Refactoring React

Patterns and practices

Who am I

Inga Pflaumer

Data analyst and Deployment specialist at SMG studio

Organiser of Sydney Unity Game Dev meetup

and Women In Game Development meetup

Corgi owner

Workshop

- https://github.com/Rukia3d/IntroToJS-Marvel/
- clone project
- branches:
  - `master` - initial state
  - `fixed` - final state
  - `api` - API implementation if you want to see it

The Project

- Marvel Universe API
- The team selection part of a fighting game
- Created with TDD
- Needs Refactoring

The Project

- Created with create-react-app
- Runs on localhost:3000
- You will need node.js
- if you didn’t do npm install - do it now

The Project

- You need React Dev Tools in your browser:
  - install React Developer Tools for Chrome
  - restart the browser

The Project

- Open browser
- Open terminal and project folder in the terminal (2 windows)
- **npm run start** to start the project
- **npm run test** to run tests

Use case

- User searches for their favourite characters
- Adds them to the team
- Original characters can be highlighted
- Characters can be deleted
- When the team is formed - game starts

The Project (API)

- originally had API requests
- replaced by captain.json for this workshop
- will read captain.json for any search
- API calls and vars are commented out
- you will need your own API key if you want to try it

The Project

- **App.js** Entry point. Creates Characters and Search
- **Search.js** Search form and Search results - Item
- **Characters.js** Displays NoCharacters or ShowCharacters
- **ShowCharacters.js** Displays and paginates Characters

Refactoring

- Do you have tests?
- If no - WRITE TESTS FIRST
- If yes - check the coverage
- If it’s not good enough - write more tests

App.test.js

- No characters selected
- Enters “Captain”. Search results are in
- Adds Captain America. No pages. Pic + Details are in.
- Adds 5 more Captains. Pagination test.
- Deletion test.

App.test.js

- What’s missing?
- No toggle test.
- Decision: Ignore it (bad decision!)
- You need to know what you decided to ignore.
- And how it can affect the result.

General Refactoring

- Separate classes
- Rename variables (use let and const!)
- Make functions more readable
- Add PropTypes
- etc…

React Refactoring

- Anonymous functions

- When do we re-render?

Anonymous functions

- Will create a copy for every render.
- Not a problem on a small project.
- What if you’re making a Stadium seat booking system? With a function generater for every seat on every render?

ShowCharacters.js

ShowCharacters. | App.js | Search.js | *package.json* | App.css | App.test.js | captain.json | Characters.js | Welcome

```
3
4   const Character = (props) => (
5     <div className="card text-white bg-primary mb-3" style={{maxWidth: "20rem"}} data-testid="character">
6       <div className="card-body">
7         <h4 className="card-title" data-testid="name">{props.character.name}</h4>
8           <img
9             style={{maxWidth: "18rem"}}
10            data-testid="picture"
11            alt={props.character.name}
12              src={props.character.thumbnail.path+"."+props.character.thumbnail.extension}
13            />
14          <p className="card-text" data-testid="descr">{props.character.description}</p>
15          <button
16            className="btn btn-outline-secondary"
17            data-testid="deleteButton"
18            onClick={() => props.removeCharacter(props.character.id)}
19          >
20            Delete
21          </button>
22        </div>
23      </div>
24    );
```

ShowCharacters.js

- Anonymous function because it has a parameter.
- OnClick (event)
- Data can be transferred via data- attribute
- Rewrite removeCharacter function in App.js to take an event, find character id from the event and delete the character

ShowCharacters.js

Old

```
    <button
      className="btn btn-outline-secondary"
      data-testid="deleteButton"
      onClick={() => props.removeCharacter(props.character.id)}
    >
      Delete
    </button>
  </div>
```

New

```
  <button
    className="btn btn-outline-secondary"
    data-testid="deleteButton"
    data-id={props.character.id}
    onClick={props.removeCharacter}
  >
    Delete
  </button>
```

App.js

Old

```
removeCharacter(c){
  const newCharacters = this.state.characters.filter( char => char.id !== c);

  this.setState({
    characters: newCharacters
  });
}
```

New

```
removeCharacter(event){
  let c = parseInt(event.target.dataset.id);
  const newCharacters = this.state.characters.filter( char => char.id !== c);
  this.setState({
    characters: newCharacters
  });
}
```

Congrats!

- Check your test!
- The easiest way to fix anonymous functions is to pass data via events.
- Be specific. Only transfer data you need, not whole objects.

The data you need…

```
removeResult(r){
  const newResult = this.state.results.filter( res => res.id !== r);
  this.setState({
    results: newResult
  });
}

update(c){
  this.removeResult(c.id);
  this.props.add(c);
}

results(){
  return this.state.results.map(r =>
    <Item highlights={this.highlights()}
      r={r}
      key={r.id}
      onClick={() => this.update(r)}
    />)
}
```

Search.js -> update

```
update(c){
  this.removeResult(c.id);
  this.props.add(c);
}
```

- Receives character object
- Gives id to Search.js removeResult
- Gives the object to App.js add
- We can find character object by ID in our Search props
- OnClick!

Search.js -> update

Old

```
update(c){
  this.removeResult(c.id);
  this.props.add(c);
}
```

New

```
update(event){
  const charID = parseInt(event.target.dataset.id);
  this.removeResult(charID);

  const character = this.state.results.find(c => c.id === charID);
  this.props.add(character);
}
```

Your test will fail as we do not have data- attribute yet

Search.js

- Know what element your dataset must be attached to
- Easiest way to attach it to event.target
- Unless you want to search for it in the tree

Search.js

- Update() function is given to Item
- It’s called as onClick in Item
- Button that has onClick needs to have data-id with character id in it
- Also name r is a very bad name for a prop
- Rename it!

Search.js -> results()

Old

```
83      results(){
84        return this.state.results.map(r =>
85          <Item
86            highlights={this.highlights()}
87            r={r}
88            key={r.id}
89            onClick={() => this.update(r)}
90        />)
91      }
```

New

```
107     results(){
108       return this.state.results.map(c =>
109         <Item
110           highlights={this.highlights()}
111           charData={c}
112           key={c.id}
113           onClick={this.update}
114       />
115       )
116     }
```

Search.js -> Item

Old

```
<button
  data-testid="addBtn"
  className="btn btn-primary btn-sm"
  onClick={() => props.onClick(props.r)}
>
  Add
</button>
```

New

```
<button
  data-testid="addBtn"
  className="btn btn-primary btn-sm"
  data-id={props.charData.id}
  onClick={onClick}
>
  Add
</button>
```

Search.js -> Item

- A lot of messy code
- Unreadable
- Not optimised. Why?

Search.js -> Item

- Use your React Dev Tools in the browser.
- You will see the Items are re-rendered each time.
- We removed anonymous function. But it still happens.
- How can we fix it?

Search.js -> Item

- React will re-render component when parent re-renders
- React will re-render Item for several reasons:
    - highlights() returns new array on every render
    - React.Component doesn’t have shouldComponentUpdate

Search.js -> Item

- How to fix this bad bad component.
    - Readability:
        - Make it a class component
        - Move highlights into a separate function
        - Assign class if Item needs to be highlighted
        - Destructure its props

Item: smart

```
class Item extends Component{
  render(){
    return(
      <li data-testid="result" data-name={this.props.charData.name}
        className={"list-group-item d-flex justify-content-between align-items-center" +
          (this.props.highlights.some(str => this.props.r.name.includes(str))
          ? ' highlighted'
          : '')}
        >
        <span data-testid="res-name">{this.props.charData.name}</span>
        <button
          data-testid="addBtn"
          className="btn btn-primary btn-sm"
          onClick={this.props.onClick}
        >
          Add
        </button>
      </li>
    )
  }
}
```

Item: highlighted

```
8     class Item extends Component{
9       highlighted(){
10        return this.props.highlights.some(str => this.props.r.name.includes(str));
11      }
12
13      render(){
14        return(
15          <li data-testid="result" data-name={this.props.charData.name}
16            className={"list-group-item d-flex justify-content-between align-items-center" +
17              this.highlighted() ? ' highlighted': ''}
18            >
19            <span data-testid="res-name">{this.props.charData.name}</span>
20            <button
21              data-testid="addBtn"
22              className="btn btn-primary btn-sm"
23              onClick={this.props.onClick}
24            >
25              Add
26            </button>
27          </li>
28        )
29      }
30    }
```

Item: css separated

```
8   class Item extends Component{
9     highlighted(){
10      return this.props.highlights.some(str => this.props.r.name.includes(str));
11    }
12
13    render(){
14      const highlightClass = this.highlighted() ? 'highlighted' : '';
15      return(
16        <li data-testid="result" data-name={this.props.charData.name}
17          className={`${highlightClass} list-group-item d-flex justify-content-between align-items-center`}>
18          <span data-testid="res-name">{this.props.charData.name}</span>
19          <button
20            data-testid="addBtn"
21            className="btn btn-primary btn-sm"
22            onClick={this.props.onClick}
23          >
24            Add
25          </button>
26        </li>
27      )
28    }
29  }
```

Item: destructuring props

```
8   class Item extends Component {
9     highlighted(){
10      return this.props.highlights.some(str => this.props.charData.name.includes(str));
11    }
12
13    render(){
14      const highlightClass = this.highlighted() ? 'highlighted' : '';
15      const { charData, onClick } = this.props;
16
17      return (
18        <li data-testid="result" data-name={charData.name}
19          className={`${highlightClass} list-group-item d-flex justify-content-between align-items-center`}>
20          <span data-testid="res-name">{charData.name}</span>
21          <button
22            data-testid="addBtn"
23            className="btn btn-primary btn-sm"
24            data-id={charData.id}
25            onClick={onClick}
26          >
27            Add
28          </button>
29        </li>
30      )
31    }
32  }
```

Search.js -> Item

- How to fix this bad bad component. Part 2
- Optimisation:
    - Make sure highlights arrays are re-used
    - Control re-rendering

Search.js

- Instead of creating new highlights array on each call, save them into const
- It will guarantee new arrays are not created every time
- Also easier to edit when client requirements change

Search.js -> highlights()

Old

```
highlights() {
  if (this.state.highlight) {
    return ['(', ')'];
  } else {
    return [];
  }
}
```

New

```
const highlightOriginal = [];
const highlightBrackets = ['(', ')'];

highlights() {
  if (this.state.highlight) {
    return highlightBrackets;
  } else {
    return highlightOriginal;
  }
}
```

Search.js -> Item

Controllable re-rendering - React.PureComponent

PureComponent changes the life-cycle method `shouldComponentUpdate` and adds some logic to automatically check whether a re-render is required for the component. This allows a PureComponent to call method `render` only if it detects changes in `state` or `props`, hence, one can change the state in many components without having to write extra checks like:

https://60devs.com/pure-component-in-react.html

Search.js -> Item

Controllable re-rendering - React.PureComponent

Old

```
class Item extends Component {
```

New

```
class Item extends React.PureComponent {
```

What else?

- To improve readability:
    - Change variable names
    - Split logic into smaller functions
    - Use destructuring to make props readable
    - Use Prop.Types

What else?

- To optimise:
  - Check your anonymous functions. Use events.
  - Give references instead of objects where possible.
  - Re-render only when you need to.

Summary

- Readability is important.
- Optimisation is important.
- Whatever the focus is - HAVE TESTS.

Thank you!

Find me here:

- ➤ Twitter: @IngaPflaumer
- ➤ Medium: @IngaPflaumer
- ➤ LinkedIn: Inga Pflaumer

If you’re looking for a job, talk to me 8)