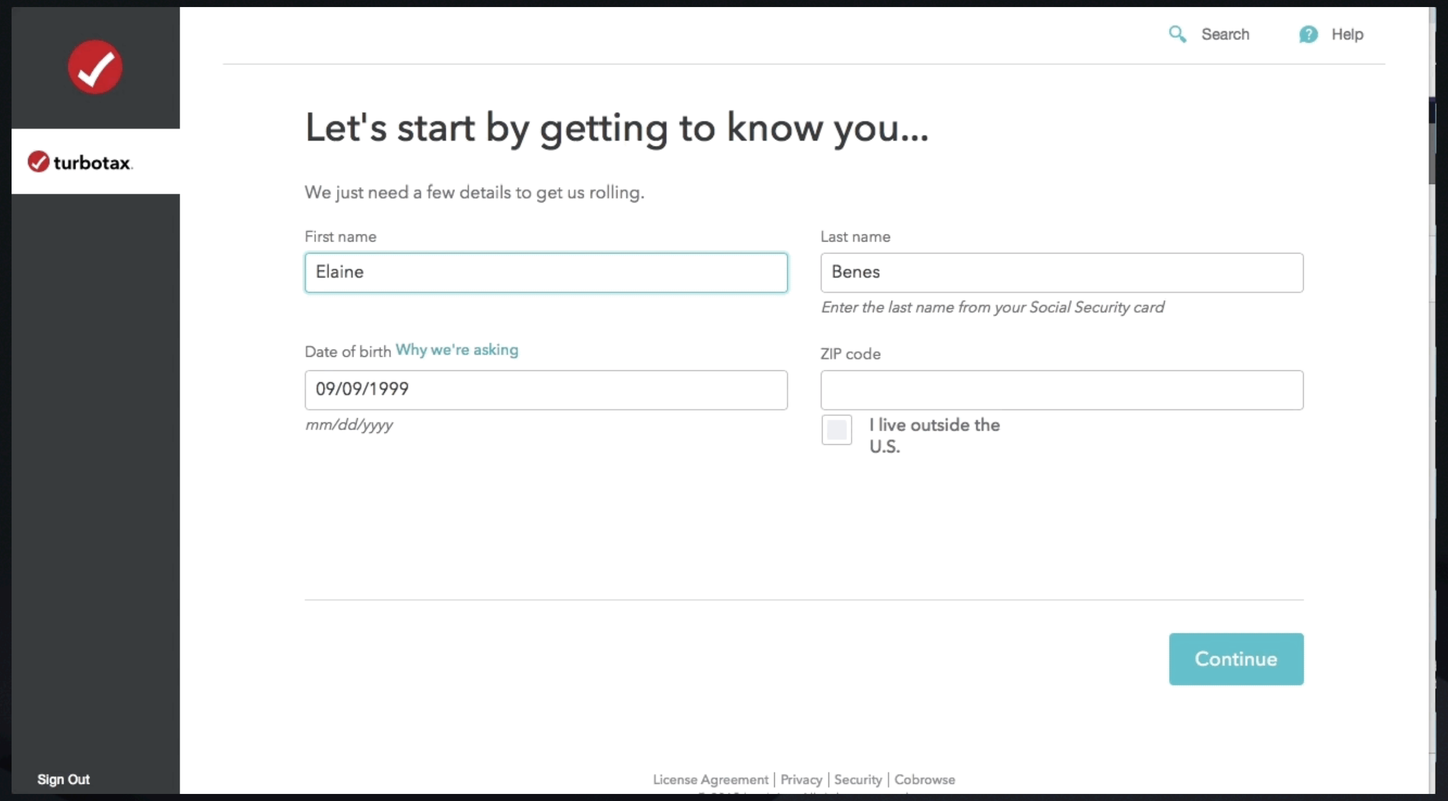
# React II

Forms

# Forms


Search
Help
turbotax
Let's start by getting to know you...
We just need a few details to get us rolling.
First name
Elaine
Last name
Benes
Enter the last name from your Social Security card
Date of birth Why we're asking
09/09/1999
mm/dd/yyyy
ZIP code
I live outside the U.S.
Continue
Sign Out
License Agreement | Privacy | Security | Cobrowse

# Forms

```
return (
  <>
    <h1>New Post</h1>
    <form>
      <input type="text" name="title" placeholder="Title" />
      <button type="submit">Create Post</button>
    </form>
  </>
);
```

- form tag is built for interactivity and submitting information
- Groups together elements within, to be submitted all at once
- A button with type="submit" is special! It means: when I click this, fire a "submit" event on the entire form

# Forms

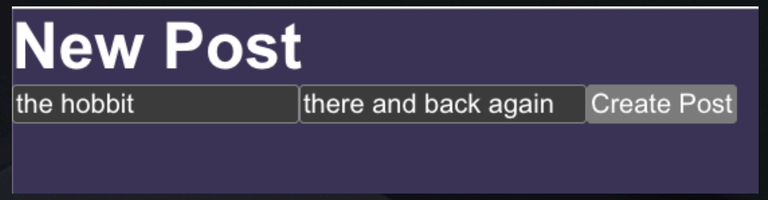
- By naming our inputs, we can access them on the event.target when the form is submitted!

```
function NewPost() {

  const handleSubmit = (e) => {
    console.log(e.target.title.value);
    console.log(e.target.content.value);
  };

  return (
    <>
      <h1>New Post</h1>
      <form onSubmit={handleSubmit}>
        <input type="text" name="title" />
        <input type="text" name="content" />
        <button type="submit">Create Post</button>
      </form>
    </>
  );
}
```

the hobbit

there and back again

localhost:3000/posts/new?title=the+hobbit&content=there+and+back+again



# Forms

```
function NewPost() {

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log(e.target.title.value);
    console.log(e.target.content.value);
  };

  return (
    <>
      <h1>New Post</h1>
      <form onSubmit={handleSubmit}>
        <input type="text" name="title" />
        <input type="text" name="content" />
        <button type="submit">Create Post</button>
      </form>
    </>
  );
}
```

- By default, a form will encode the inputs into the URL, submit the form, and refresh the page.
  - We don't want that! React doesn't like being refreshed
- We can call `event.preventDefault()` to avoid this

# Forms

```
const [title, setTitle] = useState("");
const [content, setContent] = useState("");

const handleTitleChange = (e) => {
  setTitle(e.target.value);
};
const handleContentChange = (e) => {
  setContent(e.target.value);
};

const handleSubmit = (e) => {…};

return (
  <>
    <h1>New Post</h1>
    <form onSubmit={handleSubmit}>
      <input type="text" name="title" onChange={handleTitleChange} />
      <input type="text" name="content" onChange={handleContentChange} />
      <button type="submit">Create Post</button>
    </form>
  </>
);
```

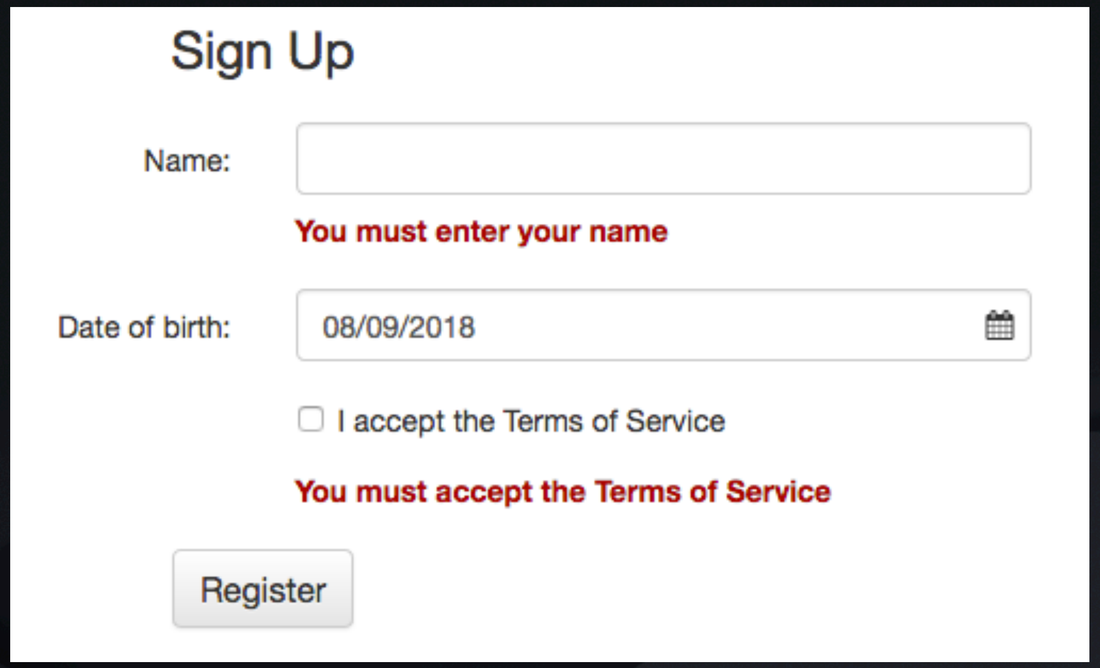
- Instead of relying on event.target, we usually want to keep track of the input in state
  - e.g. calculate fullName from firstName and lastName
- Each time the input changes, we update the corresponding state

# Forms

```
const handleClear = () => {
  setTitle("");
  setContent("");
};

const handleSubmit = (e) => {…};

return (
  <>
    <h1>New Post</h1>
    <form onSubmit={handleSubmit}>
      <input type="text" name="title" onChange={handleTitleChange} value={title} />
      <input type="text" name="content" onChange={handleContentChange} value={content} />
      <button type="button" onClick={handleClear}>Clear</button>
      <button type="submit">Create Post</button>
    </form>
  </>
);
```

- If we update the state on every change, how about we also set the value of each input from state?
- We can then modify the form *programmatically*
  - e.g. clearing the form

# Form Validations



# Form Validations

- The “required” key forces the user to input something into the field – they can’t submit the form until they do

```
const handlePasswordChange = (e) => {
  setPassword(e.target.value);
};

return (
  <>
    <h1>New User</h1>
    <form onSubmit={handleSubmit}>
      <input required type="text" onChange={handleUsernameChange} value={username} />
      <input required type="text" onChange={handlePasswordChange} value={password} />
      <button type="submit">Create User</button>
    </form>
  </>
);
```

# Form Validations

- We could keep track of the errors in an array, updating it as the user types...

```
const [passwordErrors, setPasswordErrors] = useState([]);

const handlePasswordChange = (e) => {
  setPassword(e.target.value);
  if (e.target.value.length < 8) {
    setPasswordErrors(["Password must be at least 8 characters long"]);
  } else {
    setPasswordErrors([]);
  }
};
```

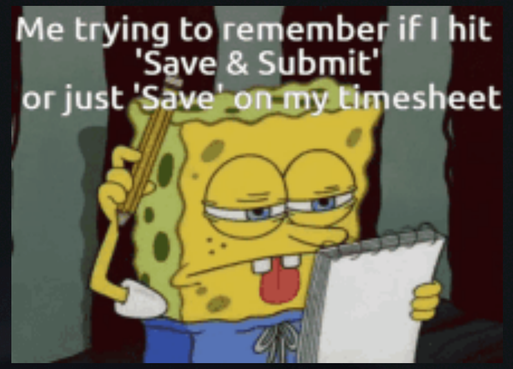
# Form Validations

- And then disable the submit button if there are any errors
- We can also render those errors (if there are any)
  - Take a look at that && syntax

```
{passwordErrors.length > 0 && (
  <>
    {passwordErrors.map((error) => (
      <p key={error}>{error}</p>
    ))}
  </>
)}
<button disabled={passwordErrors.length > 0} type="submit">
  Create User
</button>
</form>
```

# Forms Recap

1. <form> groups together the inputs within
2. type="submit" is special – it submits the form!
3. event.preventDefault() is your friend
4. onChange={handleChange} updates state when the input changes
5. value={someValue} lets you control the input programmatically
6. "required" and "disabled" keys let you enforce validations
7. Keep track of validation errors within the handleChange



# Form Libraries

TANSTACK FORM

tanstack.com/form

React Hook Form

Performant, flexible and extensible forms with easy-to-use validation.

react-hook-form.com

FORMIK

formik.org

# Forms Practice