PROJECT

Passing Thoughts

What if you could post something and know it wouldn't live forever? In this project, we'll build a place for our passing thoughts. Once you add a short thought, it'll disappear after just 15 seconds.

Let's get started!

Tasks

13/13 Complete

Mark the tasks as complete by checking them off

Adding Thoughts

1.

Run the app and take a look at what we have so far.

The app's structure is there, but…well, it doesn't actually work! There are three major missing pieces right now:

1. If you try to add a new thought, it refreshes the whole page and doesn't do anything.
2. If you try to delete a thought manually, it crashes!
3. Thoughts never disappear, defeating the whole point of the app.

Before we start, let's talk about the main piece of state that the app stores: an array of thought objects. Each of those objects will have three properties:

- `id`, a unique ID for this thought
- `text`, the thought's text
- `expiresAt`, the time that the thought expires, represented as a number

Take a look at **App.js** and you'll see that we're defining two starter thoughts like this, on lines 8-19:

```
const [thoughts, setThoughts] = useState([
  {
    id: generateId(),
    text: 'This is a place for your passing thoughts.',
    expiresAt: getNewExpirationTime(),
  },
  {
    id: generateId(),
    text: "They'll be removed after 15 seconds.",
    expiresAt: getNewExpirationTime(),
  },
]);
```
That will set the state to something like this:

```
[
  {
    id: 0,
    text: 'This is a place for your passing thoughts.',
    expiresAt: 1600624968405,
  },
  {
    id: 1,
    text: "They'll be removed after 15 seconds.",
    expiresAt: 1600624968405,
  },
]
```
When we create new thought objects, we'll do something very similar to this. Let's get started!

2.

Open the `<App>` component in **App.js**. You can see that we've already set up the array of thoughts in a variable called `thoughts`, using the `useState()` hook. We've also got `setThoughts()`, a function we can call to update the list of thoughts. Let's wire these things up so that users can add new thoughts.

First, we'll need to write a new function inside of `App()`, called `addThought()`. It will take a single argument, `thought`, and it will put itself inside of the thoughts array with `setThoughts()`.

Inside of `addThought()`, we'll call `setThoughts()` with a function that returns a new state: the array with the new `thought` at the front.

Add this function between the end of the `useState()` call (the line ending with `]);`) and the `return`.

Hint

If you were writing a different app and wanted to add a friend to a list of friends, you might do something like this:

```
const addFriend = (friend) => {
  setFriends((friend) => [friend, ...friends]);
};
```

Find the `useState()` call. After it ends (on the line ending with `]);`), you'll add your function.

3.

Now we have a function that'll update the state, but we need to use it. Specifically, let's pass it to the `<AddThoughtForm>` component as a prop.

In **App.js**, find where `<AddThoughtForm>` is rendered. Pass a prop named `addThought` and give our newly created `addThought()` function as its value.

Hint

In **App.js**, find where `<AddThoughtForm>` is rendered. It's currently rendered without props, but that will change.

Remember that you could pass a prop called `foo` with a value of `bar` like this:

```
<AddThoughtForm foo={bar} />
```

4.

Now that we're passing `addThought()` into the `<AddThoughtForm>` component, it's time for us to make that component actually call this function.

`<AddThoughtForm>` will have some state. Specifically, it'll hold the value of the text input, and when the user submits the form, we'll take that input and use it in a call to `addThought()`.

Open up **AddThoughtForm.js**.

On the first line, we'll need to import `useState()` and React. Update the import to to import both `React` and `useState()`.

Next, inside of `AddThoughtForm`, set up the initial state of the text input as an empty string.

After you have done that, you'll need to write a function called `handleTextChange()` that will be called when the input changes. It will take the event as an argument, and will call `setText()` to update the state.

Finally, you'll need to connect these two things to the input. Pass `text` in a prop called `value`, and pass `handleTextChange()` in a prop called `onChange`.

Now we're storing the state of the input!
Hint

On the first line, we'll need to import `useState()` and React. Update the import to look like this:

```
import React, { useState } from 'react';
```

Next, add this as the first line of `AddThoughtForm`:

```
const [text, setText] = useState('');
```

Next, in the first line of `AddThoughtForm`, set up the state over the input with a line like this:

```
const handleTextChange = (event) => {
  setText(event.target.value);
};
```

Finally, find where the `<input>` is rendered and add the following two props to it:

```
value={text}
onChange={handleTextChange}
```

5.

Now that we have the input's state stored, we will need to create a new thought object when the form is submitted. Handling the form submission event is the first step of doing that.

Create a new function to handle the form submission called `handleSubmit()`. It will take the event as an argument (just like how `handleTextChange()` does).

To prevent the form from refreshing the page, call `event.preventDefault()` inside `handleSubmit()`. This prevents the browser from performing its default behavior when a form is submitted.

Finally, add the submit handler to the form by passing `onSubmit={handleSubmit}`.

Once you've done this, the form should no longer refresh the page. (Nothing else will happen either, but we'll fix that soon.)
Hint

Under `handleTextChange()`, you'll add a new event handler called `handleSubmit()`. It will look like this:

```
const handleSubmit = (event) => {
  event.preventDefault();
};
```
Lastly, make sure that your `<form>` element calls `handleSubmit()` function when its `onSubmit` event is triggered:

```
<form className="AddThoughtForm" onSubmit={handleSubmit}>
```
6.

Now, we'll update `handleSubmit()` to…well, actually submit the data!

In **utilities.js** you will see that there are two functions: `generateId()` and `getNewExpirationTime()`. We will use these functions to get the values for the unique ID and the expiration time for new thought objects.

Back in **AddThoughtForm.js**, inside of `handleSubmit()` after the call to `event.preventDefault()`, create a new thought object with its three required properties: `id` (generated by `generateId()`), `text` (from prior steps), and `expiresAt` (generated by `getNewExpirationTime()`). Pass it to `addThought()`.

If you're not sure how to create a thought object, refer to how it's done in **App.js**.

Once this is done, you should be able to submit the form and see the new thought appear on the screen!
Hint

Inside of `handleSubmit()` after the call to `event.preventDefault()`, create a new thought object:

```
const thought = {
  id: generateId(),
  text: text,
  expiresAt:                          getNewExpirationTime(),
};
```
Then, pass this object to `addThought()`.
7.

Though we are creating new thoughts, you might notice a piece of the user experience that feels a little unintuitive: the input isn’t cleared when you submit the form. That means that whatever you typed stays around, even though it probably shouldn’t.

Clear the input’s text after adding a new thought.
Hint

To clear the input’s text, try calling `setText('')`.
8.

There’s just one thing left to do here: if the user hasn’t typed anything but they submit the form anyway, an empty thought will be created. We all have empty thoughts from time to time, but we probably don’t want to add those to our app.

To fix this, only call `addThought()` if the user hasn’t typed anything yet. You’ll use an `if` statement to check the length of the `text` variable before creating and adding a new thought object. (Make sure to always call `event.preventDefault()`, though, even if the user hasn’t typed anything.)
Hint

You’ll use an `if` statement that looks like this:

```
if           (text.length           >           0)           {
  //              Your              code              here
}
```
Manually Deleting Thoughts
9.

The app should feature two ways to delete thoughts:

1. Manual deletion, when the user clicks the delete button.
2. Automatic deletion after 15 seconds.

Let’s start with the first task. It will help us build the scaffolding for the second.

Open **App.js**.

Just like we added a function to add new thoughts, we will need to create a function to remove them, too.

Under `addThought()`, create a new function called `removeThought()`. It will take the ID of the thought we want to remove in an argument called `thoughtIdToRemove`, and it will call `setThoughts()` to remove the thought.

To do this, you'll call `thoughts.filter()` to filter out the thought we want to remove.
Hint

Your `removeThought()` function should look something like the following code:

```
const removeThought = (thoughtIdToRemove) => {
  setThoughts((thoughts) =>
    thoughts.filter((thought) => thought.id !==
thoughtIdToRemove)
  );
};
```
10.

Once this is done, all you'll need to do is pass this new function as a prop to the `<Thought>` component. Add a new prop called `removeThought` with the newly created `removeThought()` function as its value.

If you've done this successfully, you should now be able to manually remove thoughts by clicking the delete button next to each thought.
Hint

Find where `<Thought>` is rendered and add a new prop: `removeThought={removeThought}`.
Letting Thoughts Drift Away
11.

The app is getting close, but we're still missing the core feature: making thoughts disappear.

When a `<Thought>` component is rendered, we want to start a countdown. Once the countdown expires, we want to call `removeThought()`. We'll do this with an effect hook.

Open **Thought.js**.

Start by importing `useEffect()` from React.
Hint

Open **Thought.js** and find the line where React is imported. Like you did before, import `useEffect()` like this:

```
import React, { useEffect } from 'react';
```

12.

Let’s start by setting a timer in the `<Thought>` component with `setTimeout()`.

Using the `useEffect()` hook, call `setTimeout()`. The first argument should be a function that calls `alert('Time has passed!')` (or any sample text you want—we’ll remove it later). It should happen when the thought expires. You might calculate that like this:

```
const timeRemaining = thought.expiresAt - Date.now();
```

Make sure you remember two things:

1. Return a function that clears the timeout when you’re done! It’s always good practice to clean up your effects.
2. Add `[thought]` as a dependency, in the second argument to `useEffect()`. You want to re-run the effect every time the `thought` is different.

Once this is done, you should see “Time has passed!” alerts after 15 seconds.
Hint

Inside of the `<Thought>` component, your `useEffect()` hook should look something like the following code:

```
useEffect(() => {
  const timeRemaining = thought.expiresAt - Date.now();
  const timeout = setTimeout(() => {
    alert('Time has passed!');
  }, timeRemaining);
  return () => {
    clearTimeout(timeout);
  };
}, [thought]);
```

13.

There’s just one thing left to do: replace that `alert()` with a call to `removeThought()`. This should already be passed in as `props`, so you’ll just need to delete the `alert()` line and replace it with a call to `removeThought()`.

You’ll need to call `removeThought()` with this thought’s ID, which you can get with `thought.id`.

Once that’s done, try adding some thoughts and watch them disappear…it’s just like your brain, but you built it with React.
Hint

Replace the call to `alert()` with a call to `removeThought()`. It should look something like this:

```
removeThought(thought.id);
```

App.js

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom';
import { AddThoughtForm } from './AddThoughtForm';
import { Thought } from './Thought';
import { generateId, getNewExpirationTime } from './utilitie
s';

function App() {
  const [thoughts, setThoughts] = useState([
    {
      id: generateId(),
      text: 'This is a place for your passing thoughts.',
      expiresAt: getNewExpirationTime(),
    },
    {
      id: generateId(),
      text: "They'll be removed after 15 seconds.",
      expiresAt: getNewExpirationTime(),
    },
  ]);
  const addThought = (thought) => {
    setThoughts(prev => [thought, ...prev]);
  }

  const removeThought = (thoughtId) => {
    setThoughts((prev) => prev.filter(item => item.id !== th
oughtId));
  };

  return (
    <div className="App">
      <header>
        <h1>Passing Thoughts</h1>
      </header>
      <main>
        <AddThoughtForm
          addThought={addThought}
        />
        <ul className="thoughts">
          {thoughts.map((thought) => (
            <Thought
              removeThought={removeThought}
              key={thought.id}
              thought={thought}
            />
          ))}
        </ul>
      </main>
    </div>
  );
}

ReactDOM.render(<App />, document.getElementById('app'));
```