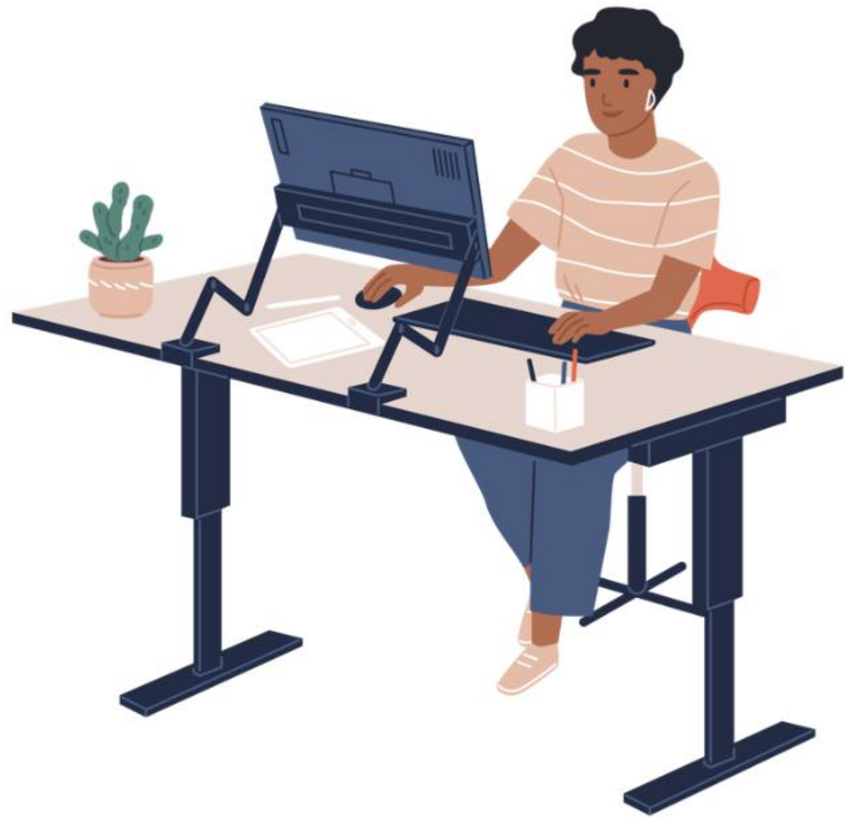
Slide Note

Learning Consolidation

Interact With Application Servers Using HTTP Protocol



Menu

00:01:00:00:26

01/ 26

Slide Note

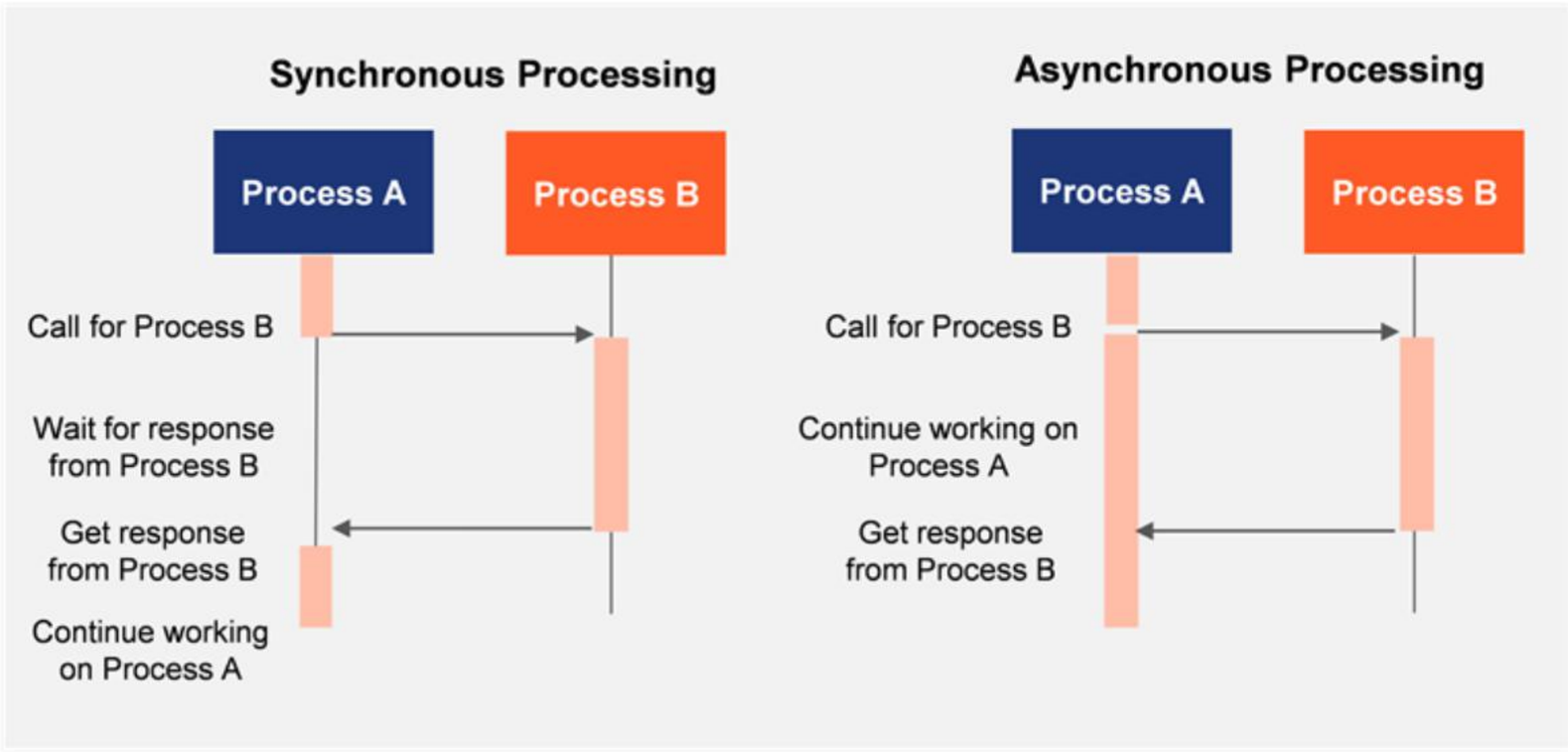
In this Sprint you learned to…

- Explain synchronous execution
- Explore the limitations of synchronous execution
- Explain asynchronous execution
- Implement asynchronous execution using Callbacks
- Implement asynchronous execution using Promise object
- Perform server operations using Axios API

Slide Note

Synchronous vs. Asynchronous Execution

- Synchronous execution: The second task will start executing only after the first task in a program finishes processing.
- Asynchronous Execution: Multiple tasks can be executed in parallel without waiting for an earlier task to be complete.



Slide Note

When Program Executes Synchronously

- Calls to an external code can result in a delayed response.
- This would block the subsequence of execution.
- It causes ‘freezing’ on the screen.
- This leads to an unresponsive user experience.
- This results in poor user engagement.

Slide Note

Asynchronous Programming

- JavaScript allows asynchronous programming.
- With this approach, wait time is utilized to proceed with subsequent instructions to execute.
- It helps provide non-blocking execution, reducing the latency.
- This helps improve user engagement.
- Examples:
  - Fetching stock market data,
  - Getting weather forecast data,
  - Sending emails
  - Timers used in online quizzes, etc.

Slide Note

Where Are Asynchronous Operations Required?

- I/O bound operations
- Downloading data from file
- CPU-bound operations
- Performing tasks in the background
- Extracting data from network
- Making REST API calls to an external server

Slide Note

Asynchronous Programming – What Needs to Be Done?

- In the case of asynchronous programming, the following questions need to be answered:
  - What is the asynchronous task that needs to be performed?
    - It could be fetching data from the server or running a timer in the background.
  - When should an asynchronous task be initiated?
    - It could be when the page loads, on button click, or when the user selects from given options in a drop-down list.
  - How do you initiate asynchronous task?
    - Based on the asynchronous task that needs to be performed, it would be required to identify the API that will help to make such requests.
  - What task should execute once the asynchronous task completes successfully?
    - It could be to display data fetched from server or notify the user about time out.
  - What task should execute in case when the asynchronous task completes with error?
    - It could be to display the error message to the user.

Slide Note

Asynchronous Programming – Design Considerations

- With asynchronous programming, implementation depends on the type of asynchronous activity.
- Based on the activity,
  - First, find out the technique to raise asynchronous requests.
    - In JavaScript, there are specialized web APIs that can help raise asynchronous requests.
      - In the case of timer events, use timer functions like `setTimeout()` or `setInterval()` to execute tasks asynchronously.
      - If the client server communication using REST API, use object like `XMLHttpRequest` or use APIs that make use of `XMLHttpRequest` object.
  - Next, find out how these APIs communicate the completion of the request.
    - Usually, they fire events on completion:
      - With timer functions, the time-out event gets fired.
      - With `XMLHttpRequest`, the event gets fired when the request processing state changes.

Slide Note

Asynchronous Programming – Design Considerations (cont'd)

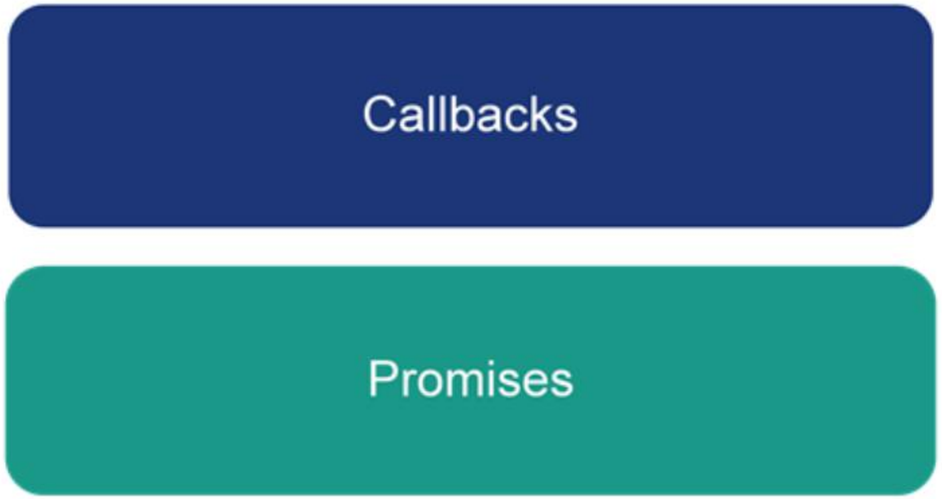
- In the next step, find out how these APIs need to be provided with the code that executes when:
  - The request completes.
  - The request completes with an error.
  - This could require the passing functions as parameters to the code that executes when the request completes – both successfully and with error.
- Once the above steps are clear, identify the approach.

Slide Note

With JavaScript, there are a couple of techniques that help to develop asynchronous code.

1. One makes use of Callbacks:
    1. It has its basis on the fact that functions are first-class citizens in JavaScript.
2. The other uses Promises:
    1. This is a popular concept, that helps manage responses that would be received in future.

Techniques for Asynchronous Programming



Slide Note

How Are Callbacks Used for Asynchronous Actions?

- The JavaScript host environment provides functions that help in asynchronous action.
- Examples of such functions:
  - `setInterval()`
  - `setTimeout()`
  - `loadScript()`
- These functions take a function as one of the parameters.
- This function is the callback, as it is "called back" once the action is completed.

Slide Note

Series of Timeout Calls: Callback Hell

Look at the code shown here.

- It has created a Callback Hell.
- Callback Hell is caused by coding with complex nested callbacks.
- Every callback takes an argument that is a result of the previous callbacks.
- Also known as ‘Pyramid of Doom’ since the code structure looks like a pyramid, making it difficult to read and maintain.

```
setTimeout(()=>{
    console.log(`1 sec lapsed`);
    setTimeout(()=>{
        console.log(`2 sec lapsed`);
        setTimeout(()=>{
            console.log(`3 sec lapsed`);
            setTimeout(()=>{
                console.log(`4 sec lapsed`);
                setTimeout(()=>{
                    console.log(`5 sec lapsed`);
                    setTimeout(()=>{
                        console.log(`6 sec lapsed`);
                    },1000);
                },1000);
            },1000);
        },1000);
    },1000);
},1000);
```

Callback Hell

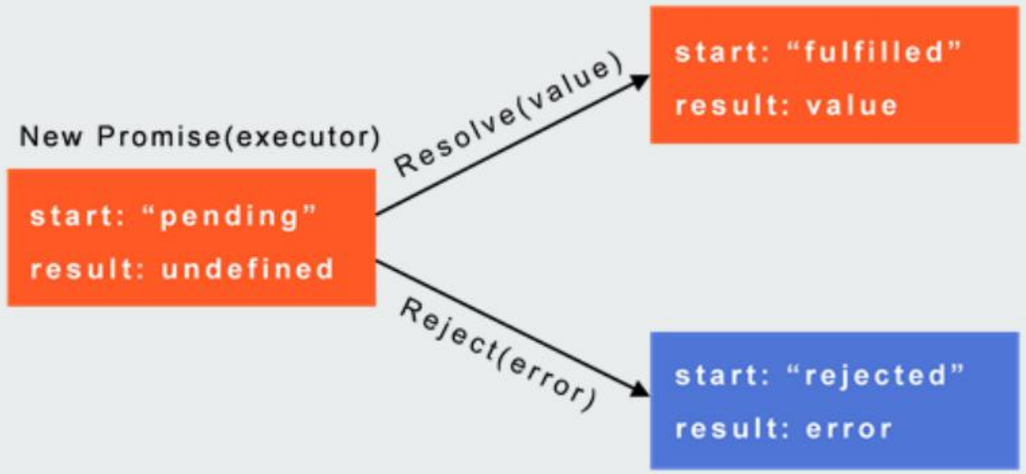
Slide Note

- Promise is an alternative to Callbacks.
- Promise is also a more effective way of handling asynchronous programming requirements.

Callback Hell: Concerns

- It is caused by coding with complex nested callbacks.
- Every callback takes an argument that is a result of the previous callbacks.
- It is also known as the 'Pyramid of Doom' since the code structure looks like a pyramid, making it difficult to read and maintain.
- The alternative approach is to use **Promise** object.
- Benefits of using Promise object:
  - No more Callback Hell
  - Better code readability
  - Efficient error handling

Slide Note



Creating Promise

- When created, Promise object takes the executor function as a parameter for its constructor.
- The executor function takes two parameters that are callbacks, popularly named resolve and reject.
- Resolve is called for successful fulfillment of promise.
- Reject is called for rejection of promise.
- The executor function can be used to make asynchronous requests.

Slide Note

Promise can be in one of the 3 states.

Pending: neither fulfilled nor rejected. This is the state when the promise is waiting for the action outcome.

Fulfilled: In this state, it indicates action was performed successfully. Causes the resolve() callback to be called.

Rejected: In this state, it indicates action failed. Causes reject() callback to be called.

Promise States

Pending

initial state, neither fulfilled nor rejected

Fulfilled

meaning that the operation was completed successfully

Rejected

meaning that the operation failed

Slide Note

A promise is a returned object to which you attach callbacks, instead of passing callbacks into a function.

Promises can be resolved or rejected.

Resolve handlers, registered with **then()** method, are called only when the action is successful.

To get the result of a promise, you can use its **then()** method.

To explicitly handle rejections, promises have a **catch()** method that registers a handler to be called when the promise is rejected.

To get the reason of rejection, you can use its **catch()** method.

Using Promises

then()

Used to attach resolve handlers

Called when the action is successful

Helps retrieve result of Promise

catch()

Used to attach reject handlers

Called when the action has failed

Helps retrieve reason of rejection of Promise

Slide Note

Asynchronous Scenario – Client Server Communication

- In case of client server communication in a web application, the client makes HTTP requests to the server through REST API, and the server returns HTTP response to the client through the API.
- JavaScript provides an `XMLHttpRequest` object that

Provides methods to:

- create request (GET, POST, PUT, DELETE)
- send request

Fires event when

- request processing state changes
- request completes
- request completes with error

Provides properties to

- know the state of request
- get the response

Slide Note

Asynchronous HTTP Request Execution With Promises

- The Axios API is a Promise-based HTTP client for the browser and Node.js.
- It allows you to make HTTP requests and provides the property to capture responses.
- **Axios** helps to make API calls and returns promises.
- Promises help to asynchronously process external interactions.
- It is an open-source library that allows you to make HTTP requests.
- It provides methods that include get(), post(), put() and delete().
- Successful call returns Promise with data.
- The failed call returns Promise with an error.
- To use Axios, in index.html or any .html file created for designing web page, include:

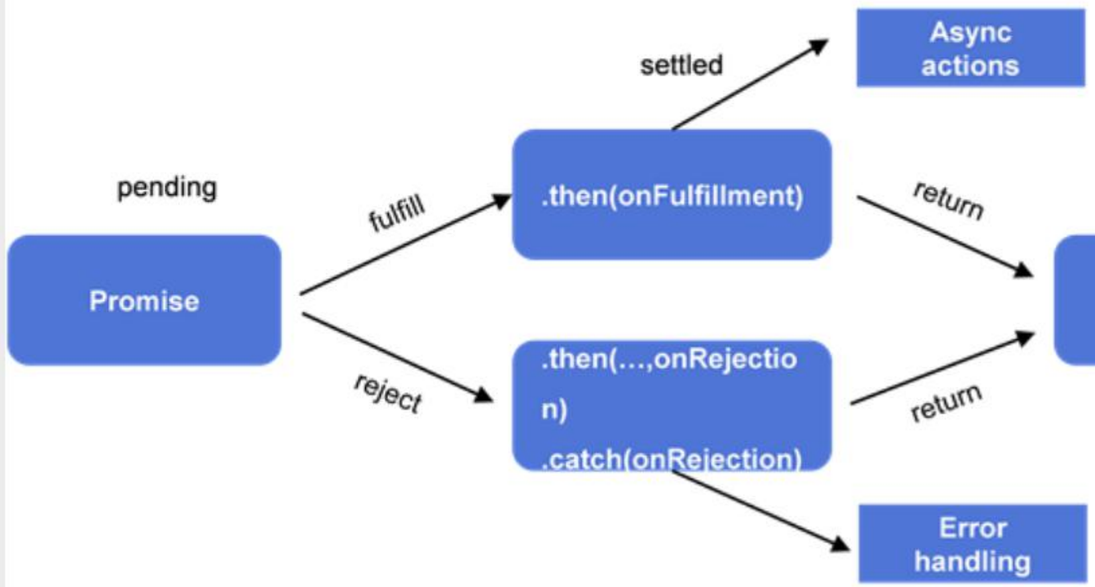
```
<script src="https://unpkg.com/axios/dist/axios.min.js "></script>
```

Slide Note

Chaining Promises

Promise.prototype.then() and Promise.prototype.catch() methods return Promises.

As a result, they can be chained.



Slide Note

Self-Check

Predict the output.

```
const promise = new Promise((res,rej)=>{
    rej(5);
});
promise
    .catch(v => {
        console.log(v);
        return v + 100;
    })
    .then(v => {
        console.log(v);
        return v * 2;
    })
    .then(v => {
        console.log(v);
        return v * 2;
    })
    .then(v => {
        console.log(v);
        return v*2;
    });
```

Slide Note

Self-Check: Solution

Predict the output.

```
const promise = new Promise((res,rej)=>{
    rej(5);
});
promise
    .catch(v => {
        console.log(v);
        return v + 100;
    })
    .then(v => {
        console.log(v);
        return v * 2;
    })
    .then(v => {
        console.log(v);
        return v * 2;
    })
    .then(v => {
        console.log(v);
        return v*2;
    });
```

Answer is

5
105
210
420

Since the Promise rejects, catch() is called first.

The returns from catch() and then() lead to subsequent then() block executions.

Slide Note

Self-Check

Predict the output.

```
let promise = new Promise(function(resolve, reject) {
  resolve(1);

  setInterval(() => resolve(2), 1000);
});

promise.then(alert);
```

Slide Note

Self-Check: Solution

Predict the output.

```
let promise = new Promise(function(resolve, reject) {
  resolve(1);

  setInterval(() => resolve(2), 1000);
});

promise.then(alert);
```

The output is: 1.

Explanation: The second call to resolve is ignored because only the first call of reject/resolve is considered. Once Promise has resolved or rejected , further calls are ignored.

Slide Note

Self-Check

What are the native JavaScript functions used to run code asynchronously?

1. `startInterval()`
2. `delay()`
3. `setTimeout()`
4. `interval()`
5. `setInterval()`
6. `timeout()`

Slide Note

Self-Check: Solution

What are the native JavaScript functions used to run code asynchronously?

1. `startInterval()`
2. `delay()`
3. **`setTimeout()`**
4. `interval()`
5. **`setInterval()`**
6. `timeout()`

Menu

00:25/00:00:26

25/ 26