1. The Intro

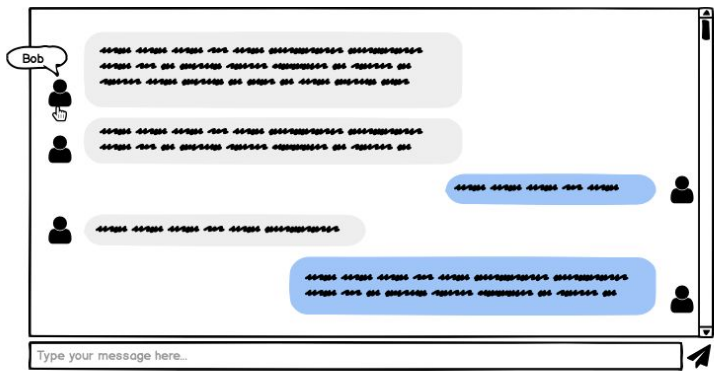
As part of this exercise you will be asked to create a running Vue application based on a wireframe mockup.

At the end of the exercise please provide us with a Github repo containing all the code you have written in the allocated time.

Naturally, you may use any resource available online (Google, SO, Vue docs/forums etc), if you have any unknowns about the implementation - just as you would on your daily job.

2. The UI

The app is a “messaging” interface as sketched below. Click here to download a higher resolution version.



For the purpose of this exercise, let’s assume the interface is `500px` wide and `400px` tall. However, it should be built in such a way that it can be used with any dimensions.

It’s made up of two sections; both will be described below.

You are not required to style the interface any further than described in this document; if something is not mentioned (e.g. what font sizes or colors to use), feel free to use whatever you want.

2.1 The conversation area

This is a list of message entries; each **message entry** should display the author’s thumbnail and a bubble containing the message.

- The thumbnail
    - Should be `48x48px` in a “circle” format
    - There should be `12px` margin between the thumbnail and the message bubble
    - Should be bottom aligned with the message bubble

- The message bubble
    - Should have `8px` rounded corners and `8px` paddings
    - Should not extend beyond the text content inside it (see the 3rd and 4th entries in the mockup above)
    - Should not extend wider than `60%` of the interface's width, at which point the text inside it must start wrapping, and the bubble will grow taller
    - Should properly display new lines (`\n`) in the message text, if any

Messages that belong to the currently logged in user (i.e. **sent** by *you*) are mirrored, right aligned and the message bubble will have a blue background (see the 3rd and 5th entries in the mockup above).

The conversation area should be scrollable if the messages overflow its available height.

2.2 The compose section

This is made up of a **textarea** where the user types his reply, and a **button**. The textarea must expand to the available width, whereas the button should always be `60px` wide. The button should be vertically centered with the textarea, and should have `12px` distance from the textarea.

Pressing enter in the textarea, or clicking on the button should

- Add a reply to the bottom of the conversation
- Scroll the conversation area to the bottom, if not already there
- Clear the compose section

The textarea should initially display one row for typing in text, but should be able to expand to display up to 3 rows as the user types, after which a scrollbar should be displayed inside the textarea.

When the textarea expands, the compose section should expand with it, thus reducing the space available to the conversation area.

3. The data

You must add this javascript file to your project and then import it as it is and use it where required.

The file has two exports, one containing the conversation, and one containing data about "the currently logged in user" (i.e. who _*you*_ are in the conversation).

4. The implementation

You are permitted to use any boilerplate or tooling to set up your project (e.g. `@vue/cli`), if you want.

You must use Vue to build the interface, and Vuex to provide the required data to the templates.

Additionally, you must not use any UI libraries (e.g. bootstrap or vuetify) to generate the components and/or layouts.

4.1 Vuex store

You should import the data file in your project and use it to create a "mock" API module providing each data point ("conversation" and "me") separately.

You should have two individual Vuex actions that perform "mock" asynchronous requests to this API to bring the data into the Vuex store.

The two actions must be called when the app is instantiated in order to generate the content.

You should define a third action, that will be used for simulating a reply being sent. This should also "mock" an asynchronous request that takes `0.5s`.

4.2 Components

The **conversation area** must be a separate component.
The **message entry** must be a separate component.

- You must use this component for both messages **received** from the other person, and **sent** by *you*.

The **compose section** must be a separate component.

- It should emit a `submit` event when the user presses Enter of clicks the button
- Ideally, the component should implement a v-model functionality, allowing the text inside the textarea to be kept in sync with a property defined in the parent component; e.g:

```
<compose-section v-model="message" @submit="sendReply" />
```

5. The extras

If time permits, you could implement the following.

5.1 Custom tooltips

When hovering your mouse over any thumbnail, a custom tooltip should appear displaying the author's name.
You may use any available UI library for these.

5.2 Optimistic UI reactivity

When submitting a reply, the new message should *instantly* appear in the conversation, instead of waiting for the asynchronous “send reply” mock api request to finish.

5.3 Error handling

Your mock API could have a 10% chance of failure (e.g `reject`ing the promise in 10% of cases). When a request doesn’t finalize correctly, implement what you consider to be adequate error handling.

If you have implemented the optimistic reactivity mentioned at 5.2, then in case of a failed request the interface should revert back to the state before the `submit` event was emitted.