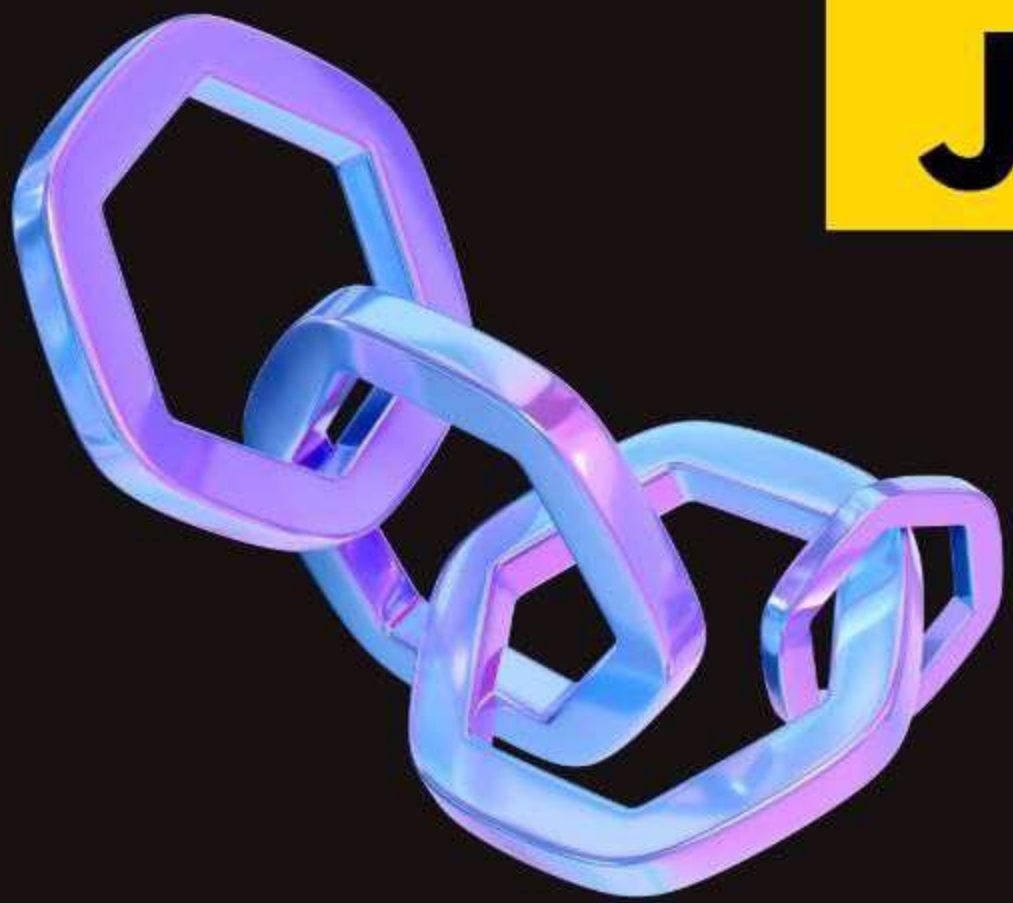


Chaining Promises in JavaScript

@new_javascript

What is a Promise?

Firstly, a Promise is a JavaScript object that represents the eventual completion or failure of an asynchronous operation. It can be in one of three states:

1. **Pending**: Initial state; neither fulfilled nor rejected.
2. **Fulfilled**: Completed successfully.
3. **Rejected**: Completed with an error.

Why Chain Promises?

Chaining promises is like creating a sequence of tasks where one task starts after the previous one finishes.

Instead of nesting callbacks (which leads to "callback hell"), we can make our code cleaner and more readable by chaining promises.

How to Chain Promises?

Chaining is made possible by the fact that the .then() and .catch() methods of a Promise always return a new Promise.

Example

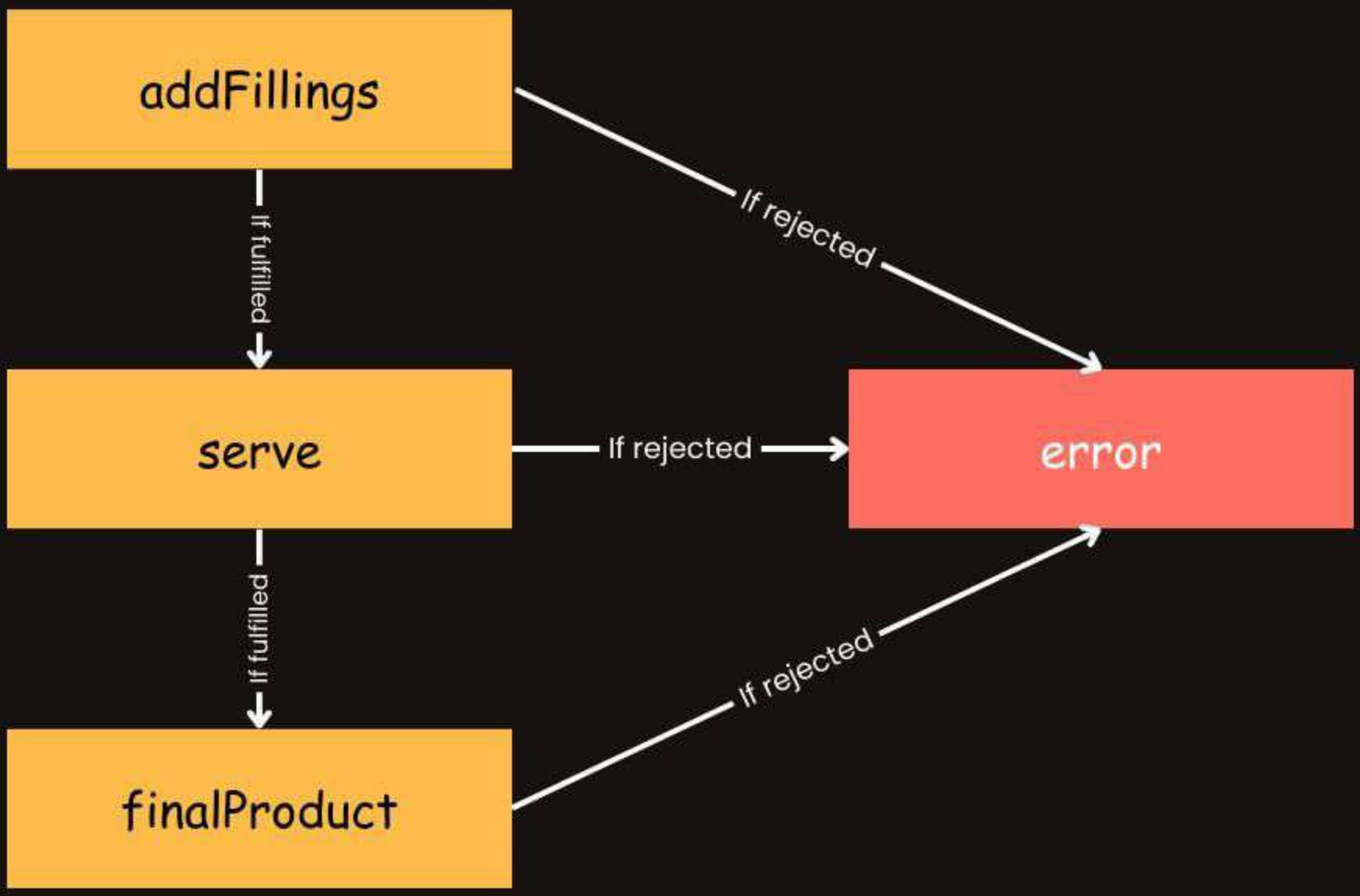
Imagine you're making a sandwich, and you have three steps:

1. Get bread.
2. Add fillings.
3. Serve the sandwich.

Each step must be done in order, and you can't start one until the previous is finished.

Let's assume each of these steps is an asynchronous task.

Here's a breakdown:



```javascript
function getBread() {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        console.log('Got the bread!');
        resolve('Bread');
      }, 1000);
    });
  }

  function addFillings(bread) {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        console.log('Added fillings!');
        resolve(bread + ' + Fillings');
      }, 1000);
    });
  }

  function serve(sandwich) {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        console.log('Sandwich is ready to eat!');
        resolve(sandwich);
      }, 1000);
    });
  }

  // Chaining the Promises
  getBread()
    .then(addFillings)
    .then(serve)
    .then(finalProduct => console.log(`Enjoy your ${finalProduct}!`))
    .catch(error => console.log(`Something went wrong: ${error}`));
```

Benefits of Chaining Promises

- **Readability**: Instead of nesting callbacks inside each other, we have a linear flow.
- **Error Handling**: With a single .catch() at the end, you can catch any rejection that happens at any point in the chain.