CS3216 - Assignment 3

Mobile Cloud Application

Group 7

A0141128R - Chiang Weng Kiat

A0161364J - Nguyen Tien Trung Kien

A0144896M - Samuel Henry Kurniawan

rateyourprofs.com

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

For university students in Singapore, **Rate Your Profs** is a peer to peer review system that leverages on peers to help others with feedback on professors to help students better decide if a professor's teaching style/module would suit them.

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?**

- **Availability** - With all data and configurations stored in the cloud, the user will be able to enjoy the same experience any time and anywhere using any mobile device.
- **Low hardware requirements** - All heavy computation is offloaded from the client to the servers, reducing its role to an interface for sending commands to the cloud. In addition, parts of the data backed up in the cloud can be served on demand. The mobile device therefore, does not need to have large processing power and storage capacity.
    - We aim to exploit these two strength by have our servers do all the heavy lifting, data and configurations on either the client-server side or the rest-server side. This would allow users with any phone or even poor data connection to use our application with a good user experience.
- **Productivity** - What we previously could do only at home or in the office can now be done on the go. We are now able to read emails the moment it is received, edit documents or even catch up with friends over Facebook and WhatsApp during a commute.
    - We aim to exploit this strength by allowing our users to access our application via any browser which we made compatible. This would allow them to be productive where they would be able to catch up on new reviews or write a review anytime, anywhere on any device.
- **Device independent** - Want to replace your iPhone with the latest Android? A web application runs in any modern browser so there is no need to worry if your favourite application has yet to be ported to other platforms.

- We aim to exploit this strength by making our app compatible with any server, such that an iphone user could use safari and get the same experience on chrome with an Android phone. This would allow users on any device to enjoy our application and not care to make it compatible.

We feel our application is perfectly suited for the cloud since

All the data on our application is shared, hence, it makes sense for us to handle all the data on our server side, without the client side needing to store any data, native applications would be useless, bulky and less portable. We do not store user data and use 3rd party authenticators such as google login to regulate people from spamming posts.

Our application does not require much computation power, hence it can be easily handed off to the server and still expect a smooth user experience with minimal latency. Also, we want users to be able to access our application anytime, anywhere with the same data and contribute to it, hence, making it device independent would be crucial. We want people to be able to use it on any device, hence this would allow less powerful hardware to get the same user experience.

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

Our target users would be University students. However, we aim to pilot the MVP within computing first and then open it up to the rest of NUS. Then hopefully, to the rest of Singapore.

How to promote the application

- Encourage friends to put reviews to attract people to the platform
    - Reviews are the main value proposition of the application, hence, this step would be crucial in getting more users and retaining existing ones. We could make the application seem fun to people of our age to want to contribute to reviews, making the chore an enjoyable task.
- Encourage communities to work with us to promote this application.
    - For example, we could start with computing, listing all the professors in computing and encouraging computing students to contribute for their juniors and to benefit from their schoolmates inputs. We could work with professors to promote this application or societies such as computing club to spread the word about this application.

- Collaboration with other university life apps, i.e. NUSMods
    - We could collaborate with perhaps NUSMods, where if the user see that the review section is empty, it could be redirect to our application to see if there are reviews for the professor for other modules. Vice versa. This would allow users to get a wider range of reviews when planning their timetables.

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.

https://rateyourprofs.docs.apiary.io/#

Client–server

This essentially means that client application and server application MUST be able to evolve separately without any dependency on each other. A client should know only resource URIs and that's all. This is what is practiced in our application and it is impossible to access the database from the client side. The client can only modify and retrieve data via our REST Api.

Stateless

All client-server interaction are stateless. Server will not store anything about latest HTTP request client made. It will treat each and every request as new. No session, no history.

If client application needs to be a stateful application for the end user, where user logs in once and do other authorized operations thereafter, then each request from the client should contain all the information necessary to service the request – including authentication and authorization details in the request header.

Layered system

REST allows you to use a layered system architecture where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C, for example. A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. In our application, our database and api is hosted on two different servers.

Uniform interface

As the constraint name itself applies, a resource in the system should have only one logical URI and that should provide a way to fetch related or additional data. The endpoints are coded such that the input from the client side is in the url, without using `q?=`. Hence, keeping the API consistent and neat.

Milestone 5: **Share with us some queries (at least 3) in your application that require database access. Provide the actual SQL queries you use (if you are using an ORM, find out the underlying query and provide both the ORM query and the underlying SQL query). Explain what the query is supposed to be doing.**

Query 1

```
SELECT p.id, p.first_name, p.last_name, p.rating, name AS department FROM (SELECT
* FROM Professor WHERE full_name ~ 'search_term') p

LEFT JOIN department ON p.department = department.id;
```

This query is used in the https://rateyourprofs.docs.apiary.io/#reference/0/get-professor-by-name/ endpoint where the client uses it to search for professors in the database. The desired data is achieved by filtering the professor table by seeing if the search term is a substring of any entry in the professor name and then joining the results on the department table to get the department the professor belongs to. The Professor table stores a foreign key in department column which references the primary key in the department table.

Query 2

```
SELECT p.id, p.content, p.rating, p.difficulty, name AS module, score AS grade,
p.upvote, p.downvote, p.time_posted, definition AS tag
FROM (
SELECT * FROM post WHERE prof_id = {prof_id}) p
LEFT JOIN post_tag ON p.id = post_tag.post_id
LEFT JOIN tag ON post_tag.tag_id = tag.id
LEFT JOIN module ON p.module = module.id
LEFT JOIN grade ON p.grade = grade.id;
```

This query is used in https://rateyourprofs.docs.apiary.io/#reference/0/get-reviews/get-all-posts-for-a-professor endpoint to get all the reviews on a professor. The desired data is achieved by first filtering and getting the posts which belongs to the professor based on the prof_id passed to the url. Upon getting that results, we would proceed to join on post_tag table and then on tag table to get the tags. This is needed as the relationship between post and tag is many to many so we would need a third table here to maintain the relationship, storing the primary keys of each table as foreign keys. Finally, joining on grades and module since the post table stores the primary key of each table as foreign keys.

Query 3

```
UPDATE post
SET upvote = upvote + 1
WHERE id = {post_id}
```

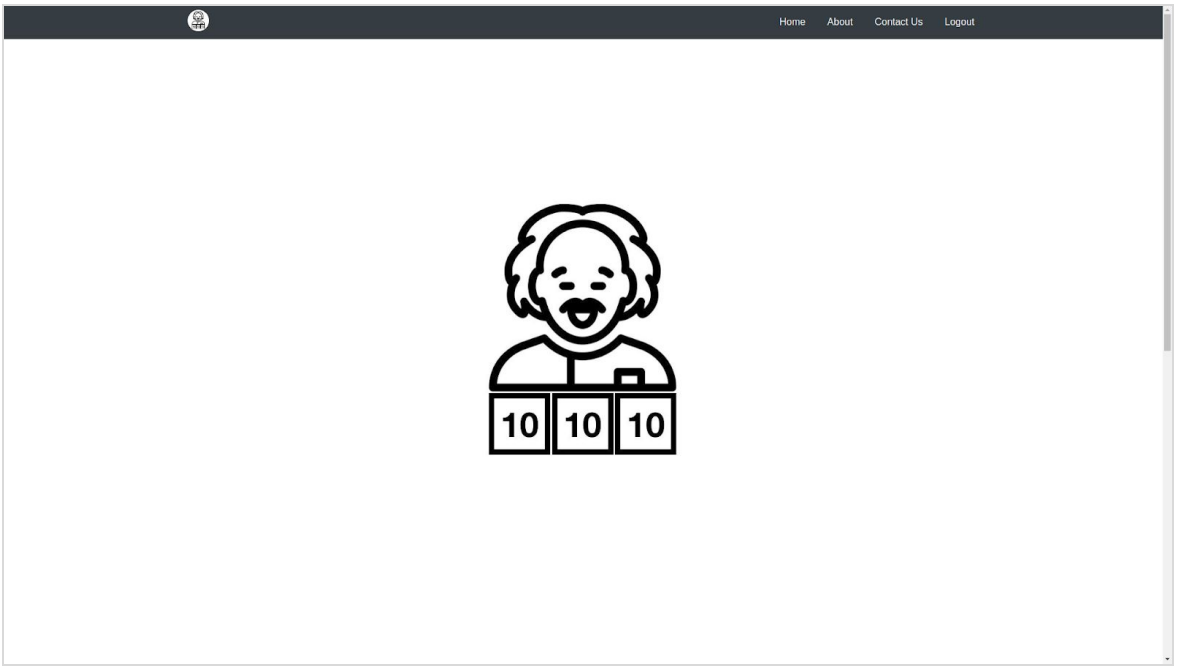
This query is used in https://rateyourprofs.docs.apiary.io/#reference/0/upvote-a-post/upvote endpoint to upvote a post. Here, we just do an update on an existing post by taking the current value and adding it by 1 to ensure atomicy.

We choose to only show these 3 queries over the mutlitude of other queries as we feel they are the more important ones. Update has significance since it must be done properly to ensure it is atomic and inline with making the API Idempotent. The select queries we have chosen are the more complicated ones with joins on other tables after filtering to pass accurate information to the frontend.

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.

Icons

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.

Almost all the element using BEM methodologies, because

- Easier to read, as there is no nested selectors and can be used using class names that are very intuitive for pointing to the selections
- No name conflicts, every unique blocks live on different namespaces
- Modularized css names, easier to develop further in the future

However, for general theme, which we limit it to font size and background color, we used principle of ACSS. The reason is that for general theme, we want to able to change them fast with minimal impact to other places, and the best methodology that allow it is ACSS (i.e. we want to change the grey background from

#F2F2F2 to #D3D3D3, if we using BEM for this, we need to change every single thing that have the grey background, where in ACSS we can just change a single class property).

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.

Three best practices for adopting HTTPS

Use robust security certificates	Obtain a security certificate as a part of enabling HTTPS for your site. The certificate is issued by a certificate authority (CA), which takes steps to verify that your web address actually belongs to your organization, thus protecting your customers from man-in-the-middle attacks. Ensure a **high level of security by choosing a 2048-bit** key during setting up your certificate,. Upgrade certificates with a weaker key (1024-bit) to 2048 bits. When choosing your site certificate, keep in mind the following: ● **Get certificates from a reliable CA that offers technical support.** ● **Choose a suitable cerfiticate:** ○ Single certificate for **single secure origin** (e.g. www.example.com). ○ Multi-domain certificate for **multiple well-known secure origins** (e.g. www.example.com, cdn.example.com, example.co.uk). ○ Wildcard certificate for a secure origin with **many dynamic subdomains** (e.g. a.example.com, b.example.com).
Support HSTS	HTTPS sites should support HSTS (HTTP Strict Transport Security). **HSTS tells the browser to request HTTPS pages automatically, even if the user enters http in the browser location bar.** It also tells Google to serve secure URLs in the search results. All these **minimizes the risk of serving unsecured content to your users**. To support HSTS, use a web server that supports it and enable the functionality. Although it is more secure, HSTS adds complexity to your rollback strategy. We recommend enabling HSTS this way: How to develop an application which supports HSTS:

	● Roll out your HTTPS pages without HSTS first. ● Start sending HSTS headers with a short max-age. Monitor your traffic both from users and other clients, and also dependents' performance, such as ads. ● Slowly increase the HSTS max-age. If HSTS doesn't affect your users and search engines negatively, you can, if you wish, ask your site to be added to the HSTS preload list used by most major browsers. ○ Consider using HSTS preloading ○ If you enable HSTS, you can optionally support HSTS preloading for extra security and improved performance. To enable preloading, you must visit hstspreload.org and follow the submission requirements for your site.
Use server-side 301 redirects	Use server-side 301 redirects **Redirect** your users and search engines to the HTTPS page or resource **with server-side 301 HTTP redirects**.

Certificate pinning

Certificate pinning is a approach to prevent the users from being victims of cyberattacks using fraudulent SSL certificates. An application using certificate pinning adds another layer of protection beyond the normal X.509 certificate validation: it only accepts the SSL certificates from trusted sources, a predefined list which is stored in the application itself.

For example, if an user connects to google.com from a browser, the certificates from Verisign, Digicert or any other SSL certificate providers will be accepted. However, if the user uses the Google app on a mobile phone, only SSL certificates signed by Google are accepted.

Pros: More secure.

Cons: Dangerous if the SSL certificates in the whitelist are compromised.

Should we use or not? No. We should only use this if our data is highly sensitive and when we are sure that the SSL certificate providers in our whitelist are more secure than others since we will end up getting the free ones due to budget constraints, we were not convinced that it is more secure. There would also be a lack of technical support which may cause issues in the future.

HTTPS or HTTP

Considering the trade offs, such as the inability to use server workers etc. We decided to proceed with using HTTP instead of HTTPS. We felt that we did not need it since we are not dealing with highly sensitive

information, credit cards etc. Hence, a man in the middle attack may not be our greatest concern and we rather push out other features or better the user experience in terms of design and workflows. However, this decision has cost us as we are unable to utilise service workers which decreased our offline functionality.

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.

The offline functionality we implemented is that the user is able to continue reading reviews for a professor when the connection is problematic. Also, having the full functionality of the review filter button to help the user to filter the reviews based on modules. That particular offline functionality is able to be achieved because after the page is fully loaded, user does not need to send anything to the server in order for read all the reviews. Moreover, the filter button, which help the user in terms of selective reading, was fully implemented using client-side javascript (it does not need to send request to the server).

The ideal offline functionality that we have not implemented for our application is that the user still available to upvote or downvote a review, and also able to submit new review, where the cache will store those POST requests, and when the user is back to full internet connectivity, the POST requests will still be transmitted to the server and the database can be updated at that moment. These would be handled by service workers.

We did not implement the ideal offline functionality as we were unable to utilise service workers in our application. To be able to use service workers, HTTPS connection is needed (https://developers.google.com/web/fundamentals/primers/service-workers/). Since we are not using HTTPS, we are not able to implement this ideal offline functionality. We decided on this trade off when we chose HTTP over HTTPS as we felt that we as a pilot, this is not a core feature since the user base would not be huge, hence, we focused more on features that would provide the user with a better user experience.

Milestone 10: Compare the advantages and disadvantages of token-based authentication against session-based authentication. Justify why your choice of authentication scheme is the best for your application.

	Session-based authentication	**Token-based authentication**
Description	Cookie-based authentication is **stateful**. An authentication record or session	Token-based authentication is **stateless**. The server does not keep a record of which

	must be kept on both server and client-side. The server needs to keep track of active sessions in a database, while on the front-end a cookie is created that holds a session identifier, thus the name cookie based authentication. It does not allow the API to be stateless.	users are logged in or which JWTs have been issued. Instead, every request to the server is accompanied by a token which the server uses to verify the authenticity of the request. The token is generally sent as an addition in the header. Authorization header in the form of Bearer {JWT}, but can additionally be sent in the body of a POST request or even as a query parameter.
Advantages	Easier to use Native	Stateless Scalable
Disadvantages	Stateful Not scalable	More tedious (has to send token on every request) Larger request payload

We used token-based authentication:

1. We decided to build a REST api, where a key concept is to ensure that the api is stateless. Tokens allow us to keep the api stateless and have a layer of authentication
2. Another key feature of a REST api is to make it scalable, making it able to serve the desired features on any server. Sessions do not allow for that since each server will have to store an authentication record, which may confuse the frontend client. Hence, we went with a token based authentication so our server can be deployed on multiple servers and requests from the frontend to any of the servers will achieve the exact same behaviour.
3. Session based authentication is really not what is needed for REST API and violates two principles we mentioned earlier, the tradeoffs mentioned in the table for the token based authentication is minimal and does not violate REST principles.

Milestone 11: Justify your choice of framework/library by comparing it against others. Explain why the one you have chosen best fulfils your needs. Lastly, list down some (at least 5) of the mobile site design principles and which pages/screens demonstrate them.

We decided to use Node.js (client server), React, React-Bootstrap, Flask, SQL Alchemy, Albemic, Flask-JWT and React google login

React-Bootstrap

For standard functional elements such as navbar, responsive div, and forms, we used bootstrap to make them. The main reason of the usage of bootstrap is that it is much easier, faster and more consistent to make the website responsive (instead of doing it manually one by one in the css using media query).

React

1. It uses one-way data binding. What the user sees depends on only two things, which is props and state. Therefore, it is easier for us to debug.
2. There are a lot of external libraries.
3. Allow us to do html in javascript instead of vice versa as in Angular, allowing development to be cleaner and much easier.

react-google-login

We decided to use a 3rd party authentication system to authenticate users, hence relieving us from the need to store user information and use server resources. We use the React google login library since we do not have to store user database, hence, it should be done in the frontend.

Node.js

We decided to go with two seperate servers to improve user experience and make it more scalable. We used node.js to render the client side server since it is lightweight and serves our purpose.

Flask

We decided to go with two seperate servers to improve user experience and make it more scalable. We used flask to render the API since we do not need a fully featured django to route our web pages which is handled by node.js in the front end. It allows us to build it up quickly and it is scalable since it is restless as explained in the milestone above.

SQL Alchemy

SQL injection is a very common way to exploit servers, since we are using Flask, it does not come with an inbuilt ORM unlike django. Hence, SQLAlchemy provides this layer of ORM for us, allowing us to prevent SQL injections since SQLAlchemy handles our sql queries for us.

Albemic

The reason we created a REST API is to make it scalable, hence, the migration feature for databases is very important. It allows databases to be created and populated easily from scratch. Flask does not come

migration unlike django, hence, we use Alembic to keep track of database structure and data changes and keep the different servers running the same API up to date with the same schema at all times.

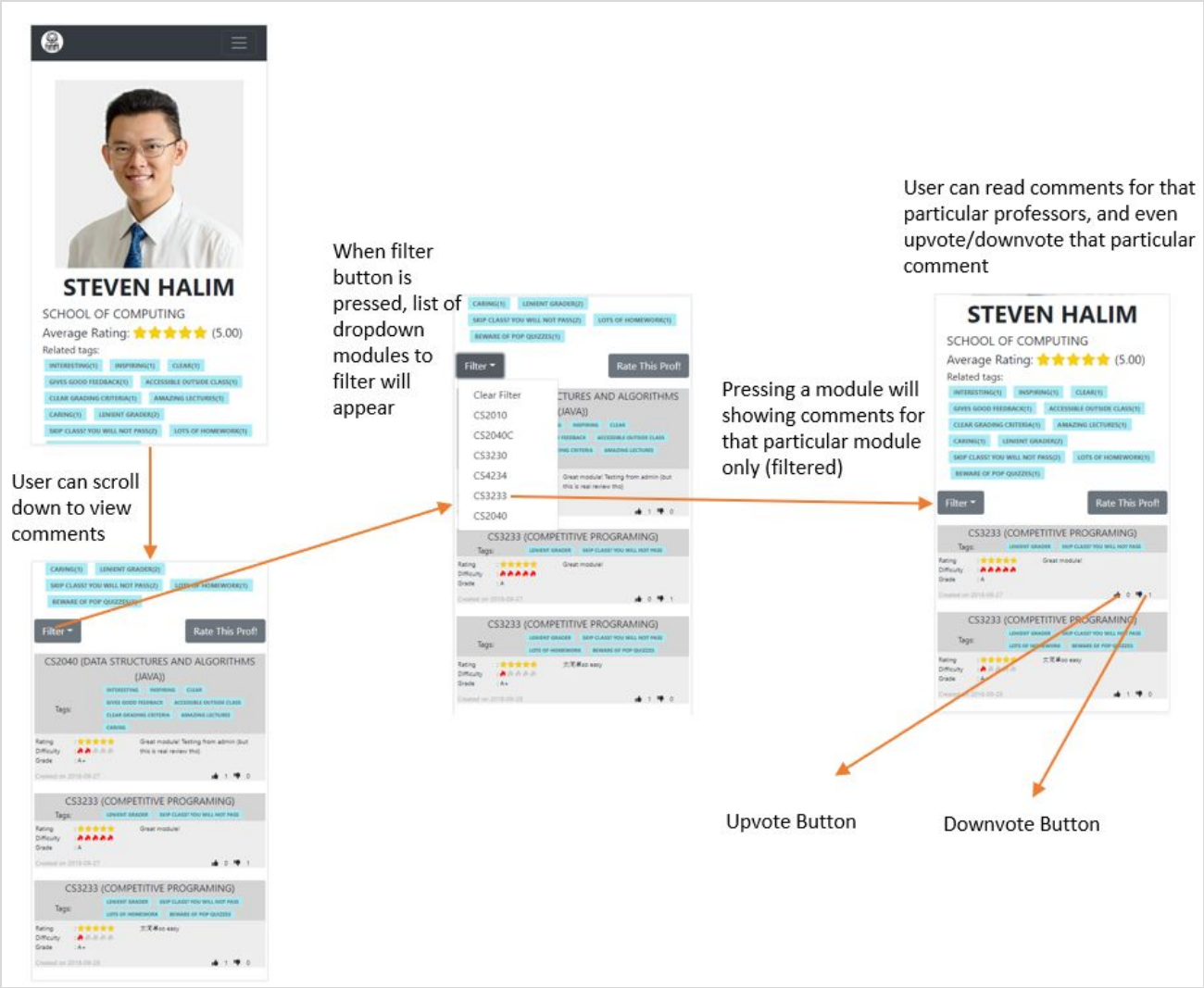
Flask-JWT

We decided earlier on that we favour token based authentication due to the way we layered our application, a separate frontend and backend. Since Flask comes bare, we have to use this library to enable token authentication on our REST API.

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.

The first common workflow is to search a professor. In order to do that, user can type in the search box in the home page, and either choosing the recommendation from the autocomplete or pressing the “GO” button to list all the professors that have that particular substring in their name (this probably will be preferred by user if the user wants to see average rating of several profs at once, or when the app is already growing and there is a lot of professors with similar name (because the number of autocomplete suggestion is limited to 5

currently)). We are choosing this workflow because this workflow is more intuitive for user and hassle free (users don't need to look around other pages, they can just type in the first input form they see when loading the app).



The second common workflow is to read the professor's page and comments that students posted. In order to do that, from the professor's page searched by the first workflow, they can just scroll down until the end, where everything can be seen in one page. There is also filter button incorporated if the user only want to know about the professor's rating on a particular module. Moreover, user can upvote and downvote comments, such that user can see whether a comment is a useful comment or just some random comment. We are choosing this workflow because the main thing about this app is the profs, such that we want to group all the comments to a particular prof into one part, and some professors are teaching differently in different module (maybe a prof can be good at teaching advanced module but not so good in teaching introductory module), so the user that want to take a particular module under a particular prof can read all the filtered comments for that particular module to weigh whether they still want to do it or not.

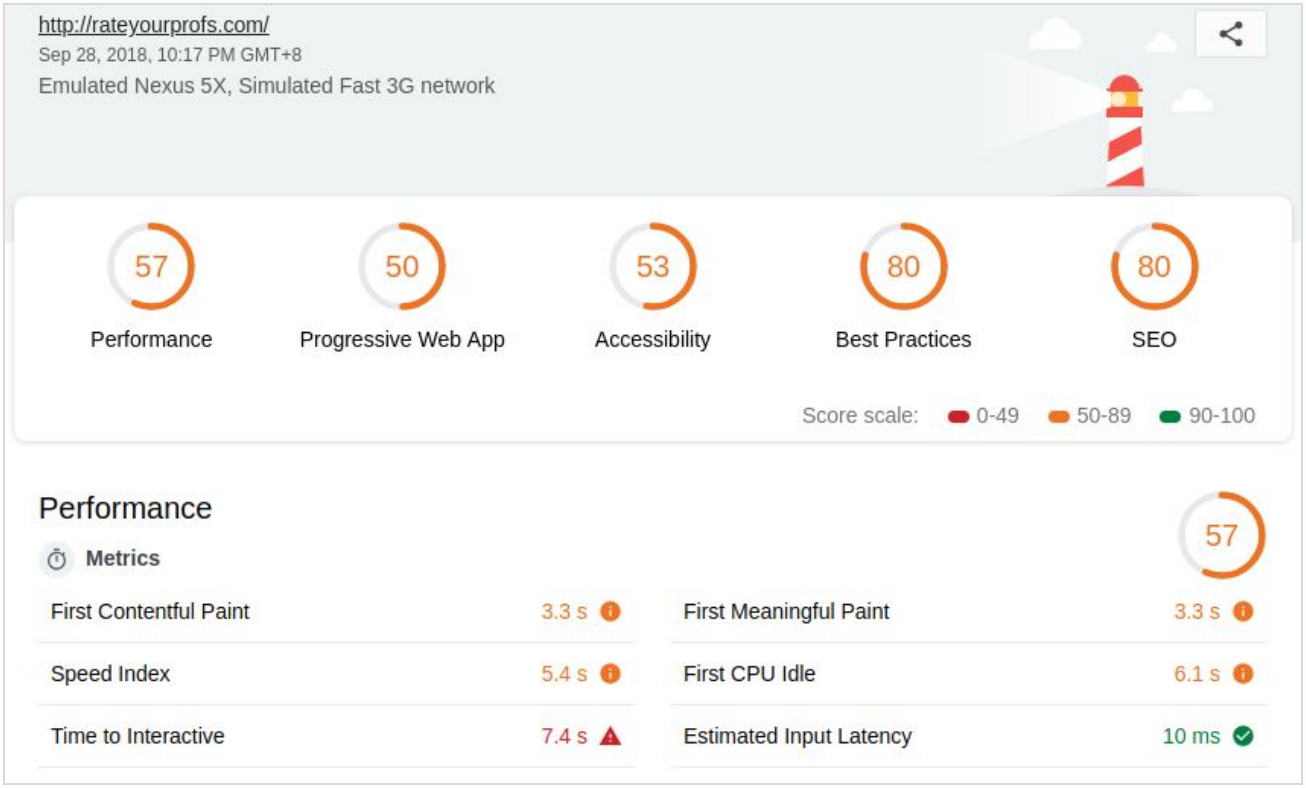
The last (but not least) common workflow in this app is to post a comment/review to a professor. To do that, from the professor's page searched by the first workflow, simply click "Rate this Prof" button. After that, if the user not logged in, the user will be directed to google login page. After the user is logged in, they can access the form to rate the profs. We implemented this workflow because we want more credibility to user that give the review, since if login is not implemented, a random person that hate one of the prof can just post similar reviews multiple times without hassle. After the login processes finished, the user can put his module and grade obtained (to help the reader to account for biases, since people that obtain lower grade tend to speak worse about the module compared to people that obtain higher grade). The user also can give rating for the prof, state the difficulty level of the module under that prof, and give "tags" that explain about the profs. Lastly, empty essay input also available to help the user justify their choices or if the user want to rant more about the prof. After the form is submitted, user will be redirected back to the prof's page, where the user can see immediately what just submitted and all the changes caused by it (i.e. the prof's average rating, taglist related to the profs)

Milestone 13: **Embed Google Analytics in your application and give us a screenshot of the report. Make sure you embed the tracker at least 48 hours before submission deadline as updates are reported once per day**

Milestone 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 was included in the repository.

The low score in the Progressive Web App was mainly (if not all) caused by the non-usage of HTTPS and the lack of service workers.



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)

We decided to integrate with google since all our reviews are anonymous hence, google would be the easiest to implement. We used google authentication so we move the work of authenticating users to 3rd parties, keeping our application lightweight and fast, as well as skipping the need to store data. We need authentication to prevent spamming of reviews, hence, of a user wants to post a review, he would have to log in with this google account. We would not store any of that information but it is done simply as a deterrent to spamming. Users are free to view all the content without login.

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

We are not implementing this feature because this feature is not needed in our application. We do not need the user location as it would not serve to improve the quality of the reviews.