جامعة دمشق

كلية الهندسة المعلوماتية

اختصاص هندسة برمجيات ونظم معلومات

React.js Compiler

إعداد:

حازم سلمان الشيباني

سنقوم في بحثنا ب بناء المرحلتين الأولى والثانية من بناء المترجم الخاص ب React.js Library بالإضافة إلى ال Symbol table ومعالجة العديد من الحالات الخاصة باللغة JavaScript.

بعد إتمام المشروع نستطيع من خلاله التعرف على العديد من القواعد والميزات الموجودة في React js Library.

قواعد اللغة:

نستطيع أن نعرف:

- **Variable Declaration:**

باستخدام الكلمة المحجوزة **const** بحيث يتم تعريف كامل المتحول على نفس السطر، بالإضافة إلى أنه يمكننا إسناد قيمة من القيم إلى المتحول المعرف:

1. Number.

   EX: 1, 4, 5.

2. String.

   EX1: "React js is very strong Library".

   EX2: 'React js is very strong Library'.

3. Boolean.

EX1: true, false.

4. Array.

EX1: [1, 2, 3, 4, 5].

EX2: ["a", "b", "c"].

EX3: [true, false].

EX4: [{ }, [ ]].

EX5: [{}, true, 0, "Jad", [1,1,2]].

3. Object.

EX1: {el1: 1, el2: 2, el3: 3}.

EX2: {el1: "a", el2: "b", el3: "c"}.

EX2: {el1: 1, el2: true, el3: [1, 2, 3]}.

يمكننا أن نخلق Array وObject من العديد من القيم وليس قيمة محددة وهذه Dynamic Arrays or objects. أن يكون لدينا JavaScript أحد ميزات لغة البرمجة

- Function Declaration:

•يمكننا من تعريف التابع بطريقتين وهما باستخدام:

3. Arrow Function

```
Ex: const Home = (attribute1, attribute2) => {
        return 5;
    }
```

2. Pure Function.

```
Ex: function Home(attribute1, attribute2) {
        return "I'm a clean code";
    }
```

يمكننا من انشاء function وتمرير قيم من خلاله وارجاع أي قيمة نريدها بالإضافة إلى تعريف variables أو functions ضمن ال body الخاص بال function واستدعاء تابع أخر مثلا:

```
function Home(attribute1, attribute2) {
    const var1 = 100;
    Size(attribute1); return "I'm a
    clean code";
}
```

- Components Declaration:

يمكننا تعريف الComponent بطريقتين:

3. Arrow Function

```
Ex: const Home = (Props) => {
        return (JSX Elements);
    }
```

2. Pure Function.

```
Ex: function Home(Props) {
        return (JSX Elements);
    }
```

ما يميز ال Component عن ال function هو أكواد ال JSX التي تكون مضمنة في قلب ال return statement بالإضافة إلى ال Hooks.

يمكننا من تضمين Props التي تمرر إلى ال component في حال استدعائه كـNested Component وإعطاء قيمة له.

- Hooks Declaration:

ال Hooks وهم :

1. useState Hook:

    Ex:

        const [value, setValue] = useState(0);

يمكننا أن نقوم باعطاء قيمة ابتدائية لل useState ويمكن أن تكون أي من القيم التي تحدثنا عنها سابقاً (Number,Boolean,array...).

2. useRef Hook.

        Const value = useRef();

3. useEffect Hook.

        useEffect(( ) => {

        }, [ ]);

يمكننا أن نقوم بتعريف متحولات و توابع بالإضافة إلى استدعاء تابع محدد ونستطيع تحديد ال dependencies ل ال useEffect Hook.

- **تم التحقق من القاعدتين الأساسيتين لخلق Hook معين وهما:**

1- لا يمكن تعريف ال hook إلا في ال component body.

2- لا يمكن تعريف ال hook في أي بنية داخل ال component body مثل أن يكون useState ضمن ال useEffecrt فهذا مرفوض

- **Tags Declaration:**

يتم تعريف ال tags ضمن ال return statement وهم:

1. h1

EX: <h1> </h1>

2. p

EX:

3. img

EX: <img />

4. div

EX:

5. Nested Components

EX: <Home prop1={value} prop2={value} />

نستطيع إضافة العديد من ال actions على كل من ال tags السابقة مثل:

- onClick={nameOfFunction}
- className={"nameOfClass" }

  className={Style.styleName} "if I use CSS Modules"
- ref={nameOfRef}
- style={{styleName: "styleValue"}}

بالإضافة إلى ال Actions الخاصة بال Img tag:

- src={"pathImgName"} src={Object.nameImg}
- alt="nameOfImg"

- نستطيع تضمين variable ضمن ال tag بالشكل التالي:

<h1>{value}</h1>

- **تم التحقق من القاعدة الأساسيتين لخلق Elements في ال Component وهو :**

يجب أن تكون ال elements ضمن parent element ضمن ال return statement أي أن تكون العناصر مشابهة لشكل الشجرة من أجل عملية الinjunction في ال root التي تتم عند عملية ال loading على موقع معين

أثناء عمل run لأكواد ال JavaScript في ال browser.

- **Import Statement:**

نستطيع تعريف أكثر من import statement في بداية ال file بالشكل:

```
Import {useState, useEffect, useRef} from 'react';
Import App from "./…/path";
Import "./…/path/file.CSS";
```

- **Export Statement:**

نستطيع تعريف نوعين من ال export:

1- export by default: export default App;

فقط لمرة واحدة في ال file.

2- export:

```
export const val = 0;
```

- **Comments:**

  1. One Line Comments:

  // the information I want to comment

  2. Group Line Comments:

  /*

  the information I want to comment

***/**

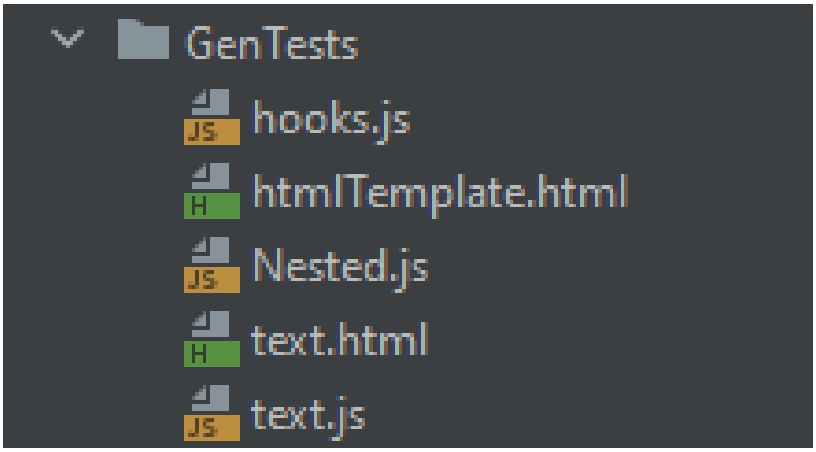
أو ضمن ال return statement ضمن ال component يكون التعليق بالشكل التالي:

{/*

the information I want to comment

***/}**

بنية اللغة الهدف:



يظهر في الصورة التالية تقسيم الملفات الناتجة عن عملية ال Code Generation بحيث:

يتم تحويل أكواد الـReact JS إلى أكواد JS + HTML.

ليتم عمل Run لها في الـBrowser.

text.html

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>React App</title>
    <style></style>
</head>
<body>
    <div id="root"></div>
    <script src="text.js"></script>
</body>
</html>
```

في الصورة أعلاه تظهر أكواد الـHTML ونميز وجود div tag لديه id = root.

أما ملف text.js يوضح تحويل أكواد الـReact JS متضمنةً الJSX و JS و Hooks إلى لغة الJS بناءً على:

- بالنسبة للـVariables والـFunctions يتم تحويلها كما هي لأنها لا تتغير في كلا اللغتين.

- بالنسبة للـJSX وهي عبارة عن tags تعرف في قلب الـ return body

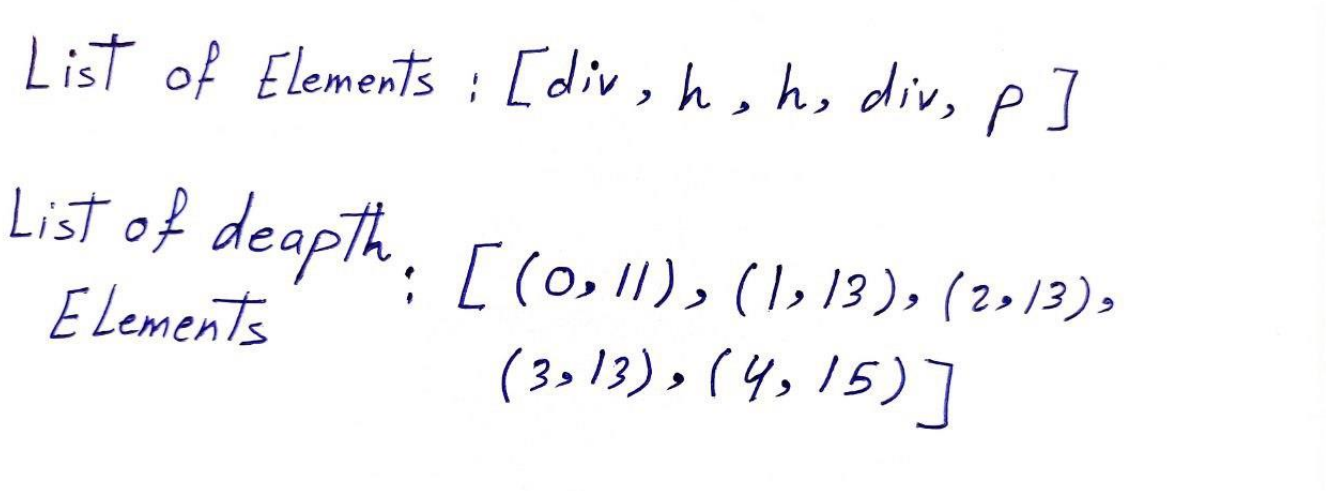
```
    return (
        <div>                                          depth = 11
            <h1>The way of</h1>                        depth = 13
            <h1>Work This code</h1>                    depth = 13
            <div>                                      depth = 13
                <p>Welcome, Here we go again</p>       depth = 15
            </div>
    </div>
    )
}

export default Home
```

للـComponent ويتم تحويلهم بناءً على الخوارزمية التالية:

في الصورة أعلاه نوضح أكواد الـJSX في الـReact JS وطريقة تحويلها إلى JS تبنى على الخطوات التالية:

أولاً: يجب أن نعلم الـdepth of tag وهو يختلف حسب مكان الtag والغاية الأساسية منه هي إسناد كل child tag إلى parent tag الصحيح له.

ثانياً: لتحقيق ذلك تم الاعتماد على الخوارزمية التالية:



بناءً على المثال الوارد سابقاً تم المرور على جميع عناصر الـJSX وتخزين نوعه في المصفوفة الأولى في الصورة السابقة، أما في المصفوفة الثانية تم تخزين depth العنصر.

وبعد المرور على جميع العناصر الموجودة في الـJSX يتم اتباع الخوارزمية كما تم الذكر سابقاً استناداً على الكود التالي:

```
public void appendChild() throws IOException {
    FileWriter updated = new FileWriter(jsFile, append: true);
    for (int i = levelMap.entrySet().size() - 1; i > 0; i--) {
        int currentLevel = levelMap.get(i);
        String element = giveMeTheRightElement(i);
        if(currentLevel == 11) {
            updated.write( str: "div" + 0 + ".appendChild("+ element + i +")" + ";\n");
        } else {
            for(int j= i - 1; j> 0; j--) {
                int moveLevel = levelMap.get(j);
                if(currentLevel > moveLevel) {
                    updated.write( str: "div" + j + ".appendChild("+ element + i +")" + ";\n");
                    break;
                }
            }
        }
    }
```

يتم المرور على المصفوفة بشكل معاكس بحيث نأخذ كل عنصر ثم نقوم بالمرور على جميع العناصر السابقة له حتى نصل إلى العنصر الأول ذو العمق الأقل.

وبناءً على ال id للعنصر ذو العمق الأقل يتم معرفة نوعه تبعاً للمصفوفة الأولى.

ويكون خرج التعليمات السابقة في اللغة الهدف هو الكود بالشكل السابق حيث تمت عملية انشاء لل Elements من ثم تم عمل append للعناصر واسندنا كل child tag إلى parent tag بالشكل الصحيح.

وكما في ال JS بالنسبة لل actions يتم تحوليها بالشكل الصحيح.

```
const Home = () => {
let app = document.createElement( tagName: "div");
app.id = "root";
let div0 = document.createElement( tagName: "div");
let h1 = document.createElement( tagName: "h1");
h1.textContent = "The way of";
let h2 = document.createElement( tagName: "h1");
h2.textContent = "Work This code";
let div3 = document.createElement( tagName: "div");
let p4 = document.createElement( tagName: "p");
p4.textContent = "Welcome Here we go again";
div3.appendChild(p4);
div0.appendChild(div3);
div0.appendChild(h2);
div0.appendChild(h1);
app.appendChild(div0);
document.body.appendChild(app);
}
export default Home;
```

- بالنسبة لل Hooks قمنا بمحاكاة عملها كما في react.js عن طريق ال file الذي يسمى hooks.js والموضح في الصفحة التالية:

```
import Component from "./text.js";
let idx = 0;
const React = (() => {
    let hooks = [];
    function useState(initVal) {
        let state = hooks[idx] || initVal;
        let _idx = idx;
        let setState = (newVal) => {
            hooks[_idx] = newVal;
            reRender();
        };
        idx++;
        return [state, setState];
    }
    function useRef(val) {
        return useState( initVal: { current: val })[0];
    }
    function useEffect(cb, depArray) {
        const oldDeps = hooks[idx];
        let hasChanged = true;
        if (oldDeps) {
            hasChanged = depArray.some((dep, i :number ) => !Object.is(dep, oldDeps[i]));
        }
        if (hasChanged) cb();
        hooks[idx] = depArray;
    }
    return {
        useState,
        useEffect,
        useRef,
    };
})();
function reRender() {
    idx = 0;
    let app = document.getElementById( elementId: "root");
    if (app) {
        app.remove();
    }
    Component();
}
export default { React, reRender };
```

بحيث سيتم عمل البرنامج باللغتين HTML + JS كما في البرامج المبنية باللغة React.js حيث في الملف السابق في كل مرة يتم عمل update لقيمة ال useState او حدوث أي تغير على ال List of dependency في ال useEffect سيتم عمل recreate لل DOM بحيث يأخذ القيمة التي تم عليها التحديث وهذا الامر مطابق تماما لما يحدث في react.js.

معالجة الأخطاء:

تمت معالجة الأخطاء على مستويين وهما:

- Syntax Errors

وهي الأخطاء النحوية التي يتم كشفها من قبل ال Listener استنادا على القواعد اللغوية وعند حدوثها تظهر في البداية قبل الأخطاء المنطقية.

- Semantic Errors

وهي الأخطاء المنطقية التي يتم كشفها في مشروعنا في ال Visitors الذي يمر على ال Parse Tree ويقوم بالتحقق من الأمور التالية قبل اسناد القيم الموجودة إلى ال Symbol table أو تطبيق ال Code generation والاخطاء التي نعالجها هي:

  - في حال تعريف ال Hooks خارج ال component.
  - في حال استدعاء Hook ولم يتم عمل import له من react.

في حال تعريف أكثر من متحول او تابع من نفس الاسم في نفس ال scope. **/تمت معالجة فكرة ال scoping بشكل كامل في التطبيق/**

- في حال استدعاء تابع غير معرف لدينا في الملف.
- في حال عمل export ل component غير معرف لدينا....

❖ يتم طباعة ال Error موضحا ما هو بالإضافة إلى السطر الواقع به.

تم تضمين ال Class Diagram الخاص بال compiler وبنية ال Symbol Table كمرفقات PDF ضمن الملف نظرا لكبر حجمهم.