Tutorific

https://tutorific.herokuapp.com/

CS3216 Assignment 3 Final Write-Up

Group 6

Cao Wenjie
Chrystal Quek Wan Qi
Goh Siau Chak
Seow Alex

Contents Page

Phase 1: Design

Milestone 0

Describe the problem that your application solves.

As of 2019, the tuition industry in Singapore alone is estimated to be worth around $1.4 billion. On one hand, we have university students and Junior College graduates who are looking to use their knowledge from pre-university education to earn some money. On the other hand, we have many parents who want their children to do well in school. It is evident that both the demand and the supply exist in this market.

Budding tutors currently find their clients via three methods, tuition agencies, messaging app channels and word of mouth. Similarly, parents use these three outlets to find prospective tutors for their children.

Tuition agencies handle most of the work of searching for a tutor or tutee for its clients by spreading listings via their network and connecting tutors and parents with matching wants and haves. However, these agencies usually charge a commission fee of about two weeks of lessons' worth of pay. This usually amounts to $50 - $150 depending on the hourly rate charged by the tutor.

Messaging app channels that we found (@sgtuitions and @astartutors on Telegram) are mostly catered to parents who put up listings and wait for tutors to apply. While this generally makes things easier for the parents, one downside is that parents whose children are in urgent need of a tutor do not have the option of proactively looking for one. Instead, they have to passively wait for tutors to find and apply to their listing.

While searching for a tutee via word of mouth may allow tutors to avoid the commission fee, this is generally a slower process for both tutors and parents because the average person only has so many contacts and connections.

It is clear that the current means of searching for tutors and tutees, particularly for one on one tuition, is problematic and lacking in various aspects. Tutorific is designed to fill this gap in the market. We discuss in further detail how Tutorific will tackle these issues in the separate pitch document.

Milestone 1

Describe your application and explain how you intend to exploit the characteristics of mobile cloud computing to achieve your application's objectives.

Tutorific is a centralised platform for both tutors and parents that aims to help university students and JC/Poly/ITE graduates find pre-university students to tutor, and for parents of pre-university students to look for private one-to-one tuition. It is completely free to use and allows both tutors and parents to create listings. It boasts granular filters and an in-app chat feature to help users specify exactly what they are searching for and contact the other party immediately when they find a listing they like. It leverages social media sites and messaging apps to help users quickly get more eyes on their listing by sharing it on these platforms.

A core part of Tutorific is our chat functionality which allows potential tutor and tutee/parent to communicate to set expectations and decide on specific arrangements before committing to a private tuition. Thus, Tutorific has to be implemented as a mobile application for users to receive real-time messages and notifications from the chat wherever they are to allow for more efficient communication between both parties.

Furthermore, Tutorific is implemented as a Progressive Web App (PWA) as this allows anyone to easily view, search and add listings. Users do not need to install an application to view or add a listing, which may be a barrier to using Tutorific. Additionally, given that Tutorific is a PWA, it provides offline capabilities, allowing users to still use the application when offline.

Tutorific also leverages cloud computing to provide the most up to date tutor and tutee listings for users. As a mobile application, users have the convenience to easily upload a listing from anywhere and this information can immediately be viewed by all other users through a central server. This speeds up the tutor/tutee-finding process, allowing for quicker matches.

Furthermore, as our user base grows, we can expect a large increase in listings but with cloud computing, all this data can be stored in the cloud without stretching the limited storage capacity of mobile phones. Heavy processing, such as complex filtering across various fields over large amounts of listings, can also be offloaded to the server, thus, the mobile application only needs to receive and display the relevant listings requested by the user without burdening the phone with excessive computation. This allows users (both tutors and tutees) to quickly view listings that fit their requirements and find a potential match.

Milestone 2

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

Target Users

1. University students and post JC/Poly/ITE (as tutors)

University students and those awaiting entry into university commonly rely on teaching part time tuition to earn some side income for their daily expenses or university education. This is due to the flexibility of teaching tuition which allows them to plan around their schedules. Furthermore, teaching tuition generally fetches a higher rate compared to other part time jobs and university students are generally qualified for the job since they have been through the same education system and learnt similar content. Therefore, Tutorific would be a helpful app for university students to find tutees more easily. They can also avoid the high commission fees charged by other tutee finding platforms if they use Tutorific.

We can share about Tutorific through the many telegram groups that university students are commonly in, such as hall residential groups, CCA groups or the informal groups for different modules. This allows us to reach a large proportion of university students. On the social media side, we can work with NUSSU to promote Tutorific on their Instagram page. Lastly, we can also distribute stickers at MRT stations, such as Kent Ridge and Pioneer MRT stations, and possibly parts of NUS with high student traffic, such as UTown. These stickers can feature a QR code of Tutorific as a point of access and are cheap to mass produce.

2. Pre-University students (Junior College and Upper Secondary students)

Some Junior College and Upper Secondary students may feel that they require additional academic support and guidance outside of school, especially with all the disruption to formal schooling that COVID has brought about. They may prefer private tuition with university students rather than tuition centers due to the one-to-one attention and more relevant tips they can receive, as well as the comfort from learning from a senior that is closer in age. Moreover, tuition from university students is generally cheaper than tuition from more qualified current/ex-MOE tutors. Tutorific precisely provides them with a targeted platform to quickly find university students to tutor them in specific subjects.

To target this group of students, we can give out free foolscap paper outside of school with our logo and a QR code to Tutorific on the cover page. This will be well received by students as foolscap paper is a staple for doing school work. Students could also easily whip out their phones to scan the QR code and check out Tutorific as a PWA. This is a feasible strategy since the cost of mass printing such foolscap paper is quite low.

3. Parents of Pre-University students (Lower Secondary and Primary students)

Tuition is heavily ingrained in Singapore's education system, especially for lower education levels like primary school. Many parents are very 'kiasu' and want their children to perform well academically early on to get into the best schools. For students in Lower Secondary or Primary, parents are usually the ones finding tutors for their children. Parents are generally more thorough and would like to know much more about the potential tutor and his/her qualifications. This can be easily facilitated via Tutorific's chat function.

To reach out to parents, we can promote Tutorific on forums that parents frequent, such as KiasuParents.com. This enables our app to spread through word-of-mouth as parents share this amazing app to their friends. Since parents are usually more familiar with Facebook, we can also launch a Facebook advertisement campaign for sponsored Tutorific posts to appear in their newsfeed.

General Marketing Strategy

Tutorific incorporates social media sharing to let tutors and tutees share listings that they have created with their followers and users are incentivised to do so in order to quickly get more eyes on their listing beyond the Tutorific platform. As more listings get shared on social media, Tutorific would gain more publicity and traction, encouraging more of our target users to come onboard.

Another way to encourage users to promote Tutorific is through a referral system. This has not been implemented but we envision existing users to be able to provide their friends with a referral code to sign up with. When friends use your referral code, the next listing you create will have a higher priority which allows it to appear higher up in the listing page to garner more views. This serves as an added incentive for users, be it tutors, tutees or parents, to share Tutorific with their friends.

Milestone 3

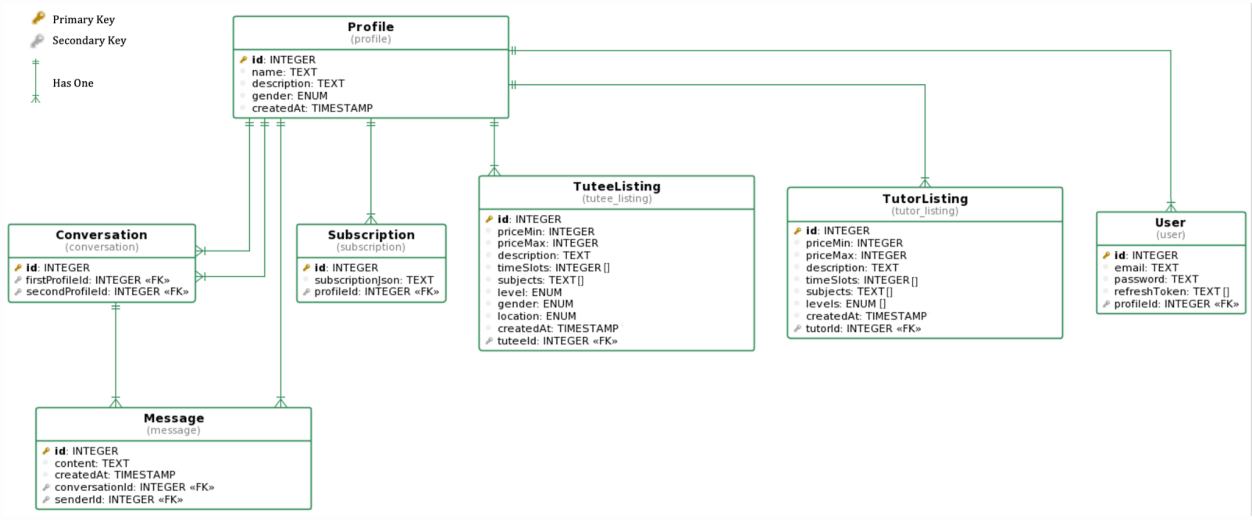
What is the primary key of the `home_faculties` *table?*

The primary key of the `home_faculties` table is faculty.

Draw an Entity-Relationship diagram for your database schema.

In the EER diagram below, the features shown are:

- Foreign keys (relationship)
- Columns, with type
- Primary key
- Table names



Milestone 4

Explore one alternative to REST API. Give a comparison of the chosen alternative against REST. Between REST and your chosen alternative, identify which might be more appropriate for the application you are building for this project. Explain your choice.

GraphQL is one popular alternative to REST API. GraphQL is a query language API whereas REST API is an architectural pattern for network-based software. The main issues that GraphQL is trying to solve are: **(1)** multiple roundtrips with REST API **(2)** over and under fetching with REST API.

Tutorific is a simple tutor and tutee listing app. We implement basic CRUD functionalities across entities like `User`, `Profile`, `TutorListing`, `TuteeListing`, `Conversation` and `Message`. Thus, we are looking for a simple and conventional way to implement client-server communication.

Pro of GraphQL	**Suitability/Necessity for our use case**
Do not need multiple round trips. In REST API, there is usually a need to make multiple API calls to fetch different types of entities. In contrast, for GraphQL, the client can just specify what it needs and get back different types of entities in a single API call. Thus, there is no need to make multiple requests in GraphQL.	There are hardly any instances where we need to make multiple API calls. Also, this problem can be solved by eager loading, which loads entities at foreign keys on GET requests. Thus we do not face the problem of making multiple API calls. Loading entities at foreign keys also does not cause over-fetching issues because most of the fields fetched will be displayed.
Client only receives what is requested, solving over and under fetching issues. In REST API, GET requests typically fetch whole entities. This is not required sometimes - for example, you only need the 'name' of the user entity. In contrast, in GraphQL, the client will specify exactly what it needs. Thus what is fetched is exactly what is needed, with no excess or dearth.	For our app, most of the data that we fetch is displayed, hence we do not suffer from over fetching issues. Moreover, the entities in our app are not very big (most have less than 10 fields). Under-fetching is solved by eager loading, as explained earlier.

GraphQL is a client-driven architecture, as compared to REST which is a server-driven architecture. This means that the client decides what he wants. Thus, there is high predictability in terms of the client knowing the format of the entities that he is getting.	Predictability in endpoints is not an important consideration, given the nature of our app. Tutor/tutee listing systems are isolated systems and do not need to serve other services. For now, we feel that REST API is sufficient for our needs. We feel that REST API is also predictable because the format of the entities returned are all standardized.
Allows for complex queries.	Our app is a simple content management system so we do not need complex queries.

Con of GraphQL	**Detriment to our use case, if any**
More complex to use and learn	Our team has minimal experience with GraphQL. Given the time and manpower we have, we felt that using GraphQL would be a risky decision. Our team has experience with REST API. We decided that using what we were comfortable with, and speed of development, is more important.
Smaller community than REST API.	This was the first time we were using a Koa backend with REST API. Thus, we felt that having ample documentation (e.g. StackOverflow) for a Koa backend with REST API was an important consideration for our app.
Lacks built-in caching abilities.	REST APIs leverage the built-in HTTP caching mechanisms to return cached responses. Since each GraphQL request is customised, it is harder to leverage built-in HTTP caching, which is essential for our offline functionalities.

Always returns HTTP status code 200. Difficult to report errors and caching	Our caching solution relies on inspecting the HTTP status code to determine what to cache, as it will only cache successful requests. Hence, GraphQL is not suitable for our application.

Given the pros and cons of GraphQL as compared to REST API, we have decided to use REST API. It is more appropriate for our application's needs. The problems that GraphQL is trying to solve, namely multiple API calls and under/overfetching, are not challenges that our app will face. Our features, mainly basic CRUD logic, does not necessitate the use of a complex query interface like GraphQL. In fact, we feel that GraphQL is likely to be more detrimental for our application, given our lack of experience with it. However, if given more time and more complex features, we are also open to the possibility of investigating the usage of GraphQL.

Milestone 5

Design and document all your REST API. The documentation should describe the requests in terms of the triplet mentioned above. Do provide us with an explanation of 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).

Our REST API is documented here: https://tutorific.docs.apiary.io.

Our API conforms to the REST principles. Three key REST principles are client server separation, stateless requests and uniform interface constraints.

Client-server separation involves a separation of user interface concerns from data storage concerns. This is required so that there is a clear separation of responsibilities between the client and the server. Our REST API is built on the basis that the client and server are isolated from one another. The server does not hold assumptions on the state of the client.

REST stands for representational state transfer. In REST API, requests should be stateless. Each request should contain all information necessary to understand the request. For our app, all session state is kept with the client instead of the server. Our app uses JWT token and such information is stored client side.

Uniform interface constraints is also an important REST principle to follow. The four constraints specified are **(1)** Identification of resources **(2)** Manipulation of resources through representations **(3)** Self-descriptive messages **(4)** Hypermedia as the engine of application state. First, resources should be identified in requests. This is done for our app. For example, the Tutor Listing resource is accessed through /tutor and the Tutee Listing resource is accessed through /tutee. Second, resources should be manipulated through representations. An example in our app would be that each user can modify his own Tutor Listing or Tutee Listing if he has the Listing id and the fields that he wants to update. Third, messages should describe all that is required to process the request (same as the Stateless explanation). Fourth, in hypermedia as the engine of application state, the client is provided with weblinks to access resources in the app. This is not relevant to our use case.

Phase 2: API Server

Milestone 6

Share with us some queries (at least 3) in your application that require database access. Provide the actual SQL queries you use. Explain what the query is supposed to be doing.

We are using TypeORM as the ORM to our PostgreSQL database. The following are 3 queries that are used in our application.

Create Profile

This is used to collect user details (name, description and gender) when the user logged in for the first time.

Endpoint:

```
POST /api/profile
```

ORM Query:

```
getRepository(Profile).save(${newProfile})
```

Underlying SQL Query:

```
INSERT INTO profile(name, description, gender)
VALUES (
    ${newProfile.name},
    ${newProfile.description},
    ${newProfile.gender}
);
```

Fetch Tutor Listings

This is used to fetch tutor listings that match specific criteria on the listing's fields (such as priceMin, priceMax, subjects, levels, etc). The queries differ depending on which criteria are specified through the query parameters passed to the endpoint so that each query is only as complex as is required, instead of having one extremely complex query that works for all possible query parameters. All queries are sorted by the listing's creation time, from the latest to the earliest, so that the newest listings would appear first. In the given example, the query fetches at most `${queries.limit}` tutor listings, excluding the first `${queries.skip}`, that have a priceMin of at least `${queries.priceMin}` and tutor of a specified gender.

Endpoint:

```
GET /api/tutor
```

ORM Query:

```
getRepository(TutorListing).findAndCount({
    relations: ['tutor'],
    where: {
        priceMin: MoreThanOrEqual(Number(${queries.priceMin})),
        tutor: {
            gender: ${queries.gender}
        }
    },
    skip: ${queries.skip},
    take: ${queries.limit},
    order: {
        createdAt: 'DESC'
    }
});
```

Underlying SQL Query:

```
SELECT *
    FROM tutor_listing
    LEFT JOIN profile ON profile.id = tutor_listing."tutorId"
    WHERE tutor_listing."priceMin" ⩾ ${queries.priceMin}
        AND profile.gender = ${queries.gender}
    ORDER BY tutor_listing."createdAt" DESC;
    OFFSET ${queries.skip}
    LIMIT ${queries.limit}
```

Fetch Conversations

This is used to fetch all conversations a user has with other users by finding conversations where the user is one of the two parties, as determined by the profileId.

Endpoint:

```
GET /api/conversation
```

ORM Query:

```
getRepository(Conversation).find({
    where: [
        { firstProfile: ${profileId} },
        { secondProfile: ${profileId} }
    ],
});
```

Underlying SQL Query:

```
SELECT *
    FROM conversation
    WHERE conversation.”firstProfile” = ${profileId}
        OR conversation.”secondProfile” = ${profileId};
```

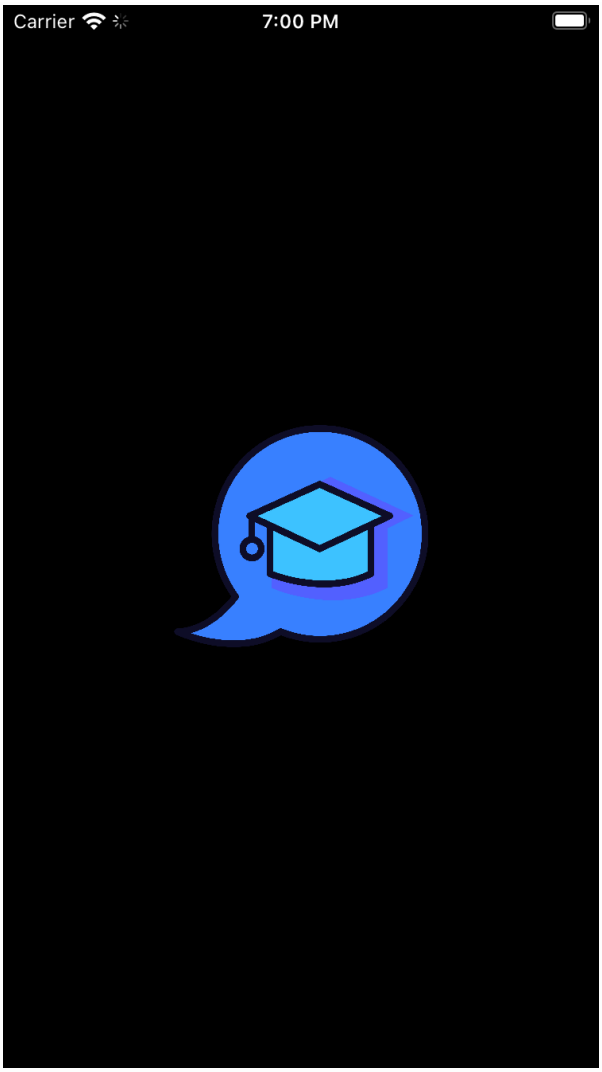
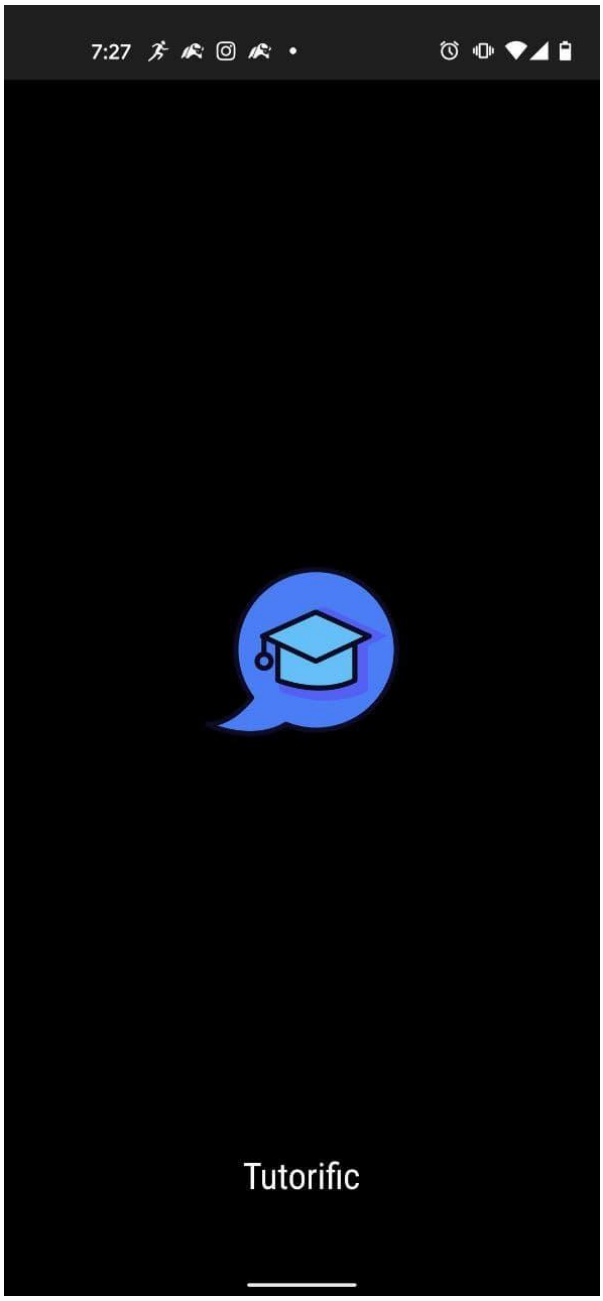
Phase 3: Mobile Client

Milestone 7

Create an attractive icon and splash screen for your application. Include an image of the icon and a screenshot of the splash screen in your write-up. 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.

Icon

Splash Screens

iOS	Android
	

Milestone 8

Style different UI components within your application using CSS in a structured way. Explain why your UI design is the best possible UI for your application. Choose one of the CSS methodologies and implement it in your application. Justify your choice of methodology.

UI Design

The goal of our UI design is to help users with the process of searching for a tutor or tutee. On the landing page of Tutorific, users immediately see a collection of tutor listings in the form of cards. These cards contain a summary of the listing's details, such as the asking price, subjects taught and level of education. The brevity of the card UI imparts the most important parts of the listing to users while simultaneously saving space, so that more listings can appear on the screen at once. If users are interested in a listing, they can tap on the card to view more details. In this detailed view, users can tap on the profile bar at the top to view more information about the person that created the listing, such as their bio and other listings they have put up.

We implemented infinite scroll for our Tutors and Tutees pages. This keeps users engaged while browsing for listings, as they can easily swipe down to see new instead of having to switch to new tabs or pages, which can feel cumbersome.

Users can filter listings by tapping the funnel icon on the top right. When there are existing filters, the funnel icon changes from an outline to a filled version, indicating to the user that they are seeing a subset of the available listings. The functions discussed thus far are made accessible to all users, including those that are not signed into an account. This eases the process of looking for a tutor or tutee and decreases the barrier to entry of using Tutorific.

Users that are logged in are able to create, update and delete listings. In addition, if they are interested in a particular listing, they can start a chat with the user that created it. The availability of this feature in Tutorific means that users do not have to risk revealing their contact information when they use other means to communicate with the tutor or tutee. This further streamlines the process of tutor/tutee searching, as almost everything can be done through Tutorific.

A notable UI component is the time slots segment used to convey information about a tutor or tutee's available times for a particular listing. This component allows users to have a good sensing of the tutor/tutee's availability at a glance. This component is also used as an input for users to add, edit or filter listings. It enables users to select and unselect time slots using

intuitive drag gestures. Since we expect users to mainly access Tutorific via their mobile devices, the time slot boxes were designed to be as large as possible while keeping the whole component compact enough to fit into the screen. Because of the deliberate design of the box size, users can accurately select time slots with their drag gestures using their finger or thumb.

CSS Methodology

We chose to adopt the Object Oriented CSS methodology and separated style attributes from the structure of the UI components in the application. By abstracting these style attributes into modules, this allowed us to reuse them across various components. For example, since we wanted both tutor and tutee listing cards to share a similar look, we abstracted their styles into a `ListingCard.module.scss` file and used them in both our tutor and tutee listing card components. This made engineering the front end more efficient as we could make changes to multiple components by updating a single file.

We also used the categorization rules recommended by the Scalable and Modular Architecture for CSS (SMACSS) Methodology for global styles. We organised our `.scss` files into separate folders for base, layout and theme rules. We chose to combine the module and style rules and stored these files in the same directory as the components using those styles for simplicity, as explained above.

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://`. List 3 best practices for adopting HTTPS for your application.

HTTPS is essential for deployment to guard against attacks. Our app is hosted on Heroku, particularly on the herokuapp.com domain, which helps us to enforce HTTPS automatically[1]. All requests to herokuapp.com are authenticated using DNSSEC, a security system that allows DNS servers to verify the reliability of the information they receive. As stated on the Heroku website, "SSL is always enabled for herokuapp.com for Common Runtime apps." The following are 3 best practices for adopting HTTPS.

Permanent Server-side Redirects

Users are automatically redirected from *http* (insecure) to *https* (secure) if they try to access http.

Enable HSTS

In HSTS (HTTP Strict Transport Security), the server requires the client to use a secure connection when communicating with it. Each HTTP response should have a header field "Strict-Transport-Security". With this header field, the "includeSubDomains" directive should also be included. The browser will force any subdomains to use HTTPS. HSTS also ensures that Google only indexes secured pages and only serves secure URLs in search results. [2] HSTS is enabled automatically and all HTTP responses include the header.

Proper Configuration of HTTPS

For example, use TLS 1.2 or 1.3. Most browsers do not support TLS 1.0 and 1.1 anymore.

[1] https://devcenter.heroku.com/articles/ss
[2] https://www.semrush.com/blog/http-to-https-best-practices-for-top-ranking/

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.

For our application, all API requests are cached. This means that any listings, chats or profiles that the user views while online will automatically be cached, and will be available to the user even when they are offline. We do this by first trying to request the latest data from the server, and falling back to the cache when the server cannot be reached. This ensures that the user always sees the latest data when online, but is still able to view any cached data when offline.

Our application also allows users to perform all the primary functions while offline. This includes adding, editing and deleting listings, sending new messages, and editing profiles. When the user reconnects to a network, these changes will be sent to the server by the app on behalf of the user.

We feel that these functionalities fit the users' expectations, as our application is not meant to be primarily used offline. As a tutor/tutee matching application, the core features of our application are most useful when online, as users will not be able to chat with potential tutors/tutees or view new listings while they're offline. Hence, our offline functionality is meant for temporary network instability, where the user has lost connectivity for a short period of time. This allows the user to have a more seamless experience while using our app.

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.

In session-based authentication, whenever a user logs in, the server will create a session for the user and provide a sessionId to the user in the form of a cookie. This sessionId is also stored in the database in order to map to the corresponding user. For subsequent requests to the server, the user would supply the cookie and the server would use the sessionId from the cookie to query the database to figure out the identity of the user in order to respond to the request accordingly.

For token-based authentication, when a user logs in, the server will encode certain user information in the form of a token signed using a secret key. This token is passed to the user who will attach it to subsequent requests. The server is able to check the integrity of the token using its secret key and decode the token to obtain the identity of the user in order to respond to the request accordingly. A popular form of this is JSON Web Token (JWT)

Token-based authentication is more scalable since the token is stored client-side in the browser without the need of any server-side storage while for session-based authentication the server needs to store the mapping between sessionId and user in a database. Thus, there may be increased stress on the server when there is a surge of logged in users, requiring frequent queries to the database to determine which user a sessionId belongs to.

Furthermore, session-based authentication is less versatile since it only works on a single domain or subdomains. This is due to the browser disabling cross-domain cookies by default. This poses an issue if APIs are served from a different domain. Tokens do not face this problem as they do not rely on cookies and are simply attached to the request headers.

However, token-based authentication can have higher data overhead since a token usually contains more user information compared to sessionId stored in a cookie. This can lead to longer loading speed if too much information is stored in the token resulting in longer processing speed for each request. Furthermore, JWTs are usually short-lived which require frequent reauthorization and thus a hassle for users.

We decided to use token-based authentication in the form of JWT as it gives us the flexibility of serving APIs from servers of different domains, which may be necessary when our user base

increases and we require multiple servers to handle the increased traffic. Koa also provides a middleware koa-jwt which makes it much simpler to implement JWT rather than having to manage additional database queries in sessions. The data overhead in our case is also negligible as the token only needs to store a small amount of user information, namely only userId and profileId. We are also able to circumvent the short-lived nature of JWTs by having longer-lived refresh tokens that can automatically be used to authenticate and create new JWTs upon expiry, thus, allowing for a more seamless user experience.

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 of the mobile site design principles and which pages/screens demonstrate them.

UI Design

We wanted the UI library to provide a native feel for the application, so that our frontend engineers could code once for both iOS and Android devices. We also wanted our UI library to provide as many components as possible so that we did not have to implement the whole thing ourselves. Keeping this in mind, we narrowed down the libraries into a few options and compared the availabilities of some of the components that we knew we wanted to use in Tutorific across the following libraries:

Components	Ionic	Framework7	Ratchet	Onsen
Inputs	✓	✓	✓	✓
Cards	✓	✓		✓
Infinite Scroll	✓	✓		
Pull to Refresh	✓	✓		
Routing with Animations	✓	✓		✓

This helped us further narrow down our options to either Ionic or Framework7. We did some further research on both libraries and discovered that Ionic provided a CLI to quickly bootstrap a Progressive Web App. This allowed us to quickly set up our application and begin developing. Furthermore, from what we gathered, Ionic seemed to be more popular among developers. This was a plus because this meant that more developers would have faced issues in the past and found solutions for them, which we can learn from if we stumble onto the same problems. Thus, we decided to go ahead with Ionic.

Mobile Design Principles

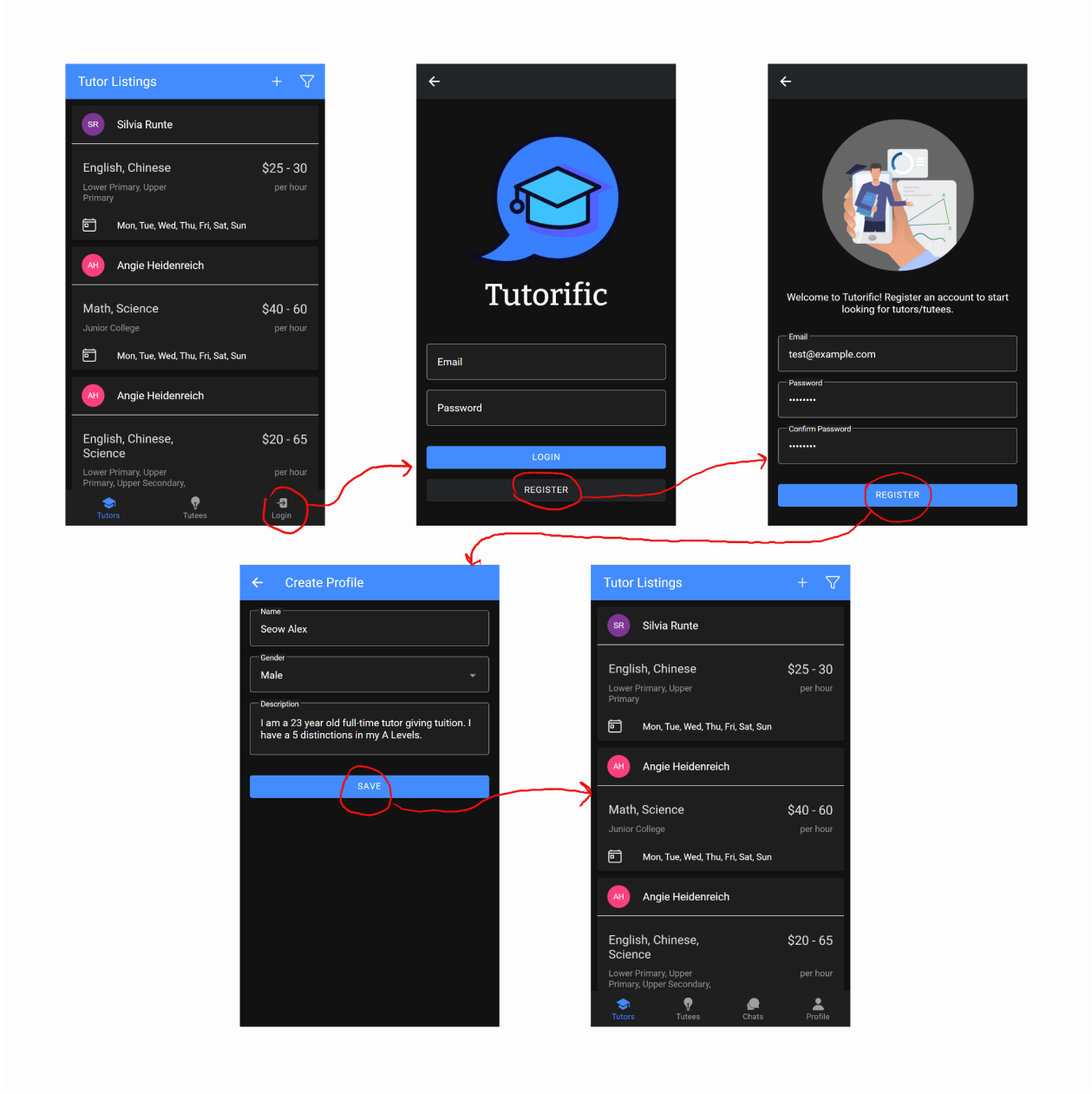
Principle	**Pages / Screens**
Filters to narrow results	● Filter tutor and tutee listings
Keep menus short and sweet	● Menu for listings that you own
Let users explore before they commit	● View listings, filter listings and view profiles are features that are accessible by unregistered users
Minimize form errors with labeling	● Create and filter listing forms
Optimize your entire site for mobile	● Entire app
Keep your user in a single browser window	● Entire app
Ability to go to previous page	● Detailed listing views ● Profile pages of other users
Drag gesture	● Time slot selection component in create, update and filter listing screens
Intuitive Navigation	● Bottom toolbar ● Tappable listing cards ● Clickable elements are generally in primary colour

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.

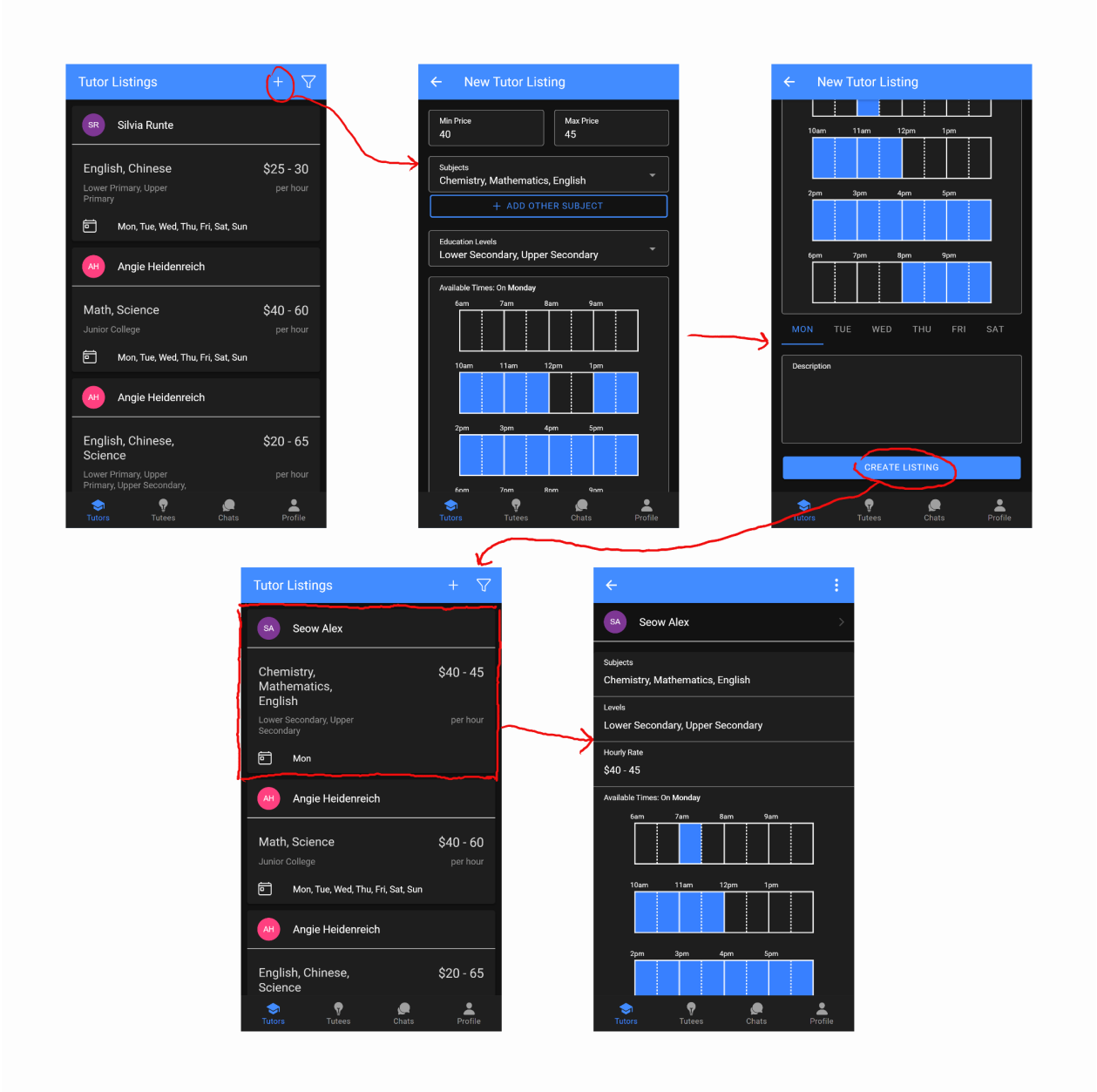
The 3 common workflows are registering an account, creating a tutor/tutee listing, and searching for tutor/tutee listings to start a conversation.

Registering an Account



While users can search for tutor/tutee listings while not logged in, functionality is limited as they are unable to add listings or chat with potential tutor/tutees. Hence, we allow users to quickly log in/register from anywhere in the app. If the user is registering a new account, they will be prompted to set up their profile, after which they will be able to create new tutor/tutee listings.

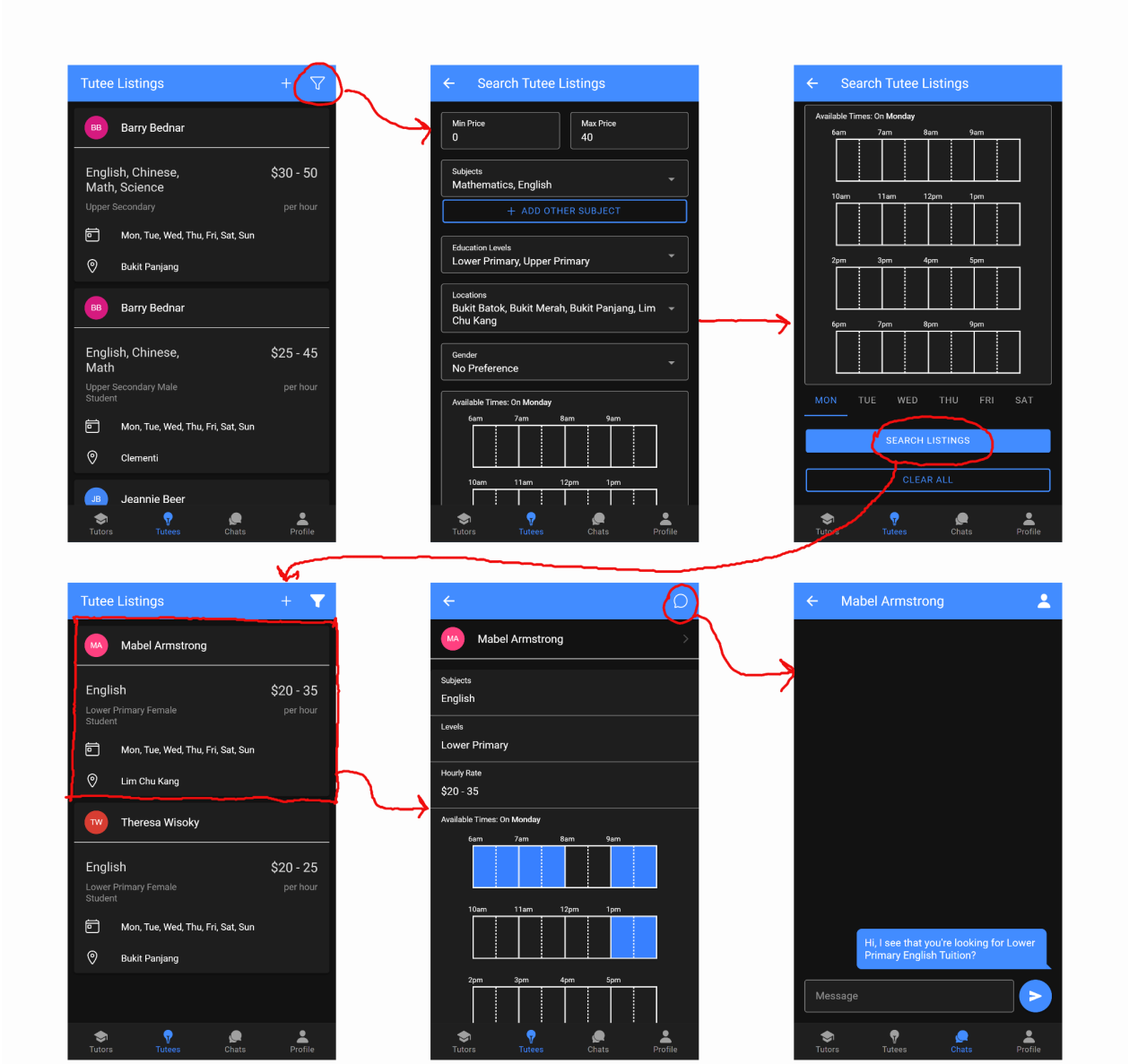
Creating a Listing



Another key workflow is creating a tutor/tutee listing. After being logged in, users will be redirected to the “Tutors” tab, where they can easily add a new listing using the toolbar buttons. Creating a new tutee listing is just as easy, as users can simply click on the “Tutees” tab

and click on the respective toolbar button. The toolbar buttons and tabs are visible at all times when scrolling through the listings for greater accessibility. Once a listing is added, it will be shown on the main tutor/tutee listings page so that they can verify that their listing has been added

Search for Listings and Starting a Conversation

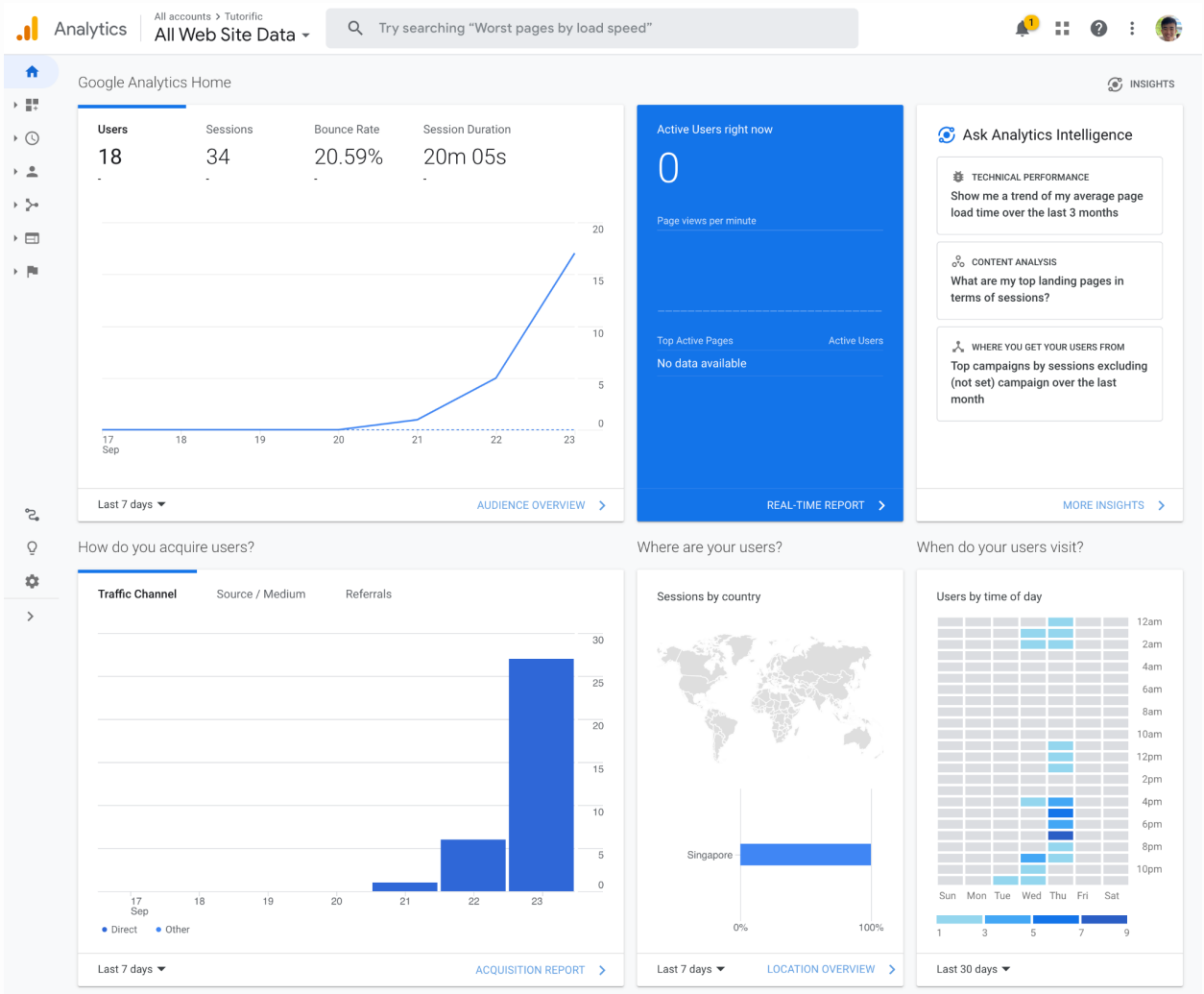


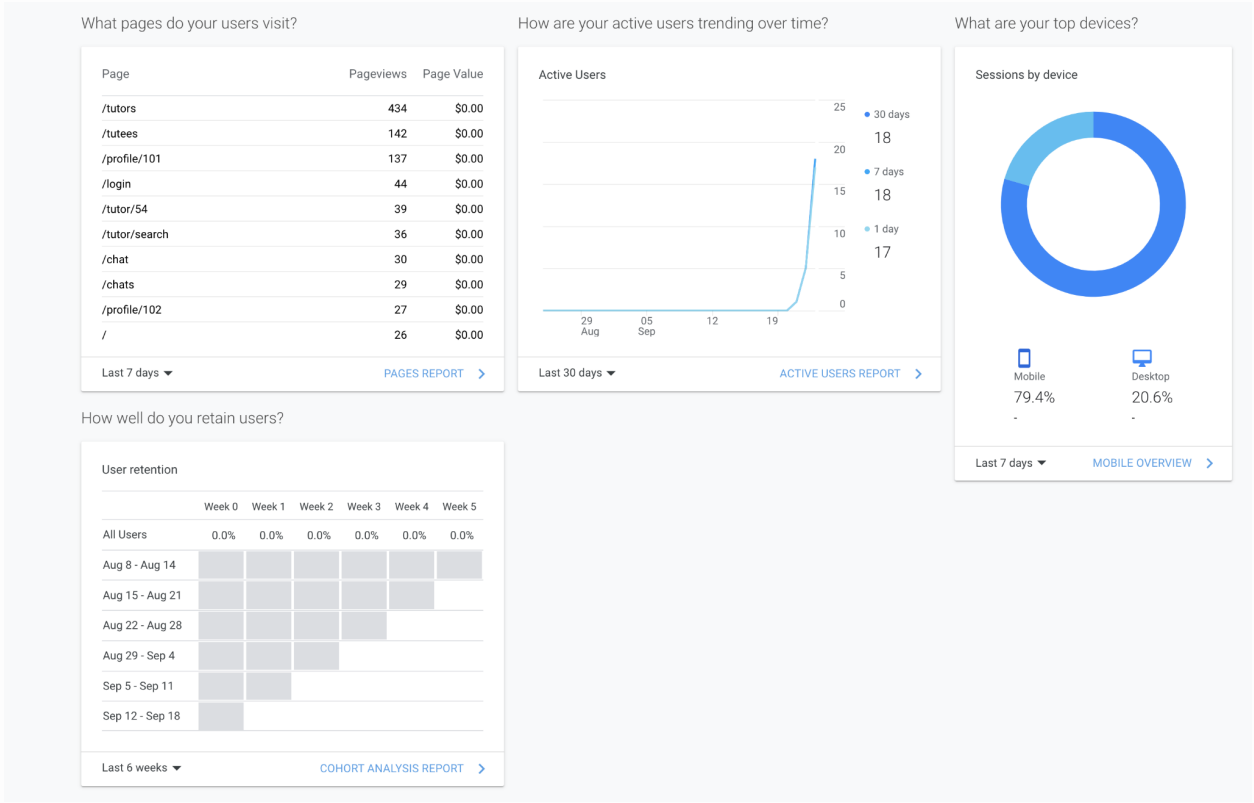
The last key workflow is searching for a tutor/tutee listing and starting a conversation. As the main use case for Tutorific is searching for tutors/tutees, this is the most important workflow, and thus we have tried to make it as seamless as possible. From the main tutors/tutees tab, users can click on any listing to view the details. If the user is interested, they can click on the

chat button to start a conversation with the corresponding tutor/tutee. Alternatively, if they want to view more details about the tutor/tutee, they can also tap on the profile header, where they can also start a conversation after viewing their details.

Milestone 14

Embed Google Analytics or equivalent alternatives in your application and give us a screenshot of the report.





Milestone 15

Achieve a score of at least 8/9 for the Progressive Web App category on mobile (automated checks only) and include the Lighthouse HTML report in your repository.

The report can be found in

`group-6-lighthouse.html`

Phase 4: Coolness Factor

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.

Tutorific leverages social networks by allowing tutors and parents to share their listings on social media sites and messaging apps. Specifically, users can share a link to their listing onto their Facebook wall or WhatsApp and Telegram chats.

Our target audience is composed of two groups, tutors and parents. Our research showed that WhatsApp is the most popular messaging app in Singapore, so we chose it as one of the implemented social plugins. We identified tutors to largely comprise post JC graduates, many of which would be university students. We knew that Telegram is almost ubiquitous among university students and that most of the existing messaging app channels discussed in Milestone 0 were based on Telegram. This also serves as evidence that many members of our target audience, both tutors and parents alike, are likely to be on Telegram. Hence, we decided to implement sharing to Telegram chats as well. Finally, Facebook is a social media site that is common among the older generation of Singaporeans. Since parents tend to be most familiar with Facebook compared to other mainstream social media sites, we chose to integrate Facebook as well.

We are aware that there are possibilities to integrate other social media sites as well, such as Twitter and Instagram. However, Facebook trumps Twitter by a large margin in terms of popularity in Singapore. As for Instagram, we felt that while it is used by many youths, it is far less prevalent among parents, so even if aspiring tutors shared their listings on their Instagram profile or story, it would not attract the attention of many parents. Hence, we decided to omit the integration of these sites from Tutorific.

Optional Milestone: Additional Features

Two additional features that our app includes are background notifications and real-time messaging. These features make extensive use of the Notifications API and Push API. The Notifications API allows our app to display notifications to the user, notifying them of new chat messages. However, this functionality would not be very useful if users could only receive notifications while the app is open. Hence, we have also integrated the Push API into our app, which gives our app the ability to receive messages pushed from our server, whether or not the web app is in the foreground. These two APIs allow us to display notifications to users whenever they receive new chat messages, allowing for more efficient communication between both parties.

The Push API also powers our real-time chat when the app is open. When a chat is open, users expect to see messages from the other party in real time. Traditionally, this requires polling, which means that the application would periodically make requests to the server so that it can display any new messages. However, this is not very tenable for our chat functionality, as we would need to make requests extremely frequently (in the order of milliseconds) to ensure that our chat feels responsive.

More recently, web apps are able to make use of WebSockets for full-duplex communication, allowing servers to push messages to clients without having to poll. However, this requires the server to establish a connection with each client, which takes a non-zero amount of resources. As the number of users grows, the server will need to make an increasing number of simultaneous connections, which may not scale well. Hence, we make use of the Push API, which allows the server to push messages to clients asynchronously through endpoints provided by web browser vendors (primarily Google and Mozilla). This reduces the load on our server, making it highly scalable to thousands of users.

One downside of using the Push API is that there is no support on Apple devices. This means that iOS users will not be able to receive chats in real-time. For these devices, we fall back to polling at regular intervals, which will feel less responsive. As the Push API is supported on ~74% of mobile devices, we feel that this is an acceptable tradeoff. However, given more time, our team could have investigated the use of WebSockets as the fallback mechanism instead.

Another downside is that the Notifications API and Push API are inextricably linked, and the ability to receive background messages and push messages asynchronously to clients requires users to grant notification permission to the app. If possible, we would have liked to be able to

push asynchronous messages to clients when the app is in the foreground, even if notification permissions are not granted, but that is an unfortunate limit of the specification

While we would have liked to be able to update in the background even without notification permissions, this is a limitation of the specification.