Async in LWC

You might have seen something like this before:

```
async retrieveData() {
    await getAccounts({ country: this.country }).then(async accounts => {
        await findPartners({ accounts: accounts }).then(partners => {
            this.partners = partners;
        });
    });
}
```

This approach results in a callback nest that is difficult to understand and debug. On the next page, you will learn how to do it properly!

beyond the cloud

Use **async/await** when you want asynchronous code to behave like a synchronous one. JavaScript will pause function execution until the promise settles:

✓

```
async retrieveData() {
    try {
        const accounts = await getAccounts({ country: this.country });
        this.partners = await findPartners({ accounts: accounts });
    } catch (error) {
        console.error(error);
    }
}
```

If the data can be displayed later, or if you wish to perform another action only after the promise has been settled, utilize the **then/catch** block:

✓

```
retrieveAccounts() {
    getAccounts({ country: this.country })
        .then(accounts => {
            this.accounts = accounts;
        })
        .catch(error => {
            console.error(error);
        })
        .finally( () => {
            this.hideSpinner();
        })
}
```

Do not combine then/catch and async/await

It creates difficult to understand code, debugging is complicated and breaks the KISS principle

```
async function example() {
    return await myPromise().then(result => {
        console.log(result);
    });
}
```

```
async example() {
    const data = await myPromise();

    return data;
}
```

Avoid nesting promises

Instead of additional then/catch blocks, try to refactor the code into smaller blocks or use async/await instead.

✗

```
example(userId) {
    getAccount(userId).then(account => {
        return getPartner(account.Name).then(partner => {
            return getPartnerOrder(partner.OrderId).then(details => {
                this.partnerOrders = details;
            });
        });
    });
}
```

✓

```
async example(userId) {
    const account = await getAccount(userId);
    const partner = await getPartner(account.Name);
    this.partnerOrders = await getPartnerOrder(partner.OrderId);
}
```

beyond the cloud

Don't create new promises

In most cases, there is no reason to explicit create promises. And especially, don't wrap promises within a promise.

✗

```
example() {
    return new Promise((resolve, reject) => {
        asyncMethod
            .then(result => {
                resolve(result);
            })
            .catch(error => {
                reject(error);
            });
    })
}
```

✓

```
async example() {
    return await asyncMethod();
}
```

✓

```
example() {
    return asyncMethod().then(result => {
        console.log(result);
    });
}
```