DISKUSIKAN MATERI

LAPORKAN MATERI

Daftar Modul

Masukkan kata kunci

Persetujuan Hak Cipta
Modul 1: Introduction Course (Gratis)
Prasyarat Kelas (Gratis)
Apa yang Akan Kita Pelajari? (Gratis)
Tools Requirement
Modul 2: ECMAScript 6 (Gratis)
Running Snippet Code (Gratis)
Persiapan Project Latihan (Gratis)

Chaining Promise

Kita sudah tahu buruknya penulisan *callback hell*. Namun kita juga tidak dapat menghindari keadaan di mana *asynchronous operation* saling bergantungan satu sama lain. Untuk menghindari penulisan callback hell, promise jadi salah satu solusinya.

Dengan promise kita dapat melakukan proses *asynchronous operation* secara berantai. Contohnya, Ketika kita memesan kopi robusta, maka tahapan yang dilalui oleh barista adalah memastikan biji kopi tersedia, membuat kopi, lalu menghidangkannya kepada pelanggan. Tahapan tersebut harus barista lakukan secara berurutan.

Bagaimana cara melakukan proses berantai pada Promise? Kita bisa melakukannya dengan cara seperti ini:

```
function reserveACoffee(type, miligrams) {
 getSeeds(type, miligrams)
 .then(makeCoffee)
 .then(servingToTable)
 .then(resolvedValue => {
   console.log(resolvedValue);
 })
 .catch(rejectedReason => {
   console.log(rejectedReason);
 })
}

reserveACoffee("liberica", 80);

/* output:
Pesanan kopi sudah selesai!
*/
```

Mengapa bisa seperti itu? Mari kita bedah kodenya satu persatu.

Ketika kita memesan kopi melalui fungsi ***reserveACoffee()***, pertama barista akan mengambil biji kopi melalui fungsi ***getSeeds()***. Fungsi ini membutuhkan 2 (dua) parameter yaitu ***type*** (tipe kopi), dan ***miligrams*** (banyak kopi yang diperlukan). Fungsi ini mengembalikan objek promise, di mana jika biji yang dipesan tersedia akan mengembalikan *resolve* -> "Biji kopi didapatkan!". Namun jika biji kopi tidak tersedia, maka akan mengembalikan *reject* -> "Maaf stok kopi habis!". Berikut kode dari fungsi ***getSeeds***:

```
const getSeeds = (type, miligrams) => {
 return new Promise((resolve, reject) => {
   if(state.seedStocks[type] >= miligrams) {
     state.seedStocks[type] -= miligrams;
     resolve("Biji kopi didapatkan!")
   } else {
     reject("Maaf stock kopi habis!")
   }
 });
}
```

Lalu kita panggil method ***.then()*** dari fungsi ***getSeeds***, dan memberikan parameter fungsi ***makeCoffee*** di dalamnya. Fungsi ***makeCoffee()*** akan menerima parameter berupa nilai yang dibawa ***resolve*** pada ***getSeeds()***. Fungsi ini juga mengembalikan nilai promise juga, di mana jika mesin kopi siap digunakan, maka akan mengembalikan resolve -> "Kopi berhasil dibuat". Namun jika sebaliknya, mesin kopi tidak siap untuk digunakan, maka akan mengembalikan reject -> "Maaf mesin tidak dapat digunakan!". Berikut kode dari fungsi ***makeCoffee***:

```
const makeCoffee = seeds => {
 return new Promise((resolve, reject) => {
   if(state.isCoffeeMakerReady) {
     resolve("Kopi berhasil dibuat!")
   } else {
     reject("Maaf mesin tidak dapat digunakan!");
   }
 })
}
```

Setelah kita mendapatkan kopi dari fungsi `makeCoffee`. Lalu kopi tersebut dihidangkan dengan menggunakan fungsi `servingToTable`. Fungsi ini juga mengembalikan promise dengan *resolve* yang membawa nilai ***"Pesanan kopi sudah selesa!"***.

```
const servingToTable = coffee => {
  return new Promise(resolve => {
    resolve("Pesanan kopi sudah selesai!")
  })
}
```

Lalu kita gunakan method `.then()` yang terakhir untuk mencetak nilai yang dikembalikan oleh fungsi `servingToTable`.

Kemudian yang paling terakhir adalah memanggil method `.catch()`. Di mana method ini akan menangani *promise rejection* yang terjadi. Entah itu disebabkan oleh biji kopi yang dipesan habis, ataupun mesin kopi tidak dapat digunakan.

Menjelaskan kode melalui skenario terkadang sulit untuk dimengerti. Sebagian dari kita mungkin lebih nyaman membaca keseluruhan kodenya secara langsung. Berikut contoh kode dalam penerapan chaining promise berdasarkan skenario di atas.

```
const state = {
  isCoffeeMakerReady: true,
  seedStocks: {
    arabica: 250,
    robusta: 60,
    liberica: 80
  }
}

const getSeeds = (type, miligrams) => {
  return new Promise((resolve, reject) => {
    if(state.seedStocks[type] >= miligrams) {
      state.seedStocks[type] -= miligrams;
      resolve("Biji kopi didapatkan!")
    } else {
      reject("Maaf stock kopi habis!")
    }
  });
}

const makeCoffee = seeds => {
  return new Promise((resolve, reject) => {
    if(state.isCoffeeMakerReady) {
```

Anda bisa mencoba dengan menjalankan kodenya melalui platform online melalui tautan berikut: https://repl.it/@dicodingacademy/163-02-chaining-promise?lite=true

⬅ KEMBALI KE MATERI SEBELUMNYA

LANJUTKAN KE MATERI BERIKUTNYA ➡