Back-End Web Architecture

This article provides an overview of servers, databases, routing, and anything else that happens between when a client makes a request and receives a response.

Software engineers seem to always be discussing the front-end and the back-end of their apps. But what exactly does this mean?

The front-end is the code that is executed on the client side. This code (typically HTML, CSS, and JavaScript) runs in the user's browser and creates the user interface.

The back-end is the code that runs on the server, that receives requests from the clients, and contains the logic to send the appropriate data back to the client. The back-end also includes the database, which will persistently store all of the data for the application. This article focuses on the hardware and software on the server-side that make this possible.

Review HTTP and REST if you want to refresh your memory on these topics. These are the main conventions that provide structure to the request-response cycle between clients and servers.

Let's start by reviewing the client-server relationship, and then we can start to put the pieces all together!

What are the clients?

The clients are anything that send requests to the back-end. They are often browsers that make requests for the HTML and JavaScript code that they will execute to display websites to the end user. However, there are many different kinds of clients: they might be a mobile application, an application running on another server, or even a web enabled smart appliance.

What is a back-end?

The back-end is all of the technology required to process the incoming request and generate and send the response to the client. This typically includes three major parts:

- The server. This is the computer that receives requests.

- The app. This is the application running on the server that listens for requests, retrieves information from the database, and sends a response.
- The database. Databases are used to organize and persist data.

What is a server?

A server is simply a computer that listens for incoming requests. Though there are machines made and optimized for this particular purpose, any computer that is connected to a network can act as a server. In fact, you will often use your very own computer as server when developing apps.

What are the core functions of the app?

The server runs an app that contains logic about how to respond to various requests based on the HTTP verb and the Uniform Resource Identifier (URI). The pair of an HTTP verb and a URI is called a *route* and matching them based on a request is called *routing*.

Some of these handler functions will be *middleware*. In this context, middleware is any code that executes between the server receiving a request and sending a response. These middleware functions might modify the request object, query the database, or otherwise process the incoming request. Middleware functions typically end by passing control to the next middleware function, rather than by sending a response.

Eventually, a middleware function will be called that ends the request-response cycle by sending an HTTP response back to the client.

Often, programmers will use a framework like Express or Ruby on Rails to simplify the logic of routing. For now, just think that each route can have one or many handler functions that are executed whenever a request to that route (HTTP verb and URI) is matched.

What kinds of responses can a server send?

The data that the server sends back can come in different forms. For example, a server might serve up an HTML file, send data as JSON, or it might send back only an HTTP status code. You've probably seen the status code "404 - Not Found" whenever you've tried navigating to a URI that doesn't exist, but there are many more status codes that indicate what happened when the server received the request.

What is a database, and why do we need to use them?

Databases are commonly used on the back-end of web applications. These databases provide an interface to save data in a persistent way to memory. Storing the data in a database both reduces the load on the main memory of the server CPU and allows the data to be retrieved if the server crashes or loses power.

Many requests sent to the server might require a database query. A client might request information that is stored in the database, or a client might submit data with their request to be added to the database.

What is a Web API, really?

An API is a collection of clearly defined methods of communication between different software components.

More specifically, a *Web API* is the interface created by the back-end: the collection of endpoints and the resources these endpoints expose.

A Web API is defined by the types of requests that it can handle, which is determined by the routes that it defines, and the types of responses that the clients can expect to receive after hitting those routes.

One Web API can be used to provide data for different front-ends. Since a Web API can provide data without really specifying how the data is viewed, multiple different HTML pages or mobile applications can be created to view the data from the Web API.

Other principles of the request-response cycle:

- The server typically cannot initiate responses without requests!
- Every request needs a response, even if it's just a 404 status code indicating that the content was not found. Otherwise your client will be left *hanging* (indefinitely waiting).
- The server should not send more than one response per request. This will throw errors in your code.

Mapping out a request

Let's make all of this a bit more concrete, by following an example of the main steps that happen when a client makes a request to the server.

1. Alice is shopping on SuperCoolShop.com. She clicks on a picture of a cover for her smartphone, and that click event makes a GET request to `http://www.SuperCoolShop.com/products/66432`.

Remember, GET describes the kind of request (the client is just asking for data, not changing anything). The URI (uniform resource identifier) `/products/66432` specifies that the client is looking for more information about a product, and that product, has an id of 66432.

SuperCoolShop has an huge number of products, and many different categories for filtering through them, so the actual URI would be more complicated than this. But this is the general principle for how requests and resource identifiers work.

2. Alice's request travels across the internet to one of SuperCoolShop's servers. This is one of the slower steps in the process, because the request cannot go faster than the speed of light, and it might have a long distance to travel. For this reason, major websites with users all over the world will have many different servers, and they will direct users to the server that is closest to them!

3. The server, which is actively listening for requests from all users, receives Alice's request!

4. Event listeners that match this request (the HTTP verb: GET, and the URI: `/products/66432`) are triggered. The code that runs on the server between the request and the response is called *middleware*.

5. In processing the request, the server code makes a database query to get more information about this smartphone case. The database contains all of the other information that Alice wants to know about this smartphone case: the name of the product, the price of the product, a few product reviews, and a string that will provide a path to the image of the product.

6. The database query is executed, and the database sends the requested data back to the server. It's worth noting that database queries are one of the slower steps in this process. Reading and writing from static memory is fairly slow, and the database might be on a different machine than the original server. This query itself might have to go across the internet!

7. The server receives the data that it needs from the database, and it is now ready to construct and send its response back to the client. This response body has all of the information needed by the browser to show Alice more details (price, reviews, size, etc) about the phone case she's interested in. The response header will contain an HTTP status code 200 to indicate that the request has succeeded.

8. The response travels across the internet, back to Alice's computer.

9. Alice's browser receives the response and uses that information to create and render the view that Alice ultimately sees!