HTTP Metódusok és Kérések

POST, PUT, PATCH és DELETE Kérések

A HTTP protokollban többféle módszer létezik az adatok küldésére és kezelésére a szerverrel való kommunikáció során. A leggyakrabban használt módszerek a POST, PUT, PATCH és DELETE, amelyek különböző típusú műveletekhez alkalmazhatók. Ezek a metódusok az adatok kezelésére és manipulálására szolgálnak a RESTful API-kban.

POST Kérés

A POST kérések adatokat küldenek a szerverre, és általában új erőforrások létrehozására használják őket. A küldött adatokat a kérés törzsében (body) kell megadni.

Példa egy POST kérésre a `fetch` API használatával:

```
fetch("https://api.example.com/resource", {
    method: "POST",
    headers: {
        "Content-Type": "application/json",
    },
    body: JSON.stringify({
        name: "John Doe",
        email: "john.doe@example.com",
    }),
})
    .then((response) => {
        if (!response.ok) {
            throw new Error("Hálózati hiba történt");
        }
        return response.json();
    })
    .then((data) => {
        console.log("Sikeresen létrehozva:", data);
    })
    .catch((error) => {
        console.error("Hiba:", error);
    });
```

PUT Kérés

A PUT kérések egy meglévő erőforrás teljes módosítására vagy létrehozására szolgálnak. Ha az erőforrás létezik, akkor az új adatokkal felülíródik, ha nem létezik, akkor létrejön.

Példa egy PUT kérésre a `fetch` API használatával:

```
fetch("https://api.example.com/resource/1", {
    method: "PUT",
    headers: {
        "Content-Type": "application/json",
    },
    body: JSON.stringify({
        name: "Jane Doe",
        email: "jane.doe@example.com",
    }),
})
    .then((response) => {
        if (!response.ok) {
            throw new Error("Hálózati hiba történt");
        }
        return response.json();
    })
    .then((data) => {
        console.log("Sikeresen módosítva:", data);
    })
    .catch((error) => {
        console.error("Hiba:", error);
    });
```

PATCH Kérés

A PATCH kérések egy meglévő erőforrás részleges módosítására szolgálnak. Csak a megadott mezők frissülnek, a többi mező változatlan marad.

Példa egy PATCH kérésre a `fetch` API használatával:

```
fetch("https://api.example.com/resource/1", {
    method: "PATCH",
    headers: {
        "Content-Type": "application/json",
    },
    body: JSON.stringify({
        email: "jane.doe@newdomain.com",
    }),
})
    .then((response) => {
        if (!response.ok) {
            throw new Error("Hálózati hiba történt");
        }
        return response.json();
    })
    .then((data) => {
        console.log("Sikeresen frissítve:", data);
    })
    .catch((error) => {
        console.error("Hiba:", error);
    });
```

DELETE Kérés

A DELETE kérések egy meglévő erőforrás törlésére szolgálnak a szerverről.

Példa egy DELETE kérésre a `fetch` API használatával:

```
fetch("https://api.example.com/resource/1", {
    method: "DELETE",
    headers: {
        "Content-Type": "application/json",
    },
})
    .then((response) => {
        if (!response.ok) {
            throw new Error("Hálózati hiba történt");
        }
        return response.json();
    })
    .then((data) => {
        console.log("Sikeresen törölve:", data);
    })
    .catch((error) => {
        console.error("Hiba:", error);
    });
```

Async-Await Magyarázat

Mi az az Async-Await?

Az async-await a JavaScript nyelv két kulcsszava, amelyeket az ECMAScript 2017 (ES8) specifikációban vezettek be. Az async-await segítségével az aszinkron műveleteket könnyebben kezelhetjük, mivel szinkron kódhoz hasonló szerkezetet biztosítanak, ezzel javítva a kód olvashatóságát és karbantarthatóságát.

Az async Kulcsszó

Az `async` kulcsszó egy függvény előtt használva azt jelzi, hogy a függvény aszinkron műveleteket tartalmaz. Egy async függvény mindig egy Promise-t ad vissza.

Szintaxis:

```
async function myAsyncFunction() {
    // kód
}
```

Az await Kulcsszó

Az `await` kulcsszó az `async` függvényen belül használható. Az `await`-et egy Promise előtt használva azt jelzi, hogy a kódnak várnia kell, amíg a Promise teljesül, mielőtt folytatná a végrehajtást.

Szintaxis:

```
let result = await somePromise;
```

Hogyan Működik az Async-Await?

Az async-await használata során az `await` kulcsszó "megállítja" az async függvény végrehajtását, amíg a Promise teljesül vagy elutasításra kerül. Ez lehetővé teszi, hogy a kód szinkron stílusban írjuk, miközben a háttérben továbbra is aszinkron módon működik.

Példa: Async-Await Használata

Hagyományos Promise Kezelés

```
function getData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve("Adatok");
        }, 2000);
    });
}

getData()
    .then((data) => {
        console.log(data); // "Adatok"
    })
    .catch((error) => {
        console.error(error);
    });
```

Async-Await Használata

```
async function fetchData() {
    try {
        let data = await getData();
        console.log(data); // "Adatok"
    } catch (error) {
        console.error(error);
    }
}

fetchData();

function getData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve("Adatok");
        }, 2000);
    });
}
```

Fontos Pontok

1. **Async függvény mindig Promise-t ad vissza**: Ha egy async függvényen belül visszaadunk egy értéket, az érték egy teljesült Promise-ba lesz csomagolva.

   ```
   async function example() {
       return "Hello";
   }

   example().then((value) => console.log(value)); // "Hello"
   ```

2. **Await csak async függvényen belül használható**: Az `await` kulcsszó csak egy async függvényen belül használható, különben szintaxis hiba lép fel.

   ```
   async function example() {
       let result = await somePromise;
       console.log(result);
   }
   ```

3. **Hibakezelés**: Az async függvényekben a hibakezelés egyszerűen megoldható a try-catch blokkok használatával.

```
async function fetchData() {
    try {
        let data = await getData();
        console.log(data);
    } catch (error) {
        console.error(error);
    }
}
```

Gyakorlati Példa: Hálózati Kérés Async-Await Segítségével

Hagyományos Promise Kezelés Fetch API-val

```
fetch("https://api.example.com/data")
    .then((response) => {
        if (!response.ok) {
            throw new Error("Hálózati hiba történt");
        }
        return response.json();
    })
    .then((data) => {
        console.log(data);
    })
    .catch((error) => {
        console.error("Hiba:", error);
    });
```

Async-Await Használata Fetch API-val

```
async function getData() {
    try {
        let response = await fetch("https://api.example.com/data");
        if (!response.ok) {
            throw new Error("Hálózati hiba történt");
        }
        let data = await response.json();
        console.log(data);
    } catch (error) {
        console.error("Hiba:", error);
    }
}

getData();
```