

# WeShare

Edmund Mok (A0093960X)

Luo Yuyang (A0147980U)

Le Trung Hieu (A0161308M)

Sreyans Sipani (A0177059U)

## **Milestone 0: Describe the problem that your application solves.**

Tired of arguing with your friends about who has to pay to whom and how much? Embarrassed while asking your friends to pay you back even though it's your money? WeShare allows you to track all your bills and payments so that you don't have to keep account of who owes you and who doesn't. It removes the hassle of splitting and assigning the debts to each other. All you have to do is upload your bill and friends who are splitting the bill and let WeShare do the rest.

## **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?**

WeShare allows users to keep track of their debts, payments and bills. Within a few taps, you can upload bills which have to be split among your friends and therefore help you remember your loans and debts. This application should be a mobile cloud application as it has the following features:

1. Interaction with other users:
  - a. Friend requests - Users can send friend requests to each other.
  - b. Bill splitting - The other users are also updated when one user adds a bill
  - c. Group creation - Users can form groups with their frequent friends to ease bill creation

2. Storing history - Users can refer back to the payments they made, bills they uploaded and therefore we need to store all the relevant user activities in a centralised database. All the data and calculations are displayed based on this centralised data and all the users are updated at once.
3. Portability - It is important that our application can be accessed from anywhere at any time. Therefore it is important that it be hosted as a cloud application so that it can be accessed from any device. This allows users to add friends, groups and bills on the go by simply accessing a URL.

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

We believe that our application is useful for all people but one can say it's more targeted to users who stay in a group (ex: housemates), hang out together (ex: university students or office colleagues), etc.

Without this application, traditionally people might make notes, keep bills or go as far as keeping an account book to keep track of their expenses. This might get too messy and confusing for them. Our application will make their life easier as the moment a bill comes, the person who paid can just add it on the application on the spot and the other users can pay him/her later. This way the user always has knowledge of how much money he must return back and how much he must get from his friends. Our application is attractive to them because of its simple and easy to use UI which makes it very intuitive for users.

Plan for promoting application:

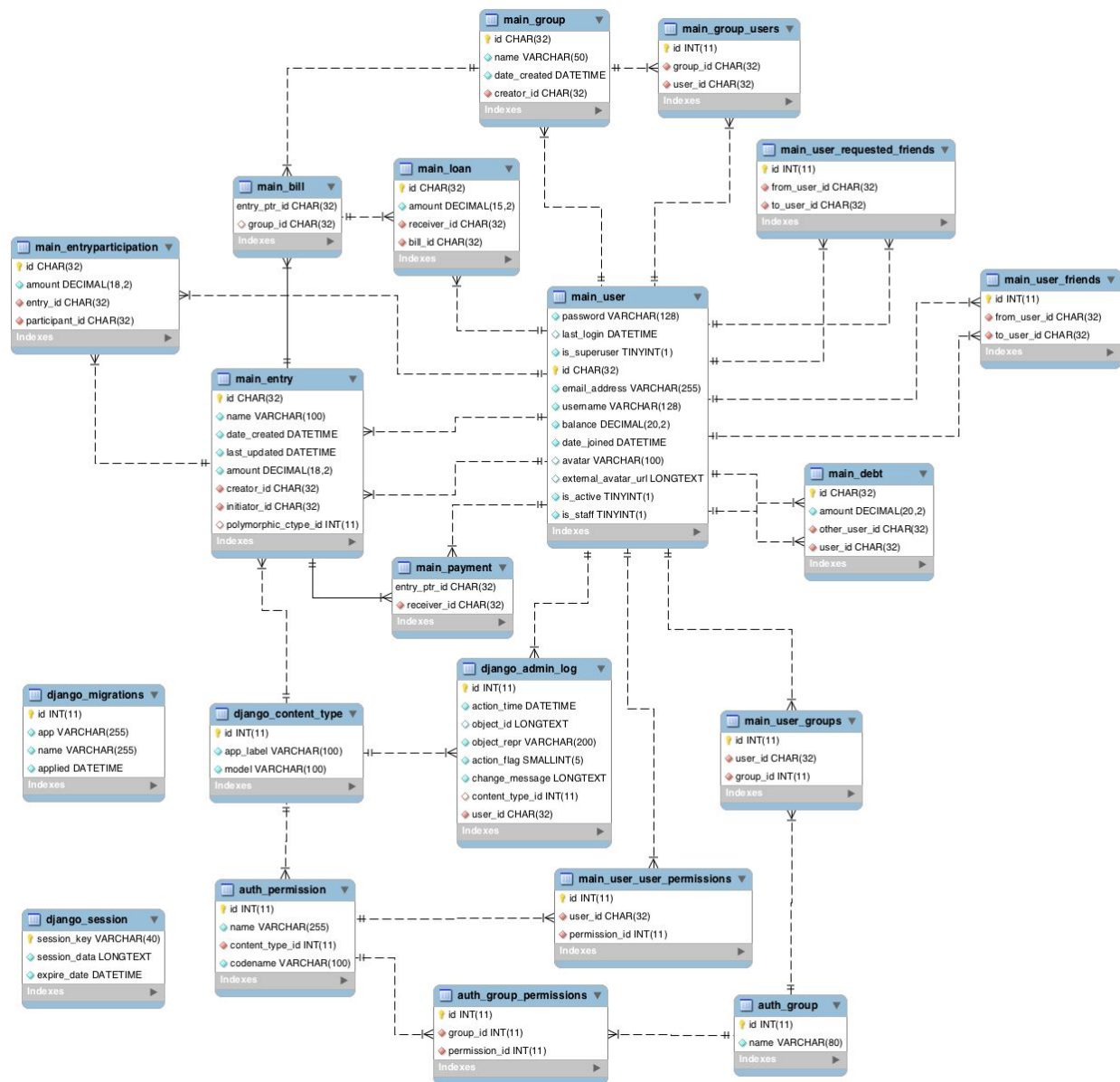
- Placing posters around university campuses like NUS, NTU
  - University students are likely to see these posters around their campus and try out the app

- Working with university camp organizers
  - Usually in camps, there will be a couple of social activities, like having supper as a group, that requires splitting of bills
  - Working directly with the camp organizers will let them know the existence of our application that helps to solve their pain point immediately, and also help us to gain new users

What is the primary key of the home\_faculties table?

Composite key of the 2 foreign keys - matric\_no in the student table and faculty ID in the faculty table

### Milestone 3: Entity-Relationship Diagram



- Bottom half is all Django auth related

## Milestone 4: REST API documentation

Apiary Link: <https://weshare2.docs.apiary.io/>

- Our API design mostly conforms to the standard REST API, except for the following details:
  - For any API involving the current user, e.g. get friends at /api/user/friends/, we do not conform to the pattern of {collection}/{item\_id}/{another\_collection}/ that REST APIs usually do.
    - Another way of saying it is we could have written the API route as /api/users/{user\_id}/friends/ in order to get all friends of the current user.
    - However, we decided against it since in our case, the current user can never get things like groups or friends of other users, so the user\_id is always the current user's id. In that case, it would be redundant to have the user\_id there and even worse, we need to check for access control (whether the specified user\_id is actually the current user's id).

## Milestone 5: Selected Queries

1. Get all friends of current user
  - a. ORM Query
    - `current_user.friends.all()`
  - b. SQL Query
    - **select** `weshare`.`main\_user`.`password` AS `password`,`weshare`.`main\_user`.`last\_login` AS `last\_login`,`weshare`.`main\_user`.`is\_superuser` AS `is\_superuser`,`weshare`.`main\_user`.`id` AS `id`,`weshare`.`main\_user`.`email\_address` AS `email\_address`,`weshare`.`main\_user`.`username` AS `username`,`weshare`.`main\_user`.`balance` AS `balance`,`weshare`.`main\_user`.`date\_joined` AS `date\_joined`,`weshare`.`main\_user`.`avatar` AS `avatar`,`weshare`.`main\_user`.`external\_avatar\_url` AS `external\_avatar\_url`,`weshare`.`main\_user`.`is\_active` AS `is\_active`,`weshare`.`main\_user`.`is\_staff` AS

```

`is_staff` from `weshare`.`main_user` join
`weshare`.`main_user_friends` where
((`weshare`.`main_user`.`id` =
`weshare`.`main_user_friends`.`to_user_id`) and
(`weshare`.`main_user_friends`.`from_user_id` =
'8cc3f7b847d04a6d91ab3ab96f514e26'))

```

c. Explanation

- `weshare`.`main\_user\_friends` stores many-to-many friendships using `from\_user\_id` and `to\_user\_id`.
- To get all friends of the current user (with id='8cc3f7b847d04a6d91ab3ab96f514e26'), we need to get all `to\_user\_id` in `main\_user\_friends` where the corresponding `from\_user\_id` matches the current user id. For each of these `to\_user\_id` (friend ids), we join back on `main\_user`, which is the users table, matching the `id` and get their information appropriately.

2. Get all groups the current user is in

a. ORM Query

- `current_user.joined_groups.all()`

b. SQL Query

```

select `weshare`.`main_group`.`id` AS
`id`,`weshare`.`main_group`.`name` AS
`name`,`weshare`.`main_group`.`date_created` AS
`date_created`,`weshare`.`main_group`.`creator_id` AS
`creator_id` from `weshare`.`main_group` join
`weshare`.`main_group_users` where
((`weshare`.`main_group_users`.`group_id` =
`weshare`.`main_group`.`id`) and
(`weshare`.`main_group_users`.`user_id` =
'8cc3f7b847d04a6d91ab3ab96f514e26'))"

```

c. Explanation

- `weshare`.`main\_group` stores information about all groups, `weshare`.`main\_group\_users` stores many-to-many group membership using the columns `group\_id` and `user\_id`.
- To get all groups of the current user (with id='8cc3f7b847d04a6d91ab3ab96f514e26'), we need to get all `group\_id` in `main\_group\_users` where the corresponding `user\_id` matches the current user id. With

these `group\_id`s, we can then join back to the `main\_group` table matching the `id`, and get group information appropriately.

3. Get all members of a group

a. ORM Query

- `group.users.all()`

b. SQL Query

- **select** `weshare`.`main\_user`.`password` AS `password`, `weshare`.`main\_user`.`last\_login` AS `last\_login`, `weshare`.`main\_user`.`is\_superuser` AS `is\_superuser`, `weshare`.`main\_user`.`id` AS `id`, `weshare`.`main\_user`.`email\_address` AS `email\_address`, `weshare`.`main\_user`.`username` AS `username`, `weshare`.`main\_user`.`balance` AS `balance`, `weshare`.`main\_user`.`date\_joined` AS `date\_joined`, `weshare`.`main\_user`.`avatar` AS `avatar`, `weshare`.`main\_user`.`external\_avatar\_url` AS `external\_avatar\_url`, `weshare`.`main\_user`.`is\_active` AS `is\_active`, `weshare`.`main\_user`.`is\_staff` AS `is\_staff` **from** `weshare`.`main\_user` **join** `weshare`.`main\_group\_users` **where** ((`weshare`.`main\_user`.`id` = `weshare`.`main\_group\_users`.`user\_id`) and (`weshare`.`main\_group\_users`.`group\_id` = '0b32055ab4544372b6a9e3c1157b974a'))"

c. Explanation

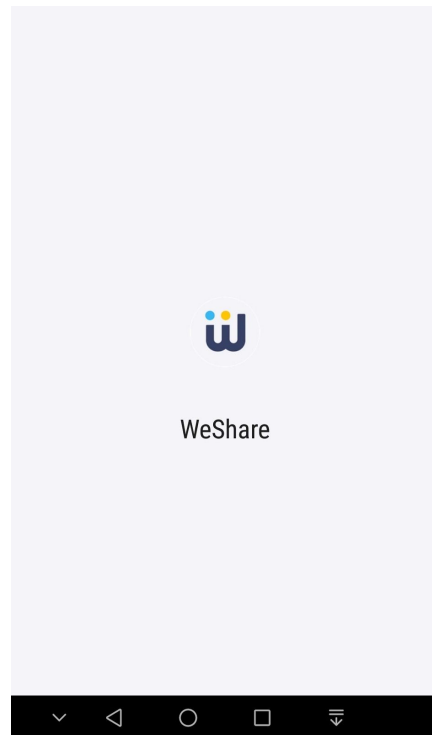
- `weshare`.`main\_user` stores information about all users, `weshare`.`main\_group\_users` stores many-to-many group membership using the columns `group\_id` and `user\_id`.
- To get all users of the given group (with id='0b32055ab4544372b6a9e3c1157b974a'), we need to get all `user\_id` where the corresponding `group\_id` matches the given group id. With these `user\_id`s, we can then join back to the `main\_user` table matching the `id` column, and get user information appropriately.

**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.**

Logo



Splash screen





**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.**

When it comes to splitting a bill, there are actually a lot of ways to do it and therefore our motivation for the UI is to make it as easy and simple as possible. Therefore we stick to displaying relevant information and try to limit user input as much as possible. We think it is much smoother for the user to be clicking and navigating through pages rather than typing values and therefore we try to provide most of the functionalities through pre-loaded dropdowns, multi-select checkbox lists and only ask for input when absolutely required.

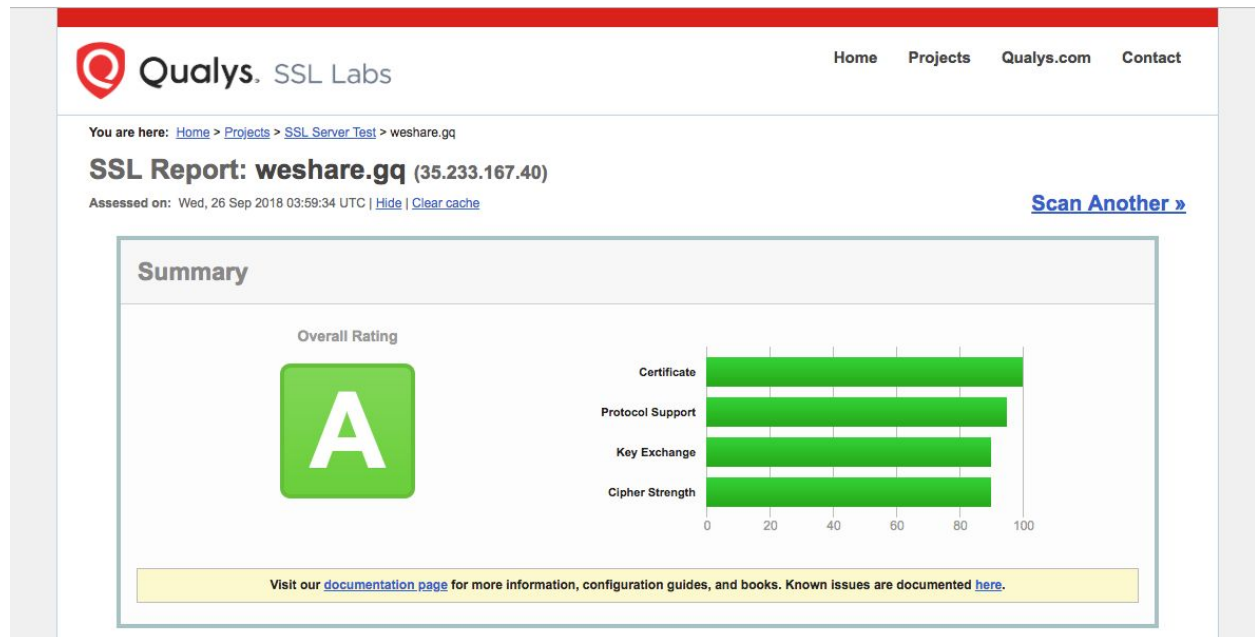
We use a few different types of styling methods in our project considering that we use React for our front-end:

1. Less modules: We have different modules hooked up with the different React components. These modules are respectively imported and applied in React with the help of class and id selectors. This way different components can have their own styles without any interference from other React components. This is because these names are scoped locally and are only valid in components that the modules are imported in.
2. Style object variable: We also make use of javascript objects to store our styles and then use the material UI library to use the styles with the component itself. <https://material-ui.com/customization/css-in-js/>
3. We also use inline styling for specific cases or overriding certain selectors in a few cases.

I would say our approach was closest to SMACSS but suited to React. I used certain guides while structuring the CSS for our codebase and I will link them here:

<https://blog.logrocket.com/the-best-styling-in-react-tutorial-youve-ever-seen-676f1284b945>  
<https://hackernoon.com/styling-react-with-css-9f6cef1823cc>

## Milestone 08: HTTPS



What are the 3 good practices that we adopted?

1. Use server-side 301 Redirect: Redirect our users and search engines to the HTTPS page or resource with server-side 301 HTTP redirects.
2. Support HSTS Preloading: Tells browser to request HTTPS automatically, even if the user enters HTTP instead. This helps minimize the risk of serving unsecured content
3. We choose Let's Encrypt as our CA due to mainly economical reasons. We also choose the most suitable type of certificate (single certificate for single secure origin).

### Explain “certificate pinning”:

Without certificate pinning, the usual way to verify certificate is through certification chain verification. For example, if your browser trusts A, then A signs B, and B signs C, then the browsers trust certificate C as well. However,

with certificate pinning, it will only trust certificate signed directly by A (which is B in this case).

**Pros:**

- + Eliminate some potential security issues, because without certificate pinning, if one of the certificate along the verification chain is compromised then everything after that will be compromised.

**Cons:**

- + Harder to implement + configuration issues.

We decide not to use the certificate pinning because of the overhead of the configurations. We aim to focus on more important tasks first.

**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.**

We allow users to view all the content which is loaded before. This is done using Service Worker and Cache Storage. When users are online, the cached response will be updated according to the response, which ensure that the data will not be outdated. When users are offline, the cached response will be served. For this feature, we do not use localStorage (or sessionStorage) because we think service worker with cache storage can serve the purpose well. Firstly, cache storage, same as localStorage, has no expiration set. Secondly, localStorage has a very limited size and type (String only) while cache storage does not. Thirdly, by using cache storage to handle fetch event, we do not need to modify our front end workflow, which mainly relies on the Ajax call with Axios. In the service worker, the online/offline is detected by

catching exception of fetch event rather than listening to online and offline event because the former is more reliable.

We are aware that allowing users to post new bills offline may be reasonable, but many features of our app rely on the calculation of bill and payment data. Due to such complexity and the time limit, we did not implement this offline feature. For now, we will display warning message to users when users attempt to create bill/payment offline.

## Milestone 10: Authentication

Django provides session-based authentication by default. In this model, the frontend will store the *sessionid* in Cookies and the backend will store the user state in a **cached** database (using memcached and postgresql), which can be accessed using the sessionid, as shown below:

Front end:

csrftoken	ypqvdBwq0CuyCHCNTv63KzLmCfWgBl7xvg3...	wesha...	/	2019-...	73
sessionid	cgsbr9eb789fopottoqg7o2ryc0onxyf	wesha...	/	2018-...	41

Back end:

Session key:	cgsbr9eb789fopottoqg7o2ryc0onxyf
Session data:	MjgwOTc2OTQ5MzZiZjJmMjFIZDU4ZTUxZjhjNzZmOTNhNTZmMmlwODp7II9hdXR0X3VzZXJfaWQiOiI3OTRkNDIyMS02YTlhLTQxYTYtOWRIYi0yNmVhNGNjYzM4ZTkilCJfYXV0aF91c2VyX2hhc2giOiI5Mml5MjAyNzZkZjdjZGQxMGNhYWVlY2MwNGVjODE2M2M3NDZlOGJlIiwiaXZlZGhfdXNlcl9iYWNRZW5kljoizGphbmdvL

One advantage is that user authentication is already handled out of the box for us. We acknowledge that it is definitely true that JWTs can be more scalable than sessions because with JWTs, the server does not need to handle queries and logics about sessions for each request, but given that our app does not have that many users now, we are fine with sessions for now and intend to switch to JWTs when we have more users. Another advantage is that using this built-in sessions allows us to work on more important user features instead of configuring an alternative authentication scheme.

**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.**

There are many frameworks, we use React with a combination of Antd design and Material-UI design, for the following reasons:

1. React is the foundation for React Native, so we can easily migrate to a native application if we want to in the future.
2. The component-centered approach of ReactJS is conducive to efficient software development. As most of us know javascript, it is easy to pick up and build components.
3. ReactJS supports both raw and JSX - XML-like syntax for writing JavaScript.
4. The UI consists of components. Those components can render in the browser, on the server, using Node.js / Django. In this way, the challenges of managing apps that need to be delivered to many different operating systems, browsers, and devices are resolved.

For our backend API, we decided to use Django for the following reasons:

1. Django's ORM: ORM is one of the notable features of Django. This takes care of databases. Django developers have a special way to manipulate the corresponding Python model object, unlike many other Python frameworks that directly deal with the database via SQL. Django by default works out-of-the-box with relational database management systems like PostgreSQL, MySQL, SQLite, and Oracle.
2. Security: Django is highly secured. The web framework comes with default protection against XSS attacks, CSRF attacks, SQL injections, which is an important concern for our application as it deals with payments and user bill information.
3. Easy Database Migrations: With Django's migrations, it is also very easy to change and track your database schema and its associated changes. Migration names help in managing version control, and a tons of options are available to merge versions and make modifications.

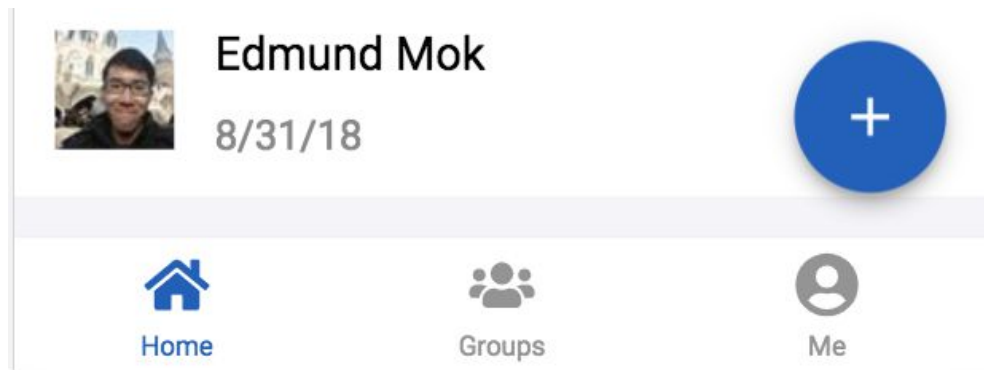
These features would help us to come up with a working product as fast as possible and therefore we chose these frameworks.

Design principles and corresponding examples:




*1. Make all of your users' most common tasks easily available.*

The most common tasks for Weshare is

- check one-to-one debt with friends
- creating new bills with a friend
- creating new bills with a group of friends



*Floating button to create a bill anytime which is the main function of our application.*

Friends owe you		total: \$5
	Maulik Jain 8/31/18	\$5 >
Friends you owe		total: \$10011
	royl8 8/31/18	\$21 >
	Varun Shaji 8/31/18	\$9990 >



*Relevant information about one-to-one debts displayed first*

*2. Design efficient form and minimize users' action as much as possible, Use existing information to maximize convenience*

We minimise the inputs in the forms by preloading information wherever possible. Examples include:

- Preloading the amounts when splitting equally
- Loading all group members when adding a bill to a group (by tapping the floating plus button in the group detail view)

talk about pressing plus button in friend detail view, group views will result in different preloaded splitters.



WeShare


Description

Dinner night










Pay by


\$ Total Amount \$ 180


Pay by  royl8

Split by

Split Equally
Split Unequally

	Sreyans Sipani	45.000	
	Trung Hiếu Lê	45.000	
	Edmund Mok	45.000	
	You	45.000	
	Add people		




SAVE

### 3. Make it easy to get back to home page

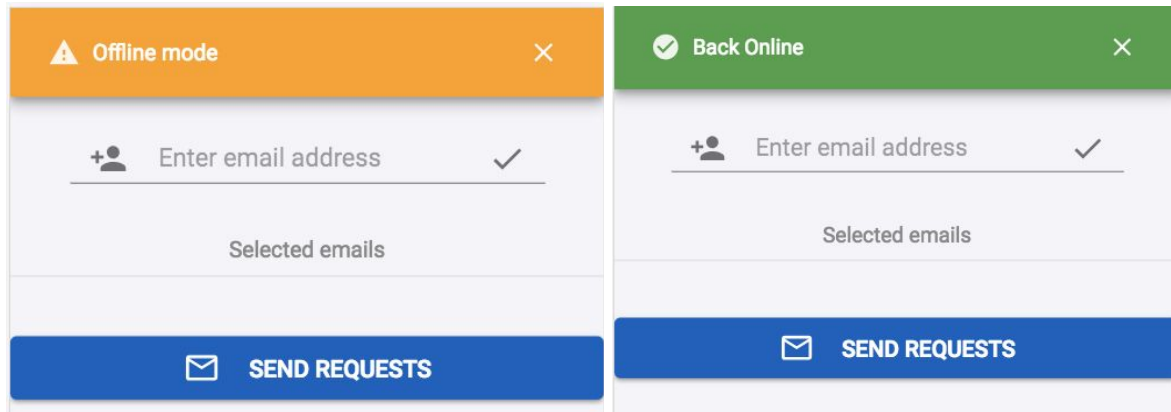
Every page has a top bar which allows users to easily navigate to the homepage or a dropdown which includes basic functionality such as add friend, create group and logout.





#### 4. Immediate feedback

The user is given immediate feedback using alert messages after sending requests, creating bills, going offline, etc.



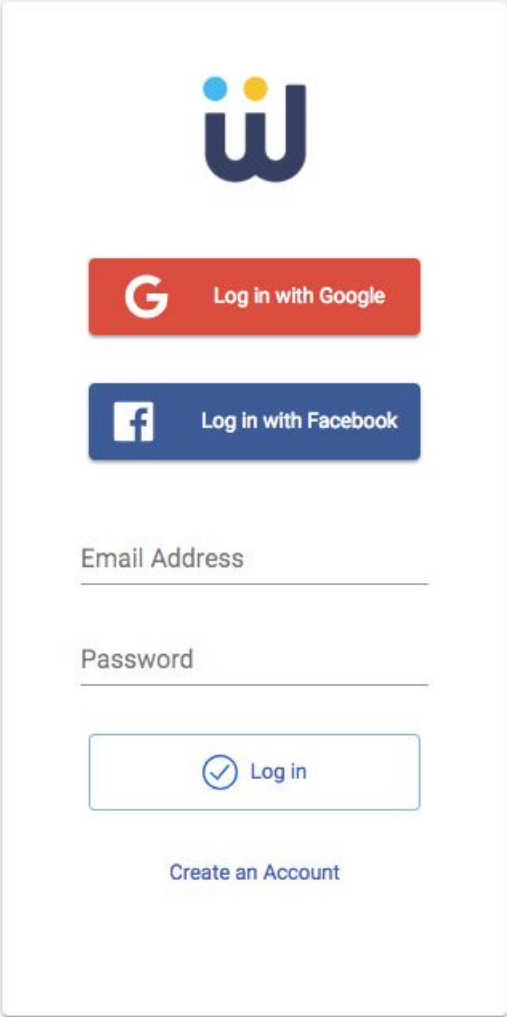
Notifications for going offline and online



After sending friend requests

#### 5. Social Login

We allow users to start using the app immediately without an account by using facebook or google. This enables user to quickly have a taste of the application before signing up.





The image shows a login form for a website. At the top is a logo consisting of a stylized 'w' in dark blue, with a small blue dot above the first vertical stroke and a small yellow dot above the second. Below the logo are two social login buttons: a red button with a white 'G' icon and the text 'Log in with Google', and a dark blue button with a white 'f' icon and the text 'Log in with Facebook'. Below these are two text input fields, one labeled 'Email Address' and the other 'Password'. At the bottom of the form is a light blue button with a checkmark icon and the text 'Log in'. Below the button is a link that says 'Create an Account'.

*Login through facebook and google*


## *6. Keep your user in a single browser window*

We never redirect our user to a new tab or window and make all necessary changes either using redirection or dynamic javascript.

 WeShare 

Friends owe you

total: \$21



Sreyans Sipani


8/31/18

\$21

>

Friends you owe

total: \$878




Trung Hiếu Lê

8/31/18

\$0.5

>




Jason Van

8/31/18

\$877.5

>

Friends settled up





Edmund Mok


8/31/18


\$0


>



 ADD FRIENDS




 Home


 Groups

 Me

 WeShare 



Trung Hiếu Lê

 Settle Up


Summary

\$ 0.50

Email address


lth08091998@gmail.com

History with Trung Hiếu Lê

 Eat after CS3216 class


9/27/2018


\$7.50 >

 Buy supper

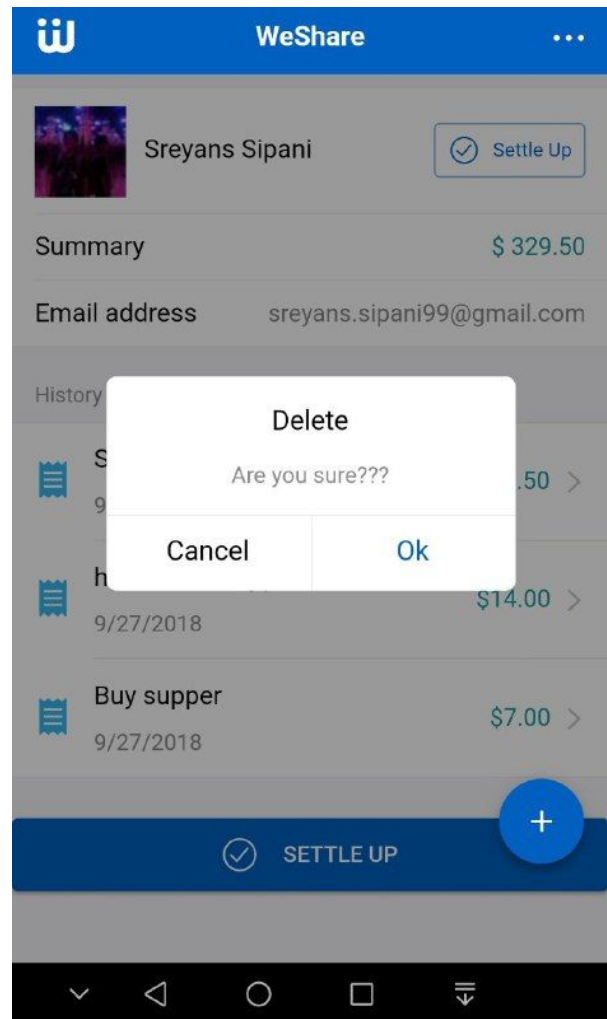
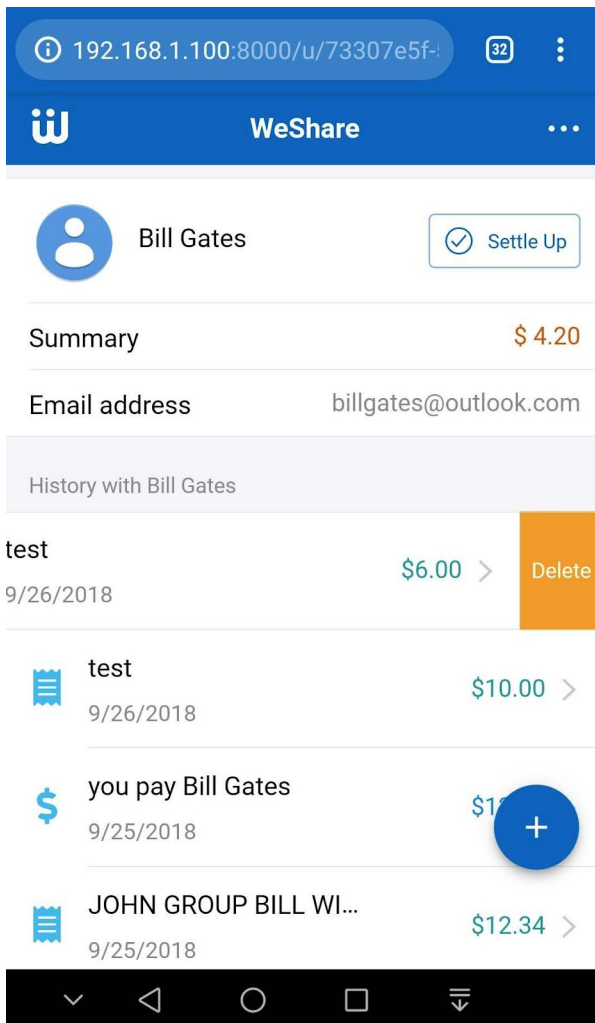
9/27/2018

\$7.00 >

 SETTLE UP



## 6. Confirmation for delete action



## Milestone 12: User Experience

1. Create a bill: In the home screen, there will be a floating button “+” where it is intuitive for the users to know that it is the creating-bill button. The user will input the name of the bill, adding the users involved in this bill, and add the amount of money for each person.

- + Inside the form to create bill, we allow users to either split equally (the amount is automatically calculated for each person) or unequally (the amount is manually inputted). Our friends suggest that it saves them the hassle to calculate money given that they often split equally.

- + Another good feature of the bill create form is that the current user is always pre-loaded into the list of users involved in this bill. It once again saves users from the hassle of calculating money because the current user does not need to subtract his own amount from the total bill.

2. Create a group: A same group of people may share bills many times together (Imagine that they are co-living in an apartment for example). So, we allow users to create group and add others to the group. Then, they can create a bill within a group, which loads all the group members automatically to the bill (save a lot of time). Moreover, it also helps users to query all bills related to a specific group later, which is very intuitive for them. However, the user cannot make a payment within a group, because the user experience will mess up. So basically a group is used for adding bill faster and organizing bill history more easily.

3. Friend Request: This is one of our easier to spot workflows in the app. There are two ways to reach the add friend page. One is by clicking a button (Add friends) on the home tab and the other is through the drop-down on the right. Once they land on this page, they are allowed to add multiple email addresses and then click on the send requests button once to send friend requests to all the selected emails/users. Once can see the status of their friend requests in the "Me" tab where they can accept/reject pending requests and also see their sent requests. As the application's functionality depends on the user's friends, this workflow must be simple, fast and as intuitive as possible.

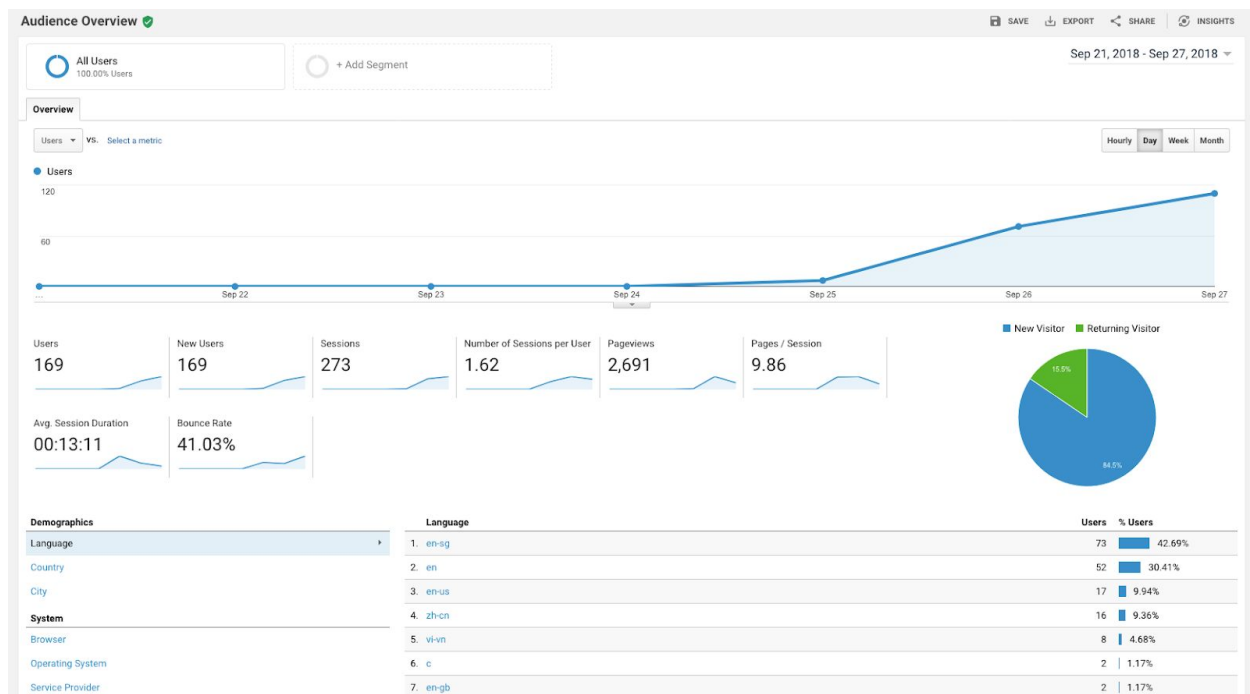
#### 4. Settle up

Once a bill is added by someone, the individual debts are updated for user to user. Between any two persons, their debts between each other are aggregated across different groups and this means that a user always owes another user directly. This also means that a user just has to pay his or her friend directly to settle any existing debts across all groups. In order to settle up, the user just needs to click on a friend's profile in the friend list, and select the "Settle Up" button that appears on the right of the profile. A popup will appear requesting for payment amount, and once the amount is input and confirmed, the debt between the two users will be updated by that amount. We chose to aggregate debts between two users across all groups because it would be easier for the user to settle any payments with another friend. An

alternative would be having separate debts for any two users within each group, which has a possible advantage of clearly showing how much one owes another person in a specific group, but also a problem if there are many groups and one has to "settle up" with another person within each group multiple times. In the end, we chose the simpler solution because it would be less confusing and more intuitive for the user.

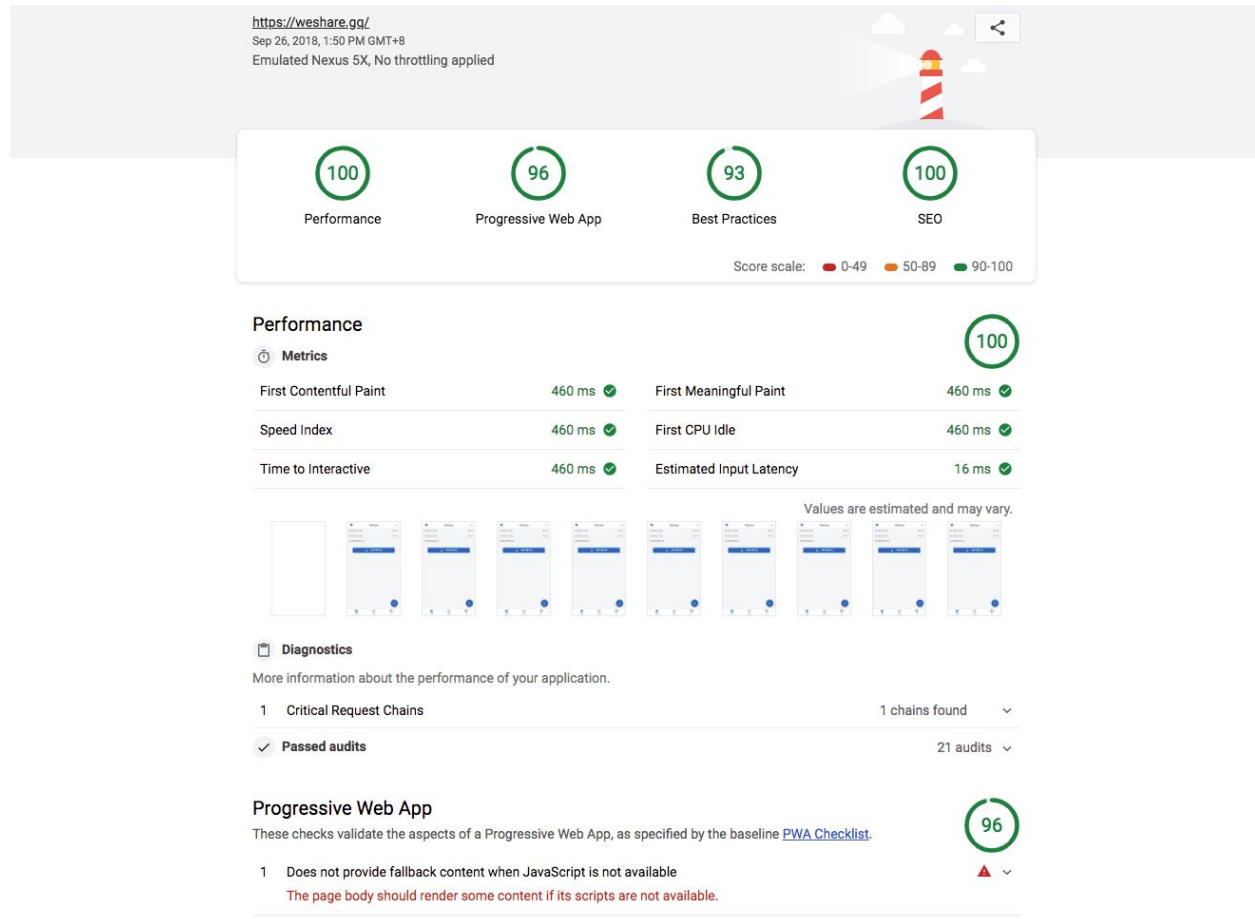
## Milestone 13: Google Analytics Embed

Till 28 Sept, 11:11 AM



## Milestone 14: Include Lighthouse html report in our repository.

The following is a screenshot, check the repo for the corresponding HTML file.



## Milestone 15: Social Network Plugins

We have integrated easy user sign-ins using OAuth APIs from both Facebook and Google. Facebook is the largest social network and almost everyone is likely to have a Facebook account, so we chose to integrate Facebook accounts as a login mechanism. Google accounts are also quite common, especially for Android users, so we also chose to add a Google sign-in feature for our application.