Rakamin Academy

React Native

Synchronous & Asynchronous

Raih Mimpi #TanpaBatas

Outline Pembelajaran

Rakamin Academy

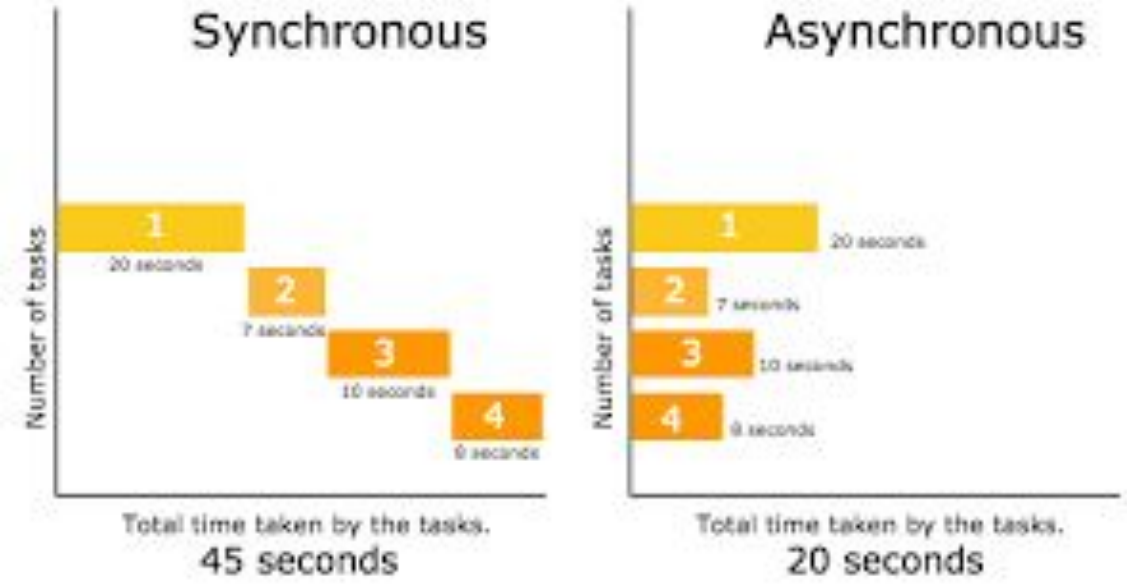
React Native Synchronous & Asynchronous

- [ ] Synchronous & Asynchronous
- [ ] Promise, Async, Await
- [ ] Callback

Objektif sesi

- Memahami proses Synchronous & Asynchronous
- Memahami dan mengimplementasikan Async & Await
- Memahami penggunaan Callback

Synchronous & Asynchronous Illustration



Synchronous

Dalam operasi synchronous, setiap langkah dieksekusi satu per satu secara berurutan.

Contohnya, dalam pengulangan for, setiap iterasi menunggu iterasi sebelumnya selesai sebelum melanjutkan ke iterasi berikutnya.

```
const SynchronousComponent = () => {
  const handleClick = () => {
    console.log('1. Start synchronous operation');

    for (let i = 0; i < 3; i++) {
      console.log(`2. Synchronous operation ${i + 1}`);
    }

    console.log('3. End synchronous operation');
  };

  return (
    <View>
      <Text>Synchronous Component</Text>
      <Button title="Run Synchronous Operation" onPress={handleClick} /
    </View>
  );
};
```

Asynchronous

Dalam operasi asynchronous, kode tidak menunggu operasi yang lama selesai sebelum melanjutkan.

Contohnya, dalam **setTimeout**, operasi asynchronous dimulai, dan kode berlanjut ke langkah berikutnya tanpa menunggu timeout selesai.

```
const AsynchronousComponent = () => {
  const handleAsyncClick = () => {
    console.log('1. Start asynchronous operation');

    // Operasi asynchronous (contoh: setTimeout)
    setTimeout(() => {
      console.log('3. Asynchronous operation complete');
    }, 2000);

    console.log('2. Continue synchronous operation');
  };

  return (
    <View>
      <Text>Asynchronous Component</Text>
      <Button title="Run Asynchronous Operation" onPress={handleAsyncCl
    </View>
  );
};
```

Rakamin Academy

Catatan:

Perlu diingat bahwa di React Native, banyak operasi seperti pengambilan data dari server atau membaca berkas dilakukan secara asynchronous untuk menghindari **UI freeze** dan memberikan responsivitas yang lebih baik kepada pengguna.

Penggunaan **async** dan **await**, serta penggunaan **Promise**, adalah umum dalam operasi **asynchronous** di React Native.

Promise, Async, Await

I'm fullfilling this ravest

Something gone wrong

```
var x = new Promise((resolve, reject) => {
  if(success) resolve('data ..')
  if(err) reject('error')
})

x.then((data)=>{...})
 .catch((err)=>{...})
```

```
async function add(x, y){
  let promise = sum(x, y)

  let result = await promise
  return result
}
```

Works only inside async functions

```
function sum(x, y) {
  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve(x + y), 100)
  })

  return promise
}
```

It can be an API call!

```
add(5, 10).then(z => console.log(z)) // 15
```

Wrapped in a resolved promise automatically

```
async function add(x, y){
  return x + y
}
```

same under the hood

```
return Promise.resolve(x + y)
```

```
add(5, 10)
  .then(z => console.log(z)) // 15
```

PROMISE

ASYNC

AWAIT

Promise

simulateAsyncOperation adalah fungsi yang mengembalikan Promise. Fungsi ini mensimulasikan operasi asinkron yang membutuhkan waktu 2 detik.

```
const simulateAsyncOperation = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const success = Math.random() > 0.5; // Menentukan apakah operasi
      if (success) {
        resolve('Async operation complete');
      } else {
        reject('Async operation failed');
      }
    }, 2000);
  });
};
```

Async Await

try dan **catch** digunakan untuk menangani hasil atau kesalahan dari operasi asinkron.

finally digunakan untuk menetapkan bahwa operasi asinkron selesai, baik berhasil atau gagal.

```
const handleAsyncClick = async () => {
  try {
    // Mulai operasi asinkron
    setLoading(true);

    // Panggil fungsi simulateAsyncOperation dan tunggu hasilnya
    const asyncResult = await simulateAsyncOperation();

    // Setel state dengan hasil operasi asinkron
    setResult(asyncResult);
  } catch (error) {
    console.error('Error during async operation:', error);
  } finally {
    // Selesai operasi asinkron (baik berhasil atau gagal)
    setLoading(false);
  }
};
```

Promise, Async, Await

Saat tombol ditekan (***handleAsyncClick***), kita memanggil ***simulateAsyncOperation*** menggunakan **await**.

useState digunakan untuk mengelola state komponen dan memberikan respons terhadap kondisi loading dan hasil operasi asinkron.

```
  return (
    <View>
      <Text>Async/Await with Promise Example</Text>
      <Button
        title="Run Async Operation with Promise"
        onPress={handleAsyncClick}
        disabled={loading}
      />

      {loading && <Text>Loading...</Text>}

      {result && <Text>{result}</Text>}
    </View>
  );
};
```

Callback

Meskipun menggunakan callback merupakan pendekatan tradisional, namun pada kasus tertentu dan untuk kejelasan tertentu, callback masih dapat digunakan. Namun, penggunaan async dan await dengan Promise umumnya lebih bersih dan lebih mudah dimengerti.

Pada contoh ini, kita akan membuat fungsi yang menggunakan callback untuk menangani hasil operasi asynchronous.

Callback

simulateAsyncOperationWithCallback adalah fungsi yang menggunakan callback untuk menangani hasil atau kesalahan dari operasi asinkron.

Saat tombol ditekan (***handleCallbackClick***), kita memanggil ***simulateAsyncOperationWithCallback*** dengan memberikan fungsi callback yang menangani hasil operasi asinkron.

```
const simulateAsyncOperationWithCallback = (callback) => {
  setTimeout(() => {
    const success = Math.random() > 0.5; // Menentukan apakah operasi b
    if (success) {
      callback(null, 'Async operation complete');
    } else {
      callback('Async operation failed', null);
    }
  }, 2000);
};
```

Callback

Fungsi callback dipanggil dengan dua argumen: **error** (jika terjadi kesalahan) dan **asyncResult** (hasil operasi asinkron).

Kode setelah pemanggilan **simulateAsyncOperationWithCallback** di dalam callback akan dijalankan setelah operasi asinkron selesai.

useState tetap digunakan untuk mengelola state komponen dan memberikan respons terhadap kondisi loading dan hasil operasi asinkron.

```
const handleCallbackClick = () => {
  // Mulai operasi asinkron
  setLoading(true);

  // Panggil fungsi simulateAsyncOperationWithCallback dengan callbac
  simulateAsyncOperationWithCallback((error, asyncResult) => {
    // Tangani hasil atau kesalahan dari operasi asinkron
    if (error) {
      console.error('Error during async operation:', error);
    } else {
      // Setel state dengan hasil operasi asinkron
      setResult(asyncResult);
    }

    // Selesai operasi asinkron (baik berhasil atau gagal)
    setLoading(false);
  });
};
```


Rakamin
Academy
Thank you
Raih Mimpi #TanpaBatas