**PRAKTIKUM**

**PENGEMBANGAN APLIKASI BERGERAK**

**MODUL 1**

**Instalasi Environment dan**

**Pengenalan React Tingkat Dasar**

**A logo with a red book and a black background

Description automatically generated**

Farid Duta Hadyanto - 1204215118-

**PROGRAM STUDI S1 SISTEM INFORMASI**

**FAKULTAS TEKNOLOGI INFORMASI DAN BISNIS**

**INSTITUT TEKNOLOGI TELKOM SURABAYA**

**2023**

* 1. **Instalasi Envirountment**

Sebelum kita memulai pengembangan aplikasih bergerak hendaknya kita menyiapkan beberapa environment pendukung untuk memulai berkreasi di mobile apps.

Yang perlu kita persiapkan kita menginstall node , npm dan git

**A screenshot of a computer program

Description automatically generated\**

## Environment selanjutnya adalah menginstall expo go dengan code

Npx create-expo-app {dengan nama bebas}

Code ini untuk membuat project react nativ dengan bundling expo

A screenshot of a computer

Description automatically generated

## Belajar Javascript Dasar

**3.1 Valuriabel**

A black rectangular object with white lines

Description automatically generated

|  |
| --- |
| console**.**log("------ Variables ------")**;**  *// Var*  **for** (var i **=** 0**;** i **<** 5**;** i**++**) {      console**.**log(i)**;**  }  console**.**log(i)**;**  *// Var*  function looping() {  **for** (var i **=** 0**;** i **<** 5**;** i**++**) {          console**.**log(i)**;**      }  }  looping()**;**  console**.**log(i)**;**  *// Let*  **for** (let i **=** 0**;** i **<** 5**;** i**++**) {      console**.**log(i)**;**  }  console**.**log(i)**;**  *// Let*  **for** (let i **=** 0**;** i **<** 5**;** i**++**) {      console**.**log(i)**;**  }  **if** (true) {      let i **=** 100**;**      console**.**log("i ="**,**i)**;**  }  *// Const*  const phi **=** 3.14**;**  *//phi = 3.147;*  console**.**log(phi)**;** |

1. **Menggunakan var dalam Loop Pertama:**

Sebuah loop for menggunakan variabel i dengan kata kunci var untuk menginisialisasi, mengupdate, dan mengakses nilai i.

Loop ini mencetak nilai i dari 0 hingga 4.

Setelah loop selesai, nilai i tetap dapat diakses di luar loop karena deklarasinya menggunakan var. Oleh karena itu, pernyataan console.log(i) di luar loop akan mencetak nilai i, yang pada saat itu bernilai 5.

1. **Menggunakan var dalam Fungsi looping:**

Sebuah fungsi bernama looping juga menggunakan variabel i dengan kata kunci var untuk melakukan loop dan mencetak nilai i dari 0 hingga 4.

Meskipun variabel i dalam fungsi looping memiliki cakupan (scope) yang berbeda dari variabel i di luar fungsi, keduanya tidak saling mempengaruhi. Variabel i dalam fungsi looping bersifat lokal untuk fungsi tersebut.

Setelah fungsi looping selesai dijalankan, pernyataan console.log(i) di luar fungsi akan mencetak nilai i yang ada di luar fungsi, yaitu 5.

1. **Menggunakan let dalam Loop Kedua:**

Sebuah loop for menggunakan variabel i dengan kata kunci let. Variabel i yang dideklarasikan dengan let memiliki cakupan yang lebih terbatas dibandingkan dengan var.

Loop ini mencetak nilai i dari 0 hingga 4.

Setelah loop selesai, pernyataan console.log(i) di luar loop akan menghasilkan ReferenceError karena variabel i dideklarasikan dengan let dan hanya memiliki cakupan dalam blok loop.

1. **Menggunakan let dalam Loop Ketiga:**

Ini adalah loop terpisah yang juga menggunakan variabel i dengan kata kunci let. Variabel i ini berbeda dari variabel i yang digunakan dalam loop sebelumnya.

Loop ini mencetak nilai i dari 0 hingga 4.

Variabel i dalam loop ini juga hanya memiliki cakupan dalam blok loop, sehingga pernyataan console.log(i) di luar loop akan menghasilkan ReferenceError jika dijalankan.

1. **Menggunakan let dalam Blok If:**

Di dalam blok if, ada deklarasi variabel i menggunakan let, dan nilainya diatur menjadi 100.

Variabel i ini hanya memiliki cakupan dalam blok if. Oleh karena itu, pernyataan console.log("i =", i) dalam blok if akan mencetak nilai 100, tetapi variabel i di luar blok if tidak terpengaruh dan tetap tidak terdefinisi.

1. **Menggunakan const untuk Variabel phi:**

Variabel phi dideklarasikan dengan kata kunci const dan diinisialisasi dengan nilai 3.14.

Karena variabel phi dideklarasikan dengan const, nilainya tidak dapat diubah setelah inisialisasi. Jika Anda mencoba untuk mengubah nilai phi dengan pernyataan phi = 3.147;, itu akan menghasilkan error (TypeError).

## 3.2 Class

console**.**log("------ Class ------")**;**

class Car {

    constructor(**name**) {

*this***.***brand* **=** name**;**

    }

    present() {

**return** "I have a " **+** *this***.***brand***;**

    }

}

let myCar **=** **new** Car("Ford")**;**

console**.**log(myCar**.**present())**;**

1. Deklarasi Kelas Car:

Kode dimulai dengan mendeklarasikan sebuah kelas dengan nama Car menggunakan kata kunci class.

Kelas Car memiliki sebuah konstruktor (constructor) yang menerima satu parameter name. Konstruktor ini digunakan untuk menginisialisasi properti brand dari objek yang akan dibuat berdasarkan kelas ini.

Kelas Car juga memiliki metode (fungsi) bernama present yang akan mengembalikan sebuah string yang berisi teks "I have a " diikuti oleh merek mobil yang diberikan saat objek dibuat.

Membuat Objek myCar:

Setelah mendeklarasikan kelas Car, kode selanjutnya membuat sebuah objek bernama myCar dengan menggunakan konstruktor kelas Car.

Objek myCar dibuat dengan merek "Ford", sehingga properti brand-nya akan memiliki nilai "Ford".

Mengakses dan Mencetak Informasi Mobil:

Setelah objek myCar dibuat, kode kemudian mencetak informasi tentang mobil tersebut dengan menggunakan metode present yang ada dalam kelas Car.

Hasil cetakan akan menjadi "I have a Ford" karena objek myCar memiliki properti brand yang bernilai "Ford" dan metode present menggabungkannya dengan teks lain.

Kode tersebut merupakan contoh penggunaan kelas dalam JavaScript. Dalam kode tersebut, kita mendefinisikan kelas Car yang memiliki properti merek mobil (brand) dan metode present untuk mengembalikan informasi tentang mobil tersebut dalam bentuk string. Setelah mendefinisikan kelas Car, kita membuat objek myCar dengan merek "Ford" menggunakan konstruktor kelas tersebut. Kemudian, kita menggunakan metode present pada objek myCar untuk mencetak informasi "I have a Ford". Ini menggambarkan cara penggunaan kelas dalam JavaScript untuk mengorganisasi dan mengelola data dan perilaku yang terkait dengan objek tertentu, sehingga memungkinkan pengembang untuk membuat objek-objek dengan struktur dan fungsi yang serupa secara efisien.

## Inheritance

Inheritance merupakan teknik untuk pewarisan

A screenshot of a computer

Description automatically generated

console**.**log("------ Inheritance ------")**;**

class Gadget {

    constructor(**name**) {

*this***.***name* **=** name**;**

    }

    present() {

**return** "I have an " **+** *this***.***name***;**

    }

}

class Handphone *extends* *Gadget* {

    constructor(**name,** **mod**) {

*super*(name)**;**

*this***.***model* **=** mod**;**

    }

    show() {

**return** *this***.**present() **+** ", it is " **+** *this***.***model***;**

    }

}

let handphone **=** **new** Handphone("iphone"**,** "14 Pro Max")**;**

console**.**log(handphone**.**show())**;**

Kode di atas menggambarkan konsep pewarisan (inheritance) dalam pemrograman berorientasi objek menggunakan JavaScript. Dalam contoh ini, kita memiliki dua kelas, yaitu **Gadget dan Handphone**. **Kelas Gadget** memiliki **properti name** dan **metode present** yang mengembalikan informasi tentang gadget**. Kelas Handphone** adalah **turunan** dari kelas Gadget yang tambahkan properti model dan metode show untuk menampilkan informasi yang lebih spesifik tentang handphone.

Ketika objek handphone dibuat dengan merek "iphone" dan model "14 Pro Max" menggunakan konstruktor kelas Handphone, kita dapat menggunakan metode show pada objek tersebut untuk mencetak informasi lengkap tentang handphone tersebut, seperti "I have an iphone, it is 14 Pro Max". Hal ini menunjukkan bagaimana pewarisan memungkinkan kita untuk menggabungkan dan memperluas fungsionalitas dari kelas yang ada untuk menciptakan hierarki kelas yang lebih terperinci dan spesifik.

## Arrow Function

Fungsi panah (arrow function) adalah fitur dalam JavaScript yang memungkinkan Anda mendefinisikan fungsi dengan sintaks yang lebih singkat dan mudah dibaca.

1. Normal function

|  |
| --- |
| *// Normal Function*  function hello() {  **return** "Hello World!"**;**  } |

1. Normal Function (Variabel):

|  |
| --- |
| *// Normal function*  hello **=** function() {  **return** "Hello World!"**;**  } |

1. Arrow function

|  |
| --- |
| *// Arrow Function*  hello **=** () **=>** {  **return** "Hello World! This is from Arrow Function"**;**  } |

1. Arrow Function sederhana

|  |
| --- |
| hello **=** () **=>** "Hello World! This is from Arrow Function"**;**  console**.**log(hello())**;** |

1. Arrow Function dengan 1 Parameter:\

|  |
| --- |
| *// Arrow Function with 1 parameter*  hello **=** **myName** **=>** `Hello ${myName}! This is from Arrow Function`**;** |

## Destructing

Destructuring adalah fitur dalam JavaScript yang memungkinkan untuk mengekstrak nilai-nilai dari array atau properti-properti dari objek dan menetapkannya ke variabel terpisah secara lebih mudah dan kompak. Dalam kode yang diberikan, telah diilustrasikan penggunaan destructuring dalam beberapa konteks yang berbeda, baik pada array maupun objek.

1. Destrukting array

|  |
| --- |
| let items **=** ["Table"**,** "Handphone"**,** "Computer"]**;**  let [item\_1, item\_2, item\_3] = items; |

Pada potongan kode ini, dideklarasikan sebuah array items yang berisi beberapa item. Kemudian, digunakan destructuring untuk mengekstrak nilai-nilai dari array tersebut dan menetapkannya ke dalam variabel item\_1, item\_2, dan item\_3. Dengan cara ini, bisa mengakses item-array tanpa harus secara manual menetapkan setiap variabel secara terpisah.

1. Destructuring Objek:

|  |
| --- |
| *// Object*  let student **=** {      name**:** "Ahmad"**,**      age**:** 22**,**      department**:** "Information System"  }  let { name, age, department } = student; |

Di sini, dideklarasikan objek student dengan properti-properti seperti nama (name), usia (age), dan departemen (department). Kemudian, digunakan destructuring untuk mengekstrak nilai-nilai properti dari objek tersebut dan menetapkannya ke variabel dengan nama yang sama, yaitu name, age, dan department.

1. Alias

|  |
| --- |
| *// Destructuring by providing aliases*  let {name: objectName**,** age: objectAge**,** department: objectDepartment} **=** student**;** |

Dalam contoh ini, diberikan alias kepada variabel yang akan digunakan untuk menampung nilai-nilai properti objek. Misalnya, properti name dari objek student disimpan dalam variabel objectName, properti age disimpan dalam variabel objectAge, dan properti department disimpan dalam variabel objectDepartment. Ini memungkinkan untuk memberikan nama variabel yang berbeda jika diperlukan.

1. Destructuring Nilai Kembali dari Fungsi:

|  |
| --- |
| const studentData **=** () **=>** {  **return** { name**:** "Roni"**,** age**:** 19**,** department**:** "Information System" }  }  let {      name: studentNameData**,**age: studentAgeData**,** department: studentDepartmentData} **=** studentData()**;** |

Dalam contoh ini, didefinisikan fungsi studentData yang mengembalikan objek dengan properti-properti seperti nama, usia, dan departemen. Kemudian, digunakan destructuring untuk mengekstrak nilai-nilai dari objek yang dikembalikan oleh fungsi tersebut dan menetapkannya ke variabel dengan alias yang sesuai.

## penggunaan operator spread

|  |
| --- |
| *// Array*  let arr1 **=** [10**,** 20**,** 30]**;**  let arr2 **=** [100**,** 200**,** 300]**;**  array pada javascipt |
| *// Without Spread Operator*  let arr3 **=** [arr1**,** arr2]**;**  console**.**log(arr3)**;**  Dalam contoh ini, mencoba menggabungkan dua array ke dalam arr3, namun yang terjadi adalah memasukkan kedua array tersebut sebagai elemen-elemen dalam array arr3, menghasilkan array bersarang. |
| *// With Spread Operator*  let arr4 **=** [**...**arr1**,...**arr2]**;**  console**.**log(arr4)**;**  *Pada contoh ini, menggunakan operator spread (...) untuk menggabungkan elemen-elemen dari arr1 dan arr2 ke dalam array arr4, menghasilkan array yang berisi elemen-elemen dari kedua array yang digabungkan.* |
| *// With Spread Operator*  let arr5 **=** [**...**arr1**,** 40**,** 50]**;**  console**.**log(arr5)**;**  *// With Spread Operator*  let arr6 **=** [30**,** 40**,** 50**,** **...**arr1]**;**  console**.**log(arr6)**;**  ]  Dalam contoh di atas, dapat menggunakan operator spread untuk menambahkan elemen baru ke array. Misalnya, arr5 akan memiliki elemen 40 dan 50 yang ditambahkan ke elemen-elemen dari arr1. |
| *// Without Spread Operator*  let employee1 **=** {      employee**,**      age**:** 25**,**      city**:** "Jakarta"  }**;**  console**.**log(employee1)**;** |
| *// With Spread Operator*  let employee2 **=** {  **...**employee**,**      age**:** 20**,**      city**:** "Surabaya"  }**;**  console**.**log(employee2)**;** |
| Dengan operator spread, dapat dengan mudah menggabungkan semua properti dari employee ke dalam objek employee2. Ini membuat objek employee2 memiliki properti yang sama seperti employee, tetapi juga memungkinkan untuk menimpa properti tertentu, seperti age dan city, sesuai kebutuhan. |

## 4.1 JSX (Javascript XML)

Basic rules

|  |
| --- |
| myElement = (  <p>Learn Javascript</p>  <p>Learn React</p> ); //wrong |

Pada contoh code ditas merupakan contoh yang salah karena dalam rules JSX mestinya kode diberikan pengelompokan beberapa elemen di dalam satu element root seperti code dibawah ini .

|  |
| --- |
| // Basic Rules  myElement = (  <div>  <p>Learn Javascript</p>  <p>Learn React</p>  </div> ); // correct |

Pada contoh kedua, kelompokkan kedua elemen <p> di dalam satu elemen <div>. Ini adalah cara yang benar dalam JSX. Elemen <div> berperan sebagai elemen root utama yang mengelilingi elemen-elemen lainnya. Dengan demikian, Anda mematuhi aturan dasar JSX yang memerlukan satu elemen root.

4.2 Class Component

|  |
| --- |
| // Class Component  class Practice1 extends React.Component {  render() {  return <h3>Practice: ReactJS</h3>  }  }  let myElement = <Practice1 />; |

A screen shot of a computer program

Description automatically generated

Pada code sebelumnya belum ada code untuk merender component react kedalam DOM setelah di berikan component react dapat terender ke dalam dom.

|  |
| --- |
| ReactDOM**.**render(myElement**,** document**.**getElementById('root'))**;** |

Baris kode ini mengatakan kepada React untuk merender komponen yang disimpan dalam myElement ke dalam elemen DOM yang memiliki id "root" pada halaman web. Dengan demikian, setelah komponen React diberikan kepada ReactDOM dengan perintah ini, komponen tersebut dapat di-render ke dalam DOM dan akan ditampilkan di halaman web dengan mengambil referensi ke elemen dengan id "root".

Prop

A white background with black text

Description automatically generated

|  |
| --- |
| *// Props Children in Class Component*  class Practice3 *extends* React**.***Component* {      render() {  **return** <h3 *id***=**{*this***.***props***.***module*}>Practice: {*this***.***props***.***children*}</h3>**;**      }  }  *// Buat elemen React dengan komponen Practice3 dan prop children="ReactJS"*  const myElement **=** <Practice3 *module***=**"01">ReactJS</Practice3>**;**  *// Merender elemen React ke dalam elemen dengan id "root"*  ReactDOM**.**createRoot(document**.**getElementById('root'))**.**render(myElement)**;** |

4.3 Functional Component

A white background with black text

Description automatically generated

**A screenshot of a computer program

Description automatically generated**

**ReactTitle dan JavascriptTitle** merupakan dua functional components. Mereka adalah komponen React sederhana yang hanya mengembalikan elemen-elemen JSX. ReactTitle mengembalikan elemen <h3> dengan teks "Currently learning ReactJS," dan JavascriptTitle mengembalikan elemen <h3> dengan teks "Currently learning Javascript."

**MyComponent** juga merupakan functional component. Ini adalah komponen yang lebih kompleks yang menggabungkan dua komponen sebelumnya, yaitu ReactTitle dan JavascriptTitle, dalam sebuah elemen <div>. Dalam komponen ini, Anda menggunakan <hr /> (garis horizontal) untuk memisahkan kedua judul.

**ReactDOM** untuk merender MyComponent ke dalam DOM. ReactDOM adalah bagian dari React yang digunakan untuk menghubungkan komponen React dengan elemen DOM dalam halaman web Anda. Dalam kode tersebut, komponen MyComponent akan dirender ke dalam elemen dengan id 'root' pada halaman web Anda.

**4.4. Event Handler**

A white background with black text

Description automatically generated

*// Button Component*

    const MyButton **=** (**props**) **=>** {

        const handleButtonClick **=** () **=>** {

            alert(props**.***message*)**;**

        }

**return** (

            <button *onClick***=**{handleButtonClick} *style***=**{{ margin**:** "10px" }}>

                {props**.***children*}

            </button>

        )**;**

    }

*// App Component*

    const MyApp **=** () **=>** {

**return** (

            <div>

                <h1>Learn React</h1>

                <MyButton *message***=**"Learn React">React</MyButton>

                <MyButton *message***=**"Learn JavaScript">JavaScript</MyButton>

            </div>

        )**;**

    }

    const myElement **=** <MyApp />**;**

    const myApp **=** myElement**;**

    ReactDOM**.**createRoot(document**.**getElementById('root'))**.**render(myApp)**;**

Terdapat dua komponen utama, yaitu MyButton dan MyApp, yang digunakan untuk membuat tombol dan mengintegrasikannya ke dalam aplikasi React.

Komponen MyButton:

Ini adalah komponen fungsional (functional component) yang menerima properti (props) sebagai input. Komponen ini mengandung sebuah tombol (<button>) yang akan menampilkan teks dari props.children (dalam hal ini, teks "React" atau "JavaScript"). Ketika tombol diklik, ia memicu fungsi handleButtonClick, yang akan menampilkan pesan yang diterima dari properti props.message dalam sebuah alert.

Komponen MyApp:

Ini juga adalah komponen fungsional yang digunakan untuk merender tampilan utama aplikasi. Di dalamnya, terdapat elemen <h1> yang menampilkan judul "Learn React". Selain itu, komponen MyApp juga mengandung dua instance dari komponen MyButton. Setiap MyButton memiliki pesan yang berbeda yang ditentukan melalui properti message dan teks yang ditampilkan di dalam tombolnya, seperti "React" dan "JavaScript".

Render Aplikasi:

Terakhir, aplikasi di-render dengan menggunakan ReactDOM.createRoot dan ditempatkan di dalam elemen HTML dengan id 'root'.

**4.5 State**

Tampilan code 1

A screenshot of a computer

Description automatically generated

Tampilan code 2

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

I mport React dan ReactDOM: Pertama-tama, kita mengimpor pustaka React dan ReactDOM yang diperlukan untuk memulai pengembangan aplikasi React. Membuat Komponen Cat: Kami membuat komponen Cat, yang bertindak sebagai representasi dari seekor kucing. Di dalam komponen ini, kami mendeklarasikan state untuk kucing dengan properti isHungry yang awalnya diatur sebagai true, menunjukkan bahwa kucing tersebut lapar.

Membuat Tombol Klik di Komponen Cat: Di dalam komponen Cat, kami menambahkan sebuah tombol yang dapat diklik. Saat tombol tersebut diklik, sebuah fungsi yang disebut handleButtonClick() dipanggil untuk mengubah status kenyang atau lapar kucing dengan mengubah nilai isHungry menjadi false.

Membuat Komponen Cafe: Kami membuat komponen Cafe, yang bertujuan untuk menampilkan beberapa kucing dalam konteks "kafe". Di dalam komponen Cafe, kami menambahkan beberapa instansi dari komponen Cat dengan nama yang berbeda.

Render Aplikasi: Kami membuat elemen menggunakan komponen Cafe dan menetapkannya ke dalam variabel myElement. Selanjutnya, kami menggunakan ReactDOM.createRoot untuk merender myApp (elemen Cafe) dan menempatkan hasil render ini di dalam elemen HTML dengan id 'root'.

Interaksi: Setelah aplikasi dijalankan, dua kucing ("Munkustrap" dan "Spot") akan muncul dalam sebuah div di halaman web. Setiap kucing memiliki tombol yang dapat diklik. Ketika tombol pada kucing yang lapar diklik, status kucing akan berubah menjadi kenyang, dan tombol tersebut akan menjadi tidak dapat diklik lagi.

## PROGRAM COUNTER

1. Class Component

A screenshot of a computer

Description automatically generated

Pada kesempatan ini saya membuat program counter sederhana menggunakan class component. Yang pertama adalah importing React dan ReactDOM dengan

|  |
| --- |
| Class Conter Extends Reac.Component { |

Code diatas adalah importing React dan ReactDom untuk membuat dan merender komponent React.

Kemudian Setlah itu membuat contructor component dari aplikasi counter

|  |
| --- |
| constructor(props) {  super(props);  this.state = {  count: 0  };  } |

Di dalam konstruktor komponen Counter, Anda menginisialisasi this.state dengan properti count yang awalnya diatur ke 0. Properti count akan digunakan untuk melacak nilai counter.

Kemudian membuat methode increment untuk menambah dan decrement mengurng

|  |
| --- |
| increment **=** () **=>** {  *this***.**setState({ count**:** *this***.***state***.***count* **+** 1 })**;**      }      decrement **=** () **=>** {  *this***.**setState({ count**:** *this***.***state***.***count* **-** 1 })**;**      } |

Kemudinan Render

|  |
| --- |
| render() {  **return** (          <div>            <h1>Counter</h1>            <h1> {*this***.***state***.***count*} </h1>            <button *onClick***=**{*this***.***increment*}>Tambah +1</button>            <button *onClick***=**{*this***.***decrement*}>Kurang -1</button>          </div>        )**;**      } |

Code ini adalah metode untuk merender tampilan komponen. Pada tek counter sebegai judul kemudaian menampilkan nilai this.stete.count. didalam element <H1> .

|  |
| --- |
| ReactDOM**.**render(<Counter />**,** document**.**getElementById('root'))**;** |

Terkhir adalah menggunakan ReactDOM.render. untuk merender komponent counter ke dalama HTML dengan elemetent yang akan kita masuk adalah element id yang bernama root

1. Functional Component

*import* React**,** { useState } *from* 'react'**;**

*import* ReactDOM *from* 'react-dom'**;**

function Counter() {

    const [count**,** setCount] **=** useState(0)**;**

    const increment **=** () **=>** {

        setCount(count **+** 1)**;**

    }

    const decrement **=** () **=>** {

        setCount(count **-** 1)**;**

    }

**return** (

        <div>

            <h1>Counter</h1>

            <h1> {count} </h1>

            <button *onClick***=**{increment}>Tambah +1</button>

            <button *onClick***=**{decrement}>Kurang -1</button>

        </div>

    )**;**

}

ReactDOM**.**render(<Counter />**,** document**.**getElementById('root'))**;**

Komponen Counter adalah sebuah komponen React yang dibuat sebagai functional component. Dalam komponen ini, kita menggunakan React Hook useState untuk mengelola state bernama count. State ini menyimpan nilai yang akan ditampilkan pada halaman web. Awalnya, nilai count diatur ke 0.

Dua fungsi, yaitu increment dan decrement, digunakan untuk mengubah nilai count. Fungsi increment akan menambahkan nilai count sebanyak 1 setiap kali tombol "Tambah +1" ditekan, sementara fungsi decrement akan mengurangi nilai count sebanyak 1 saat tombol "Kurang -1" ditekan.

Di dalam elemen HTML yang di-return oleh komponen, kita memiliki judul "Counter" yang ditampilkan sebagai elemen <h1>. Di bawah judul tersebut, nilai count yang terkini ditampilkan dalam elemen <h1> juga, sehingga pengguna dapat melihat jumlah saat ini.

Selain itu, terdapat dua tombol yang digunakan untuk menginteraksi dengan komponen. Ketika tombol "Tambah +1" ditekan, fungsi increment akan dipanggil dan nilai count akan bertambah. Ketika tombol "Kurang -1" ditekan, fungsi decrement akan memanggil dan nilai count akan berkurang.