Ecommerce website

Type something

Frame 1

Component responsibilities

1. Host website - embeds the widget as an iframe
2. App Server - Renders the widget UI as a website by serving th html, css, JS needed
3. API Server - returns poll results, records new polls
4. Client Store: Interacts with the API server and stores the data for the UI
5. Polls UI : UI for the poll options

Frame 2

Requirements

1. The widget displays a list of options for users to vote on.
2. The latest number of votes for each option is shown after the user has submitted the vote

Non Functional

- Easily able to embed in websites
- User experience as a voter must be good
- Can user vote on multiple options?
- Display the length of the votes casted relative to the total votes
- Display From most-to-least
- Poll options cannot be modified by the user.
- Set a max number of options
- Votes should be persisted for the user whether or not he has logged in or not

Frame 8

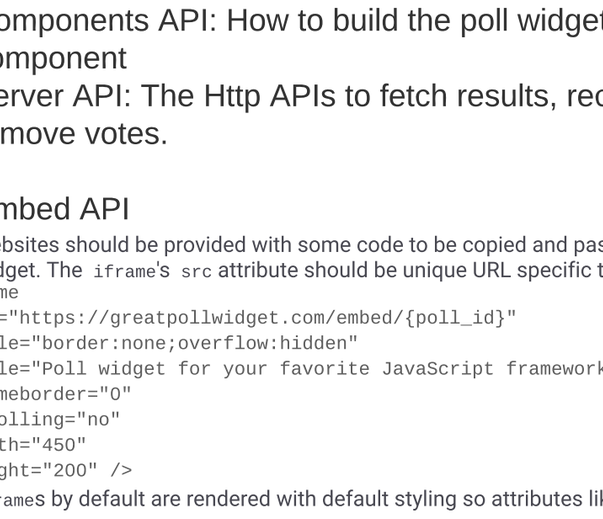
Architecture

Rendering Approach

1. Render in an iframe (different browser context)
2. Render directly within the page (Same browser context)
3. Iframe tag accepts the url of the website that needs to be embedded in the host wesite (ex: youtube embeddebale videos)
4. Pros - Styling of the widget is unaffected tby the host website css
   - The widget's JS environment is not affected by scripts running on the host which can contain polyfills, monkeypatching etc that can affect the runtime behavior in an unpredictable mannner
   - Simple to set up
5. Cons - It is slower to load the seperate content than to directly render within the page
   - Because of the isolation, the host website cannot use css to customize the stylings
6. Can load it either as a script or directly embed as an iframe. The latter is easy but less flexible.

In case of loading it as script, the page downloads and executes the external JS. the widget is then packaged as npm project, which can be added and used as a dependency. But its not a good idea because, not all websites are JS based.

Frame 3



Data Model

Frame 10

```
const state = {
  lastUpdated: 1628634891,
  totalVotes: 421,
  question: 'Which is your favorite JavaScript library/framework?',
  options: [
    {
      id: 123,
      label: 'React',
      count: 234,
      userVotedForOption: false,
    },
    {
      id: 124,
      label: 'Vue',
      count: 183,
      userVotedForOption: false,
    },
    {
      id: 125,
      label: 'Svelte',
      count: 51,
      userVotedForOption: false,
    },
    // ...
  ],
  // Which option(s) the user has selected.
  selectedOptions: [124, 125],
};
```

Frame 6

Interface Definition

1. Embed API: How websites should embed the <iframe>
2. Components API: How to build the poll widget and the props for component
3. Server API: The Http APIs to fetch results, record new votes, remove votes.

4. Embed API

   Websites should be provided with some code to be copied and pasted to render the poll widget. The `iframe`'s `src` attribute should be unique URL specific to a poll instance.

```
<iframe
  src="https://greatpollwidget.com/embed/{poll_id}"
  style="border:none;overflow:hidden"
  title="Poll widget for your favorite JavaScript framework"
  frameborder="0"
  scrolling="no"
  width="450"
  height="200" />
```

1. `iframe`s by default are rendered with default styling so attributes like `frameborder="0"`, `scrolling="no"` and the inline styles help remove borders and scrollbars to make the widget appear to be part of the page and not obvious that the widget is rendered by an `iframe`.

5. Components API

poll - server url, polloptions - Max options to display, PolloptionItem: label, number of votes for an option, eventhandlers.

```
// Example code in React
<Poll submitUrl="https://greatpollwidget.com/submit/{poll_id}">
<PollOptionList>
  {options.map((option) => (
     <PollOptionItem
       key={option.id}
       label={option.label}
       count={option.count}
       isSelected={option.userVotedForOption}
       onVote={() => {
         submitVote(option.id);
       }}
       onUnvote={() => {
         removeVote(option.id);
       }}
   />
))}
</PollOptionList>
</Poll
```

```
6. Server API: List of responses for each options
{
  "question": "Which JS library is better"?,
  "totalVotes": 421,
  "options": [
    {
      "id": 123,
      "label": "React",
      "count": 234,
      "userVotedForOption": false
    }
   ]
}

The vote submission/ remove api also returns the poll results in the above format.
```

Frame 7

OPtimizations

1. Persisting votes across sessions (because user may not have logged in)
2. Rendering the poll options (% based or relative to the top voted, where top voted takes the 100% width)
3. User shimmer effect while the poll is still loading
4. Hide the poll results before user has voted to reduce the bias.
5. Consider having a "See responses" function for people who just want to see the results without voting.

Performance

1. Fast rendering - Compute the total votes on the server side, rather than client side. Better yet, use the SSR for initial render.
2. Fast interactions with optimistic updates
3. In most cases, the client can render an optimistic update first and render again with the most updated results from the server.
4. Given there are only maximum of 6 options, we won't normally run into large resulting DOM. But if that happens and there are too many options, then we can use virtualized lists.

Accessibility

1. These are inherently visual elements. So ensure that screen readers can esily understand.
2. Screen reader users will not know how long a bar is, hence `aria-label`, `title`, `aria-describedbys` need to be used for the poll options to indicate the option name, the number of votes, and the percentage if they are not in the rendered visual output.
   - Use `aria-live` to announce updates for any change in values of the results when the client receives a server response.
   - ARIA roles for options: `role="radiogroup"` and `role="radio"` for polls where only one option can be selected.
3. If we are using divs instead of buttons, make them ppropriately focusable with role="button"

Network

Request responses could be out-of-order if someone votes/unvotes in quick succession
   - Keep track of the latest response (using tiimestamps) and ignore stale responses.
- Show errors in the UI if the API submission fails.