Method 2: getSnapshotBeforeUpdate

Let's discuss the "getSnapshotBeforeUpdate" method which stores the previous version of the application before uploading the changes.

WE'LL COVER THE FOLLOWING

- Understanding the Mechanism
- Example: Chat Panel
- Implementation of Chat Panel
- Functionality of the `Add Chat` Button
- Issue in the Scrolling
- Solution of Scrolling Issue

In the updating component phase, the `getSnapshotBeforeUpdate` lifecycle method is called right after the `render` method.

Understanding the Mechanism

This one is a little tricky, but I'll take my time explaining how it works.

Chances are you won't always reach for this lifecycle method, but it may come in handy in certain cases. Specifically, when you need to grab some information from the **DOM** (and potentially change it) just after an update is made.

Here's the important thing. The value queried from the **DOM** in `getSnapshotBeforeUpdate` will refer to the value just before the **DOM** is updated, even though the render method was previously called.

An analogy that may help has to do with how you use version control systems such as git.

A basic example is that you write code and stage your changes before pushing to the repo.

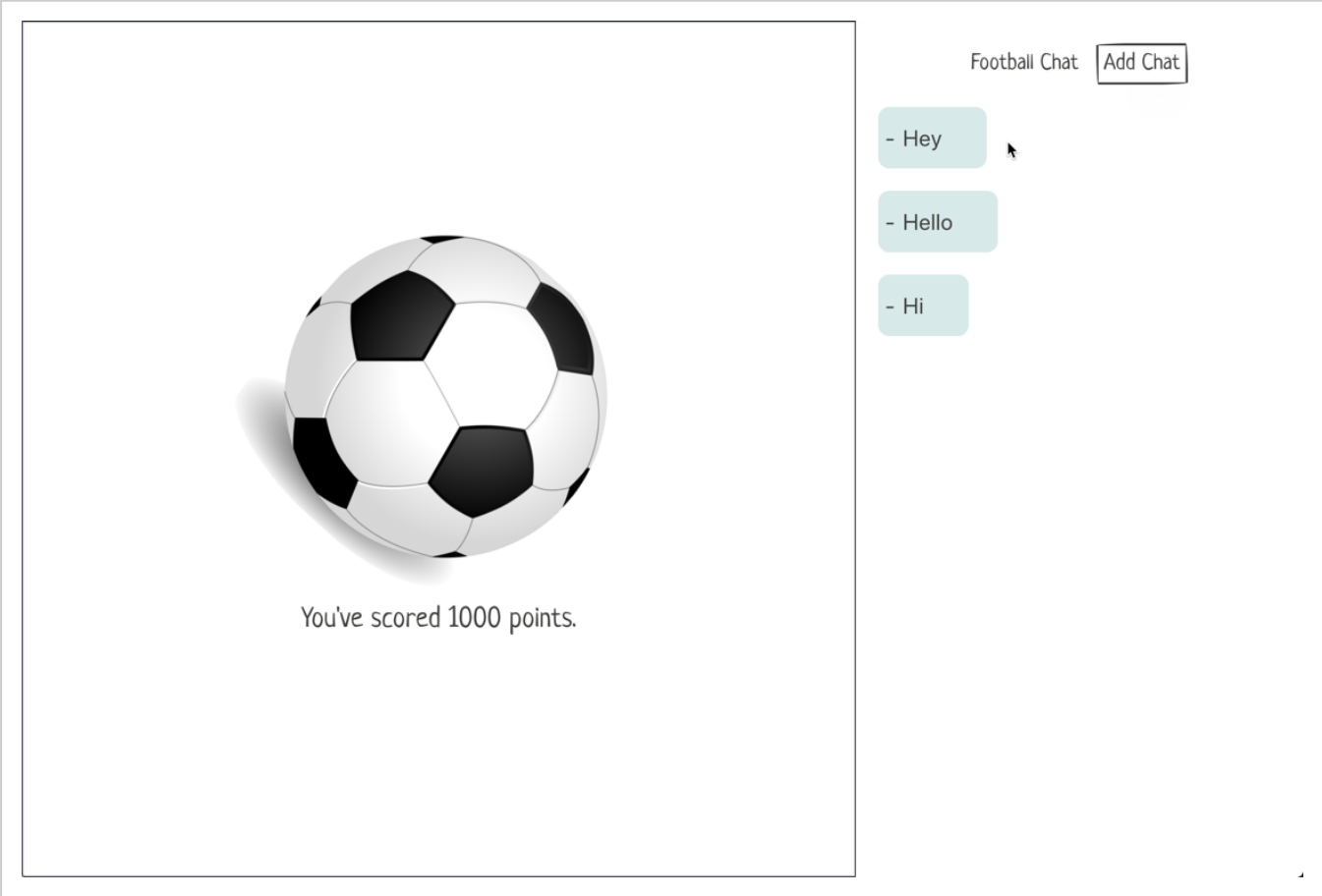
to the repo.

In this case, assume the render function was called to stage your changes before actually pushing to the **DOM**. Before the actual **DOM** update, information retrieved from `getSnapshotBeforeUpdate` refers to those before the actual visual **DOM** update.

Actual updates to the **DOM** may be asynchronous, but the `getSnapshotBeforeUpdate` lifecycle method will always be called immediately before the DOM is updated.

Example: Chat Panel

Don't worry if you don't get it yet. I have another example for you.



Implementation of Chat Panel

The implementation of the chat panel is as simple as you may have imagined. Within the `App` component is an unordered list with a `Chats` component:

```
<ul className="chat-thread">
    <Chats chatList={this.state.chatList} />
 </ul>
```

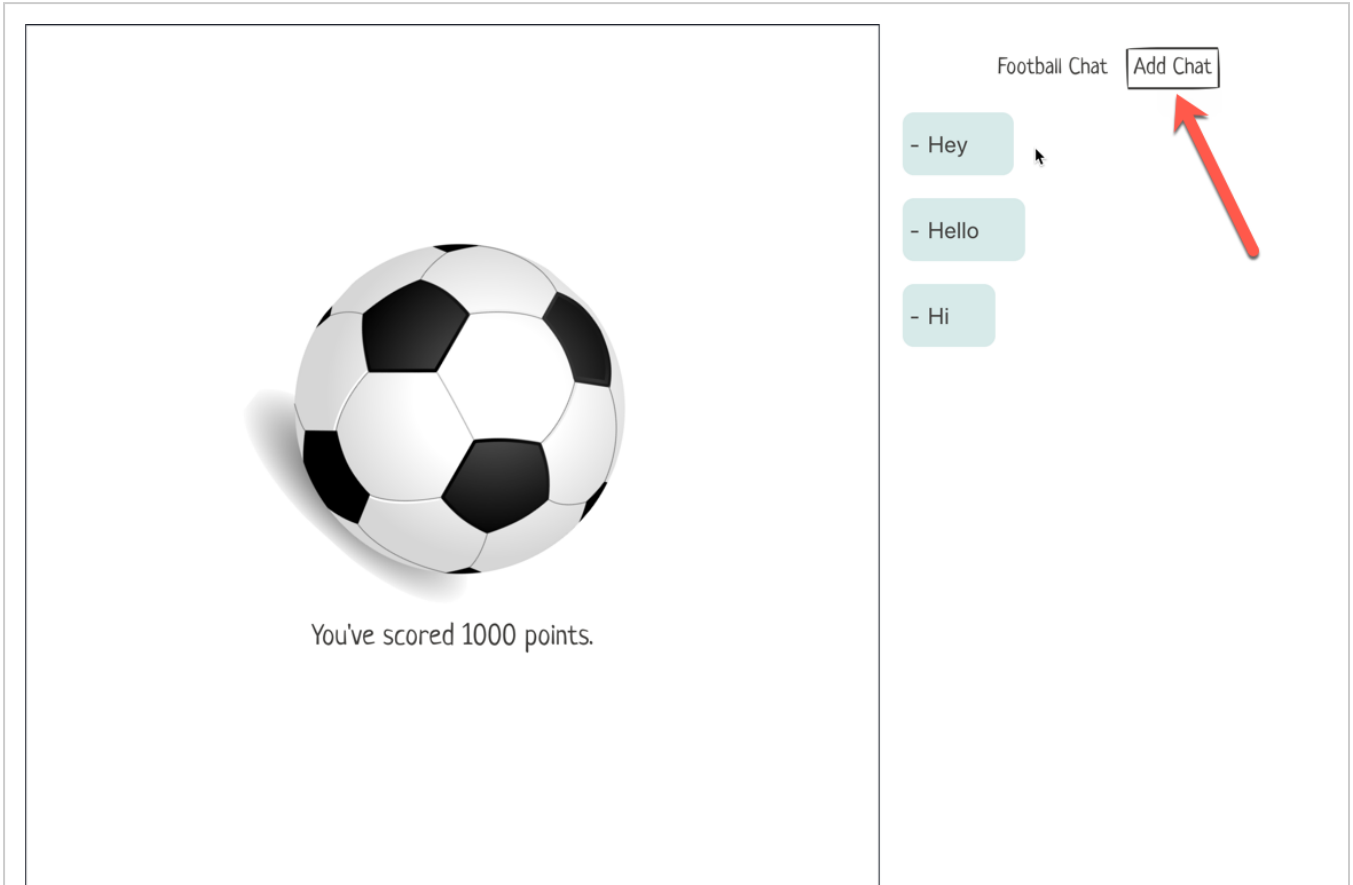
The `Chats` component renders the list of chats, and for this, it needs a `chatList` prop. This is basically an array. In this case, an array of 3 string values: **Hey**, **Hello**, **Hi**.

The `Chats` component has a simple implementation as follows:

```
class Chats extends Component {
  render() {
    return (
      <React.Fragment>
        {this.props.chatList.map((chat, i) => (
          <li key={i} className="chat-bubble">
            {chat}
          </li>
        ))}
      </React.Fragment>
    );
  }
}
```

It maps through the `chatList` prop and renders a list item which is, in turn, styled to look like a chat bubble.

There's one more thing though. Within the chat pane, the header is an **Add Chat** button.



Functionality of the `Add Chat` Button

Clicking this button will add a new chat text, “Hello”, to the list of rendered messages.

Here’s that in action:

Adding new chat messages

Issue in the Scrolling

The problem here, as with most chat applications, is that whenever the number of chat messages exceeds the available height of the chat window, the expected behavior is to auto-scroll down the chat panel so that the latest chat message is visible. That’s not the case now.

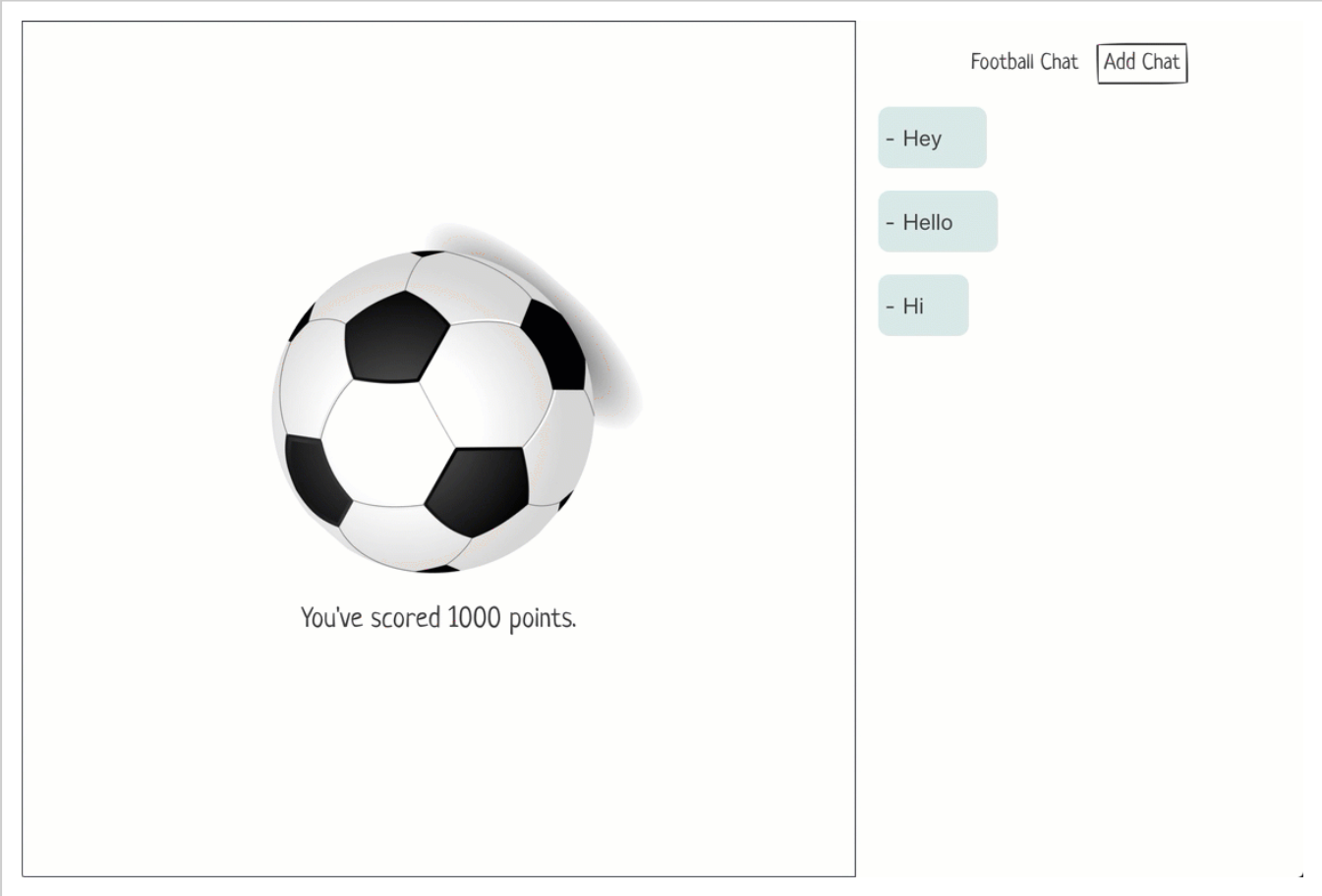
Let’s look at the implementation of the component up until now:

```
.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);
  }
}
```

You can see the output accordingly in the given gif:



I have to scroll manually to find the most recent message

Solution of Scrolling Issue

Let's see how we could solve this using the `getSnapshotBeforeUpdate` lifecycle method.

The way the `getSnapshotBeforeUpdate` lifecycle method works is that when it is invoked, it gets passed the previous props and state as arguments.

So, we can use the `prevProps` and `prevState` parameters as shown below:

```
getSnapshotBeforeUpdate(prevProps, prevState) {

}
```

Within this method, you're expected to either return a value or `null`:

```
getSnapshotBeforeUpdate(prevProps, prevState) {
   return value || null // where 'value' is a  valid JavaScript value
}
```

Whatever value is returned here is then passed on to another lifecycle method. You'll see what I mean soon.

In the next lesson, we'll discuss the `getSnapshotBeforeUpdate` component in more detail.