3SA03 – React

1. เครื่องมือที่จำเป็น

- Chocolatey (for Windows), Brew (for OSX)
- Node.js
- Yarn
- Git
- create-react-app CLI
- Visual Studio Code

หลังจากที่ได้ติดตั้ง Chocolatey ในเครื่องแล้ว สามารถติดตั้ง Node.js, Yarn และ Git ได้ผ่าน Chocolatey ผ่าน Command Prompt (ที่รันด้วยสิทธิ์ Administrator)

```
>> choco install nodejs          **for brew use >> brew install node
>> choco install yarn
>> choco install git
```

ในการติดตั้ง create-react-app CLI สามารถทำผ่านการใช้คำสั่ง yarn ผ่าน Command Prompt (ที่รันด้วยสิทธิ์ Administrator)

```
>> yarn global add create-react-app
```

** สำหรับคอมพิวเตอร์ให้ห้องปฏิบัติการ เครื่องมือข้างต้นได้ติดตั้งไว้ล่วงหน้าเรียบร้อยแล้ว

2. Checkpoint

ในการทดลองนี้ ให้นักศึกษาสร้าง Repository ใหม่ใน github.com ด้วย Account ของตนเอง โดยให้มีการ commit และ push ทุกครั้งที่สิ้นสุดในแต่ละตอนของการทดลอง นศ. จะต้องนำเสนอโค้ดให้กับ TA ในแต่ละตอนจาก Github เป็นการ Checkpoint

3. Hello world

สร้างโปรเจ็คใหม่ชื่อว่า card-game และรันโค้ดโปรเจ็คดังกล่าว โดยใช้คำสั่งนี้

```
>> create-react-app card-game
>> cd cd-app
>> yarn start
```

ปรับปรุงไฟล์ App.js ให้มีโค้ดตามแสดงข้างล่าง บันทึกไฟล์ แล้วสังเกตุผล

ไฟล์ App.js

```
import React from 'react';
import './App.css';

function App() {
  return (
    <div>
      Hello World
    </div>
  );
}
export default App;
```

JSX

JSX จะมีลักษณะคล้ายกับ HTML อย่างไรก็ดี JSX มีความพิเศษตรงที่สามารถเขียนโค้ดร่วมกับ JavaScript ได้ และมีรูปการใช้งานพิเศษในบางกรณี

- JavaScript Expressions เช่น {2 +3} จะให้ผลลัพธ์เป็น 5
    - สามารถเขียนโค้ด JavaScript ที่ซับซ้อนได้ใน JavaScript Expressions
- ในกรณีที่ต้องการระบุ CSS class ของ element ให้ใช้คำว่า **`className`** แทน class (เนื่องจาก class เป็นคำสงวนของ JavaScript)

ทดลองเปลี่ยน ข้อความ Hello World ในไฟล์ App.js เป็น Hello {“World”}

4. Components & Props

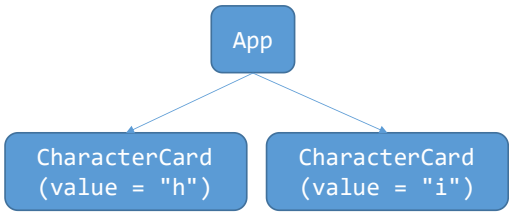
การสร้างโปรแกรมด้วย React คือการสร้างคอมโพเนนต์แล้วนำคอมโพเนนต์เหล่านั้น มาประกอบกันในรูปแบบโครงสร้างของ Tree ลักษณะเดียวกันกับ DOM เพียงแต่ React จัดจัดการกับ Virtual DOM ก่อนที่ Synchronized ไปยัง HTML DOM ของ Brower

สร้างไฟล์ CharacterCard.js เพื่อนิยามคอมโพเนนต์ CharacterCard

```
import React from 'react';
export default function CharacterCard(props) {
    return (
        <div>{props.value}</div>
    )
}
```

แก้ไขไฟล์ App.js

```
import CharacterCard from './CharacterCard';
function App() {
  return (
    <div>
      <CharacterCard value="h"/>
      <CharacterCard value="i"/>
    </div>
  );
}
```



จะสังเกตุได้ว่า มีการ import คอมโพเนนต์ CharacterCard มาจากอีกไฟล์ CharacterCard.js เพื่อนำคอมโพเนนต์มาใช้ในคอมโพเนนต์ App และมีการส่งผ่าน props ชื่อ value จากคอมโพเนนต์ App ไปยังคอมโพเนนต์ CharacterCard

ในกรณีที่เราต้องการวนซ้ำเพื่อสร้างคอมโพเนนต์หลายๆ คอมโพเนนต์พร้อมกัน ก็สามารถทำได้โดยการเขียนโค้ด JavaScript ลงไปใน JSX ได้ โดยแก้ไขไฟล์ App.js ดังนี้

```
const word = "Hello";
function App() {
  return (
    <div>
      {
         Array.from(word).map((c, i) => <CharacterCard value={c} key={i}/>)
      }
    </div>
  );
}
```

* การใส่ Props ชื่อว่า Key จะช่วยในกระบวนการอ้างอิงของ React ในกรณีวนซ้ำเพื่อสร้างคอมโพเนนต์

Styling

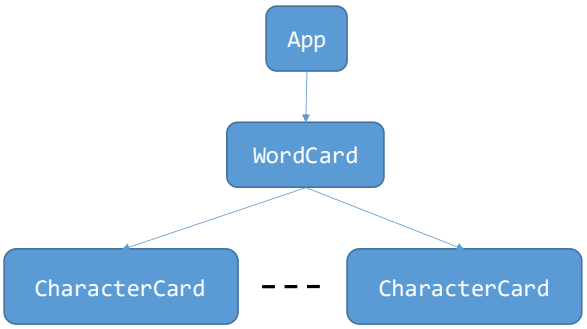
เพิ่ม CSS Class ชื่อ card ในไฟล์ App.css และกำหนด className ให้กับ div ของคอมโพเนนต์ CharacterCard

```
.card {
    display: inline-block;
    text-align: center;
    width: 3em;
    font-size: 2em;
    box-shadow: 0 4px 8px 0 rgba(0,0,0,0.2);
    margin: 1em;
    user-select: none;
}
```

h e l l o

More

เราสามารถสร้าง Tree ที่มีหลายลำดับขั้นมากขึ้น เพื่อรองรับการทำงานที่ซับซ้อนมากขึ้น



เพิ่มไฟล์ WordCard.js

```
import React from 'react';
export default function WordCard(props){
    return (
      <div>
       { Array.from(props.value).map((c, i) => <CharacterCard value={c} key={i}/>) }
      </div>
    );
}
```

ปรับเปลี่ยนไฟล์ App.js ให้ทำการ render คอมโพเนนต์ WordCard แทนที่จะ render คอมโพเนนต์ CharacterCard โดยตรง

```
  return (
    <div>
      <WordCard value="hello"/>
    </div>
  );
```

5. Components & States

Props เป็นข้อมูลที่ถูกส่งจากคอมโพเนนต์แม่ (อยู่ข้างบนใน DOM) ไปสู่คอมโพเนนต์ลูก (อยู่ด้านล่างใน DOM) เมื่อส่งผ่านข้อมูลไปแล้ว หากคอมโพเนนต์ต้องการจัดการกับข้อมูลภายใน เพื่อให้การแสดงผลหรือลอจิกการทำงานเป็นไปตามที่ต้องการ คอมโพเนนต์สามารถจัดการกับข้อมูลเหล่านั้นได้ผ่าน State ทั้งนี้เมื่อ State ถูกอัพเดต (ผ่านฟังก์ชันสำหรับการเปลี่ยน state เท่านั้น ซึ่งในตัวอย่างข้างล่างคือฟังก์ชัน steActive) การแสดงผล (UI) จะตอบสนองต่อการเปลี่ยนแปลงนั้นโดยอัตโนมัติ

ปรับปรุง CharacterCard ให้รองรับการถูกคลิก โดยจะเปลี่ยนรูปแบบการแสดงผลผ่านการกำหนด className เมื่อถูกคลิก ทั้งนี้จะตอบสนองต่อการคลิกเพียงครั้งเดียวเท่านั้น

```
import React, { useState } from 'react';
export default function CharacterCard(props) {
    const [active, setActive] = useState(false);
    const activate = () => {
        setActive(true)
    }

    const className = `card ${active ? 'activeCard': ''}`
    return (
        <div className={className} onClick={activate}>{props.value}</div>
    )

}
```

ปรับปรุงไฟล์ App.css เพื่อเพิ่มคลาส activeCard

```
.activeCard {
    color: red;
    background: pink;
}
```

H L E O L

Handler as Props

หาก WordCard ต้องการทราบถึงเหตุการณ์ที่ CharacterCard ถูก Activated ในกรณีทั่วไปทั้งสองคอมโพเนนต์สามารถสื่อสารกันได้ โดยให้ WordCard ส่ง Handler (function ที่จัดการกับเหตุการณ์) ไปเป็น props ให้กับ CharacterCard

แก้ไขไฟล์ WordCard.js จำนวนสองจุด คือการนิยามเมธอดในฟังก์ชัน WordCard และการส่งผ่าน handler ไปเป็น props

```
const activationHandler = c => { console.log(`${c} has been activated.`) }
...
<CharacterCard value={c} key={i} activationHandler={activationHandler}/>
```

แก้ไขไฟล์ CharacterCard.js ให้เรียกใช้ activationHandler

```
    const activate = () => {
       if(!active){
           setActive(true)
           props.activationHandler(props.value)
       }
    }
```

6. Game Logic

ติดตั้งไลบรารี lodash

```
>> yarn add lodash
```

** ก่อนการติดตั้ง ให้หยุดการการทำงานของ yarn start ที่สั่งไว้ก่อนหน้า

ปรับปรุงไฟล์ WordCard.js เพื่อให้ทำการสุ่มลำดับของตัวอักษรในคำที่จะแสดงผล รับรู้ตัวอักษรทีละตัวที่ผู้เล่นทำการกด เมื่อกดครบทั้งคำแล้ว ให้ตรวจสอบว่า คำที่ผู้เล่นสร้างจากการกด กับคำต้นแบบตรงกันหรือไม่ หากตรงกันถือว่าชนะ หากไม่ตรงกันให้เล่นใหม่

```
import _ from 'lodash';
const prepareStateFromWord = (given_word) => {
    let word = given_word.toUpperCase()
    let chars = _.shuffle(Array.from(word))
    return {
      word,
      chars,
      attempt: 1,
      guess: '',
      completed: false
    }
}
```

หมายเหตุ

word คำต้นฉบับแบบตัวอักษรใหญ่

chars ตัวอักษรแต่ละตัวที่สุ่มลำดับแล้ว

attempt จำนวนครั้งที่พยายามเล่น

guess ข้อความที่ผู้เล่นเดาเข้ามา

completed การเล่นสิ้นสุดแล้วหรือไม่

```
const activationHandler = (c) => {
  console.log(`${c} has been activated.`)

  let guess = state.guess + c
  setState({...state, guess})

  if(guess.length == state.word.length){
    if(guess == state.word){
      console.log('yeah!')
      setState({...state, guess: '', completed: true})
    }else{
      console.log('reset')
      setState({...state, guess: '', attempt: state.attempt + 1})
    }
  }
}
```

ให้ส่งผ่านค่าของ attempt ไปยัง CharacterCard (ให้ นศ. เขียนโคดด้วยตนเอง) และ CharacterCard จะได้รีเซ็ตค่า activated เป็น false อีกครั้งเมื่อ attempt ใหม่ถูกส่งผ่านมา

แก้ไขไฟล์ CharacterCard.js เพื่อตรวจสอบว่า เมื่อ attempt เปลี่ยนไป component จะต้องถูก render ใหม่อีกครั้ง

```
const attemptRef = useRef(props.attempt);
...
useEffect(() => {
    if(attemptRef.current != props.attempt){
        setActive(false)
        attemptRef.current = props.attempt
    }
})
```

7. งานหลังการทดลอง

ศึกษาข้อมูลเพิ่มเติมเกี่ยวกับ React แล้วให้ นศ. ปรับปรุงเกม Logic ตามที่นศ. ต้องการ แล้วทำการ push ไว้ใน Github โดยจะต้องมีการ commit และ push ระหว่างทำอย่างน้อย 5 commit

ให้เขียนสรุปสิ่งที่ทำไว้ (เป็น text file หรือ md format) ที่ root ของ project โดยให้ตั้งชื่อว่า submission.txt หรือ submission.md แล้ว push มาด้วยใน Github ในไฟล์จะต้องระบุ ชื่อ-สกุล และรหัสนักศึกษาไว้ด้วย