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:
  - Meteor - a full-stack (front and back end) javascript framework
  - Node.js - a server-side javascript framework
  - Ruby on Rails - a full-stack framework built using ruby
  - Django - a full-stack framework built using python
  - Bootstrap - a UI (user interface) framework for building with HTML/CSS/Javascript
  - Angular.js - a front-end javascript framework.
  - Laravel **-** a free, open-source PHP web framework, following the model–view–controller (MVC) architectural pattern.
  - ASP.NET **-** an open-source server-side web application framework developed by Microsoft.

Common web framework functionality

- Frameworks provide functionality in their code or through extensions to perform common operations required to run web applications. These common operations include:
  - 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. There is a spectrum between minimal functionality with easy extensibility on one end and including everything in the framework with tight integration on the other end.

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.
- If you are a beginner programmer and just want to work on a web application as a learning project then a framework can help you understand the concepts listed above, such as URL routing, data manipulation and authentication that are common to the majority of web applications.
- On the other hand, if you're an experienced programmer with significant web development experience you may feel like the existing frameworks do not match your project's requirements. In that case, you can mix and match open source libraries with your own code to create your own framework.
- In short, whether or not you need to use a web framework to build a web application depends on your experience 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.

Databases

- Your web application needs a place to store its data, and that’s what a **database** is used for.
- There are two types of databases: relational and non-relational databases
- Relational databases provides more structure which helps with making sure all the data is correct and validated.
- Non-relational databases (NoSQL) provides a lot of flexibility for building and maintaining applications.
- Examples:
  - MongoDB - an open-sourced NoSQL database and is currently the only database supported by Meteor.
  - 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.
6. The **User** types something into the search bar, and presses Enter.
7. The **Client** submits that data to the **Server**.
8. The **Server** processes that data, and replies with a page matching the search results.
9. The **Client**, once again, renders that page for the **User** to view.

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. Here are the 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, while CSS styles that content. Bootstrap is a helpful framework for managing HTML and CSS.
  - JavaScript (JS). JS makes web pages interactive. There are many JavaScript libraries (such as jQuery, React.js) and frameworks (such as Angular, Backbone) 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. In web development it's the browser, in the user's machine, that runs the code, and it's mainly done in **javascript, flash,** etc. This code must run in a variety of browsers.
- **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.
- For server-side programming, they are 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)

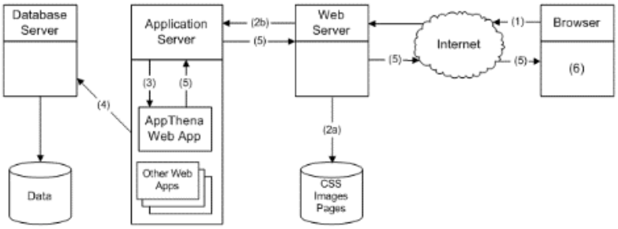
Typical uses of server-side programming

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

Web Application Architecture

- A web application is a client-server application, where there's a browser (the client) and a web server.
- The logic of a web application is distributed among the server and the client, there's a channel for information exchange, and the data is stored mainly on the server.
- Further details depend on the architecture: different ones place and distribute the logic in different ways.

Web Application Architecture

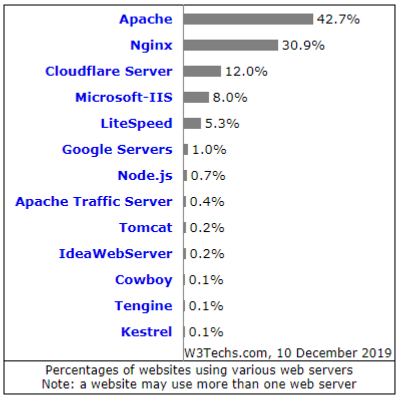


Web Server

- A web server is a system that delivers content or services to end users over the internet. A web server consists of a physical server, server operating system (OS) and software used to facilitate HTTP communication.
- **Web servers** are computers that deliver (serves up) Web pages. Every **Web server** has an IP address and possibly a domain name.
- The primary function of a web server is to store, process and deliver web pages to clients.

Web Server - market share

- For reference, below is the latest statistics of the *market share of all sites* of the top web servers on the Internet by W3Techs, as of December 2019.



W3Techs.com, 10 December 2019

Percentages of websites using various web servers
Note: a website may use more than one web server

Application Server

- A Web server exclusively handles HTTP requests, whereas an application server serves business logic to application programs through any number of protocols.
- An application server is a program that handles all application operations between users and an organization's backend business applications or databases.
- The purpose of an application server is to provide software abstractions for commonly used services. Many application servers accept network requests from Web browsers and manage connections to large databases. Typically found in business environments, application servers often run on the same network hardware as *Web servers*.
- Most application servers also contain a Web server, meaning you can consider a Web server a subset of an application server.
- Over time, application servers and web servers have morphed from two previously distinct categories, blended features, and arguably have merged.
- Examples:
  - IBM WebSphere Application Server
  - The Oracle WebLogic 11g application server

Difference between Web Server and Application Server: An Example

- Consider an online store that provides real-time pricing and availability information. Most likely, the site will provide a form with which you can choose a product. When you submit your query, the site performs a lookup and returns the results embedded within an HTML page. The site may implement this functionality in numerous ways.

Difference between Web Server and Application Server: An Example

Scenario 1: Web server without an application server

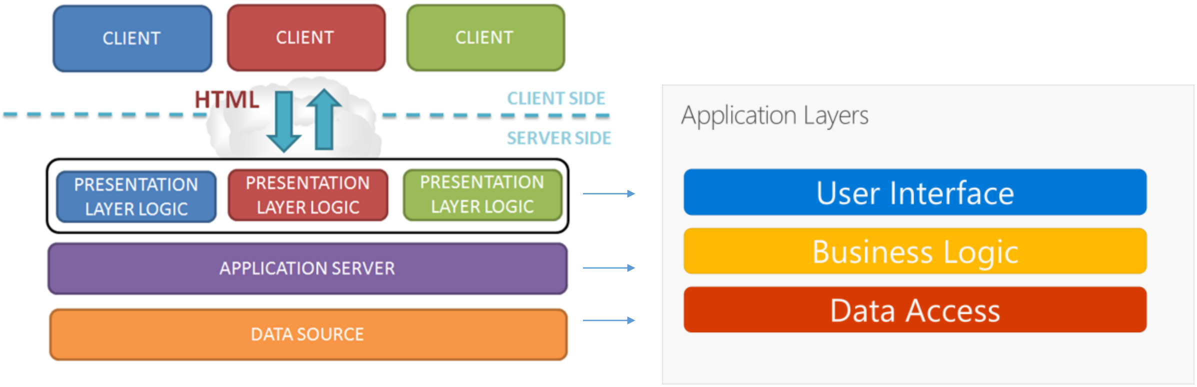
- A Web server alone provides the online store's functionality.
- The Web server takes your request, then passes it to a server-side program able to handle the request.
- The server-side program looks up the pricing information from a database or a flat file.
- Once retrieved, the server-side program uses the information to formulate the HTML response, then the Web server sends it back to your Web browser.

Difference between Web Server and Application Server: An Example

Scenario 2: Web server with an application server

- The Web server still delegates the response generation to a script.
- However, you can now put the business logic for the pricing lookup onto an application server.
- With that change, instead of the script knowing how to look up the data and formulate a response, the script can simply call the application server's lookup service.
- The script can then use the service's result when the script generates its HTML response.

Traditional "N-Layer" architecture applications



CLIENT

CLIENT

CLIENT

HTML

CLIENT SIDE

SERVER SIDE

PRESENTATION LAYER LOGIC

PRESENTATION LAYER LOGIC

PRESENTATION LAYER LOGIC

APPLICATION SERVER

DATA SOURCE

Application Layers

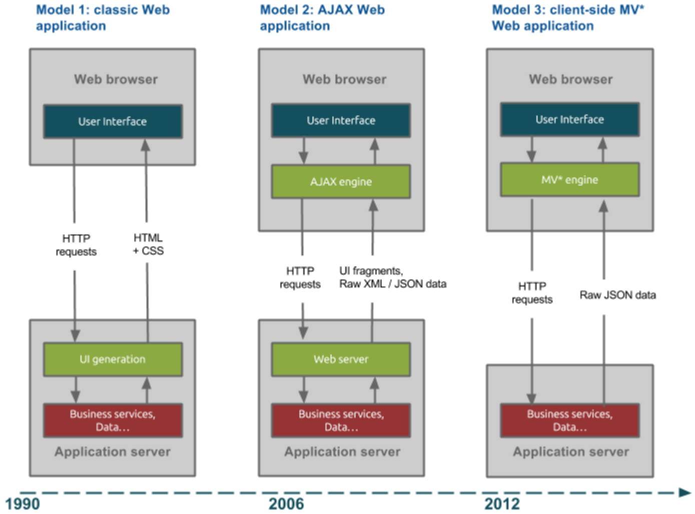
User Interface

Business Logic

Data Access

https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/common-web-application-architectures

Evolution of Web application architectures



https://blog.octo.com/en/new-web-application-architectures-and-impacts-for-enterprises-1/

Evolution of Web application architectures (cont'd)

Model 1: classic Web application

- On the first diagram, the Web application is mainly executed on the server side.
- It sends directly to the browser HTML pages, CSS and possibly JavaScript enhancing the behavior.
- Then, for each user action requiring new data, the server is queried and returns a whole new HTML page.

Evolution of Web application architectures (cont'd)

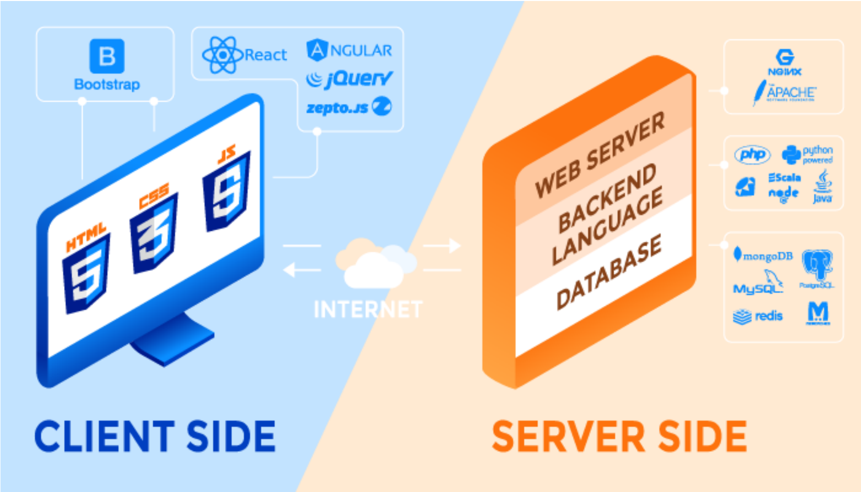
Model 2: AJAX Web application

- This architecture principle can make the application more responsive by reducing exchanges between browser and server.
- When a user action generates a client call to retrieve new data, the server only returns view fragments.
- Thus, only a small part of the screen is refreshed, rather than the entire page.
- This requires the development of client-side JavaScript in order to manage partial refreshments, for example by using the jQuery library and its $.Ajax function or others tools more integrated with server platforms.

Evolution of Web application architectures (cont'd)

Model 3: client-side MV* Web application

- The term **MV*** refers to the MVC pattern, for *Model View Controller*, widely used server-side to separate data and views management.
- More and more we use this term **MV***, in order to highlight the little differences from pure MVC implementations.
- The important point in this new architecture is the **shift of all the UI logic from the server to the client**.
- Now **the server sends only raw unformatted data and the client is responsible for generating the screen** out of it.



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