

Input and Validation

Mendel Rosenblum

Early web app input: HTTP form tag

```
<form action="/product/update" method="post">  
  Product: <input type="text" name="product"/><br />  
  Deluxe: <input type="checkbox" name="delux" /><br />  
  <input type="submit" value="Submit"/>  
</form>
```

- method="get" - Encode form properties as query params
HTTP GET product/update?product=foobar&delux=on
- method="post" - Encode form properties as query params in message body
HTTP POST product/update
Content-Type: application/x-www-form-urlencoded
product=foobar&delux=on

Rails input pattern using form POST

- GET Page containing form
 - Contains a method="post" form to a POST Page
- POST Page - Validate and perform operation (typically create or update)
 - If successful, redirect to a "done "page (possibly another GET Page) if successful
 - If failed validation, redirect page to the GET Page with incorrect fields highlighted
 - If error, redirect to some oops page

Validation requirements in web applications

- Protect integrity of storage (required fields, organization, security, etc.)
 - Can not let HTTP request either from web app or generated out the web app damage us
 - Need to enforce at web server API
- Provide a good user experience
 - Don't let users make mistakes or warn them as soon as possible
 - Pushing validation closer to the user is helpful
- Validation in JavaScript frameworks (ReactJS)
 - Rule #1: Still need server-side validation to protect storage system integrity
 - Rule #2: Let user know about validity problems as early as possible

Input in ReactJS: familiar HTML form/input model

- ReactJS uses HTML form elements: `<input>`, `<textarea>`, and `<select>`

```
render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Name: <input type="text" value={this.state.inputValue} onChange={this.handleChangeInput} />
      </label>
      <label>
        Essay: <textarea value={this.state.textValue} onChange={this.handleChangeText} />
      </label>
      <label>
        <select value={this.state.selValue} onChange={this.handleChangeSelect}>
          <option value="yes">Yes</option>
          <option value="no">No</option>
          <option value="maybe">Maybe</option>
        </select>
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
```

Input in ReactJS handling events

```
class MyForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value}); // Common approach to push into component state
  }

  handleSubmit(event) {
    // Process submit from this.state
    event.preventDefault(); // Need to stop DOM from generating a POST
  }
}
```

JSX and `this` handling - No ideal way

- Specifying a method as DOM event callback doesn't work:

```
<form onSubmit={this.formSubmit}> ... // Wrong! Calls with this undefined
```

- Arrow function embedded in JSX render: Can call instance method

```
<form onSubmit={event => this.formSubmit(event)}> ...
```

- Redefine method function in instance to have correct `this` in constructor:

```
this.formSubmit = this.formSubmit.bind(this); // In component constructor
```

- Use new JavaScript class **fields**: `class Foo { fieldName = value;`

Validation in ReactJS

- Unopinionated! Lots of different packages
 - Example: [Formik](#) - Build forms in React, without tears.
 - Handles specifying form, validation methods, form error reporting, etc.
- Flexible: Can do validation anyway you want

```
handleChange(event) {  
  if (this.validateIt(event.target.value, this.state) {  
    this.setState({renderValidationError: true});  
  }  
  this.setState({value: event.target.value});  
}
```

Arbitrary JavaScript can look at `event.target.value` and `this.state` and use `setState` causing `render()` be called again.

Asynchronous validation

- Can in background communicate with web server to validate input
 - Example: username already taken
- Example: Autocomplete with [React-AutoSuggest](#)

```
<Autosuggest
  suggestions={suggestions}
  onSuggestionsFetchRequested={this.onSuggestionsFetchRequested}
  onSuggestionsClearRequested={this.onSuggestionsClearRequested}
  getSuggestionValue={getSuggestionValue}
  renderSuggestion={renderSuggestion}
  inputProps={inputProps}
/>
```

- Trend towards using recommendation systems for input guidance

Single Page App Input

- Rather than POST with redirect you can do a XMLHttpRequest POST/PUT
- React: Unopinionated - Many options to choose from.
 - Example: [Axios](#) - Promise based HTTP client for the browser and node.js

```
axios.get(url)
```

```
axios.delete(url)
```

```
axios.post(url, body)
```

```
axios.put(url, body)
```

Axios Model fetch

```
axios.get(URLpath)
  .then((response) => {
    // response.status - HTTP response status (eg 200)
    // response.statusText - HTTP response status text (eg OK)
    // response.data - Response body object (JSON parsed)
  })
  .catch((err) => {
    // err.response.{status, data, headers} - Non-2xx status HTTP response
    // if !err.response - No reply, can look at err.request
  });
```

Axios Model fetch - Alternative Error Handling

```
axios.get(URLpath)
  .then((response) => {
    // response.status - HTTP response status (eg 200)
    // response.statusText - HTTP response status text (eg OK)
    // response.data - Response body object (JSON parsed)
  },
  (err) => {
    // err.response.{status, data, headers} - Non-2xx status HTTP response
    // if !err.response - No reply, can look at err.request
  });
```

Axios Model uploading

```
axios.post(URLpath, objectWithParameters)
  .then((response) => {
    // response.status - HTTP response status (eg 200)
    // response.statusText - HTTP response status text (eg OK)
    // response.data - Response body object (JSON parsed)
  })
  .catch((err) => {
    // err.response.{status, data, headers} - Non-2xx status HTTP response
    // if !err.response - No reply, can look at err.request
  });
```

Server-side validation

- Regardless of validation in browser server needs to check everything
 - Easy to directly access server API bypassing all browser validation checks
- Mongoose allows validator functions

```
var userSchema = new Schema({
  phone: { type: String,
    validate: {
      validator: function(v) {
        return /d{3}-d{3}-d{4}/.test(v);
      },
      message: '{VALUE} is not a valid phone number!'
    }
  }
});
```

Some integrity enforcement requires special code

- Maintaining relationship between objects
- Resource quotas
- Examples related to our Photo App
 - Only author and admin user can delete a photo comment.
 - A user can only upload 50 photos unless they have a premium account.