Lesson 1: Web Application Architecture

Website vs Web Application

Most people will use the term website for just about everything, **and they're not wrong**. However, in the opinion of an IT professional, the difference boils down to this:

- A website is informational
- A web application is interactive

To illustrate the difference, let's take the example of a restaurant's web presence. If you visit the site for your local restaurant and find nothing more than the hours of operation, a menu, directions from the nearest highway or a static map, and contact information listed then you've got yourself a **website**.

However, if the site for your local restaurant gives you all the "static" information and then additional functionality like make a reservation, view a customized menu that includes prices, order your food online, or purchase a gift card. This sort of interactivity is specific to a web application and is what differentiates it from a website. Put simply, **a web application is a web site that the user can control**.

What is Web Application Architecture?

Web Application Architecture is a framework that is comprised of the relationships and interactions between application components, such as middle ware systems, user interfaces, and databases. The general concept of Web Application Architecture is in line with the concept of a browser user who triggers an application that is capable of running in multiple websites.

In essence, Web Application Architectures can be defined with the depiction of this process:

- **A user browses for a specific URL, which the browser locates and requests.**
- **Over the network, data is sent from the server to the browser, then executed by the browser so that it is able to display the requested page.**
- **The user views and interacts with the page.**

Components of Web Applications Architectures

Web application architectures are comprised of several components that help build its digital makeup. These components can be categorized into two areas: user interface app components and structural components.

User interface app components refer to web pages displaying dashboards, logs, notifications, configuration settings, and more. They are not relevant to the structural development of the application and are more user interface/experience oriented.
The structural components, which are the real meat of the app development process, are:

- The web browser or client.
- The web application server.
- The database server.

The web browser or client is the interface rendition of a web app functionality, with which the user interacts with. This content delivered to the client can be developed using HTML, JavaScript, and CSS and doesn't require operating system related adaptations. The web browser or client manages how end users interact with the application.

The web application server manages business logic and data persistence and can be built using PHP, Python, Java, Ruby, .NET, Node.js, among other languages. It's comprised of at least a centralized hub or control center to support multi-layer applications.

The database server provides and stores relevant data for the application. Additionally, it may also supply business logic and other information that is managed by the web application server.

Types of web application architecture

Regardless of the model, all web application components always work simultaneously and create an integral web app. Depending on how the app logic is distributed among the client and server sides, there can be various types of web application architecture.

According to the very first and basic web app architecture, a server, consisting of *web page construction logic* and *business logic* interacts with a client by sending out a complete HTML page. To see an update, the user needs to reload the page fully or, in other words, to have the client send a request for an HTML page to the server and load its entire code once again. Look at this type's web application architecture diagram below.

Since all the logics and data are stored on the server and the user doesn't have any access to it, this architecture type is highly secure. Still, due to constant content reload and huge data exchange, it is more common for static websites than actual web apps.

Widget web app

In this type, the *web page construction* logic is replaced by *web services*, and each page on the client has separate entities called *widgets*. By sending AJAX queries to web services, widgets can receive chunks of data in HTML or JSON and display them without reloading the entire page.

With real-time widget updates, this type is more dynamic, mobile-friendly and almost as popular as the next type. Yet, this web application architecture requires longer development time and is less secure due to the app logic partially shifted to the exposed client side.

Single-page web app architecture

This is the most modern web application architecture, where you download a single page only once. On the client side, this page has a JavaScript layer that can freely communicate with web services on the server and, using the data from web services, make real-time updates to itself. The way it works is shown on the web app architecture diagram below:

Chunks of data transferred from the server to the client here are minimal, especially compared to the first type. It's very agile, responsive and lightweight web app that can be easily transformed into a mobile app with the help of hybrid wrappers such as Cordova/PhoneGap.

A full stack web developer is a person who can develop both client and server software.
Besides mastering HTML and CSS, he/she also knows how to:

- Program a browser (like using JavaScript, jQuery, Angular, or Vue)
- Program a server (like using PHP, ASP, Python, or Node)
- Program a database (like using SQL, SQLite, or MongoDB)

Popular Stacks

- JavaScript. Linux. Apache. MySQL. PHP. (LAMP Stack)
- JavaScript. Linux. Nginx. MySQL. PHP. (LEMP Stack)
- JavaScript. MongoDB. Express. AngularJS. Node.js. (MEAN Stack)
- JavaScript. Python. Django. MySQL. (Dango Stack)
- JavaScript. Ruby. SQLite, PHP. (Ruby on Rails)

Advantages

The advantage of being a full stack web developer is:

- You can master all the techniques involved in a development project
- You can often make a prototype very rapidly
- You can often provide help to all the team members
- You can often reduce the cost of the project
- You can often reduce the time used for team communication
- You can switch between front and back end development based on requirements
- You can better understand all aspects of new (upcoming) technology
- You can get a high-paying tech job and outgrow your peers in the industry

Lesson 2 : Git and GitHub

Description of version control and Git

Version control is a concept in software engineering which applies to the management of source code. There are many systems for applying version control practices to source code. We will be focusing on one of the most popular, "Git".

Git

Git was created in 2005 by Linus Torvalds. Git allows a team to work concurrently on a single project, or "repository", all while staying current and up to date. This is done through branching, each member, or feature, can have a branch from the master branch. That branch can be edited as needed without interfering with the "master". When the time comes to merge the branches back together, git will evaluate where the changes were made and will correct the master to reflect those changes without interfering with other changes. Git also acts as a sort of time machine, allowing a team to revert ANY changes made to the source code throughout the history of the code.

Github.com

Github.com is a network to store your repositories, essentially it is a repository of repositories. It is one of many available on the internet, and the most popular. Git != Github, although they work very well together. Github is a place for you to store your code or find other projects. It also acts as a portfolio for any code you've worked on. If you plan on being a developer you should have a Github account. We will be using Github extensively throughout the boot camp.

Basic terminal commands

> We will be using the "terminal" or "command line" throughout this boot camp. Install "git-bash" based on your operating system. Within our terminal we can: traverse our file structure, add files, remove files, update files, and tons more! The terminal is a very powerful tool for developers and you will be using it a lot in your professional development career. It might look scary at first, but in time you enjoy using it. It is better to get accustomed to it now. To begin we will learn the 'basic' commands:

"List": Allows us to view the contents of the current folder we are in.
$ ls

"Change Directory": Allows us to move to a new folder or 'directory'.
$ cd [folder]

"Make Directory": Makes a new folder in the directory you are currently in.
$ mkdir [folder name]

"Touch" will create a new file.
$ touch [file]

"Remove": permanently deletes a file. (WARNING! This PERMANENTLY deletes the file)
(Note: This will not remove folders, we need a special command for that)
$ rm [file]

Exercise

In this exercise we will create a new folder titled: "10000Coders"

To get to your C directory type:
cd /c

and press enter. Once in your top level type:
mkdir 10000Coders

and press enter. Congratulations, you have created a directory!

Github

As mentioned before, Github is a central place to store, view, and download repositories, it is not synonymous with "git". You need to have a Github account in order to complete this exercise. A github account is mandatory if you want to seriously build your career as a developer.

Create Repository

After you create your account on Github and verify your email, create a repository on Github and check Initialize this repository with a README during repository creation. Make sure you give a clear name and description to the repository.

Cloning

In order to work on a project you must clone (download) it to your local machine. To do this, visit the top-level of the forked repo on your own account, and click on the green button in the upper right hand side of the page that says: "Clone or Download". A drop down should appear and you can click on the clipboard icon to copy the address. (Note: You can download the entire repo, but this is not advised as cloning will do quite a few steps for you behind the scenes that will make your life much easier.)

Once you have the address copied, return to your terminal window and enter the following:

```
$ cd 10000Coders

$ git clone [copied address]
```

This will download the repo and you now have a local copy of the repo saved to your machine!

Git commands

Throughout these lessons, we will interact with git through our terminal. In the future you may wish to use a Git GUI interface, but during these lessons we will need to use the terminal for all git functions.

In this exercise, we will add a file to our project and then commit this change to memory in git.

In your terminal, cd into your Github repository folder name
Then enter the following in your terminal:
$ touch index.html

This will add a new file to your project titled "index.html". Add the following content inside index.html

```
<html>
    <head>
        <title>My First Web Page</title>
    </head>
    <body>
        <h1>This is my first web page. Oh, I'm excited!!!!</h1>
    </body>
</html>
```

Save the file. At this time we can use the git command "status". Status will read back the status of all changes made to your repo. Use status often, if you're not sure if something worked, using status will tell you.
$ git status
You should see something along these lines:

```
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    index.html

nothing added to commit but untracked files present (use "git add" to track)
```

This tells us that we have a file that has been changed, but is not saved into the git history yet.

To do this we will use the 'add' command:

```
git add index.html
```

This has added our changes to the history, now to save that history, we will use "commit". Commit will take all of our added changes and save it to git history. For future reference you will be able to leave a message about the changes, this will make it easier to go back and find the changes you (or anyone) are looking for in the future, if you should need to. It is always a good idea to leave a concise description of the changes in your commit. A shorthand way of leaving an inline message is using the '-m' flag and writing your message directly after, in quotes.

```
git commit -m "added index.html"
```

Now that we have our changes saved locally, we want to share those changes in our Github. To do this we will "push"

```
git push
```

You will be prompted for your username and password. Enter these and then you will receive a message if your changes were pushed successfully:

```
Counting objects: 2, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 2.97 KiB | 0 bytes/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To git@github.com:[your username]/[Your repository name]
 c1d2dc4..646cd88 master -> master
```

Congratulations! You just pushed your first git commit!

Lesson 3 : Hosting your website on GitHub Pages

Web hosting

In order to get a website active and live on the internet, you need to host a website. Web hosting is basically the space that you buy on a web server to store your website files. When you buy website hosting you basically rent server space on a server where your web files will be placed. So whenever somebody will look up for your website by entering your domain, he will get directed to your website. You can design a website on your own computer but unless you upload it on a hosting server it can never be accessed by anyone.

There are multiple web hosting companies available to host your website ranging from major cloud platforms like Azure, AWS to normal hosting platforms like Hostgator. or your learning, we will use GitHub Pages which is a free resource to host your website.

Create a repository

Head over to GitHub and create a new repository named username.github.io, where *username is your username* on GitHub.

If the first part of the repository doesn't exactly match your username, it won't work, so make sure to get it right.

Clone the repository

Go to the folder where you want to store your project, and clone the new repository:

cd /c

cd 10000Coders

git clone <<github_repository_url_created above>>
cd <<your github username>>.github.io

cp ../<<your first repository foldername>>/index.html .

Push the changes

Add, commit and push the changes to your Github repository

git add index.html
git commit -m "Adding index.html to my GitHub Pages
git push -u origin/master

... and you're done!

Fire up a browser and go to https://[your github username].github.io.

Ex. Check mine at https://meetmranil.github.io