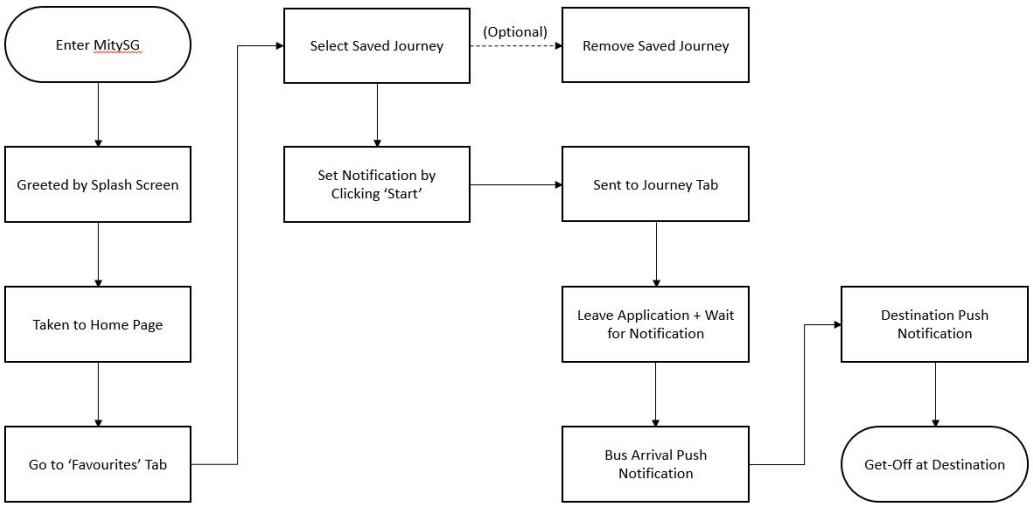
Every input field has auto-complete with data values based on stored names in our database. There are as many fields as minimally required.

Milestone 12: 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.

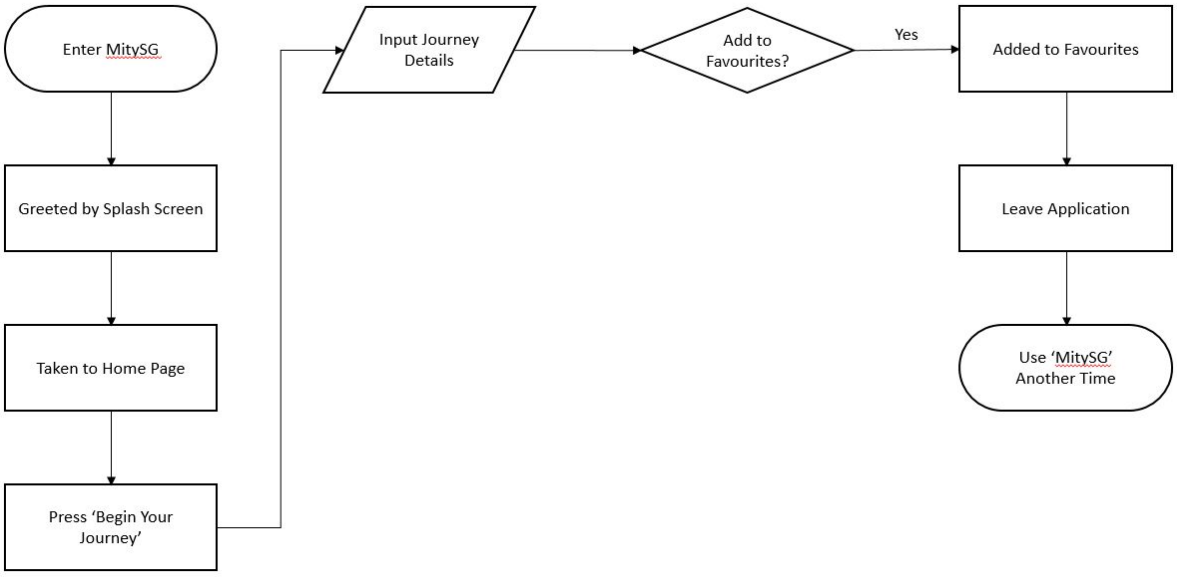
1. Normal User (First Time or New Journey)



2. Repeated User (Used MitySG Before and Saved)



3. Setting Journey In Advance (Pre-Setting or Exploring)



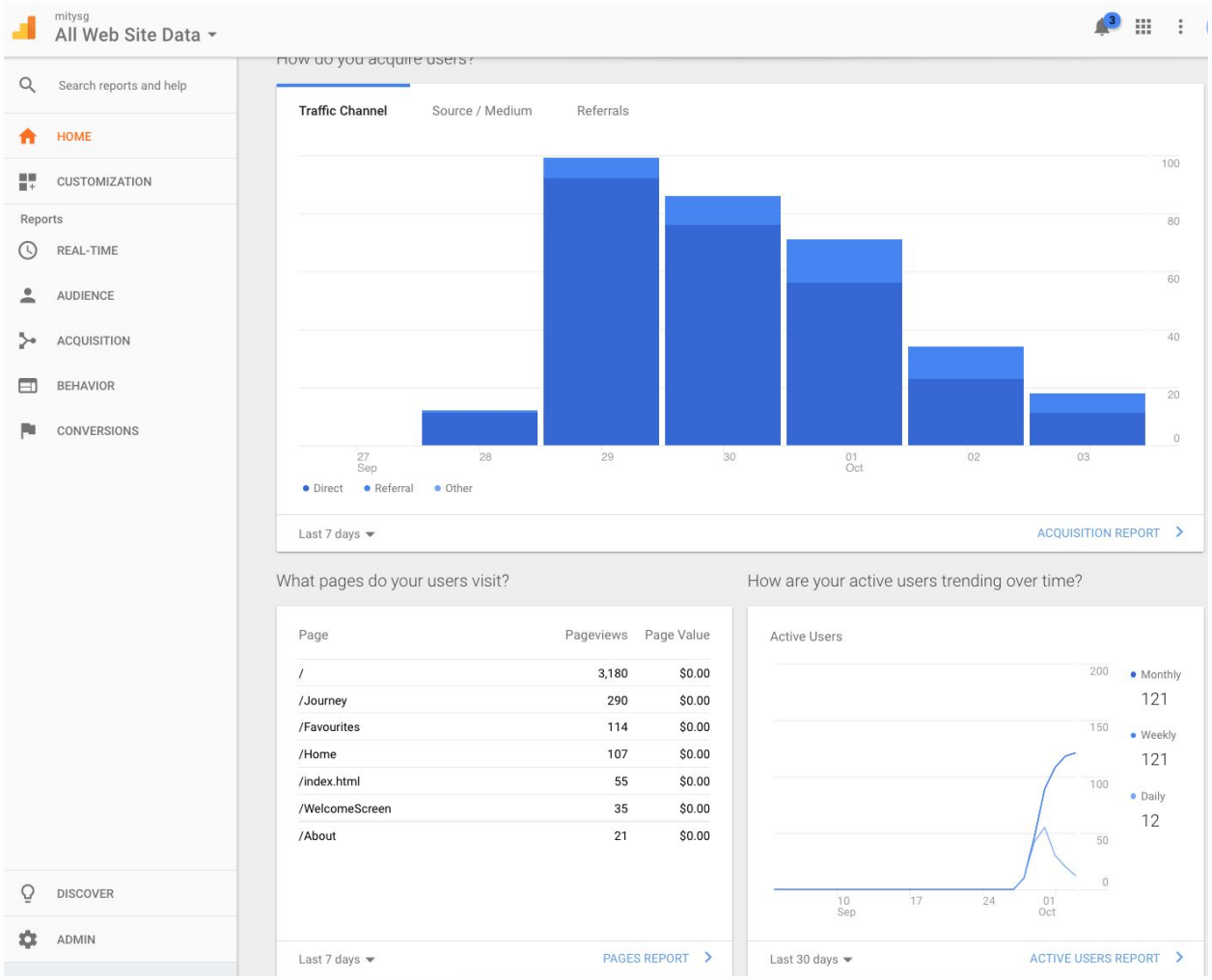
For our application, we have identified 3 main types of users. Firstly, normal users who want to use MitySG for the first time or for a one-time occasion.

Secondly, we have targeted repeated users to make their experience less troublesome and more intuitive. They simply need to go to their favourites tap and within a click, they can run their journey again. Saving journeys will simply use cookies.

Lastly, for users who may not be wanting to start a journey just yet or are simply exploring the application, they can set their journey up in advance without activating 'MITY' notifications. Users can access this saved journeys whenever they want without logging in to any accounts.

Alternative workflows could involve logging in to save/access journeys. This is not ideal as it adds unnecessary layers and barriers to using our application. Why tag your saved journeys to an account when you can use cookies to store them?

Milestone 13: 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.



Milestone 16: Make use of the Geolocation API in your application. (Optional)

We use Geolocation to show the user their current location if they have not started a trip. Once they start the trip, we also make use of geolocation to update the estimated arrival time whenever user checks the app so that they get a better estimate of the ETA.