International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 04 | Apr 2022 www.irjet.net p-ISSN: 2395-0072

ReactCodemod: An automated approach for refactoring class based components to Hooks

Shubham Ghule[1], Nikita Aware[2], Nishant Pandey[3], Kiran Mahajan[4], Prof. Anant Bagade[5]

[1,2,3,4]*Dept. Information Technology, SCTR'S Pune Institute of Computer Technology, Pune, Maharashtra, India*
[5]*Head of Information Technology, Dept. of SCTR'S Pune Institute of Computer Technology, Pune, Maharashtra, India*

Abstract - *React is a popular front-end framework for creating online interfaces. With React version 16.8, Facebook has introduced concepts of Hooks and started promoting it as an efficient alternative to class based components. The current method to convert class-based into hooks relies on manual refactoring. This paper proposes reactCodemod an automated solution for refactoring of class-based components to equivalent hooks. ReactCodemod, uses jscodeshift API by Facebook for manipulating of abstract syntax tree. Jscodeshift takes class-based react code as input generates an abstract syntax tree, identifies the required code that needs to be refactored using filters and logic runs over the source code, transforms the code by manipulating the abstract syntax tree. In the end, output code is generated. Output code is equivalent function based code.*

Key Words: React.js, Class-Based Components, Function-Based Components, Hooks, Refactoring, Jscodeshift

1.INTRODUCTION

React is the most widely used open-source, modern JavaScript library for creating the front end of websites, which comprises web pages, layouts, and content. From version 16.8.0 React is gradually moving towards popularly known as hooks. As a result, there will be a lot of classbased code that has to be refactored into hooks. The existing approach relies on manual refactoring of class-based components into equivalent hooks which is cumbersome and time consuming. Hence there is a need of automated solution to tackle this problem. The proposed method, ReactCodemod, takes react code which consists of class-based components and needs to be refactored. First an AST will be created for code by babel/eslint parser after that a transform file which contains the filter and logic that manipulates the AST nodes will run over the source code and will convert the necessary class based code to function based code. ReactCodemod makes use of jscodeshift API by Facebook. The recast and ast-types packages are wrapped in the jscodeshift package. Ast-types manages the low-level interface with the AST nodes, whereas Recast performs the source to AST and back conversion. Jscodeshift enables us to perform complicated transformations over a large codebase in seconds, allowing us to make code changes quickly. Jscodeshift API takes source code file as input and replaces it accordingly. Inside transform source code gets parsed into Abstract Syntax Tree. Babel parser is used for generation of AST. The AST is a tree-based data structure that contains all the variables, calls, and flow control structures in your code. When the application runs, the JavaScript engine iterates through the tree and executes the commands represented by each node. All the transformation work is done on AST and then the altered AST is used to regenerate source code. As opposed to manual refactoring of react class-based code this automated approach ReactCodemod looks promising and offers various advantages. It will save developers time and efforts which are required for manual refactoring [5].

2. LITERATURE SURVEY

Existing class components were refactored into function components, which was thought to have some value and to be necessary in some but not all circumstances. While the developers mentioned the benefits of refactoring, such as increased readability and maintainability, the ability to accommodate new developers unfamiliar with old styles, and more learning resources, their main objection to refactoring components from class components to function components was a lack of time and budget [1].

The traditional technique of refactoring react class-based components to hooks necessitates rewriting the whole codebase, developers with necessary expertise of react and JavaScript, such as components, props, state, context, refs, and top-down data flow, are required. Developers needs to have understanding of a class without state or lifecycle methods.

Over the course of five years of creating and managing tens of thousands of components, hooks have solved a wide range of seemingly unrelated problems in React. The main problems which were solved by release of hooks in React includes Wrapper Hell, classes and side effect. Before the appearance of React hooks, there was no way to reuse the logic of behavior to the component. Stateful logic is difficult to reuse between components. Classes make complex components difficult to understand. React class-based components are easier to maintain at initially, but as the complexity of the class grows, they become

unmanageable. Classes become unreadable as they grow and often store different logic in one place. Hooks let you achieve the same thing by splitting down the logic between components into simple functions that can then be used within the components. Hooks allows extracting stateful logic from a component and allows independent testing and reuse. Reusing stateful logic without changing the hierarchy of its components is made possible by hooks. This makes it easy to share links between many components. Comparing hooks with class-based components of equivalent complexity, hooks provide considerably cleaner, easier-to understand components. This way React Hooks assists us in resolving a variety of issues.

2.1 ReactJS

ReactJS is a JavaScript toolkit for creating reusable user interface components that is declarative, fast, and adaptable [8]. ReactJs is a component-based, open-source front-end library that's just responsible for the application's front-end. A ReactJS application consists of several components, each of which is responsible for producing a tiny, reusable piece of HTML code. All React apps are built around components. These Components may be layered with one other to create sophisticated applications from simple building blocks [7].

2.2 JavaScript

JavaScript is a text-based programming language that allows you to build interactive web pages on both the client and server side. Web browsers and web-based applications are the most common places where JavaScript is used. ReactJS uses JSX which is just a syntactic extension of JavaScript. JavaScript is also used in metaprogramming. Metaprogramming is a programming method that allows computer programs to treat other Programs like data. For the scope of this research jscodeshift toolkit is used for metaprogramming.

2.3 JSX

The abbreviation for JavaScript XML is JSX. It's merely a syntactic extension for JavaScript. JSX allows you to write HTML/XML-like structures in the same file where we write JavaScript code. (e.g., DOM-like tree structures), and the preprocessor will turn these expressions into actual JavaScript code. JSX tags contain a tag name, attributes, and children, much like XML/HTML tags. ReactJS heavily relies on JSX for development, allowing us to build HTML directly in the browser (within JavaScript code).

2.4 JScodeshift

The Jscodeshift toolbox allows you to perform code mods across several JavaScript or TypeScript files. Jscodeshift is a runner that applies the transform provided to each file it receives. Programmers can use the Jscodeshift toolkit to input a batch of source files through a transform and then replace them with the results. Programmers scan the source code into an abstract syntax tree (AST), make modifications, then regenerate the source code from the new AST. It also provides a summary of how many files have been converted (or have not). It's a wrapper for recasting with a new API. Recast is an AST-to-AST conversion tool that attempts to preserve the original code's style as much as possible.

3. METHODOLOGY

Rewriting class-based components involves steps like changing class declarations to function declaration, removing constructors, replacing render with return, adding const keyword before all methods, removing this.state throughout the component and removing all references to 'this' throughout the component. Developers have to spend extra time to do these tasks such as set initial state with useState(), replace compentDidMount with useEffect, replace componentDidUpdate with useEffect. All these tasks are done by developers, which costs time and efforts [2].

3.1 General Identified Steps for Refactoring

1. Import required hooks we will need in the hook based component. Remove the render() function.
2. Remove state object and replace it with useState hook.
3. Add const before all classProperties.
4. Remove this.state throughout the component.
5. Remove all references to 'this' throughout the component.
6. Replace this.setState with newly defined setter function.
7. Replace lifecycle methods with the corresponding hooks.

A codemod is a piece of code that alters original source code. In this proposed method we are developing a codemod named ReactCodemod based on the same identified steps with the help of metaprogramming libraries. Metaprogramming is a programming technique that allows computers to treat other computers as data. It means that a program may be designed to read, produce, generate, or convert other programs as well as modify itself while it is running [3]. There are various libraries available which are helpful for developing codemods. The two popular options are namely comby and jscodeshift. For this codemod we are using jscodeshift because comby is more general library for converting codes from various languages, jscodeshift is more focused towards building codemods. It is a toolkit for running codemods over multiple JavaScript or TypeScript files. Jscodeshift includes a runner that runs the supplied

transform on each file it receives. It also shows the number of files that haven't been updated. A wrapper for recast that exposes an alternative API. Recast is an abstract syntax tree to abstract syntax tree transform tool that tries to preserve as much of the current code's style as possible. Jscodeshift is a toolbox. It combines a variety of parsers, ast types for node construction, and recast for pretty-printing to assure code that follows particular formatting constraints. It neatly bundles these tools and gives a cohesive API for finding, filtering, mapping, and replacing the AST nodes. It also contains a runner feature that allows thousands of files to be transformed at once.

4. SYSTEM ARCHITECTURE

Reactcodemod lets you perform a transform on a set of class based source files and replace them with the results. You parse the source into an abstract syntax tree (AST), change the AST according to the filters, and then build the hook-based source file from the updated AST within the transform.

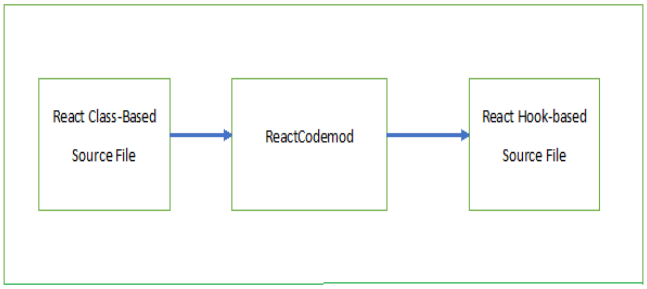

Fig. 1: ReactCodemod Block Diagram

Reactcodemod achieves the transformation of class-based components to hook based components using jscodeshift library and performs following steps:

1. Generate an Abstract Syntax Tree (AST) from given react source file.
2. Traverse the AST and look out for matches.
3. Make the appropriate changes to the AST.
4. Regenerate the source file with function based code.

Jscodeshift is the refactoring tool, created by Facebook. It allows us to run ReactCodemod across different documents. Jscodeshift is a straightforward and simple to-utilize API that is controlled by Recast. Recast is an AST (Abstract Syntax Tree) to AST (Abstract Syntax Tree) refactoring tool.

4.1 Transform Module

The transform module just exports a function. The fileinfo, api, and options parameters are accepted by the transform module. The location and source of the file to be handled are included in Fileinfo. The source relates to the contents of the file, while the path refers to its location. The transform module's jscodeshift api object has three properties: jscodeshift, stats, and report. Jscodeshift is a jQuery-like AST navigation and transformation API. Stats is a dry-run function that reports the number of times a certain value has been called; this data is important in transformation. The report command prints a string to stdout, which may then be used by other applications. Option stores all the options given to the runner, enabling the user to give the transform more possibilities. The state of the transformation is determined by the value returned from the Transform Module. The transformation was successful if the return value differed from the file provided to it; if the return value was the same as the original file, the transformation failed. If the Transform module returns nothing, the transformation is considered successful.

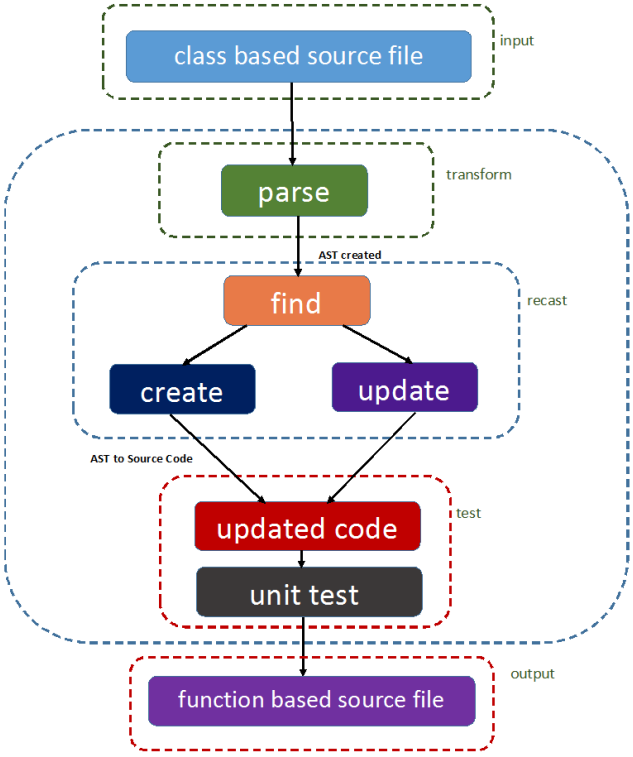

Fig. 1: ReactCodemod Architecture

4.1.1. Parser

Jscodeshift can pick a parser to parse the source file using this transform. Transform module exports name of the parser as a string, which can be "babel", "stream", "Babylon", "tsx", "ts", or some other parser compatible with recast. In the transformation module, several architectural components, and notions of recast and JavaScript are used.

4.2. Abstract Syntax Tree

The given code is converted to an abstract syntax tree by the JavaScript engine. The AST is a tree-based data structure that represents the code's variables, calls, and flow control structure.

4.2.1 AST Nodes

A JavaScript object with a defined set of fields is known as an AST Node. The type of a node is the most common way to identify it, but it also has a number of additional variables that help to identify it.

4.2.2 Path Objects

Every AST Node has a wrapper object that specifies a method to traverse the AST. Paths also contain meta-data and processing assistance methods for AST nodes. To traverse the tree up AST nodes, path objects must have enough information about parent nodes to enable an excellent node iteration and traversal mechanism.

4.2.3 Collections

Jscodeshift is constructed around the idea of collection of paths. a collection has strategies to process the nodes internal a collection that often leads to a new collection. Collections are typed which means method defined for one type of collection will not work on any other type of collection.

4.3 Extensibility

Jscodeshift provides API to extend collections as to make transforms more readable. The two forms of extensions are generic extensions and type specific extensions. Generic extensions do not work with node data and instead traverse the whole collection, whereas Type-specific extensions only work with a particular node type.

5. EXPERIMENTAL RESULTS

5.1 Remove 'this' keyword

- **Input**

```
import React, { Component } from "react";
class Three extends Component {
  state = {
    year: 1995,
    type: "Mercedes",
    used: true,
  };
  render() {
    return (
      <div>
        <ul>
            <li>{this.state.year}</li>
            <li>{this.state.used ? "Used Car" : "Brand
New!"}</li>
        </ul>
      </div>
    );
  }
}
```

- **Transform code**

```
export default function transformer(file, api) {
    const j = api.jscodeshift;
    let code = file.source;
    code = code.replaceAll("this.", "");
    return j(code).toSource();
}
```

- **Output**

```
import React, { Component } from "react";
class Three extends Component {
  state = {
    year: 1995,
    type: "Mercedes",
    used: true,
  };
  render() {
    return (
      <div>
        <ul>
            <li>{state.year}</li>
            <li>{state.used ? "Used Car" : "Brand
New!"}</li>
        </ul>
      </div>
    );
  }
}
```

5.2 Class declaration to function declaration

- **Input**

```
class App extends Component {
    handleClick = () => {
      console.log("Hello World!");
    }
}
```

- **Tranform code**

```
export default function transformer(file, api) {
  const j = api.jscodeshift;
  const root = j(file.source);
  const { statement } = j.template;
  root.find(j.ClassDeclaration).replaceWith((p) =>
{
    return statement`function ${p.value.id.name}
() {
                ${p.value.body.body}
            }`;
  });
  return root.toSource();
}
```

- **Output**

```
function App() {
    handleClick = () => {
      console.log("Hello World!");
    };
}
```

6. CONCLUSION

This paper proposed a method that manipulates the AST of the class-based code written in JSX and converts it into an equivalent hook-based component by manipulating the AST of the code using jscodeshift. As a result of applying proposed method, we found that the AST of the existing class-based component can be redesigned and manipulated

into an equivalent hookbased component. We also found that this method may not work in all cases but will surely reduce the efforts of anyone doing transformations manually which in turn will save time and efforts of the person using this tool. As React is gradually promoting hook-based components, it will be beneficial as it is reducing human efforts and increasing efficiency of work.

ACKNOWLEDGEMENTS

This work was done under the Krixi Ideas and Technology Solutions mentorship program. We would like to express our gratitude towards our mentors Mr Pankaj Lagu,Director Product Strategy and Mr Tejas Joshi at Krixi Ideas and Technology Solutions for their continuous support and encouragement.

REFERENCES

[1] Tuomas Luojus, "Usability and adaptation of react hooks" , April 2021.

[2] Daniel Bugl, "Learn react hooks: build and refactor modern react.js applications using hooks", October 2019

[3] Robertas Damaˇseviˇcius, Vytautasˇ Stuikys, "Taxonomy of the fundamental concepts of metaprogramming" , 2008.

[4] Mohit Thakkar, "Unit Testing Using Jest" , April 2020.

[5] Emerson Murphy -Hill, Chris Parnin, and Andrew P. Black, "How WeRefactor, and How We Know It", February 2012 .

[6] Jaehyun Kim, Yangsun Lee, "A Study on Abstract Syntax Tree forDevelopment of a JavaScript Compiler" , June 2018.

[7] Arshad Javeed, "Performance Optimization Techniques for ReactJS",2019

[8] Sanja Delcev, Drazen Draskovic, "Modern JavaScript frameworks: ASurvey Study", August 201.