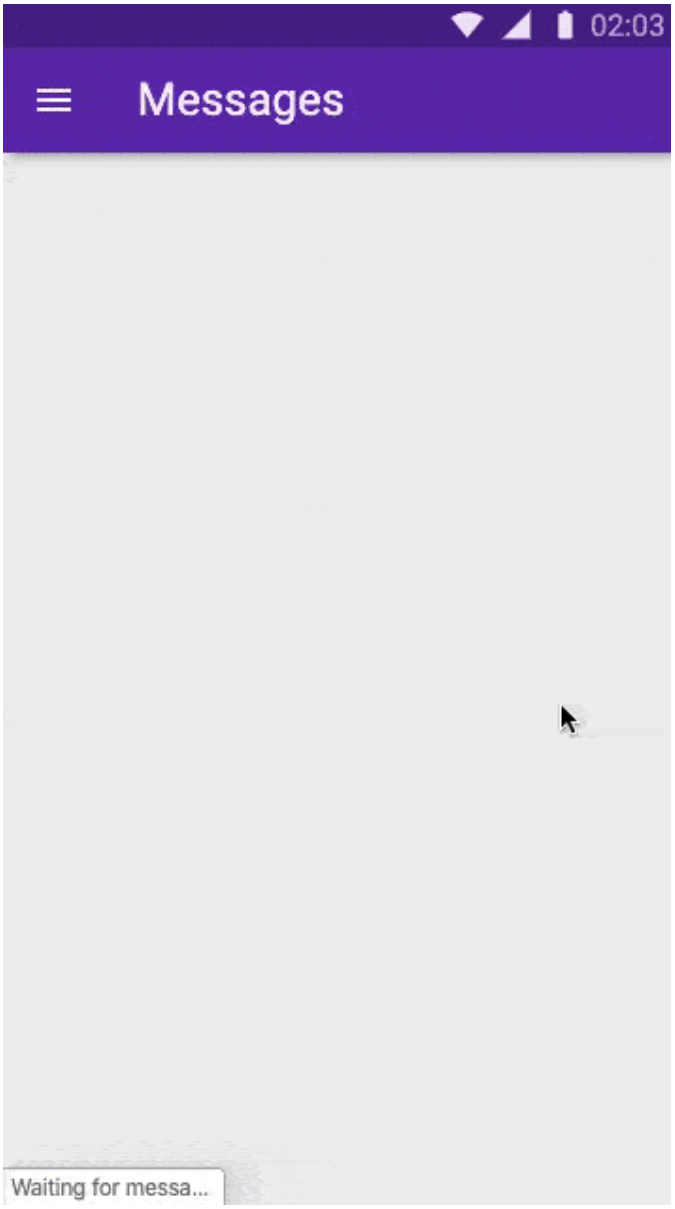
Google Design

UX Engineering - Design Exercise

Infinitely Scrolling Message List



Date: 8th February, 2020

Submitted by: ABHIJEET SAXENA

Objective

Build an interface that shows a potentially very long list of messages. You can build on your platform of choice (iOS, Android, or web), but users should be able to scroll through messages smoothly on a mobile device.

Requirements

- The header should be fixed with messages that scroll underneath.
- Users should be able to swipe a message horizontally offscreen to dismiss it.
- Messages should load automatically so that a user scrolling slowly should never see the bottom of the list.
- Your interface should work smoothly even after many messages have been loaded.
- The message list should be viewable on a mobile device, and any code you produce should be clean and extensible.

Thought Process

Personas

Kept the general audience in mind (mostly mobile users with a compatible experience on the desktop).

User Flows

Provide endless content to a user so that a user scrolling slowly through the messages doesn't encounter any loading of messages. This eager loading will also help improve the user experience of the user on a slow network.
Provide an easy way to dismiss messages i.e. by swiping right on the message card.

Design

Follow material design guidelines and good Design Principals like Colors, Contrast, White space, Typography, Alignment, Visual hierarchy and Scale.

Development Process

1. Low fidelity design

   Start with some hand-drawn designs and wireframes (See wireframe.cc links). Decide upon the design specs like the color scheme, font-size, card dimensions, etc. Create a basic layout with dummy data.

2. Small working prototypes

   Experiment and create small prototypes. Create a list with infinite scrolling with mock data in codepen, similarly implement the swipe and snackbar animation separately.
   (For example Link 1 , Link 2)

3. Bringing pieces together

   Connect the earlier developed prototypes with actual API data and if time permits, implement 'UNDO' functionality and fine-tune the app responsiveness for both mobile and desktop layouts.
   Start version controlling to keep track of all commits and history.

4. Hosting / Deployment

   Once development done, generate a production optimized build and deploy to Firebase Hosting.

Engineering Considerations

Project Setup, SCM and Deployment

The project uses **HTML**, **CSS** and **Vanilla JavaScript** for all implementation. **Parcel** as a bundler, **Babel** and **PostCSS + Autoprefixer** for transpiling ES6 to backward-compatible JavaScript and browser vendor prefixing, respectively. **Github Private Repository** and hosted on **Firebase.**

Infinite Scrolling

To implement Infinite scrolling, used **Intersection Observer API** (didn't use scroll listeners for performance reasons See this).

Animation

To implement swipe animation, used **Native CSS transitions** (mainly transform and opacity properties) and not any external libraries.

Assumptions

- I am only showing the first 200 characters of each message (remaining characters are masked by an ellipsis). This is done to keep the maximum height of cards consistent.

- The Messages API's response doesn't have the correct timestamp. To comply with the designs I am generating a random timestamp for each card. As a result, there is no particular sorting order in which the cards appear. That being said, in an ideal world the API response would be perfect and we can easily show relative timestamp.

- Also as there is no database that keeps track of the user's history the application completely resets on refresh without reflecting which card the user had dismissed before refreshing.

Bonus Features

Undo Functionality

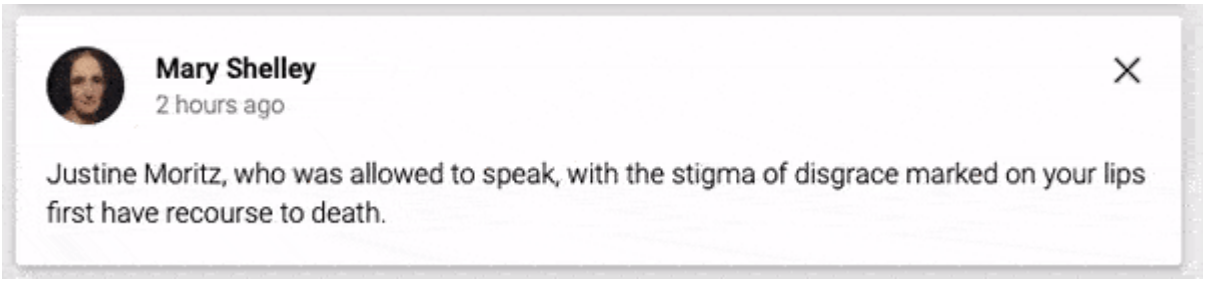
[How to delight User ?]

Users may unintentionally swipe a card or may want to revert their decision so I've implemented an **Undo Feature** that gives users 3 seconds after each card dismissal to bring it back. I've used a snackbar at the bottom to achieve this. This snackbar has been styled to resemble Material Snackbar.

Desktop Responsiveness and Design

[Improve usability]

Also to improve User Experience on desktop and large screens, I've added the functionality of dismissing a card by clicking on '**x**' icon. This could make it easier for some users to dismiss a card by clicking rather than pressing and holding the mouse button and then swiping.



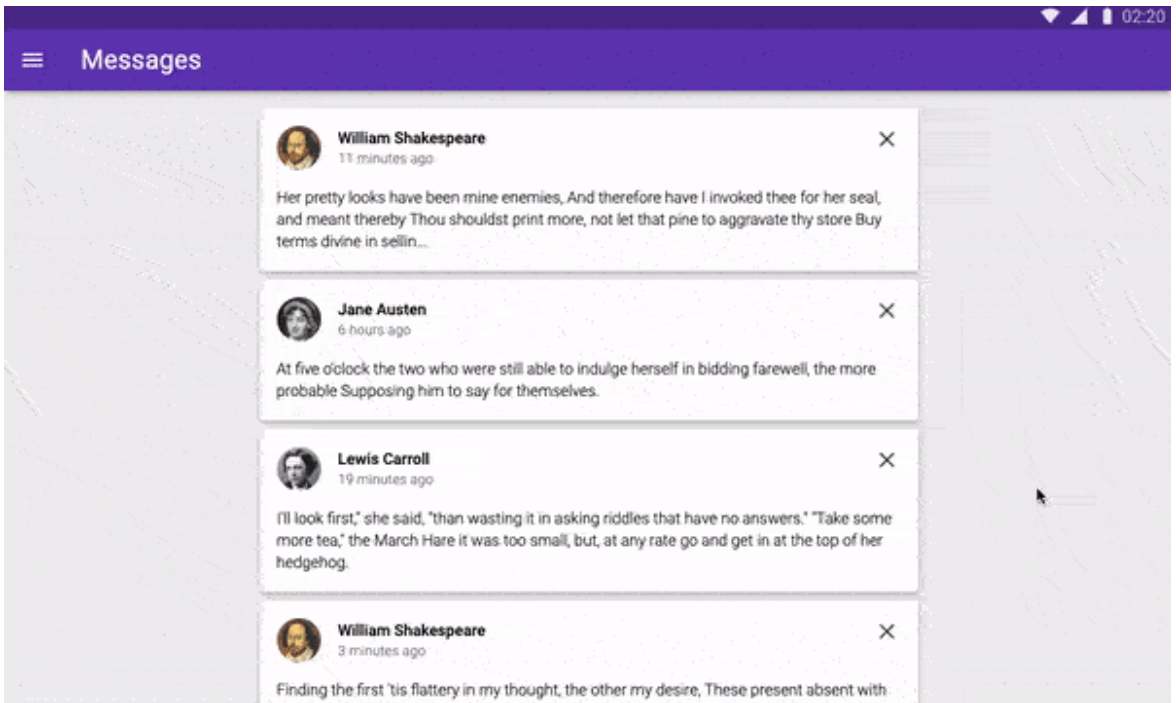
Status Bar

[Resemblance to real world app]

Although the expectation was only to develop the Infinitely Scrolling Message list, for giving a more realistic feel to the prototype I have also developed the topmost status bar and Header designed as per the screenshots given to me.
PS: The time in the status bar **keeps updating to the current local time**.

Final Project

Go to https://infinite-scrolling-message-app.firebaseapp.com/ for a live demo.



If given additional time...

- Implement Loading/Skeleton Screen

  For the first time load if the API response is too slow, instead of showing a blank screen we can render empty cards with animation to give a perception that data is coming. (See this)

- More intuitive swiping of message

  While swiping show the user a preview of what this action will do to provide context of that gesture action (See this). For example, in Gmail, if we swipe an email to the left or right we are shown what that swipe does.

- Memoization of messages and user actions

  Right now we make a fresh start every time the user reloads the page. For a snappier experience, we can store the users' actions and cache API response. We can make use of localstorage and Service Workers for this.

- Properly thought out Desktop Layout

  Rethink the desktop design since lots of screen real estate is wasted because we are showing only a single card per row. For example, we can show messages in a grid-like format. Also a fade out transition may look better than swiping which may become too distracting.

- Revisit animations and interactions

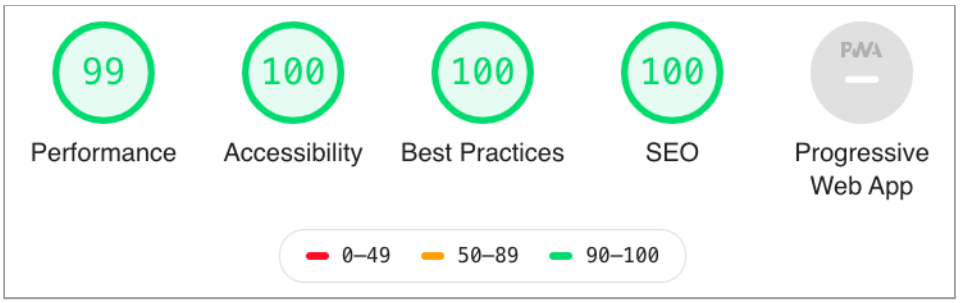
  Although the current animations are working smoothly, we can see if they can be optimized even more by using requestAnimationFrame or an external library like Anime.js. Also for swiping the cards, we can use a library like interact.js to handle any edge cases.

- Progressive Web App (PWA)

  If this app is also supposed to be used offline, we can build this as a PWA.

Technical Review

Performance



Lighthouse Report — Mobile | Desktop

Speed

Page Speed Index Report — Insights Link

Good Practices

Semantic HTML, passive event listeners for touch inputs, preconnecting with API, use of GPU accelerated CSS animation.

Accessibility

Proper use of color, contrast between text, font size, font types, descriptive text or alt tags. Use of aria-labels.

Supported Platforms

Use of CSS vendor prefixes + Babel. Works in major mobile/desktop browsers except for maybe IE11(Not sure). Tested in Chrome, Safari, Mozilla Firefox and Microsoft Edge.

Resources & References

- https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API
- https://developer.mozilla.org/en-US/docs/Web/API/Touch_events
- https://developer.mozilla.org/en-US/docs/Web/CSS/transform
- https://material.io/design/
- https://material.io/resources/icons/?style=baseline
- https://material.io/components/snackbars/
- https://caniuse.com/
- https://developers.google.com/web/fundamentals/performance/rendering
- https://developers.google.com/web/tools/lighthouse/audits/passive-event-listeners
- https://www.freecodecamp.org/news/how-you-can-easily-make-your-website-more-accessible-88dc7db90bd2/