# Lecture 2

## What we discussed in the last class

- **What is a server?**
  - In a very nutshell, a server is something that responds to clients requests. A client can make various types of request like **GET** request, **POST** request, **PUT** request, etc. And the server responds to these requests doing necessary magic the data they get.

- **What is HTML?**

  **HTML**
  - HTML is hypertext markup language. Its initial purpose was to create Web Documents that linked with each other.
  - HTML documents have something known as tags which look like this :
    - <form action = '/' method = 'post'> (HTML is case insensitive)
    - Form is the **tag**, action is an **attribute** and '/' is the action **attribute value**.
    - Each tag you open has to be closed using a forward slash </form>
    - When tags have no children as in nothing between the opening and closing tags then you can close the tag there itself for example <img src = 'pic.png'/>

  - Today you saw your first HTML tags <html> tells that browser that it is a HTML document, <head> specifies properties like <titlle> which sets the body title <body> is where your content goes.

**Today's HTML code snippet :**

```
<html>
        <head>
        <title>My First Application</title>
        </head>
        <body>
                <form action = '/' method = 'POST'>
                        <input type = 'text' name = 'firstName'/>
                        <input type = text name = 'lastName'/>
                        <input type = 'submit'/>
                </form>
        </body>
</html>
```

- Here form action attribute specifies the URL the request is going to in this case it will go to the root of the server. For example if server is hosted at localhost:3000 then it will go at localhost:3000
- Form method attribute specifies the type of request to be sent, here we are sending a post request.

- **What is CSS?**
  - Initially html was pre-styled by your browser and you could not change how the components looks. But with CSS(Cascading Style Sheets) you can change the way the components look.
  - We will cover CSS in the next class.

- **What is JS or Javascript?**
  - Javascript as explained in the class is the programming language or more precisely the scripting language of the web. We will teach you how javascript is different from other programming languages in the next class.

- **What is Node.js?**
  - Someone had the brilliant idea of making Javascript programming being able everywhere instead of just being limited to the web browser. So someone took the Google V8 Javascript engine and created Node.js which can interpret Javascript. So basically now Javascript can be used everywhere from Servers to Windows applications.

- **What is NPM?**
  - NPM is node package manager, just like we used apt-get as a package manager on ubuntu and windows control panel on windows. This manages packages for node.js.
  - **npm install express -g** installs globally for all projects.
  - **npm install express** installs only in the current directory, makes a node_modules directory if it doesn't already exists and saves the packages in the node_modules folder.

- **What is express?**
  - Node is low level so express is used to make some things easy for us like allowing easy routing. We will go in greater detail later on in the course.

- **Today's Class**

    - Today's class was sort of demo class, I wanted to tell you that starting your own server is not magic but rather can accomplished very easily in a very few commands.

## Code snippets of what we did in the last class

- Installing Node.js from a PPA.
    - ***curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -***
    - **sudo apt-get install nodejs**
        - These commands change the default source of node to node 6 and install node and nodejs.
- Checking node and npm version
    - Check node version by : **node -v**
    - Check npm version by : **npm -v**
- Installing express and express-generator globally
    - **npm install express express-generator -g**
- Creating a new boilerplate using **express-generator**
    - **express myapp**
- Create index.html using the above given html snippet and place it in public directory of myapp folder.
- Do **npm start** on your terminal once you get into the myapp directory using cd to start the server.
- Go to **localhost:3000** in your browser and see that your html file should be served there.
- Always restart your server when you make changes on your server with **control + c** to close it and **npm start** to start it again.
- Finally to accept the form data and print the response on the screen edit the **routes/index.js** file in myapp directory.
    - Change **router.get** to **router.post** to accept the post request of the form.
    - Finally replace the **res.render line** with **res.send(req.body);**
- You should have a working form showing information about what you just input in the last screen.

## What happened?

- The moment you went on localhost:3000 and hit enter, the browser sent a **GET** request to the server running at 3000 port.
- The server first searched the public folder for index.html and returned it to the browser.
- The browser read you html code and rendered the page.
- You entered the details and hit the submit button, the form had **action** set to **'/'** and **method** set to **'POST'.**
    - So the form sent a post request to the server at localhost:3000 with the form data. It used the input **name attributes** to name each field.
- The request reached the server and was handled by the **router.post** at **'/'** which received the **req**, **res** and **next** parameters thanks to express.
- You did res.send(req.body) which sent a response back to the browser
    - **req.body** is where the data for the form was sent.
- You see {'firstName' : 'Ishan', 'lastName' : 'Sharma'} in the browser.

## FAQ's:

**Q :** I did ctrl-z instead of ctrl-c, and now the server is not starting?
**Ans :** ctrl-z does not close the server properly and the server is still running in the background using the port 3000.

- **ctrl+z** means "**pause**". After you've paused a process, you can type **fg** (to resume it normally), or **bg** (to resume it in the background).

Use **Ctrl+C** to exit the node process gracefully
To clean up the mess depends on your platform, but basically you need to find the remains of the process in which node was running and kill it.

For example, on Unix: ps -ax | grep node will give you an entry like:

```
1039 ttys000    0:00.11 node index.js
```
where index.js is the name of your node file.

In this example, 1039 is the process id (yours will be different), so kill -9 1039 will end it, and you'll be able to bind to the port again.

Taken from this answer :
http://stackoverflow.com/questions/6958780/quitting-node-js-gracefully

**No Assignment this time, deadline extended for Assignment 1 till Sunday 11PM.**

**Please get a Digital Ocean IP too, it is free $50 credits :)**

**No Quiz in the next class**

**Next class snapshot**

- Learning to style using HTML5 and CSS3.
- Basic Javascript introduction
- Be on time.

**Important resources**

- **HTML Starter :** https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started
- **CSS Starter :** https://css-tricks.com/how-css-selectors-work/
- **JS Expert :** http://eloquentjavascript.net/

Before you come to the next class make sure you understand HTML and CSS starter links that I have written above.

Keep on reading eloquent javascript (JS Expert Link) whenever you have time.