Write-ups for Group 1: Quizard

Check out our app at https://cs3216-a3-quizard.web.app/

Team Members

- Alistair Ong
- Caryn Heng
- Herbert Ilhan Tanujaya
- Nguyen Thanh Son

Milestone 0

Describe the problem that your application solves. (Not graded)

We are promoting the sharing of knowledge through quizzes! People can make and solve quizzes on any topic.

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?

Our application allows people to create and solve quizzes, and challenge their friends to see who has the best domain / general knowledge. With mobile cloud computing, we will be able to store the quizzes, leaderboards, and user information in the cloud for ease of retrieval anytime, anywhere with a simple login. It is also easy to use on the go, while waiting, or just to kill time.

Milestone 2

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

There are many people who can benefit from our application. For example:

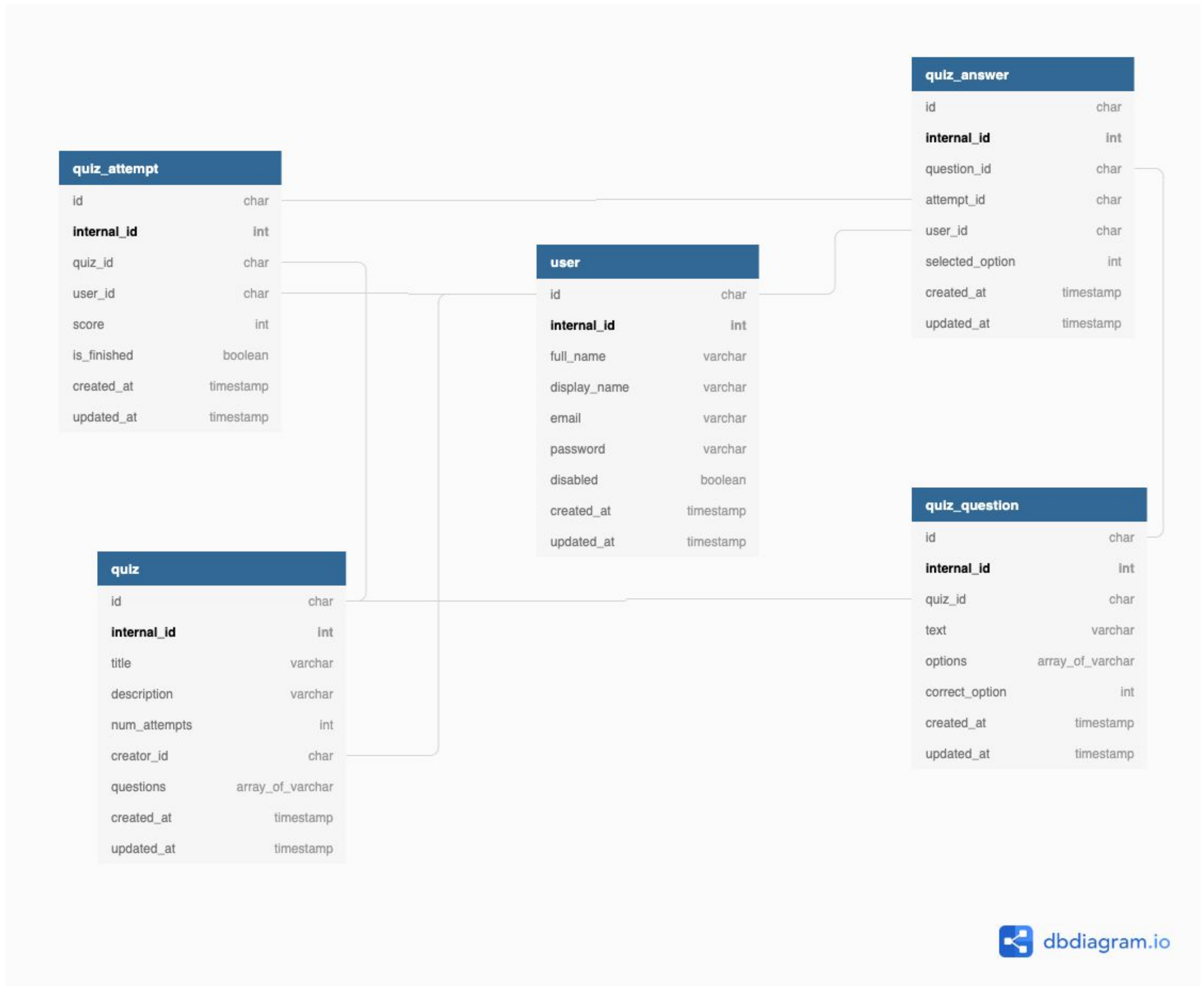
- Casual users who wish to challenge their friends. Maybe Caryn can make a quiz called "Do you know Caryn well?" that can be shared to her friends.
- Primary/secondary/JC teachers to create quizzes for their students that like pictures of cute owls.
- Communities of trivia junkies to share knowledge and test each other.

For these different sets of users, we can do different things to promote to them.

- For casual users, it is best to let people create quizzes. By allowing them to share quizzes easily via social media such as WhatsApp, Telegram, and Facebook, we can let them spread the app by word of mouth.
- For school teachers, we can visit schools and build partnerships with them.
- For trivia junkies, we can partner with trivia night events.

Milestone 3

Draw an Entity-Relationship diagram for your database schema.



Milestone 4

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

The documentation can be found at https://github.com/cs3216/a3-mobile-cloud-group-01/blob/master/api/README.md#endpoints.

Our API conforms to REST principles:

- URLs are in the form of e.g. `/quizzes`, `/quizzes/<uuid>`
- We utilise HTTP verbs to make intent clear, e.g. `GET /quizzes` to get all quizzes; `GET /quizzes/<uuid>` to get a quiz according to some UUID; `POST /quizzes` to create a quiz; and so on.
- HTTP status codes semantically describe the result of the request. This is documented at https://github.com/cs3216/a3-mobile-cloud-group-01/blob/master/api/README.md#common-http-codes-of-responses.
- We use JSON to communicate with the API.

Milestone 5

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

1. **Get the latest attempt for a quiz of a user**
   When a user opens a quiz, the client will send a request to check if there is any unfinished attempt of this user, or to display the score of the latest attempt.

   This SQL query is executed to retrieve the latest attempt for a quiz of a user.

```
SELECT
        quiz_attempt.id,
        quiz_attempt.internal_id,
        quiz_attempt.quiz_id,
        quiz_attempt.user_id,
        quiz_attempt.score,
        quiz_attempt.is_finished,
        quiz_attempt.created_at,
        quiz_attempt.updated_at
FROM quiz_attempt
WHERE
        quiz_attempt.quiz_id = $1
        AND quiz_attempt.user_id = $2
ORDER BY quiz_attempt.internal_id DESC
```

```
LIMIT 1
```

2. **Get quizzes**
   All endpoints returning multiple rows of any resources support pagination.
   Using keyset pagination on column *internal_id* (containing an incremental integer for each row in the table, starting from 1), the SQL query only returns the next rows after the last row of the previous returned response.
   ($1 defaults to 0 and $2 defaults to 15 rows per response)

```
SELECT
        quiz.id,
        quiz.internal_id,
        quiz.title,
        quiz.description,
        quiz.num_attempts,
        quiz.creator_id,
        quiz.questions,
        quiz.created_at,
        quiz.updated_at
FROM quiz
WHERE quiz.internal_id > $1
ORDER BY quiz.internal_id
LIMIT $2
```

3. **Logging in**
   A user must submit an email and password in order to login. The password will be hashed before being used to query against the DB. The application will try to retrieve a user using the email and password.

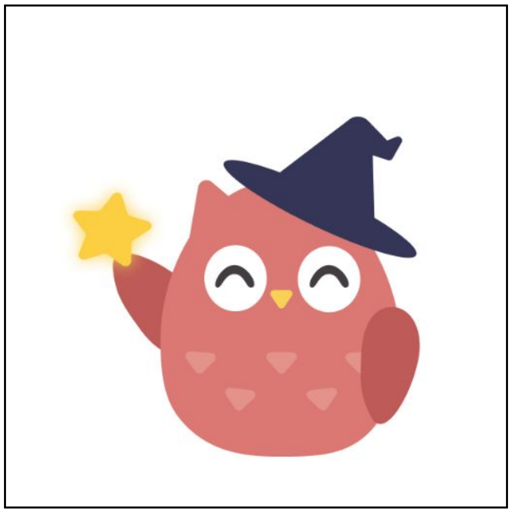
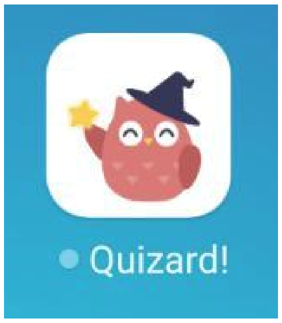
```
SELECT
        "user".id,
        "user".internal_id,
        "user".full_name,
        "user".display_name,
        "user".email,
        "user".password,
        "user".disabled,
        "user".created_at,
        "user".updated_at
FROM "user"
WHERE
        "user".email = $1
        AND "user".password = $2
```

LIMIT 1

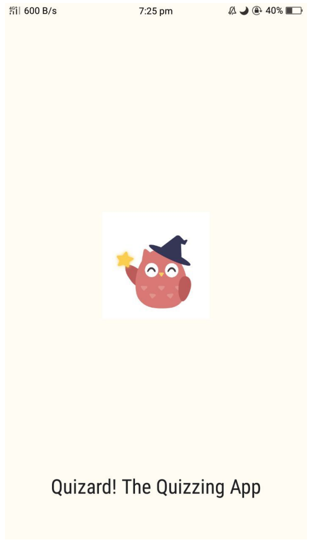
Milestone 6

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.

Icon

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.

We used styled-components to style our components. It works like Scalable and Modular Architecture for CSS. styled-components generates an actual stylesheet with classes, and attaches those classes to the DOM nodes of styled components via the className prop. It injects the generated stylesheet at the end of the head of the document during runtime. It allows the creation of components (divs, bodies, forms, inputs, buttons, etc) and adds styles to each and every one with the same tag in the end (styled-components does the tagging and injection for us).

This is one of the easiest and fastest ways to make scalable, reusable, modular components with style, without having to manually style them for every single page.

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. List 3 best practices for adopting HTTPS for your application. Explain the term "certificate pinning" and discuss the pros and cons of adopting it, as well as justify your choice whether or not to use it in your app.

Best practices

- For front-end, we used Firebase Hosting and they implemented HTTPS for us. Hence, we do not need to care about such infrastructure. We trust Google. Life is good.
- For back-end, we used Let's Encrypt, managed by Certbot. Typically, we need to renew the certificates so that they do not expire. Fortunately, Certbot has automatically installed cron jobs so that we do not need to care about it anymore. We trust EFF. Life is good.
- In our API setup, we are using nginx as a reverse proxy. Typically, we need to set some parameters in nginx, such as the encryption algorithms accepted. Fortunately, Certbot automatically configures our nginx files so that we do not need to care about it anymore. We trust EFF. Life is good.

By trusting them to do the best practices for us, we have achieved A+ and A on SSL tests in both our front-end and back-end respectively:

- https://www.ssllabs.com/ssltest/analyze.html?d=cs3216-a3-quizard.web.app&hideResults=on

- https://www.ssllabs.com/ssltest/analyze.html?d=quizard.xyz&latest

Certificate Pinning

Typically, when a browser issues a HTTPS request to a certain webpage, it will validate the certificate hierarchy. What this means is that, if the website's certificate is signed by A, which is signed by B, which is signed by C, and so on, until it is signed by a root certificate, the browser checks if the root certificate is something the browser wishes to trust.

What certificate pinning does is basically telling the browser to just ignore all of that, and only ask the browser to accept certificates signed by certain entities. For example, if I establish a connection with Google, Google might tell the browser: for the next 30 days, only trust certificates that are signed by Google. In the future, if the browser connects to Google and it is presented by a certificate not signed by Google, the browser will reject it.

Pros:

- If the certificate gets compromised later, the browser will reject it

Cons:

- If somehow the certificate needs to be changed (e.g. the certificate is lost / some malicious actor gets access to the private key), needs to wait until the pin age expires

We decided not to implement certificate pinning. It is another hassle to implement, without clear gains in our threat model -- if our certificate somehow needs to be changed, it is more likely that it is because we are still configuring our HTTPS, and not because some actor hijacks our certificate.

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.

Network falling back to the cache
For our offline functionality, we were trying to store the quizzes and their accessed data into the service worker cache so that if we fetch them again, if we are offline, then we will fallback to the cache, to take the last retrieved data from the cache. This makes it such that we can always keep our cache store up to date and always try network first especially since there may have been updates on the quizzes and their data.

In this way, users can check their quiz scores and review their quiz results that they have done even if they are offline but accessed their quiz results before. Also, they can see which quizzes they have created, finish attempting, and are still doing when offline so that they can possibly share it with other people as well, or think about it even when there is no network connection.

We also cache all the CSS, image, svg files so that we do not have to fetch them again when accessing the same website from the same browser. This makes it seem as though our website still runs when offline, and makes subsequent load times much faster.

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.

Authentication Type	Advantages	Disadvantages
Token (JWT)	- Scales well as the token is stored on the client side. - Works well even with servers on different domains. - As the user's information is stored in the token, there's no need to query for it.	- As the data in the token can be exposed, sensitive information should never be stored inside the token. - Without care, the size of the token can impact the requests' network latency. - If a user's information is updated, the token needs to be updated as well.
Session	- As only the session ID is stored in the request, it is very lightweight.	- Doesn't scale well, as the sessions are stored in the server side (memory or DB). - Have issues with cookie management if the servers are in different domains. - Require DB round trip to retrieve the user's information from the session.

The reasons why we choose JWT token-based authentication against session-based one are:
- It is very easy to scale horizontally (as the token is stored on the client side).
- Tokens are easier to maintain and debug.
- Tokens have built-in expiration functionality.
- JWT tokens are self-contained.
- No need for maintaining session management.

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.

Front-end
We decided to go with React and Redux for the front-end stack as it is something we were familiar with and is easy to set up, use and modify for our own needs and usage. As our app is quite a small scale project, it isn't a good idea to use a heavyweight framework like Angular. Vue could have been good for us as well, but we decided to go with React just based on the package available, the time constraints, and the general ease of using React over Vue. Vanilla HTML, CSS, JS would have been a detriment as we would not be able to utilize the optimizations, and functionality in terms of component building that React provides.

Redux is used as the application state management library because of how easy it is to use, visualize, and just aggregate all currently stored data into a single place to be used and modified, without having to constantly pass the state down from component to component - components can just be hooked up to the redux store where everything is stored. This makes it less messy and easier to debug as well.

Back-end
We use Sanic (https://sanic.readthedocs.io/en/latest/) for our web framework.

- Sanic is the first Async web framework to be stable in Python (for over a year), its community is very active and well-supported.
- Sanic is fast by utilizing *uvloop* - an event loop that claims to be "*at least 2x faster than nodejs, gevent*" and "*is close to that of Go programs*".
- Being asynchronous, Sanic is a lot more efficient to be used for I/O-bound applications.
- Unlike *Django*, Sanic is a micro-framework (just like *Flask*). Sanic aspires to be simple, make getting started quick and easy, with the ability to scale up to complex applications.
- It offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developers to choose the tools and libraries they want to use.

Mobile site design principles

1. Single column site layout throughout the entire web app for small screens - just like how most native mobile apps are designed.
2. Users don't ever have to pinch to zoom throughout the entire web app when on mobile. All font sizes are clearly visible and our pages are responsive and does not try to squeeze too much information into the tiny width of the mobile device.
3. Use only high quality assets. Our brand assets are all svgs which look good in all screen sizes.

4. Finger-friendly touch targets. All our buttons are huge enough even when tested on mobile. Ensuring this prevents users from missing the button that they indent to press. There is also sufficient padding between buttons (at least 10px of spacing) to prevent users from clicking the wrong button.
5. No complicated navbars so that users can easily navigate around the screens.

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.

Creating a quiz

Upon logging in and going into the homepage, the user can easily create a new quiz by clicking the huge “Create New Quiz” button. This leads the user into the create quiz page, where they can view the fields to add Name, Description, and Questions. To add a new question, there is a huge “+ Add Question” floating button that will add a new question card into the page. For each question option that the user, clicking on the Answer button will select the option as the correct answer to that question. After the user is done and satisfied, clicking on the green “Done” button that is on the fixed navbar will bring the user to the quiz summary page where the user can review the questions, options, and also share their quiz to users. Sharing of quiz can be done via Telegram, Whatsapp, Email and also copying of the shareable link to their clipboard.

Creating a quiz is made easily accessible from the homepage with a huge call to action button. This makes it easy for the user to quickly reach the create quiz page, without having to go through nested menus.
In the create quiz page, a huge floating button is used to add questions to the quiz. A floating button is easily accessible, whether on desktop or on mobile, and users do not have to scroll all the way to the top or bottom just to add a new question into the quiz.
The *Done* button is placed in the fixed navigation bar for better accessibility and user experience as well, user can easily complete their quiz creation whenever they feel like they are satisfied.
We bring the users to the quiz summary page so that they can review the quiz questions and answers before sharing the quiz out to their audience. Once they are happy, they can share the quiz through various options. We chose to give them a few options so that they can easily share the quiz link through their preferred platform, instead of going through the hassle of copy pasting a link to send it out.

Doing a quiz

Once the user receives a link to do a quiz, the user clicks "Start" to start the quiz. This brings the user to the questions directly. In every question, the user gets to see the question as well as the choices (for the currently supported MCQ mode). The user may attempt to answer each question by clicking on an option. Once the user chooses an option, the interface will feedback to the user if the option chosen is correct or wrong. If it is correct, the option becomes green; if not, the option becomes red, and the correct answer is highlighted green. The owl at the bottom right (only visible on desktop) will also become sad if the wrong answer is selected. A “Next” button will appear after a question is answered, prompting the user to go to the next question. A progress bar at the top will also reflect the progress of the user in the quiz. In the end, the results of the quiz (how many questions are correct) is shown. The user also has the choice to try the quiz again, or to review the summary, or simply go back to home.

We predicted that the main way people will use our app to do quizzes is by getting a link from a teacher/friend. This means, we do not really need to show any quizzes at the front page, as we rely on the other person to share the quiz so that the user can do it. After they have attempted it, they can always access it through their homepage.
We decided to have the user select the option directly, instead of clicking an option and confirming it, so that the user can save time. An option to toggle these will be helpful, so that the user can decide by themselves if they want to confirm options or not.
A “Next” button will show before proceeding to the next question. This is to give the user some time to review the question again for a better learning experience. Since this is not a timed quiz, we want to give a non-stress environment optimal for learning.

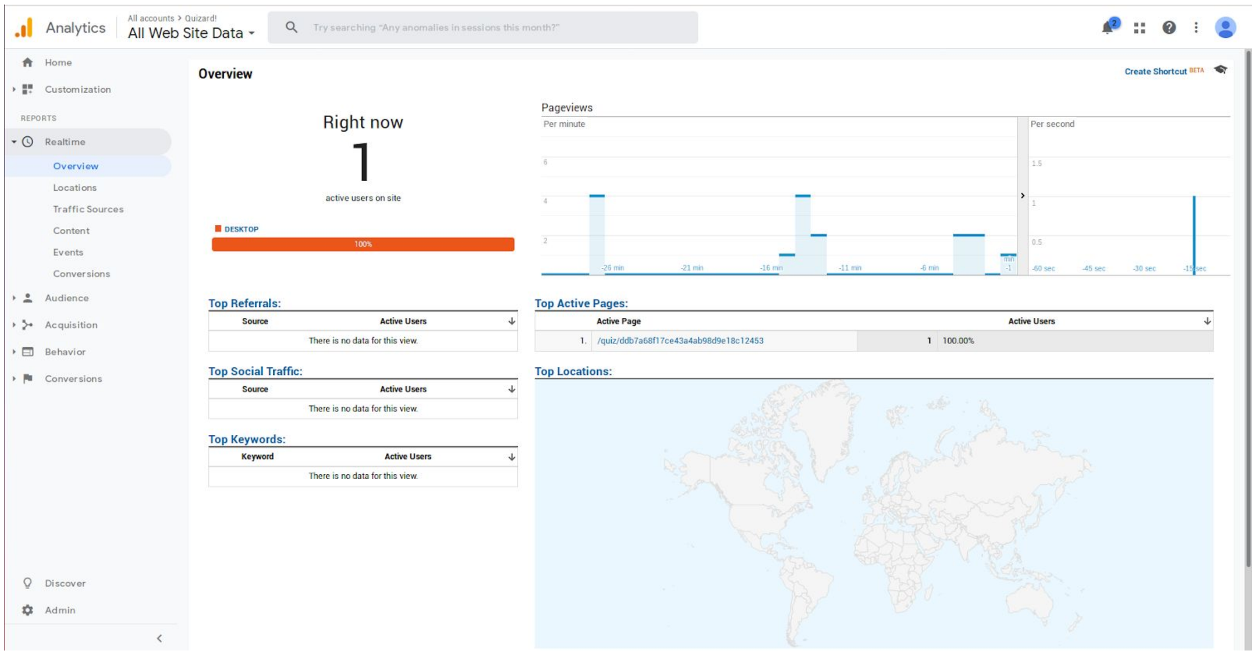
Reviewing a quiz

Once the user finishes a quiz, the user can review the results in a separate page. This page details all the questions, the correct answers, as well as the answers the user selected, and their score. The users can also access this page from the homepage, in the attempted quizzes section. Finally, the user can also try the quiz again from this page.

We decided to have this page so that the user can reflect on their mistakes and improve. We also kept this page to remind users of their own options as compared to the correct answer as posted by the quiz creator. This way, disputes can easily be settled (if there are disputes between answers given and the correctness of given answers). Also, in the future, comments can be given by the quiz creators on why they think that the answers should be as such, as a form of further explanation of their answers or thought process.

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.



Top Referrals:

Source	Active Users
There is no data for this view.	

Top Social Traffic:

Source	Active Users
There is no data for this view.	

Top Keywords:

Keyword	Active Users
There is no data for this view.	

Top Active Pages:

	Active Page	Active Users	
1.	/quiz/ddb7a68f17ce43a4ab98d9e18c12453	1	100.00%

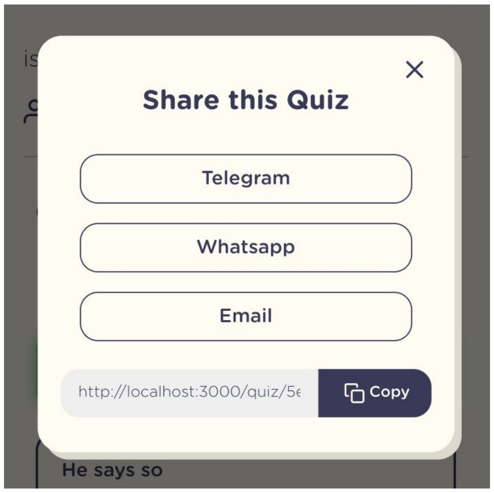
Milestone 14

Achieve a score of at least 90 for the Progressive Web App category and include the Lighthouse html report in your repository.

The report is in https://github.com/cs3216/a3-mobile-cloud-group-01/blob/master/lighthouse.html.

Milestone 15

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



For sharing of quizzes, other than just allowing users to copy the shareable link to clipboard, we also integrated the ability to easily share quizzes through Telegram, Whatsapp, and Email. For teachers to create quizzes for their students, the Email option would be useful for them, and even the Telegram and Whatsapp options if there is a group chat consisting of the teacher and their students. For other users, the Telegram and Whatsapp options would be useful for them to easily share their quizzes with their peers. We chose these 3 platforms as we think that it is the easiest way for our users to share their quizzes to a targeted group of people (the audience they want for their quiz).