Web stack animation narrative

Let us take a look at how the traditional Web stack works. It is a process which involves four components, or parties, each of which contributes its own functionality.

Two of the parties are the HTTP client and an HTTP server. The job of the HTTP client, which is often but certainly not always a web browser, is to issue HTTP requests to an HTTP server, which is typically located somewhere on the Internet. The HTTP server's job is to serve that HTTP request and send the response back to the HTTP client.

In case serving the request requires the involvement of a backend program, the HTTP server will pass on part of the request to the backend program, which itself may request the services of a backend database. If this all happens, at this point in the process we are four layers deep:

> from HTTP client to HTTP server; from HTTP server to backend program; and from backend program to database.

Once the database returns its data to the backend program, the program returns its results to the HTTP server, which in turn returns its response to the HTTP client.

In computing, this notion of going in several layers and then coming back from these layers one layer at the time, is known as a 'computational stack;' hence the term 'Web stack.' You may liken this to the HTTP client being the lowest brick in a stack of four bricks, with the HTTP server representing the next brick on the stack, and the backend program and database being the third and fourth the brick on the stack. Once the database has served its part of the request, it is taken off of the stack. Next, the backend program is done and is taken off of the stack, etc. So we build the stack up, brick by brick, and once at the top, we take it down again in the reverse direction. Another term for a process like this is a 'last-in, first-out' system or LIFO system.

Let's follow this LIFO system from start to end and see in a little more detail what each of the parties actually does. Our example considers us sending a request to the website *flights.com* requesting the availability of flights from Portland, Oregon to Denver, CO on December 25th 2019.

We model this example using an activity diagram. In an activity diagram, each actor involved in the modeled process is represented by a separate lane. In our case, each of the four actors in the Web stack gets its own lane in the diagram. The order of activities as executed by actors and the exchanges of information between actors are indicated by arrows.

- The process always starts with the HTTP client. Its job is to generate and send HTTP requests, wait for a response and then process the response based on its content.

  Our process diagram shows the three parts of the HTTP request: request line, optional request headers and, in this case, an empty body.

  The request line itself has its three parts: an HTTP method, a document address and the version of HTTP the client uses.

  Looking at the document address, we can already see that serving this request requires the involvement of a backend program, in this case a program written in the PHP language. We can paraphrase the document address as follows: “HTTP server at *flights.com*, please run the backend program *find_flights.php* and pass it the following variable/value pairs: the variable *orig* must be set to *PDX* (for Portland Oregon), the variable *dest* must be set to *DEN* (for Denver, Colorado) and the variable *date* must be set to the 25th of December 2019.”

Next, the HTTP client sends this request to the HTTP server at *flights.com* and waits for a reply.

- The HTTP server at flight.com receives the request and recognizes it as a request to run the program *find_flights.php,* passing it the variable/value pairs included in the document address. However, since its only job is to serve HTTP requests, not running PHP programs, it calls in the help of the PHP interpreter, that is the program that runs PHP programs, and passes the request to run the program and the three variable/value pairs to it. It then waits for the results to be returned to it.

- The PHP interpreter now goes to work and runs the *find_flights.php* program, using the three variable/value pairs as input. However, the information on flights from Portland to Denver on the 25th of December 2019, or for that matter for any and all flights on any and all dates, resides not in the PHP program but in a separate relational database. Hence, the PHP program, formulates a SQL query asking the database for flights from Portland to Denver on the 25th of December 2019 and fires the query off to the database.

  It is important to realize that at this point in the process three of the four involved parties are waiting. The HTTP client is waiting for the HTTP server, the server is waiting for the PHP interpreter and the PHP interpreter is waiting for the database to return the results of the SQL query. In ‘stack’ terms: the database is active and on top of the stack. All the other components are deeper in the stack and waiting for the layers above them to be taken off.

- Looking at the database, we see that it has a table *flights* which contains four flight records, two of which satisfy the criteria of the SQL query. Hence, the database extracts these two records from the *flights* table and sends them back to the PHP program.

- The PHP program now comes back alive, receives the two rows returned, extracts the data from those rows and, in this case, formats these results as HTML so that when they finally arrive back at the HTTP browser, they show up as a nice little table on someone’s screen.

  Once ready formatting the data, it sends the HTML back to the HTTP server which has been waiting for the results of the PHP program.

- The HTTP server now receives the HTML from *find_flights.php*, stores it in the body of an HTTP response and sends the response back to the waiting HTTP client.

- In case the HTTP client is a web browser, it receives the HTTP response, recognizes it to contain HTML, extracts the HTML from the body of the response and renders it as a nice little table on the screen.

We have now made one full Web stack return trip; from HTTP client, to server, to backend program to database and back. We built a four-layer stack and took it apart again on our way back. In reality this whole process typically only lasts less then a second or so, but it is good to have an appreciation of its underlying structure. Kinda nice, hey?