The high-level goal of a server is this:

1. Accept a request
2. Process the request
3. Send a response

Many*web servers*(i.e. Apache, Nginx, etc) are used primarily for just accepting requests. They can target specific ports and protocols to receive these requests.

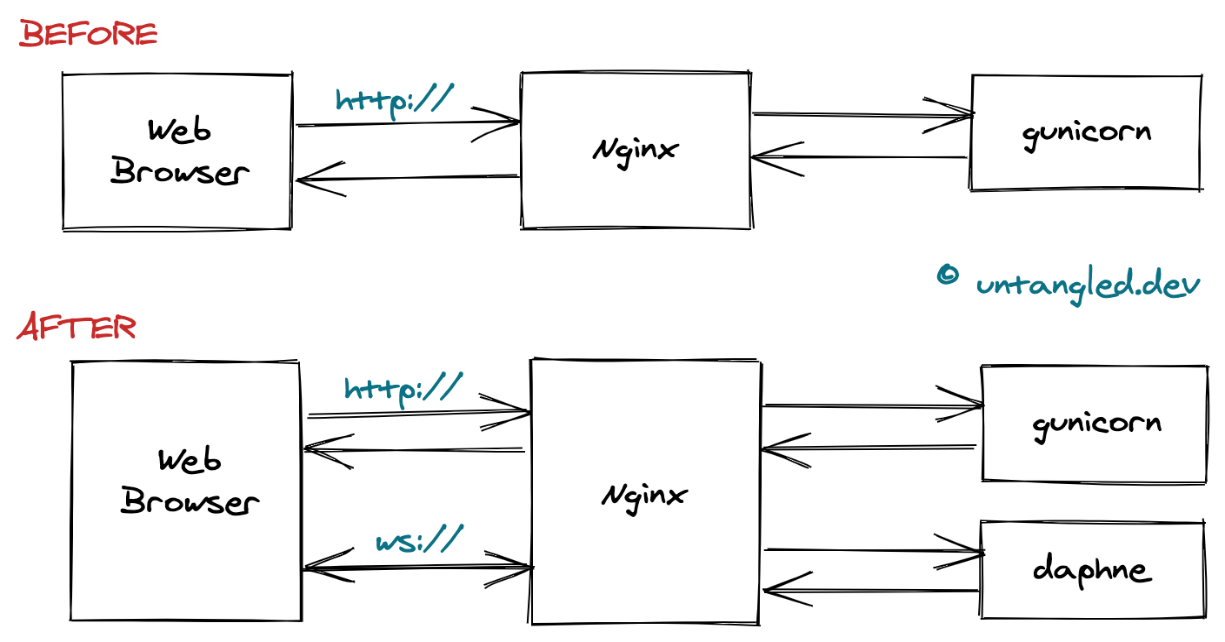
Next, there are *web applications*, which take data from the request, acts on it, and prepares data for the response. These applications can be written in any language as long as there is the necessary "glue" between it and the web server.

For example, Django is a web application framework that can use WSGI glue to bind itself to an apache web server. In other words, WSGI connects two different pieces of software so they play nicely. As a result, we have 2 modular systems, which is nice.

Now, you are probably asking: what is this glue? The glue is not another piece of software; the glue is an **agreement between the web server and the web application**.

*The story of Apache, WSGI, Django, and a Request:*

**Apache**: "I just received a request! Hey WSGI, I need you to prepare Django over there to handle this request"  
**WSGI**: "Okey Doke. I'll set up the environment, take the request, and pass it along to Django."  
**Django**: "Thanks WSGI! Give me a second and I'll have a response for you."  
**WSGI**: "Alright, I'll just wait right here."  
**Django**: "Okay, done! Here is my response to that request. Go give it to Apache"  
**WSGI**: "Cool! Apache, I have a response for you. I'm giving it to you just the way you like it!"  
**Apache**: "Perfect, I got it. Responding the requester now. Good work team!"



**Why use the WSGI rather than directly pointing the web server to the Django or Flask application?**

If you directly point your web server to your application, it reduces the **flexibility**of your application. Since your web server now directly points to your web application, you are unable to swap out web stack components. Now, let’s have a look at an example to make you clear about the applicability of WSGI. For instance, today you have decided to deploy your application using Gunicorn but after some years you decide to switch from Gunicorn to mod\_wsgi. Now, in this case, you can easily switch to mod\_wsgi without making any changes in the application or framework that implements WSGI. Hence, WSGI provides flexibility to your application.

Another reason for using WSGI is due to its **scalability**. Once your application is live, up and running there can be thousands of requests in your application. Hence, WSGI is capable of serving thousands of requests at a time. As we know, the WSGI server is responsible for handling the requests from the web server and takes decision for carrying out the communication of those requests to an application framework’s process. Here, we can divide the responsibilities among the servers for scaling web traffic.

HTTP is a set of protocols designed to enable communication between clients and servers. It works as a request-response protocol between a client and server.

**HttpResponseNotModified** (source code) returns the HTTP 304 status code from a Django web application view. The HTTP 304 status code indicates that the resource has not been modified since the client last requested it.

So, to request a response from the server, there are mainly two methods:

1. **GET** : to request data from the server.
2. **POST** : to submit data to be processed to the server.

