Promises in JS:

A promise in JavaScript is an object that represents the eventual completion (or failure) of an asynchronous action and its resultant value. Promises provide a way to handle asynchronous code in a more synchronous and readable way.

A promise is in one of three states:

- Pending: initial state.
- Fulfilled: means the operation completion is a success.
- Rejected: means the operation has failed.

Promises have two main methods:

- .then(): is called when the promise is fulfilled and it returns a promise that is fulfilled with the result of the callback.
- .catch(): is called when the promise is rejected and it returns a promise that is fulfilled with the error.

For example, the following code uses a promise to load a JSON file:

```
fetch("data.json")
  .then(response => response.json())
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.log(error);
  });
```

In this example, fetch returns a promise that is fulfilled with a response object. The response object has a json() method that returns another promise that is fulfilled with the parsed JSON data. The then method is used to attach callbacks that are called when the promise is fulfilled and the catch method is used to handle any errors that occur during the loading of the JSON file.

Promises can also be combined using the Promise.all() and Promise.race() methods, which can be useful for working with multiple asynchronous operations.

Promises are widely used in JavaScript, and most modern JavaScript libraries and frameworks provide support for promises. They are a powerful tool for handling asynchronous code and provide a more manageable and readable way to handle the complexity of asynchronous operations.

Communication with Server using JS:

In JavaScript, communication with a server can be achieved using the XMLHttpRequest object or the newer Fetch API. These APIs allow JavaScript to send and receive data from a server using various methods such as GET, POST, PUT, DELETE, etc.

The XMLHttpRequest object, also known as XHR, is a built-in JavaScript object that allows for client-server communication. It is supported by most modern browsers.

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "http://example.com/data.json", true);
xhr.onreadystatechange = function() {
  if (xhr.readyState === 4 && xhr.status === 200) {
    let data = JSON.parse(xhr.responseText);
    console.log(data);
  }
};
xhr.send();
```

The Fetch API is a more recent addition to JavaScript and is considered to be a more modern and simpler alternative to XHR. It uses the fetch() function to make requests and returns a promise that resolves to a response object.

```
fetch("http://example.com/data.json")
  .then(response => response.json())
  .then(data => {
    console.log(data);
  });
```

It's also worth mentioning that there are libraries such as Axios and Superagent, that can simplify the process of making requests and handling responses.

It's important to note that browser-based JavaScript is subject to the same-origin policy, which means that it can only make requests to the same domain that served the web page. To make requests to other domains, a server-side proxy or CORS (Cross-Origin Resource Sharing) headers on the server are required.

Communicating with a server is a fundamental aspect of web development, and these APIs provide the means to retrieve data from a server and update it as well. Understanding how to use these APIs is essential for creating dynamic and interactive web applications.