Tuesday, June 25, 2019

Design Document for Infinite Scroller

Purpose: Design Infinite Scroller with extensibility, quality and performance in mind.

Requirements

1. Platform of choice (iOS, Android, or web),
2. The header should be fixed with messages that scroll underneath
    - Use Position fixed, top: 0px, and z-index this way header will always be sticky to the top. Messages can be added so it will scroll underneath.
3. Users should be able to swipe a message horizontally offscreen to dismiss it.
    - User should be able to swipe right and dismiss items as shown in the mock, I could implement my own touch gesture using onDrag events, but decided to use Hammer Js to save time. (HammerJs is only 3kb)
    - Using translateX property to drag item, which avoids the reflow of dom.
    - Find out delta from HammerJs and check if its more than 50%, if so dismiss the item
    - If user drags item to dismiss, but decides to not dismiss, then use transform:translateX(0) to move back to its original position.
4. Messages should load automatically so that a user scrolling slowly should never see the bottom of the list.
    - On Scroll, we can check the position of the scroll and determine if user has scrolled to the end of page, if so fetch more data and append more messages to the page.
5. Your interface should work smoothly even after many messages have been loaded.

- Originally i thought i would not do dom recycle. But if user keeps scrolling for 10 mins. eventually dom can have thousands of elements and specially for mobile devices, that can be a big issue.
- Will do dom recycle and show limited amount of message on dom at any given point. (In my case 30 messages at a time in dom (keep it configurable))
- Keep tack of pages loaded and pageToken associated with that page. I decided to put pageId attribute on the elements so i could easily keep track of previous or next page needed to load.
- When scroll bottom, take the last child's pageId check in the map and get pageToken and fetch next data.
- when scroll top, take the first child's pageId check in the map and get pageToken and fetch previous data.

6. The message list should be viewable on a mobile device, and any code you produce should be clean and extensible.
    - Writing css to support mobile, tablet or webpage devices.

Architecture:

Building Infinite Scroller as library or utility component. 100% reusable and configuration driven. Usable at pageLevel or within smaller section on page.

Currently configuration takes 4 arguments but in future more configuration can be added to make component more extensible.

- container (Dom element id where Infinite Scrolller will be initialized)
- url: (url to fetch data)

- renderTemplate: (callback function(ele, data) responsible for rendering message UI given data)
- swipeDirection: (pass direction to swipe left or right)

Parent Page Duties:

- parent page needs to invoke the Infinite Scroller with mandatory arguments
- parent page is responsible for rendering message (tile)

Infinite Scroller Duties:

- Scroller should fetch data when necessary (configurable url)
- Scroller should do dom recycling (configurable)
- Scroller will handle touch gestures
- Scroller should expose api to clean component and destroy events when done

Dom Recycling:

- Simple dom recycling where based on batch_size, we fetch 10 records from API.
- Trademill or Sliding window approach.
- Each fetch call data, when rendered on Dom, they will be given unique PageId.
- If user Scrolled down, and fetched 3 times, there will be PageId 1, PageId 2, and PageId 3 on dom. In current case, each page has 10 items, so total 30 items.
- when user fetches 4th time, 10 new records will be appended to the bottom, but at the same time 10 old records with PageId 1 will be removed or recycled.
- when page 4 is loaded, now dom will have page2, page3 and page4. and page1 elements has been removed.

- Keep tack of pages loaded and pageToken associated with that page. I decided to put pageId attribute on the elements so i could easily keep track of previous or next page needed to load.
- When scroll bottom, take the last child's pageId check in the map and get pageToken and fetch next data.
- when scroll top, take the first child's pageId check in the map and get pageToken and fetch previous data.

Future Plan:

- Need to write the tests
- Handle fetch API errors and show correct error messages on UI
- Component should work without url API, what about use case where data is already present and no need to make ajax calls.
- Touch Gestures can be configurable.
- Dismissing Items can be stored in map and can be cached and also ajax call can be made to update backend when possible.
- Support LTR vs RTL language support.
- **Bug:** When user scrolls towards the top rapidly, scrollTop becomes 0, since we are removing and adding new items to the top, there are times where scrollTop position stays to 0. In this case, scrolling to top is not possible, so you need to scroll down and scroll up to invoke fetch call.
  Fix: I would like to fix this bug by removing the items only after fetch is completed and render is complete, so we prepend items first so scroll would work correctly.
- **Scroll Size Improvement**: In current design scroll size remains same, since we are always keeping 30 nodes on dom. I have noticed that facebook keeps items in position: absolute. and keep tracks of item height.

  First item will be placed at top: 0px; and second after first element's height and

so on. With this approach above scrollTop bug will be resolved as well as it will maintain scroll height

- **Cross Browser compatibility**, since we are using element.closest and flex box css, this component may not work well in older browsers, I would find alternative to them by using some lib or write css without flex-box or use polyfills.
- **Accessibility**: I have already wrote semantic html, alt, title, but also add tabIndex, aria-label and other attributes. Implement Lighthouse suggestions for accessibility.
- **Lighthouse Tool**: Run LightHouse Google performance tool and make changes based on suggestions for performance improvement.

Workspace Setup:

- Using web pack create build output and to make code production ready
- Generated build output has been deployed to firebase

- Checkout the repository or unzip the folder
- root directory —— npm install
- root directory —— npm run build
- root directory —— npm start

Go to url: http://localhost:8080/

Dependencies:

- Lodash
- moment

- hammerJs for touch gestures

Component Usage:

```
const scroll = new InfiniteScroller({
    container: '#message_list_1',   // root element
    url: `${BASE_URL}/messages`,
    renderTemplate: renderTemplateCallback, // developer can implemente any template they need
    swipeDirection: SWIPE_DIRECTION, // left or right ( // TODO: more tough gestures can be configurable )
});
```

```
    // Future configs
    data: {},   // can it be implemented for static data, without fetching data
    batch_size: 10,
    isRTL: false, // is RTL language
```