Promesas, then y catch

En esta sección se definirá correctamente el resultado de una promesa al utilizar las funciones proporcionadas **resolve** y **request**. También se definirá un error personalizado en la consola para una promesa en el caso de que esta no se cumpla.

Definir una promesa

Las promesas en JavaScript son objetos que representan la finalización o el fracaso eventual de una operación asíncrona. Al crear una promesa, se proporciona un constructor **Promise** que contiene las funciones **resolve** y **reject** proporcionadas como parámetros. Estas funciones se utilizan para determinar el estado de la promesa.

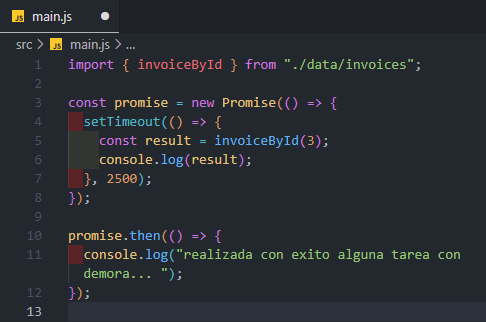
* **resolve:** Se utiliza para indicar que la promesa se ha cumplido con éxito y devuelve el valor esperado.
* **reject:** Se utiliza para indicar que la promesa ha fallado y devuelve un error.

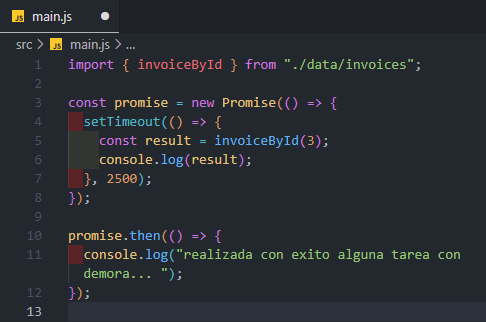
Sigue estos pasos para implementar una promesa:

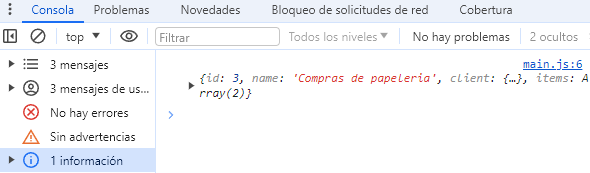
Paso 1: Obtener el resultado de la promesa fuera del método setTimeout

Para empezar, se tiene en cuenta los siguientes métodos asociados en una promesa, cada uno de estos métodos se definen mediante una función de flecha:

* **then:** Se ejecuta si la promesa se resuelve exitosamente.
* **catch:** Se ejecuta si la promesa se rechaza.
* **finally:** Se ejecuta siempre, independientemente de si la promesa se resuelve o se rechaza.







Paso 2: Mostrar el mensaje de la consola definido en el método then

En el paso anterior, el mensaje no se mostrará en la consola si solo se imprime el resultado en la promesa sin ejecutar el método **then**. Para que el **then** se ejecute correctamente, debes invocar y pasar algún argumento a la función **resolve**.

import { invoiceById } from "./data/invoices";

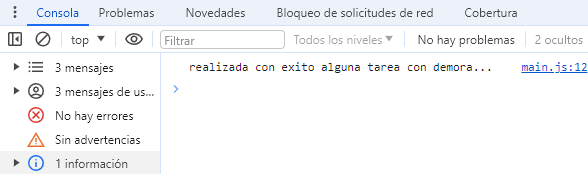
const promise = new Promise((resolve) => {
  setTimeout(() => {
    const result = invoiceById(3);
    // console.log(result);
    resolve();
  }, 2500);
});

promise.then(() => {
  console.log("realizada con exito alguna tarea con demora... ");
});


import { invoiceById } from "./data/invoices";

const promise = new Promise((resolve) => {
  setTimeout(() => {
    const result = invoiceById(3);
    // console.log(result);
    resolve();
  }, 2500);
});

promise.then(() => {
  console.log("realizada con exito alguna tarea con demora... ");
});

Paso 3: Pasar el resultado como argumento

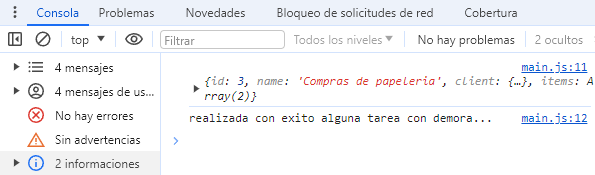
Como ya se había mencionado antes, para obtener el resultado en el **then** y realizar una acción con el resultado; primero se debe pasar el resultado como argumento a **resolve** y luego obtenerlo en el método **then**.

Ten en cuenta que el nombre definido en el parámetro de la función contenida en el método **then** (**"result"** en este caso) es arbitrario; puede ser cualquier nombre válido en JavaScript.

import { invoiceById } from "./data/invoices";

const promise = new Promise((resolve) => {
  setTimeout(() => {
    const result = invoiceById(3);
    resolve(result);
  }, 2500);
});

promise.then((result) => {
  console.log(result);
  console.log("realizada con exito alguna tarea con demora... ");
});

Definir el error de una promesa

Al trabajar con promesas en JavaScript, es común encontrarse con errores al consumir APIs de forma remota o al tener un retraso en la ejecución. Estos errores pueden incluir errores HTTP, errores de sistema, o errores de recurso no encontrado (como el error 404).

Paso 1: Buscar un objeto que no existe por su identificador

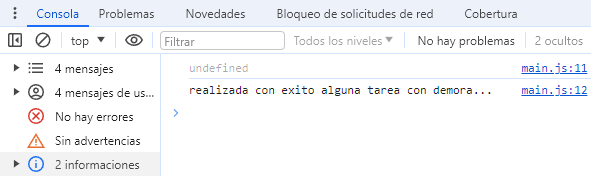
Para simular un error, se cambia el parámetro de la función **invoiceById** por un ID que no exista, como el **4**.

El resultado que se obtiene en la consola es **undefined**.

import { invoiceById } from "./data/invoices";

const promise = new Promise((resolve) => {
  setTimeout(() => {
    const result = invoiceById(4);
    resolve(result);
  }, 2500);
});

promise.then((result) => {
  console.log(result);
  console.log("realizada con exito alguna tarea con demora... ");
});

Paso 2: Mostrar un mensaje de error en la consola

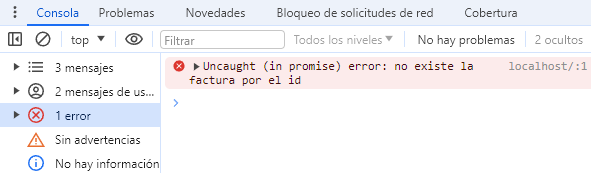
Para manejar este caso, se puede utilizar la función **reject** de la promesa. Antes de resolver la promesa, se verifica si el resultado existe mediante un condicional.

import { invoiceById } from "./data/invoices";

const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    const result = invoiceById(4);

    if (result) {
      resolve(result);
    } else {
      reject("error: no existe la factura por el id");
    }
    
  }, 2500);
});

promise.then((result) => {
  console.log(result);
  console.log("realizada con exito alguna tarea con demora... ");
});

Paso 3: Manejar la excepción en la promesa

Para manejar el error de una forma más amigable, procede a utilizar el método **catch** después del método **then**. El método **catch** se encarga de manejar el caso en que la promesa sea rechazada.

También se tiene en cuenta que los mensajes en la consola:

* **console.log** es para mostrar algo normal que está bien, como un registro
* **console.error** es para mostrar un registro de tipo error
* **console.warn** es para mostrar una advertencia.

import { invoiceById } from "./data/invoices";

const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    const result = invoiceById(4);

    if (result) {
      resolve(result);
    } else {
      reject("error: no existe la factura por el id");
    }
    
  }, 2500);
});

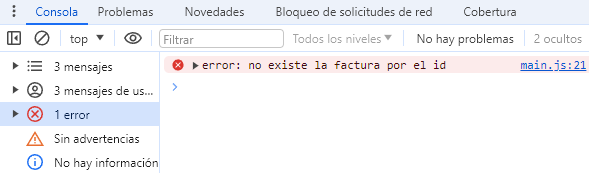
promise
  .then((result) => {
    console.log(result);
    console.log("realizada con exito alguna tarea con demora... ");
  })
  .catch((error) => {
    console.error(error);
  });


import { invoiceById } from "./data/invoices";

const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    const result = invoiceById(4);

    if (result) {
      resolve(result);
    } else {
      reject("error: no existe la factura por el id");
    }
    
  }, 2500);
});

promise
  .then((result) => {
    console.log(result);
    console.log("realizada con exito alguna tarea con demora... ");
  })
  .catch((error) => {
    console.error(error);
  });

Paso 4: Simplificar el código fuente

Las promesas se utilizan en aplicaciones frontend, como Angular o React, para manejar tareas asíncronas que tienen un retraso. Cuando se resuelve, se maneja con **then**, y si no, con **catch**.

Recordar que si el contenido una función ocupa una sola línea de código se puede simplificar.

import { invoiceById } from "./data/invoices";

const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    const result = invoiceById(4);

    if (result) {
      resolve(result);
    } else {
      reject("error: no existe la factura por el id");
    }
    
  }, 2500);
});

promise
  .then((result) => console.log(result))
  .catch((error) => console.error(error));


import { invoiceById } from "./data/invoices";

const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    const result = invoiceById(4);

    if (result) {
      resolve(result);
    } else {
      reject("error: no existe la factura por el id");
    }
    
  }, 2500);
});

promise
  .then((result) => console.log(result))
  .catch((error) => console.error(error));


En esta versión simplificada, el método **then** recibe directamente la función **console.log**, que se ejecutará con el resultado de la promesa cuando esta se resuelva. De igual manera, el método **catch** recibe la función **console.error**, que se ejecutará si la promesa es rechazada. Además, se ha cambiado el argumento que recibe la función **invoiceById**.

import { invoiceById } from "./data/invoices";

const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    const result = invoiceById(3);

    if (result) {
      resolve(result);
    } else {
      reject("error: no existe la factura por el id");
    }
  }, 2500);
});

promise.then(console.log).catch(console.error);