Introduction to Web Application Development Technologies

Technology stack for web application development

- To develop a web application, you need to select the server, database, programming language, framework, and frontend tools. These web development technologies build on top of each other and are, in fact, collectively called a stack.
- Let’s take a look at the following terminologies and concepts,
  - Frameworks
  - Databases
  - Client and server
  - Front-end and back-end
  - Client-side programming
  - Server-side programming
  - Web Application Architecture

Frameworks

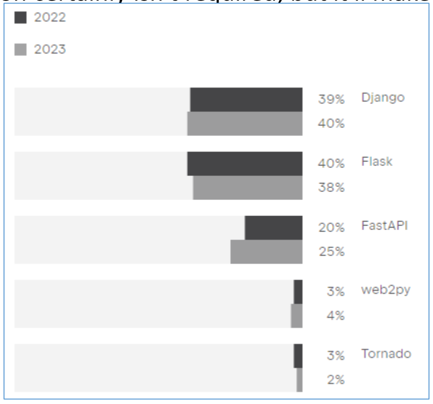
- A **framework** is a big library or group of libraries that provides many services (rather than perhaps only one focused ability as most libraries do).
- Frameworks typically take all the difficult, repetitive tasks in setting up a new web application and either does them for you or make them very easy for you to do.
- Examples:
  - Django - a full-stack server-side web application framework built using python
  - Ruby on Rails - a full-stack server-side web application framework built using ruby
  - Bootstrap - a free and open-source CSS framework directed at responsive, mobile-first front-end web development with HTML, CSS and Javascript
  - Laravel **-** a free, open-source PHP web framework, following the model–view–controller (MVC) architectural pattern.
  - ASP.NET **–** a server-side web application framework developed by Microsoft.

Common web framework functionality

- Frameworks provide functionality to perform common operations required to run web applications, such as:
  - URL routing
  - HTML, XML, JSON, and other output format templating
  - Database manipulation
  - Security against Cross-site request forgery (CSRF) and other attacks
  - Session storage and retrieval
- Not all web frameworks include code for all of the above functionality.
- For example, the Django web application framework includes an Object-Relational Mapping (ORM) layer that abstracts relational database read, write, query, and delete operations. However, Django's ORM cannot work without significant modification on non-relational databases such as MongoDB.
- Some other web frameworks such as Flask and Pyramid are easier to use with non-relational databases by incorporating external Python libraries.

Do I have to use a web framework?

- Whether or not you use a web framework depends on your experience with web development and what you're trying to accomplish.
- Using a web framework to build a web application certainly isn't required, but it'll make most developers' lives easier in many cases.



https://www.jetbrains.com/lp/devecosystem-2023/python/

Databases

- Your web application needs a place to store its data, and that’s what a **database** is used for.
- There are two main types of databases: relational and non-relational databases
- Examples:
  - MongoDB - an open-sourced NoSQL database.
  - PostgreSQL - a popular open-sourced SQL database.
  - MySQL - another popular open-sourced SQL database.
  - Oracle - an enterprise SQL database.
  - SQL Server - an SQL server manager created by Microsoft.

Client (or Client-side)

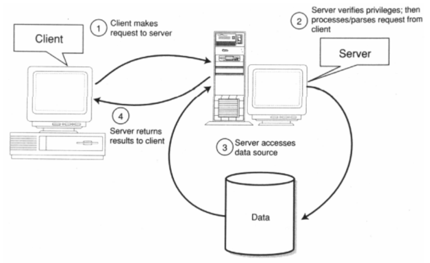
- This party *requests* pages from the **Server**, and displays them to the user. In most cases, the client is a **web browser**.
  - The **User** - The user *uses* the **Client** in order to surf the web, fill in forms, watch videos online, etc.
  - It’s you and me when we visit http://google.com.
- Client can be desktop computers, tablets, or mobile devices.
- There are typically multiple clients interacting with the same application stored on a server.

Server (or Server-side)

- This party is responsible for **serving** pages.
- A server is where the application code is typically stored.
- Requests are made to the server from clients, and the server will gather the appropriate information and respond to those requests.

Typical Client-Server Model

1. The **User** opens his web browser (the **Client**).
2. The **User** browses to http://google.com.
3. The **Client** (on the behalf of the **User**), sends a request to http://google.com (the **Server**), for their home page.
4. The **Server** then acknowledges the request, and replies the client with some meta-data (called *headers*), followed by the page's source.
5. The **Client** then receives the page's source, and *renders* it into a human viewable website.


Client
1 Client makes request to server
2 Server verifies privileges; then processes/parses request from client
Server
4 Server returns results to client
3 Server accesses data source
Data

https://www.troon.org/middleware/

Front-end & Back-end

- The front-end is comprised of HTML, CSS, and JavaScript. This is how and where the website is shown to users.
- The back-end is comprised of your server and database. It’s the place where functions, methods, and data manipulation happens that you don’t want the clients to see.

Client-side programming

- Client-side programming is writing code that will run on the client, and is done in languages that can be executed by the browser, such as JavaScript.
- Client-side (i.e. frontend) web development involves everything users see on their screens. Some major frontend technology stack components:
  - Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS).
    - HTML tells a browser how to display the content of web pages,
    - CSS styles that content.
    - Bootstrap is a helpful framework for managing HTML and CSS.
  - JavaScript (JS): making web pages interactive.
    - There are many JavaScript libraries (such as jQuery, Vue.js, React.js - maintained by Meta) for faster and easier web development.

Typical uses of client-side programming

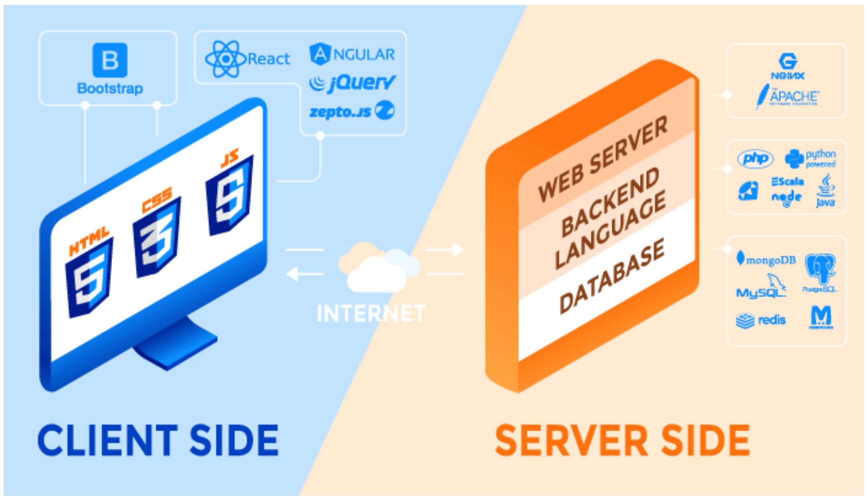
- **Client side** programming has mostly to do with the user interface, with which the user interacts.
- **Its main tasks are:**
  - Validating input (Validation must be done in the server. A redundant validation in the client could be used to avoid server calls when speed is very critical.)
  - Animation
  - Manipulating UI elements
  - Applying styles
  - Some calculations are done when you don't want the page to refresh so often

Server-Side Programming

- The server side isn't visible to users, but it powers the client side, just as a power station generates electricity for your house.
- Server-side programming is writing code that runs on the server, using languages supported by the server.
- Server-side programming is used to create the logic of websites and applications.
- Frameworks for programming languages offer lots of tools for simpler and faster coding. Some of the popular programming languages and their major frameworks (in parentheses):
  - Ruby (Ruby on Rails)
  - Python (Django, Flask, Pylons)
  - PHP (Laravel)
  - C# (ASP.NET)

Typical uses of server-side programming

- Process user input.
- Display pages.
- Structure web applications.
- Interact with permanent storage (SQL, files).
  - Querying the database
  - Insert and update information onto the database

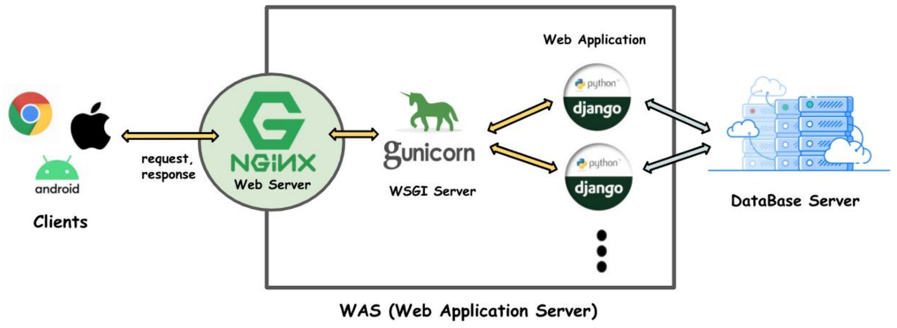


https://www.upwork.com/hiring/for-clients/how-to-choose-a-technology-stack-for-web-application-development/

15

Production environment: An example

A production environment where we will incorporate **Gunicorn** and **Nginx** as our **Server and Proxy server** respectively for the application.



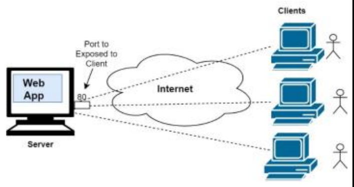
https://moondol-ai.tistory.com/467

16

Notes on HTTP

HTTP

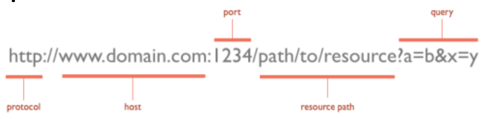
- HTTP stands for Hypertext Transfer Protocol.
- It's a stateless, application-layer protocol for communicating between distributed systems.
- The communication usually takes place over TCP/IP, but any reliable transport can be used. The default port for TCP/IP is **80**, but other ports can also be used.
- Communication between a host and a client occurs, via a **request/response pair**.
- The client initiates an HTTP request message, which is serviced through a HTTP response message.



https://nimesha-dilini.medium.com/simple-introduction-to-client-server-architecture-concept-7d2979bed31d

URLS

- At the heart of web communications is the request message, which are sent via Uniform Resource Locators (URLs).
- URLs have a simple structure that consists of the following components:

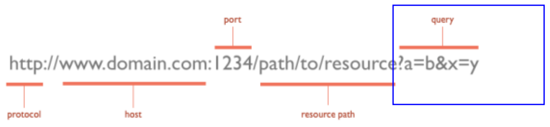


  - The protocol is typically http, but it can also be https for secure communications.
  - The default port is 80, but one can be set explicitly
  - The resource path is the *local path* to the resource on the server.

19

Request

- URLs reveal the identity of the particular host with which we want to communicate.
- The action that should be performed on the host is specified via HTTP verbs. The common request verbs are:
  - **GET**: *fetch* an existing resource. The URL contains all the necessary information the server needs to locate and return the resource.



  - **POST**: *create* a new resource. POST requests usually carry a payload that specifies the data for the new resource.

20

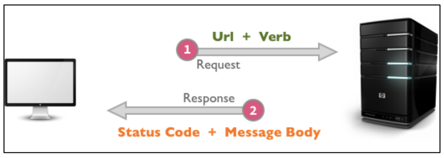
Status Codes

- With URLs and verbs, the client can initiate requests to the server. In return, the server responds with status codes and message payloads.
- Some of the common types of responses:
  - 2xx: Successful
    This tells the client that the request was successfully processed. The most common code is 200 OK. For a GET request, the server sends the resource in the message body.
  - 4xx: Client Error
    These codes are used when the server thinks that the client is at fault. The common codes in this class include:
    - **400** Bad Request: the request was malformed.
    - **403** Forbidden: server has denied access to the resource.
    - 404 **Not Found:** indicates that the resource is invalid and does not exist on the server.

21

Request and Response Message Formats

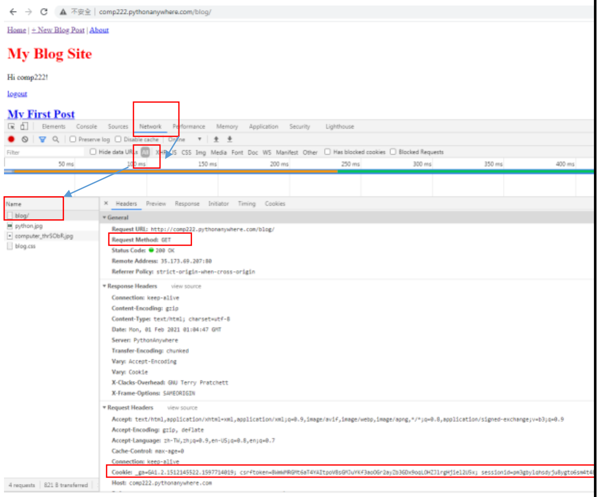
- *Verbs and status codes* make up the fundamental pieces of an HTTP request/response pair.



22

Viewing HTTP headers

- In Chrome, right click to select "Inspect" from the menu to bring up the Chrome developer tools.
- Choose the "Network" tab.
- Initially, it is possible the page data is not present/up to date. Hit Ctrl+R to record the reload.
- Observe the page information appears in the listing. Make sure "All" is selected next to the "Hide data URLs" checkbox.


comp222.pythonanywhere.com/blog/

Home | + New Blog Post | About

My Blog Site

Hi comp222!

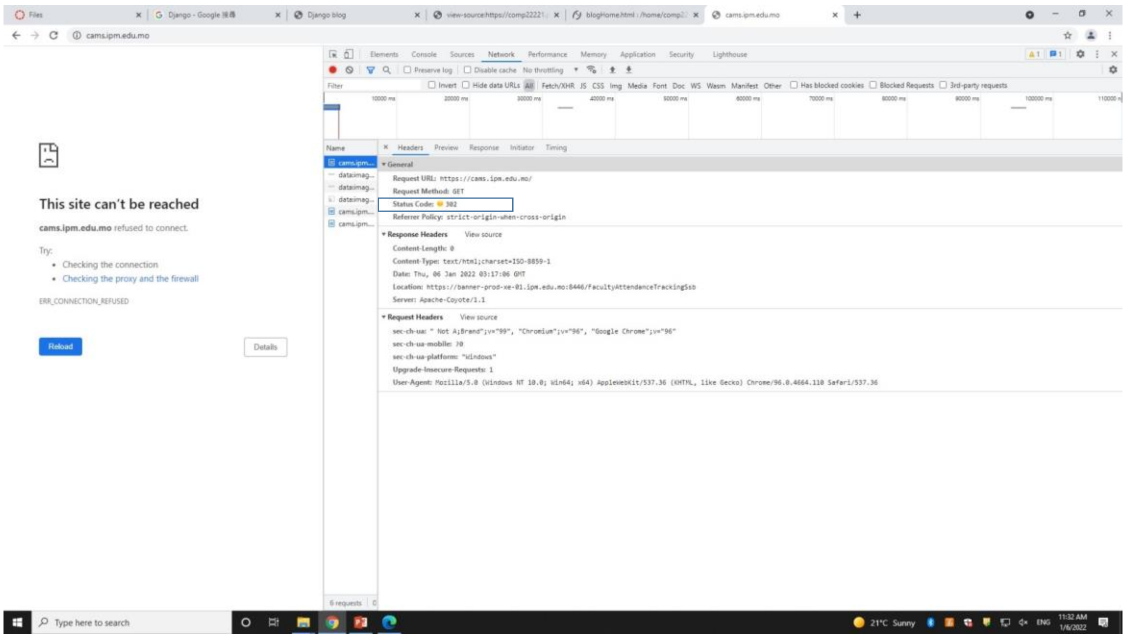
logout

My First Post

```
General
  Request URL: http://comp222.pythonanywhere.com/blog/
  Request Method: GET
  Status Code: 200 OK
  Remote Address: 35.173.69.207:80
  Referrer Policy: strict-origin-when-cross-origin
Response Headers
  Connection: keep-alive
  Content-Encoding: gzip
  Content-Type: text/html; charset=utf-8
  Date: Mon, 01 Feb 2021 01:04:47 GMT
  Server: PythonAnywhere
  Transfer-Encoding: chunked
  Vary: Accept-Encoding
  Vary: Cookie
  X-Clacks-Overhead: GNU Terry Pratchett
  X-Frame-Options: SAMEORIGIN
Request Headers
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
  Accept-Encoding: gzip, deflate
  Accept-Language: zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7
  Cache-Control: max-age=0
  Connection: keep-alive
  Cookie: _ga=GA1.2.1512145522.1597714019; csrftoken=8WmMRGMt6aT4YAItpoV8sGMJuYKf3aoOGr2ayZb3GDx9oqLOHZJlrgHj1el2USx; sessionid=pm3gbylghsdyju8ygto6sm4t4
  Host: comp222.pythonanywhere.com
```

Elements Console Sources Network Performance Memory Application Security Lighthouse
Preserve log Disable cache Online
Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other Has blocked cookies Blocked Requests
Name
blog/
python.jpg
computer_thrSObR.jpg
blog.css
Headers Preview Response Initiator Timing Cookies
4 requests | 821 B transferred

23


This site can't be reached

cams.ipm.edu.mo refused to connect.

Try:
- Checking the connection
- Checking the proxy and the firewall

ERR_CONNECTION_REFUSED

```
General
  Request URL: https://cams.ipm.edu.mo/
  Request Method: GET
  Status Code: 302
  Referrer Policy: strict-origin-when-cross-origin
Response Headers
  Content-Length: 0
  Content-Type: text/html;charset=ISO-8859-1
  Date: Thu, 06 Jan 2022 03:17:06 GMT
  Location: https://banner-prod-xe-01.ipm.edu.mo:8446/FacultyAttendanceTrackingSsb
  Server: Apache-Coyote/1.1
Request Headers
  sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"
  sec-ch-ua-mobile: ?0
  sec-ch-ua-platform: "Windows"
  Upgrade-Insecure-Requests: 1
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36
```

Files
Django - Google 搜尋
Django blog
view-source:https://comp2221
blogHome.html - /home/comp2
cams.ipm.edu.mo
Reload
Details
Elements Console Sources Network Performance Memory Application Security Lighthouse
Preserve log Disable cache No throttling
Invert Hide data URLs All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other Has blocked cookies Blocked Requests 3rd-party requests
Name
cams.ipm...
dataimag...
Headers Preview Response Initiator Timing
6 requests
Type here to search
21°C Sunny
ENG
11:32 AM
1/6/2022

24