Javascript for React (operators, Async+ REST API)

Preparation

Membuat 2 files, `index.html` dan `index.js`

index.html

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For In Loop</h2>
<p>The for in statement loops through the properties of an object:</p>

<p id="demo"></p>

<script src="index.js"></script>
```

```
</body>
</html>
```

index.js

```
const demo = document.getElementById("demo");
```

Operators

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

Conditional branching

Statement `if(...)` mengevaluasi kondisi pada parentheses dan, jika hasilnya `true`, mengeksekusi kode blok

If-Else

Syntax

```
if (condition) {  //  block of code to be executed if the condition is true}
```

Contoh:

```
const getDay = () => {
  if (new Date().getDay() === 0) {
    return "Sunday";
  }
  if (newDate().getDay() === 1) {
    return "Monday";
  }
  if (newDate().getDay() === 2) {
    return "Tuesday";
  }
  if (newDate().getDay() === 3) {
    return "Wednesday";
  }
  if (newDate().getDay() === 4) {
    return "Thursday";
  }
  if (newDate().getDay() === 5) {
    return "Friday";
  }
  if (newDate().getDay() === 6) {
    return "Saturday";
  }
};
const today = getDay();
demo.innerHTML = `Today is ${today}`;
```

Ternary operator

Operator yang direpresentasikan oleh tanda tanya ?. Disebut ternary karena operator memiliki 3 operands.

```
const today = {
  day: 1,
  name: "Monday",
};

// Single Condition
new Date().getDay() === today.day
  ? (demo.innerHTML = `Today is ${today.name}`)
  : (demo.innerHTML = `Today is not ${today.name}`);
```

```
const anotherDay = {
  day: 5,
  name: "Friday",
};

// Multiple Condition
new Date().getDay() === today.day
  ? (demo.innerHTML = `Today is ${today.name}`)
  : new Date().getDay() === anotherDay.day
  ? (demo.innerHTML = `Today is ${today.anotherDay.day}`)
  : (demo.innerHTML = `Today is neither ${today.name} nor ${anotherDay.name}`);
```

Switch Case

```
const getDay = () => {
  switch (new Date().getDay()) {
    case 0:
      return "Sunday";
      break;
    case 1:
      return "Monday";
      break;
    case 2:
      return "Tuesday";
      break;
    case 3:
      return "Wednesday";
      break;
    case 4:
      return "Thursday";
      break;
    case 5:
      return "Friday";
      break;
    case 6:
      return "Saturday";
  }
};

const today = getDay();
demo.innerHTML = `Today is ${today}`;
```

Callbacks

> ## "I will call back later!"

Callback merupakan fungsi yang dilempar sebagai argumen ke fungsi yang lain. Fungsi Callback dapat berjalan setelah fungsi lain selesai.

```
const myNumbers = [4, 1, -20, -7, 5, 9, -6];

// Call removeNeg with a Callback
const posNumbers = removeNeg(myNumbers, (x) => x >= 0);

// Display Result
document.getElementById("demo").innerHTML = posNumbers;

// Remove negative numbers
function removeNeg(numbers, callback) {
  const myArray = [];
  for (const x of numbers) {
    if (callback(x)) {
      myArray.push(x);
    }
  }
  return myArray;
}
```

Promises

> ## "I Promise a Result!"

Fungsi yang berjalan secara pararel dengan fungsi yang lain disebut dengan **asynchronous**. Dengan asynchronous programming, program JavaScript dapat memulai long-running task dan melanjutkan task yang lain secara pararel. Karena hal tersebut, metode JS tidak lagi menggunakan callback tetapi menggunakan Promises.

Dalam promises, ada 2 istilah:

Promise adalah JavaScript object yang menghubungkan fungsi *executor* dan fungsi *consumer.*

Ketika producing code memperoleh hasil, kita harus memanggil salah satu atau kedua callback berikut:

Result	Call
Success	myResolve(result value)
Error	myReject(error object)

```
let myPromise = new Promise((myResolve, myReject) => {
// "Producing Code" (May take some time)
  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  (value) => {
    /* code if successful */
  },
  (error) => {
    /* code if some error */
  }
);
```

contoh:

```
const myDisplayer = (some) => {
  document.getElementById("demo").innerHTML = some;
};

let myPromise = new Promise((myResolve, myReject) => {
  let x = 0;

  // some code (try to change x to 5)

  if (x == 0) {
    myResolve("OK"); // mengirimkan flag OK melalui callback
  } else {
    myReject("Error"); // mengirimkan flag Error melalui callback
  }
});

myPromise.then(
  (value) => {
    myDisplayer(value); // menangkap hasil value untuk ditampilkan
  },
```

```
  (error) => {
    myDisplayer(error); // menangkap hasil error untuk ditampilkan
  }
);
```

Konstruktor `new Promise()` mengembalikan suatu objek `promise`. Ada 2 property yang dikembalikan:

1. State
    - `pending`: state awal ketika fungsi eksekutor mulai mengeksekusi
    - `fulfilled`: Ketika promise di-*resolved*.
    - `rejected`: Ketika promise di-*rejected*.
2. Value
    - `undefined`: value awal ketika `state` adalah `pending`.
    - `value`: Ketika `resolve(value)` dipanggil.
    - `error`: Ketika `reject(error)` dipanggil.

Async Await

> *"async dan await membuat promises lebih mudah ditulis"*

async membuat suatu fungsi mengembalikan Promise

await membuat fungsi menunggu suatu Promise

Async

Keyword async diletakan seb

```
const myFunction = async () => {
    return "Hello";
  }
```

sama dengan

```
const myFunction = () => {
  return Promise.resolve("Hello");
};
```

contoh penggunaan async-await agar lebih mudah:

```
const myDisplayer = (some)=> {
  document.getElementById("demo").innerHTML = some;
}

const myFunction = async () => {return "Hello";}

myFunction().then(
  (value) => {
    myDisplayer(value);
  },
  (error) => {
    myDisplayer(error);
  }
);
```

Await

Keyword `await` hanya dapat digunakan didalam fungsi `async`.

Keyword `await` membuat fungsi menghentikan sementara suatu eksekusi dan menunggu promise diselesaikan (resolved) sebelum dilanjutkan.

```
const value = await promise;
```

contoh:

```
const myDisplay = async () => {
  const myPromise = new Promise((resolve) => {
    setTimeout(function () {
      resolve("I love You !!");
    }, 3000);
  });
  document.getElementById("demo").innerHTML = await myPromise;
};
```

💡 Resolve dan reject merupakan callback predefined dari JavaScript sehingga jika salah satu tidak digunakan (dalam hal ini reject) maka dapat dihapus. Begitu pula saat menghandle promise, bila yang diinginkan hanya hasil dari error nya saja, dan ingin membuang resolved, maka saat *handle* promise, resolve dapat di *pass* sebagai null — sesuai kebutuhan.

REST API

REST API merupakan standard paling umum yang digunakan antara *Clients* (user ato aplikasi) yang ingin mengakses informasi dari web servers (aplikasi atau database). Client harus punya akses untuk mendapatkan informasi yang diinginkan.

Application Programming Interface atau disingkat API merupakan cara dimana 2 komputer saling berkomunikasi. Contohnya, aplikasi pengiriman makanan dapat menggunakan Google Maps API untuk mendukung track lokasi daripada membuat sendiri dari awal atau suatu website dapat menggunakan model AI dalam memberikan saran menggunakan ChatGPT REST API.

API yang mengikuti standar REST disebut dengan RESTful.

Base URL

Base URL atau disebut juga dengan request URL merupakan URL yang dibutuhkan untuk mengakses API

Endpoint

Endpoint merepresentasikan resource yang ingin diakses. Resource dapat berupa text, image, dokumen, atau data apapun.

Query

Query merepresentasikan hasil filter dari resoure yang ingin dikembalikan, contohnya resource track spotify dengan query negara=ID dan genre=Jazz

HTTP Verb

- **GET**: membuat read only request untuk membaca suatu resource
- **POST**: membuat resource baru berdasarkan payload yang dikirimkan pada body request
- **DELETE**: menghapus resource berdasarkan id
- **PUT**: update beberapa field dari suatu resource berdasarkan body request atau membuat yang baru jika resource sudah ada.
- **PATCH**: update hanya untuk resource yang sudah ada.

Pada dasarnya konsep tersebut seperti halnya aplikasi CRUD

Acronym	HTTP verb
Create	POST
Read	GET
Update	PUT & PATCH
Delete	DELETE

Status Code

Status code	Meaning
200 OK	Request was successful.
301 Moved Permanently	For SEO purposes when a page has been moved and all link equity should be passed through.
401 Unauthorized	Server requires authentication.
403 Forbidden	Client authenticated but does not have permissions to view resource.
404 Not Found	Page not found because no search results or may be out of stock.
500 Internal Server Error	Server side error. Usually due to bugs and exceptions thrown on the server side code.
503 Server Unavailable	Server side error. Usually due to a platform hosting, overload and maintenance issue.

💡 Kita bisa menggunakan REST API dengan bantuan Async Await dan membuat HTTP request dengan menggunakan ajax, fetch (Web API) atau library seperti axios.

contoh: JSON Placeholder

- BaseURL : https://jsonplaceholder.typicode.com
- Endpoints:
  - /posts
  - /comments
  - /albums
  - /photos
  - /todos
  - /users

```
const listEl = document.createElement("ul");
const getUsers = async () => {
  const response = await fetch("https://jsonplaceholder.typicode.com/users");
  const albums = await response.json();
  return albums;
};
const userList = getUsers().then((value) => {
  value.map((album) => {
    let titleText = "";
    let bodyText = "";
    const titleEl = document.createElement("li");
    const bodyEl = document.createElement("p");
    titleText += album.name;
    bodyText += album.email;
    titleEl.innerHTML = titleText;
    bodyEl.innerHTML = bodyText;
    titleEl.appendChild(bodyEl);
    listEl.appendChild(titleEl);
```

```
  });

  document.getElementById("demo").appendChild(listEl);
});
```

Reference

- https://mannhowie.com/rest-api
- https://rows.com/docs/getting-data-from-any-api