IBTISSAM OULAHCHOUM

JavaScript for Beginners Series

(مع شرح بالدارجة)

Part11

IBTISSAM OULAHCHOUM

Asynchronous JS & Promises

IBTISSAM OULAHCHOUM

Technical Explanation

By default, JavaScript is synchronous, meaning it runs one line of code at a time and must finish a task before moving to the next. If a task takes a long time (like fetching data from a server), it blocks the entire program.

جافاسكريبت فالأصل ديالو "متزامن" (synchronous)، يعني كينفذ الأوامر سطر بسطر، ومكيدوز للسطر الثاني حتى كيكمل الأول. إلا كانت شي عملية كتعطل (بحال تجيب معلومات من السيرفور)، البرنامج كامل كيتبلوكا.

Asynchronous code allows long-running tasks to happen in the background. A Promise is a special object that represents the eventual completion (or failure) of an asynchronous operation. It’s a placeholder for a future value.

الكود "اللامتزامن" (Asynchronous) كيخلي هاد العمليات الطويلة تخدم فالخلفية بلا ما تبلوكي كلشي. الـ Promise هو بحال شي "وعد"، هو واحد الكائن (object) كيمثل النتيجة المستقبلية ديال ديك العملية، سواء نجحات أو فشلات.

IBTISSAM OULAHCHOUM

Key Promise States

A Promise can be in one of three states:

- .pending: The initial state; the operation has not completed yet

pending (قيد الإنتظار): العملية مزال مسالاتش.

- .fulfilled (Resolved): The operation completed successfully, and the Promise now has a resolved value. We handle this with .then().

fulfilled (تم بنجاح): العملية نجحات وعندها نتيجة. كنتعاملو معاها بـ .then().

- .rejected: The operation failed. We handle this with .catch().

rejected (فشل): العملية فشلات. كنتعاملو معاها بـ .catch().

IBTISSAM OULAHCHOUM

```
// We create a new Promise that simulates fetching
data
const fetchData = new Promise((resolve, reject) => {
  const success = true; // Change to false to see the
.catch() work

  setTimeout(() => {
    if (success) {
      resolve("Data fetched successfully!"); // This
is the success value
    } else {
      reject("Error: Could not fetch data."); // This
is the error message
    }
  }, 2000); // Simulates a 2-second delay
});

console.log("Starting the fetch...");

// We use .then() and .catch() to handle the outcome
fetchData
  .then((successMessage) => {
    // This runs only if the promise is resolved
    console.log(successMessage); // Output after 2s:
"Data fetched successfully!"
  })
  .catch((errorMessage) => {
    // This runs only if the promise is rejected
    console.error(errorMessage);
  });

console.log("...code continues without blocking.");
```

IBTISSAM OULAHCHOUM

QUESTION FOR YOU:

- Besides fetching data, what is another real-world example where a task might take a long time and would be a perfect use case for a Promise?
- (Hint: Think about user actions).

IBTISSAM OULAHCHOUM

See you in other parts :)