BSI.Snapview test assignment

Imagine you're writing a client application for screen sharing conferencing. Once your client app is started, a connection to the server via Web Sockets is established. The server sends two commands to your app:

1. Role bit. One bit of information telling whether your client would be considered a presenter (a person who shares a screen with others) or participant (a person who will receive screen sharing updates from a presenter and just participate in a session as a passive viewer).
2. In case the first bit is "you are presenter, you can share a screen with others", you receive the configuration preferences for the screen sharing, namely: the list of screen-sharing technologies which your client must try to apply when the user presses a start button in the UI. There are 2 supported technologies: "VNC Screen Sharing" (hereinafter referred to as "VNC") and "WebRTC Screen Sharing" (hereinafter referred to as "WebRTC"). The order these technologies are placed in the list defines the priority, i.e., if VNC is specified first, then your client should try to start VNC and use WebRTC only in case of failure (fallback).

In case your client has been configured as a presenter, there is one single button that must be displayed in the UI. Participants don't have any buttons and only contain a single canvas element (which will be used to display screen updates that will be received from the server), i.e., the participant UI is straightforward and does not have any special logic.

Upon clicking on a "Start" button in presenter's mode, it is expected that your client sends a special command to the server specifying which screen sharing technology it intends to start. The server will reply with either an acknowledgement (which means that your screen sharing is connected to the server and the server will relay screen updates to the viewers) or an error; a fallback should be tried in case of error, otherwise the error is displayed in the UI.

If the transmission starts successfully (server acknowledged client's request) and the requested screen sharing mode is VNC, the server tells whether the updates are sent from the mobile operating system or desktop operating system (hereinafter referred to as "mobile flag").

Now, when the screen sharing is started, the UI / state of your app changes:

- The "Start" button becomes a "Stop" button, pressing it should trigger a stop command sent to the server (assume it never fails).
- In case the current transmission mode is set to VNC without mobile flag set, the server will regularly send messages to your client, informing about the list of the available windows and screens. There should be 2 buttons in the UI: "Select Screen" and "Select Window", clicking on which will open a pop-up window with 2 tabs ("Screens" or "Windows"). If user clicks on "Select Screen", then active tab will be set to "Screens" (the same applies for "Windows").

After selection of a particular window or screen, your client should inform a server about user's decision (we assume that it also never fails), once this is done you can consider that you have an active transmission of a given window or screen. The name of the entity (let's say that windows and screens only have a title and an identifier) should be displayed above the canvas, so that the user knows what's being shared. During the transmission, the user can still click on any of "Select Screen" or "Select Window" buttons to change the transmitted entity. Additionally, there is a "pause" button on top of the canvas, which can temporarily pause the transmission, until the user presses on it again (resumes transmission). Clicking on this button should also produce a message sent to the server, informing it that we want to pause/resume the transmission (we can assume that this command never fails as well).

- In case the current transmission mode is set to VNC with mobile flag set, the transmission begins immediately and there are no "Select Window" or "Select Screen" buttons. The pause button is still present and has the same behavior. Instead of a title for the shared window or screen, there is just a simple label "Mobile Display".
- In case the current transmission mode is set to WebRTC, only one "Select Screen" button is shown, pressing it also triggers a pop-up where the user can select a screen, but in this case no additional information should be sent to the server, just assume that after selecting this screen, the transmission starts automatically (browser logic). The pause button should be present as well and have the same behavior as in all cases described above.
- There is a canvas in the middle of the screen which shows a preview of the content which participants will see. The canvas is only displayed when something is being transmitted.

Important

- You can develop the solution using TypeScript + React or Elm.
- You can mock the server response, no need to write the actual server.
- You need to design an efficient and safe model for such a task and write the corresponding view functions.
- The design does not matter, just roughly describe the structure of page in the views. You can ignore the update function (reducer) implementation for now.
- The most important thing is to design a good **data model**, which would allow upholding important business logic invariants and avoid impossible states.
- The result should be provided as a public GitHub repo.