Method 2: getSnapshotBeforeUpdate Extension in Detail

This lesson is an extension of the previous lesson. We'll discuss getSnapshotBeforeUpdate components and the chat panel in more detail.

WE'LL COVER THE FOLLOWING

The `componentDidUpdate` Lifecycle Method

The `getSnapshotBeforeUpdate` lifecycle method doesn't work on its own. It is meant to be used in conjunction with the `componentDidUpdate` lifecycle method.

Whatever value is returned from the `getSnapshotBeforeUpdate` lifecycle method is passed as the third argument to the `componentDidUpdate` method. Let's call the returned value from `getSnapshotBeforeUpdate`, *snapshot*. Here's what we get:

```
componentDidUpdate(prevProps, prevState, snapshot) {
}
```

The `componentDidUpdate` lifecycle method is invoked after the `getSnapshotBeforeUpdate`. As with the `getSnapshotBeforeUpdate` method, it receives the previous props and state as arguments. It also receives the returned value from `getSnapshotBeforeUpdate` as the final argument.

Here's all the code required to maintain the scroll position within the chat panel:

```
.App {
  text-align: center;
  display: flex;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 40vmin;
}

.App-header {
  border: 1px solid #282c34;
  flex: 1;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
}

.App-header img {
  border: 0
}

.App-link {
  color: #61dafb;
}

.App-chat {
  min-width: 400px;
  /* background: linear-gradient(
      -45deg,
      #183850 0,
      #183850 25%,
      #192c46 50%,
      #22254c 75%,
      #22254c 100%
    )
    no-repeat; */
}

/**
  chat
**/
.chat-thread {
  padding: 0 20px 0 20px;
  list-style: none;
  display: flex;
  flex-direction: column;
  overflow-y: scroll;
  max-height: calc(100vh - 50px);
}

.chat-bubble {
  position: relative;
  background-color: rgba(25, 147, 147, 0.2);
  padding: 16px 40px 16px 20px;
  margin: 0 0 20px 0;
  border-radius: 10px;
}
```

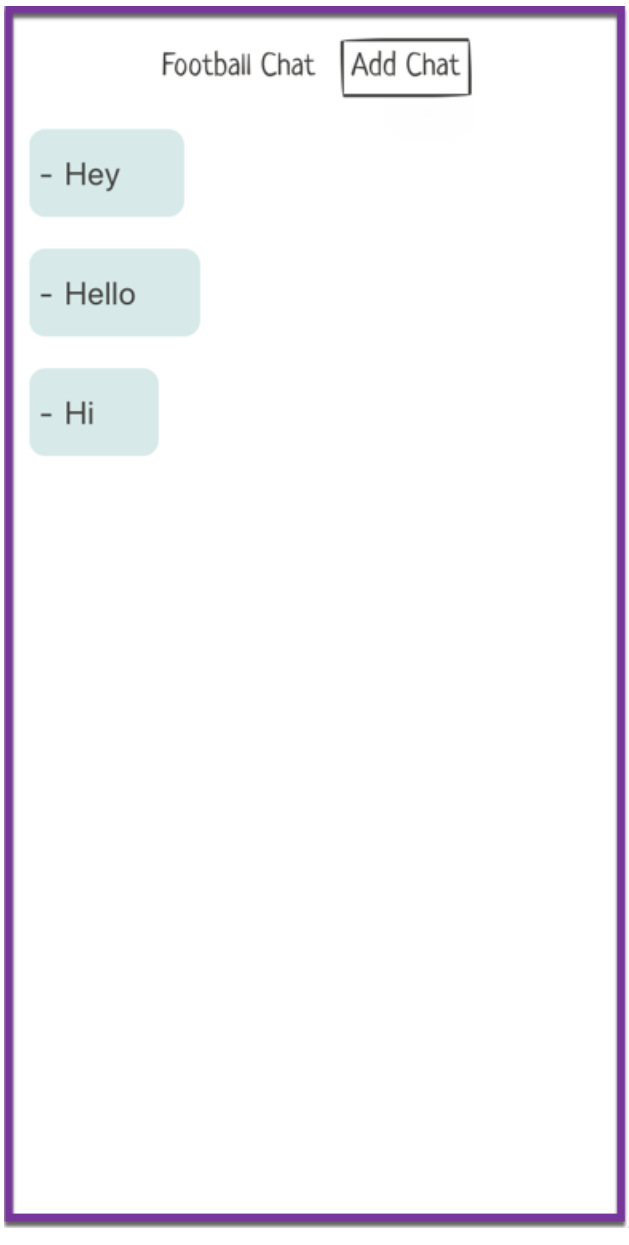
```
.chat-bubble:nth-child(n) {
  margin-right: auto;
}

.chat-btn {
  padding: 5px;
  margin: 0 0 0 10px;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Let me explain what's going on here.

Below is the chat window:



The full chat window

The graphic below highlights the actual region that holds the chat messages the unordered list, `ul` which houses the messages.

The actual region holding the chat messages

It is this `ul` we hold a reference to when using a React ref:

```
<ul className="chat-thread" ref={this.chatThreadRef}>
   ...
</ul>
```

First off, because `getSnapshotBeforeUpdate` may be triggered for updates via any number of props or even a state update, we wrap the logic in a conditional that checks if there is indeed a new chat message:

```
getSnapshotBeforeUpdate(prevProps, prevState) {
    if (this.state.chatList > prevState.chatList) {
      // write logic here
    }
}
```

The `getSnapshotBeforeUpdate` method above still has to return a value. If no chat message was added, we will just return `null`:

```
getSnapshotBeforeUpdate(prevProps, prevState) {
    if (this.state.chatList > prevState.chatList) {
      // write logic here
    }
    return null
}
```

Now, consider the full code for the `getSnapshotBeforeUpdate` method:

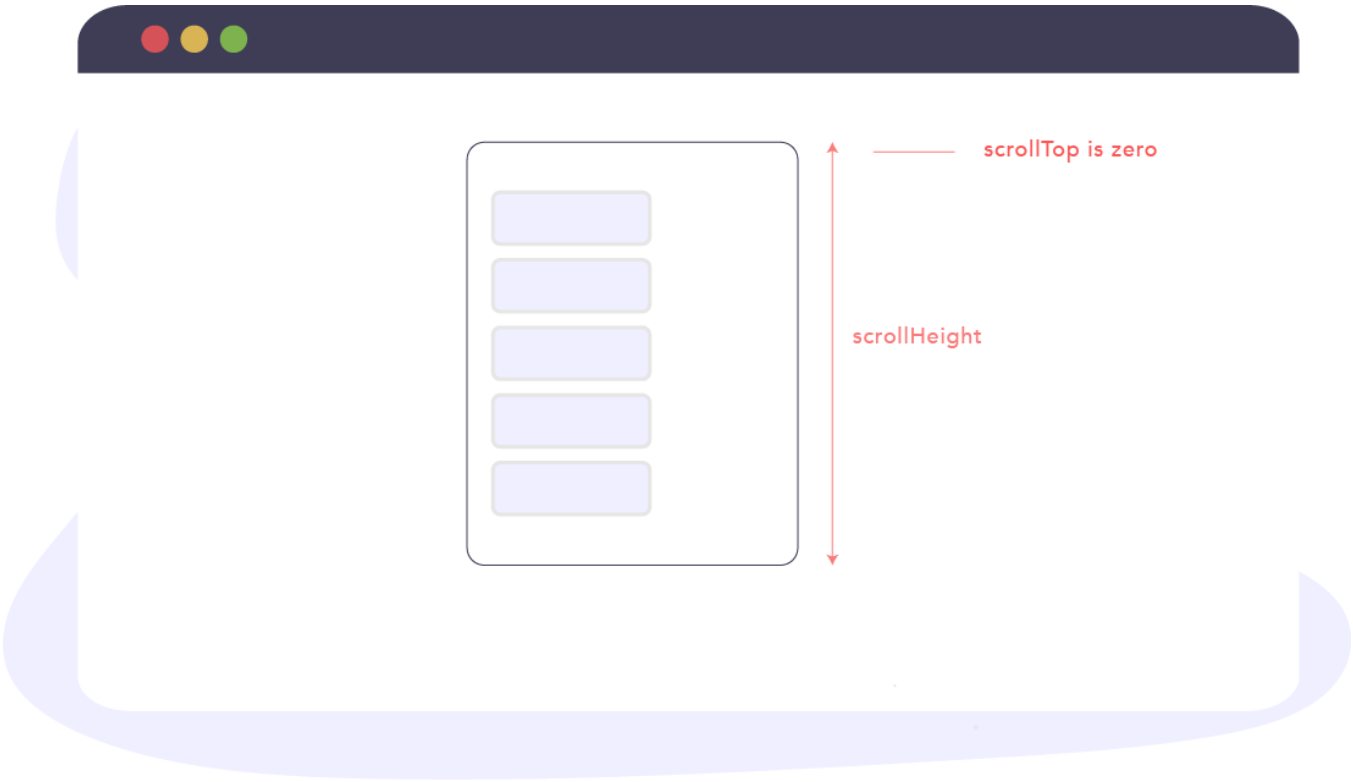
```
getSnapshotBeforeUpdate(prevProps, prevState) {
    if (this.state.chatList > prevState.chatList) {
      const chatThreadRef = this.chatThreadRef.current;
      return chatThreadRef.scrollHeight - chatThreadRef.scrollTop;
    }
    return null;
  }
```

Does it make sense to you?

Not yet, I suppose.

Chat Panel Height

First, consider a situation where the entire height of all chat messages doesn't exceed the height of the chat panel.



Here, the expression `chatThreadRef.scrollHeight - chatThreadRef.scrollTop` will be equivalent to `chatThreadRef.scrollHeight - 0`.

When this is evaluated, it'll be equal to the `scrollHeight` of the chat panel—just before the new message is inserted to the **DOM**.

If you remember from the previous explanation, the value returned from the `getSnapshotBeforeUpdate` method is passed as the third argument to the `componentDidUpdate` method. We call this `snapshot`:

```
componentDidUpdate(prevProps, prevState, snapshot) {
}
```

The value passed in at this time is the previous `scrollHeight` before the update to the **DOM**.

Revising the `componentDidUpdate` Method

In the `componentDidUpdate` method we have the following code, but what does it do?

```
componentDidUpdate(prevProps, prevState, snapshot) {
```

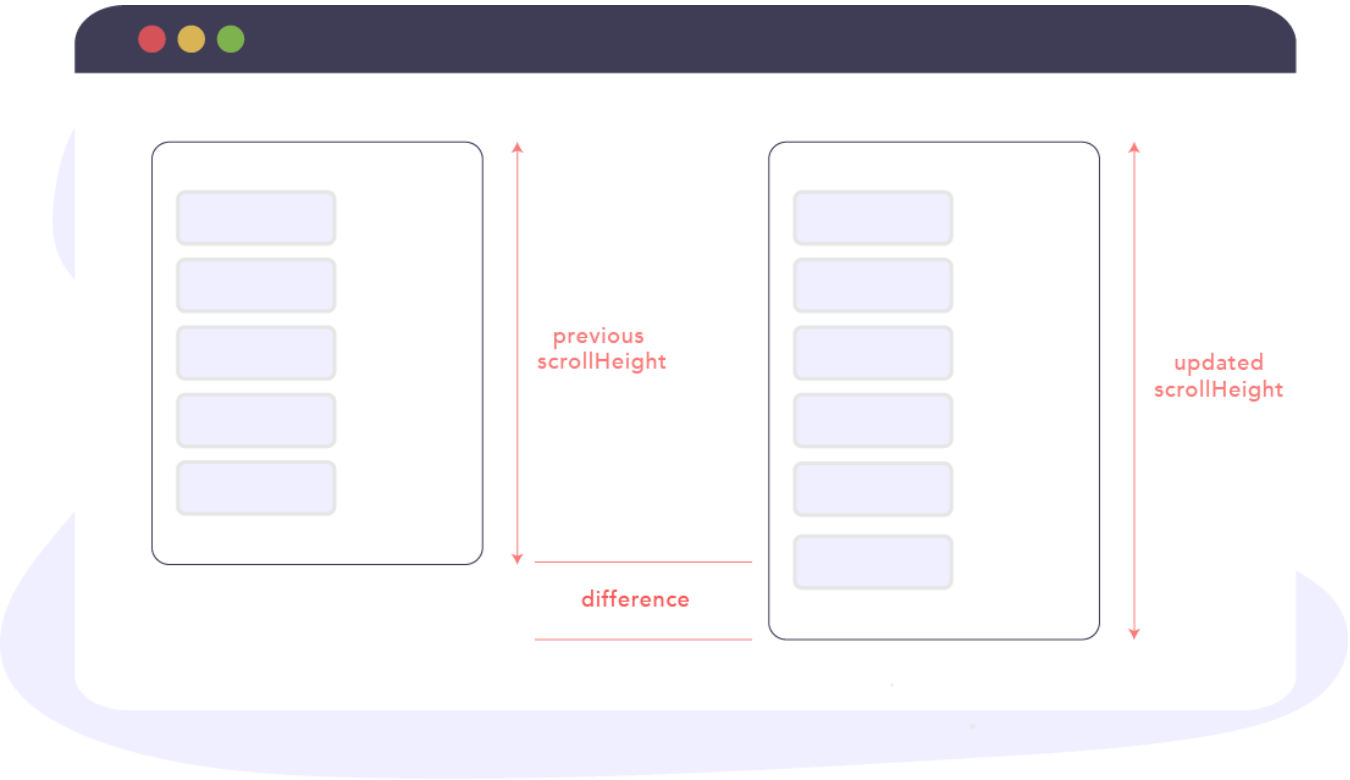
```
    if (snapshot !== null) {
      const chatThreadRef = this.chatThreadRef.current;
      chatThreadRef.scrollTop = chatThreadRef.scrollHeight - snapshot;

    }
  }
```

In actuality, we are programmatically scrolling the panel vertically from the top down by a distance equal to `chatThreadRef.scrollHeight - snapshot`;

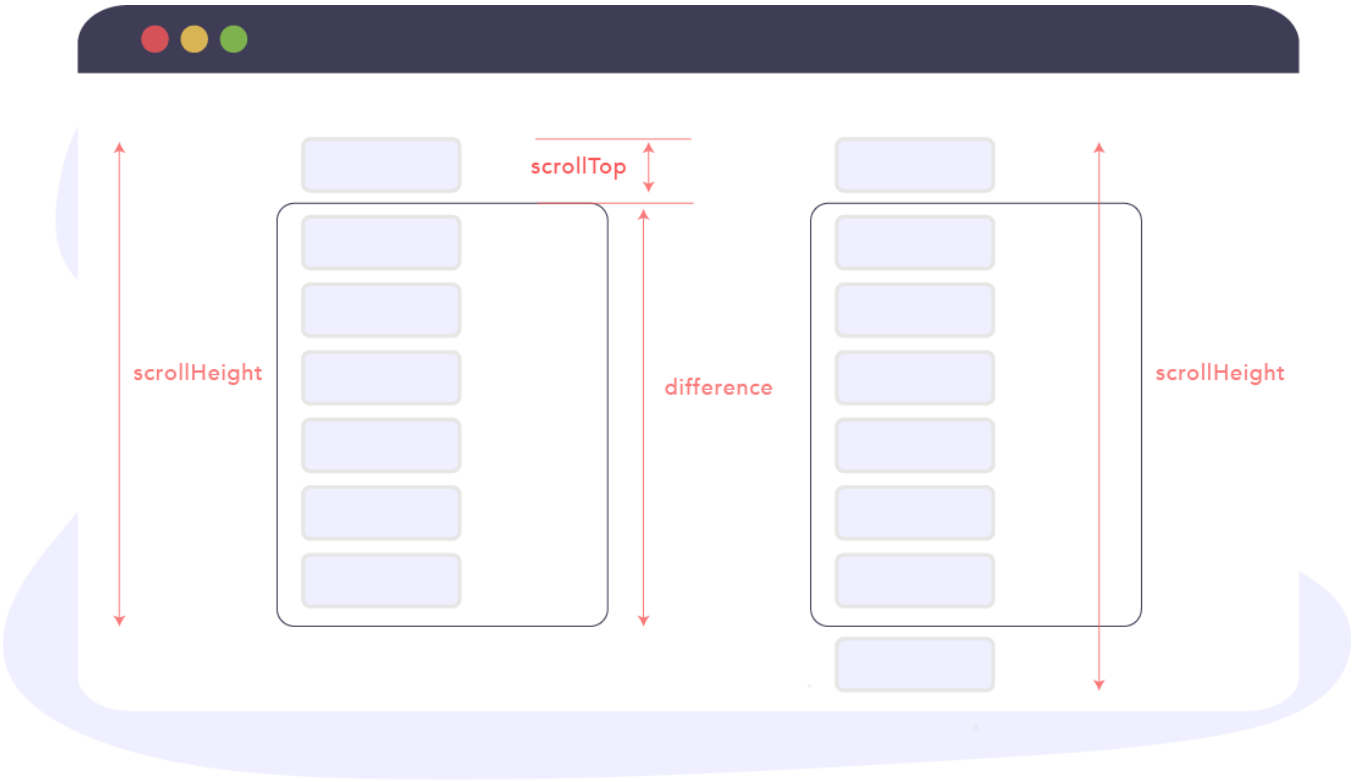
Since `snapshot` refers to the `scrollHeight` before the update, the above expression returns the `height` of the new chat message plus any other related height owing to the update.

Please see the graphic below:



When the entire chat panel height is occupied with messages and the window is already scrolled up a bit, the `snapshot` value returned by the `getSnapshotBeforeUpdate` method will be equal to the actual height of the chat panel.

The computation from `componentDidUpdate` will set the `scrollTop` value to the sum of the heights of extra messages which is exactly what we want.



Yeah, that's it.

If you got stuck, I'm sure going through the explanation one more time or checking the source code will help clarify your questions.

Let's look at the implementation of the running project:

```
.App {
  text-align: center;
  display: flex;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 40vmin;
}

.App-header {
  border: 1px solid #282c34;
  flex: 1;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
}

.App-header img {
  border: 0
```

```
}

.App-link {
  color: #61dafb;
}

.App-chat {
  min-width: 400px;
  /* background: linear-gradient(
      -45deg,
      #183850 0,
      #183850 25%,
      #192c46 50%,
      #22254c 75%,
      #22254c 100%
    )
    no-repeat; */
}

/**
  chat
**/
.chat-thread {
  padding: 0 20px 0 20px;
  list-style: none;
  display: flex;
  flex-direction: column;
  overflow-y: scroll;
  max-height: calc(100vh - 50px);
}

.chat-bubble {
  position: relative;
  background-color: rgba(25, 147, 147, 0.2);
  padding: 16px 40px 16px 20px;
  margin: 0 0 20px 0;
  border-radius: 10px;
}

.chat-bubble:nth-child(n) {
  margin-right: auto;
}

.chat-btn {
  padding: 5px;
  margin: 0 0 0 10px;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

In the next lesson, we'll add a component that handles errors when descendant components throw any exceptions.

descendant components throw any exceptions.