The Client-Server Architectural Model

This lesson provides a high-level summary of the concept of the client-server architectural model.

WE'LL COVER THE FOLLOWING

Introduction

The crux of any communication over the Internet is the client-server architecture. The discussion on this topic is extensive and could fill a whole separate course. Therefore, in this course, we will keep the discussion brief and easy to understand, so you can grasp the concepts that we will study later on in this course.

So, what exactly are clients and servers?

> 💡 **The `client` and `server` are the main entities between which all communication of data over the Internet takes place.**

What is the client's role?

A client is any web browser or application on a desktop, a cellular device, or any other smart device. In fact, right now, you must be viewing this course through any such device. The role of the client is to send a request to the server and send back the required data. Simply put, when we type a URL into

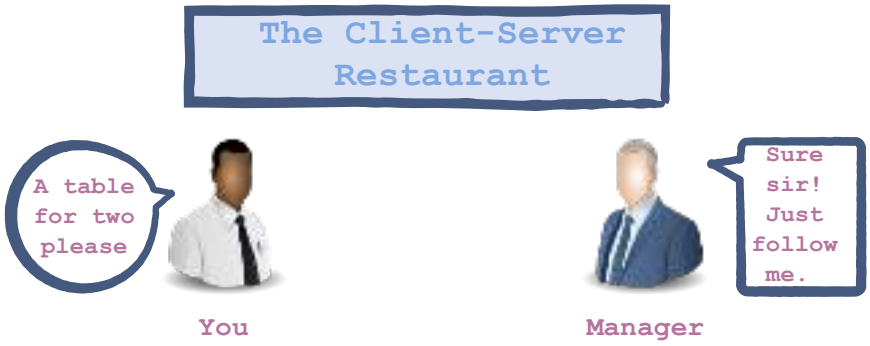
a browser or press a button, it sends a request to a server.

What is the server's role?

The primary role of a server is to wait for requests to arrive from clients and then respond to them. In other words, when a request from the client is received, the server determines the appropriate response to send back. Afterward, the server responds to these requests and sends back the information the client asked for.

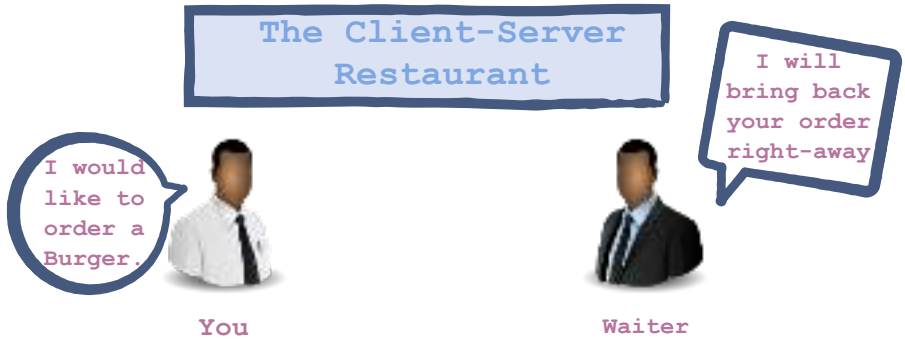
A real-world analogy

Consider this analogy: when we visit a restaurant, we go straight to the counter and ask to be seated. The manager listens to the request and directs us to a table.



Conversation between a client and a manager at a restaurant.

Then, we decide what to order and convey our requests to the waiter. Afterward, the waiter returns with the food that we ordered.



Conversation between a client and a waiter at a restaurant.

In the above analogy, we considered ourselves **the client** and the restaurant staff as **the server**. Each request we sent had an outcome, i.e., *a response*. The

response was dependent upon the request. In the above example, we first

made the request to be seated; therefore, the manager directed us to an empty table and also placed a menu for us. The manager interpreted our **request** and replied with an appropriate **response**. The same occurred in the other transaction when we asked the waiter to bring us the food. The waiter returned with the same dish we ordered. Had we asked for a different meal, we would have gotten a different response.

> 💡 **The `request` and `response` cycle is the *core* of all communication that takes place between a client and a server!**

How do the client and server communicate with each other?

The client and server communicate through multiple different **protocols**. In our context, the `HTTP` or `HTTPS` protocols are used. Think of the protocols as a language that they both understand. A client and a server adhere to a set of request-response messages to be sent in a specific order. For instance, you wouldn't enter a restaurant and ask for a cheque. This set of messages is known as a protocol. The discussion on the `HTTP` protocol is too extensive for this course. Therefore, it is a prerequisite that you have some basic knowledge about `HTTP` and its different methods:

- `GET`
- `POST`
- `PUT`
- `DELETE`

A Flask application as a server

The flask application that we will be learning to create in this course will be on the server-side. The browser will be acting as a client and sending `requests` to our web application. Then, the application will send back an appropriate `response`.

In the next lesson, we will go through a brief overview of the **MVC** and **MTV**

architecture.