STACK-X

MICROTRAIN'S
DEV BOOTCAMP

microtrain®

Table of Contents

MicroTrain's Dev Bootcamp

Resources for MicroTrain's Developer Bootcamp.

Chapter 1: Dev Environment

A development (dev) envrionment is just as it sounds. It is any system or configuration of systems in which a product is developed. This may be anything from a single laptop that runs code in a browser to a cluster of machines that perfectly emmulates a production environment. Our environment will consist of a single laptop running full LAMP and MEAN stacks. Since the bulk of the course involves web development we can run most code in browser. Additional code will run in a terminal window, mobile device emmulator or on your own mobile device.

In the section you will learn:

- How to a dev enviroment from scratch.
- Linux basics.
- How to install Linux software.
- How to install the LAMP stack.
- The basics of NodeJS and NPM.
- The basics of Git and GitHub.
- How to use and SSH key for authentication.

The goal of this section is to gain an understanding of the dev environemnt and gain some comfort on a Linux commandline.

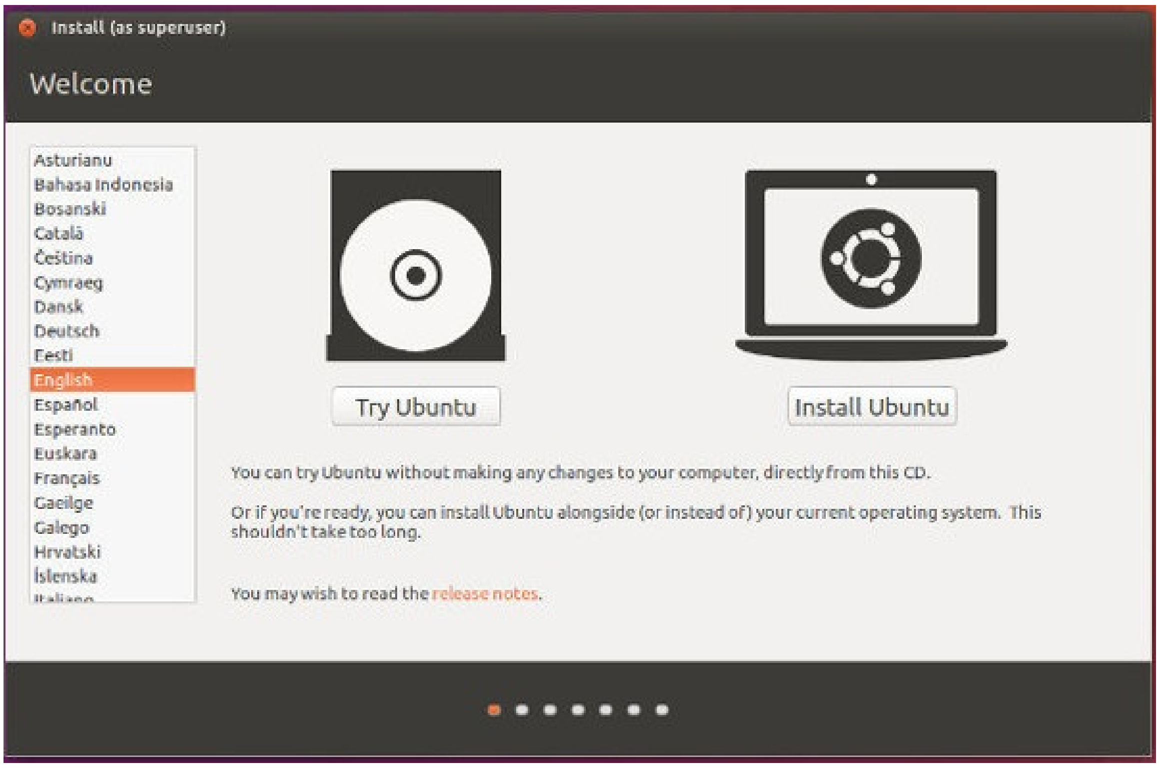
Next: Install Ubuntu

Install Ubuntu 18.04

Your development (dev) environment is any system upon which your develop your applications. When possible, building your dev environment as close to production (prod) as possible is helpful. When this is not possible running tests in an emulator is a good option. This course will make use of both methods. Lets start by installing the operating system; Ubuntu Linux.

Insert the preloaded Ubuntu 18.04 dongle into an open USB port and power up the system. Press f12 when the Dell splash screen appears and choose *USB Storage Device* under legacy boot legacy.

You will see a window titled *Install (as superuser)* from this window choose English and click the *Install Ubuntu* button.

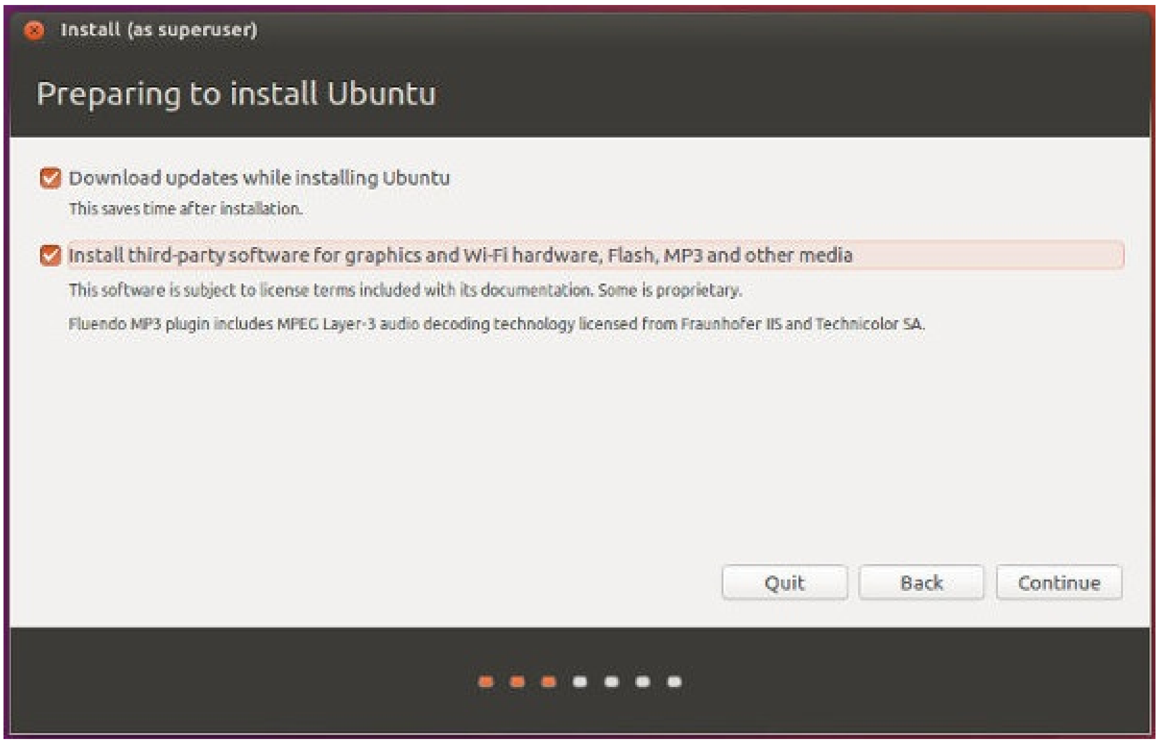


You can try Ubuntu without making any changes to your computer, directly from this CD.

Or if you're ready, you can install Ubuntu alongside (or instead of) your current operating system. This shouldn't take too long.

You may wish to read the release notes.

You will now be given two options *Download updates while installing Ubuntu* and *Install third-party software...* select both of these options.

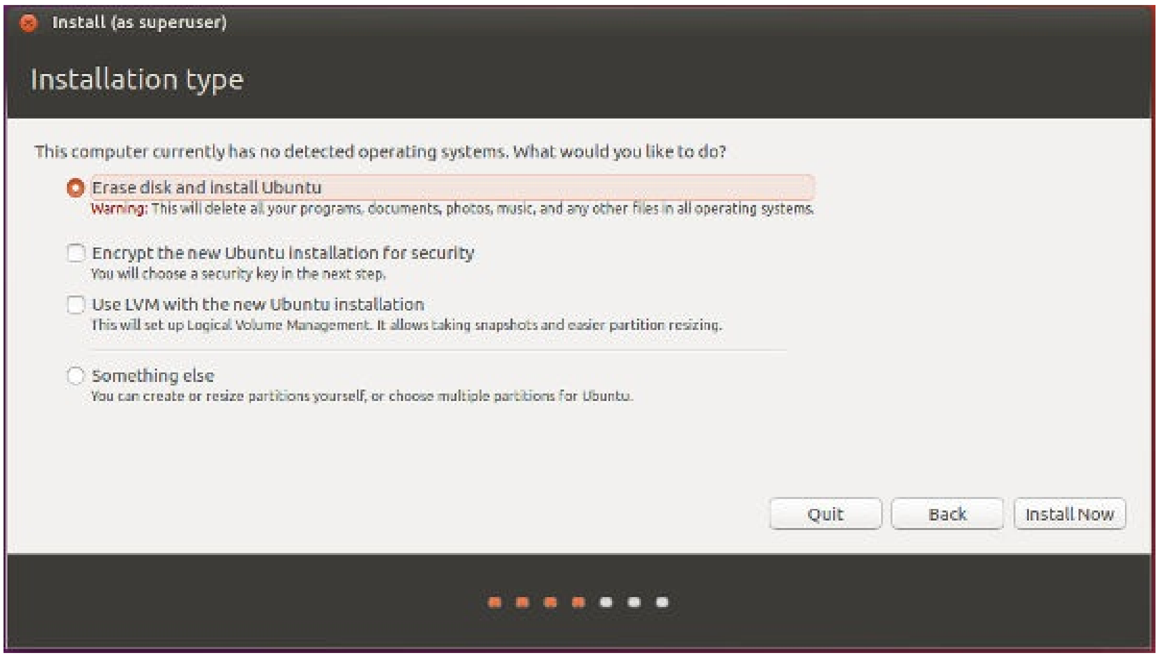

Install (as superuser)

Preparing to install Ubuntu

[✓] Download updates while installing Ubuntu
This saves time after installation.

[✓] Install third-party software for graphics and Wi-Fi hardware, Flash, MP3 and other media
This software is subject to license terms included with its documentation. Some is proprietary.
Fluendo MP3 plugin includes MPEG Layer-3 audio decoding technology licensed from Fraunhofer IIS and Technicolor SA.

Now it's time to install the system. To keep things simple we will do a simple install. Be sure the *Erase disk and install Ubuntu* option is selected and click the *Install Now* button. You may see a dialog that asks *Write changes to disk?* with some additional details, simply click the *continue* button.


Install (as superuser)

Installation type

This computer currently has no detected operating systems. What would you like to do?

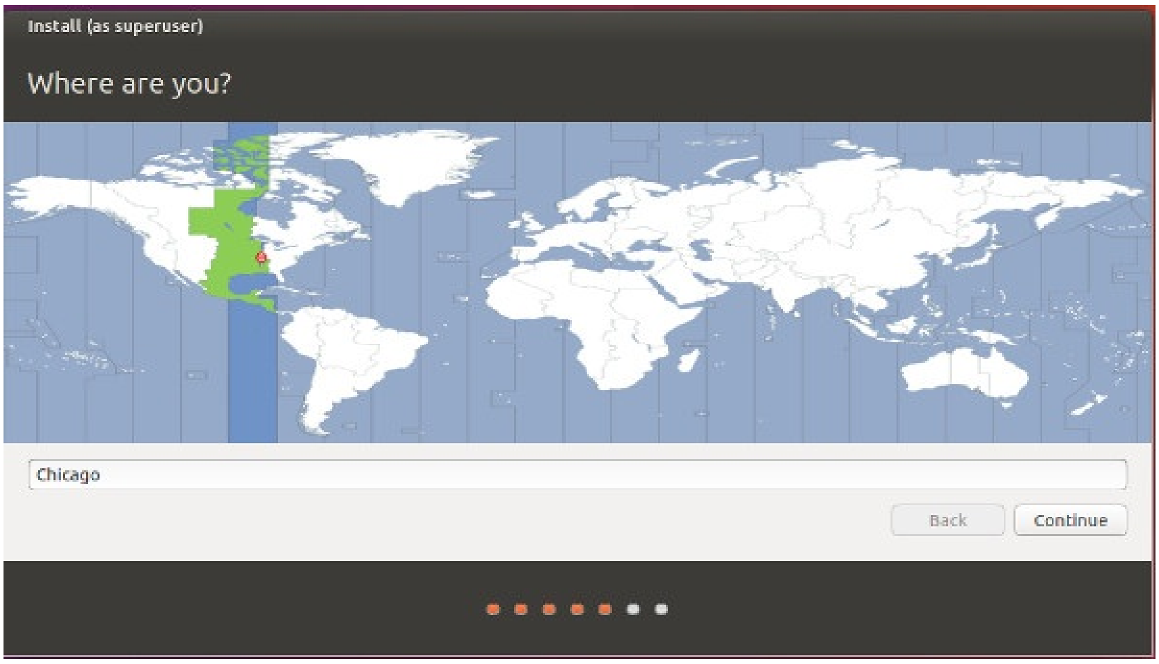
[✓ Erase disk and install Ubuntu]
Warning: This will delete all your programs, documents, photos, music, and any other files in all operating systems.

Encrypt the new Ubuntu installation for security
You will choose a security key in the next step.

Use LVM with the new Ubuntu installation
This will set up Logical Volume Management. It allows taking snapshots and easier partition resizing.

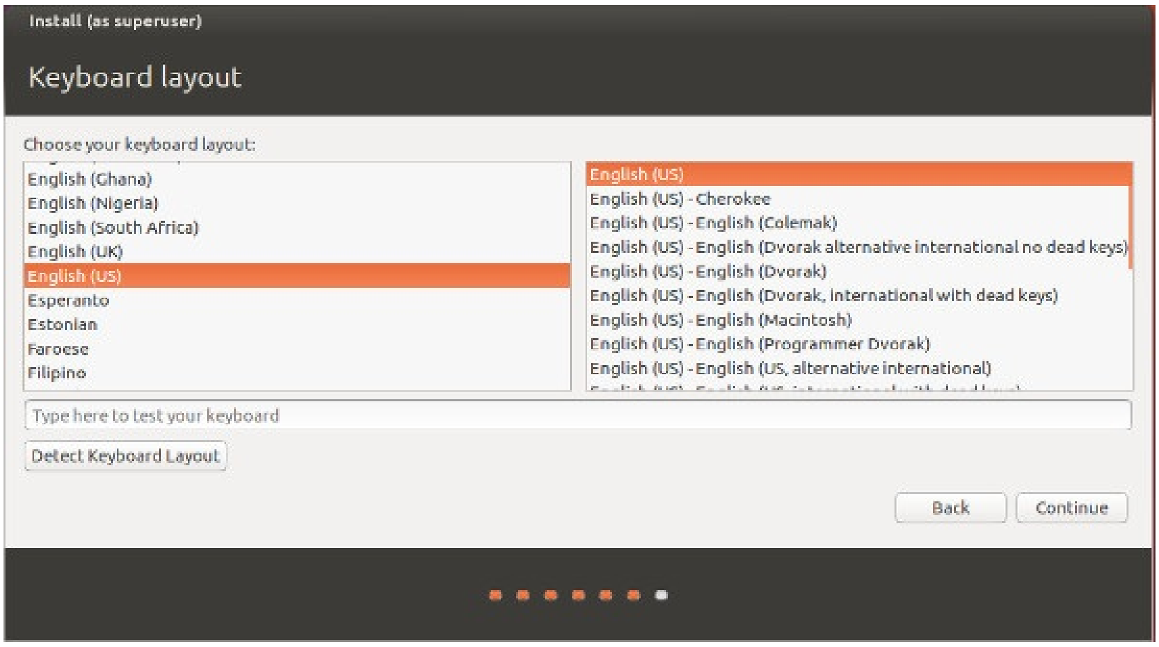
Something else
You can create or resize partitions yourself, or choose multiple partitions for Ubuntu.

You will now see the *Where are you?* screen. This should default to Chicago (or your default location), if not choose Chicago (or your default location) as your location and click the *Continue*.



Chicago

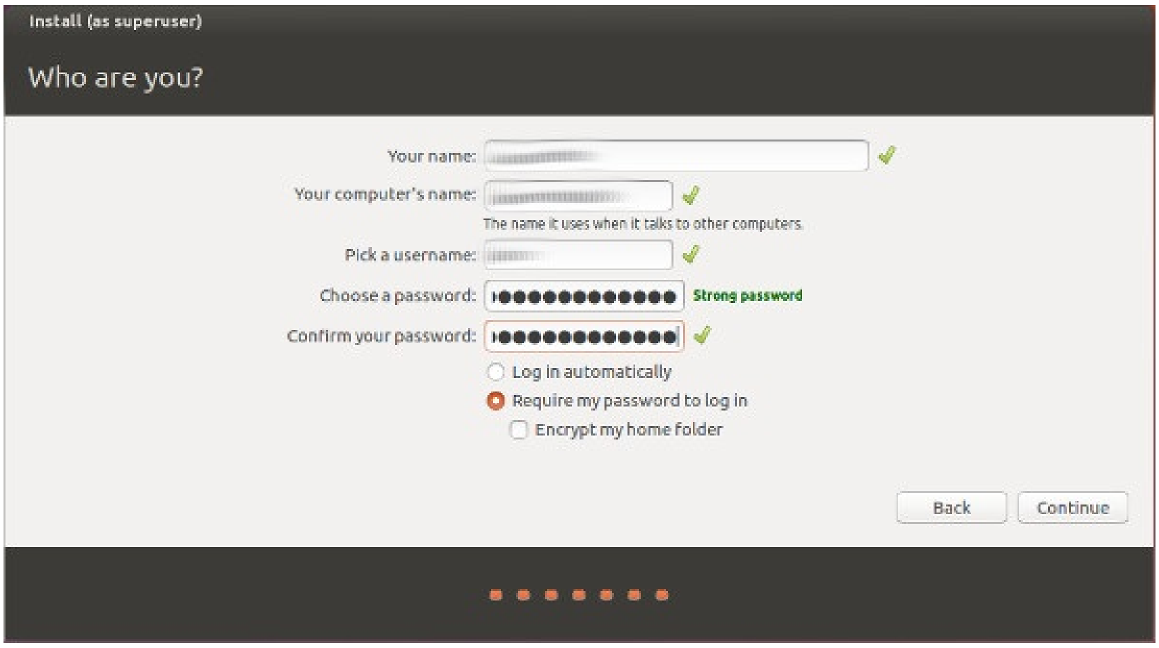
This sets your systems local timezone. For *Keyboard layout* choose English (US) and English (US) and click the continue button.



English (Ghana)
English (Nigeria)
English (South Africa)
English (UK)
English (US)
Esperanto
Estonian
Faroese
Filipino

English (US)
English (US) - Cherokee
English (US) - English (Colemak)
English (US) - English (Dvorak alternative international no dead keys)
English (US) - English (Dvorak)
English (US) - English (Dvorak, international with dead keys)
English (US) - English (Macintosh)
English (US) - English (Programmer Dvorak)
English (US) - English (US, alternative international)

On the *Who are you?* screen enter *Your name:* as your first an last name, accept the default computer and usernames and choose a password. Do not forget your password, we are not able to recover this for you. Be sure the *Require my password to log in* option is select and that the *Encrypt my home folder* option **IS NOT** selected. Click the continue button and wait for the system to install.


Install (as superuser)

Who are you?

Your name: ✓

Your computer's name: ✓

The name it uses when it talks to other computers.

Pick a username: ✓

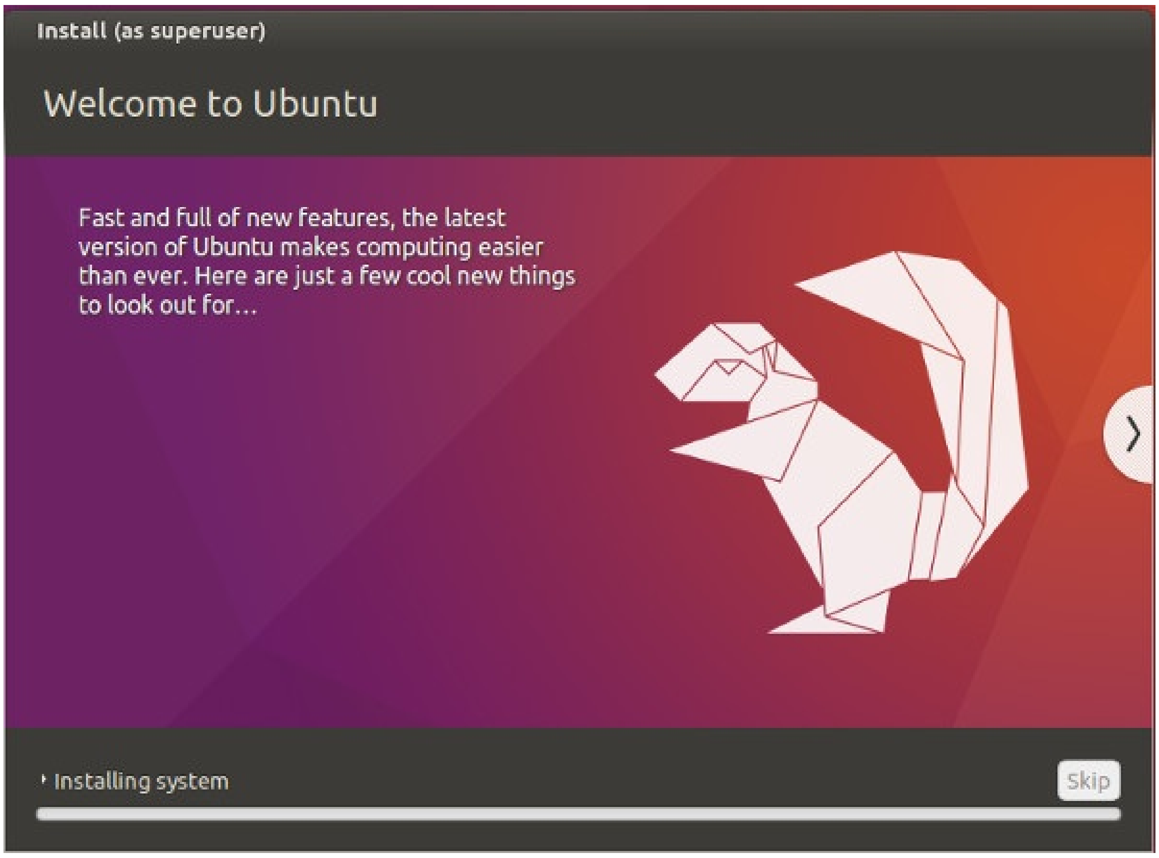
Choose a password: ●●●●●●●●●●●●● Strong password

Confirm your password: ●●●●●●●●●●●●● ✓

Log in automatically [✓ Require my password to log in]

Encrypt my home folder

While the system installs, you will see series of flash screens telling you the features of Ubuntu.

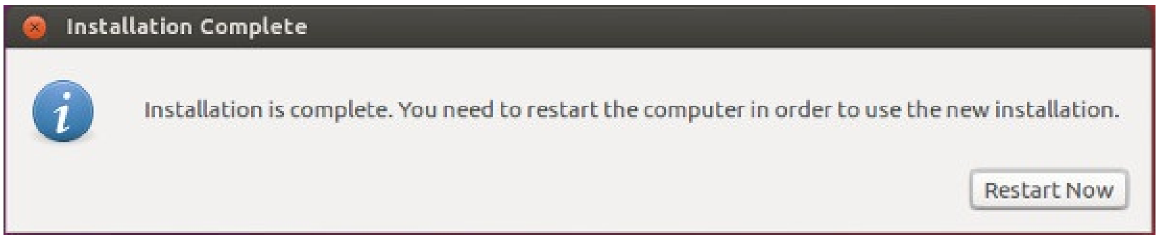

Install (as superuser)

Welcome to Ubuntu

Fast and full of new features, the latest version of Ubuntu makes computing easier than ever. Here are just a few cool new things to look out for...

Installing system

One installation has completed you will be prompted to restart your system. Click the *Restart Now* button.


Installation Complete

Installation is complete. You need to restart the computer in order to use the new installation.

You should see a prompt that says *Please remove the installation medium, then press Enter:*. Pull the dongle from the USB drive and press the enter key.

Summary

In this lesson you learned

- how to install Ubuntu Linux (Ubuntu, Linux)

Next: Linux Basics

Linux Basics

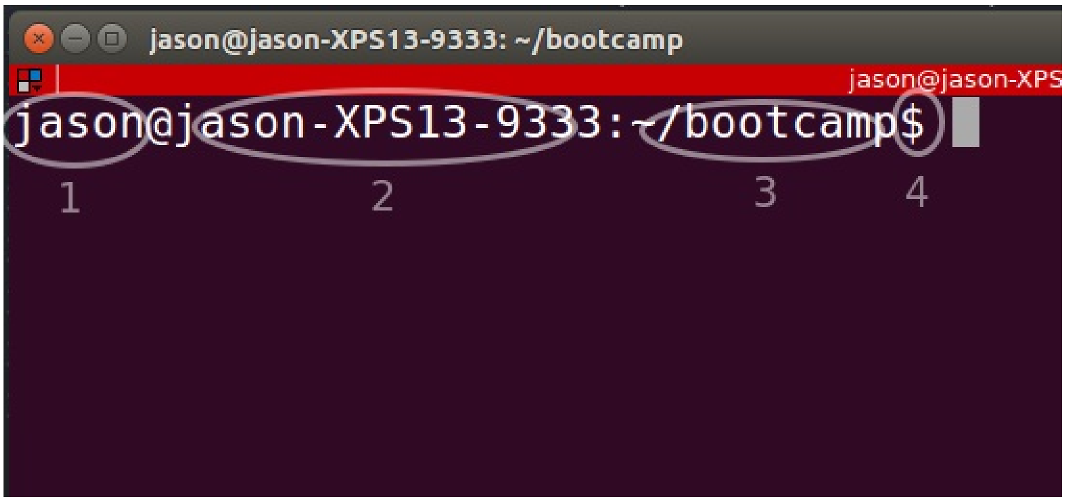
This section is intended to provide a reference for the basic commands needed to navigate a Linux system from the command line. We will take a deeper dive and use these commands in later lessons. While we call these commands, they are really programs. In other words, when you type a command in Linux, you are really invoking a program. Most programs can be invoked with out any additional arguments (aka parameters), but in most cases, you will want to pass additional arguments into the program.

For example `vim filename.txt` would use an editor called vim to open a file that is named filename.txt. Some programs expect arguments to be passed in a specific order while others have predefined arguments. All Linux programs predefine `--help` so you might type `chown --help` to learn how to use the chown command (or program).

The Linux File System

Linux does not use drive letters as you may be used to if you come from a Windows environment, rather everything is mounted to a root name space. */*.

- / - root
- */etc* - sytem configuration
- */var* - installed software
- */var/log* - log files
- */proc* - real time system information
- */tmp* - temp files, cleared on reboot



```
jason@jason-XPS13-9333:~/bootcamp$
```

1. The user name of the logged in user
2. The name of the current machine
3. The current working directory (CWD)
4. User level ($ for standard user, # for root user)
   - The space immediately to the right of # or $ is the command line entry point.

Item 3, the current working directory is what we want to focus on at the moment. This tells us how to navigate. The tilde `~` is a short hand for the home directory of the current user which would be */home/[username]* in my case this would be */home/jason* which says start at root */* and find the *home* directory from there find a directory called *jason*. The CWD in the above image `~/bootcamp`. The `cd` (change directory) is how you navigate the file system using a terminal window (aka - console, cli, command line). If i say `cd 01-DevEnvironment` it will look for the *01-DevEnvironment* directory inside on

/home/jason/bootcamp as that is my current working directory and the lack of a preceding `/` tells the system to look on a relative file path. If however I add a / so that the command reads `cd /01-DevEnvironment` it tells the system to look at the absolute path so the system will look for *01-DevEnvironment* directory to exist directly under root.

Basic Commands

- `[command] --help` - Returns a help file for any command (or program).
- `sudo` - Super-user do (elevates privs to admin).
- `chown` - CHange OWNership, changes the ownership of a file.
- `chown user1:group1 some-file` - Changes the ownership of some-file to user1 and group1.
- `chmod` - CHange MODe, changes file permissions.
- `chmod +x filename` - Makes a file executable.

- `apt-get` - Retrieves and maintains packages from authenticated sources.

- `apt-get install [package]` - Installs a target package from a repository.
- `apt-get update` - Update your package list..\
- `apt-get upgrade` - Upgrade all packages from the updated list
- `apt-get remove` - Remove all packages.
- `apt-get purge` - Remove all packages and their config files.

- `dpkg` - A package manager for Debian-based systems, this is primarily used to deal with files ending with a `.deb` extension.

- `sudo dpkg --install some-pkg.deb` - Installs some-pkg.deb.

- `pwd` - Print working directory (where am I?).

- `ls` - List (a list of files and directories).
- `ls -l` - List long format (file permissions, owner, group, size, last mod, directory name).
- `ls -a` List all (show hidden files).
- `ls -la` List all in long format(ls -l + ls -a).
- `ls -R` List recursive (shows all child files).
- `cd` - Change directory.
- `cd ~` - Change directory to home (a shortcut to your home directory).
- `cat` - Concatenate (dumps a file to the console, a handy read only hack).
- `cat [filename]|less` - Pipe the cat command into a paginated CLI (less --help).
- `cat [filename]|tail` - Last line of the file, great for looking at log files.
- `cat [filename]|tail -f` - Last line of the file, great for looking at log files.
- `find -name [x]` Find all file for whom the name matches x.
- `find -name [x]` Print|less find all files for which the name matches x and print them to a paginated CLI.

On this system Apache writes log files to /var/log/apache2. For this example I only want to retrieve a list of the error logs.

- `cd /var/log/apache2` - Change to the Apache error log directory.
- `find *error.log.*` - Find all error logs in the current working directory (CWD).
- `cd / && locate access.log` - Locate all access logs (recursively) under the root directory.
- `grep` - Globally Search a Regular Expression and Print (Uses Regular Expressions (regex) to search inside of files).
- `* - wildcard`
- `/^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\z/` - Check an ip address regex in code.

Beyond Linux: Regex in code.

```
//Returns 1 if $string matches a valid IP, returns 0 if it does not.
$valid = preg_match('/^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\z/', $string);
```

- `grep ssl error*|less` - Find all ssl errors.
- `grep '\s404\s' access*|less` - Find all 404 errors in the access logs, this is 404 surounded by whitespace.
- `grep '^127' access*|less` - Find every line that begins with 127 (access logs begin with an IP).
- `sudo grep -ir error /var/log | less` - Find all errors (-i case insensitive) in all logs, we sudo because some log files are only accessible to root.
- `pgrep` - Returns a list of process id(s) for given processes. The process can be requested using regex.
- `pgrep chrome` - Returns a list of all chrome process ids.
- `pgrep chrome | xargs kill -9` - Kills all running chrome processes.
- `cat /etc/passwd|less` - A nice hack to get a list of all users on a system.

tip: Use tab expansions to auto-complete a command or an asterisk as a wild card. The up and down arrows can allow you to browse your command history and replay previous commands. Use `Ctrl + r` to search your command history by entering partial commands.

Summary

In this lesson you learned

- about the Linux file system (Linux)
- basic Linux commands (Linux)

Additional Resources

- Ten Steps to Linux Survival
- Linux Fundamentals

Next: System Basics

System Basics

In this section we will learn use some of the basic Linux commands from the previous section to install a few basic software packages.

Login in to your system and open a terminal window using `Ctrl + Alt + T`. Then type the following commands. Pressing enter after each command.

```
sudo apt update
sudo apt upgrade
```

Lets look at these commands a little closer.

- sudo - in short `sudo` will run what ever commands that follow it with root level privleges.
- `apt` - Apt is a package manager for Linux. This works by maintaining a list of remote repositories from which packages can be installed. Most of the management is done automatically.
    - `apt update` - tells the system to update everything it knows about the repositories.
    - `apt upgrade` - tells the system to upgrade all packages to their latest versions.

Terminator

Terminator is a terminal emulator that allows for multiple tabs and split screens. This makes life a lot easier when you are dealing with several command line applications and background processes all at once.

```
sudo apt install terminator
```

Now close your terminal window and use Dash to find and open Terminator. Open Dash `Super + F` and type *terminator* into the search field. Click the Terminator icon to launch Terminator. You'll notice the Terminator icon is now in the Launcher right click the Terminator icon and select *Lock to Launcher* from the context menu.

VIM

An old school command line text editor. This is really nice to know when you need to edit files on a server or directly on a command line.

```
sudo apt install vim
```

Google Chrome

Download Google Chrome from https://www.google.com/chrome/browser/desktop/index.html be sure to save the file. If this returns no errors then your good to go, however, this sometimes fails and if it does you can clean it up with using Apt.

```
cd ~/Downloads
```

- `cd` - Changes your working directory (Change Directory)
- `~` - Tilde Expansion in this case it's a short cut to the home directory (~ evaluates to /home/jason). So `cd ~/Downloads` will change my current working directory to `/home/jason/Downloads`.

You will have downloaded a file called google-chrome-stable_current_amd64.deb. `.deb` files are software packages designed for Debian based Linus distros.

```
sudo dpkg --install google-chrome-stable_current_amd64.deb
```

- `dpkg` - A package manager for Debian based systems. Primarily used to work with locally installed .deb packages.

```
sudo apt install -f
```

Use Dash to find and open Chrome. Open Dash `Super + F` and type *chrome* into the search field. Click the Chrome icon to launch the Chrome browser. You'll notice the Chrome icon is now in the Launcher right click the Chrome icon and select *Lock to Launcher* from the context menu. Now right click the FireFox icon in the launcher and click choose *Unlock from Launcher* from the context menu.

Visual Studio Code

Visual Studio Code (VSC) is the IDE we will be using to write code. Install Atom using the same steps you used to install Chrome. Remember to pin VSC to your launcher after the install. If you cannot find Atom using Dash try launching it from the commandline by typing `atom` .

Cleanup

Check the contents of your Downloads directory by typing `ls` at the command prompt. You should see the two files we just downloaded and installed.

```
ls ~/Downloads
code*.deb  google-chrome-stable_current_amd64.deb
```

Now type `ls -l` and note the difference between the two result sets.

```
$ ls -l
total 129080
-rw-rw-r-- 1 jason jason 86270030 Feb 16 16:11 code_1.20.0-1518023506_amd64.deb
-rw-rw-r-- 1 jason jason 45892904 Feb 16 16:18 google-chrome-stable_current_amd64.deb
```

Since the have been installed we no longer the files lets remove them from the system.

```
rm ~/Downloads/*
```

- `rm` - removes a file or a folder
- - `-fR` - Force Recursive
  - `-f` - ignore nonexistent files and arguments, never prompt [fn1]
  - `-R` - remove directories and their contents recursively. [fn1]
- * - A wildcard for matching all characters and strings. `rm ~/Downloads/*` will remove everything on the ~/Downloads path

Now typing `ls ~/Downloads` into the command line will return an empty result set.

Meld

Meld is a visual diff tool. This is the default tool called by Atom when doing a file comparison.

```
sudo apt install meld
```

Filezilla

Filezilla is my goto FTP client. While FTP by itself is insecure and not recommended, running FTP over SSH is secure and Filezilla allows us to do just that. We work with FTP and SSH in later lessons.

cURL

The best way to think of cURL is as a browser that is used by code.

```
sudo apt install curl
```

Enable new file creation via right click

```
touch ~/Templates/Empty\ Document
```

Summary

In this lesson you learned

- how to install programs using apt (Linux)
- how to install Debian package using dpkg (Linux)
- how to force a broken install using apt (Linux)

Additional Resources

- VIM Book
- Chrome Dev Tools
- VSC Docs
- Filezilla Docs
- SSH Man Page

Next: LAMP Stack

LAMP Stack)

The LAMP stack (Linux, Apache, MySQL, PHP) is one of the oldest and most mature and popular technology stacks on the web. Ubuntu allows you to install the entire stack with a single command.

```
sudo apt install lamp-server^
```

DO NOT LEAVE THE PASSWORD FIELD BLANK,when prompted; enter the Password for the MySQL root user. Since this is a local development environment just enter *password*; **NEVER** do this on a public facing server.

Apache

To test the server open a browser window and type localhost into the address bar. If your see a page the says *Apache2 Ubuntu Default Page* your Apache web server is up and running.

The Apache web server will create a path called `/var/www/` this is the default path for all of your web application(s). By default root owns this path. Lets make sure we are able to work with path with out requiring elevated privileges.

```
ls -l /var
total 52
drwxr-xr-x  2 root root     4096 Feb 16 15:09
drwxr-xr-x 17 root root     4096 Feb 16 16:06 cache
drwxrwsrwt  2 root whoopsie 4096 Apr 20  2016 crash
drwxr-xr-x 73 root root     4096 Feb 16 16:07 lib
drwxrwsr-x  2 root staff    4096 Apr 12  2016 local
lrwxrwxrwx  1 root root        9 Feb 16 14:28 lock -> /run/lock
drwxr-xr-x 14 root syslog   4096 Feb 16 16:07 log
drwxrwsr-x  2 root mail     4096 Apr 20  2016 mail
drwxrwsrwt  2 root whoopsie 4096 Apr 20  2016 metrics
drwxr-xr-x  2 root root     4096 Apr 20  2016 opt
lrwxrwxrwx  1 root root        4 Feb 16 14:28 run -> /run
drwxr-xr-x  2 root root     4096 Apr 19  2016 snap
drwxr-xr-x  7 root root     4096 Apr 20  2016 spool
drwxrwxrwt  5 root root     4096 Feb 16 16:00 tmp
drwxr-xr-x  3 root root     4096 Feb 16 16:06 www
```

Note the last line of the result set, the one ending with *www*. That is the path we are interested in.

Lets break the result set down by column

- drwxr-xr-x - file permissions by [user]-[group]-[public]
- 3 - the number of links to the file or directory
- root - ownership by user
- group - ownership by group
- 4096 - size
- Feb 16 16:06 - created date
- www - the name of the file or folder

Since you will be the one updating the files on the server, change the ownership to yourself by running the following command. Replace *username* and *usergroup* with your user name.

We dive deeper into Apache throughout this course. For now, bookmark Apache's official documentation

```
sudo chown username:usergroup -fR /var/www
```

Now if you run `ls -l` and you will see yourself as the owner of the path.

PHP

PHP was installed as a part of `lamp-server^` give this a quick test. Open a terminal and type `php -v`

```
php -v

PHP 7.0.13-0ubuntu0.16.04.1 (cli) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
    with Zend OPcache v7.0.13-0ubuntu0.16.04.1, Copyright (c) 1999-2016, by Zend Technologies
    with Xdebug v2.4.0, Copyright (c) 2002-2016, by Derick Rethans
```

- `php -v` This one is pretty simple, we are just asking PHP for it's version number. If you see something begining with *PHP 7.* then is up and running. Make a note of PHP's documentation to review at your leisure.

MySQL

Now Apache is up and running we want to be able to work with our database. We will start by invoking MySQL from the command line.

```
mysql -u root -p -h localhost
```

- `mysql` - tells the system to launch a program called mysql.
- `-u root` - the MySQL user that will be logging into the system. *root* is the default user and has the highest level of privilege.
- `-p` - tells MySQL to provide a password prompt.
- `-h localhost` - the host to which MySQL will connect, in this case it is the local machine.

When prompted type *password* or whatever password you entered during setup into the command line. If successful you will be presented with a MySQL prompt.

```
mysql>
```

We will work with MySQL later in the course, for now bookmark the MySQL Reference Manual. You'll want to familiarize yourself with this guide.

phpMyAdmin

PhpMyAdmin is a webbased admin tool for MySQL. This is written in PHP and runs on the Apache webserver. We will work with phpMyAdmin throughout this course Here is the offical documentation for your convenience. Let's start by installing the software.

```
sudo apt install phpmyadmin
```

As the installer is running you will be presented with several prompts. Arrows keys allow you to navigate the the option menus, the space bar is used to select and de-select while enter accepts your selected option.

- We are not running in production enter *password* at each of the prompts.
- Choose apache2
- Choose YES to db-common

At this point phpMyAdmin is installed but it is not accessible. Now we will configure Apache to grant us access to phpMyAdmin. The following command will use `vim` to open an Apache config file with root level privs.

```
sudo vim /etc/apache2/apache2.conf
```

Type `Shift + G` to move to the bottom of the file. Use the *Up Arrow* to move the cursor to the next to the last line (directly above the line beginning with *# vim:...*).

Now enter insert mode by typing the letter i, if you see --*insert*-- in the bottom left hand corner enter the following.

```
# Include the phpmyadmin configuration
Include /etc/phpmyadmin/apache.conf
```

The first line is just a comment so that you or future dev, admin's, etc will know/remember why that line is there.

Now you will want to restart Apache to upadate the configuration.

```
sudo service apache2 restart
```

- `service` - Apache runs a service so all this command does is tell the system to restart the Apache service.

Open a browser and type *localhost/phpmyadmin* into the address bar. If you see a login page for phpmyadmin then the configuration was successful. Let's go ahead and login using MySQL's root credentials. Smile, you just installed and configured your first web application. While this configuration is fine for a dev environment, you will not want to run this configuration in a production environment. Checkout my post for hardening your phpMyAdmin PhpMyAdmin Post Configuration

Composer

Composer is package manager for PHP libraries. We will git into the details later, for now, let's just install it.

```
sudo apt install composer
```

We have installed several packages, this will be a good time to make sure all of out repositories are still up to date.

```
sudo apt update
sudo apt upgrade
```

Summary

In this lesson you learned

- how to install a full LAMP stack (Linux)
- how to check package versions using the command line (Linux)
- how to login to MySQL client (Linux, MySQL)

Additional Resources

- Apache Web Server Docs
- Building a LAMP Stack on Ubuntu 12.04

Udemy

- Host Your Own Web Development Lab from Home!
- Next: Apache Basics

Apache Basics

Open a browser and navigate to http://localhost and you will land on your machines default apache landing page. That is because the default Apache's configuration points to the path `/var/www/html` , html is folder that contains a single file called index.html. By default Apache looks for files named index.* (where * can be any file extension). Let's configure Apache's default path to `/var/www` and see what happens.

First, open a command line an navigate to sites-available. On Debian based systems this is where Apache stores per site configurations also known as vhost (virtual host) files, on non-Debian systems these may be part of a larger configuration file. Once in the sites-available directory run the command to list the directory contents.

```
cd /etc/apache2/sites-available
ls
```

The ls command should yield the following results.

```
000-default.conf  default-ssl.conf
```

000-default.conf is the default configuration for Apache on Debian based systems. Let's take a look at it's contents. You can read this file without sudo but we want to make some changes to it, so lets `sudo` .

```
sudo vim 000-default.conf
```

You will notice a few default settings more commonly called directives in Apache terms. Right now we are only concerned with the `DocumentRoot` directive. Move your cursor down to line 12 and remove */html* from the end of this line. Remember `i` enters insert mode and `Esc` followed by `:x` saves the file.

The final result will be.

```
DocumentRoot /var/www
```

Now we want to tell apache to reload the configuration. This is a four step process.

- Disable the site configuration `sudo a2dissite 000-default`
- Reload Apache `sudo apache reload`
- Enable the new site configuration `sudo a2ensite 000-default`
- Reload Apache `sudo service apache2 reload`

You can execute all four commands at once with the following.

```
sudo a2dissite 0* && sudo service apache2 reload && sudo a2ensite 0* && sudo service apache2 reload
```

The double ampersand `&&` appends two commands running them one after the other. For example

```
cd /etc/apache2/sites-available && vim 0*
```

Would change your current working directory to */etc/apache2/sites-available && vim 0** and use vim to open any files that match the pattern *0** in this case it would only one file *000-default.conf*. Using wild cards can save a few key strokes and cut down on typos. But be careful you don't open the wrong files.

Once you have reloaded the the configuration, open your browser and return to *http://localhost*. Now, rather that the default Apache landing page you will see a directory listing with a link to the html directory. Clicking this will open the default Apache landing page.

`mod_ssl`

Mod_ssl extends the Apache webserver allowing it to work with TLS (Trainsit Layer Security) connections. While TLS long ago replaced SSL (Secure Sockets Layer), SSL is still the common term used for referring to secure connections be it over SSL or TLS.

In your browser navigate to *https://localhost* notice the *s* in *https* this tells the Apache we want to request a secure version of the website. The page should crash, this is because we haven't told Apache we want to enable a secure version of the site. Now lets open the other configuration in the sites-available directory.

```
cd /etc/apache2/sites-available
sudo vim default-ssl.conf
```

As in the previous configuration we are only concerned with the `DocumentRoot` directive. Move your cursor down to line with this directive and remove */html* from the end of this line. Remember `i` enters insert mode and `Esc` followed by `:x` saves the file.

While the `DocumentRoot` is all we need to change lets take another look at the file. Reopen the file but this time with out `sudo`. Around line 25 you will see the `SSLEngine` directive is set to on. This tells Apache that this configuration wants to use the SSL module, which we have not enabled yet. Notice lines 32 and 33

```
SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
```

These directives tell Apache where to find information about out SSL certificates. By default Apache creates a snake oil certificate. This is an invalid certificate that offers encryption with out verification from a CA (certificate authority) such as DigiCert or RapdidSSL. While this is an invalid SSL certificate, it is adequate for testing an SSL connection. Lets enable the SSL module and load load the new configuration. To exit vim without saving any changes press `ESC` followed by `:quit!`.

Now that you have returned to the command line load `mod_ssl` (the SSL module).

```
sudo a2enmod ssl
```

Now load the new configuration and restart apache. This time we will enable the site using *restart* instead of *reload*. Reload is a graceful restart that allows minor configuration without killing existing connections. Restart is a hard restart of the server that kills all existing connections. Restart covers reload but reload does not cover restart.

```
sudo a2dissite d* && sudo service apache2 reload && sudo a2ensite d* && sudo service apache2 restart
```

Now when you navigate to *https://localhost* you will get an insecure warning. Accept the warning and proceed to the site against the browsers advise. You will now see the same file listing you saw when navigating to *http://localhost*

`mod_rewrite`

Mod_rewrite provide a rules based rewriting engine to Apache. This allows the servers admin to rewrite a users request to do something else. Let's use mod_rewrite to force SSL connections when the user makes an http request.

Be sure the mod_rewrite is enabled

```
sudo a2enmod rewrite
```

Open the Apache's default configuration.

```
cd /etc/apache2/sites-available && sudo vim 000-default.conf
```

Find the following lines

```
ServerAdmin webmaster@localhost
DocumentRoot /var/www
```

and change them to

```
ServerAdmin webmaster@localhost

RewriteEngine On
RewriteCond %{HTTPS} !=on
RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI} [R=301,L]

# DocumentRoot /var/www
```

- `RewriteEngine On` - Activates mod_rewrite
- `RewriteCond %{HTTPS} !=on` - Check for a conditions, it this case the https protocol is not active
- `RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI} [R=301,L]` - If the condition is true http is rewritten to https as a permanent redirect.
- # DocumentRoot /var/www - Comment out the DocumentRoot directive. While not required it would gaurentee no files get served should the redirect fails (in theory).

In a browser navigate to *http://localhost/* and you will be redirected to *https://localhost/*

Summary

In this lesson you learned

- how to reload and Apache configuration using a2dissite and a2ensite (Linux, Apache)
- how to restart the Apache web server using service diretives (Linux)
- how to enable Apache modules (Apache)
- how to use Mod Rewrite (Apache)
- how to force a site to use SSL (Apache)

Next: NPM

NPM

Non-Parametric Mapping or (as it is most commonly but incorrectly known) Node Package Manager (NPM) is a package management system for managing Node.JS (JavaScript) packages. This is akin to Gems in relation Ruby platform, Composer as it relates to PHP or Python's PIP. Under the hood these may be very different in terms of how they operate but practically speaking they all accomplish the same goal; package management.

Node.JS is an insanely popular platform that allows you to build cross-platform applications using web technology. As a result many web development tools are written in Node. These tools can be installed using NPM. We will need to onstall Node.JS to get started with these tools. Follow the Debian and Ubuntu Based Linux Distribution instructions from the Node.JS web site. Install the latest greatest version.

After install run the following commands

```
node -v
npm -v
```

These should return versions >= 8.10 and 5.7.1 respectively.

Summary

In this lesson you learned

- how to install Node.JS (Linux, Node)

Additional Resources

- Debian and Ubuntu Based Linux Distribution

Next: Git

GIT

Git is a free and open source distributed version control system (VCS). Google Trends suggested git is the most popular VCS in the world.Git is now a part of the Ubuntu standard installation so there is nothing to install.

For a list of git commands type

```
git --help
```

GitHub

GitHub is a hosted solution popular among both open and closed source developers and will be the solution we use in this course. While we do not need to install Git we will want to bind our GitHub account to our dev machine.

Setup

The following tutorials will walk you through the process of binding you development machine to your GitHub account using an SSH key. An SSH key uses public key crypto to authenticate your machine against your GitHub account. This allows you to push and pull to and from GitHub without needing to providea username and password.

Generating a new SSH key and adding it to the ssh-agent

Accept the defaults for the following. Do not change the file paths, do not enter any passwords.

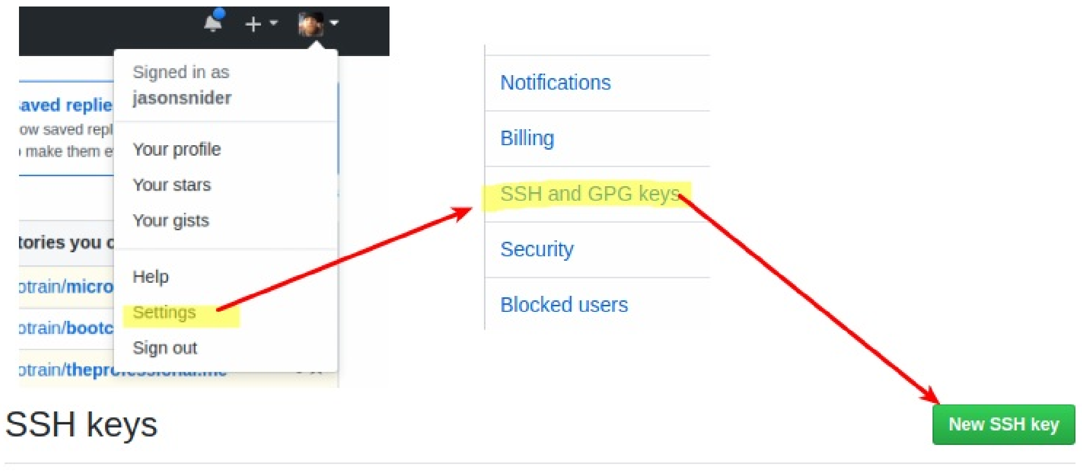
```
ssh-keygen -t rsa -b 4096 -C "YOUR-EMAIL-ADDRESS"
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa
```

Adding a new SSH key to your GitHub account

Intall x-xlip and copy the contents of the key to your clipboard

```
sudo apt install -y xclip
xclip -sel clip < ~/.ssh/id_rsa.pub
```

Log into your GitHub account and find *Settings* in the top right corner under your avatar. Then click on SSH and GPG Keys in the left hand navigation and click the green **New SSH Key** button. Enter a title, this should be somthing that identifies your machine (I usally use the machine name) and paste the SSH key into the key field.



Testing your SSH connection

```
ssh -T git@github.com
```

First-Time Git Setup

Setup your git identity, replace my email and name with your own. For this class we will use VI as our Git editor.

```
git config --global user.email "YOUR-EMAIL-ADDRESS"
git config --global user.name "YOUR-FIRST-LAST-NAME"
git config --global core.editor "vim"
```

In this lesson you learned how to

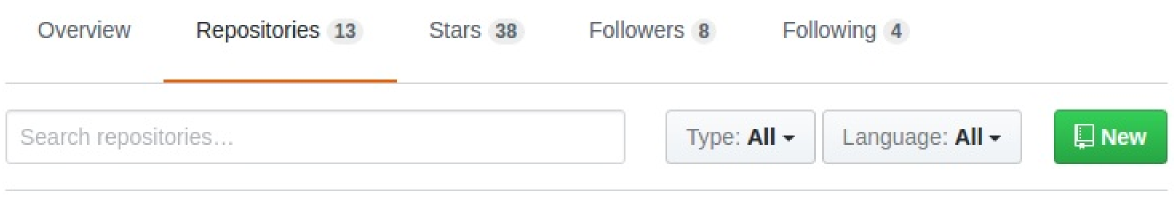
- create an SSH key
- add an SSH key to your GitHub account
- establish a Git identity on your local machine

Exercise 1 - Create a repository

For this exercise we will create your working directory for this course on GitHub and clone into your local environment. You will do most your work from this reporisitory and push the results to GitHub.

Create a new repository

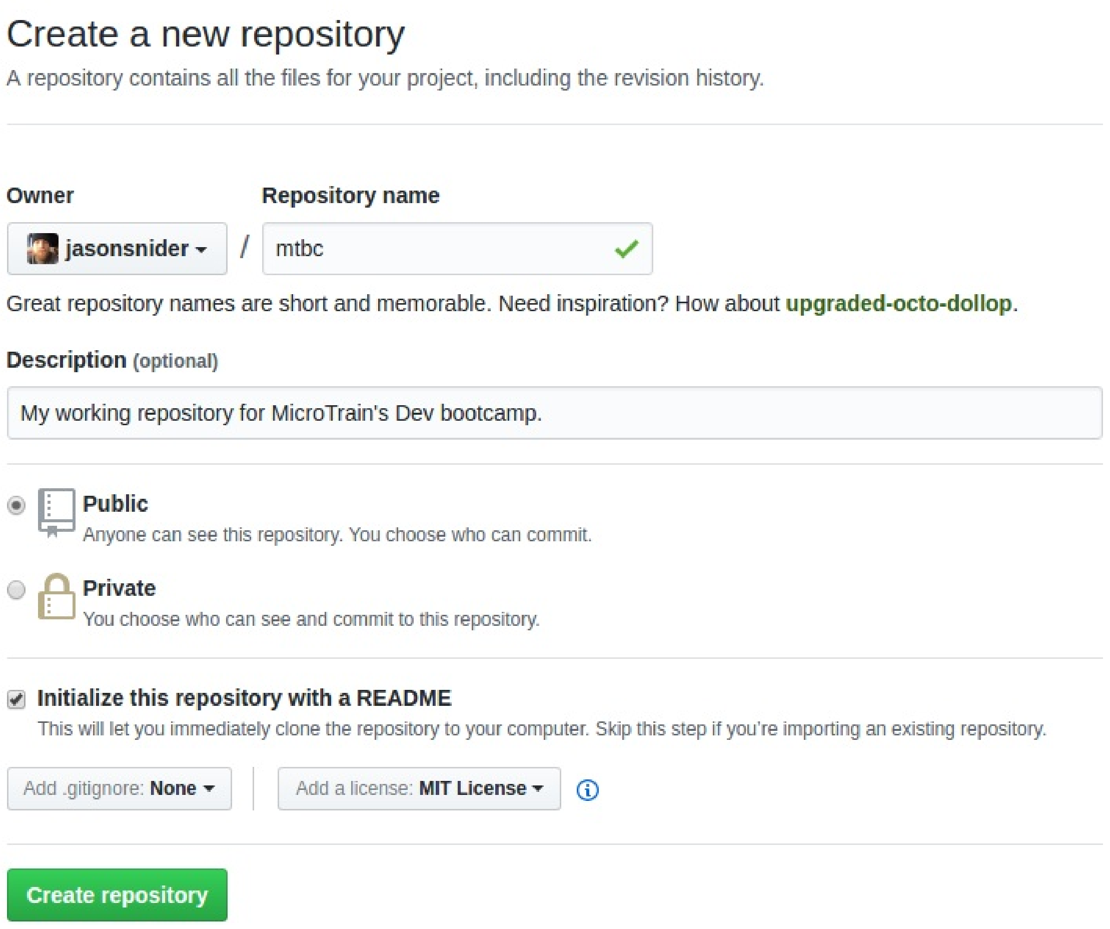
Login to your GitHub account and click on the repositories tab then find the *New Repository* button.



Create a project called mtbc (MicroTrain Bootcamp). This is the project you will use for much of this course.

- Create a description, something like *My working directory for MicroTrain's Dev Bootcamp* (you can change this later).
- Be sure *Initialize this repository with a README* is checked.
- Choose a License, for this course I would recommend the MIT License. Since this isn't product it really doesn't matter. Then

click the create button.



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: jasonsnider / Repository name: mtbc

Great repository names are short and memorable. Need inspiration? How about upgraded-octo-dollop.

Description (optional): My working repository for MicroTrain's Dev bootcamp.

[✓ Public] Anyone can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

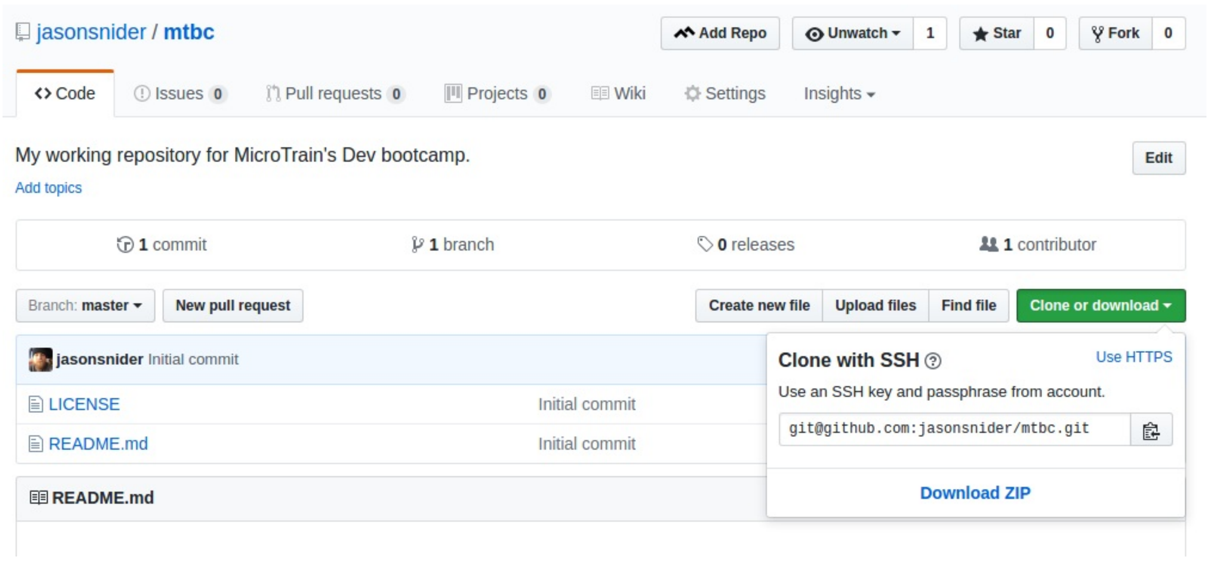
[✓] Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: MIT License

Clone the repository

Now you will want to pull the repository from GitHub onto your local machine. Then clone your fork onto your local machine. After creating your repository you should been directed to the new repository. If not, you can find it under the the repositories tab. Find the *Clone or Download* and copy the URL which will look something like `https://github.com/[your-github-user-name]/mtbc`



jasonsnider / mtbc

My working repository for MicroTrain's Dev bootcamp.

1 commit | 1 branch | 0 releases | 1 contributor

jasonsnider Initial commit

LICENSE	Initial commit
README.md	Initial commit

README.md

Clone with SSH — Use HTTPS

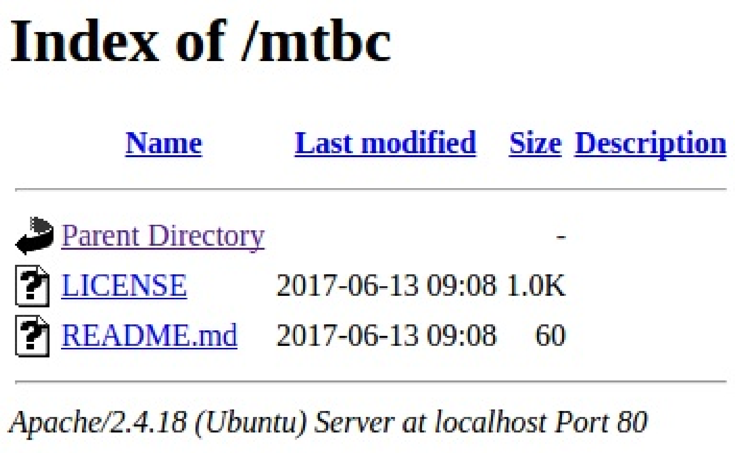
Use an SSH key and passphrase from account.

```
git@github.com:jasonsnider/mtbc.git
```

Now we will clone this repository into the root directory of our web server. This will allow us to access all of our work through the browser by way of *localhost*.

```
cd /var/www
git clone https://github.com/[your-github-user-name]/mtbc
```

Now use the `ls` command to verify the existence of mtbc. If you open your browser and navigate to http://localhost/mtbc you will see the following directory structure.



Index of /mtbc

Name	Last modified	Size	Description
Parent Directory		-	
LICENSE	2017-06-13 09:08	1.0K	
README.md	2017-06-13 09:08	60	

Apache/2.4.18 (Ubuntu) Server at localhost Port 80

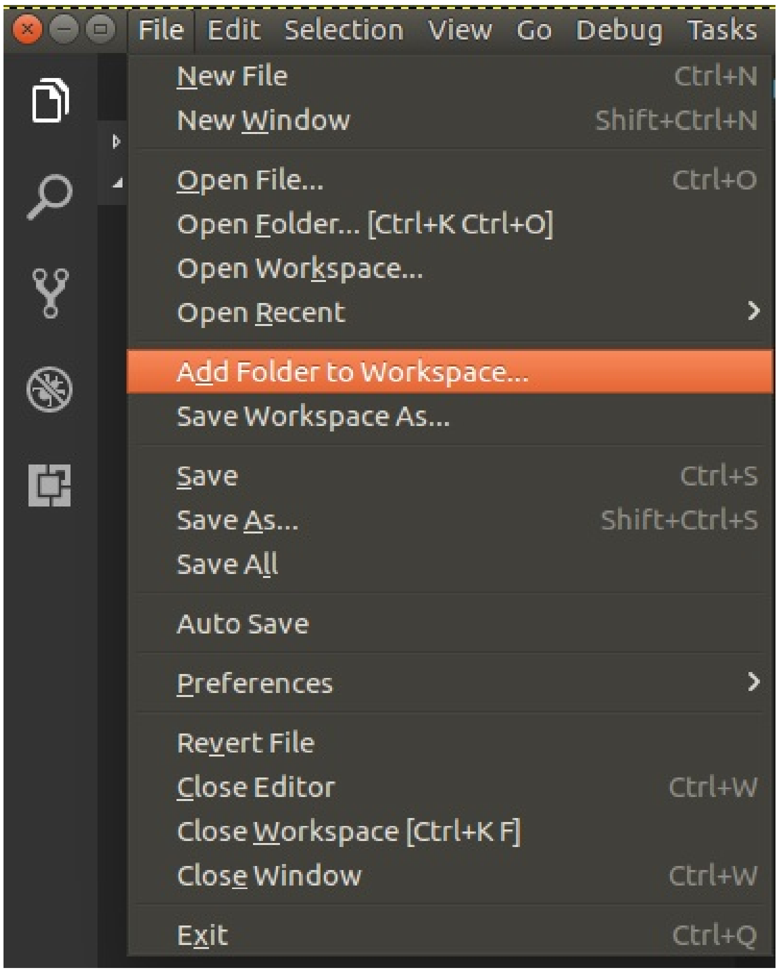
Summary

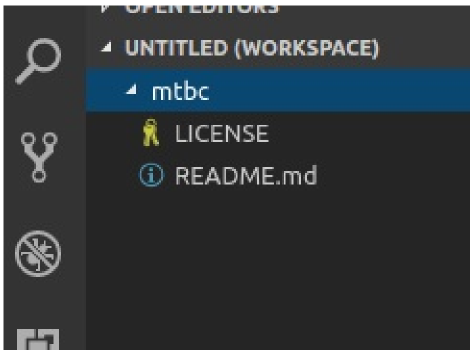
In this exercise you learned how to

- create a Git repository on GitHub (Git)
- clone a repository (Git)

Exercise 2 - Commit a Code Change

Now open VS Code and click into the workspace. Right click choose add a new folder to the workspace. Navigate to */var/www/mtbc* and press `OK` .





Now click on the file README.md from the *mtbc* project folder. README files are a best practice in software development. README files are human readable files that contain information about other files in a directory or archive. The information in these files range from basic infomation about the project team to build instructions for source code. A emerging defacto standard is to write in a format called Markdown (.md). A raw Markdown file should be human readable but if you want a formatted version you can use VS Code's *Markdown Preview* by opening the file and pressing `Shift + Ctrl + V`.

Open the file README.md from the mtbc project folder in the VSCode sidebar and open the *Markdown Preview*. Change the content of the level 1 heading *# mtbc* to *# MicroTrain's Dev Boot Camp*. Save your changes with the keyboard shortcut [Ctrl + S].# MicroTrain's Dev Boot Camp

Open a terminal (command line or CLI) and navigate to the mtbc directory.

```
cd /var/www/mtbc
```

Check your repository for changes, this will return a list of dirty files. A dirty file is any file containing a change.

```
git status
```

Commit your code changes

```
git commit README.md
```

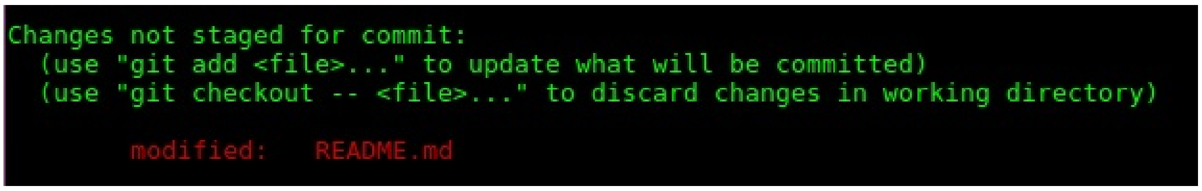
VI will open an ask you to enter a commit message.

1. Press the letter [i] to enter insert mode.
2. Then type the message *Proof of concept version*.
3. Press [esc] followed by [:x] and enter to save the commit message.

Push your changes to the master branch.

```
git push origin master
```

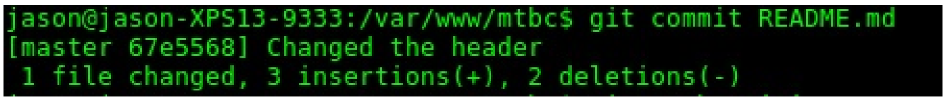
You will see a message that indicates the README.md file has been changed.



```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md
```

```
git commit README.md
```

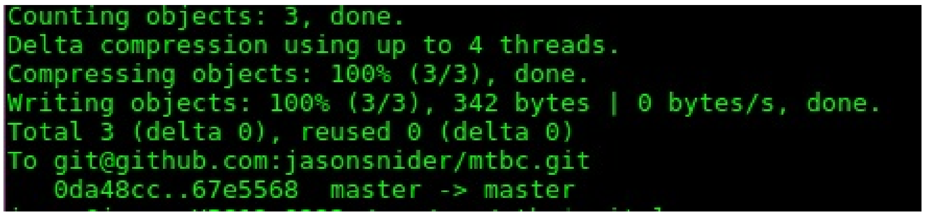
This will open an editor window that ask you to enter a commit message. Enter *Changed the header* and save the file. You will see a message that indicates the changes to README.md file have been committed.



```
jason@jason-XPS13-9333:/var/www/mtbc$ git commit README.md
[master 67e5568] Changed the header
 1 file changed, 3 insertions(+), 2 deletions(-)
```

Finally, push your changes to GitHub.

```
git push origin master
```



```
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 342 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:jasonsnider/mtbc.git
   0da48cc..67e5568  master -> master
```

In the previous example we committed our code changes directly to the master branch. In practice you will never work on a master branch. At the very least you should have a development branch (we call it dev). I like to create branches using the *checkout* command.

```
git checkout -B dev
```

You will see the message *Switched to a new branch 'dev'*.

- `checkout` - This tells git to switch to a different branch.
- `-B` - When following the *checkout* command this tells git to create a new branch. This is a copy of the branch you are

currently on.
- `dev` - The name of the new branch.

In plain English `git checkout -B dev` says make a copy of the branch I am on, name it *dev* and switch me to that branch.

If your are working directly on your dev branch and wanted to push those changes into master it might look like this.

```
git checkout master
git pull origin master
git checkout dev
git rebase master
git checkout master
git merge dev
```

1 `git checkout master` - Switch to the master branch. 1 `git pull origin master` - Pull any new changes into master. 1 `git checkout dev` - Switch back to the dev branch. 1 `git rebase master` - Apply outside changes from master into the dev branch. This will rewind the branch and apply the outside changes. All commits you made after branch deviation will applied on top of the branch deviation. 1 `git checkout master` - Move back to the master branch. 1 `git pull origin master` - Recheck for changes, if any new changes have been applied return to step 3 and repeat. 1 `git merge dev` - Once you have a clean pull, merge your changes into master. 1 `git push origin master` - Push your new changes to the repository.

There is no right way to use git. The only real wrong way to use git is to deviate from that projects branching model. The Diaspora* Project has a very well defined branching model that is typical of what you will see in the real world. I had to come up with one and only one rule it would be to never build on the master branch.

Git Guidelines

A set of guidelines for working with a git repository. See RFC 2119 for keyword usage.

- master - pristine
  - You MUST NOT work on or apply changes directly to dev
  - Only approved changes SHALL be merged into master
- dev - main working branch
  - You SHOULD NOT work on or apply changes directly to dev
  - You SHOULD create a feature/bug (working) and commit changes to that branch
  - The working branch SHOULD always be a decendent of dev

Replace *GITHUB-USER_NAME* with your GitHub user name.

```
#Clone an existing repository
git clone origin git@github.com:GITHUB-USER_NAME/mtbc.git
cd mtbc

#If there IS NOT a dev branch, create one
git checkout -B dev
git push origin dev

#If there IS a dev branch, fetch it
git fetch origin dev:dev
git checkout dev

git checkout -B feature/some-feature
git push origin feature/some-feature

# After you have made you conde changes
# Add and commit the new feature
git add .
git commit -a
git push origin feature/some-feature
```

```
# Integrate the new feature with dev
git checkout dev
git pull origin dev
git checkout feature/some-feature
git rebase dev
git checkout dev
git merge feature/some-feature
git push origin dev

## After integration testing
## Integrate the changes with your production branch
git checkout master
git pull origin master
git checkout dev
git rebase master
git checkout master
git merge dev
git push origin master

## Claenup
# Delete the remote branch
git push origin :feature/some-feature

#Delete the local branch
git branch -D feature/some-feature
```

Summary

In this exercise you learned

- how to create a new branch with the checkout command (Git)
- basic Git commands (Git)

Additional Resources

- Documentation
- ProGit
- 3 Git Commands I use every day

Next: Programming Basics

Chapter 2: Programming Basics

In this section you will be introduced to the most basic concepts of programming; variables, data structures, and control statements.

Programming is little more that reading data and piecing together statements that take action on that data. This is typically done by making a decision tree. In programming these decision trees are constructed using control-strucutres which tend to flow based upon the current value of a memory addresses most commonly known as a variable.

Next: Variables

Variables

A variable is an identifier that points to a given memory address (aka: a pointer). The memory address is simply a storage location. The addressing is delt with by the underlying system and is not something we need to worry about for modern/higher level languages. All we need conern oursleves with for the majority of this course is the name of the pointer (the variable name), it's value and it's data type.

In PHP the following statement would reserve a space in memory indentifed by a pointer called `$a` and set it value to `1` with a data type of ``integer```.

```
var $a = 1;
```

In Bash the following statement would reserve a space in memory indentifed by a pointer called `b` and set it value to `Hello` with a data type of ``string``` (kind-of technically Bash is untyped).

```
b='Hello'
```

In JavaScript the following statement would reserve a space in memory indentifed by a pointer called `c` and set its value to a set of given string (as seen bellow) with a data type of ``array```.

```
var a = [
    'Hello',
    'World',
    'dog',
    'cat',
    'mouse'
];
```

Next: Data Types

Data Types

In short a data type determines how a given variable can be treated, that is to say what operations are allowed on a given varaible. What is the value of 'Hello' + 1? That depends on the language the the data types. In JavaScript, hello + 1 would return 'Hello1' that is because JavaScript is loosely typed while when working with numbers the `+` operator is mathmatical, it also is a used for string concatination when in the string context. Since 'hello' is not a number JavaScript will assume both sides of the equations are a string and as such will performa string concatination rather than than a mathmatical operation. This where we get the meme if 20 + '20' is 2020 you might be a JavaScript developer. In JavaScript 20 + '20' = 2020 but 20 + 20 = 40 this is becasue the latter is 2 numbers while the former is a string and a number.

In PHP `20 + '20'` would return 40 and `'hello' + 1` would return 1 this is because `.` is the opera*tor for concatinating strings and `+` is strictly a mathmatical operator. In the former, since the string can be a number it is converted to a number in the later, the value of 'hello' cannot be numeric but PHP will still try to make since of it, so 'hello' equates to 0;

These are both examples of loosley or dynamically typed languages. In a strongly typed language such a Java or TypeScript `'20' + 20` would throw a type error.

DataTypes in Various Languages

Every language has it's own specific set of data types ad well as it's own set of rules for those types. Most languages will have some notion of the following data types strings, booleans, numbers, null, arrays and objects.

- PHP
- Bash
- JavaScript
- TypeScript

Next: Control Structures

Control Structures

Control structures control the flow of a program.

If we assume the following variables

- *variable a* is an integer equal to **1**
- *variable b* is a string equal to '**hello**'
- *variable c* is an array (or set) with the following elements
    - **Hello**
    - **World**
    - **dog**
    - **cat**
    - **mouse**

Based on the aforementioned variables some basic control structures might look like the following

- While *variable a* is less than **7**, call some arbitrary block of logic and increment *variable a* by **1**.
- If *variable a* exists as **an element** of *variable b*, then return **true**, else return **false**.
- foreach *element* in *variable c*, check to see if that **element** is equal to the *variable b*.

Comparison Operators

As mentioned - PHP, BASH and JavaScript are all dynamically typed meaning the type changes to try to fit a context. Meaning the value of integer 1 and string 1 are the same. So 1 == '1' evaluates to true, in this case the double equals sign says is the value of integer 1 string 1 the same? In contrast 1 === '1' evaluates to false, this asks does string 1 and integer 1 have the same value and data type?

- `==` - equal to, value only
- `===` - equal to, value and type
- `>` - greater than
- `<` - las than
- `>=` - greater than or equal to
- `<=` - less than or equal to

For the following statments assume a = 0 and b = 1

> `=` is an assignment operator, `==` is a comparison operator.

- a == b //false
- a < b //true
- a > b //false
- a == 0 //true
- a == '0' //true
- a === '0' /false
- a >= 0 //true
- a <= 0 //true

Conditional Statements

- if/if-then - If a condition is (or is not) met, then do something
- if-else/if-then-else - If a condition is (or is not) met, then do something. Otherwise, do something else.

- switch - Execute something depending on the state of a given condition.

```
var $a=1;

//if/if-then
if($a === 1){
    echo 'Match!';
}

//if-then/if-then-else
if($a === 1){
    echo 'Match!';
}else{
    echo 'Mot a match';
}

//switch
switch($a){

    case 1:
        echo 'One';
        break;

    case 2:
        echo 'Two';
        break;

    default:
        echo 'Unknown Number';
        break;

}
```

```
var a=1;

//if/if-then
if(a === 1){
    echo 'Match!';
}

//if-then/if-then-else
if(a === 1){
    echo 'Match!';
}else{
    echo 'Mot a match';
}
```

```
a=1;

#if/if-then
if [ "$a" == 1 ]
then
    echo 'Match!';
fi

#if-then/if-then-else
if [ "$a" == 1 ]
then
    echo "Match!"
else
    echo "Mot a match"
fi
```

Loops

- for - executes a fixed number of repetive operations
- foreach/for-in/for-of - executes a repetive operation for each element in a set
- while - executes a repetive operation while a condition is (or is not) true. Requires 1 truth prior to execution.
- do-while - executes a repetive operation while a condition is (or is not) true. Executes prior to first truth.

```
//Iterate a set number of times
for($i=0; $i<10; $i++){
    echo $i++;
}

//Iterate over each item of an array
$items = [1,2,3,4,5,6,7,8,9];

foreach($items as $item){
  echo $item;
}

//Test then execute
$i = 0;
while ($i <= 10) {
    echo $i++;
}

//Execute then test
$i = 11;
do {
    echo $i++;
}while ($i <= 10);
```

```
//Iterate over an object
var obj = {a: 1, b: 2, c: 3};
for (var prop in obj) {
  console.log(prop);
}

//Iterate over an array
let array = [10, 20, 30];
for (let el of array) {
  console.log(el);
}
```

```
#!/bin/bash

ITEMS=( 1 2 3 4 5 5 6 7 8 9 )

# Classic for loop
for ((i=0; i<${#ITEMS[*]}; i++));
do
    echo ${ITEMS[i]}
done

# For in - special array loop
for ITEM in "${ITEMS[@]}"
do
  echo "$ITEM"
done

# While loop
STRING=''
while [ "$STRING" != "Hello World" ]
```

```
do
    if [ -z  "$STRING" ]
    then
      STRING="Hello"
    else
      STRING="${STRING} World"
    fi
done

echo "$STRING"
```

Next: Bash Scripting

Chapter 3: Scripting Basics

In this section you will learn

- Basic control structures
- The basics of Bash scripting

Next: Bash Scripting Basics

Bash Scripting and Programming Basics

In this section you will learn haw to create a Bash script to automate repetitive tasks.

Exercise 1 - Scripting Repetitive Tasks

In the previous lesson you learned the four commands for reloading virtual-host configuration. While that may not seem to cumbersome when your not updating your site all that often; it gets a little rannoying when your testing updates. We will write a Bash script to reduce the burden of this task. A typical Bash script is little more than scripted arrangement or sequence of Linux commands. In addition to Linux commands shell scripts may accept parameters, may utilize control statements, variables, and functions.

Requirements

Write a bash script that will reduce the four commands for reloading a virtual host configuration and restarting a server on a Debian based LAMP stack to a single command.

In the previous lesson we used the following four command to reload a vhost configuration and restart the Apache web server.

```
sudo a2dissite * && sudo service apache2 reload && sudo a2ensite * && sudo service apache2 restart
```

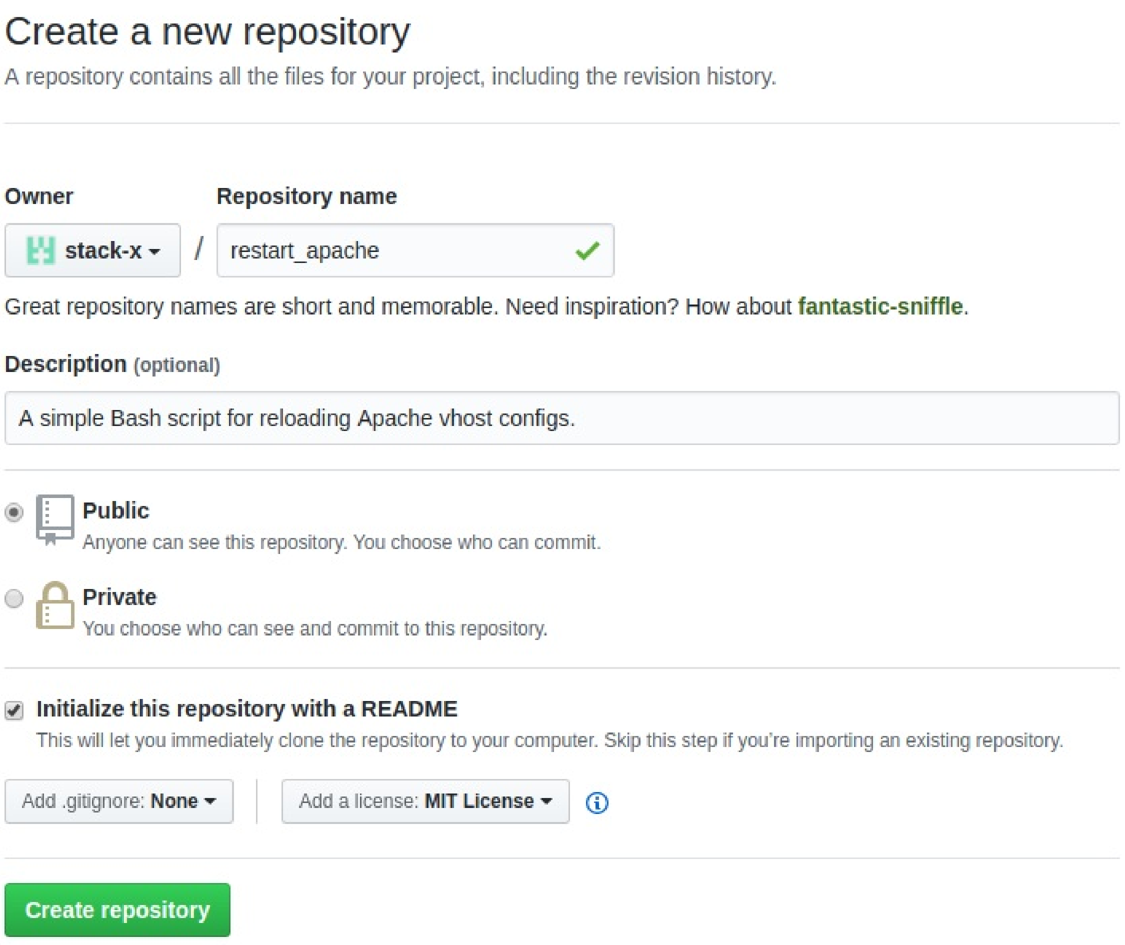
That single line equates to the following four lines.

```
sudo a2dissite *
sudo service apache2 reload
sudo a2ensite *
sudo service apache2 restart
```

Anding these statements togeather makes a copy/paste easier but that is about the only advantage.

</> code **Create a repository and initial commit**

On GitHib create a repository called *restart_apache*.


Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: stack-x / Repository name: restart_apache ✓

Great repository names are short and memorable. Need inspiration? How about fantastic-sniffle.

Description (optional): A simple Bash script for reloading Apache vhost configs.

[✓ Public] Anyone can see this repository. You choose who can commit.

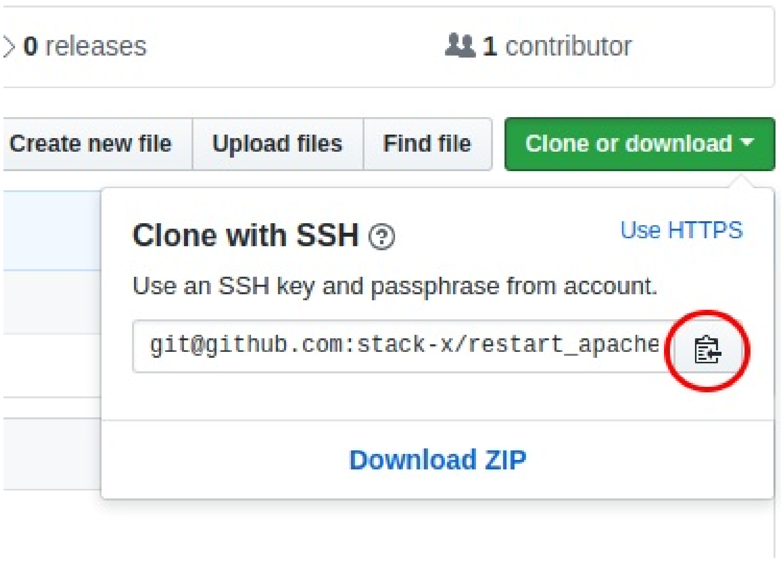
Private — You choose who can see and commit to this repository.

[✓] Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: MIT License

Clone the restart_apache repository onto your local development machine.



Clone with SSH

Use an SSH key and passphrase from account.

```
git@github.com:stack-x/restart_apache
```

```
cd ~
git clone https://github.com/YOUR-USERNAME/restart_apache
```

</> code **Proof of Concept**

Often times I like to start with a simple proof of concept, this is working code working code that either gives you a starting point or talking point. I some projects proof of concept code may represent a complete working solution but may not be considered the optimal solution.

Add *~/restart_apache* as a new folder in your VSC workspace and and create a new file *re.sh*.

By default, Ubuntu executes shell scripts using the Dash interpreter. Dash is faster than Bash by virtue of a lack of features and limited syntax, making it ideal for quickly parsing out a large number of simple start up scripts. Bash is better suited for interactive scripts, since these are typically run as one off programs the performance hit is a non-issue. Our scripts will invoke the Bash shell. Add the following to *re.sh*, this will allow you reload all apache configurations with a single command.

A shebang `#!` followed by a path is used to invoke an interpreter, this must be the first line of the file.

```
#!/bin/bash
```

In Bash any line that begins with a # denotes a comment and does not processed by the interpreter. Comments are used to explain the program to other humans.

```
# Move the current execution state to the proper directory
cd /etc/apache2/sites-available

# Disable a vhost configuration
sudo a2dissite *
sudo service apache2 restart

# Enable a vhost configuration
sudo a2ensite *
sudo service apache2 restart
```

Now we want to make sure the file is executable by adding the executable flag.

```
cd ~/restart_apache
chmod +x re.sh
```

Since many of the commands require root access you will want to sudo this script when you run it.

```
sudo ./re.sh
```

You should see the following results.

```
Site 000-default disabled.
Site default-ssl disabled.
To activate the new configuration, you need to run:
  service apache2 reload
  Enabling site 000-default.
  Enabling site default-ssl.
To activate the new configuration, you need to run:
  service apache2 reload
```

Commit you changes and push them to GitHub

```
git add .
git commit -a
```

VI will open an ask you to enter a commit message.

1. Press the letter [i] to enter insert mode.
2. Then type the message *Proof of concept version*.
3. Press [esc] followed by [:x] and enter to save the commit message.

Push your changes to the master branch.

```
git push origin master
```

Semantic Versioning

</> code Semantic versioning is s community standard that helps you communicate the backwards compatibility of a change. We will use it here as an introduction to the concept.

1. Create the file VERSION.txt
2. Add the text *1.0.0*
3. git add VERSION.txt git commit VERSION.txt

VI will open an ask you to enter a commit message.

1. Press the letter [i] to enter insert mode.
2. Then type the message *Version 1.0.0*.
3. Press [esc] followed by [:x] and enter to save the commit message.

Push your changes to the master branch.

```
git push origin master
```

Add a tag

In addtion to Semantic Versioning a common practice is to tag significant versions.

1. git tag 1.0.0
2. git push origin --tags

Go to your project on GitHub and find everything that is tagged.

Summary

In this exercise you learned how

- to create a basic shell script (Bash, Programming)
- commit your code changes (Git)
- apply semamtic versioning (Git)
- use tags to create code releases (Git)

Arguments and conditionals.

Conditionals

A conditional (aka if-then-else) is a programming construct that uses equality to make decisions. Examples of equality given the variable a is equal to one and the variable b is equal to 2.

If a = 0 and b = 1

- a == b (a -eq b) //false

- a < b (a -lt b) //true
- a > b (a -gt b) //false
- a == 0 (a -eq 0) //true
- a >= 0 (a -gte 0) //true
- a <= 0 (a -lte 0) //true

Arguments

Arguments or parameters are additional data that are supplied when invoking a program, function, method, sub-route, etc. These provide a specific context upon which a called block of logic will execute.

Examples

- vim hello.txt - Opens vim and loads the file hello.txt
- add(1, 2) - Passes two arguments into a function called add (the values 1 and 2). Once might suspect that this would return the number 3.
- Check.word('random') - This passes *random* as an argument into the word method of the (fictitious) Check class. Perhaps we are checking the spelling of the word *random* or maybe this is an argument that tells the method to return a random word.

Exercise 2 - Working with Arguments and Conditionals

In this exercise we will work with the file *~/restart_apache/re.sh* on a *branch called feature/arguments*. Create a feature branch called *feature/arguments*.

```
cd ~/restart_apache
git checkout -B feature/arguments
```

We have reduced four repetitive commands down to a single command, but there is a problem. This only works with a single immutable configuration and an immutable single service directive. It would be far more useful if we could specify which virtual host configuration and which service directive we wanted to use. Lets rewrite re.sh to take some arguments.

Our new shell will take two arguments; the target virtual host configuration and the service directive. Bash accepts arguments using a numeric index which starts at zero, zero however, is the name of the script so the argument that sits at index one will access the first parameter. In Bash, the value of stored variables are accessed using a dollar sign. Combining a dollar sign with a number `"$1"` will allow you to access a given argument.

Our first argument will be the virtual host configuration we want to work with and the second argument will be the service command. We will set these to an aptly named variable to make them easier to work with. We will store the first argument in a variable called *CONFG* and the second in a variable called *COMMAND*. When referencing a variable in bash it advisable to always quote the varaible.

</> code Add the following lines right after *#!/bin/bash*.

```
CONFIG="$1"
COMMAND="$2"
```

The first thing we want our program to do is to verify we have the correct number of arguments. We will do this with an equality check (if-then-else). In Bash `$#` will return the number of input parameters (starting at index number one). We can check this with an if/then statement *if [$# -ne 2] then*. In this context *-ne* translates to *not equal* or in plain English *if the number of input parameters is not equal to 2 then do something*. In our case we will want to provide the user feedback about the expected arguments and exit the program.

Add the following lines to the file. Bellow the CONFIG and COMMAND variables but above the lines from the previous example. In bash `echo` is a command that writes it arguments to the standard output while `exit` stops the execution of the program and returns control back to the caller. In this case both the standard output and the caller would be the terminal.

```
if [ $# -ne 2 ]
then
    echo "$0 requires two paramters {virtual-host} {restart|reload}"
    exit 1
fi
```

Finally, replace the * with a call to the *CONFIG* variable by prefixing CONFIG with a dollar sign *$CONFIG* and do the same for *COMMAND*

```
sudo a2dissite "$CONFIG"
sudo service apache2 "$COMMAND"

sudo a2ensite "$CONFIG"
sudo service apache2 "$COMMAND"
```

./re.sh 000* restart

Commit your changes to *feature/arguments* with the message *Added the ability to specify virtual hosts and service command*.

Push your new feature branch to GitHub, you can delete this branch once the feature is complete.

```
git push origin feature/arguments
```

</> code Update README.txt so people know how to use it. Add something like the following.

~~~
## Usage
Clone the repository or download the latest release.

From a command line call re.sh with two arguments.
1. The vhost configuration
1. The service directive {restart|reload}
```sh
./re.sh 000* restart
```
~~~

Commit the change with the message *README updates*

```
git commit -a
git push origin feature/arguments
```

Then open VERSION.txt and move the version to 1.1.0 and commit with a message of *Version 1.1.0*.

```
git commit -a
git push origin feature/arguments
```

Push your changes to the master branch.

```
git push origin master
```

Merge your changes into master

```
git checkout master
```

```
git merge feature/arguments
git push origin master
```

</> code Tag a new version

1. git tag 1.1.0
2. git push origin --tags

Now that all code changes have been applied to master you can remove your working branch.

```
git branch -D feature/arguments
git push origin :feature/arguments
```

Summary

In this exercise you learned how

- work with arguments and variables (Bash, Programming)
- use an if-then statement (Bash, Programming)
- create a feature branch (Git)
- merge a feature branch into master (Git)
- remove old working branches (Git)

Exercise 3 - Reject unwanted service commands

For this exercise, create a feature branch called *feature/validate*. When you are finished increment the version to 1.2.0 then merge into and push to master.

Requirements

The product owner has requested that we only be allowed to pass *reload* or *restart* into the service command. To achieve we will need to run a test against the second argument to verify it matches a valid command.

To accomplish this we can then test against the value of the $COMMAND argument to make sure it is in the approved list. If it is then we can allow the program to proceed. Oterwise we will return an error message to the user.

We will start with a simple if-then-else statement. This if statement is differes from the previous exercise in that it adds an OR statement. In Bash OR statements are represented as a double pipe `||`. If either of these conditions are true then the code inside the `then` block will execute. Otherwise we will drop into the `else` block.

```
# only allow reload or restart.
if [ "$COMMAND" == "reload" ] || [ "$COMMAND" == "restart" ]
then

else
    echo "ERROR: $COMMAND is an invalid service command {restart|reload}"
    exit 1
fi
```

</> code Once we have our basic statement in place cut and paste the reload/reset logic into the `then` block.

```
if [ "$COMMAND" == "reload" ] || [ "$COMMAND" == "restart" ]
then
    # Move the current execution state to the proper directory
    cd /etc/apache2/sites-available

    # Disable a vhost configuration
    sudo a2dissite "$CONFIG"
    sudo service apache2 "$COMMAND"

    # Enable a vhost configuration
    sudo a2ensite "$CONFIG"
    sudo service apache2 "$COMMAND"
else
    echo "ERROR: $COMMAND is an invalid service command {restart|reload}"
    exit 1
fi
```

```
git add .
git commit -m 'Added the ability to reject unauthorized service directives'
git checkout master
git merge feature/validate
git push origin master
```

</> code Update VERSION.txt to 1.2.0

```
git add .
git commit -m 'Version 1.2.0'
git push origin master

git tag 1.2.0
git push origin --tags
```

Attention to Detail

</> code Check out the format of our new error message. It begins with the word *ERROR:* in all caps. This is good UX in that it remove any ambiguity about what the message. To keep user feedback consitant prefix the error message in the first if statement with *ERROR:*.

```
git commit -am 'Improved messaging'
git push origin master
```

</> code Since this commit does not add or remove it is considered a patch so we will increment version as a patch. Update VERSION.txt to 1.2.1

```
git commit -am 'Version 1.2.1'
git push origin master

git tag 1.2.1
git push origin --tags
```

Exercise 4 - Loops and Arrays

When I think of a loop I'm usually thinking about iterating over or parsing out some sort of a list. This might be an array of service commands or all of the configuration files in the */etc/apache2/sites-available/* directory. In this exercise we build an array of valid service commands and iterate over those commands.

For each element in the COMMANDS array where an element is defined by the variable COMMAND, if an element exists (meaning we have not iterated past the end of the list) `do` echo the value of COMMAND back to the user otherwise `break` the loop or *do echo the value of COMMAND until the list is done*.

Create the file */var/www/mtbc/bash/loop.sh* and make it executable

```
mkdir -p /var/www/mtbc/bash
cd /var/www/mtbc/bash
touch loop.sh
chmod +x loop.sh
vim loop.sh
```

Add the following lines and execute the file.

```
#!/bin/bash

# A list of service commands
COMMANDS=( reload restart )

for COMMAND in "${COMMANDS[@]}"
do
  echo $COMMAND
done
```

Execute the code

```
./loop.sh
```

</> code Commit your code and push it to the master branch of the mtbc project. *Bash exercise 4 - loop example*

Summary

In this exercise you learned

- how to create an array
- how to iterate over an array
- how to create a file using the *touch* command

Exercise 5 - Loop through all files in a Directory

For each file in VHOSTS*PATH array where a file is defined by FILENAME, if an element exists (meaning we have not iterated past the end of the list)* `do` *echo the value of FILENAME back to the user otherwise* `break` *the loop or _do echo the value of FILENAME until the list is done*.

Add the following to */var/www/mtbc/bash/loop.sh* with the following.

```
# List all of the configuration files in the _/etc/apache2/sites-available/_ directory
VHOSTS_PATH=/etc/apache2/sites-available/*.conf

for FILENAME in $VHOSTS_PATH
do
  echo $FILENAME
done
```

Running `./loop.sh` will now yield the following.

```
reload
restart
/etc/apache2/sites-available/000-default.conf
/etc/apache2/sites-available/default-ssl.conf
```

</> code Commit your code and push it to the master branch of the mtbc project. *Bash exercise 5 - loop example*

Summary

In this exercise you learned

- loop through files in a directory

Exercise 6 - Strings

String concatenation is the addition of one string to another typically through the use of variables. Create an executable Bash file at */var/www/mtbc/bash/string.sh* and add the following code.

```
#!/bin/bash

STRING1='Hello'
STRING2='World'
echo "${STRING1} ${STRING2}"
```

</> code Commit your code and push it to the master branch of the mtbc project. *Bash exercise 6 - string example*

Summary

In this exercise you learned how to

- concatenate two strings.

Exercise 7 - Not Empty

The comparison operator -z returns true if a string has a length of zero. *!* is the operator for not so *if [! -z "$STRING"]* equates to true if the string contains any characters.

-z is an equality check for zero.

Create an executable Bash file at */var/www/mtbc/notEmpty.sh* and add the following logic.

- While the variable STRING is not equal to *"Hello World"* continually check the value of string.
- If STRING has a length of zero change the value of STRING to *"Hello"*.
- If STRING has anything other than a zero length append *" World"* to the current value.

```
#!/bin/bash

STRING=''
while [ "$STRING" != "Hello World" ]
do
    if [ -z  "$STRING" ]
    then
      STRING="Hello"
    else
      STRING="${STRING} World"
    fi

    echo "$STRING"

done
```

</> code Commit your code and push it to the master branch of the mtbc project. *Bash exercise 7 - not empty example*

Exercise 8 - String Position

String position allows you extract parts of a string by sepcifiy a numeric index starting at 0. Bash allows you to define string positions using colons with a varaible.

A starting position of 0 will return the entire string.

```
${VARIABLE:strarting-position}
```

A starting position of 0 and end position of 4 will return the first four charaters of a string.

```
${VARIABLE:strarting-position:length}
```

A starting position of 0 and end position of -4 will return everything between first character and fourth from the end.

```
${VARIABLE:strarting-position:end-position}
```

Create the executable Bash file */var/www/mtbc/stringPosition.sh* and add the following logic.

```
#!/bin/bash

STRING="The quick brown fox jumped over the lazy dog"

echo "${STRING:41}"
echo "${STRING:4:5}"
echo "${STRING:36:-4}"
```

This string will return

```
dog
quick
lazy
```

</> code Commit your code and push it to the master branch of the mtbc project. Bash exercise 8 - string position example

Lab

Update the re.sh script such that

- The user will recieve an error message if they attempt to reload/restart a virtual-host file that does not exist in /etc/apache2/sites-available.
- The system will return a list of valid files as a part of the error message.
- The system will exit prior to evaluating the reload command if an invalid virtual-host has been chosen.

Extra Credit

Google "formatting bash", find and read some results about fomatting output text in bash. Look for things like escape characters,font colors, bold and italic text, carrige returns/newlines etc. Using this new knowledge improve the output of the error messages so that they may read in a more intuitive, scan friendly manner.

Questions

1. In Bash, why should you quote variables when referencing them?
2. In Bash, what is *-z*?
3. In Bash, when do you prefix a variable with a dollar sign?
4. How many ways can you come up with to create an array in Bash.
5. What is an array?
6. What does the ***** in */etc/apache2/sites-available/*.conf* do?

Additional Resources

- A Few Words on Shell Scripts
- Bash Guide for Beginners
- Advanced Bash-Scripting Guide
- service
- Better Bash Scripting in 15 Minutes
- Better Bash Scripting in 15 Minutes (Discussion)
- Bash String Manipulation
- FLOZz' MISC » bash:tip_colors_and_formatting

Next: Servers and the Internet

Chapter 4: Servers and the Internet

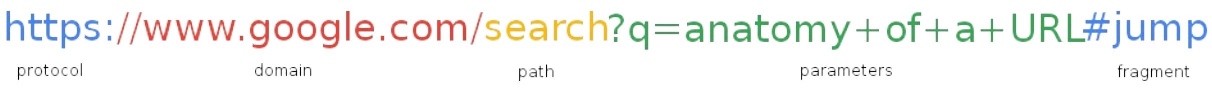
In this chapter you will gain a basic understanding as to how the internet handles traffic and how servers deal with requests.

Next: URL

The Anatomy of a URL

Uniform Resource Locators (URL) is an interface through which a user can communicate with a web page. In modern browsers this is accessed through the address bar. If you were to open a CLI based browser such as wget this would be passed as an argument into the command line..

Let's break this down by taking a look at https://www.google.com/search?q=anatomy+of+a+URI#jump.



PROTOCOL	DOMAIN	PORT	PATH	QUERY STRING	FRAGMENT
https://	www.google.com	:443	/search	?q=anatomy+of+a+URL	#jump

Simply stated a website is a program accessed through using a series of web protocols. For example `http://wwww.example.com` would tell the broswer to go to the root of www.example.com (typically this is a file called index). Everything you see after that is a series of commands. Originally the web served static files so what I am calling commands was really just a directory structure. For example `http://wwww.example.com/blog/post/27` is most likely requesting the 27th post from the websites blog. At one point in time this would have been a file named 27 in a directory called which would have been in a directory called blog. Today, this is most likely a stored in a data base and post is probably a script that has just been instructed to return a given row (in this case 27) from the database.

GET and POST parameters

When dealing with web applications the most common types of requests are GET and POST. These types of requests the user to pass data to the server either through the URL (a GET request) or through a form submission (a POST request). Modern web platforms and programming languages will provide an interface for dealing with the data passed through either of these request types.

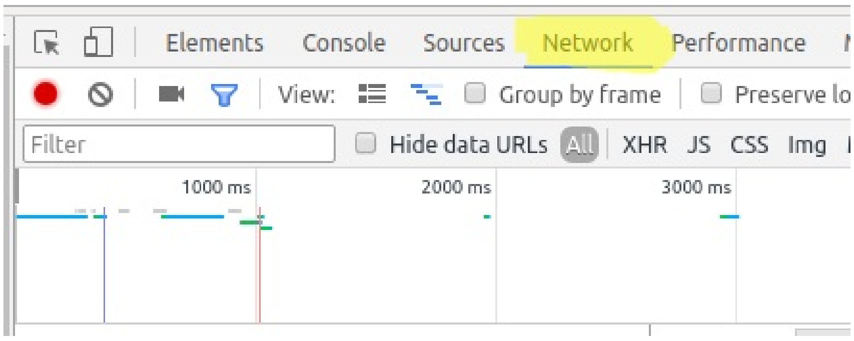
PHP provides access through the use of superglobals `$_GET['email']` and `$_POST['email']`. CakePHP and MVC framework written in PHP uses a request class `$this->request->params->named['email']` and `$this->request->data['email']`. If your running Express on top of Node.JS you'll use something like the following `req.query.email` (GET) `req.body.email` (POST).

It would be common to refer to these practices as retrieving GET parameters (params) and retrieving POST data.

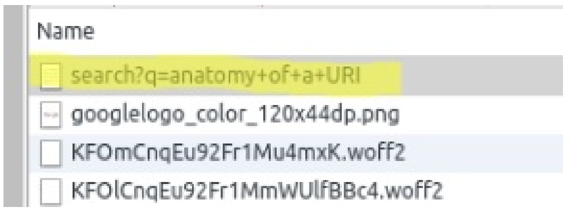
Exercise - Analyze a Server Response with DevTools

Make sure your browser is in focus by cliking anywhere on the screen the open Chrome DevTools by pressing [f12], this will open a dev panel.

Find and click on the Network tab and paste the following URL into the address bar https://www.google.com/search?q=anatomy+of+a+URI#jump.

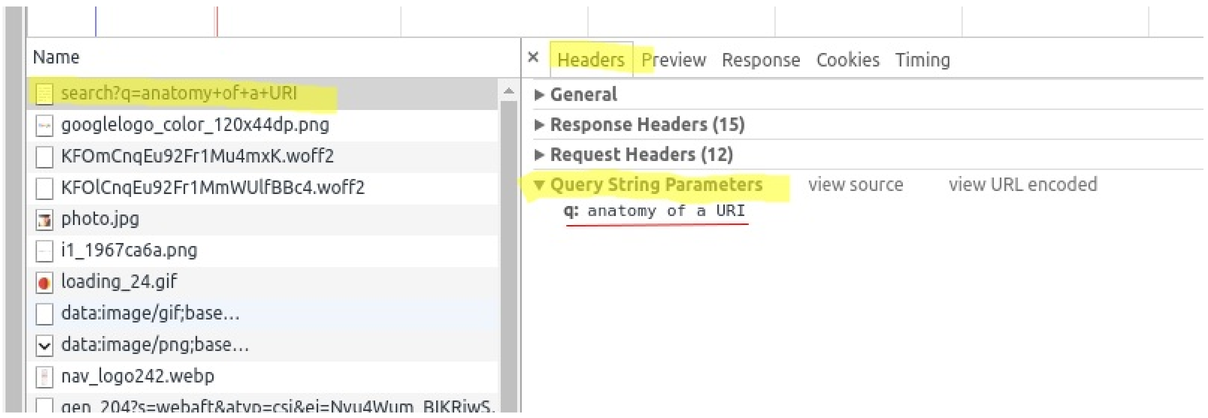


You will see a list of resources that get called when this page loads. Find the one that reads *search?q=atomy+of+a+URI* and click on it.



Name
search?q=anatomy+of+a+URI
googlelogo_color_120x44dp.png
KFOmCnqEu92Fr1Mu4mxK.woff2
KFOlCnqEu92Fr1MmWUlfBBc4.woff2

This will open another tabbed panel that holds information about the page request, click on the headers tabs and find the *Query String Parameters* this shows us the data payload being sent to the server from the browser.



Name
search?q=anatomy+of+a+URI
googlelogo_color_120x44dp.png
KFOmCnqEu92Fr1Mu4mxK.woff2
KFOlCnqEu92Fr1MmWUlfBBc4.woff2
photo.jpg
i1_1967ca6a.png
loading_24.gif
data:image/gif;base...
data:image/png;base...
nav_logo242.webp
gen_204?s=webaft&atyp=csi&ei=Nyu4Wum_BIKRiwS

q: anatomy of a URI

All sections provide good information as to how the page is told to interact with the browser.

Summary

In this exercise you learned how to

- open Chrome DevTools
- use Chrome DevTools to analyze a server response.

Additional Resources

- The Difference Between URLs and URIs
- Uniform Resource Identifier (URI)
- StackOverflow: Node.JS GET Params
- StackOverflow: Node.JS POST Data

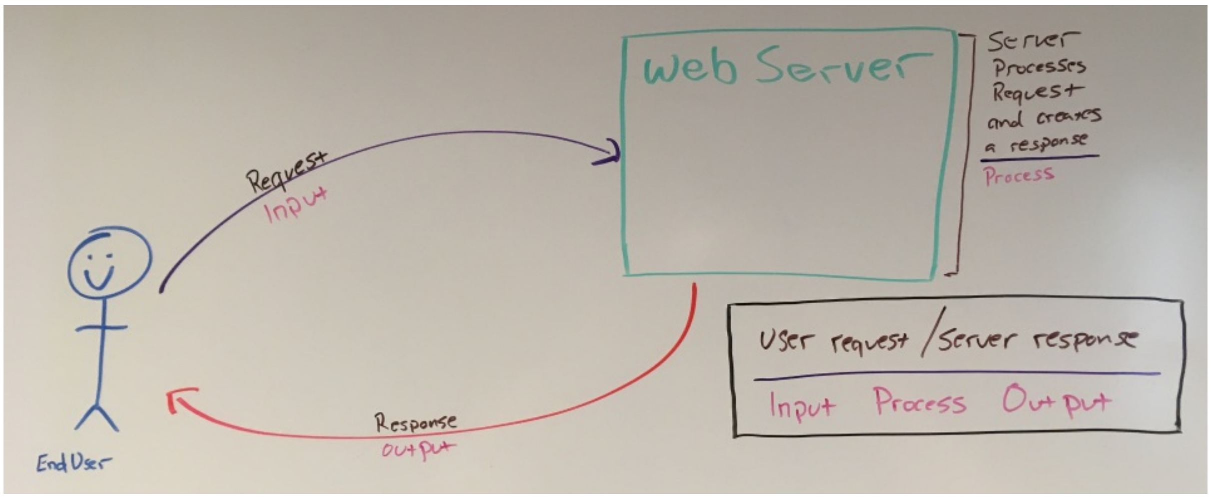
Khan Academy

IP addresses and DNS

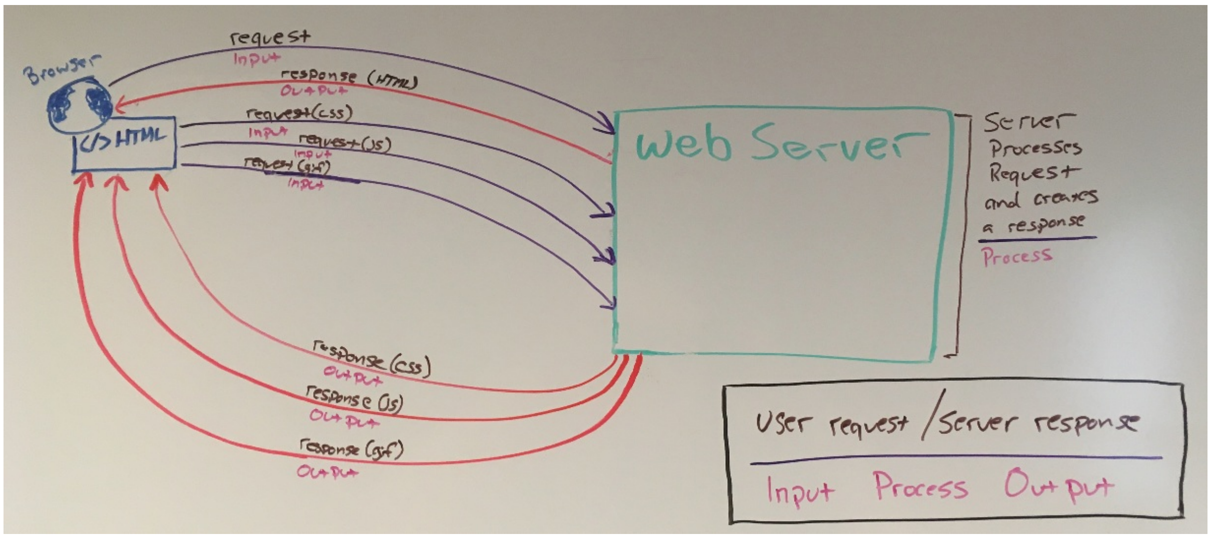
Next: User Requests and Server Responses

User Request and Server Reponse

Every web page is created by a server responding to a user request. The user makes a request, the server process that request and creates a response. That reponse is most commonly in the form of an HTML document. In reality this response could be sent in any imaginable document format. This is an accurate but simplistic view of an IPO (input, process, output model). In practice, this process may be repeated multiple times in loading/interacticng with a single web page.


Web Server
Server Processes Request and creates a response
Process
Request
Input
End User
Response
output
User request / Server response
Input Process Output

Let's replace the work user with client. In this description we will use *end user* to represent a human actor and *client* to represent software, in this case the browser. An end user opens a web browser (the client) and enters *www.youtube.com* into the address bar and presses [ENTER]. At this point the client (browser) makes a request with no paremters to the server at *www.youtube.com* the server sees their are no paramters, process the response and returns the YouTube homepage as an HTML document. This only returns and HTML document that describes how the page should be layout. There are no images, no videos, no styles, no colors, no javascript, no css, no special fonts, and so on[1]. These are all additional requests made by the client. The HTML document contains special references that when interpreted by the client (in this case a web browser) provides instructions for that client to make a request to a server (on behalf of the end user). Each of these additional requests are processed by a server which in turn creates a response and serves that reponse to the client. These responses are not typically HTML documents, rather these tend to by images, videos, stylesheets, javascript files, etc.



In summary a basic web page is served as follows

- A client makes a request to a server.

- That server processes the user request and returns an HTML document.
- The cleint processes that HTML document and makes additional requestsr as instructed by the document.
    - Each of these requests a processed a server.
    - Each of these servers creates a response and serves it to the client accrodingly.
    - The client processes each of these reponses and processes the instruction sets accordinly.
        - These may or may not lead to additional server requests.

Summary

In this section you learned how a user request leads to a server response.

Footnotes

> 1. unless these are hardcoded into the page whitch may be done with descrition. ↩

Next: HTML

Chapter 5: HTML

This unit will focus on the funedmentals of HTML and building a personal website aimed at personal branding. The Mozilla Developer Network (MDN) is great resource for development. Many of the lessons and pre-reading assignmens will refer to MDN.

Pre-reading

- HTML (MDN)
- HTML Basics (MDN)
- Learn HTML in 5 Minutes (freeCodeCamp)

Next: HTML Basics

HTML Basics

Hypertext Markup Language (HTML) is a system of elements and attributes that defines the layout of a web page. This system uses markup tags to represent elements (`<p>This is a paragraph.</p>`) and attributes (`<p style="color: blue;">This is a paragraph with blue text.</p>`) to further describe these elements; to define the context of text and objects on a page.

The offical HTML documentation is available from the W3C this is the standards version of the documentation aimed at browser makers. The MDN web docs in my opinion is a better practical source of documentation.

HTML Elements

A typical web page is rendered an an HTML document. HTML documents are made up of HTML elements (aks tags). HTML has two types of elements; inline elements and block-level elements. An HTML element will typically have an opening and a closing tag. <[element]>Closing tags have a slash</[element]>.

A Basic Web Page

The World Wide Web started off as a markup standard for sharing academic research over the internet. In those days you get by with just a few tags. In most cases you can think about laying out a webpage in the same way you might layout a college paper. A headers, sub headers, paragraphs, images, links, bold, italic, lists and tables. Let's start by exploring a basic web page in the form of an academic paper.

Most HTML documents start with a header. Headers are marked up `<h*>` the *can be any value between* 1-6 *with *1* being the highest. `<h1>` opens a header while `</h1>` closes a header. Thus these are called opening and closing tags respectively. `<h1>` is generally accepted as a page title

Headers

Headers are used for creating sections on a web page. A typical webpage only has a single **h1** tag which servers as a page or article title. Eveything under the **h1** tag should be in some way related to that title. **h2** is the first sub-heading and creates a sub-section under **h1**. Likewise **h3** creates a sub section under **h2** and this repeates all the way down to **h6**. A second **h2** tag creates a new sub section that should be related to **h1** but not necessarily to the first **h2** element. This same notion holds true for all tags all the way down to **h6**.

```
<h1>This is a level 1 header.</h1>
<h2>This is a level 2 header.</h2>
<h3>This is a level 3 header.</h3>
<h4>This is a level 4 header.</h4>
<h5>This is a level 5 header.</h5>
<h6>This is a level 6 header.</h6>
```

Paragraphs

As it's name suggests a paragraph element holds a pargraph. In the academic paper on or many paragraphs may appear under a single header.

```
<p>This is a paragraph, paragraphs have an opening and closing tags. Paragraphs are block level elements with m
argins on the top and bottom.</p>
```

Images

The image element **img** is used to reference an image. Moderen graphical browsers display these as a part of the webpage. Adding an image tag does not place in image in your page. Rather it creates a refrence to an image and may define a space for that image. It is up to the browser to get the image and display as per the authors wishes.

An image has two required attributes **src** and **alt**. **src** is the URI of the image while **alt** holds a non graphical description.

```
<img src="https://www.w3.org/html/logo/badge/html5-badge-h-css3-semantics.png" alt="The Semantic Web">
```

Links

Anchor **a** elements are used to create hyperlinks commonly called links. These are used to link documents together on the web.

An image has one required attribute **href**. This is the URL to any web resource to which you want to link the current document.

```
<a href="https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a">The a tag (HTML Anchor Element)</a>
```

Bold

The *strong* element is typcially rendered in boldfaced type as is the *b* element. The *b* element is considered irrelevent on today's web.

```
<strong>details of strong importance</strong>
```

Italic

The *em* element is typcially rendered in italic type as is the *i* element. The *i* element is considered irrelevent on today's web.

```
<em>emphasis placed here</em>
```

Lists

Ordered **ol** and unordered **ul** present a list of items in a numeric or bulleted format. Each item in a list must be represented as a list item element **li**.

```
<ol>
  <li>list item number 1</li>
  <li>list item number 2</li>
</ol>

<ul>
  <li>first bulleted item</li>
  <li>second bulleted item</li>
</ul>
```

Tables

A table represents tabular data, think of a simple spreadsheet.

```
<table>
  <thead>
    <tr>
      <th>Element</th>
      <th>Display</th>
      <th>Style</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>div</td>
      <td>block</td>
      <tdh>none</td>
    </tr>
    <tr>
      <td>span</td>
      <td>inline</td>
      <td>none</td>
    </tr>
    <tr>
      <td>p</td>
      <td>block</td>
      <td>
        margin-top: 1em;
        margin-bottom: 1em;
      </td>
    </tr>
    <tr>
      <td>a</td>
      <td>inline</td>
      <td>
        color: (internal value);<br>
        text-decoration: underline;<br>
        cursor: auto;<br>
      </td>
    </tr>
  </tbody>
</table>
```

HTML Comments

In programming, comments are strings of text that are not processed by a compiler or an interpretor. Comments are for human consumption and are used to help humans follow the flow of source code. Most languages will define their own special syntax for presenting a comment. In HTML, comments are wrapped in comment tags `<!-- this is a comment -->`.

Content Example

The following would appear inside the body tags.

```
<h1>HTML5 </h1>
<p>HTML5 adds semantics, ARIA and multimedia support to the HTML specification.</p>

<h2>Semantics</h2>

<p>New elements such as <em>article, header, footer, nav, section, aside</em> allow screen readers to better un
derstand your pages layout. This make the web a better place for people with vision problems.</p>

<ul>
  <li><strong>article</strong> - a self contained composition.</li>
  <li><strong>header</strong> - introductory content for a given section.</li>
  <li><strong>footer</strong> - genenrally contains meta data for a given section. </li>
  <li><strong>nav</strong> - provides navigation links for a website/page.</li>
  <li><strong>section</strong> - a stand alone section similar to **div**.</li>
  <li><strong>aside</strong> - indirectly related to the main content.</li>
</ul>

<h2>ARIA</h2>
<p><a href="https://www.w3.org/TR/html-aria/">ARIA</a> allows developer to specify page landmarks by assigning
roles to specific elements. Much like semantic elements these landmarks mak it easier for screen reader to unde
rstand a webpage.</p>

<h2>Multimedia</h2>
```

```
<p>New elements such as <em>audio, video and canvas</em> add built in support for audio, video, gamming and web
 applications.</p>

<h3>Audio</h3>
<p>The <a href="https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio">audio</a> element embeds audio
 into a webpage without the use of akward embed or object elements.</p>

<h3>Video</h3>
<p>The <a href="https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video">video</a> elements embeds vide
o into a webpage without the use of akward embed or object elements.</p>

<h3>Canvas</h3>
<p>The <a href="https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas">canvas</a> element allocates
a section of the page as JavaScript application. Here you can create games, interactive applications or visual
effects. Prior to canvas we were forced to use plugins or other non-standard methods such as Flash, SilverLight
, ActiveX, Java Applets, etc.</p>

<h2>Element Details</h2>
<p>This section describes a number of default attributes associated with various HTML tags.</p>
<table>
  <thead>
    <tr>
      <th>Element</th>
      <th>Display</th>
      <th>Style</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>div</td>
      <td>block</td>
      <tdh>none</td>
    </tr>
    <tr>
      <td>span</td>
      <td>inline</td>
      <td>none</td>
    </tr>
    <tr>
      <td>p</td>
      <td>block</td>
      <td>
        margin-top: 1em;
        margin-bottom: 1em;
      </td>
    </tr>
    <tr>
      <td>a</td>
      <td>inline</td>
      <td>
        color: (internal value);<br>
        text-decoration: underline;<br>
        cursor: auto;<br>
      </td>
    </tr>
  </tbody>
</table>
```

Minimal Template

The following represents the minimal template for a valid HTML5 document.

```
<!DOCTYPE html>
<html>
    <head>
       <!-- meta data goes here -->
       <title><!-- Page Title --></title>
    </head>
    <body>
        <!-- layout and content goes here-->
    </body>
</html>
```

Exercise 1 - Display a Basic Page

Complete the following task, after each task commit your chagnes to Git. When all tasks are complete push all chagnes to the mtbc project on GitHub.

</> code Create the path */var/www/mtbc/html/basic.html* and paste the above template into the file. Then navigate to http://localhost/mtbc/html/basic.html and note the tab in the browser.

</> code The **title** element is on of the few required elements and will be displayed in the browser tab. Change the value of the **title** element to *HTML5* and refresh the page.

</> code Finally add the page content from the example above.

Summary

In this exercise you leanred how to

- build a baisc web page (HTML)
- update a page title (HTML)
- view local changes in a browser (Dev)

Additional Resources

- MDN - HTML
- Introduction to HTML
- Learn HTML in 5 Minutes (freeCodeCamp)
- HTML5 Doctor
- Choosing a Language Tag
- IANA Language Subtag Registry
- Why Character Entities
- XML and HTML Entities
- The W3C Reference

Next: GitHub Pages

Github Pages

GitHub provides a free hosting service called GitHub Pages. This is a standard GitHub repository that is hosted as a website. This allows you to post static web content (HTML, CSS, JavaScript, images, videos, etc) to a repository and that content will be served as a live webpage. This is a front-end only service. The exercises in this section will focus on building a website personal web site tailored to developing a personal brand, building a portfolio, etc.

Exercise 1 - Create your GitHub Pages Repository.

1. Login to github
2. Create a new repository
   - Repository Name: **YOUR-GITHUB-USERNAME.github.io**
   - Description: *My GitHub Pages Site*.
   - [x] Public
   - [x] Initialize this repository with a README
   - Add a license: MIT License
3. Clone the repository

   ```
   cd /var/www
   git clone git@github.com:YOUR-GITHUB-USERNAME/YOUR-GITHUB-USERNAME.github.io.git
   cd YOUR-GITHUB-USERNAME.github.io.git
   ```

4. Add YOUR-GITHUB-USERNAME.github.io.git to your VSCode workspace.
5. Create the file index.html
6. Add the minimal template with your name as title content

   ```
   <!DOCTYPE html>
   <html>
    <head>
       <title>Jason Snider</title>
    </head>
    <body>
    </body>
   </html>
   ```

7. </> code Commit your changes.
8. Push your changes to master
9. Navigate to http://localhost/YOUR-GITHUB-USERNAME.github.io/ to view your changes locally.
10. Navigate to YOUR-GITHUB-USERNAME.github.io/ to view your changes in production.

Summary

In this section you learned how to

- launch a GitHub Pages site

Exercise 2 - Getting Started

This portfolio site will start off with three pages.

- index.html - Your home page, an introduction.
- resume.html - An HTML based resume.

- contact.html - A form that will allow people to contact you.

We will start by adding navigation and and an header to our form. This version of your site might look a bit odd. Don't worry, we will add some CSS later, this will make it look a modern website.

</> code First add navigation to index.html.

```
<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="resume.html">Resume</a></li>
    <li><a href="contact.html">Contact</a></li>
  </ul>
</nav>
```

</> code Make two copies of index.html and rename them to resume.html and contact.html. Locally you will no be able click through all three pages. Commit your changes and push master. Go to the production version of you website and test your changes.

Exersice 3 - HTML Validation

Sometimes our pages do not display as we expect, this is often due to invalid HTML. You can check the validity of your HTML using the W3C Markup Validation Service.

1. Open a browser and go to the W3C Markup Validation Service.
2. Select the third tab *Validate by URI*
3. Enter the address of your GitHub pages site and press the *Check* button.

Meta Data

Meta data is data about data. For a typical web page, the data is the content that falls between the opening and closng heade tags. Meta data helps to describe and classify that data and/or the functionality of your web page. Meta data can be an attribute of a single element or added to the `head` of a document in the form of a meta tag.

Best practices are things that ought to be done given there is not a good reason not to and provided their is not an alternative that better suits a given situation. If I had to pick three pieces of meta data that should always be implemented, they would be as follows.

- `<html lang="en">` - Defines the language of the web page. This would most likely be used by assistive technologies such as screen readers or an automated translator.
- `<meta charset="UTF-8">` - Defines the character set you are using so that there will be no confusion between your source code and the rendering engine. For a data driven web site you will want your websites encoding to match that of your database; UTF-8 is the most common encoding.
- `<meta name="viewport" content="width=device-width, initial-scale=1.0">` - Used by the browser to allow the developer of the site to declare how the site should be viewed across devices.

Exercise 4 - Meta Data Best Practices

</> code Update all pages on your GitHub site so that

- The language is declared as *English*.
- The charset is declared as *UTF-8*.
- The view port is declared as *content="width=device-width, initial-scale=1.0*.

Commit all changes and push all changes to master

HTML Elements

As stated in the previous lesson; HTML elements more commonly know as tags are bits of markup that provide semantic meaning to text and objects. This markup is interpreted by outside programs (such as browsers, bots, spiders, crawlers, etc) which will often act on your content based on that contents markup. For example `<h1>title</h1><h2>Sub Title</h2>` tells the program reading your page that *Sub Title* belongs to *title* and that *title* should be treated as title of the page (or all content until the next `<h1>` tag is encountered) while **Sub Title** identifies the next block of or all content until the next `<h2>` or `<h1>` tag is encountered. The original goal of HTML was to provide a common format in which we could send academic research papers over the wire. In short, HTML was designed to mimic a word processor. The body of one of those documents may resemble the following. In most cases your HTML elements will have both an opening and a closing tag. Elements open with `<[element]>` and close with `</[element]>` the difference here is `<` vs `</` .

Sample Markup

Just a quick review.

```
<h1>HTML Elements</h1>
<p>HTML elements more commonly know as tags are bits of markup...</p>
<h2>HTML Global Attributes</h2>
<p>Attributes bring your markup to life. Attributes allow for programming...</p>
<h3>Event Handler Attributes</h3>
<p>Event Handler Attributes (UI Events) allow a user to interact...</p>
<p>Here a list of...</p>
<ul>
    <li>Item one</li>
    <li>Item two</li>
    <li>Item three</li>
</ul>
<h2>Summary</h2>
<p>In summation...</p>
```

Lets review some of these tags.

- h1, h2, h3, h4, h5, h6 - HTML supports 6 header levels, these should always be nested.
- p - Identifies text as a paragraph.
- ul - Identifies an unorganized list. This will create a bulleted list. An unorganized MUST contain one or more list items `<li>` .
- ol - Identifies an organized list. This will create a numbered list. An organized MUST contain one or more list items `<li>` .
- li - A list item, represents an item in either an unorganized or organized list. This MUST be wrapped in a `<ul>` or `<li>` element.
- div - Divs are block level elements that are used to represent divisions in an HTML document. These are typically used to divide a page into sections. These may be used for a logical page division or as anchor to apply style and attributes.
- a - An anchor tag. This is used to create links and is arguably the foundation of the World Wide Web. An anchor tag becomes a link by adding a href (Hypertext REFerence) attribute `<a href="https:\\www.">Example</a>` .

Sample Markup

Just a quick review

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>World</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <h1>Hello World</h1>
    <p>Welcome to my web site.</p>
    <h2>HTML Elements</h2>
    <p>
    To paraphrase the lesson text [...] For example &lt;h1&gt;h1&lt;h2&gt;title&lt;/h1&gt;... Notice the use of
 character entities when wanting show the tags in HTML.
    </p>
    <h2>Character Entities</h2>
    <p>Since the keyboard does not have a &copy; key we need a way to reference this so we say &amp;copy;. Addi
tionally, greater than and less than are interpreted as HTML tags. These are examples of symbols that we may wa
nt to display but will not be able to with out a work around. This is where character entities come into play.
</p>
    <ul>
        <li><a href="https://stackoverflow.com/questions/1016080/why-are-html-character-entities-necessary">A S
tack Overflow thread on the topic.</a></li>
        <li><a href="https://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references">A Wikipedi
a artcile on the topic.</a></li>
        <li><a href="https://dev.w3.org/html5/html-author/charref">The W3C reference.</a></li>
    </ul>
    <h2>Summary</h2>
    <p>In summation...</p>
  </body>
</html>
```

HTML Global Attributes

Attributes bring your markup to life. Attributes allow for programming hooks, style and meta-data to be applied to your web page. While HTML has defined a number of standard attributes it allows for custom attributes to be defined. Attributes are key to value pair that can add additional semantics, style or functionality to a element. We learn more about attributes as needed.

Image Tag

When it comes to working with attributes, the image (img) element is a great place to start. Image is a self-closing tag. Self-closing tags do not require a closing tag because they do not contain any content, they only use attributes to mark content to the screen.

Image has two required attributes src and alt `<img src="..." alt="...">`. Src tells the document where to find the image to be displayed while alt displays alternative text in case the image cannot be displayed. In most cases an attribute will always have a value, this is written as attribute="Some Value". This is often referred to as a key-value pair. The attribute or left side of the equation is the key and the right side of the equation is the value (conceptually speaking it's key="value").

Before you can add an image, you will need an image to link to. Lets use a Gravatar. Gravatar is a free service that allows a user to upload an avatar that will follow them from site to site by hashing the users email address into the name of a file. This is popular in the dev community and used by other services such as GitHub.

Head over to Gravatar and create a profile and use the provided URL, if you do not want to create a Gravatar account use the following URL.

- *https://www.gravatar.com/avatar/4678a33bf44c38e54a58745033b4d5c6?d=mm&s=64*

You would mark this up as:

```
<img src="https://www.gravatar.com/avatar/4678a33bf44c38e54a58745033b4d5c6?d=mm&s=64" alt="My Avatar">
```

Exercise 5 - Introduce Yourself

</> code Update index.html such that

- The title tag reads *Hello, I am YOUR-NAME*
- Add an H1 element to the body of the document.
    - The contents of this tag SHOULD read *Hello I am YOUR-NAME*
- Create a paragraph tag and say a few sentances about yourself.
- Add an image element to the paragraph tag above the text you just added.
    - The src attribute MUST point to a Gravatar image
    - The value of alt attribute SHOULD be YOUR-NAME

The Style Attribute

Cascading Style Sheets (CSS) is a language for describing the style of an element. The style attribute in HTML allows you to add a CSS description to a single HTML element. Describing a font's color is a common use of CSS. In CSS we describe style using property's which are key to value pairs separated by a colon [key]:[value]. Using the color property will allow me to describe the font color of an element. We will do a deep dive on CSS late, for now we will give you what you need.

Exercise 6 - Style an Element

This exercise introduces you to inline styles. Due to there blocking nature we will not use these in production. We will learn more about inline styles in latter leasons.

</> code Change index.html such that

- Text wraps around the image.
- The image provide some margin between itself the the text.
- The image is presented as a circle.

CSS Attributes

- border-radius: 50%; - Rounds the corners of a give element. A value of 50% makes a complete circle.
- float: right; - Floats an element to a given position. All surrounding content flows around that float.
- margin-right: 1em; - Provide 1em of margin to the right side of an element.

```
<img style="border-radius: 50%; float: right; margin-right: 1em;" src="..." alt="...">
```

Next: Contact Form

Contact Form

Traditionally, forms have been the most common way to collect data from a user. A form submission is the simplest way to post data to a server. This section will start with a simple POST request and end with complex processing.

Form tags `<form></form>` are used for creating forms in HTML. Every form should have at least two attributes *action* and *method*.

- action - the web address to which the form data will be sent.
- method - the type of request the form should make (probably GET or POST).

Form Elements

While HTML has a slew of form elements you can mostly get by with just a few.

- input - uses the type attribute morph into various fields.
  - text - covers single line input fields
  - date - provides a date picker
  - hidden - a hidden field
  - checkbox - provides checkboxes
  - radio - provides a radio option
  - file - upload files to the server
  - button - provides a submit button
- textarea - a large field for entering multiline text.
- select - Provide a picklist.
- label - Provides a label for a given element.

Label elements use the for attribute to associate a label to given form field. The form field requires an **id** attribute with a value that matches that of a given **for** attribute. While visually there is no advantage to using **for** and **id** it helps screen readers communicate to visually impared users.

> Never omit form labels. If you want to hide use `display:none;` in CSS. This will hide it from the UI while keeping it visable to screen readers.

Additional Form Fields Attributes

- name - provides a data key to the form element. This is the name of the field passed in the payload (required).
- type - defines the type on input field (input fields only) (required).
- value - allows you to prepopulate the value of an input field (optional).
- checked - auto checks a checkbox (optional).
- selected - pre-selects a picklist option (optional).

CodePen Demo **Accessible Form Examples**

```
<!-- Plain text input -->
<div>
  <em>Sample plain text entry</em>
  <label for="name">Name</label>
  <input id="name" name="name" type="text">
</div>
<!-- Datepicker -->
<label for="date">Date</label>
<input id="date" name="date" type="date">

<!-- Textarea -->
```

```
<label for="description">Description</label>
<textarea id="description" name="description"></textarea>

<!-- Picklist-->
<label for="pickOne">Choose an option</label>
<select id="pickOne" name="pickone">
  <option>Please Choose an Option</option>
  <option value="option1">Option 1</option>
  <option value="option2" selected>Option 2</option>
  <option value="option3">Option 3</option>
</select>

<!-- Radio Options -->
<fieldset>
  <legend>Choose yes or No</legend>

  <label for="r1">Yes</label>
  <input id="r1" name="yn" value="y" type="radio">

  <label for="r2">No</label>
  <input id="r2" name="yn" value="n" type="radio">
</fieldset>

<!-- Checkbox Options -->
<fieldset>
  <legend>Choose all that apply</legend>
  <label for="c1">Option 1</label>
  <input id="c1" name="all" value="0" type="checkbox">

  <label for="c2">Option 2</label>
  <input id="c2" name="all" value="1" type="checkbox">
</fieldset>

<!-- Form Buttons -->
<input type="submit" name="submit" value="Submit">
<input type="reset" name="reset" value="Reset">
```

Formspree

Normally you would implement a contact form by implementing some backend code on your webserver. Since GitHub Pages does not allow you to execute server side (aka backend) code we will use a free service called Formspree. This will allow upto 1000 emails per month through our web form. Head over to Formspree and give it a test, you will be sent a confirmation email which you must confirm to continue using the product.

> By adding your email address to this form you are making it publicly available(or you could say potentially increasing it's public exposure). If this is an issue for you create an email account for soley for this purpose or even a service such a https://www.mailinator.com/. *If you choose the latter be aware that these inboxes are temporary and public.*

Exercise 1 - Basic Form

When building a contact form think about what information you need. This will be a simple form so we will only ask for a few things: name, email and a message. This gives us a total of three form fields.

</> code Open *contact.html* and add opening and closing `form` tags.

```
<form action="https://formspree.io/YOUREMAIL@EXAMPLE.COM" method="POST"></form>
```

</> code Collect the name of the person contacting you. We use N/name as the label content, for, id, and name values. Note the div tags, they are not required but they will help us later when we go to style the form.

```
<div>
  <label for="name">Name</label>
  <input id="name" type="text" name="name">
</div>
```

</> code Collect the email of the person contacting you. We use E/email as the label content, for and id values but we will change the name to _replyTo this will allow us to access Formsprees relpyTo feature of setting the replyTo address to the user supplied value.

```
<div>
  <label for="email">Email</label>
  <input id="email" type="text" name="_replyto">
</div>
```

</> code We will create a label and a text area to collect the mesage from the person contacting us.

```
<div>
  <label for="message">Message</label>
  <textarea id="message" name="message"></textarea>
</div>
```

</> code We will create a hidden form field with the name of _subject. This will aceess another feature for dealing with the email subject. We will make this a hidden field so that we can control what it looks like in our inbox.

```
<div>
  <input type="hidden" name="_subject" value="New submission!">
</div>
```

</> code Finally, we will add a submit button.

```
<div>
  <input type="submit" value="Send">
</div>
```

Add the file thanks.html (this should be a copy of contact.html) to your GitHub Pages site. Add a header and message thanking the user for contacting you.

Exercise 2 - Thank You

</> code Make a copy of the file resume.html and name it *thanks.html*. Change the title to say "Thank You" and add a nice massage for the user.

</> code Add a hidden field to your form and set the name to *_next* add a value attribute the points to the thanks.html page on you GitHub pages site. This use another special feature of Formspree that the sends the user to a target page aftet the form submit.

```
<input type="hidden" name="_next" value="//YOUR-GITHUB-USERNAME.github.io/thanks.html">
```

Exercise 3 - Captcha and Honeypots

Completely Automated Public Turing (Captcha) is any test that would be trival for a human to solve but difficult if not impossible for a computer to solve. We use these to reduce spam and wasted resources by attempting to fiter out robots (or non-human traffic) all Formspree forms provide captcha by default.

</> code Add a hidden field to your form and set the name to *_next* add a value attribute the points to the thanks.html page on you GitHub pages site. This uses another special feature of Formspree that rejects any form for which the hoenypot is not empty. This is a second layer ontop of Captcha making it redundant but we show it to demonstrate another technique for dealing with spam.

> Any bot that is is aware of Formspree has likely been updated to account for the _gotcha name field. This would be more effective on a custom for, we will disscuss this in later lessons.

```
<input type="text" name="_gotcha" style="display:none">
```

Summary

In this lesson you learned how to

- create a webform
- interact with a thrid party service

Additional Resources

- Creating Accessible Forms

Next: Resume

HTML Resume

Using MicroTrain's preferred resume template as a model create an HTML resume. This template hasbeen preloaded with skills you will learn in this course. For now do not worry about centered text, columns and left/right justiifcations. We will deal with all of that in CSS. For now We will build a top down version of your resume. Startgin with your name in an `h1` tag. You might consider putting your contact information in a `div` using an HTML Entities, DEC or Hex values.

> **Reminder:** This will be on the public web, do not add any details you do not want to have posted to the web. You might consider showing only your city and links to LinkedIn and GitHub

```
<h1>First Name Last Name</h1>
<div>
  <a href="..." target="_blank" rel="noopener">LinkedIn</a>
  &#x25CF;
  Some City, IL
</div>
```

You will follow that up with your desired job title in an `h2` tag then a paragraph `p` containing a breif summary, then a list `ul` of one or two highlights. Core Competencies is an `h3` followed by two bulleted list `ul` (just keep them stacked for now). Certifications / Technical Proficiencies willbe an `h2` followed by another bulleted list (CSS will make this no look like a list). Prfoessional Exepeience is back to an `h2` and each job starts with an `h3`. From here try to figure out those tags for yourself.

Lab - HTML Resume

Create an HTML version of your resume for your GitHub Pages site. Commit your changes and push them to master.

Lab 2 - Semantic Meaning

Use HTML5 sectioning tags such as `main` and `section` to segment your resume into logical sections. This will help better commuicate your resumes structure and provide additional support for CSS.

- Wrap the entire resume in `main`.
  - You may choose to wrap the resume in an `article` tag and place that inside of main.
- Start a `section` before an `h*` end the section bfore the next `h*` tag of the same level.

Additional Resources

- UTF-8 Geometric Shapes
- Using HTML sections and outlines
- How to Use The HTML5 Sectioning Elements
- HTML5 Resume: Jared Pearce

Next: MetaData

MetaData

MetaData comes in many forms from the `meta` elements in the the document head to microdata and aria attributes applied directly to your HTML tags or even in a separate text file.

HTML

Improve SEO with `<meta>`

Search Engine Optimization (SEO) is the practice of improving your site compatibility with a search engine which, in theory, should help to increase your traffic. Meta data is one ingredient. We will start with the classics *description* and *keywords*. When writing a site description your looking to sum up a page in 25 words or less the description MUST be relevant to the content on the page. The same idea applies to keywords 25 or less comma separated words or short phrases that are relevant to the page in question. Google will penalize you if you try to cheat. In both cases we will use the meta tag with the name and description attributes.

The name attribute denotes the type of meta data while the content attribute contains the description, keywords, etc.

```
<meta name="description" content="The best thing about hello world is the greeting">
<meta name="keywords" content="">
```

MicroData

A search engine is the most common classifier of web pages. Typically, web pages are classified using seed data against a machine learning type of algo. Some of these are very good while others are not. Applying a micro data schema to your pages markup can help these algos better classify your data. Typically these will be waited against some sort of AI as a means of fraud detection. For instance, Google will likely honor meta data so long as it doesn't think you you are trying to game the system, if it detects something of that sort, it may de-index your site.

A good example would be a job posting. If I'm posing job ads I would want to apply this schema assuming other services are interested in pull in job ads and making use of its data.

Aria

Accessible Rich Internet Applications (ARIA) is a standard for helping a web page work with a screen reader. You saw a sample of this earlier with `role="navigation"` .

.txt

robots.txt

robots.txt is a file that sits in your sites root directory. This tells allows you to provide a set of preferences to well behaved search engines. At the very least you should allow all bots until you have a reason not to. Bad bots will ignore this so trying to keep those out using a robots.txt file is pointless, keep it simple and assume all bots are good. This will prevent you from making mistakes that block the good bots.

Exercise

Create the path */var/www/robots.txt* and add the following lines.

```
User-agent: *
Disallow:
```

humnas.txt

humans.txt is a file that gives credit to the sites creators. While it is not required it's a nice touch and if they solve the spam problem they will give you a way to submit your site to there directory, which is a link back so that is always good.

humans.txt requires a tag in the head plus the text file with all of the content

```
<link rel="author" href="humans.txt" />
```

```
/* TEAM */
Your title: Your name.
Site: email, link to a contact form, etc.
Twitter: your Twitter username.
Location: City, Country.

/* THANKS */
Name: name or url

/* SITE */
Last update: YYYY/MM/DD
Standards: HTML5, CSS3,..
Components: Modernizr, jQuery, etc.
Software: Software used for the development
```

security.txt

security.txt lets security researchers know how to get hold of you should an issue be found.

Images

favicon.ico This is the image that appears in the browser tab beside the title. This is found in your websites root directory.

browserconfig.xml For creating custome tiles in windows.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Please read: https://msdn.microsoft.com/en-us/library/ie/dn455106.aspx -->
<browserconfig>
    <msapplication>
        <tile>
            <square70x70logo src="tile.png"/>
            <square150x150logo src="tile.png"/>
            <wide310x150logo src="tile-wide.png"/>
            <square310x310logo src="tile.png"/>
        </tile>
    </msapplication>
</browserconfig>
```

- tile.png
- tile-wide.png
- icon.png

manifest.json

```
{
    "icons": [{
        "src": "icon.png",
        "sizes": "192x192",
        "type": "image/png"
    }],
    "start_url": "/"
}
```

Additional Resources

- What’s in the head? Metadata in HTML
- Steps to a Google-friendly site
- The Beginner's Guide to SEO
- How to write meta descriptions for SEO
- Getting started with schema.org using Microdata
- WAI-ARIA Overview

Next: CSS

Chapter 6: CSS

In this section you will learn the basics of

- CSS
- Sass
- Gulp
- CSS Based Layouts

Next: CSS Basics

CSS

Cascading Style Sheets (CSS) is a language that describes the style of a web page. CSS uses selectors and definitions to apply the these style descriptions or styles to a page.

Lets look at CSS Zen Garden, click on the download example html file link and open the file in your browser. This is just a plain HTML file, now download the example CSS file and refresh the page. This is what you can do with a few CSS definition. Now go the the design gallery each of these examples uses the came HTML file the only changes here are to the style definitions.

Selectors correspond to HTML elements, there attributes, decedents and and siblings.

Common Selectors

These are the most common selectors. While there are many more

- Class Selector `.danger` - *(.) + (CLASS-NAME)*
- Tag Selector `p` - *(TAG-NAME)*
- ID Selector `#mastHead` - *(#) - (ID-NAME)*

```
/* All paragraph elements have a blue font */
p{
  color: blue;
}

/* All paragraph elements with a class attribute of danger will have a red font */
p.danger{
  color: red;
}

/* All div elements with any child element that has with a class attribute of
danger will have a red font */
div > *.danger{
  color: red;
}

/* This will also work as described above */
div *.danger{
  color: red;
}

/* So will this */
div .danger{
  color: red;
}

/* You can apply a single description to multiple selectors with comma separation */
p,
.info,
div > *.info{
  color: blue;
}
```

Specificity Rules

The following list from the weakest to stringest selectors when it comes to implementing css rules. Inline style trump all, while `!important` can force an override of a higher lower level selector; this should be used sparingly.

- universal selector (*)
- element selector (h1, p, a)
- class selector (.blue, .link, .content)
- attribute selector ([href], [target], [src])
- psuedo selector (:link, :hover, :visited)
- id selector (#firstName, #lastName, #phoneNumber)
- inline style (style="text-align: center;)

```
/* Make all font for all elements red */
* {
    color: red;
}

/* overrides * */
p, a {
    color: blue;
}

/* overrides p  */
p.green {
    color: green;
}

/* overrides *, a, p */
.indigo {
    color: indigo;
}

/* overrides a */
a[href="/"] {
    color: aquamarine;
}

/* overrides a */
a:link,
a:visited{
    color: maroon;
}

/* overrides a:link, a:visited */
a.indigo:link,
a.indigo:visited,{
    color: purple;
}

/* overrides everything */
#home{
    color: #efefef;
}
```

Add a Stylesheet

</> code In your GitHub Pages project create the directory path *dist/css* and add the file *main.css*. To the document head in index.html use the `link` element to reference the stylesheet.

```
<link rel="stylesheet" type="text/css" href="./dist/css/main.css">
```

You can verify the file loads by checking networking tab in dev tools.

Remove the Inline Styles

</> code We will start by removing the inlie style we added to our index.html file. We will start by adding an avatar class to our stylesheet.

dist/css/main.css

```
.avatar {
  border-radius: 50%;
  float: left;
  margin-right: 1em;
}
```

Reset the body element

Now we can start buildinga layout, we will start by reseting the body elements, adding a base font and then we will move on to navigation.

It's common practice to remove the defualt padding and margin from the body, add the following to main.css.

```
body{
  padding: 0;
  margin: 0;
}
```

Set a base font

Next we will add a base font. To assure consitancey across all platforms we will pull a font from a font foundary, we will use Google Fonts and use Open Sans. We will import the font style into our style sheet and use a universal selector to set all elements to this font by default.

```
@import url('https://fonts.googleapis.com/css?family=Open+Sans');

* {
  font-family: 'Open Sans', sans-serif;
}
```

Header and Navigation

No we are ready to tackle navigation, lets start by wrapping the `nav` elements in a `header` element. We will add a `span` element with a class of `.logo` outside of the `nav` with the navigation. We will align all navigation links horizontally and pull them to the top right corner of the screen.

```
<header>
  <span class="logo">My WebSite</span>
  <nav>
    <ul>
      <li><a href="index.html">Home</a></li>
      <li><a href="resume.html">Resume</a></li>
      <li><a href="contact.html">Contact</a></li>
    </ul>
  </nav>
</header>
```

```
header {
  height: 50px;
  background: #000;
  color: #fff;
  padding: 0 .5rm 0 1em;
}

header .logo{
  line-height: 50px;
  font-weight: bold;
}

nav {
  float: right;
}

nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}

nav ul li{
  display: inline;
}

nav ul li a,
nav ul li a:link,
nav ul li a:visited{
  display: inline-block;
  padding: 0 .5em;
  color: #fff;
  line-height: 50px;
  text-decoration: none;
}

nav ul li a:hover{
  background: #444;
}
```

Main Content (set a landmark)

Add a `main` element and center the page content.

```
<main>
  <h1>Hello,...
</main>
```

```
main {
  /* Use a margin to center the page */
  margin: 0 auto;

  /* Set the width of the content */
  width: 960px;

  /* For samller screens to not exceed the max screen width */
  max-width: 100%;
}
```

Make it Responsive

Use dev tools to toggle the device toolbar (enter mobile testing mode) and choose the responsive option. Set the width to 960px and drag in and out crossing back and fourth over that 960px mark. Notice the padding, we loose all of padding under 960px meaning the text runs to the edge of the screen. Popular opinion is that makes for a better read if you can get some seperation from the edge of the screen. You could use a media query and change the page width on a break point sush as below.

Add Gutters to Small Screens

```
@media only screen and (max-width: 960px)  {
    main {
        max-width: 90%;
    }
}
```

</> code Or you could set the default `max-width` to **90%** which is what we will do here.

```
  /* For samller screens to not exceed the max screen width */
  max-width: 90%;
```

Switch to Veritcal Navigation on Small Devices

</> code It's common practice, on smaller screens, to change from a horizontal navigation in the header to a hidden vertical navigation that appears on a button press. We can do this by setting the nav display to none.

```
@media only screen and (max-width: 960px)  {
    nav {
        nav {
            display:none;
        }
    }
}
```

</> code Add an anchor element to the `header` element right before the `nav` element. This will act as the control to toggle the menu on smaller screens. Give this element and `id` attribute with a value value of *toggleMenu*.

```
<a id="toggleMenu">Menu<a>
```

Create a CSS selector for the toggleMenu attribute. This style will be similar to but set breakpoints opposite to that of `nav` meaning this will only appear on smal

```

#toggleMenu {
  display: none;
}

@media only screen and (max-width: 960px)  {

  #toggleMenu {
    display: block;
    float: right;
    line-height: 50px;
  }
  ...
```

</> code We will cover JavaScript in great detail later, for now copy and paste the following into index.html right before the closing body tag.

```
    <script>

      var toggleMenu = document.getElementById('toggleMenu');
      var nav = document.querySelector('nav');
      toggleMenu.addEventListener(
        'click',
        function(){
          if(nav.style.display=='block'){
            nav.style.display='none';
          }else{
            nav.style.display='block';
          }
        }
      );
    </script>
```

</> code Now if you reload the page, mave the width to less than 960px and press menu, you notice not much has changed. We will fix this by setting the nav to a block level component.

```
@media only screen and (max-width: 960px)  {

  #toggleMenu {
    display: block;
    float: right;
    line-height: 50px;
  }

  nav {
      display: none;
      background: #000;
  }

  nav ul li a,
  nav ul li a:link,
  nav ul li a:visited{
    display: block;
    border-bottom: 1px solid #777;
  }

}
```

Reload the page and make sure the resolution in under 960px, press the Menu button. It's now more closley resembles a traditional flyou menu. Make the resolution small and you'll notice the position of the menu changes. To fix this you will absolutly position `nav` relative to `header` .

Start by making assigning the relative position to the `header` element. Then assign an absolut position to the `nav` element. When an absolute element is the child a relative element the positioning of the absolute element is relative to that of it's parent.

```
@media only screen and (max-width: 960px)  {

  header {
    position: relative;
  }

  nav {
    position: absolute;
    display: none;
    background: #000;
  }
  ...
```

</> code Next you will want to position the menu. By default the menu will want to position itself in the top left corner. If you were to think of a box as having four boundaries: top, left, right and bottom; then `top` , `left` , `right` , `bottom` are commandes that move the top left corner of a target element a set distance from the specified boundaries. In this case we will set top to `50px` this will allow it to clear the 50px height of the header element. By setting `left` and `right` to `0` you will stretch nav element to the width of the entire screen.

```
@media only screen and (max-width: 960px)  {
```

```
header {
  position: relative;
}

nav {
  position: absolute;
  display: none;
  background: #000;
  top: 50px;
  left: 0;
  right: 0;
}
...
```

Exercise - Restyle All Pages

Apply the new navigation to all pages.

Additional Resources

- Specificity
- CSS Tricks
- MDN - CSS
- Why do navigation bars in HTML5 as lists?
- Nav Element
- Bad Design vs. Good Design

Udemy

- Flexbox Tutorial

Next: SASS

SASS

SASS is a a CSS preprocessor is a superset of CSS which means all CSS syntax is considered valid *.scss* it's a superset because because it extends CSS with programming like capabilities; variables and limited control statements. This is advantageous especially when building large front-end frameworks such Bootstrap or creating a product with a customizable theme. For example, Bootstrap has a deafult shade of red *#dc3545* [1]. If we were to change that color globally we would have to track down every instance of the color and change it manually or we could make a single change to the variable that holds that color.

Install Sass

Install ruby-sass via apt.

```
sudo apt-get install ruby-sass
```

Variables in Sass. Sass denotes variables with a *$* dollar sign. For these lessons we will use the newer SCSS syntax for writing our sass files. These files must have the *.scss* extensions.

Exercise 1 - Sass Variables

Create the paths */var/www/mtbc/scss/index.html* and */var/www/mtbc/scss/main.scss* and add the following to *index.html*,

```
<!DOCTYPE html>
<html>
    <head>
       <title>SASS DEMO</title>
       <link href="./main.css" type="text/css" rel="stylesheet">
    </head>
    <body>
        <h1>SASS DEMO</h1>

        <h2>FOUR COLUMN GRID</h2>
        <div class="row">
          <div class="col">ONE</div>
          <div class="col">TWO</div>
          <div class="col">THREE</div>
          <div class="col">FOUR</div>
        </div>
        <div class="row">
          <div class="col">FIVE</div>
          <div class="col">SIX</div>
          <div class="col">SEVEN</div>
          <div class="col">EIGHT</div>
        </div>

        <h2>TEXT CLASSES</h2>
        <div class="text-success">Success Text</div>
        <div class="text-error">Error Text</div>
        <div class="text-warning">Warning Text</div>

        <h2>MESSAGE CLASSES</h2>
        <div class="message">Default Message</div>
        <div class="message-success">Success Message</div>
        <div class="message-error">Error Message</div>
        <div class="message-warning">Warning Message</div>

    </body>
</html>
```

Load the page https://localhost/mtbc/sass. Then add the following the to *main.scss*

```
/* variables */

$primary-font-stack: "Helvetica Neue",Helvetica,Arial,sans-serif;
$primary-color: #333;

/* universal settings */

* {
  box-sizing: border-box;
}

body {
  font: 100% $primary-font-stack;
  color: $primary-color;

  margin: 0 auto;
  padding: 0;
  width: 1170px;
}
```

Compilation is the next step. From the command line we will compile sass into css.

```
sass /var/www/mtbc/scss/main.scss /var/www/mtbc/css/main.css
```

This will create the file main.css which is a compiled version of your scss file.

```
* {
  box-sizing: border-box;
}

body {
  font: 100% "Helvetica Neue",Helvetica,Arial,sans-serif;
  color: #333;

  margin: 0 auto;
  padding: 0;
  width: 1170px;
}
```

Exercise 2 - Live Reload / Watch a File

The downside to a preprocessor is the compilation step. This takes time and slows down development. We remedy this by creating a *watcher* this watches a target file for changes and rebuilds it's CSS version in the background. This is one less thing you need to think about which can help keep you in flow. Open a split console window and run the following command in one of the panels.

```
sass --watch /var/www/mtbc/scss/main.scss:/var/www/mtbc/css/main.css
```

You will see the following output

```
>>> Sass is watching for changes. Press Ctrl-C to stop.
  directory ~/scss
      write ~/scss/main.css
      write ~/scss/main.css.map
```

In the second panel open the scss file in vim, make a change and save it using [esc] then `:x` ; You'll notice a change in the first console window with the following output.

```
>>> Change detected to: main.scss
      write ~/scss/main.css
      write ~/scss/main.css.map
```

Open the file */var/www/mtbc/scss/main.css* and verify your changes.

Mixins

Later we will learn about the Bootstrap framework. Bootstrap is among the most popular frameworks and as such it gets a lot of criticism. One of the those criticisms is the practice of calling mulitple class on a single element. The claim is that this can increase load time. For example styling a button in Bootstrap often looks as follows `class="btn btn-default btn-xs"`. The idea is the it would be faster to combine all of those classes into a single definition in which case something like `class="btn-default-xs"` would load faster. In native CSS this would mean having a lot of duplicate code. SASS allows us to reuse style deifintions, meaning we can write them once a call them as many times as we need to. This means we can globally chane an entire style sheet by changing only one line of code. Rather that calling multiple classes we can define mixins and extend base classes making all of our code reusable..

Clearfix

I have always viewed `.clearfix` as a containment element for any number of floats. Meaning if you apply clearfix to a parent element any child element, that is has a float property, cannot escape the parent element. `.clearfix` is a common hack used by front end developers to solve the problem of using floats in a way they were never indtended to be used[2].

- CodePan Floating grid without a clearfix
- CodePan Floating grid with clearfix

Exercise 3

Create a `.clearfix` mixin by adding the following to the top of *main.scss*.

```
/* mixins */
/* clear floats */
@mixin clearfix() {
  &:after {
    content: "";
    display: table;
    clear: both;
  }
}
```

Then you can create two classes applying `clearfix()` to both classes. We will go ahead and add a column class. We will discuss columns in greater detail in later lessons. For now we will use to demo other concepts.

```
/* Utility Classes */

.clearfix {
  @include clearfix();
}

/** Rows and Columns */
.row {
  @include clearfix();
}

.col {
  float: left;
  width: 25%;
  padding: .5rem;

  /* debug */
  border: 1px solid #990000;
  background: #fff;
}
```

Extend/Inheritance

Other examples of calling multiple classes is in the footer navigation as well as the #Content and #Sidebar divs.

```
<ul class="nav-inline pull-right" role="navigation">
```

Another method of reuse in SASS is *@extend* so `.sample{@extend .example;}` would apply the *.example*'s style declaration to *.sample*.

Exercise 4 - Add Response Classes (Quick Lab 15 minutes)

Add the following classes to main.scss update the style declarations so that redundant values are called as variables.

```
.text-success {
  color: #3c763d;
}

.text-error {
  color: #8a6d3b;
}

.text-warning {
  color: #a94442;
}

.message {
  border: 1px solid #ccc;
  border-radius: 4px;
  padding: 10px;
  color: #333;
}

.success {
  @extend .message;
  border-color: #3c763d;
  color: #3c763d;
}

.error {
  @extend .message;
  border-color: #8a6d3b;
  color: #8a6d3b;
}

.warning {
  @extend .message;
  border-color: #a94442;
  color: #a94442;
}
```

Exercise 5 - Implement sass in your project

Move *dist/css/main.css* to *src/scss/main.scss*

```
cd /var/www/YOUR-GITHUB-USERNAME.github.io
mkdir -p src/scss
mv dist/css/main.css src/scss/main.scss
```

Then compile the sass file

```
sass src/scss/main.scss dist/css/main.css
```

Additional Resources

- SASS Reference
- MDN CSS Clear

References

1. bootstrap/scss/_variables.scss
2. What Does the Clearfix Class Do in Css?

Next: Gulp

Gulp

Gulp allows you to build workflows for optimizing your frontend assets. As with SASS you can create a wathcer to auto build on save. GUlp offers additional feature such combining and minifing files. Since Gulp is a JavaScript package and the config file is written in JavaScript you can add any functionality you want.

Globally install Gulp

```
cd ~
sudo npm install -g gulp
```

Confirue NPM for you local project

Add a file named package.json to you GitHub Pages project. This must be added to the projects top level directory. Gulp is built in NodeJS and lives in the NPM ecosystem. You run a series of NPM commands to initialize you project as an NPM project and intall each dependency manually or you can use a prebuilt config file. The latter is the path we will take for this project. Add the following to you package.json file.

```
{
  "name": "YOUR-GITHUB-USERNAME.github.io",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "gulp watch"
  },
  "dependencies": {},
  "devDependencies": {
    "gulp": "^3.9.1",
    "gulp-clean-css": "^3.9.2",
    "gulp-concat": "^2.6.1",
    "gulp-rename": "^1.2.2",
    "gulp-scss": "^1.4.0",
    "gulp-uglify": "^3.0.0",
    "gulp-watch": "^5.0.0",
    "merge-stream": "^1.0.1"
  }
}
```

Install all Packages

By virtue of having a package.json file your project is an NPM project. At this point we have deinfed the project dependencies; now we need to install them. Run the following from the command line.

```
cd /var/www/YOUR-GITHUB-USERNAME/github.io
npm install
```

.gitignore

Git will stage every file it sees. There are cases in which you project requres files that you will never want to stage and commit. You can try to track these manually but that will inevitably fail. Add a file called .gitignore to your project and these files will not available for staging. Create a file called .gitignore in the top level of your project and add the following.

```
node_modules
.gulp-scss-cache
```

```
.sass-cache
```

gulpfile.js

Gulp is an ES6 (JavaScript) script designed for frontend compilations. These are typically written as small, single script programs.

```
var gulp = require('gulp');
var watch = require('gulp-watch');
var cleanCSS = require('gulp-clean-css');
var uglify = require('gulp-uglify');
var rename = require('gulp-rename');
var concat = require('gulp-concat');
var merge = require('merge-stream');
var scss = require('gulp-scss');

gulp.task('default', ['watch']);

gulp.task('build-css', function(){
  //Create an unminified version
  var full = gulp.src([
    'src/scss/main.scss'
  ])
  . pipe(scss())
  . pipe(concat('main.css'))
  . pipe(gulp.dest('dist/css'));

  //Create a minified version
  var min = gulp.src([
    'src/scss/main.scss'
  ])
  . pipe(scss())
  . pipe(cleanCSS())
  . pipe(concat('main.min.css'))
  . pipe(gulp.dest('dist/css'));

  return merge(full, min);
});

gulp.task('watch', function(){
  gulp.watch('./public/src/scss/**/*.scss', ['build-css']);
});
```

Run any of the following commands execute your Gulp script.

```
gulp
gulp watch
gulp build-css
```

Since we defined `gulp watch` as our NPM start up script you can use `npm start` execute the watcher.

```
npm start
```

Next: CSS Layouts

CSS Layouts

Three ways to do one layout.

Floats

- CodePen Base Layout Example

Additional Floating Grid Examples

- CSS Floating Grid with Clearfix

Flexbox

- CodePen Base Layout Example

Additional Flexbox Examples

- Flexbox with fixed and scrolling columns

CSS Grid

- CodePen Base Layout Example

Additional CSS Grid Examples

- CSS Responsive 3 to 2/1 Column Grid
- CSS Responsive 6 Column Grid

Lab - Add Gulp to Your GitHub Pages site

- Add Gulp to your GitHub pages project
- Your GitHub Pages project must now contain an src and a dist directory
- All new CSS must be written to src and compiled into dist

Using all the techniques we have elarned thus for style your resume so that it looks like the template.

Additional Reading

- Modern CSS Explained For Dinosaurs

Next: JavaScript

JavaScript

Chapter 7: JavaScript

Next: JavaScript Basics

JavaScript Basics

JavaScript is simple programming language created by Netscape's Brendan Eich in 1995. The goal was to make web pages more interactive by allowing the browser to execute native code without the need of a plugin. JavaScript was standardized by ECMA thus providing a standard implementation reference. While we still call it JavaScript the reference standard is ECMA-262

Exercises

Modern browsers make use of DevTools (aka Web Console). While Chrome, FireFox and Edge have similar definitions for their DevTools, I'll steal this line from Chrome's documentation "Use the DevTools to iterate, debug, and profile your site".

- Chrome
- FireFox
- Edge

Create the file `/var/www/mtbc/js/console.html` that contains the following markup.

```
<!DOCTYPE html>
<html>
  <head>
      <title>Using the development console.</title>
  </head>
  <body>
    <p>Using the development console. Press [f12] and click the console tab.</p>
        <script>
      console.log();
        </script>
  </body>
</html>
```

Exercise 1 - Print data to the console

Exercise 1.1 - Pass a static string as an argument

Find the `<script>` tags in the markup you just added to the new file. You will see a single line of code that reads `console.log()` . Modify this line to read `console.log('Hello World');` . Then open a browser and navigate to [http://localhost/mtbc/js/console.html(http://localhost/mtbc/js/console.html). Once the page opens, press [f12] and click on the console tab. You will see the text *Hello World* followed by the file name and line number.

Notes

The command `console.log()` call the log() function from the console API. This is what is know as dot concatenation, in this case `log()` is a property (method, function or variable) of the `console` API. This syntax is also used in object oriented programming (OOP) in which it would read something like `ObjectX.propertyY` or `ObjectX.methodZ()` . Some languages such a PHP use `->` instead of a dot, but the idea is the same `$ObjectX->propertyY` or `$ObjectX->methodZ()` .

While not required, most function (or method) calls will accept an argument. In this case, the console APIs `log()` method accepts a single parameter, the data to be printed to the screen. This data may be of any type number, string, array, object, etc. In exercise 1.1 we passed a static string. You can read more about JavaScript data types here.

Exercise 1.2 - Pass a variable as an argument

Modify the contents of the script tags to read as follows.

```
var msg = 'Hello World';
console.log(msg);
```

In this exercise we are creating the variable msg (which is a common variable name for message) and assigning it a value of *Hello World*.

Exercise 1.3 - Working with strings

Modify the contents of the script tags to read as follows.

```
var greeting =  'Hello';
var who = 'World';
var msg = greeting + ' ' + who;

console.log(msg);
```

In this example we are combining the value of two variables and assigning the result to the `msg` variable.

Exercise 1.4 - Working with numbers

Modify the contents of the script tags to read as follows.

```
var num1 =  35;
var num2 = 7;
var result = num1 + num2;

console.log(result);
```

In this example we are adding the values of `num1` and `num2` .

Notes

Notice we are not adding quotes to the value of these variables. That is because we want to treat these values as numbers not strings. This will only work with numbers, if we were to set the value of `num1` to *ab* instead of *'ab'* we would get an error, this is because *ab* is not a number. If on the other hand, we had a variable named `ab` and that variable had a numeric value, it would treat `ab` as a numeric data type.

Exercise 1.4 - Working with mixed data types

Modify the contents of the script tags to read as follows.

```
var num1 =  '35';
var num2 = 7;
var result = num1 + num2;

console.log(result);
```

As in the previous example we are adding *7* to *35*, only this time we have added quotes to *35*. Adding quotes changes the data type from a number to a string so rather than adding two numeric values we are concatenating two strings.

Notes

JavaScript is loosely typed, meaning it will try to resolve the data type in to something that makes since to a given problem. Since, in this case we are adding a number to a string the JavaScript interpreter will conclude (since a number can always fit into a string, but a string cannot always fit into a number) that we are attempting to concatenate two strings rather than add two numbers which will result in a value of *357*.

Exercise 2 - Write to the page

Duplicate `/var/www/mtbc/js/console.html` to `/var/www/mtbc/js/write.html` that contains the following markup and add an element of type `div` to the body of the HTML document. Give the `div` an attribute of type `id` with a value of *MyText* `<div id="MyText"></div>` .

Exercise 2.1 - Working with mixed data types

Replace the contents of the script tags with the following.

```
var msg = 'Hello World';

document.getElementById('MyText').innerHTML = msg;
```

Checkyour changes at [http://localhost/mtbc/js/write.html(http://localhost/mtbc/js/write.html)

In this exercise we are calling the `getElementById()` method of the `document` API. This accepts a single attribute and that is the id of the element we want to access. `innerHTML` is a property of the element. This property represents all of the content that appears between the opening and closing tags of the target element. In the exercise we are setting the value of `innerHTML` .

Exercise 2.1 - Working with mixed data types

In the js.html file change `<div id="MyText"></div>` to `<div id="MyText">Hello World</div>` and add a new element `<div id="NewText"></div>`

Inside the script tags, write some JavaScript that will read the content of *MyText* and write it to *NewText*.

Notes

The `innerHTML` property hold the content of a target element.

Additional Resources

- MDN - JavaScript
- Eloquent_JavaScript
- Douglas Crawford

Next: JavaScript Control Structures

JavaScript Control Structures

Programming is little more that reading data and piecing together statements that take action on that data. Every language will have it's own set of control structures. For most languages a given set of control structures will be almost identical. In the Bash and PHP lessons we learned a few control structures most of which exist in JavaScript. While the syntax may be a little different, the logic remains the same.

Exercise - If, Else If, Else

Create the following path */var/www/mtbc/js/if_else.html* and open it Atom. Then add the following lines.

```
<script>

//Initialize your variables
var label = null;
var color = null;
var GET = {};

//parse the GET params out of the URL
if(document.location.toString().indexOf('?') !== -1) {
    var query = document.location
       .toString()
       // get the query string
       .replace(/^.*?\?/, '')
       // and remove any existing hash string (thanks, @vrijdenker)
       .replace(/#.*$/, '')
       .split('&');

    for(var i=0, l=query.length; i<l; i++) {
       var aux = decodeURIComponent(query[i]).split('=');
       GET[aux[0]] = aux[1];
    }
}

//Check for get parameters
if(GET['color'] !== 'undefined'){
    color = `#${GET['color']}`;
}

//Can we name the color by it's hex value
if(color == "#ff0000") {
  label = "red";
} else if(color == "#00ff00") {
  label = "green";
} else if(color == "#0000ff") {
  label = "blue";
} else {
  label = "unknown";
}

//Output the dataa
document.write(`<div style="color:${color}">The color is ${label}</div>`);

</script>
```

Now open a browser and navigate to https://localhost/js/if_else.html and you will see the message *The color is unknown*. Now add the following string to the end of the URL *?color=ff0000*. Now your message will read *The color is red* and it will be written in red font. That string you added to the end of the URL is know as a query string. A query string allows you to pass arguments into a URL. A query string consists of the query string Identifier (a question mark) *?* and a series of key to value pairs that are separated by an ampersand (&). In our example the the *key is color* color *and the _value is ff0000*. If you wanted to submit a query of a first and last name that might look like *?first=bob&last=smith* where first and last are your keys (aka your GET params) bob and smith are your values.

Now let's take a close look at the code. Initializing your variables is a good practice.

```
//Initialize your variables
var label = null;
var color = null;
```

JavaScript does not have a `$_GET` super global like PHP so we will build one by parsing out the URL

```
//parse the GET params out of the URL
if(document.location.toString().indexOf('?') !== -1) {
    var query = document.location
       .toString()
       // get the query string
       .replace(/^.*?\?/, '')
       // and remove any existing hash string (thanks, @vrijdenker)
       .replace(/#.*$/, '')
       .split('&');

    for(var i=0, l=query.length; i<l; i++) {
       var aux = decodeURIComponent(query[i]).split('=');
       GET[aux[0]] = aux[1];
    }
}
```

If GET['color'] is defined then the set the variable color to the value of GET['color']

```
//Check for get parameters
if(!empty($_GET['color'])){ //This is a control statement
    //This is the body of the statement
    color = `#${GET['color']}`; //ES^ String literal
}
```

The user has submitted a hex value in the form of a get parameter. Do we know what to the call that hex value? If the answer is yes set the value of *label* to that color. Otherwise set the value of *label* to *Unknown*. Or you could say *if the hex value is red then say it is red; otherwise if it green then say it is green; otherwise if it blue then say it is blue; otherwise say unknown*.

```
//Can we name the color by it's hex value
if(color == "#ff0000") {
  label = "red";
} else if(color == "#00ff00") {
  label = "green";
} else if(color == "#0000ff") {
  label = "blue";
} else {
  label = "unknown";
}
```

Finally we will print some output back to the screen. This time we will wrap the output in some HTML and give it a little style by setting the font color to that of the user input.

```
document.write(`<div style="color:${color}">The color is ${label}</div>`);
```

Exercise - For Loop

Add the following to the path */var/www/mtbc/js/for.html*.

```
<script>
var items = [
  'for',
  'do...while',
  'for...in',
  'for...of'
];

var msg = 'JavaScript supports ' + items.length + ' types of loop.';

var li = '';
for(i=0; i<items.length; i++){
  li += `<li>${items[i]}</li>`;
}

msg += `<ul>${li}</ul>`;

document.write(msg);
</script>
```

Exercise - For...in Loop

Add the following to the path */var/www/mtbc/js/forin.html*.

```
<script>
var items = {
  0:'for',
  1:'do...while',
  2:'for...in',
  3:'for...of'
};

var msg = 'JavaScript supports ' + items.length + ' types of loop.';

var li = '';
for(var item in items){
  li += `<li>${items[item]}</li>`;
}

msg += `<ul>${li}</ul>`;

document.write(msg);
</script>
```

Exercise - For...of Loop

Add the following to the path */var/www/mtbc/js/forof.html*.

```
<script>
var items = [
  'for',
  'do...while',
  'for...in',
  'for...of'
];

var msg = 'JavaScript supports ' + items.length + ' types of loop.';

var li = '';
```

```
for(var item of items){
  li += `<li>${item}</li>`;
}

msg += `<ul>${li}</ul>`;

document.write(msg);
</script>
```

Exercise - While Loop

```
<script>
var items = [
  'for',
  'do...while',
  'for...in',
  'for...of'
];

var msg = 'JavaScript supports ' + items.length + ' types of loop.';
var i=0;
var li = '';
while(i < items.length){
  li += `<li>${items[i]}</li>`;
  i++;
}

msg += `<ul>${li}</ul>`;

document.write(msg);
</script>
```

Exercise - Do...While

Add the following to the path */var/www/mtbc/js/do_while.php*.

```
<script>
var items = [
  'for',
  'do...while statement',
  'labeled statement',
  'break statement',
  'continue statement',
  'for...in statement',
  'for...of statement'
];

var msg = 'JavaScript supports ' + items.length + ' types of loop.';
var i=0;
var li = '';
do {
  li += `<li>${items[i]}</li>`;
  i++;
} while (i < items.length)

msg += `<ul>${li}</ul>`;

document.write(msg);
</script>
```

Exercise - Switch Statement

```
<script>

var color = null;
var GET = {};

//parse the GET params out of the URL
if(document.location.toString().indexOf('?') !== -1) {
    var query = document.location
       .toString()
       // get the query string
       .replace(/^.*?\?/, '')
       // and remove any existing hash string (thanks, @vrijdenker)
       .replace(/#.*$/, '')
       .split('&');

    for(var i=0, l=query.length; i<l; i++) {
       var aux = decodeURIComponent(query[i]).split('=');
       GET[aux[0]] = aux[1];
    }
}

//Check for get parameters
if(GET['color'] !== 'undefined'){
    color = `#${GET['color']}`;
}

switch (color) {
  case 'ff9900':
    console.log('The color is red');
    break;
  case '00ff00':
    console.log('The color is green');
    break;
  case '0000ff':
    console.log('The color is blue');
    break;
  default:
    console.log('Sorry, I cannot determine the color');
    break;
}

</script>
```

Additional Resources

- Control Flow and Error Handling
- Loops and iteration
- Switch

Next: WalkingTheDom

Walking the DOM

The Document Object Model (DOM) is an API that treats markup languages (xml, xhtml, html, ect) as a tree structures. A easier way to think of this might be as an interface that allows a programmer to access tags and the attributes of tags. Later we will learn about jQuery; a library for querying the DOM (among other things). First we will learn basic manipulation using straight JavaScript.

In the previous lesson we used `document.getElementById();` this method queries the DOM for an element with a matching id. There are many similar methods.

Collection Live vs Static (not live)

> A collection is an object that represents a lists of DOM nodes. A collection can be either live or static. Unless otherwise stated, a collection must be live.[1]

> If a collection is live, then the attributes and methods on that object must operate on the actual underlying data, not a snapshot of the data.[1]

> When a collection is created, a filter and a root are associated with it.[1]

> The collection then represents a view of the subtree rooted at the collection's root, containing only nodes that match the given filter. The view is linear. In the absence of specific requirements to the contrary, the nodes within the collection must be sorted in tree order. [1]

By ID

document.getElementById(id_string)

Return a element object.

Create the path *mtbc/js/dom/by_id.html*, then open the source file in your editor. Add the following lines to the script tags then refresh the page and note the changes.

```
var elem = document.getElementById("a"); // Get the elements
elem.style = 'color: #FF0000;'; // change color to red
```

By Tag

document.getElementsByTagName(tag_name)

Return a live HTMLCollection (an array of matching elements).

The tag_name is "div", "span", "p", etc. Navigate to *mtbc/js/dom/by_tag_name.html*, then open the source file in your editor. Add the following lines to the script tags then refresh the page and note the changes.

```
var list = document.getElementsByTagName('blockquote'); // get all p elements
list[0].style = 'color: #FF0000;';
```

By Class

document.getElementsByClassName("class_values")

Return a live HTMLCollection.

The class_values can be multiple classes separated by space. For example: "a b" and it'll get elements, where each element is in both class “a” and “b”. Navigate to *mtbc/js/dom/by_class_name.html*, then open the source file in your editor. Add the following lines to the script tags then refresh the page and note the changes.

```
// get all elements of class a
var list = document.getElementsByClassName('b');

// Make it red
list[0].style = 'color: #FF0000;';

// get all elements of class a b
var list2 = document.getElementsByClassName('a b');

//Make them bold and apply the color from list[0]
list2[0].style = 'font-weight: bold; color:' + list[0].style.color;
```

By Name

document.getElementsByName("name_value") Return a live HTMLCollection, of all elements that have the name="name_value" attribute and value pair.

Navigate to *mtbc/js/dom/dom/by_name.html*, then open the source file in your editor. Add the following lines to the script tags then refresh the page and note the changes.

```
//Get all elements with a name of a
var list = document.getElementsByName('a');

//Loop through all of the matching elements
for (var i=0; i<list.length; i++) {
  //Make them red
  list[i].style = 'color: #FF0000;';
}
```

By CSS Selector

document.querySelector(css_selector)

Return a non-live HTMLCollection, of the first element that match the CSS selector css_selector. The css_selector is a string of CSS syntax, and can be several selectors separated by comma.

Navigate to *mtbc/js/dom/by_css_selector.html*, then open the source file in your editor. Add the following lines to the script tags then refresh the page and note the changes.

```
//Find the element with an id of bold
var bold = document.querySelector('#bold');

//Make it bold
bold.style = 'font-weight: bold;';

//Find all elements with a class of blue
var blues = document.querySelectorAll('.blue');

//Loop through all of the matching elements
for (var i=0; i<blues.length; i++) {
  //Make them blue
```

```
    blues[i].style = 'color: #0000FF;';
  }
```

Additional Resources

- [1]Collections

Udemy

Javascript Intermediate level 1 - Mastering the DOM

Next: Events

Events

Events allow a user to interact with a web page. Listeners are used to detect an event being triggered. One might say the web page listens for a click event.

Click Event

In your browser navigate to *mtbc/js/events/onclick.html* and open to corresponding file in your editor. Add the following to the body tag of the document and refresh the page.

Exercise 1 - Catch the Click

```
<div id="demo"></div>
<button onclick="document.getElementById('demo').innerHTML = Date()">What time is it?</button>
```

The tag *div#demo* loads with no content. The button's on click attribute is an event listener for a mouse click. The value of onclick is the code that will be executed when the click event is triggered.

Exercise 2 - Refactor the Code

You can execute JavaScript directly in the HTML tag but this is considered bad practice. Let's Add some script tags and function that accepts the target element as an argument.

```
<button onclick="showDate('demo');">What time is it?</button>
<script>
  function showDate(element) {
     document.getElementById(element).innerHTML = Date();
  }
</script>
```

This is looking pretty good but some circles even argue against using the HTML elements onclick attribute. Let's write a listner in JavaScript.

```
<button id="time">What time is it?</button>
<script>
  function showDate(element) {
     document.getElementById(element).innerHTML = Date();
  }

  // Function to add event listener to button#time
  var btn = document.getElementById("time");
  btn.addEventListener("click", function(){showDate('demo')}, false);
</script>
```

Now let's change this to an arrow function. Arrow functions use shorter syntax and do not bind *this*.

```
<button id="time">What time is it?</button>
<script>
  function showDate(element) {
     document.getElementById(element).innerHTML = Date();
  }

  // Function to add event listener to button#time
  var btn = document.getElementById("time");
  btn.addEventListener("click", ()=>{showDate('demo')}, false);
</script>
```

Change Event

A change event listens for a change to a target element. Change events are typically used for picklists.

Exercise 3 - Catch a Change

In your browser navigate to *mtbc/js/events/events/onchange.html* and open to corresponding file in your editor. Add the following to the body tag of the document and refresh the page.

Add the following to the form tag. Be sure to start the list with an empty option.

```
<select id="options">
  <option>--Pick an Option--</option>
  <option value="a">A</option>
  <option value="b">B</option>
  <option value="c">C</option>
</select>
```

Add the following to the script tag.

```
function setValue(element) {
   let option = document.getElementById('options').value;
   document.getElementById(element).innerHTML = option;
}

// Function to add event listener to button#time
var list = document.getElementById("options");
list.addEventListener("change", ()=>{setValue('demo')}, false);
```

Intervals

While an interval is not techincally and event, it does provide a trigger for an interaction to occur. In this case we will use it with a click event to grow another form element on the screen.

In your browser navigate to *mtbc/js/events/interval.html* and open to corresponding file in your editor. Add the following to the script tags of the document and refresh the page. Be sure to read the comments and understand the reasoning behind each lie of code.

```
function Grower() {
  //Get current width of the target element
  this.width  = document.getElementById('demo').offsetWidth;

  //Set a counter to the current width
  this.num = this.width

  //Execute the counter on a set interval
  this.timer = setInterval(() => {
    //Increment the counter
    this.num++;

    //Add one pixil to the width of the target element
    document.getElementById('demo').style = 'width:' + this.num + 'px;';

    //After fifty iterations, kill the loop.
    if(this.num === this.width + 50){
      clearInterval(this.timer);
    }
  }, 1);
}

// Function to add event listener to button#time
var btn = document.getElementById("GrowBtn");
btn.addEventListener("click", function(){new Grower()}, false);
```

Additional Resources

Arrow Functions for Beginners Arrow Functions Mastering the Module Pattern

Next: Canvas

Canvas

The `canvas` element was introduced in HTML5, it can be used to draw graphics via JavaScript. This can be used for many purposes including games, visualizations, charts, graphs, etc.

Create a GitHub project called draw.

- Use the HTML Starter pacakge from GitHub.
- Add your new GItHub project as the origin remote.
- Add a canvas tag to the body.
- Change the README file.

  ```
  <canvas id="canvas" height="600px" width="800px"></canvas>
  ```

Then you will need to write some JavaScript, for the sake of brevity we will add this directly to the HTML using `script` tags.

Start by defining a canvas, this tells JavaScript which element to use as the canvas. This MUST always be a `canvas` element.

```
<script>
  var canvas = document.getElementById('canvas');
</script>
```

Then add a context, we will start with the 2D context. A canvas may have only one context so everything you draw gets merged into a single bitmap. Once drawn it is a ll a single image. When the canvas was created it was stored in the object we named `canvas` to apply a fill color access the `getContext()` method of the `canvas` object.

```
<script>
  var canvas = document.getElementById('canvas');
  var ctx = canvas.getContext('2d');
</script>
```

We will start by drawing a rectangle, the first thing you will want to do is choose a fill color for your rectangle. Let's choose this at random. When the context was created it was stored in the object we named `ctx` to apply a fill color access the `fillStyle` property of the `ctx` object.

```
<script>
  var canvas = document.getElementById('canvas');
  var ctx = canvas.getContext('2d');
  ctx.fillStyle = '#' + Math.floor(Math.random()*16777215).toString(16);
</script>
```

Finally, we will access the `fillRect()` of our `ctx` object.

```
<script>
  var canvas = document.getElementById('canvas');
  var ctx = canvas.getContext('2d');
  ctx.fillStyle = '#' + Math.floor(Math.random()*16777215).toString(16);
  ctx.fillRect(10, 10, 100, 100);
</script>
```

Save the follow to */var/www/mtbc/draw/rand.html*. We will use `Math.random()` to draw a square in a random location in the canvas. We will set this to be 100 pixels shy of the canvas height and width so that the box will always appear in range. Navigate to http://localhost/mtbc/draw/rand.html and refresh the screen a few times to watch the box move.

```
<!DOCTYPE html>
<html lang="en">
    <head>
       <meta charset="UTF-8">
       <title>Hello World</title>
       <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>

      <canvas id="canvas" height="600px" width="800px"></canvas>

      <script>

        //Plot a random x
        var x = Math.random() * 700;

        //Plot a random y
        var y = Math.random() * 500;

        //Initialize the canvas
        var canvas = document.getElementById('canvas');

        //create a context
        var ctx = canvas.getContext('2d');

        //Create a random color
        ctx.fillStyle = '#' + Math.floor(Math.random()*16777215).toString(16);

        //Create the rectangle
        ctx.fillRect(x, y, 100, 100);

      </script>
    </body>
</html>
```

Add a circle

CanvasRenderingContext2D.arc() is the method used for drawing a circle. For a rectangle the x,y coordinates represent the the top left corner of a the rectangle for a circle x,y represents the center of a circle. Where the third and fourth arguments of fillRect represent the height and width the third and fourth arguments represent the start and ending point of the arc to be drawn, for a complete circle you will always start at 0 and end at tau or pi^2 (2*pi).

Add the following to the bottom of the script tags in your */var/www/mtbc/draw/rand.html* file and refresh the page.

```
//Create a circle
ctx.beginPath();
ctx.arc(230, 400, 83, 0, 2 * Math.PI);
ctx.stroke();
```

Drawing Program

For this exercise we will create a drawing program. This will allow the user to choose shapes and colors then they will be able to drop images onto the canvas.

Create the paths

- */var/www/draw/program.html*
- */var/www/draw/src/js/main.js*
- */var/www/draw/src/css/main.css*

Add the following to program.html. In this exercise we load Normailize.css to maintain consistency across browsers.

```
<!DOCTYPE html>
<html lang="en">
    <head>
       <meta charset="UTF-8">
       <title>Draw</title>
       <meta name="viewport" content="width=device-width, initial-scale=1.0">
       <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/7.0.0/normalize.min.css">
       <link rel="stylesheet" href="src/css/main.css">
    </head>
    <body>
      <div class="wrapper">
        <nav>
          <ul>
            <li id="trackX"></li>
            <li id="trackY"></li>
          </ul>
        </nav>
        <main></main>
      </div>
      <script src="src/js/main.js"></script>
    </body>
</html>
```

Add the following to */var/www/draw/src/js/main.js*

```
//Get the height and width of the main we will use this set canvas to the full
//size of the main element.
var mWidth = document.querySelector('main').offsetWidth;
var mHeight = document.querySelector('main').offsetHeight;

//Create the canvas
var canvas = document.createElement("canvas");
canvas.width = mWidth;
canvas.height = mHeight;
document.querySelector('main').appendChild(canvas);

//Create the context
var ctx = canvas.getContext("2d");

//Draw some sample rectangles
ctx.fillStyle = "rgb(200,0,0)";
ctx.fillRect (10, 10, 55, 50);

ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
ctx.fillRect (30, 30, 55, 50);
```

Add the following to */var/www/draw/src/css/main.css*.

```
html {
  font-family: sans-serif;
}

.wrapper {
  display: flex;
}

nav {
  flex: 0 0 300px;
  background: #ccc;
  min-height: 100vh;
}

main {
  flex: 1;
}

nav ul {
  list-style-type: none;
  padding: 0;
  margin: 0;
}

nav ul li {
  padding: 0;
  margin: 0;

}

nav ul li span,
nav ul li button {
  display: block;
  padding: 1em;
  text-align: center;
}

nav ul li button {
  width: 100%;
}
```

Naviagte to http://localhost/draw/program.html and you'll see a canvas beside a space for a side nav. The canvas will have two rectangles.

In order to be able to draw a we will need to know where our cursor is on the canvas. Let's start by tracking our mouse movements adding an event listener to the canvas. The listener will will trigger functionality every time the pixel position changes by a pixel. We will use Element.getBoundingClientRect() to get the position of the cursor.

```
//Track the x,y position
canvas.addEventListener('mousemove', function(evt) {

  //Calculate the x,y cords.
  var rect = canvas.getBoundingClientRect();
  let x = evt.clientX - rect.left;
  let y = evt.clientY - rect.top;

  //Write the cords back the UI.
  document.getElementById('trackX').innerHTML = 'X: ' + x;
  document.getElementById('trackY').innerHTML = 'Y: ' + y;

}, false);
```

As you move your mouse around the canvas you'll notice the x,y cords now appear in your sidebar.

Refactoring

So far all of our code has been written out in the global name space which means the data also lives in the global name space. It is considered good practice to restrict access to you program data and methods (functions) to a non-global scope. For this we will use a (closure)[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures]. To me more precise we will use a closure that emulates private data.

Replace your JavaScript with the following closure.

```
var draw = (function() {

  //Get the height and width of the main we will use this set canvas to the full
  //size of the main element.
```

```
    var mWidth = document.querySelector('main').offsetWidth,
      mHeight = document.querySelector('main').offsetHeight,

      //Create the canvas
      canvas = document.createElement("canvas"),

      //Create the context
      ctx = canvas.getContext("2d"),

      //Create the initial bounding rectangle
      rect = canvas.getBoundingClientRect(),

      //current x,y position
      x=0,
      y=0;

    return {
      //Set the x,y coords based on current event data
      setXY: function(evt) {
        x = (evt.clientX - rect.left) - canvas.offsetLeft;
        y = (evt.clientY - rect.top) - canvas.offsetTop;
      },

      //Write the x,y coods to the target div
      writeXY: function() {
        document.getElementById('trackX').innerHTML = 'X: ' + x;
        document.getElementById('trackY').innerHTML = 'Y: ' + y;
      },

      //Draw a rectangle
      drawRect: function() {
        //Draw some sample rectsangles
        ctx.fillStyle = "rgb(200,0,0)";
        ctx.fillRect (10, 10, 55, 50);
      },

      getCanvas: function(){
        return canvas;
      },

      //Initialize the object, this must be called before anything else
      init: function() {
        canvas.width = mWidth;
        canvas.height = mHeight;
        document.querySelector('main').appendChild(canvas);

      }
    };

  })();

  //Initialize draw
  draw.init();

  //Add a mousemove listener to the canvas
  //When the mouse reports a change of position use the event data to
  //set and report the x,y position on the mouse.
  draw.getCanvas().addEventListener('mousemove', function(evt) {
    draw.setXY(evt);
    draw.writeXY();
  }, false);

  //draw a sample rectangle
  draw.drawRect();
```

The previous code creates a draw object which will love on the global name space. The advantage is that the objects data and methods are encapsulated within the closure so they are protected from manipulation from any code that might share a common property name. For example, `x` might be a variable name in a third party library you have included in your project, the value of that libraries x could interfere (or collide) with the value of your programs x. Encapsulating as draw.x reduces the chance of collision; treating this a private property protects this even further. You could further protect the draw method by creating a namespace. Reversing your domain name is a common practice for namespacing. The real goal is to try and create something that is as unique as possible, a domain name you own should accomplish this, but it is not required. For the sake of this exercise we will forgo the use of a namespace.

```
var com.example.app = com.example.app || {};
com.example.app.draw = (function() {});
```

So far we have created a draw object which creates a canvas, draws a rectangle and reports any mouse movement that occurs on the canvas. It's time to let the user decide what to draw. We will start by removing the command to draw the red rectangle.

Remove the following from your JavaScript

```
//draw a sample rectangle
draw.drawRect();
```

In order to draw a rectangle we need to know the x,y coordinates of the top left corner as well as the desired height and width. The `mousemove` event tells us where we are on the canvas but we need to know where the user wants to draw the triangle and how big they want it. We can figure this out by using two more events `mousedown` and `mouseup`. This will give us the starting and ending points of a mouse drag. `mousedown` gives us the starting x,y coordinates or x1,y1. `mouseup` gives us the ending x,y coordinates or x2,y2. from tis we can calculate the height and width as *width=x2-x1* and *h=y2-y1*

We will start by adding variables for x1,y1,x2,y2.

Change

```
//current x,y position
x=0,
y=0;
```

to

```
//current x,y
x=0,
y=0,

//starting x,y
x1=0,
y1=0,

//ending x,y
x2=0,
y2=0;
```

Update the drawRect() method as follows.

```
drawRect: function(x,y,h,w) {

  //Start by using random fill colors.
  ctx.fillStyle = '#'+Math.floor(Math.random()*16777215).toString(16);

  ctx.fillRect (x1,y1,(x2-x1),(y2-y1));

}
```

Now we need to add two methods to set the starting and ending coordinates. Add the following bellow the writeXY() function. In OOP this knod of method is typically referred to a setter. In that the purpose of this method is to set/change the value of a property, typically a private property.

```
setStart: function() {
  x1=x;
  y1=y;
},

setEnd: function() {
  x2=x;
  y2=y;
},
```

That covers the draw object's implementation details for drawing a rectangle, now all that is left is for us to allow our user to draw the rectangle. We will start by adding a `mousdown` listener to the canvas. When this event is triggered we simply call the `setStart()` method to record the starting position, this defines the top left corner of the triangle. Add the following to the end of your JavaScript.

```
//Set the starting positon
draw.getCanvas().addEventListener('mousedown', function() {
  draw.setStart();
}, false);
```

To get the ending position we add a `mouseup` listener to the canvas and call the `setEnd()` method. At this point x1,y1,x2 and y2 are all set, this all we need to draw the rectangle, so we will also call the `drawRect()` function. Add the following to the end of your JavaScript.

```
draw.getCanvas().addEventListener('mouseup', function() {
  draw.setEnd();
  draw.drawRect();
}, false);
```

At this point your JavaScript should read as follows.

```
var draw = (function() {

  //Get the height and width of the main we will use this set canvas to the full
  //size of the main element.
  var mWidth = document.querySelector('main').offsetWidth,
    mHeight = document.querySelector('main').offsetHeight,

    //Create the canvas
    canvas = document.createElement("canvas"),

    //Create the context
    ctx = canvas.getContext("2d"),

    //Create the initial bounding rectangle
    rect = canvas.getBoundingClientRect(),

    //current x,y position
    x=0,
    y=0,

    //starting x,y
    x1=0,
    y1=0,
    //ending x,y
    x2=0,
    y2=0;

  return {
    //Set the x,y coords based on current event data
    setXY: function(evt) {
      x = (evt.clientX - rect.left) - canvas.offsetLeft;
      y = (evt.clientY - rect.top) - canvas.offsetTop;
    },

    //Write the x,y coods to the target div
    writeXY: function() {
      document.getElementById('trackX').innerHTML = 'X: ' + x;
      document.getElementById('trackY').innerHTML = 'Y: ' + y;
    },

    //Set the x1,y1
    setStart: function() {
      x1=x;
      y1=y;
    },

    //Set the x2,y2
    setEnd: function() {
      x2=x;
      y2=y;
    },

    //Draw a rectangle
    drawRect: function() {
      //Start by using random fill colors.
      ctx.fillStyle = '#'+Math.floor(Math.random()*16777215).toString(16);
      ctx.fillRect (x1,y1,(x2-x1),(y2-y1));
    },

    getCanvas: function(){
      return canvas;
    },

    //Initialize the object, this must be called before anything else
    init: function() {
      canvas.width = mWidth;
      canvas.height = mHeight;
      document.querySelector('main').appendChild(canvas);

    }
  };

})();

//Initialize draw
draw.init();

//Add a mousemove listener to the canvas
//When the mouse reports a change of position use the event data to
//set and report the x,y position on the mouse.
draw.getCanvas().addEventListener('mousemove', function(evt) {
  draw.setXY(evt);
  draw.writeXY();
}, false);

//Add a mousedown listener to the canvas
//Set the starting position
draw.getCanvas().addEventListener('mousedown', function() {
  draw.setStart();
}, false);

//Add a mouseup listener to the canvas
```

```
//Set the end position and draw the rectangle
draw.getCanvas().addEventListener('mouseup', function() {
  draw.setEnd();
  draw.drawRect();
}, false);
```

At some point we want to draw shapes other than a rectangle. Let's start by creating a button that allows to choose a rectangle then we can add more buttons for more shapes. Let's by adding a rectangle button to the navigation list. We will give this button an id of *btnRect*.

```
<li><button id="btnRect">Rectangle</button></li>
```

We will need to two things to make this work (1) wire the button up to an `onclick` event listener and (2) wire that `onclick` event up draw object so that the object knows what to do. Let's start by adding a shape variable as a private property of the draw object.

Let's change

```
//ending x,y
x2=0,
y2=0;
```

to

```
//ending x,y
x2=0,
y2=0,

//What shape are we drawing?
shape='';
```

Now we will need a setter, add the following below the setEnd() method.

```
//Sets the shape to be drawn
setShape: function(shp) {
  shape = shp;
},
```

Now add a listener to the bottom of the JavaScript file. This is listen for a `click` on *btnRect*.

```
document.getElementById('btnRect').addEventListener('click',function(){
    draw.setShape('rectangle');
}, false);
```

Now that we have set the shape, we need it to cause an effect in the draw object. Add the following below the `setShape()` method.

```
draw: function() {
  if('shape'==='rectangle')
  {
    this.drawRect();
  } else {
    alert('Please choose a shape');
  }
},
```

and replace the call to `draw.drawRect()` in the `mouseup` listener with `draw.draw()` as follows.

```
draw.getCanvas().addEventListener('mouseup', function() {
  draw.setEnd();
  draw.draw();
}, false);
```

Since drawing on the canvas can change the underlying grid it is a good practice to reset the grid before the next item is drawn on the canvas. This is easily achieved using `ctx.restore()` and `ctx.save()`. The most efficient way to implement this is by adding it to the draw method, that way all future shapes will automatically perform these tasks. You can read more about save and restore here.

```
//Draws the selected shape
draw: function() {
  ctx.restore();
  if(shape==='rectangle')
  {
    this.drawRect();
  } else {
    alert('Please choose a shape');
  }
  ctx.save();
},
```

Now we want to add more shapes, let's start with a line. To begin drawing a line you need to begin a path, set x1,y1 and x2,y2, then call the stroke method to draw the line. Which looks like the following.

```
ctx.beginPath();
ctx.moveTo(x1, y1);
ctx.lineTo(x2, y2);
ctx.stroke();
```

We will start by creating a drawLine() method and adding it to our draw method add the following below the `draw()` method.

```
//Draw a line
drawLine: function() {
  //Start by using random fill colors.
  ctx.strokeStyle = '#'+Math.floor(Math.random()*16777215).toString(16);
  ctx.beginPath();
  ctx.moveTo(x1, y1);
  ctx.lineTo(x2, y2);
  ctx.stroke();
},
```

and change the draw method as follows.

```
//Draws the selected shape
draw: function() {
  ctx.restore();
  if(shape==='rectangle')
  {
    this.drawRect();
  } else if(shape==='line') {
    this.drawLine();
  } else {
    alert('Please choose a shape');
  }
  ctx.save();
},
```

Add a line button the the nav list.

```
<li><button id="btnLine">Line</button></li>
```

Then a listener for the click event.

```
document.getElementById('btnLine').addEventListener('click',function(){
    draw.setShape('line');
}, false);
```

Repeat the previous steps but in the context of drawing a circle. The `drawCircle()` will simply create an `alert()` stating it doesn't do anything. At this point your JavaScript should read as follows.

```
var draw = (function() {

  //Get the height and width of the main we will use this set canvas to the full
  //size of the main element.
  var mWidth = document.querySelector('main').offsetWidth,
    mHeight = document.querySelector('main').offsetHeight,

    //Create the canvas
    canvas = document.createElement("canvas"),

    //Create the context
    ctx = canvas.getContext("2d"),

    //Create the initial bounding rectangle
    rect = canvas.getBoundingClientRect(),

    //current x,y position
    x=0,
    y=0,

    //starting x,y
    x1=0,
    y1=0,

    //ending x,y
    x2=0,
    y2=0,

    //What shape are we drawing?
    shape='';

  return {
    //Set the x,y coords based on current event data
    setXY: function(evt) {
      x = (evt.clientX - rect.left) - canvas.offsetLeft;
      y = (evt.clientY - rect.top) - canvas.offsetTop;
    },

    //Write the x,y coods to the target div
    writeXY: function() {
      document.getElementById('trackX').innerHTML = 'X: ' + x;
      document.getElementById('trackY').innerHTML = 'Y: ' + y;
    },

    //Set the x1,y1
    setStart: function() {
      x1=x;
      y1=y;
    },

    //Set the x2,y2
    setEnd: function() {
      x2=x;
      y2=y;
    },

    //Sets the shape to be drawn
    setShape: function(shp) {
      shape = shp;
    },

    //Draws the selected shape
    draw: function() {
      ctx.restore();
      if(shape==='rectangle')
      {
        this.drawRect();
      } else if( shape==='line' ) {
        this.drawLine();
      } else if( shape==='circle' ) {
        this.drawCircle();
      } else {
        alert('Please choose a shape');
      }
      ctx.save();
    },


    //Draw a circle
    drawCircle: function() {
      alert('I don\'t do anything yet.');
    },

    //Draw a line
    drawLine: function() {
      //Start by using random fill colors.
      ctx.strokeStyle = '#'+Math.floor(Math.random()*16777215).toString(16);
      ctx.beginPath();
      ctx.moveTo(x1, y1);
      ctx.lineTo(x2, y2);
      ctx.stroke();
    },

    //Draw a rectangle
    drawRect: function() {
      //Start by using random fill colors.
      ctx.fillStyle = '#'+Math.floor(Math.random()*16777215).toString(16);
      ctx.fillRect (x1,y1,(x2-x1),(y2-y1));
    },

    getCanvas: function(){
      return canvas;
    },

    //Initialize the object, this must be called before anything else
    init: function() {
      canvas.width = mWidth;
      canvas.height = mHeight;
      document.querySelector('main').appendChild(canvas);

    }
  };

})();

//Initialize draw
draw.init();

//Add a mousemove listener to the canvas
//When the mouse reports a change of position use the event data to
//set and report the x,y position on the mouse.
draw.getCanvas().addEventListener('mousemove', function(evt) {
  draw.setXY(evt);
  draw.writeXY();
}, false);

//Add a mousedown listener to the canvas
//Set the starting position
draw.getCanvas().addEventListener('mousedown', function() {
  draw.setStart();
}, false);

//Add a mouseup listener to the canvas
//Set the end position and draw the rectangle
draw.getCanvas().addEventListener('mouseup', function() {
  draw.setEnd();
  draw.draw();
}, false);

document.getElementById('btnRect').addEventListener('click', function(){
    draw.setShape('rectangle');
}, false);

document.getElementById('btnLine').addEventListener('click', function(){
    draw.setShape('line');
}, false);

document.getElementById('btnCircle').addEventListener('click', function(){
    draw.setShape('circle');
}, false);
```

and the HTML

```
<!DOCTYPE html>
<html lang="en">
    <head>
       <meta charset="UTF-8">
       <title>Draw</title>
       <meta name="viewport" content="width=device-width, initial-scale=1.0">
       <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/7.0.0/normalize.min.css">
       <link rel="stylesheet" href="src/css/main.css">
    </head>
    <body>
      <div class="wrapper">
        <nav>
          <ul>
            <li><span id="trackX"></span></li>
            <li><span id="trackY"></span></li>
            <li><button id="btnRect">Rectangle</button></li>
            <li><button id="btnLine">Line</button></li>
            <li><button id="btnCircle">Circle</button></li>
          </ul>
        </nav>
        <main></main>
      </div>
      <script src="src/js/main.js"></script>
    </body>
</html>
```

We have everything we need to draw a circle which in this context is a 360 degree arc. Our starting x,y position (x1, y1) represents the center of the circle our stopping point (x2,y2) rests on the circumference of the circle. With these two points we can use Pythagoreans theorem $A^2 + B^2 = C^2$ to calculate the radius of the circle.

In code this might look like.

```
let a = (x1-x2)
let b = (y1-y2)
let radius = Math.sqrt( a*a + b*b );
```

I think the first three arguments in the `arc()` method are pretty clear *x,y,radius*. *x,y* defines the center of the circle and of course *radius* is what we just calculated. That leaves us with *startAngle* and *endAngle*. Since we are drawing a complete circle the angle will always start where it ends. This is represented as 0 and 2 times pi `2*Math.PI`.

Update your `drawCircle()` method to the following.

```
//Draw a circle
drawCircle: function() {

  ctx.strokeStyle = '#'+Math.floor(Math.random()*16777215).toString(16);
  ctx.fillStyle = '#'+Math.floor(Math.random()*16777215).toString(16);

  let a = (x1-x2)
  let b = (y1-y2)
  let radius = Math.sqrt( a*a + b*b );

  ctx.beginPath();
  ctx.arc(x1, y1, radius, 0, 2*Math.PI);
  ctx.stroke();
  ctx.fill();
},
```

Drawing Paths

We will define a path as a line the follows your cursor during a *mousedown* (drag) event. There is no context method for this so we will need to create one. How would we define this in code? We start by understanding what it is we want to do.

1. Select a shape called *path* this will tell the object that we want to draw a path.
2. On *mousedown* we will start drawing.
3. On *mouseup* we will stop drawing.
4. Between the *mouseup* and *mousedown* we will draw a line every time the mouse reports a move event.
    i. This line will connect the previously reported coordinates to the current coordinates.
        - OR you could say we a drawing a one pixel by one pixel line everytime the mouse moves by one pixel in any direction.

We will start by adding a new button and listener for drawing the path.

The button

```
<li><button id="btnPath">Path</button></li>
```

and the listener.

```
document.getElementById('btnPath').addEventListener('click', function(){
    draw.setShape('path');
}, false);
```

As in the previous example we will add a `drawPath()` method and add `drawPath()` to the draw method.

drawPath()

```
drawPath: function() {
  alert('I don\'t do anything yet.');
},
```

draw()

```
//Draws the selected shape
draw: function() {
  ctx.restore();
  if(shape==='rectangle')
  {
    this.drawRect();
  } else if( shape==='line' ) {
    this.drawLine();
  } else if( shape==='circle' ) {
    this.drawCircle();
  } else {
    alert('Please choose a shape');
  }
  ctx.save();
},
```

At this point, for all previous draw* methods we would complete our implementation details by hooking a few context methods and we would be done with it. For this method to work we need to track the previous x,y state and draw on *mousemove* instead of *mouseup* which means we also need to capture a drawing state (we will call that *isDrawing*).

Lets start with by tracking the previous coordinates, we will call these lx,ly (short hand for *last x* and *last y*).

Add the following variables to as private properties.

```
//Tracks the last x,y state
lx = false,
ly = false,
```

Update your setXY() method so that it updates lx,ly.

```
//Set the x,y coords based on current event data
setXY: function(evt) {

  //Track the last x,y position before setting the current position.
  lx=x;
  ly=y;

  //Set the current x,y position
  x = (evt.clientX - rect.left) - canvas.offsetLeft;
  y = (evt.clientY - rect.top) - canvas.offsetTop;
},
```

Now we need to call the draw method from our *mousemove* listener but only if we have chosen to draw a path. This means we need to expose the private *path* property through a public api we will call this *getShape()*.

```
getShape: function() {
  return shape;
},
```

Now that the draw object has a way to report its shape we can use that call the *draw()* method from the *mousemove* listener.

```
draw.getCanvas().addEventListener('mousemove', function(evt) {
  draw.setXY(evt);
  draw.writeXY();
  if(draw.getShape()=='path') {
    draw.draw();
  }
}, false);
```

Now lets implement our *drawPath()* method. This is almost identical to the *drawLine()* method. Instead of drawing from x1,y1 to x2,y2 we will draw from lx,ly to x,y.

```
drawPath: function() {
  //Start by using random fill colors.
  ctx.strokeStyle = '#'+Math.floor(Math.random()*16777215).toString(16);
  ctx.beginPath();
  ctx.moveTo(lx, ly);
  ctx.lineTo(x, y);
  ctx.stroke();
},
```

Now choosing the path shape and mousing over the canvas will start drawing with out a mouse click. We want a drag to define weather or not the mouse movement should draw we will do this by capturing a state called *isDrawing*.

Add the a *isDrawing* variable as a private property.

```
isDrawing=false;
```

Now we will create a setter and getter to allow us access through a public interface.

```
setIsDrawing: function(bool) {
  isDrawing = bool;
},

getIsDrawing: function() {
  return isDrawing;
},
```

Now that we have an accessible *isDrawing* property that we can be toggled from true to false we can use this with our *mousedown* and *mouseup* events to control the when and how the path is drawn.

```
draw.getCanvas().addEventListener('mousemove', function(evt) {
  draw.setXY(evt);
  draw.writeXY();
  if(draw.getShape()=='path' && draw.getIsDrawing()===true) {
    draw.draw();
  }
}, false);
```

At this point your JavaScript should appear as follows.

```
var draw = (function() {

  //Get the height and width of the main we will use this set canvas to the full
  //size of the main element.
  var mWidth = document.querySelector('main').offsetWidth,
    mHeight = document.querySelector('main').offsetHeight,

    //Create the canvas
    canvas = document.createElement("canvas"),

    //Create the context
    ctx = canvas.getContext("2d"),

    //Create the initial bounding rectangle
    rect = canvas.getBoundingClientRect(),

    //current x,y position
    x=0,
    y=0,
    //starting x,y
    x1=0,
    y1=0,

    //ending x,y
    x2=0,
    y2=0,

    //Tracks the last x,y state
    lx = false,
    ly = false,

    //What shape are we drawing?
    shape='',

    //Are we drawimg a path?
    isDrawing=false;

  return {

    //Set the x,y coords based on current event data
    setXY: function(evt) {

      //Track last x,y position before setting the current posiiton.
      lx=x;
      ly=y;

      //Set the current x,y position
      x = (evt.clientX - rect.left) - canvas.offsetLeft;
      y = (evt.clientY - rect.top) - canvas.offsetTop;
    },

    //Write the x,y coods to the target div
    writeXY: function() {
      document.getElementById('trackX').innerHTML = 'X: ' + x;
      document.getElementById('trackY').innerHTML = 'Y: ' + y;
    },

    //Set the x1,y1
    setStart: function() {
      x1=x;
      y1=y;
    },

    //Set the x2,y2
    setEnd: function() {
      x2=x;
      y2=y;
    },

    //Sets the shape to be drawn
    setShape: function(shp) {
      shape = shp;
    },

    getShape: function() {
      return shape;
    },

    setIsDrawing: function(bool) {
      isDrawing = bool;
    },

    getIsDrawing: function() {
      return isDrawing;
    },

    //Draws the selected shape
    draw: function() {
      ctx.restore();
      if(shape==='rectangle')
      {
        this.drawRect();
      } else if( shape==='line' ) {
        this.drawLine();
      } else if( shape==='path' ) {
        this.drawPath();
      } else if( shape==='circle' ) {
        this.drawCircle();
      } else {
        alert('Please choose a shape');
      }
      ctx.save();
    },

    //Draw a circle
    drawCircle: function() {

      ctx.strokeStyle = '#'+Math.floor(Math.random()*16777215).toString(16);
      ctx.fillStyle = '#'+Math.floor(Math.random()*16777215).toString(16);

      let a = (x1-x2)
      let b = (y1-y2)
      let radius = Math.sqrt( a*a + b*b );

      ctx.beginPath();
      ctx.arc(x1, y1, radius, 0, 2*Math.PI);
      ctx.stroke();
      ctx.fill();
    },

    //Draw a line
    drawLine: function() {
      //Start by using random fill colors.
      ctx.strokeStyle = '#'+Math.floor(Math.random()*16777215).toString(16);
      ctx.beginPath();
      ctx.moveTo(x1, y1);
      ctx.lineTo(x2, y2);
      ctx.stroke();
    },


    drawPath: function() {
      //console.log({x1:x,y1:y,x2:x2,y2:y2});
      //Start by using random fill colors.
      ctx.strokeStyle = '#'+Math.floor(Math.random()*16777215).toString(16);
      ctx.beginPath();
      ctx.moveTo(lx, ly);
      ctx.lineTo(x, y);
      ctx.stroke();
    },

    //Draw a rectangle
    drawRect: function() {
      //Start by using random fill colors.
      ctx.fillStyle = '#'+Math.floor(Math.random()*16777215).toString(16);
      ctx.fillRect (x1,y1,(x2-x1),(y2-y1));
    },

    getCanvas: function(){
      return canvas;
    },

    //Initialize the object, this must be called before anything else
    init: function() {
      canvas.width = mWidth;
      canvas.height = mHeight;
```

```
        document.querySelector('main').appendChild(canvas);

      }
    };

  })();

  //Initialize draw
  draw.init();

  //Add a mousemove listener to the canvas
  //When the mouse reports a change of position use the event data to
  //set and report the x,y position on the mouse.
  draw.getCanvas().addEventListener('mousemove', function(evt) {
    draw.setXY(evt);
    draw.writeXY();
    if(draw.getShape()=='path' && draw.getIsDrawing()===true) {
      draw.draw();
    }
  }, false);

  //Add a mousedown listener to the canvas
  //Set the starting position
  draw.getCanvas().addEventListener('mousedown', function() {
    draw.setStart();
    draw.setIsDrawing(true);
  }, false);

  //Add a mouseup listener to the canvas
  //Set the end position and draw the rectangle
  draw.getCanvas().addEventListener('mouseup', function() {
    draw.setEnd();
    draw.draw();
    draw.setIsDrawing(false);
  }, false);

  document.getElementById('btnRect').addEventListener('click', function(){
      draw.setShape('rectangle');
  }, false);

  document.getElementById('btnLine').addEventListener('click', function(){
      draw.setShape('line');
  }, false);

  document.getElementById('btnCircle').addEventListener('click', function(){
      draw.setShape('circle');
  }, false);

  document.getElementById('btnPath').addEventListener('click', function(){
      draw.setShape('path');
  }, false);
```

your HTML should read as follows

```
<!DOCTYPE html>
<html lang="en">
    <head>
       <meta charset="UTF-8">
       <title>Draw</title>
       <meta name="viewport" content="width=device-width, initial-scale=1.0">
       <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/7.0.0/normalize.min.css">
       <link rel="stylesheet" href="src/css/main.css">
    </head>
    <body>
      <div class="wrapper">
        <nav>
          <ul>
```

```
              <li><span id="trackX"></span></li>
              <li><span id="trackY"></span></li>
              <li><button id="btnRect">Rectangle</button></li>
              <li><button id="btnLine">Line</button></li>
              <li><button id="btnCircle">Circle</button></li>
              <li><button id="btnPath">Path</button></li>
            </ul>
          </nav>
          <main></main>
        </div>
        <script src="src/js/main.js"></script>
      </body>
</html>
```

and your CSS

```
html {
  font-family: sans-serif;
}

.wrapper {
  display: flex;
}

nav {
  flex: 0 0 300px;
  background: #ccc;
  min-height: 100vh;
}

main {
  flex: 1;
}

nav ul {
  list-style-type: none;
  padding: 0;
  margin: 0;
}

nav ul li {
  padding: 0;
  margin: 0;

}

nav ul li span,
nav ul li button {
  display: block;
  padding: 1em;
  text-align: center;
}

nav ul li button {
  width: 90%;
  margin: 0 auto;
}
```

LAB

Now that you can add basic shapes to the canvas lets work on the colors.

- Add color pickers that will allow the user to select the stroke and fill colors for various objects they are using.
    - Hint: HTML5 has a built in form element that spawns a color picker.

- Can you figure out how to draw a triangle?
- A value of 0 should load the the *trackX* and *trackY* divs when at page render.

Additional Resources

- Canvas API
- Understanding save() and restore() for the Canvas Context
- Random Hex Color Code Generator in JavaScript

Khan Academy

- Radius, diameter, circumference & π
- Pythagorean Theorem

Udemey

- Introduction to HTML5 Canvas basics of drawing
- Search Canvas

Next: jQuery

jQuery

According to the jQuery website "jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript."[1].

Why we really use it

- Cross browser compatibility.
- Simply DOM manipulation.
- Nicely encapsulates AJAX functionality.
- A good eco-system, lots of third party plugins.
  - A requirement for Bootstrap.

jQuery allows you to simply manipulate the DOM by calling CSS selectors. This is a little more consitent than the native *querySelector*. Legacy browsers do not support *querySelector()* but that is less of a problem now-a-days.

```
// When an element with the id of _colorChanger_ is clicked apply a red font to all _paragraphs_
$( "#colorChanger" ).on( "click", function( event ) {
  $('p').attr('style', 'color: #ff9900');
});

// When an element with the id of _colorChanger_ is clicked apply a red font to all _paragraphs_ with a class o
f _.red_
$( "#colorChanger" ).on( "click", function( event ) {
  $('p.red').attr('style', 'color: #ff9900');
});

// When an element with the id of _colorChanger_ is clicked apply a red font to all _elements_ with a class of
_.red_
$( "#colorChanger" ).on( "click", function( event ) {
  $('.red').attr('style', 'color: #ff9900');
});
```

```
//Show an AJAX example
```

Exercise - NASA API

- Get an API KEY from NASA by filling out the form and checking your email.
- Clone the html-starter project to */var/www*
- Rename the project to *nasa*
- Add the remotes from your newly created GitHub Project.
- Install your NPM dependencies
- Install normalize.css (NPM)
- Install JQuery (NPM)
- Add normalize.css and jQuery to your Gulp file
- Start your watcher

Create a basic HTML structure and add it to *index.html*. For this example, lets use NPM and those types of tools???

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="utf-8">
    <title>NASA - Astronomy Picture of the Day
  </title>
    <link rel="stylesheet" href="./dist/css/main.css">
  </head>
  <body>

    <script src="./dist/js/main.js"></script>
  </body>
</html>
```

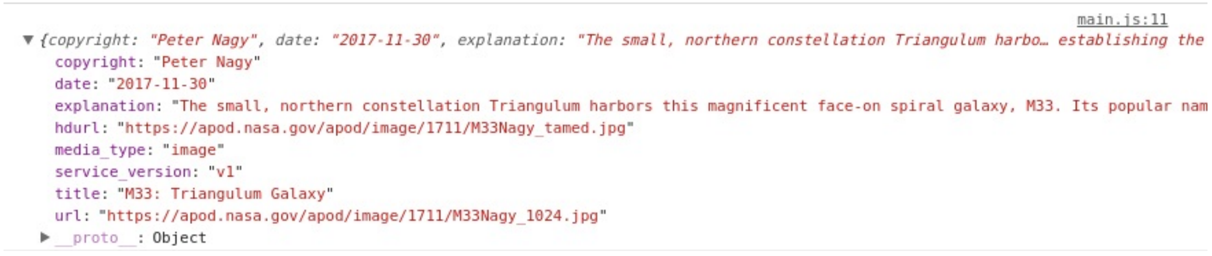
Let's create an object (and clos ure) called apod (Astronomy Picture of the Day). We will make an AJAX call to the API which will return a JSON string, this is what we will use to build the program. We will test our API access by returning the result of the AJAX request to a console log. Press [F12] and find the console tab in your browsers developer tools. Add the following to *src/js/main.js*.

```
var apod = {
    // Application Constructor
    init: function() {

        var url = "https://api.nasa.gov/planetary/apod?api_key=YOUR-KEY-HERE";

        $.ajax({
            url: url
        }).done(function(result){
          console.log(result);
        }).fail(function(result){
          console.log(result);
        });
    },
};

apod.init();
```

and add the follow to *src/scss/main.scss*.

```
body {
  padding: 0;
  margin: 0;
}

main {
  width: 720px;
  margin: 0 auto;
}

#apodImg {
  max-width: 100%
}

div[id^=apod] {
  padding: .6rem 0;
  font-size: 20px;
}

/* https://alistapart.com/article/creating-intrinsic-ratios-for-video */
/* 9/16 = 56.2 */
.video {
    position: relative;
    padding-bottom: 56.25%; /* Assumes a 16:9 ratio */
    padding-top: 25px;
    height: 0;
}

.video iframe {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
}
```

If eveything worked you will see results similar to the following.



```
                                                                                                              main.js:11
▼{copyright: "Peter Nagy", date: "2017-11-30", explanation: "The small, northern constellation Triangulum harbo… establishing the
   copyright: "Peter Nagy"
   date: "2017-11-30"
   explanation: "The small, northern constellation Triangulum harbors this magnificent face-on spiral galaxy, M33. Its popular nam
   hdurl: "https://apod.nasa.gov/apod/image/1711/M33Nagy_tamed.jpg"
   media_type: "image"
   service_version: "v1"
   title: "M33: Triangulum Galaxy"
   url: "https://apod.nasa.gov/apod/image/1711/M33Nagy_1024.jpg"
 ▶ __proto__: Object
```

In looking at the JSON data you'll notice a date field. By default only pull today's picture, looking at the query parameters section in the API documentation I see I can pass a date in the form of *YYYY-MM-DDD* as an additional GET parameter. To make things interesting lets add pass a random date every time we call the API.

Add a random date function to the apod object. A good place to start would be MDN's date documentation. A qucik Google search will return this gist which provides us a good randomizer for an unformatted date in between a given start and date. This is important because the date cannot be greater than today or less that the first apod *June 16, 1995*.

```
//Create a random date
randomDate: function(start, end) {
  //Randomize the date https://gist.github.com/miguelmota/5b67e03845d840c949c4
  let date = new Date(start.getTime() + Math.random() * (end.getTime() - start.getTime()));

  //Format the date
  let d = date.getDate();
  let m = date.getMonth() + 1; //In JS months start at 0
  let y = date.getFullYear();

  //Change the maonth and day strings so that they match the documented format.
  if(m < 10){
    m = '0'+m
  }

  if(d < 10){
    m = '0'+d
  }

  return `${y}-${m}-${d}`;
},
```

Update the init() method as follows.

```
let date = this.randomDate(new Date(1995, 5, 16), new Date());
var url = "https://api.nasa.gov/planetary/apod?api_key=LcGRMR8ReXp7B91eLkhqcSag0JYHQKh2Y5MAAXHY&date=" + date;
```

Now when you refresh the page you will see different JSON strings.

At this point your JS should resemble the following.

```
var apod = {
    //Create a random date
    randomDate: function(start, end) {
      //Randomize the date https://gist.github.com/miguelmota/5b67e03845d840c949c4
      let date = new Date(start.getTime() + Math.random() * (end.getTime() - start.getTime()));
```

```
        //Format the date
        let d = date.getDate();
        let m = date.getMonth() + 1; //In JS months start at 0
        let y = date.getFullYear();

        //Change the maonth and day strings so that they match the documented format.
        if(m < 10){
          m = '0'+m
        }

        if(d < 10){
          m = '0'+d
        }

        return `${y}-${m}-${d}`;
      },

      // Application Constructor
      init: function() {
          let date = this.randomDate(new Date(1995, 5, 16), new Date());
          var url = "https://api.nasa.gov/planetary/apod?api_key=LcGRMR8ReXp7B91eLkhqcSag0JYHQKh2Y5MAAXHY&date="
  + date;

          $.ajax({
              url: url
          }).done(function(result){
            $("#apodImg").attr("src", result.url);
            $("#apodCopyright").text("Copyright: " + result.copyright);
            $("#apodDate").text("Date: " + date);
            $("#apodDesc").text(result.explanation);
          }).fail(function(data){
            console.log(data);
          });
      },
  };
```

Now it's time to build the UI. We will start by deciding what we want to show on the page. For this exercise we will use the *result.url* (to show the image), *result.copyright* to give proper attribution, *result.date*, *apodTitle* and *result.explanation*.

Replace the success callback in your AJAX call with the following. This assumes the DOM has an element for each of the following ids.

- .attr()
- .text()

```
.done(function(result){
$("#apodTitle").text(result.title);
$("#apodImg").attr("src", result.url).attr('alt', result.title);
$("#apodCopyright").text("Copyright: " + result.copyright);
$("#apodDate").text("Date: " + date);
$("#apodDesc").text(result.explanation);
}).
```

Update *index.html* with the following.

```
<h1 id="apodTitle"></h1>
<img id="apodImg">
<div id="apodCopyright"></div>
<div id="apodDate"></div>
<div id="apodDesc"></div>
```

Hard code the date *6/6/2013* as follows and you will notice there is no image. That is because the picture of the day for this date is a video. In this situation we will want to tell our program how to render a video.

```
let date = new Date(2013,6,6);//new Date(start.getTime() + Math.random() * (end.getTime() - start.getTime()));
```

If we refer to the JSON string produced by the API we see that there is a *media_type* field. We can use this to tell our application how to handle the url.

```
//If the media type is video hide the image elements and display a video.
if(result.media_type === 'video') {
  $("#apodImage").hide();
  $("#apodVideo > iframe").attr("src", result.url).show();
}else{
  $("#apodVideo").hide();
  $("#apodImg").attr("src", result.url).attr('alt', result.title).show();
}
```

Add the following markup either above or below *apodImg_*. Refer to the CSS to see how the video class helps us properly size a video.

```
<div class="video" id="apodVideo">
  <iframe frameborder="0" allowfullscreen></iframe>
</div>
```

Single Responsibility Principle

The Single Responsibility Principle is the notion that a class, module, method, etc should only be responsible for one thing. For instance a method called *writeName()* might be expected to write a name to something. If the method were written as follows, it would be a good example of single responsibility.

```
writeName(name) {
  $('#firstName').text(name);
}
```

If however, I were to write the following

```
writeName(id) {
  let db = new DB{'user': 'root', 'password':'1234', 'db':'crm'});
  let results = db.sql('SELECT `name` FROM `contacts` WHERE contact.id=' + id);
  $('#firstName').text(results.name);
}
```

it would be a bad example of single responsibility in that the *writeName()* method is now responsible for

- Connecting to a database
- Executing a query to to find the name of the desired user.
- Then writing the users name to the DOM. And we have not even mentioned error handling yet.

As you can see based on a few comments, trying to d o to much on a single method can get out of hand pretty quickly.

```
writeName(id) {
  let db = new DB{'user': 'root', 'password':'1234', 'db':'crm'});
  //If you cannot connect the the DB
  //Create an error message
  //Figure out the best way to present that error to the user
  ////that would probably have something todo with calling a messaging object or else you'll have even more res
ponsibility in this method
  let results = db.sql('SELECT `name` FROM `contacts` WHERE contact.id=' + id);
  //rinse and repeat
  $('#firstName').text(results.name);
```

```
}
```

In our current implementation we are asking a lot of out *init()* method.

- It makes an AJAX request
- Processes the results
- Deals with errors (kind of)
- Rebuilds the DOM on success
- It initializes the page onload
- Rebuilds the page onclick

My original intent for the *init()* method was to build the page onload. It's sole responsibility then should be to call the methods required to make that happen.

Straight away I see at least to new methods that get me object closer to SRP.

- *buildDOM()*
- *getRequest()*

```
//Injects the results of the API call into the DOM
buildDOM: function(result) {
  $("#apodTitle").text(result.title);

  if(result.media_type === 'video') {
    $("#apodImage").hide();
    $("#apodVideo > iframe").attr("src", result.url).show();
  }else{
    $("#apodVideo").hide();
    $("#apodImg").attr("src", result.url).attr('alt', result.title).show();
  }

  $("#apodCopyright").text("Copyright: " + result.copyright);
  $("#apodDate").text("Date: " + result.date);
  $("#apodDesc").text(result.explanation);
},

//Executes an AJAX call to an API.
getRequest: function() {
  let _this = this;
  let date = this.randomDate(new Date(1995, 5, 16), new Date());
  let url = "https://api.nasa.gov/planetary/apod?api_key=LcGRMR8ReXp7B91eLkhqcSag0JYHQKh2Y5MAAXHY&date=" + date
;
  $.ajax({
      url: url
  }).done(function(result){
      _this.buildDOM(result);
  }).fail(function(result){
    console.log(result);
  });
},

// Initialization method.
init: function() {
  this.getRequest();
},
```

This may cause my execution to change as follows. You could also call *apod.gerRequest()* onload or call *apod.init()* on click. This is a simple example so the *init()* method may make a little less sense, in a more complex example the page load would likely have different responsibilities than the click event. In this case one might argue that calling a set of responsibilities in an *init()* method violates SRP and that these methods should be called individually in a script and I could not argue. At some point you have to make a decision and go with it.

```
apod.init();

/* https://learn.jquery.com/using-jquery-core/document-ready/ */
$(function() {
    $('#btnRandApod').on('click',function(){
      apod.getRequest();
    });
});
```

Lab 1 - SRP

Our code now has a pretty good break down, but I see a couple of more things that could be generalized. Review the code and see if you can break some methods down a little further.

LAB 2 - Convert to Vanilla JS

NASA API in Vanilla JS

- Create the GitHUb project apod-vanilla.
- Clone html-stater or your exisiting apod project.
- Rewrite the project without the use of jQuery.

Using the jQuery based code from the previous example as a guide, create the same functionality using vanilla JS. This will give you experience in writing AJAX logic using both jQuery and Vanilla JS.

Lab 3 - Port to jQuery

Create the project draw-jquery. Recreate the draw program using jQuery

Additional Resources

- jQuery
- jQuery vs document.querySelectorAll
- Creating Intrinsic Ratios for Video
- JavaScript Object Basics

Udemy

- Up and Running with jQuery

Next: Front End Toolkits

Chapter 8: Front End Toolkits

Next: Toolkits

Toolkits

Utilizing a toolkit can reduce the time it takes to get your product to market. Toolkits often take care of the repetitive tasks that while necessary change very little from project to project. These ofter employ "best" or "common" practices. While we will discuss a few toolkits our focus will be on only one; Bootstrap.

HTML Boilerplate

I would describe HTML5 Boilerplate as a collection of good practices, especially for those using Apache and it comes with a nice .htacces file that explains each setting in great detail.

Material Design Light

Material Design is Google's take on UI.

Bootstrap

A responsive, mobile first front end library.

Next: Bootstrap

Bootstrap

Bootstrap is a front end component library designed to allow for rapid front end development of responsive websites.

Lets start by creating a basic minimal Bootstrap layout that loads the Bootstrap assets from a CDN. Create the path and add the following */var/www/bootstrap/index.html*.

```
<DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>Bootstrap Demo</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css">
  </head>

  <body>
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.3/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/js/bootstrap.min.js"></script>
  </body>
</html>
```

Add some basic content at the top of the body.

```
<p>Hello World, I'm a Bootstrap layout.</p>
```

Lets center the content by adding a container.

```
<div class="container">
  <p>Hello World, I'm a Bootstrap layout.</p>
</div>
```

Grid layout

Add the following to the *.container* div below the paragraph. View this in both full screen and in phone mode to see how various view ports change the display.

```
<div class="row">
  <div class="col-md-4">Left Column</div>
  <div class="col-md-4">Center Column</div>
  <div class="col-md-4">Right Column</div>
</div>
```

- Add a navigation bar with a logo (just your name is fine)
- Add navigation links
- Add cards in a grid
- Add a jumbotron

LAB - Personal Website

Add a *bootstrap* directory to either example.com or you GitHub Pages project and redesign the UI using bootstrap.

Additional Resources

- Bootstrap

Udemy

- Learn Bootstrap 4 by Example

Next: PHP

Chapter 9: PHP

This chapter will introduce the student to basic and intermediate PHP concepts.

We will re-review programming basics with in the context of PHP. Cover object oriented programming (OOP) concepts. Integrate a user request into a thrid party API. Introduce the student to the idea of a template engine and provide a few final details that should be considered when launching a basic website.

- Programming basics with PHP
- Object oriented programming
- Third party APIs
- Template engines
- SEO and meta data
- Miscellaneous items for rounding out a website.

Next: PHP Basics

PHP Basics

PHP (the P in LAMP stack) is a popular server side scripting language. Once you have a LAMP server up an running, getting started with PHP is pretty easy.

Since this is a dev environment we want to able to debug errors. Lets tell Apache to show us errors when accessing PHP files.

```
sudo vim /etc/php/7.0/apache2/php.ini
```

Find the *display_errors* directive by typing *\display_errors =* in vim. This will be around line 426 enter insert mode and change *display_errors = Off* to *display_errors = On* then restart Apache.

> **Security Check Point**
> It is never advisable to show errors in a productions environment. This provides information to hackers that can be used to compromise your system.

Hello World

Since you can mix PHP and HTML in the same file the parser will need a way to know if it is being asked to parse PHP or HTML. We tell the parser what to expect by using PHP tags `<?php ?>` where `<?php` in the opening tag and `?>` is the closing tag. Any text outside of the

Exercise 1 - Hello World

Create the path */var/www/mtbc/php* and that as a project to Atom's side bar. Then from your IDE create the file *hello.php*. Add the following lines.

```
<?php

//Great the user with a hello message
echo "Hello World";
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
      <meta charset="UTF-8">
      <title>Contact</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
       <h1><?php echo 'Hello World'; ?></h1>
    </body>
</html>
```

```
<?php
  $msg = 'Hello World';
?>
<!DOCTYPE html>
<html lang="en">
    <head>
      <meta charset="UTF-8">
      <title>Contact</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
       <h1><?php echo $msg; ?></h1>
    </body>
</html>
```

```
<?php
  //Literal string, no processing
  $name = 'Bob Smith';
  //Varaible string, allows processing
  $msg = "Hello {$name}";
  //The following would work as above
  //$msg = "Hello $name";
  //The following is an example of contactination
  //$msg = 'Hello ' . $name;
?>
<!DOCTYPE html>
<html lang="en">
    <head>
      <meta charset="UTF-8">
      <title>Contact</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
       <h1><?php echo $msg; ?></h1>
    </body>
</html>
```

```
<?php
  $name = 'Bob Smith';
  $greeting = "Hello {$name}";
  $count=5;
  $price=3.00;
  //Note the escape character before the dollar sign
  $msg = "I see you have {$count} oranges, that will be \${$price]}";

?>
<!DOCTYPE html>
<html lang="en">
    <head>
      <meta charset="UTF-8">
      <title>Contact</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
       <h1><?php echo $greeting; ?></h1>
       <p><?php echo $msg;?></p>
    </body>
</html>
```

Mathmatical opeations

```
<?php
  $name = 'Bob Smith';
  $greeting = "Hello {$name}";
  $count=5;
  $price=.6;
  $total=($count*$price);
  $msg = "I see you have {$count} oranges, that will be \${$total}";

?>
<!DOCTYPE html>
<html lang="en">
    <head>
```

```
        <meta charset="UTF-8">
        <title>Contact</title>
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
      </head>
      <body>
         <h1><?php echo $greeting; ?></h1>
         <p><?php echo $msg;?></p>
      </body>
</html>
```

Additional Resources

Next: PHP Control Structures

PHP Control Structures

Programming is little more that reading data and piecing together statements that take action on that data. Every language will have it's own set of control structures. For most languages a given set of control structures will be almost identical. In the Bash lesson we learned a few control structures most of which exist in PHP. While the syntax may be a little different, the logic remains the same.

Exercise 3 - If, Else If, Else

Create the following path */var/www/mtbc/php/if_else.php* and open it Atom. Then add the following lines.

```
<?php

//Initialize your variables
$label = null;
$color = null;

//Check for get parameters
if(!empty($_GET)){
    $color = "#{$_GET['color']}";
}

//Can we name the color by it's hex value
if($color == "#ff0000"){
  $label = "red";
}elseif($color == "#00ff00"){
  $label = "green";
}elseif($color == "#0000ff"){
  $label = "blue";
}else{
  $label = "unknown";
}

//Output the data
echo "<div style=\"color:{$color}\">The color is {$label}</div>";
```

Now open a browser and navigate to https://localhost/php/if_else.php and you will see the message *The color is unknown*. Now add the following string to the end of the URL *?color=ff0000*. Now your message will read *The color is red* and it will be written in red font. That string you added to the end of the URL is know as a query string. A query string allows you to pass arguments into a URL. A query string consists of the query string Identifier (a question mark) *?* and a series of key to value pairs that are separated by an ampersand (*&*). In our example the the *key is color* color *and the _value is ff0000*. If you wanted to submit a query of a first and last name that might look like *?first=bob&last=smith* where first and last are your keys (aka your GET params) bob and smith are your values.

Now let's take a close look at the code. Initializing your variables is a good practice.

```
//Initialize your variables
$label = null;
$color = null;
```

In PHP `$_GET` is a superglobal so it is always available, this is NOT something you want to try to initialize. `empty()` is used to determine if a variable is truthy or falsey where falsey values equate to empty or falsey returns true. Prefixing `empty()` with an *!* `!empty()` reverses the return values. In plain English `if(!empty($_GET['color'])){` would read *if $_GET['color'] is not false then do something* or you could say *if $_GET['color'] has any value then do something*. You will see a lot of curly braces in PHP code due to it's use of C style syntax.

If $_GET['color'] has any value then the set the variable $color to the value of $_GET['color']

```
//Check for get parameters
if(!empty($_GET['color'])){ //This is a control statement
    //This is the body of the statement
    $color = "#{$_GET['color']}";
}
```

The user has submitted a hex value in the form of a get parameter. Do we know what to the call that hex value? If the answer is yes set the value of *$label* to that color. Otherwise set the value of *$label* to *Unknown*. Or you could say *if the hex value is red then say it is red; otherwise if it green then say it is green; otherwise if it blue then say it is blue; otherwise say unknown*.

```
//Can we name the color by it's hex value
if($color == "#ff0000"){
  $label = "red";
}elseif($color == "#00ff00"){
  $label = "green";
}elseif($color == "#0000ff"){
  $label = "blue";
}else{
  $label = "unknown";
}
```

Finally we will print some output back to the screen. This time we will wrap the output in some HTML and give it a little style by setting the font color to that of the user input.

```
echo "<div style=\"color:{$color}\">The color is {$label}</div>";
```

Exercise 4 - For Loop

Add the following to the path */var/www/mtbc/php/for.php*.

```
<?php

$items = array(
  'for',
  'foreach',
  'while',
  'do-while'
);

echo 'PHP Supports ' . count($items) . ' of loops.';

$li = '';
for($i=0; $i<count($items); $i++){
  $li .= "<li>{$items[$i]}</li>";
}

echo "<ul>{$li}</ul>";
```

Exercise 5 - Foreach Loop

Add the following to the path */var/www/mtbc/php/foreach.php*.

```
<?php

$items = array(
  'for',
  'foreach',
  'while',
  'do-while'
);

echo 'PHP Supports ' . count($items) . ' of loops.';

$li = '';
foreach($items as $item){
  $li .= "<li>{$item}</li>";
}

echo "<ul>{$li}</ul>";
```

Exercise 6 - While Loop

```
<?php

$items = [
  'for',
  'foreach',
  'while',
  'do-while'
];

$count = count($items);

echo "PHP Supports {$count} of loops.";

$i = 0;
$li=null;
while ($i < $count) {
  $li .= "<li>{$items[$i]}</li>";
  $i++;
}

echo "<ul>{$li}</ul>";
```

Exercise 7 - Do While Loop

Add the following to the path */var/www/mtbc/php/do_while.php*.

```
<?php

$items = [
  'for',
  'foreach',
  'while',
  'do-while'
];

echo 'PHP Supports ' . count($items) . ' of loops.';

$i = 0;
$li=null;
do {
  $li .= "<li>{$items[$i++]}</li>";
} while ($i > 0);
```

Additional Reading

- Which Loop

Next: Object Oriented Programming in PHP

Object Oriented Programming in PHP

Exercise 1 - Hello World

Create the path */var/www/mtbc/php* and that as a project to Atom's side bar. Then from Atom create the file *hello.php*. Add the following lines.

```
<?php
/**
 * Greets the user with the current date and time.
 */

//Create a date object http://php.net/manual/en/book.datetime.php from PHP's built
$date = new DateTime();

#Format the date http://php.net/manual/en/datetime.formats.date.php
$formattedDate = $date->format('Y-m-d h:i:s');

echo "Hello World it is {$formattedDate}";
```

Open a browser and navigate to *https://localhost/php/hello.php*. You will see the string *Hello it is x* where x is the current date and time.

Now lets have another look at what we wrote, don't worry if you do not get this, we will go over it all again when we dive into the programming lessons.

The first line is `<?php` , this tells the server to begin interpreting PHP. PHP tags open with `<?php` and close with `?>` if there is no closing tag it is assumed all text after the opening PHP tag is to interpreted as PHP.

Line 2 is a comment there are three types of comments in PHP.

- `# This is a comment`
- `// This is a comment`
- `/* this is a comment */`

Line 4 creates an object[1], in the case the object is an instance of the date class `$date = new DateTime();`

- `$date` - this is a variable that will hold an instance of the DateTime() class (*$date* holds the instantiated object)
- `new` - says I want to create a new copy (instantiation) of a class.
- `DateTime()` - the class to be instantiated. The parenthesis indicate an empty constructor[2].

Line 7 stores the result of an operation in a variable called `$formattedDate` .

A method is a property of an object in PHP we can access a propery of an object using an arrow `->` . `format()` is a method (a type of property) of the `DateTime()` class. Here the operation is simply formatting the current Unix timestamp to a human readable format. `$date->format('Y-m-d h:i:s')` says I want to call the *format()* method of the *$date* object with the following parameters `'Y-m-d h:i:s'` .

- `$formattedDate` - A variable that holds a formatted date string.
- `=` - Sets the value of the left to that of the right.
- `$date` - The instantiated date object.
- `->` - Object operator, this is used to request properties from an instantiated object. In most languages use dot concatenation `.` .
- `format()` - A method of the DateTime() class, a property of the instantiated object.
- `'Y-m-d h:i:s'` - A parameter for the format method. In this case we are providing a string representation of what we want

the formatted date to look like.

In the final line we are printing results to the screen.

- `echo` - Tells PHP to write something to a web page.
- `""` - Defines the string to be written, everything inside the quotes will be printed.
- `$formattedDate` - The formatted date string.
- `{}` - Not required, but makes it easier to separate variables from strings.

PHP Classes

Object Oriented Programming (OOP) has been supported by PHP (at least in some fashion) since version 4. OOP support has improved with each version. Generally speaking a class is blue print for an object. If we were to compare this to the physical world the source code contained in the class would be akin to an architects blue prints say for a house. Upon instantiation the class is used to build the object. In short, you could view a house as an instantiation of a set of blue prints. I give the blue prints to the builder (in this case a compiler) and the builder goes of and does its thing. The end result is that builders interpretation of the blue prints.

A class has properties comprised of instance variables, methods. A class may be dependent on other classes, this is known as a dependency and is often dealt with using dependency injection. A class may inherit from other classes this is known as either a parent class to a child class or super class to sub class.

The classic example of a class and class properties is to think of a person. A person class would have properties such as a head, arms, legs, etc. I prefer to think in terms of completing work as this is really what we want our classes to do. For example if I were to have a class for reading and writing to and from the database I might call it DBWorker. Now the question is what do I need DBWorker to do?

I'll need my DBWorker class to

- connect to the database
- know which table to access
- write new records to the database
- read records in from the database
- update records in the database
- delete records from the database

So far my properties would be as follows

- Instance Variable - *table*
- Method - *connect()*
- Method - *create()*
- Method - *read()*
- Method - *update()*
- Method - *delete()*

In PHP this might look like the following

```
class DBWorker
{
    private $table = null;

    public function __construct($connection, $table) {
      $this->connect($connection);
      $this->$table = $table;
    }

    private function connect($connection){//do something}
    public function create($data) {//do something}

    public function read($whereClause) {//do something}

    public function update($data, $whereClause) {//do something}

    public function delete($whereClause) {//do something}

}
```

Instantiation may look like this

```
// This would probably be in a config file somewhere.
$config = [];
$config['db'] = '{'db':sample_db', 'user':'sample_user', 'password':'123456', 'host':'localhost'}';

// Instantiate DBWorker with a given db configuration.
$dbw = new DBWorker($config['db'], 'People');

// Read from the database with given parameters.
$results = $dbw->read('{'email':'%@example.com'}');

//Process the results
foreach($results as $result){
  //do something
}
```

Exercise 2 - Hello Class

Create the path */var/www/mtbc/php/hello_class.php*.

```
<?php

/**
 * A mock up of session data
 */
class Session
{
  /**
   * Returns the current user session
   * @return array Session Data
   */
  public function read()
  {
    return ['id'=>'1234', 'name'=>'YOUR-NAME'];
  }
}

/**
 * Returns a greeting to a given user
 */
class Hello
{
  /**
   * An instance variable to hold the name of the user
   * @var string
   */
  private $who;

  /**
   * A constructor method - Constructor injection with type hinting. Constructor injection is a form of type hi
nting.
   * @param  Object $session A user session
   */
  public function __construct(Session $session) {

    $sessionData = $session->read();

    $this->setWho($sessionData['name']);
  }

  /**
   * A setter method for Hello::who
   * @param String $who - The name of a given user
   */
  public function setWho($who)
  {
    $this->who = $who;
  }

  /**
   * Returns a greeting to a target user
   * @param  {[type]} $message [description]
   * @return {[type]}          [description]
   */
  public function greet($message)
  {
    return "{$message} {$this->who}";
  }

}

//Instantiate the Session class
$session = new Session();

//Instantiate the Hello class. Inject the $session object into the constructor.
$greeting = new Hello($session);

//Provide a message for the user (Ternary Logic)
$message = 'Good ' . (date("H")<12?'Morning':(date("H")<17?'Afternoon':'Evening'));

echo $greeting->greet($message);
```

Exercise 3 - example.com

Start by making a copy of your GitHub pages project

```
cd /var/www
mkdir -p example.com/public
cp /var/www/YOUR-GITHUB-USERNAME /var/www/example.com
```

Move src, dist and all html files into the public directory.

Additional Resources

- Object Oriented PHP for Beginners
- PHP Shorthand If/Else Using Ternary Operators (?:)
- Dependency Injection in PHP

Footnotes

1

[1] PHP supports both procedural (functional) programming and object oriented programming (OOP). In OOP an instantiated class is an object. A class could be considered a blue print for an object. PHP has a built in class for working with dates called *DateTime*. PHP instantiates classes with the `new` operator. The line `$date = new DateTime();` stores an instance of the *DateTime* object in a variable called *$date*. PHP uses an arrow `->` as it's Object Operator. To access the *format* method of the *DateTime* class you would say `$date->format();`. In this case we are storing the formatted date in the variable *formattedDate*. As in Bash, PHP uses *echo* to write content to the screen. In this case, a web page.

[2] An empty constructor, constructor with no argument. In programming arguments allow us to pass data into objects, methods and functions.

Next: Form Validation

HTML Forms with PHP Validation

Traditionally, forms have been the most common way to collect data from a user. A form submission is the simplest way to post data to a server. This section will start with a simple POST request and end with complex processing.

Form tags `<form></form>` are used for creating forms in HTML. Every form should have at least two attributes *action* and *method*.

- action - the web address to which the form data will be sent.
- method - the type of request the form should make (probably GET or POST).

Exercise 1 - Create and Inspect a Contact Form

Rename *public/contact.html* to *public/contact.php* and complete the following three steps.

1. Change the action to contact.php

```
<form action="contact.php" method="POST">
```

2. Remove the lines

```
<input type="hidden" name="_next" value="https://YOUR-GITHUB-USERNAME.github.io/thanks.html">
<input type="text" name="_gotcha" style="display:none">
```

3. Change _subject to subject and _replyTo to email

```
<input type="hidden" name="_subject" value="New submission!">
...
<input id="email" type="text" name="_replyto">
```

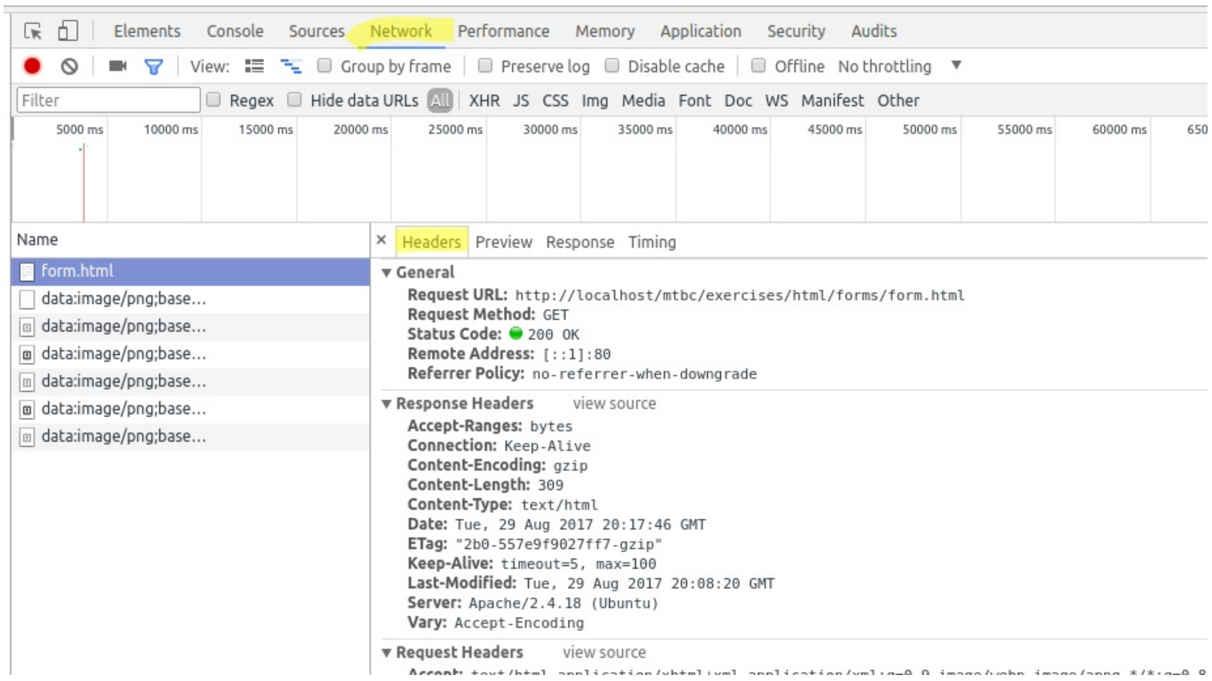
The end result will be as follows

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Contact Me - YOUR-NAME</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./dist/css/main.css" type="text/css">
  </head>
  <body>
    <header>
      <span class="logo">My Website</span>
      <a id="toggleMenu">Menu</a>
      <nav>
        <ul>
          <li><a href="index.html">Home</a></li>
          <li><a href="resume.html">Resume</a></li>
          <li><a href="contact.html">Contact</a></li>
        </ul>
      </nav>
    </header>
    <main>

      <h1>Contact Me - YOUR-NAME</h1>
      <form action="contact.php" method="POST">
```

```
          <input type="hidden" name="subject" value="New submission!">

          <div>
            <label for="name">Name</label>
            <input id="name" type="text" name="name">
          </div>

          <div>
            <label for="email">Email</label>
            <input id="email" type="text" name="email">
          </div>

          <div>
            <label for="message">Message</label>
            <textarea id="message" name="message"></textarea>
          </div>

          <div>
            <input type="submit" value="Send">
          </div>

        </form>

      </main>
      <script>
          var toggleMenu = document.getElementById('toggleMenu');
          var nav = document.querySelector('nav');
          toggleMenu.addEventListener(
            'click',
            function(){
              if(nav.style.display=='block'){
                nav.style.display='none';
              }else{
                nav.style.display='block';
              }
            }
          );
        </script>
    </body>
  </html>
```

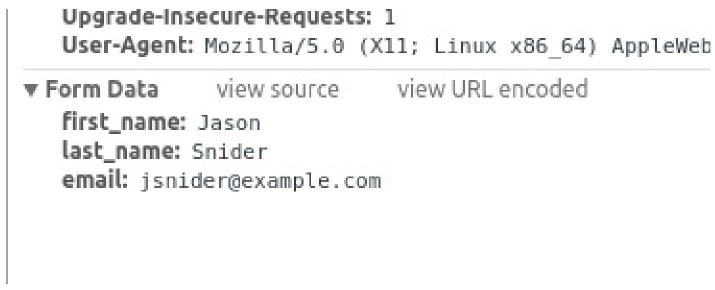
Now, lets inspect a post request.

- Open the Chrome browser and navigate to http://loc.example.com.
- Press the [f12] key to open Chrome's developer tools.
- Click on the network tab.
- Refresh the page.
- Find and click on contact.php
- The headers tab should now be highlighted.



```
▼ General
    Request URL: http://localhost/mtbc/exercises/html/forms/form.html
    Request Method: GET
    Status Code: ● 200 OK
    Remote Address: [::1]:80
    Referrer Policy: no-referrer-when-downgrade
▼ Response Headers     view source
    Accept-Ranges: bytes
    Connection: Keep-Alive
    Content-Encoding: gzip
    Content-Length: 309
    Content-Type: text/html
    Date: Tue, 29 Aug 2017 20:17:46 GMT
    ETag: "2b0-557e9f9027ff7-gzip"
    Keep-Alive: timeout=5, max=100
    Last-Modified: Tue, 29 Aug 2017 20:08:20 GMT
    Server: Apache/2.4.18 (Ubuntu)
    Vary: Accept-Encoding
▼ Request Headers     view source
```

- Fill out the web form and submit the data.
- Once again, find and click on form.html from the network panel.
- Under the headers tab you will see the contents of your web form as key to value pairs. This is how the data will be given to the server.s



```
    Upgrade-Insecure-Requests: 1
    User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWeb
▼ Form Data     view source     view URL encoded
    first_name: Jason
    last_name: Snider
    email: jsnider@example.com
```

Find the opening form tag and set the action attribute as follows. *action="contact.php"*

Add the following to the top of the document, above the DOCTYPE declaration.

```
<?php

$data = $_POST;

foreach($data as $key => $value){
  echo "{$key} = {$value}";
}
?>
```

Validation and Basic RegEx

Regular Expressions (RegEx) are strings of text that describe a search pattern. You may be familiar with wild cards in which a search for *b** would return all words that start with the letter b. Now lets say you want your wild card search to still return all words starting with the letter b but only if the word does not contain a number; this is where RegEx comes in `\b(b)+([a-z])*\b`.

- `\b` - a word boundary, the beginning of a word. This would return all words.
- `\b(b)*` - MUST start with the letter b. This would return all words starting with the letter b.

- `\b(b)+([a-z])*` - MAY also contain any lower case letters after the first letter.
- `\b(b)+([a-z])*\b` - Stops each match at the end of a word

Try It

Exercise 2 - RegEx

/var/www/example.com/public/contact.php

```
<?php
//Create a RegEx pattern to determine the validity of the use submitted email
// - allow up to two strings with dot concatenation any letter, any case any number with _- before the @
// - require @
// - allow up to two strings with dot concatenation any letter, any case any number with - after the at
// - require at least 2 letters and only letters for the domain
$validEmail = "/^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,})$/";

//Extract $_POST to a data array
$data = $_POST;

//Create an empty array to hold any error we detect
$errors = [];

foreach($data as $key => $value){
  echo "{$key} = {$value}<br><br>";

  //Use a switch statement to change your behavior based upon the form field
  switch($key){
      case 'email':
        if(preg_match($validEmail, $value)!==1){
            $errors[$key] = "Invalid email";
        }

      break;

      default:
        if(empty($value)){
            $errors[$key] = "Invalid {$key}";
        }
      break;
  }

}

var_dump($errors);
?>
```

RegEx is extremely powerful, flexible and worth learning. Having said that there are a million and one libraries for validating form submissions. I would advise finding a well supported library that meets your projects needs. As of PHP5 data filters have been natively supported by the language.

Security Check Point *Never trust user input. User input is anything come into the server from the client. Even if you have written client side JavaScript to filter out malicious code, the filtered input is still left alone with the client and can be manipulated prior to transit (or even in transit). If it has ever existed outside of the server it CANNOT be trusted.*

Exercise 3 - Adding a Validation Class

Replace the contents of */var/www/example.com/public/contact.php* with the following.

Explain the code to the class.

1. The user completes and submits a web form
2. The system
    i. Validates each field for errors
        i. If error return a message to the user
            i. Show a page level message
            ii. Show each field level mesasge
            iii. Do not clear the form data on error
        ii. If no errors, send the user to a thank you page

2.1 - Loop through each form feild and test that field for errors.

2.2 - Set truthy/falsey value or a boolean on error. This will tell the system how to rpoceed after the form is processed.

2.2.1 - See 2.2

2.2.2 - Store each error in an array using the the name of the field as the array key. The value will be the error message to display to the user. The form will call a method in the validate array using the name of the field as the argument. This will retireve any error messages for that field.

2.2.3 - Store POST data in an instance varaible using key value pairs in which the key is the name of the form field and value is the user submitted data. Set the value attribute of each form field to a method in the Validate class and pass the name of the field as the methods argument. This will retrieve the original data as submitted by the user and pre-populate that form field.

```
<?php

class Validate{

    public $validation = [];

    public $errors = [];

    private $data = [];

    public function notEmpty($value){

        if(!empty($value)){
            return true;
        }

        return false;

    }

    public function email($value){

        if(filter_var($value, FILTER_VALIDATE_EMAIL)){
            return true;
        }

        return false;

    }

    public function check($data){

        $this->data = $data;

        foreach(array_keys($this->validation) as $fieldName){

            $this->rules($fieldName);
        }

    }
    public function rules($field){
        foreach($this->validation[$field] as $rule){
            if($this->{$rule['rule']}($this->data[$field]) === false){
                $this->errors[$field] = $rule;
            }
        }
    }

    public function error($field){
        if(!empty($this->errors[$field])){
            return $this->errors[$field]['message'];
        }

        return false;
    }

    public function userInput($key){
        return (!empty($this->data[$key])?$this->data[$key]:null);
    }
}

$valid = new Validate();

$input = filter_input_array(INPUT_POST);

if(!empty($input)){

    $valid->validation = [
        'email'=>[[
                'rule'=>'email',
                'message'=>'Please enter a valid email'
            ],[
                'rule'=>'notEmpty',
                'message'=>'Please enter an email'
        ]],
        'name'=>[[
            'rule'=>'notEmpty',
            'message'=>'Please enter your first name'
        ]],
        'message'=>[[
            'rule'=>'notEmpty',
            'message'=>'Please add a message'
        ]],
    ];

    $valid->check($input);

    if(empty($valid->errors)){
        $message = "<div class=\"message-success\">Your form has been submitted!</div>";
        //header('Location: thanks.php');
    }else{
        $message = "<div class=\"message-error\">Your form has errors!</div>";
    }
}
?>

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Contact Me - YOUR-NAME</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./dist/css/main.css" type="text/css">
  </head>
  <body>

    <header>
      <span class="logo">My Website</span>
      <a id="toggleMenu">Menu</a>
```

```
        <nav>
          <ul>
            <li><a href="index.html">Home</a></li>
            <li><a href="resume.html">Resume</a></li>
            <li><a href="contact.html">Contact</a></li>
          </ul>
        </nav>
      </header>

      <main>
        <h1>Contact Me - YOUR-NAME</h1>

        <?php echo (!empty($message)?$message:null); ?>

        <form action="contact.php" method="POST">

          <input type="hidden" name="subject" value="New submission!">

          <div>
            <label for="name">Name</label>
            <input id="name" type="text" name="name" value="<?php echo $valid->userInput('name'); ?>">
            <div class="text-error"><?php echo $valid->error('name'); ?></div>
          </div>

          <div>
            <label for="email">Email</label>
            <input id="email" type="text" name="email" value="<?php echo $valid->userInput('email'); ?>">
            <div class="text-error"><?php echo $valid->error('email'); ?></div>
          </div>

          <div>
            <label for="message">Message</label>
            <textarea id="message" name="message"><?php echo $valid->userInput('message'); ?></textarea>
            <div class="text-error"><?php echo $valid->error('message'); ?></div>
          </div>

          <div>
            <input type="submit" value="Send">
          </div>

        </form>
      </main>

      <script>
          var toggleMenu = document.getElementById('toggleMenu');
          var nav = document.querySelector('nav');
          toggleMenu.addEventListener(
            'click',
            function(){
              if(nav.style.display=='block'){
                nav.style.display='none';
              }else{
                nav.style.display='block';
              }
            }
          );
        </script>
    </body>
  </html>
```

Include Files and Namespaces

So far we have a lot of code in the what would typically be considered a *view* (the presentation layer) which should be separated from from data and logic as much as possible. In this lesson we will create include and class files that will help us keep out views clean.

PHP Fig is a working group aimed at developing interoperability standards for PHP libraries. This will help us build a standardized directory structure.

- PSR 0
- PSR 4

Include Files

An include file is a file that contains a snippet of code that is referenced by another file. In PHP include files can be accessed using include, include _once, require or require_once functions along with a relative or absolute file path.

```
// relative include
include 'util.php';

// absolute include
include '/var/www/example.com/util.php';
```

Namespace

At the end of the day a name space is simply a way to disambiguate name collisions. Earlier we created a class called Validate(). Validation classes are fairly common and let's say you liked specific methods from two different vendors both of who named the class Validate, suddenly you have a collision.

Lets say we have two vendors Sally and Bob and I like Sally's email method and Bob's phone method. I want to load this class from both vendors but without a name space the autoloader would not know which class to load into a given object. I might try to include then instantiate but there is no guarantee this will work as classes tend to get cached.

```
include 'vendor/Sally/src/Validation/Validate.php';
$v1 = new Validate();
$v1->Validate->email($email); //This will probably work

include 'vendor/Bob/src/Validate/Validate.php';
$v2 = new Validate();
$v2->Validate->phone($phone); //Sally's version of the class may or may not be cached so the method we want may
 or may not be there.
```

With name spaces.

```
// You can probably use an autoloader so you will not have to worry about this.
include 'vendor/Sally/src/Validation/Validate.php';
include 'vendor/Bob/src/Validate/Validate.php';

// The namespace disambiguates class names so $v2's object will have the target class.
$v1 = new \Sally\Validation\Validate();
$v2 = new \Bob\Validate\Validate();

$v1->Validate->email($email);
$v2->Validate->phone($phone);
```

Exercise 4

Create the path */var/www/example.com/core/About/src/Validation/Validate.php* and copy the Validates class into the file. Add a name space declaration as the first line of the file.

/var/www/example.com/core/About/src/Validation/Validate.php

```
<?php
```

```
namespace About\Validation;

class Validate{

    public $validation = [];

    public $errors = [];

    private $data = [];

    public function notEmpty($value){

        if(!empty($value)){
            return true;
        }

        return false;

    }

    public function email($value){

        if(filter_var($value, FILTER_VALIDATE_EMAIL)){
            return true;
        }

        return false;

    }

    public function check($data){

        $this->data = $data;

        foreach(array_keys($this->validation) as $fieldName){

            $this->rules($fieldName);
        }

    }

    public function rules($field){
        foreach($this->validation[$field] as $rule){
            if($this->{$rule['rule']}($this->data[$field]) === false){
                $this->errors[$field] = $rule;
            }
        }

        //Make sure the array is empty if no errosrs are detected.
        if(count($this->errors) == 0){
            $this->errors = [];
        }
    }

    /**
     * Detects and returns an error message for a given field
     * @param  string $field
     * @return mixed
     */
    public function error($field){
        if(!empty($this->errors[$field])){
            return $this->errors[$field]['message'];
        }

        return false;
    }

    /**
```

```
     * Returns the user submitted value for a give key
     * @param  string $key
     * @return string
     */
    public function userInput($key){
        return (!empty($this->data[$key])?$this->data[$key]:null);
    }
}
```

Since we have pulled the validation logic into a library all we need to do in the contact form is call the class and process it.
/var/www/example.com/public/contact.php

```
<?php
//Include non-vendor files
require '../core/About/src/Validation/Validate.php';

//Declare Namespaces
use About\Validation;

//Validate Declarations
$valid = new About\Validation\Validate();

$args = [
  'first_name' => FILTER_SANITIZE_STRING,
  'last_name' => FILTER_SANITIZE_STRING,
  'email' => FILTER_SANITIZE_EMAIL,
  'subject' => FILTER_SANITIZE_EMAIL,
  'message' => FILTER_SANITIZE_EMAIL,
];
$input = filter_input_array(INPUT_POST, $args);

if(!empty($input)){

    $valid->validation = [
        'first_name'=>[[
            'rule'=>'notEmpty',
            'message'=>'Please enter your first name'
        ]],
        'last_name'=>[[
            'rule'=>'notEmpty',
            'message'=>'Please enter your last name'
        ]],
        'email'=>[[
                'rule'=>'email',
                'message'=>'Please enter a valid email'
            ],[
                'rule'=>'notEmpty',
                'message'=>'Please enter an email'
        ]],
        'subject'=>[[
            'rule'=>'notEmpty',
            'message'=>'Please enter a subject'
        ]],
        'message'=>[[
            'rule'=>'notEmpty',
            'message'=>'Please add a message'
        ]],
    ];


    $valid->check($input);
}

if(empty($valid->errors) && !empty($input)){
    $message = "<div>Success!</div>";
}else{
    $message = "<div>Error!</div>";
}

?>

<!DOCTYPE html>
...
```

Additional Resources

- PSR-0: Autoloading Standard
- PHP filter_input_array()
- PHP filter_var()
- Email RegEx Examples
- RegEx 101
- HTML Purifier
- Yahoo's HTML Purify
- Google Caja
- Sanitize HTML

Udemy

- Object Oriented Programming (OOP) in PHP - Build An OOP Site

Next: Working with APIs

API

An Application Programming Interface (API) is a means of providing programming hooks into a larger system. For example `document.getElementById('el').style` is a programming hook that allows a JavaScript programmer to work with the browser. Some APIs are open to the public and others are closed and accessible only by those who have been granted special access. Some APIs are accessible over Internet protocols.

JSON

Before we dive into the Mailgun API we should take a moment to talk about JavaScript Object Notation (JSON). This is the notation used by JavaScript for creating objects it is a string of key pairs (similar to an array) and in recent years has become the defacto standard for interacting with API. Since we are using PHP to interact with JSON we will use `json_decode()` and `json_encode()` to convert a JSON string into an array and vice versa.

Curly braces *{}* encapsulates a JSON string (creates an object in JS)

Colons *:* separate key to value pairs, fields are comma separated (aka CSV or comma separated values).

```
{
  '_id':1234,
  'firstname':'Sally',
  'lastname':'Smith'
}
```

A company with multiple addresses (in this case an array of addresses)

```
{
  '_id'':1,
  'company':'MicroTrain Technologies',
  'addresses':[
    {
      'street':'200 W Adams Suite 410',
      'city':'Chicago',
      'state':'IL',
      'zip':60606
    },
    {
      'street':'720 E Butterfield Rd Suite 100',
      'city':'Lombard',
      'state':'IL',
      'zip':60148
    },
  ]
}
```

One JSON string with multiple company objects

```
{
  {
    '_id'':1,
    'company':'42 North Group, Inc,
    'addresses':[
      {
        'street':'720 E Butterfield Rd Suite 100',
        'city':'Lombard',
        'state':'IL',
        'zip':60148
      },
    ]
  },
  {
    '_id'':2,
    'company':'TheProfessional.Me',
    'addresses':[
      {
        'street':'200 W Adams Suite 410',
        'city':'Chicago',
        'state':'IL',
        'zip':60606
      },
      {
        'street':'720 E Butterfield Rd Suite 100',
        'city':'Lombard',
        'state':'IL',
        'zip':60148
      },
    ]
  }
}
```

In any case PHP will convert this to an array and you would parse out the array to find your desired value. So the first example would appear as.

```
array(
  '_id' => 1234,
  'firstname' => 'Sally',
  'lastname' => 'Smith'
)
```

Let's this was returned via a fictitious *User->get(1234)* method (in which User is a class and get() is a member (property/method) of that class) you might access *firstname* as follows.

```
$users = new User();
$user = $users->get(1234);

//Writes Sally to the screen
echo $user['firstname'];
```

The Mailgun SDK abstracts most of the JSON interaction away from us, so we are left working mostly with PHP objects. JSON will come up again so this seems a good time introduce it.

Mailgun API

Exercise - Getting Started with Mailgun.

Go to mailgun.com and create a free account. You may choose not to add a credit card as we will be working with the sandbox. You will need to activate your account via email and SMS, this requires a phone number that receive texts. If you do not have one try creating a Google Voice account.

The landing page will provide you the details you need need to send a test email. Start by creating a shell script and pasting the curl example into the shell.

```
cd ~
vim mailgun.sh
```

Add the foloowing lines

```
#!/bin/bash
curl -s --user 'api:YOUR_API_KEY' \
    https://api.mailgun.net/v3/YOUR_DOMAIN_NAME/messages \
    -F from='Excited User <mailgun@YOUR_DOMAIN_NAME>' \
    -F to=YOU@YOUR_DOMAIN_NAME \
    -F subject='Hello' \
    -F text='Testing some Mailgun awesomness!'
```

Next

- Make the file executable.
- Execute the script.

```
chmod +x mailgun.sh
./mailgun.sh
```

If you see a json response similar to this, your sand box account is working.

```
{
  "id": "<20171009154755.125901.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.mailgun.org>",
  "message": "Queued. Thank you."
}
```

Exercise 2 - Mailgun and PHP

The first thing you will want to do is Mailgun's PHP library to your project. You will do this via Composer.

Let's start by creating a feature branch. This will allow us to keep any changes separate from our production code until we are ready for it.

```
cd /var/www/about
git checkout -B feature/mailgun
```

If you get the message *Switched to a new branch 'feature/mailgun'* you are ready to proceed.

```
composer require mailgun/mailgun-php php-http/curl-client guzzlehttp/psr7
```

This may take a minute or so. The install was successful if you see a series of *Installing* messages ending with

```
Writing lock file
Generating autoload files
```

Now, let's see what was installed.

```
git status
```

You should see something like the following.

```
composer.json
composer.lock
vendor/
```

We do not want to commit the vendor directory to our repo. So we will create a *.gitignore* file. From you Atom sidebar create a file called *.gitignore* under the about project (Do not forget the preceding dot) and add the following line.

```
/vendor
```

Now if you run `git status` you will see the following.

```
.gitignore
composer.json
composer.lock
```

Commit these files and push to the new feature branch.

```
git add .
git commit -m 'Added mailgun lib'
git push origin feature/mailgun
```

Create the file */var/www/example.com/test.php* and copy and paste the PHP sample code from the Mailgun landing page. Below the pasted code add the line `var_dump($reults);` .

```
<?php
# Include the Autoloader (see "Libraries" for install instructions)
require 'vendor/autoload.php';
use Mailgun\Mailgun;

# Instantiate the client.
$mgClient = new Mailgun('key-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx');
$domain = "sandboxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.mailgun.org";

# Make the call to the client.
$result = $mgClient->sendMessage("$domain",
          array('from'    => 'Mailgun Sandbox <postmaster@sandboxdxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.mailgun.org>'
,
                'to'      => 'YOUR-NAME <YOUR-EMAIL-ADDRESS>';
                'subject' => 'Hello YOUR-NAME',
                'text'    => 'Congratulations YOUR-NAME, you just sent an email with Mailgun! You are truly awe
some! '));

var_dump($reults);
```

From a browser window, navigate to http://localhost/YOUR-PROJECT-NAME/test.php and you should get a json string similar to: ``php object(stdClass)#24 (2) { ["http_response_body"]=> object(stdClass)#19 (2) { ["id"]=> string(91) "<20171009164718.79178.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.mailgun.org>" ["message"]=> string(18) "Queued. Thank you." } ["http_response_code"]=> int(200) }

~~~
Then check your email to see if it worked.


**Security Check Point**
_Never push a key to a public repository, use a key file the exists outside of the public repo_

## Exercise - Pass Variables into the API Call

Create a key file
```sh
mkdir -p /var/www/example.com/config
vim /var/www/example.com/config/keys.php
~~~

Add the following line to your .gitignore file and commit your changes to the *feature/mailgun* branch.

```
/config/keys.php
```

Add the following (Where YOUR-KEY-HERE is the key provided by Mailgun):

```
<?php
define('MG_KEY', 'dYOUR-KEY-HERE');
define('MG_DOMAIN', 'YOUR-DOMAIN-HERE');
```

Add the following to the top of *http://localhost/YOUR-PROJECT-NAME/test.php*

```
require '../config/keys.php';
```

Change

```
$mgClient = new Mailgun('key-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx');
$domain = 'sandboxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.mailgun.org';

$from = 'Mailgun Sandbox <postmaster@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.mailgun.org>';
```

to

```
$mgClient = new Mailgun(MG_KEY);
$domain = MG_DOMAIN;

$from = "Mailgun Sandbox <postmaster@{$domain}>";
```

Instead of passing static values into the method pass variables.

- Replace the existing method call with the following.
- Remove the var_dump($results);

```
$from = "Mailgun Sandbox <postmaster@{$domain}>";
$to = 'YOUR-NAME <YOUR-EMAIL-ADDRESS>';
$subject = 'Hello YOUR-NAME';
$text = 'Congratulations YOUR-NAME, you just sent an email with Mailgun! You are truly awesome!';

$result = $mgClient->sendMessage(
  $domain,
  array('from'    => $from,
        'to'      => $to,
        'subject' => $subject,
        'text'    => $text
    )
);
```

Update your README file.

Navigation to *README.md* from your ATOM sidebar.

```

# example.com

An about me web site.

## Dependencies
```

~~~
* PHP 7.0 or higher
* Composer
* An API key for Mailgun

## Installation

Clone this project into a web directory
```sh
cd /var/www
git clone git@github.com:jasonsnider/example.com.git
~~~

Install composer dependencies

```
cd /var/www/example.com
composer install
```

Additions Resources

- JSON
- MDN

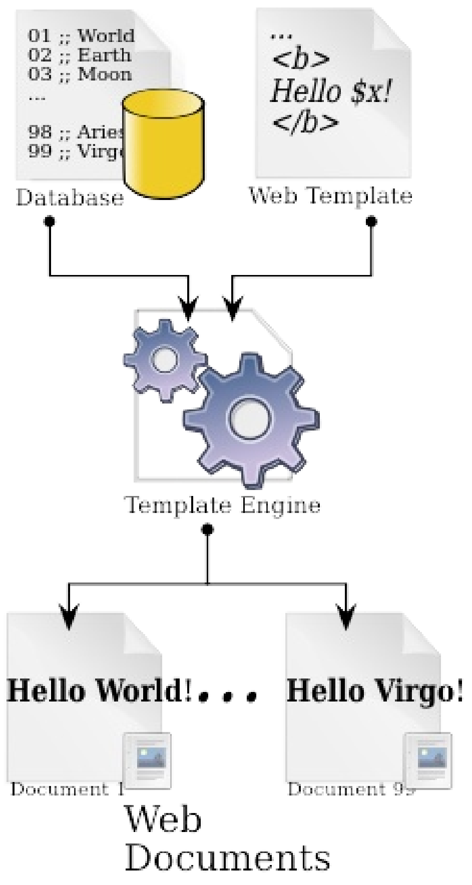
Udemy

- JSON and APIs

Next: Heredoc

Templates

Now that we have all of our styles into a single style sheet, it is time use a single template for managing our layout. This will allow us to store out sites main layout in a single file while the page itself will only hold the content specific to that page. This is a huge win once your site exceeds more that a few pages. HTML does not offer such a system so we will either need to choose a template engine such as Pug or build a template using a programming language. We will learn about Pug once we get into the section about the MEAN stack. For now we will use PHP.



About: TempEngWeb016.svg

The general flow is as follows.

1. The user asks the server for an endpoint perhaps *https://localhost/example.com/public/contact.php*.
2. The server processes data and business logic as required by that endpoint.
   i. This may include calls from a database, a flat file, an API, etc.
3. The server generates the content to be presented to the user for that end point.
   i. This may be from a database, a flat file, an API, etc.
4. The server calls the template for that endpoint.
5. The server passes the content into the template.
6. The endpoint presents the data to the user. This may be in the form of an HTML document, a PDF, XML/JSON file or whatever else may be required.

Heredoc

A heredoc allows large amount of text to be written as a string without the need of escaping. Heredoc will process any PHP code it encounters, which makes it a subtitle candidate for our simple use case. I should note this a very simple example of a template system and is intended simply as an introduction to the concept. We will discuss more complex examples in later lessons.

In previous lessons we used `echo` to write the value of some variables into an HTML document. This is a good way to view a template system. This is something that allows you to have a single or a select few templates that treat the contents in which the content of a page is being passed as a variable onto the template.

The basic idea is as follows. We are simply passing the variable `$content` into an HTML document using PHP's echo statement.

```
<?php $content="<h1>Hello World</h1><p>Welcome to my web page.</p>"; ?>

<html>
  <head></head>
  <body><?php echo $content; ?></body>
</html>
```

The previous example works for simple strings, but what about a page that has tens if not hundreds of lines of HTML? Writing all of that as a PHP string is tedious and prone to error, this is where the heredoc syntax can help.

```
<?php
$who = 'World';
$content = <<<EOT
<h1>Hello {$who}</h1>
<p>Welcome to my web site.</p>
EOT;
?>
<html>
  <head></head>
  <body><?php echo $content; ?></body>
</html>
```

A heredoc string start by declaring a variable and setting a delimiter. PHP uses `<<<` to declare a delimiter. In our case `<<<EOT` (End of Text) is our delimiter. When a delimiter is encountered followed by a semicolon with no spaces around it the string is terminated.

The previous example passes the content into an HTML string but it still requires the entire document to be embedded in every page. Let's push the HTML document into an include file.

contact.php

```
<?php
$who = 'World';
$content = <<<EOT
<h1>Hello {$who}</h1>
<p>Welcome to my web site.</p>
EOT;

require 'layout.php';
```

layout.php

```
<html>
  <head></head>
  <body><?php echo $content; ?></body>
</html>
```

Now we only have to include the layout on each page of our site. The advantage is we will only need to change a single file to cascade Look and Feel changes across the entire website.

Exercise 1

Create the path */var/www/example.com/core/layout.php* and add the following lines.

```
<!DOCTYPE html>
<html lang="en">
  <head>
      <meta charset="UTF-8">
      <title>About Jason Snider</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <link rel="stylesheet" type="text/css" href="css/dist/main.css">
  </head>
  <body>

    <div id="Wrapper">
        <nav class="top-nav">
            <a href="index.html" class="pull-left" href="/">Site Logo</a>
            <ul role="navigation">
                <li><a href="index.php">Home</a></li>
                <li><a href="resume.php">Resume</a></li>
                <li><a href="contact.php">Contact</a></li>
            </ul>
        </nav>

        <div class="row">
            <div id="Content">
                <?php echo $content; ?>
            </div>
            <div id="Sidebar">
              <div id="AboutMe">
                <div class="header">Hello, I am YOUR-NAME</div>
                <img src="https://www.gravatar.com/avatar/4678a33bf44c38e54a58745033b4d5c6?d=mm" alt="My Avatar
" class="img-circle">
              </div>
            </div>
        </div>

        <div id="Footer" class="clearfix">
            <small>&copy; 2017 - MyAwesomeSite.com</small>
            <ul role="navigation">
                <li><a href="terms.html">Terms</a></li>
                <li><a href="privacy.html">Privacy</a></li>
            </ul>
        </div>
    </div>

  </body>

</html>
```

Change the path */var/www/example.com/public/contact.php* to the following. Pay special attenention to the echo statements, these are now treated as variables being passed into a string such that `<?php echo $valid->userInput('first_name'); ?>` becomes `{$valid->userInput('first_name')}` .

```
<?php

require 'core/processContactForm.php';

$content = <<<EOT
<form method="post" action="contact.php">
  {$message}
  <div>
    <label for="firstName">First Name</label><br>
    <input type="text" name="first_name" id="firstName" value="{$valid->userInput('first_name')}">
    <div class="text-error">{$valid->error('first_name')}</div>
  </div>

  <div>
    <label for="lastName" id="lastName">Last Name</label><br>
    <input type="text" name="last_name" value="{$valid->userInput('last_name')}">
    <div class="text-error">{$valid->error('last_name')}</div>
  </div>

  <div>
    <label for="email" id="email">Email</label><br>
    <input type="text" name="email" value="{$valid->userInput('email')}">
    <div class="text-error">{$valid->error('email')}</div>
  </div>

  <div>
    <label for="subject" id="subject">Subject</label><br>
    <input type="text" name="subject" value="{$valid->userInput('subject')}">
    <div class="text-error">{$valid->error('subject')}</div>
  </div>

  <div>
    <label for="message" id="message">Message</label><br>
    <textarea name="message">{$valid->userInput('message')}</textarea>
    <div class="text-error">{$valid->error('message')}</div>
  </div>


  <input type="submit">

</form>
EOT;

require 'core/layout.php';
```

Navigate to *https://loc.example.com/public/contact.php* and submit the form. The functionality should not have changed in anyway but the code is now cleaner and easier to maintain this is know as a refactoring.

Additional Resources

- Web template system
- Output Buffering Control

Udemy

- The Complete MySQL Developer Course

Next: Meta Data

Since meta tags belong in the document we will want to add them to the template. We will need to be able to set these on a page by page basis so these should be passed as a variable from the page in question.

```
$meta = [];
$meta['description'] = "The best thing about hello world is the greeting";
$meta['keywords'] = "hello world, hello, world";

<meta name="description" content="<?php echo $meta['description']; ?>">
<meta name="keywords" content="<?php echo $meta['keywords']; ?>">
```

For example *contact.php* might read as follows. Rather than an array, perhaps a herdoc containing any meta tags you want to include would be preferred.

```
<?php

require 'core/processContactForm.php';

//Build the page metadata
$meta = [];
$meta['description'] = "The best thing about hello world is the greeting";
$meta['keywords'] = "hello world, hello, world";

//Build the page content
$content = <<<EOT
<form method="post" action="contact.php">
  {$message} ...
EOT;

require 'core/layout.php';
```

Next: MySQL

Chapter 10: MySQL

Next: SQL

SQL

Structure Query Language (SQL) is language designed specifically for working with databases. SQL is an open standard maintained by the American National Standards Institute (ANSI). Despite being an actively maintained open standard you'll most likely alot of time working with vendor specific variants. These differences are typically subtle and quick Google search will typically get you around them. For instance if you're used to working with Oracle's `TO_DATE()` and now you're working with MySQL; `TO_DATE()` will not work. Google somthing like *TO_DATE in mysql* and one of the first results will liekly point to MySQL's `STR_TO_DATE()` method. Point being, learning any vendor variant of SQL will be enough to allow you to work with just about any vendor variant. The same Google tricks can even be used for NoSQL databases which we will learn about later. Knowledge of SQL is desired in many industries outside of tech as it is considered to be the *English* of the data world. Understanding SQL helps you understand other data paradigms such as *noSQL*.

Database

Not to be confused with a Database Management System (DBMS) is a container. This may be a file or a set files, it really doesn't matter as you will likely never see the data. You will be working exclusively with the DBMS. The DBMS is the software that works with the database. For example products such as Oracle, MySQL, SQL Server and even MongoDB are all database management systems. The first three are also known as a Relation Database management System (RDBMS) while the latter is a NoSQL DBMS.

Schema

Schema is defined as *a representation of a plan or theory in the form of an outline or model*. Unfortunately, the terms database and schema are often used interchangeably. For our purposes *schema* will be used to describe a database. This may be a visualization or the actual SQL file that contains the build commands for the database.

Tables

A table is a structured list of data typically designed to represent a collection like items. These collections may be users, customers, products, etc. Naming your tables, especially in larger project can prove difficult; defining a style guide and sticking to it can save you some headaches in the long run. Once common rule is for all tables to end in a plural form of the entity it represents.

Columns

A column is often refereed to as a field. A column has several attributes a name, data type, default value and indices are typically what you'll be concerned with. The data type itself may have additional parameters such a length.

Column names should be short and two the point. While style guides vary typical rules state a column name should differ from that of the table and must not over lap with any reserved words.

Data types cast restraints on a column. For example a column of type integer (INT) would not allow any non-numeric characters. A VARCAHR(10) would allow any combination of characters but will truncate the string after 10 characters.

Default values define what is to be entered if no value is present. This may be null or not null (which will force a value) or any other value compatible with the data type.

Indicies are used to improve performance. These take specific columns from a table or tables and stores them in a way that allows them to be looked up quickly with out needing to reanalyze all of the content of all tables involved in the query. I have seen good good indexing literally cut minutes of page load times. While not required all tables should have a primary key.

A primary key is unique index for a row of data. The most common primary key is a simple id. This can be an auto-incrementing number, a universally unique identifier (UUID) or any other bit of unique data such an email address.

> **Security Checkpoint**
> I have seen a number of systems that ask for sensitive data such as social security or employer identification numbers simply because that is the only way they could think of not to risk duplicate data. Think twice before doing this, if your business does not absolutely require this sort of information do not even think about storing it. If it is required, please consult a security professional.

Additional Resources

- NoSQL Not Only SQL
- Naming Conventions: Stack Overflow Discussion
- Integer Types

Udemy

- SQL for Beginners

Next: MySQL

MySQL

MySQL is a free and open source relational database management system (RDBMS) currently under the control of Oracle. An RDBMS will store data a container called a table. This data is organized by rows and columns similar to a spreadsheet (aka tabular data). Relationships are joined by creating foreign key relationships between rows of data across multiple tables. MySQL uses Structured Query Language (SQL) to retrieve data from tables.

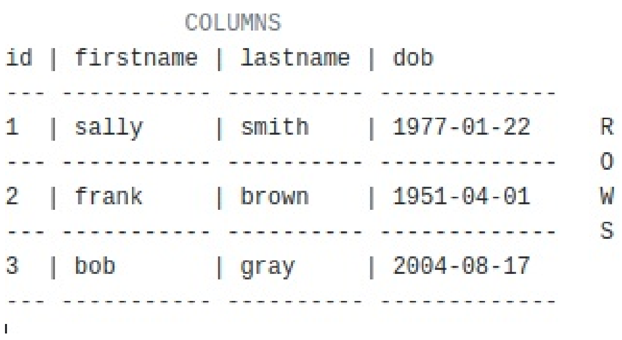
Much like SQL Server, Oracle or PostgreSQL, MySQL is client-server software. This means the database and database management software is running on a server. Even if your running a local instance of MySQL, your running a server. The client is the software you interact with. When you interact with the software, the software makes a request to the server on your behalf. Some common clients are the MySQL CLI, phpMyAdmin and MySQL Workbench. Even programming languages act as clients; for example PHP uses the PDO library to interact with a DBMS. ORMs are another type of client package; an example of this would be Doctrine for PHP.

Core Concepts of an RDBMS

Table Structure

A data base is made up of tables which are defined by columns, rows, data types and values. Conceptually you can think of a table in the same way you think of a spreadsheet.

My customers table might look like the following.



```
                COLUMNS
id | firstname | lastname | dob
--- ----------- ---------- -------------
1  | sally     | smith    | 1977-01-22     R
--- ----------- ---------- -------------    O
2  | frank     | brown    | 1951-04-01     W
--- ----------- ---------- -------------    S
3  | bob       | gray     | 2004-08-17
--- ----------- ---------- -------------
```

Keys and Indicies

Index

An index provides a faster means of lookup for an SQL query. A good rule of thumb is to create an index for every column that appears in a WHERE clause. You should not create an index for a column that is not used for looking up data.

Primary Key (Unique Index)

A primary key in MySQL is both an index and a unique identifier for a given column. Typcially an aut-incrementing integer or a UUID. In the above example the id column would be my primary key.

Forgien Key (Index)

This is a column that links one table to another. For example if I have an address table that I want to link to my customers table I would add a column called customer_id and the value of this column would match an id in the customers table. In this case customer_id is said to be a forgien_key.

Unique Keys (Unique Index)

This is an indexed column that requires a unique value for every row in the table.

Composite Keys (Unique Index)

This is an indexed key pair that can uniquely identify a row in the table.

FullText Index (Unique Index)

This is a special type of index that allows various configurations string searches against rows containing large amounts of text.

Relationships

One-to-one

Table A may (or must) have one and only one corresponding rows in table B.

One-to-many

Table A may (or must) have one or more corresponding rows in table B.

Many-to-many

Table A may (or must) have one or more corresponding rows in table B and table B may (or must) have one or more corresponding rows in table A. A many-to-many relationship must be resolved by an associative entity. An associative entity is table that links two or more tables togeather using foreign keys.

Additional Resources

- PDO Library
- MySQL for NodeJS
- MySQL Client Programs
- ORM Is an Offensive Anti-Pattern
- To ORM or Not to ORM
- ORM Hate

Udemy

- The Complete MySQL Developer Course

Next: Working with MySQL

Working with MySQL

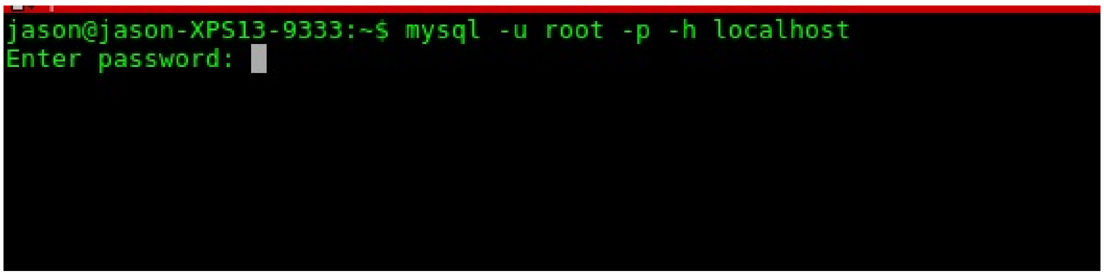
Making a Connection

Before you can work with MySQL you need to make a connection; we will start with the CLI. If you followed the setup instructions in section 01-04 you should have a local instance of the DBMS running with the credentials of root:password and this will be running on localhost

```
mysql -u root -p -h localhost
```

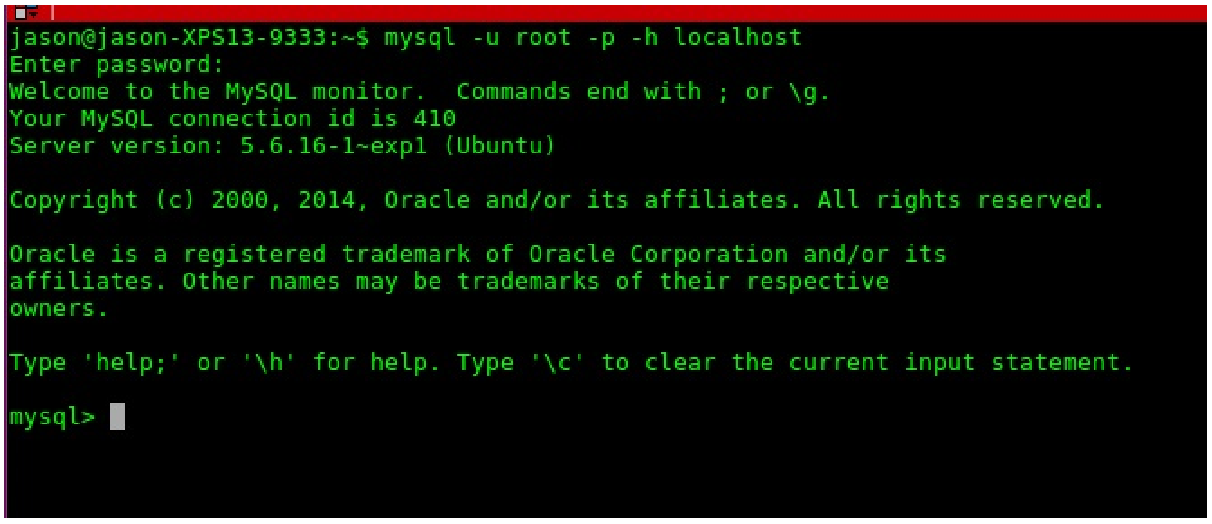
- `mysql` - invokes the MySQL CLI
- `-u root` - *-u* is the user argument *-u root* says login to MySQL as the user root
- `-p` - *-p* is the password argument. You do not want the enter the password here, so this tells the client to prompt you for a password.
- `-h` - *-h* host; locally this will be localhost a remote connection would likely be an IP address or a URL.

As you may have guessed; by entering the aforementioned command you will be prompted a password.



```
jason@jason-XPS13-9333:~$ mysql -u root -p -h localhost
Enter password: 
```

Upon connection you will be presented with a MySQL prompt.



```
jason@jason-XPS13-9333:~$ mysql -u root -p -h localhost
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 410
Server version: 5.6.16-1~exp1 (Ubuntu)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

At the prompt the following command to get a list of databases.

```
show databases;
```

You will see something similar to the following. By default MySQL installs a few system databases that it uses for it's own purposes. It is unlikely you ever need to access these yourselves. Just be sure not to delete them.

```
+---------------------+
| Database             |
+----------------------+
| information_schema   |
| mysql                |
| performance_schema   |
+----------------------+
3 rows in set (0.03 sec)
```

Now let's create a databases.

```
CREATE DATABASE bootcamp;
```

You should get something similar to the following response. What you are really looking for is an affected row count.

```
Query OK, 1 row affected (0.00 sec)
```

Now verify the database has been created. Run the following command and you will see *bootcamp* in the list of databases.

```
SHOW DATABASES;
```

Now lets use our new database.

```
USE bootcamp;
SHOW TABLES;
```

show tables; shows a list of tables from the current database. After running this command the DBMS should respond with *Empty set (0.00 sec)*.

Let's say we were going to build our own blogging software in the form of a web application. We will start by gathering some requirements.

- As a blogger I need to be able to create blog posts.
- As a web site owner I need to assure that only authorized people can post to the blog.

From a data perspective the first item can be achieved with a table named *users* the second with a table named *posts*.

Tables are created using the *CREATE TABLE* command. `CREATE TABLE _tablename_()` where *tablename* is the name of the table to be created. The columns in your table are passed into the parentheses as comma separated values.

Add the following table to your bootcamp database.

```
CREATE TABLE users (
    id VARCHAR(36) PRIMARY KEY COMMENT 'Primary Key UUID',
    first_name VARCHAR(40) DEFAULT NULL COMMENT 'The users first name',
    last_name VARCHAR(40) DEFAULT NULL COMMENT 'The users last name',
    email VARCHAR(200) DEFAULT NULL COMMENT 'A unique identifier for a user',
    created DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT 'When the post was created',
    modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT 'When the post was last edi
ted'
) ENGINE=INNODB;
```

We already talked about *CREATE TABLE* now lets look at the columns definitions. *id VARCHAR(36) PRIMARY KEY COMMENT 'Primary Key UUID'*

- id - This is the name of the column. Most tables will have an id column.
- VARCHAR(36) - This allows any character on the keyboard but caps the length at 36 characters. This is usually indicative of a UUID or GUID.

- PRIMARY KEY - Every table SHOULD have a primary id. This serves as a unique index for that table.
- COMMENT 'Primary Key UUID' - A short description about the purpose of this column. In practice I wouldn't comment obvious fields.

Lets jump to the modified column *modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP*

- modified - The name of the column.
- DATETIME - Sets the datatype to a mysql time stamp *YYYY-MM-DD HH:MM:SS*
- DEFAULT CURRENT_TIMESTAMP - When a new row is created the modified column will default to the time of creation.
- ON UPDATE CURRENT_TIMESTAMP - When a new row is created the modified column will change to the time of update.

Now let's create the table for holding our blog posts.

```
CREATE TABLE posts (
    id VARCHAR(36) PRIMARY KEY COMMENT 'Primary Key UUID',
    title VARCHAR(255) COMMENT 'The title of the blog post',
    slug VARCHAR(255) COMMENT 'A human and SEO friendly lookup key',
    keywords VARCHAR(255) COMMENT 'Meta data for SEO',
    description VARCHAR(255) COMMENT 'Meta data for SEO',
    body TEXT COMMENT 'The content of the blog post',
    user_id VARCHAR(36) COMMENT 'The creator of the blog post',
    created DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT 'When the post was created',
    modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT 'When the post was last edi
ted',
) ENGINE=INNODB;
```

Now let's take a look at our data structure.

```
USE bootcamp;
SHOW TABLES;

SHOW COLUMNS FROM users;
DESCRIBE posts;
```

Now that we have created a couple of tables. Let's add some data. Run the following command from your bootcamp database replacing xxxx with you information.

```
INSERT INTO users SET id=UUID(), first_name='xxxx', last_name='xxxx', email='xxxx';
```

Now let's look up your user record.

```
SELECT * FROM users WHERE email='xxxx';
```

What if only want a list of names?

NOTE: As queries get longer this style may be easier to read.

```
SELECT
  first_name,
  last_name
FROM
  users
WHERE
  email='xxxx';
```

Id rather see the user's name in a single column formatted as *last, first*

```
SELECT
  CONCAT('last_name', ', ', first_name) AS user
FROM
  users
WHERE
  email='xxxx';
```

Add another user.

```
INSERT INTO
  users
SET
  id=UUID(),
  first_name='Bob',
  last_name='Smith',
  email='bsmith@exampl.com
```

Let's find all users with a *.com* and sort in ascending order by last name.

```
SELECT
  CONCAT('last_name', ' ', first_name) AS user
FROM
  users
WHERE
  email LIKE '%.com'
ORDER BY last_name ASC
```

```
INSERT INTO
  posts
SET
  id=UUID(),
  slug='hello',
  title='Hello',
  user_id = '1a804677-7953-11e7-8397-180373ae98cc';

SELECT
  posts.title,
  CONCAT(first_name, ' ', last_name) AS author
FROM
  posts,
  users
WHERE
  posts.created_user_id = users.id ;
```

For full authentication you will want to add password and salt columns

- password VARCHAR(60) COMMENT 'A salted hash of the password'
- salt VARCHAR(128) COMMENT 'User specific salt'

```
ALTER TABLE
  ADD
    password VARCHAR(60) COMMENT 'A salted hash of the password',
    salt VARCHAR(128) COMMENT 'User specific salt';
```

> **Security Checkpoint**
> Never store a password in plain text, always store a hashed version of the password. Always create a user specific salt this will protect against ranibow table attacks.

Additional Resources

- Safe Password Hashing
- Rainbow Tables

Next: DataModels

Data Models

https://en.wikipedia.org/wiki/First_normal_form Customer data with one or many phone numbers stored in a single row.

customers

id	first_name	last_name	phone
123	John	Smith	(555) 861-2025, (555) 122-1111
456	Jane	Doe	(555) 403-1659, (555) 929-2929
789	Cindy	Who	(555) 808-9633

```
SELECT phone FROM customers WHERE id = 123
```

1 result (555) 861-2025, (555) 122-1111

Next: MVC with CakePHP

Chapter 11: MVC with CakePHP

Next: MVC

MVC

MVC is a software design patter that splits your application into three distinct layers; the data layer (Model), the logic layer (Controller), and the presentation layer (View).

Model

A model can be anything that provides data such a table in a database, a reference to an API, a spreed sheet, etc. For our user a model will reference a table in a database.

View

View's are the presentation layer. Views will typically be HTML but can be anything a client can read such as .html, .json, .xml, .pdf or even a header that only a machine can access.

Controller

Business and application logic.

Next: CakePHP

CakePHP

CakePHP is and MVC (Model, View, Controller) based rapid application development (RAD) framework built using PHP. CakePHP has a solid eco-system and is designed around test driven development (TDD).

CRUD

Create, Read, Update, Delete

Migrations

Rolling snapshots of the database structure. These allow you migrate your database's strucutre forward and backwards across snapshots.

Create a Repository on GitHub

- Create the repository *cake.example.com*.
- **DO NOT** Initialize with a README
- Add the MIT License

</> code Initial Commit

Installation

First make sure you have installed internationalization functions for PHP.

```
sudo apt-get install php-intl
```

Create a CakePHP project via composer. Sticking with the *example.com* nomenclature we will call this one *cake.example.com*.

```
cd /var/www
composer create-project --prefer-dist cakephp/app cake.example.com
```

Answer yes to the following

```
Set Folder Permissions ? (Default to Y) [Y,n]?
```

Your First App

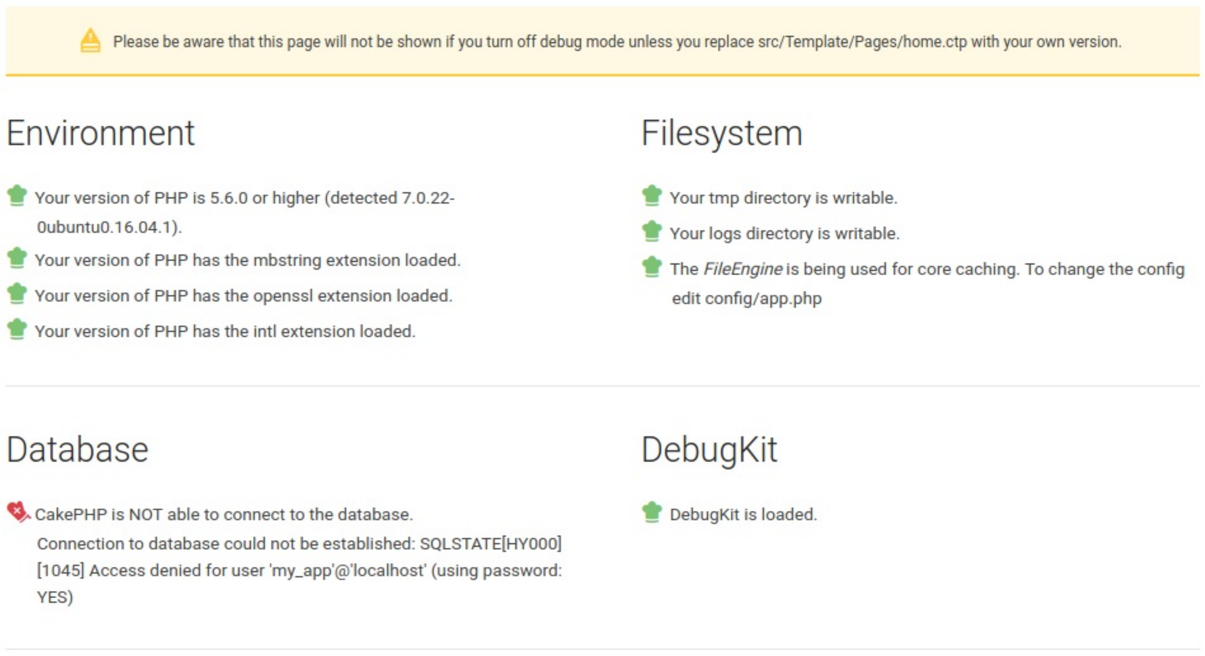
Move into the new project folder

```
cd cake.example.com
```

Spin up a development web server.

```
bin/cake server
```

In a browser go to http://localhost:8765/. You will be presented a default home page that shows that gives you plenty of resources to help you learn CakePHP and it will return a system status that makes sure you system is set up correctly. Everything should be green except for the database.



Please be aware that this page will not be shown if you turn off debug mode unless you replace src/Template/Pages/home.ctp with your own version.

Environment

Your version of PHP is 5.6.0 or higher (detected 7.0.22-0ubuntu0.16.04.1).
Your version of PHP has the mbstring extension loaded.
Your version of PHP has the openssl extension loaded.
Your version of PHP has the intl extension loaded.

Filesystem

Your tmp directory is writable.
Your logs directory is writable.
The *FileEngine* is being used for core caching. To change the config edit config/app.php

Database

CakePHP is NOT able to connect to the database.
Connection to database could not be established: SQLSTATE[HY000] [1045] Access denied for user 'my_app'@'localhost' (using password: YES)

DebugKit

DebugKit is loaded.

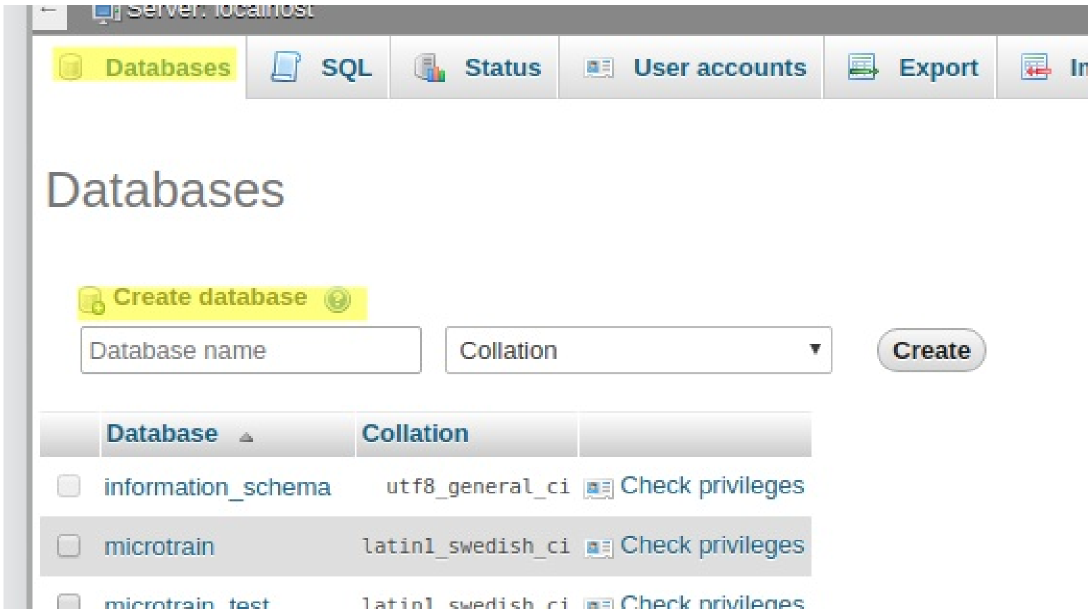
Add *cake.example.com* as a project in Atom. Navigate to *config/app.php* this is the default configuration file for your application. CakePHP stores it configuration as an array, find the *Datasources* attribute somewhere arounf the the line *220* (you can use the shortcut [ctrl] + [g] and enter *220*). You will notice two child attributes *default* and *test*. *default* holds the configuration for your application's database while test holds the configuration for running unit tests.

Setup Your Database

We will use PhpMyAdmin to create two databases

- cake_app
- cake_test

Go to http://localhost/phpmyadmin and login as with *root:password*. Find the Databases tab and under the *Create database* header enter *cake_app* as your first database. This will now ask you to create a table, skip this step and find your way back to the Databases tab and create another database called *cake_test* you can now close out of phpMyAdmin and return to the *app.php* file in Atom.



	Database	Collation	
☐	information_schema	utf8_general_ci	Check privileges
☐	microtrain	latin1_swedish_ci	Check privileges
☐	microtrain_test	latin1_swedish_ci	Check privileges

Update the database configurations as follows.

default

```
'username' => 'root',
'password' => 'password',
'database' => 'cake_app',
```

test

```
'username' => 'root',
'password' => 'password',
'database' => 'cake_test',
```

Return to http://localhost:8765/](http://localhost:8765/) and refresh the page, all settings should now be green.

Configure Apache

Let's set up an Apache configuration with a local hosts entry for development purposes.

```
sudo vim /etc/apache2/sites-available/cake.example.com.conf
```

```
<VirtualHost 127.0.0.101:80>

        ServerName loc.cake.example.com

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/cake.example.com/webroot

        # Allow an .htaccess file to ser site directives.
        <Directory /var/www/cake.example.com/webroot/>
                AllowOverride All
        </Directory>

        # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
        # error, crit, alert, emerg.
        # It is also possible to configure the loglevel for particular
        # modules, e.g.
        #LogLevel info ssl:warn
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Add the following to */etc/hosts*

```
127.0.0.101      loc.cake.example.com
```

and finally load the new site.

```
sudo a2ensite cake.example.com && sudo service apache2 restart
```

Navigate to http://loc.cake.example.com/ and you should encounter write permission issues. Resolve this by changing ownership of these files to the Apache process (www-data) and Yourself (replace jason with your Ubuntu username).

```
sudo chown www-data:jason logs
sudo chown www-data:jason logs/*
sudo chown www-data:jason tmp
sudo chown www-data:jason tmp/*
sudo chown www-data:jason tmp/*/*
```

Return to http://loc.cake.example.com/ and all systems should now be a go.

Merge with the Exisiting GitHub Project

</> Code Run the following commands, be sure to replace YOUR-GITHUB-USERNAME with your github username.

```
cd /var/www/cake.example.com
git init
git remote add origin git@github.com:YOUR-GITHUB-USERNAME/cake.example.com.git
git pull origin master
git add .
git commit -am 'Initial build'
```

Cake File Structure, Callbacks and Routing

@todo navigate to src, explain the directory structure Model, Views and Controller @todo callback methods and lifcycles as it pertains a CakePHP and Programming in general.

Blog Tutorial

We will start by using Composer to install CakeDC's User Authentication plugin. We will then build an Articles CRUD based on CakePHP's CMS tutorial. We will then tie Users to Articles (a blog post). Our lad will apply everything we just learned to building a comment system for our blogging platform. In you console, please navigate to **/var/www/cake.example.com**, this tutorial assumes **/var/www/cake.example.com** as the base path for all cd, file and folder creation commands.

Users

1. We will install the users plugin developed by CakeDC. The documentation is available here.
2. Install the core by running.

```
composer require cakedc/users
```

1. Add the following line to the end of *config/bootstrap.php*, this bootstraps the plugin to application start up.

   ```
   Plugin::load('CakeDC/Users', ['routes' => true, 'bootstrap' => true]);
   ```

2. Use the migrations plugin to install the required tables.

```
bin/cake migrations migrate -p CakeDC/Users
```

1. </> code Commit your changes with a message of *Added CakeDC's user plugin*.

1. Remove deprication warnings from line 168 of *config/app.php*

   ```
   'errorLevel'=>'E_ALL & ~E_USER_DEPRECATED',
   ```

2. Set the mail transport to debug on line 196 of *config/app.php*

   ```
   'className' => 'Debug',
   ```

3. Navigate to http://loc.cake.example.com/users/users/login and have a look around. Use the navigation links to find the registration page and create an account.

4. Notice you will not be able to login, this is because you have not yet clicked the autorization link out of your email. THe local server cannot send emails so you will have manually flip that switch in the database.

5. Login to PhpMyAdmin, find your user record in the cake_app database and flip the active and superuser flags to 1.

6. Now return to the login page and try to login. On success you will be redirected to the CakePHP debuggin page.

Add the Database Tables

- Login to phpMyAdmin https://localhost/phpmyadmin
- Click into cake > cake_app from the side bar
- Click on the SQL tab
- Copy and Paste the following the text area and hit submit
- Repeat this process for cake > cake_test

```
-- First, create our articles table:
CREATE TABLE articles (
    id CHAR(36) NOT NULL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    slug VARCHAR(191) NOT NULL,
    body TEXT,
    published BOOLEAN DEFAULT FALSE,
    created DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT 'When the post was created',
    modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT 'When the post was last edi
ted',
    UNIQUE KEY (slug)
) ENGINE=INNODB;


-- Then insert some articles for testing:
INSERT INTO articles (id,title,slug,body)
    VALUES ('6f814dc0-4adb-11e8-842f-0ed5f89f718b', 'The Title', 'the-title', 'This is the article body.');
INSERT INTO articles (id,title,slug,body)
```

```
    VALUES ('6f8155ae-4adb-11e8-842f-0ed5f89f718b', 'Hello World', 'hello-world', 'This is the article body aga
in.');
INSERT INTO articles (id,title,slug,body)
    VALUES ('6f815964-4adb-11e8-842f-0ed5f89f718b', 'Hello World Again', 'hello-world-again', 'This is the arti
cle body again and again.');
```

</> Code Since we have the table in our database we can automate the build by *baking* the model. Run the following command and take note of what files get created. In addition to creating an Entity and a Table classes, fixtures and tests will be created, thie will provide a placeholder for building unit tests.

```
bin/cake bake model Articles
```

Navigate to the *src/Model* and check out the files in *Entity* and *Article* folders.

Unit Test

</> Code Start by installing PHP unit

```
composer require --dev phpunit/phpunit:"^5.7|^6.0"
```

Run a unit test for each table

```
vendor/bin/phpunit tests/TestCase/Model/Table/ArticlesTableTest
```

Navigate to the *tests/TestCase/Model/Table* walk through the default test cases.

Managing Articles

</> code Create the file */src/Controller/ArticlesController.php* and add the following.

```
<?php
namespace App\Controller;

class ArticlesController extends AppController
{

}
```

Add an Index Method

</> code The index method will return a paginated list articles and send them a the view.

```
public function index()
{
    $this->loadComponent('Paginator');
    $articles = $this->Paginator->paginate($this->Articles->find());
    $this->set(compact('articles'));
}
```

Add the View

Create the file */src/Template/Articles/index.ctp*

```
<h1>Articles</h1>
```

```
<table>
    <tr>
        <th>Title</th>
        <th>Created</th>
    </tr>

    <?php foreach ($articles as $article): ?>
    <tr>
        <td>
            <?php echo $this->Html->link($article->title, ['action' => 'view', $article->id]); ?>
        </td>
        <td>
            <?php echo $article->created; ?>
        </td>
    </tr>
    <?php endforeach; ?>
</table>
```

Add a View Method

</> code

```
public function view($slug = null)
{
    $article = $this->Articles->findBySlug($slug)->firstOrFail();
    $this->set(compact('article'));
}
```

Add the View

Create the file */src/Template/Articles/view.ctp*

```
<h1><?php echo $article->title; ?></h1>
<div>Created: <?php echo $article->created; ?></div>
<div><?php echo $article->body; ?></div>
<div><?php echo $this->Html->link('Edit', ['action' => 'edit', $article->slug]); ?></div>
```

Add an Initialize Method

</> code Since the Flash Component will be reused elsewhere we will add an initialization method and add load our components there.

```
public function initialize()
{
    parent::initialize();

    $this->loadComponent('Flash');
    $this->loadComponent('Paginator');
}
```

Add an Create Method

</> code

```
public function create()
{
    $article = $this->Articles->newEntity();
    if ($this->request->is('post')) {
        $article = $this->Articles->patchEntity($article, $this->request->getData());

        $article->slug = Text::slug(
            strtolower(
                substr($article->title, 0, 191)
            )
        );

        if ($this->Articles->save($article)) {
            $this->Flash->success('Your article has been created.');
            return $this->redirect(['action' => 'index']);
        }

        $this->Flash->error('An error has occured.');
    }
    $this->set('article', $article);
}
```

Add the View

Create the file */src/Template/Articles/create.ctp*

```
<h1>Create an Article</h1>
<?php
    echo $this->Form->create($article);

    echo $this->Form->control('title');
    echo $this->Form->control('body', ['rows' => '5']);
    echo $this->Form->button('Save Article');
    echo $this->Form->end();
```

Add an Edit Method

</>code

```
public function edit($id = null)
{
    $article = $this->Articles->findById($id)->firstOrFail();
    if ($this->request->is(['post', 'put'])) {
        $this->Articles->patchEntity($article, $this->request->getData());

        $article->slug = Text::slug(
            strtolower(
                substr($article->title, 0, 191)
            )
        );

        if ($this->Articles->save($article)) {
            $this->Flash->success('Your article has been updated.');
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error('An error has occured.');
    }

    $this->set('article', $article);
}
```

Add the View

Create the file */src/Template/Articles/edit.ctp*

```
<h1>Edit: <?php echo $article->title; ?></h1>
```

```
<?php
echo $this->Form->create($article);
echo $this->Form->control('id', ['type'=>'hidden']);
echo $this->Form->control('title');
echo $this->Form->control('body', ['rows' => '5']);
echo $this->Form->button('Update Article');
echo $this->Form->end();
```

Add a Delete Method

</> code

```
public function delete($id = null)
{
    $this->request->allowMethod(['post', 'delete']);

    $article = $this->Articles->findById($id)->firstOrFail();
    if ($this->Articles->delete($article)) {
        $this->Flash->success("The article: {$article->title} has been deleted.");
        return $this->redirect(['action' => 'index']);
    }
}
```

Add a delete link to view.ctp

Add a delete link to the view template */src/Template/Articles/view.ctp*

```
<div>
    <?php
        echo $this->Html->link(
            'Edit',
            ['action' => 'edit', $article->id]
        );
        echo '&nbsp;|&nbsp;';
        echo $this->Form->postLink(
            'Delete',
            ['action' => 'delete', $article->id],
            ['confirm' => __('Are you sure, you want to delete {0}?', $article->title)]
        );
    ?>
</div>
```

A Create, Edit and Delete Links to the Index View

</> code

Create Slugs in the Background

</> code We will move our slug creation logic from the Articles controller to the Articles table. Here we will use a callback to create slugs in the background. We will create a unit test to verifiy our slug creation logic.

Call the namespace for the Utility class. *src/Model/Table/ArticlesTable.php*

```
// the Text class
use Cake\Utility\Text;
```

Create the method for creating a slug. *src/Model/Table/ArticlesTable.php*

```
public function createSlug($title)
```

```
{
    return Text::slug(
        strtolower(
            substr($title, 0, 191)
        )
    );
}
```

Add and run a unit test for the create slug method. *tests/TestCase/Model/Table/ArticlesTableTest.php*

```
public function testCreateSlug()
{
    $result = $this->Articles->createSlug('Hello World');
    $this->assertEquals('hello-world', $result);
    $result = $this->Articles->createSlug('Hello!, World');
    $this->assertEquals('hello-world', $result);
    $result = $this->Articles->createSlug('Hello   World*$');
    $this->assertEquals('hello-world', $result);
    $result = $this->Articles->createSlug('Hello-   World-');
    $this->assertEquals('hello-world', $result);
}
```

Rerun the unit test

```
vendor/bin/phpunit tests/TestCase/Model/Table/ArticlesTableTest
```

Call `createSlug()` from the `beforeMarshal()` callback. *src/Model/Table/ArticlesTable.php*

```
public function beforeMarshal($event, $data)
{
    if (!isset($data['slug']) && !empty($data['title'])) {
        $data['slug'] = $this->createSlug($data['title']);
    }
}
```

Test before marshal by creating a new record *tests/TestCase/Model/Table/ArticlesTableTest.php*

```
public function  testBeforeMarshal()
{
    $article = $this->Articles->newEntity();
    $article = $this->Articles->patchEntity($article, ['title'=>'Hello World, It\'s a fine day']);
    $this->Articles->save($article);

    $result = $this->Articles->find()->first();
    $this->assertEquals('hello-world-it-s-a-fine-day', $result['slug']);
}
```

Rerun the unit test

```
vendor/bin/phpunit tests/TestCase/Model/Table/ArticlesTableTest
```

Add User Authentication

</> code Add an instance varaible to hold the session object, set this varaible during initialization. The following reads as set this instance of `$session` to the session object from this instance of the request object. This will provide a short hand for accessing the session data in controllers.

src/Controller/AppController.php

```
    protected $session;

    public function initialize()
    {
        ...
        $this->session = $this->getRequest()->getSession();
    }
```

Deny Aceess by Default

</> code *src/Controller/AppController.php*

```
public function initialize()
{
    parent::initialize();

    // Load the AUTH component
    $this->loadComponent('Auth');

    $this->loadComponent('RequestHandler');
    $this->loadComponent('Flash');
    $this->session = $this->getRequest()->getSession();

    // Deny unauthorized access by default
    $this->Auth->deny();
}
```

At this point when you try to access http://loc.cake.example.com the application will try to reridect you to *users/login*. This will error out due to us (a) not having a UsersController at the top level and (b) not having a route that dirercts us to the CakeDC plugin. We will resolve this by adding a route.

</> code Add the following at line 80 of *src/config/routes.php*

```
Router::connect(
    '/users/login',
    [
        'plugin' => 'CakeDC/Users',
        'controller' => 'Users',
        'action' => 'login'
    ]
);
```

Now acessing http://loc.cake.example.com will redirect you to a login page.

Tie Users to Articles

Add a column called user_id to the articles table.

```
-- Add the user_id column
ALTER TABLE articles ADD user_id CHAR(36) AFTER id NOT NULL DEFAULT 0;

 -- Swap out the user_id with YOUR user id from the database
UPDATE articles SET user_id='xxxx'
```

Configuration

Navigate to http://loc.cake.example.com/users/users and create a user account.

Labs

Lab 1 - Composer

Using the documentation for the users plugin add the ability to login using a social media platform of your choice.

Lab 2 - Comment System

1. Create a table
    - id - the primary key of the comment system
    - article_id - the id of the article for which the comment is being made
    - first_name - the first name of the reader making a comment
    - last_name - the last name of the reader making a comment
    - email - the email of the reader making a comment
    - comment - the readers comment
    - created - current time stamp at the time of submission
2. At the bottom of each article provide a form that will collect the above data and on submit
    - Save the data to the comments table
    - Use the MailGun API to send your self an email telling you someone has commented on your article.

Lab 3 - Contact Form

1. Create a contact form.
2. When a user submits the form, save the contents to a database.
3. When the user submits the form, use the MailGun API to send your self an email every time someone submits the contact form.

Additional Resources

- CakePHP

Next: MongoDB

Chapter 12: MongoDB

MongoDB is a distributed noSQL noSchema document database designed for scalability, high availability, and high performance.

Next: Working with MongoDB

Working with MongoDB

MongoDB is a distributed noSQL noSchema document database designed for scalability, high availability, and high performance.

Install MongoDB

Install MongoDB on Ubuntu 16.04

By default the Ubuntu package manager does not know about the MongoDB repository so you'll need to add it to your system. First add MongoDB's private key to the package manager. Then, update the repository list. Finally, reload the package database.

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5

echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.6 multiverse" | sudo te
e /etc/apt/sources.list.d/mongodb-org-3.6.list

sudo apt-get update
```

Now you can install the latest stable release of MongoDB.

```
sudo apt-get install -y mongodb-org
```

Working with MongoDB

Start the database.

```
sudo service mongod start
```

Access the database by typing `mongo` at the command line.

```
mongo
```

Create a database called cms.

```
use cms
```

Show a list of all databases.

```
show Databases
```

You will not see your new database listed until you insert at least on document into it. This will create a collection called users.

```
db.users.insert({"email":"youremail@example.com","firstname":"YourFirstName","lastname":"YourLastName"})
```

Let's break that last command down.

db this calls the database object.

insert() is a method of the collection object, this inserts a record into a target collection. The JSON string contains the key/value pairs that will be inserted into the document. Documents in MongoDB are JSON strings.

users this is the collection upon which collection methods are called. If a collection does not exists, calling insert against the collection will create the collection.

Now if run you `sh show databases` you will see *cms* in the list. Next, you will want to look up your list of collections.

```
show collections
```

Now we will create some new documents in the users collection.

```
db.users.insert({"email":"sally@example.com","firstname":"Sally","lastname":"Smith"})
db.users.insert({"email":"bob@example.com","firstname":"Bob","lastname":"Smith"})
```

Now we will return all of the documents in the database by calling the find method (will no arguments) against the users collection of the database.

```
db.users.find()
```

We can look up specific documents by building a JSON string.

```
db.users.find({"email":"sally@example.com"})
```

RegEx allows you to search partial strings like every user who's last name ends in *th*.

```
db.users.find({"lastname":/.*th/})
```

Update a document by calling *update()* with the update modifier *$set* against a target collection and passing in the *_id* object. Passing the same JSON string into the save method of a collection would completely replace the document.

Update only, using the update modifier.

```
db.users.update({ "_id" : ObjectId(PASTE TARGET ID HERE)}, {$set:{"email":"new@email.com"}})
```

Full document replacement by ommiting the update modifier

```
db.users.update({ "_id" : ObjectId(PASTE TARGET ID HERE)}, {"email":"new@email.com"})
```

The same JSON string you build for looking up documents can be used for deletion by passing that string into the remove method of collection.

```
db.users.find()
db.users.remove({"email":"bob@example.com"})
db.users.find()
```

Use *drop()* to remove an entire collection;

```
show collections
use cms
db.users.drop()
show collections
```

You can drop the entire database by running *dropDatabase()* directly against the *db* object.

```
use cms
db.dropDatabase()
```

Atlas

1. Create an Atlas Account
2. Explain the IP Addresses

Compass

1. Install Compass

Additional Resources

- Why You Should Never Use MongoDB - Read the article then dig into the comments.

Next: NodeJS with Express

Chapter 13: ExpressJS

Express is a web application framework for NodeJS.

Next: NodeJS

NodeJS

NodeJS is a system level run time environment for JavsScript. This allows JavaScript to be executed outside of a browser.

Next: Express

Express

Express is a web application framework for NodeJS.

Getting Started with Express

Create an ExpressJS based Project

Go to GitHub and create a new project *mean.example.com*. You can change this to your own domain name if you have one *mean.YOUR-DOMAIN*.

Install the Express Generator

```
sudo npm install express-generator -g
```

Create the repository

mean.example.com

A MEAN Stack build for my personal website. Provides a CMS and user management via a REST API.

[x] Public

[x] Initialize this repository with a README

Add gitignore: None Add a license: MIT License

Clone the project into your home directory. Replace YOUR-GITHUB-ACCOUNT with the actual account details (for the sale of this lesson we will refer to this as *mean.exxample.com*).

```
cd ~/
git clone git@github.com:YOUR-GITHUB-ACCOUNT/mean.example.com.git
```

Create an Express based website, we will use PUG as our template engine.

Setup ExpressJS with Pug

```
cd mean.example.com
npm install express --save
express --view=pug
npm install
npm start
```

Split your Terminator window and run `git status`

Running `git status` at this point will show the following.

```
$ git status
app.js
bin/
node_modules/
package-lock.json
package.json
public/
routes/
views/
```

@todo - Explain routing and the express file structure.

- app.js
- bin/
- node_modules/
- package.json
- package-lock.json
  - package.json vs package-lock.json
  - Everything You Wanted To Know About package-lock.json But Were Too Afraid To Ask
- public/
- routes/
- views/

The *node_modules* directory contains third party software that should not be a part of our repository. Npm will handle these dependencies so we do not need them in our repo. Open Atom and add *mean.example.com* as a project folder.

</code> Add a .gitignore file to exclude *node_modules* from future commits.

Commit the *.gitignore* file.

```
git add .gitignore
git commit -m 'Ignore node_modules from future commits' .gitignore
```

Commit the remaining files created during setup and push both commits to the master branch.

```
git add .
git commit -am 'Setup ExpressJS with Pug'
git push origin master
```

Navigate to Then http://localhost:3000/ to access your new app.

Exercise - Express Basics

In this exercise we will make a few basic changes to familiarize ourselves with the basics of ExpressJS. If something breaks don't worry; our last commit created a rollback point. At the end of this exercise we will stash out changes and roll back to our last commit. From Atom open *mean.example.com/routes/index.js* and change the following

```
//change
{ title: 'Express'}
//to
{ title: 'Express', name: 'YOUR-NAME'}
```

From Atom navigate to *mean.example.com/views/index.pug* and change the following. Please note the indentation, indentation matters when it comes to Pug.

```
//change
extends layout

block content
  h1= title
  p Welcome to #{title}
//to
extends layout

block content
```

```
  h1= title
  p Welcome to #{title}

  p My name is #{name}
```

From the same terminal in which your NodeJS server is running press [ctrl+c] then type `sh npm start` this will restart the server. Navigate to http://localhost:3000/ and refresh the page. You will now see a tag line which states "My name is *YOUR-NAME*"

Let's break this down a bit. In the routing file *mean.example.com/routes/index.js* you define your application's end points (aka actions) by calling an HTTP request request type (typically get or post) in the form of a router method against the router object. The first argument of this method names the end point, the second argument is a call back function that defines the server side functionality for the action.

```
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express', name: 'YOUR-NAME'});
});
```

So far we we have been dealing with the sites home page. This is handled as a get request via `router.get()`. The first argument denotes the desired end point and the last argument is a call back function. The callback function tells the server how to deal with the request. The call back will always have three arguments *req*, *res*, and *next*.

- req - shorthand for *request*, this is the request object. This represents the request the user is making of the server. This will contain user supplied data in the for of *req.params* and *req.body* and provides a number of other useful properties.
- res - shorthand for *response*, this is the response object. This represents the response the server is building for the user. This will tell the server which views to render, the type of response headers to include and provides a number of other useful properties.
- next - *next()* tells the current request to move to the next piece of middle ware.

Take note of the line `res.render('index', { title: 'Express', name: 'YOUR_NAME'});`. The `render()` method is a member of the of response object. As it's name suggests, `render()` renders a view and serves it to the user. The first argument is a view directory path. In our project the views directory is located at *mean.example.com/views* making the path to index *mean.example.com/views/index*; express knows where to find the views directory so all we need to call is the path relative to views. For example, `render('users/index')` would have the path *mean.example.com/views/users/index*. The second argument is a json object. This object holds a list of key to value pairs that get passed into the view.

The view is a *.pug* file. Pug is a template engine designed for NodeJS. Pug files are processed server side and the resulting HTML is sent to the user. Calling `res.render('index', { title: 'Express', name: 'YOUR-NAME'});` pass two variables `title` and `name` into the view files *mean.example.com/views/index.pug* and build the HTML document that will be served to the user. A pug can handle variables in one of two ways.

```
//An element in which the innerHTML consists of only the variable
h1= title

//An element in which the the variable is injected into a string
p Welcome to #{title} how do you do?
```

index.pug

```
//The layout into which this page will be injected
extends layout

//creates a content variable, every thing below this will be injected into the layout
block content
  // Pug flavored markup
  h1= title
  p Welcome to #{title}
```

layout.pug

```
//standard markup
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    //the content block of index.pug will be injected here.
    block content
```

Open a browser and navigate to http://localhost:3000/. You'll see an unstyled web page the say "respond with a resource". This because this endpoint is not calling `res.render()` rather it is calling `res.send()`. The `send()` method makes no attempt to render a view, this method simply prints any text passed into it to the screen. This is similar to `res.json()` in that no view is rendered.

```
router.get('/', function(req, res) {
  res.send('respond with a resource');
});
```

Now that we have a general feel for our ExpressJS application. Let's stash our changes and start building our application.

```
git add .
git stash
git stash clear
```

Create a Data API

An API end point (in this case a data API) is a server response that does not render an HTML view. Rather, the end point would return json, jsonp, xml, soap or the like. Typically, the point of an API is to either accept a data request or return a server response in the form of unformatted data. The intent is to create an end point that can interact with other software rather than a human. For example, */api/users/create* could allow a user to be created from a web app, desktop app, mobile app, console app, etc. The point is having a single end point that can serve data between to any medium. Our applications will use HTTP verbs (GET, PUT, POST, DELETE) to create a CRUD based REST API that interacts with JSON data.

- Relevant HTTP Verbs - POST, GET, PUT, DELETE
- CRUD - Create, Read, Update, Delete
- REST - REpresentational State Transfer

Translate HTTP Verbs to CRUD

- Create - POST
- Read - GET
- Update - PUT
- Delete - DELETE

Create a Data Model

We will start with a JSON based users API. This will provide end points for creating and managing users. This is also known as a data API, since this deals with data; the data model is a logical place to start.

Create the Database

Open MongoDB

```
sudo service mongod start
mongo
```

From the MongoDB shell (be sure you see a `>`) create a database called *mean-cms*. Then create a test document by inserting running an insert against the users collection.

```
use mean-cms
db.users.insert({email: 'test@example.com', username: 'testuser'})
```

Add a Config File

</> code A config file is a good practice for storing and managing API keys and other configuration variables from a central location. I often create two files, one for production and one for development.

- Create the configuration file *config.dev.js*

config.dev.js

```
var config = {};
config.mongodb = 'mongodb://localhost/mean-cms';
module.exports = config;
```

- Call the configuration file from *app.js*

app.js

```
var config = require('./config.dev');

//Test the file
console.log(config);
```

Define the Schema

Next we will create a *models* directory in the root of our project and in it a file named *user.js* resulting is *mean.example.com/models/user.js*.

```
npm install mongoose
git commit -am 'Install Mongoose'
git push origin master
```

Validation is a key feature of Mongoose. Executes before making a save to MongoDB. This allows us to define rules at the model level. We will want to make sure usernames and email addresses are unique. Mongoose does not have a built in validator for this but the community does. We will install the mongoose-unique-validator plugin and set uniqueness as needed.

```
npm install mongoose-unique-validator
git commit -am 'Install mongoose-unique-validator'
git push origin master
```

```
var mongoose = require('mongoose'),
  Schema = mongoose.Schema,
  uniqueValidator = require('mongoose-unique-validator');

//Create a schema
```

```
var User = new Schema({
  email: {
    type: String,
    required: [true, 'Please enter an email'],
    unique: [true, 'Email must be unique']
  },
  username: {
    type: String,
    required: [true, 'Please enter a username'],
    unique: [true, 'Usernames must be unique']
  },
  first_name: String,
  last_name: String,
  admin: {
    type: Boolean,
    default: false
  }
});

User.plugin(uniqueValidator);

module.exports  = mongoose.model('User', User);
```

Implement the REST/CRUD Functionality

- create - A POST request that creates a user
- read - A GET request that returns either a single user record
- read - A GET request that returns a list of user records
- update - A PUT request that updates an existing record
- delete - A delete request that removes an existing user

GET/Read All

</> code GET/Read all users

```
curl -H "Content-Type: application/json" -X GET http://localhost:3000/api/users/
```

GET/Read One

</> code GET/Read a single user

```
curl -H "Content-Type: application/json" -X GET http://localhost:3000/api/users/5a763b67a5d70c115d81536a
```

POST/Create

</> code CREATE/Create a user

Test with a simple curl request

```
curl -d '{"email":"test2@example.com", "username":"testuser2", "first_name": "Bob", "last_name": "smith"}' -H "
Content-Type: application/json" -X POST http://localhost:3000/api/users

curl -d '{"email":"test3@example.com", "username":"testuser3", "first_name": "Sally", "last_name": "Smith"}' -H
"Content-Type: application/json" -X POST http://localhost:3000/api/users
```

PUT/Update

</> code PUT/Update a user

```
curl -d '{"_id":"5a77536ad7e4c37d6f792716", "first_name":"Robert"}' -H "Content-Type: application/json" -X PUT
http://localhost:3000/api/users
```

DELETE/Delete

</code> DELETE/Delete a user

```
curl -H "Content-Type: application/json" -X DELETE http://localhost:3000/api/users/5a766d52cbe495128293baef
```

Add Created and Modified Dates

</code> Add and auto-populate created and modified dates on document creation.

Auto Update the Modified Date When a Document is Saved

</code> Auto update the modifed timestamp prior to saving the document.

User Authentication with Passport

Passport Local Strategy

</> code Install all the packages needed for building a passport session and storing it in the database. The commit points to the package.json files, you can install these packages over the commandline or

```
npm install passport
npm install passport-local
npm install passport-local-mongoose
npm install express-session
npm install connect-mongo
```

</> code Require session packages *app.js*

```
var session = require('express-session');

//Add this after the Mongo connection
var MongoStore = require('connect-mongo')(session);
```

</> code Add configuration objects for sessions and cookies *config.dev.js*

```
//Session configuration object
config.session = {};

//Cookie configuration object
config.cookie = {};

//Create a renadom string to sign the session data
//Bigger is better, more entropy is better
//The is OK for dev, change for production
config.session.secret = '7j&1tH!cr4F*1U';

//Define the domain for which this cookie is to be set
config.cookie.domain = 'localhost:3000';
```

</code> Configure the express session

```
var passport = require('passport');
```

```
...

app.use(require('express-session')({
  //Define the session store
  store: new MongoStore({
    mongooseConnection: mongoose.connection
  }),
  //Set the secret
  secret: config.session.secret,
  resave: false,
  saveUninitialized: false,
  cookie: {
    path: '/',
    domain: config.cookie.domain,
    //httpOnly: true,
    //secure: true,
    maxAge: 1000 * 60 * 24 // 24 hours
  }
}));
app.use(passport.initialize());
app.use(passport.session());
```

</> codeSerialize the session data

```
passport.serializeUser(function(user, done){
  done(null,{
    id: user._id,
    username: user.username,
    email: user.email,
    first_name: user.first_name,
    last_name: user.last_name
  });
});

passport.deserializeUser(function(user, done){
  done(null, user);
});
```

User Registration

Passport Local Mongooose

</> code Update the user model to require hash and salt as strings.

models/user.js

```
    hash: {
        type: String,
        required: [
            true,
            'There was a problem creating your password'
        ]
    },
    salt: {
        type: String,
        required: [
            true,
            'There was a problem creating your password'
        ]
    },
```

</> code Add passport-local-mongoose

models/user.js

```
var passportLocalMongoose = require('passport-local-mongoose');

...

User.plugin(passportLocalMongoose);
```

</>code Require and use Passport Local Strategy as defined in the user model

```
var LocalStrategy = require('passport-local').Strategy;

//Require models
var User = require('./models/user');

...
//Use LocalStrategy as defined in the user model
passport.use(User.createStrategy());
```

Clean Up

- </> code Improve Comments
- </> code Improve Variable Names

Registration Endpoint

</> code Add a users route with an end point for registering a user.

Create the file *routes/users.js*. This will need access to express, passport and the user model.

```
var express = require('express');
var router = express.Router();
var passport = require('passport');
var User = require('../models/user');

router.get('/register', function(req, res, next) {
  res.render('users/register', {
    title: 'Create an Account'
   });
});

module.exports = router;
```

</> code Add the new user route to *app.js*

```
var usersRouter = require('./routes/users');

...


app.use('/users', usersRouter);
```

Registration View

</> code Create a users view directory with a registration view.

views/users/register.pug

```
extends ../layout

block content
```

```
  h1 Create an Account
  form(method='post' action='/api/users/register')
    div
      label(for='username') Username
      input(type='text' name='username' id='username')
    div
      label(for='email') Email
      input(type='text' name='email' id='email')
    div
      label(for='first_name') First Name
      input(type='text' name='first_name' id='first_name')
    div
      label(for='last_name') Last Name
      input(type='text' name='last_name' id='last_name')
    div
      label(for='password') Password
      input(type='password' name='password' id='password')
    div
      input(type='submit' value='submit')
```

</> code Add some simple fomr style to *public/stylesheets/style.css*

```
label {
  display: block;
  font-weight: bold;
}

input[type="text"],
input[type="password"],
textarea{
  font: 24px monospace;
  padding: .75rem 1.25rem;
  margin: .5rem 0 1rem;
  display: inline-block;
  border: 1px solid #ddd;
  box-sizing: border-box;
  width: 100%;
  border-radius: 2px;
}

input[type="submit"],
button,
a.button{
  padding: .75rem 1.25rem;
  font: 24px monospace;
  text-decoration: none;
  appearance: button;
  color: #fff;
  border: 1px solid #0055ee;
  background: #fff;
  cursor: pointer;
  background: #0099ee;
  border-radius: 2px;
}
```

Post the Registration Form to the Users API

</> code Add a registration end point the the users API. */users/register* is a GET request that will load a registration form. */api/users/register* is a POST request that creates a user record complete with salt and has values. For now, registartion will end with duping a JSON string onto the screen. Later we can convert this to an AJAX application.

Add the following to *routes/api/users*

```
//Register a new user
router.post('/register', function(req,res,next){
    var data = req.body;

    User.register(new User({
      username: data.username,
      email: data.email,
      first_name: data.first_name,
      last_name: data.last_name
    }),
    data.password,
    function(err, user){

      if(err){

        return res.json({
          success: false,
          user: req.body,
          errors: err
        });

      }

      return res.json({
        success: true,
        user: user
      });

    });

});
```

User Login/Logout

</> code Create a GET and POST end points for login

routes/users.js

```
router.get('/login', function(req, res){
  res.render('users/login');
});

router.post('/login', passport.authenticate('local'), function(req, res){
  res.redirect('/users');
});
```

</> code Create a login form

views/users/login.pug

```
extends ../layout

block content
  form(method='post' action='/users/login')
    div
      label(for='username') Username
      input(type='text' name='username' id='username')
    div
      label(for='password') Password
      input(type='password' name='password' id='password')
    div
      input(type='submit' value='Login')
```

</> code Add an index end point for users

routes/users.js

```
router.get('/', function(req, res){
  res.render('users/index');
});
```

</> code Add a view for the users index.

views/users/index.pug

```
extends ../layout

block content
  h1 Users Management
```

</> code Create a GET end point for logout

routes/users.js

```
router.get('/logout', function(req, res){
  req.logout();
  res.redirect('/users/login');
});
```

Authenticated Whitelist

```
//Session based access control
app.use(function(req,res,next){
  //return next();

  var whitelist = [
    '/',
    '/favicon.ico',
    '/users/login'
  ];

  if(whitelist.indexOf(req.url) !== -1){
    return next();
  }

  //Allow access to dynamic end points
  var subs = [
    '/stylesheets/',
  ];

  for(var sub of subs){
    if(req.url.substring(0, sub.length)===sub){
      return next();
    }
  }

  if(req.isAuthenticated()){
    return next();
  }

  return res.redirect('/users/login');
});
```

Create an AJAX Appliction to for Managing Users

CORS Cross-Origin Resource Sharing

```
//Set up CORS
app.use(function(req, res, next) {
  res.header('Access-Control-Allow-Credentials', true);
  res.header("Access-Control-Allow-Origin", "*");
  res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');
  res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, X-HTTP-Method-Override, Content-Type, A
ccept');
  if ('OPTIONS' == req.method) {
    res.send(200);
  } else {
    next();
  }
});
```

AJAX

Create a method called viewIndex. This method will make an API call to the users index end point aka /.

```
function viewIndex(){
  //Define the target url
  var url = 'http://localhost:3000/api/users';

  //Create a new AJAX request
  var xhr = new XMLHttpRequest();
  xhr.open('GET', url);
  xhr.send();

  xhr.onload = function(){
    console.log(JSON.parse(xhr.response));
  }
```

```
function viewIndex(){
    var url = 'http://localhost:3000/api/users';

    var xhr = new XMLHttpRequest();
    xhr.open('GET', url);
    xhr.send();

    xhr.onload = function(){
        let data  = JSON.parse(xhr.response);
        var rows = '';

        for(var i=0; i<data['users'].length; i++){
            let x = data['users'][i];
            let name = `${x.first_name} ${x.last_name}`;
            rows = rows + `<tr>
                <td>
                    <a href="#edit-${x._id}"
                        onclick="viewUser('${x._id}')">
                        ${name}
                    </a>
                </td>
                <td>${x.username}</td>
                <td>${x.email}</td>
            </tr>`;
        }

        var app = document.getElementById('app');
        app.innerHTML = `<table class="table">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Username</th>
                    <th>Email</th>
                </tr>
            </thead>
            <tbody>${rows}</tbody>
        </table>`;

    }
}

function viewUser(id){
    var url = 'http://localhost:3000/api/users/' + id;

    var xhr = new XMLHttpRequest();
    xhr.open('GET', url);
    xhr.send();

    xhr.onload = function(){
        var data = JSON.parse(xhr.response);
        var user = data.user;
        var app = document.getElementById('app');

        app.innerHTML = `
            <h2>${user.last_name}, ${user.first_name}</h2>
            <table class="table">
                <tbody>
                    <tr><th>ID</th><td>${user._id}</td></tr>
                    <tr><th>First Name</th><td>${user.first_name}</td></tr>
                    <tr><th>Last Name</th><td>${user.last_name}</td></tr>
                    <tr><th>Username</th><td>${user.username}</td></tr>
                    <tr><th>Email</th><td>${user.email}</td></tr>
                </tbody>
            </table>

            <h3>Edit the User Record</h3>
            <form id="editUser" action="/api/users" method="put">
                <input type="hidden" name="_id" value="${user._id}">
                <div>
                    <label for="username">Username</label>
                    <input
                        type="text"
                        name="username"
                        id="username"
                        value="${user.username}">
                </div>

                <div>
                    <label for="email">Email</label>
                    <input
                        type="text"
                        name="email"
                        id="email"
                        value="${user.email}">
                </div>

                <div>
                    <label for="first_name">First Name</label>
                    <input
                        type="text"
                        name="first_name"
                        id="first_name"
                        value="${user.first_name}">
                </div>

                <div>
                    <label for="last_name">Last Name</label>
                    <input
                        type="text"
                        name="last_name"
                        id="last_name"
                        value="${user.last_name}">
                </div>

                <input type="submit" value="Submit">
            </form>

            <div class="actions">
                <a href="#deleteUser-${user._id}"
                    onclick="deleteUser('${user._id}');"
                    class="text-danger"
                >Delete</a>
            </div>
            `;

        var editUser = document.getElementById('editUser');
        editUser.addEventListener('submit', function(e){
            e.preventDefault();
            var formData = new FormData(editUser);
            var url = 'http://localhost:3000/api/users';
            var xhr = new XMLHttpRequest();
            xhr.open('PUT', url);
            xhr.setRequestHeader(
                'Content-Type',
                'application/json; charset=UTF-8');

            var object={};
            formData.forEach(function(value, key){
                object[key] = value;
            });
            xhr.send(JSON.stringify(object));

            xhr.onload = function(){
                let data = JSON.parse(xhr.response);
                if(data.success===true){
                    viewIndex();
                }
            }

        });

    }
}

function createUser(){
    var app = document.getElementById('app');
    app.innerHTML = `<h2>Create a New User</h2>
    <form id="createUser" action="/api/users" method="post">
        <div>
            <label for="username">Username</label>
            <input type="text" name="username" id="username">
        </div>

        <div>
            <label for="email">Email</label>
            <input type="text" name="email" id="email">
        </div>

        <div>
            <label for="first_name">First Name</label>
            <input type="text" name="first_name" id="first_name">
        </div>

        <div>
            <label for="last_name">Last Name</label>
            <input type="text" name="last_name" id="last_name">
        </div>

        <input type="submit" value="Submit">
    </form>`;
    var createUser = document.getElementById('createUser');
    createUser.addEventListener('submit', function(e){
        e.preventDefault();

        var formData = new FormData(createUser);
        var url = 'http://localhost:3000/api/users';
        var xhr = new XMLHttpRequest();
        xhr.open('POST', url);

        xhr.setRequestHeader(
            'Content-Type',
            'application/json; charset=UTF-8'
        );

        var object = {};
        formData.forEach(function(value, key){
            object[key]=value;
        });

        xhr.send(JSON.stringify(object));
        xhr.onload = function(){
            let data = JSON.parse(xhr.response);
            if(data.success===true){
                viewIndex();
            }
        }
    });
}

function deleteUser(id){
    if(confirm('Are you sure?')){

        var url = 'http://localhost:3000/api/users/' + id;

        var xhr = new XMLHttpRequest();
        xhr.open('DELETE', url);
        xhr.send();

        xhr.onload = function(){
            let data = JSON.parse(xhr.response);
            if(data.success===true){
                viewIndex();
            }
        }
    }
}

var hash = window.location.hash.substr(1);
if(hash){
    let chunks = hash.split('-');

    switch(chunks[0]){

        case 'edit':
            viewUser(chunks[1]);
        break;

        case 'createUser':
            createUser();
        break;

        default:
            viewIndex();
        break;

    }
}else{
    viewIndex();
```

```
}
```

Lab

1. Create an articles model
   i. The schema MUST contain the following fields
      - title - String
      - slug - String
      - keywords - String
      - description - String
      - body - String
      - published - Date
      - created - Date
      - modified - Date
   ii. Created and modified dates MUST default to to now
   iii. Published date MUST be required
   iv. Published date SHOULD default to now
   v. The modeified datae MUST autoset on on update (pre save) 1.The slug MUST autoset from the title (pre validation)
2. Create a full CRUD REST API for articles

Walkthrough

Create a New Feature Branch

A feature branch gives us a place to develop a new feature. Here, we can commit changes specific to a given feature. Once that feature is complete we can merge our changes into the master branch.

```
git checkout -B articles
```

</>code Create the Arcitle Model

</>code Auto gen a slug prior to validation

</>code Add routes

UI

1. Build a CMS
   - Build a JS app that allows us to work with the API.
   - Build a static view page that is not Ajax.
2. Add an additional Passport strategy of your choosing (GitHub, Facebook, Twitter, etc)
3. Add the ability to reset passwords
4. Create a SPA (single page application) called auth, this will provide
   - user registration
   - user login
   - the ability to change a password
5. Build a layout for website and style the app.
6. Gulp Task
   - Add *src* and *dist* folders to the public directory.
   - Create a Gulp task that will automate the front end packaging

Additional Resources

- Express
- Pug
- NodeJS Security Checklist

Next: Reverse Proxy

ReverseProxy

A reverse proxy allows another web server such Apache, IIS, Nginx, etc to stand in front of ExpressJS and is recommended as a production best practice by the ExpressJS team. In this example ExpressJS will run on localhost:3000 and Apache will field the requests to *http:loc.mean.example.com* (over port 80) and direct those to *http://localhost:3000*

Install the required Apache mods.

```
sudo a2enmod proxy
sudo a2enmod proxy_http
```

Create a VHOST

```
<VirtualHost 127.0.0.102:80>

    ServerName loc.mean.example.com
    ServerAdmin webmaster@localhost

      <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>

    ProxyRequests Off
    ProxyPreserveHost On

    ProxyPass / http://localhost:3000/
    ProxyPassReverse / http://localhost:3000/

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Update your hosts file

```
sudo vim /etc/hosts
```

and add the entry

```
127.0.0.102    loc.mean.example.com
```

Open a browser and navigate to loc.mean.example.com

Additional Resources

- Advantages of a reverse proxy in front of Node.JS
- Production best practices: performance and reliability
- (Using nginx as a reverse proxy in front of your Node.js application)[http://www.nikola-breznjak.com/blog/javascript/nodejs/using-nginx-as-a-reverse-proxy-in-front-of-your-node-js-application/]
- Expose Node.js on an IIS Server by Reverse Proxying With ARR

Next: Web Sockets

WebSockets

"The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. [1]"

Simple Chat App

Additional Resources

- The WebSocket Protocol
- sokcet

Footnotes

[1] RFC 6455 - The WebSocket Protocol.

Next: PM2

PM2

Introduction

PM2 is one of many process managers for NodeJS. We use it primarily to allow the server to auto-recover from errors and managing the number of processes running per machine.

We will start by forcing MongoDB to start up on boot. *rc.local* executes as the system boots.

```
vim /etc/rc.local
```

Add the line `service mongod start` right before `exit 0`. */etc/rc.local* should now read as follows, if it does reboot your system (DO NOT delete anything that was there before, we are only adding to the file).

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# start MongoDB on boot
service mongod start

exit 0
```

When the system boots up type the command `mongo` into a terminal window, if MongoDB starts up you ready for the next step.

```
sudo npm -g install pm2
```

In Atom navigate to *mean.example.com* and add the file *process.yml* and add the following.

~~~
```

*.bashrc* executes when a given user account is loaded.

```sh
cd ~
vim .bashrc
~~~

Add the following line.

```
cd /var/www/mean.example.com && pm2 start process.yml && cd ~
```

Then restart your bash script.

```
. .bashrc
```

Navigate to *loc.mean.example.com*

Additional Resources

- PM2

Next: Angular

Chapter 14: Angular

Angular is a toolkit that makes it easy to build web, mobile, or the desktop applications with using web technology. Angular combines TypeScript, design patterns and auto bundling (think automated gulp) to compile, transpile and package front end assets in realtime. Angular is a CLI based application that runs a dev server on port 4200. Once your satisfied with your project it can be packaged into a distribution package consisting of HTML, CSS and JavaScript.

In this chapter we will

- Learn the basics of TypeScript.
- Complete Angular's Tour of Heroes (TOH) tutorial.
- Port your NASA APOD application to Angular.
- Build a CMS application that connects to the API you developed in the previous chapter.

Next: TypeScript

TypeScript

Introduction

TypeScript is a superset of JavaScript that adds strict typing and class decorators as a way to extend functionality to other components in a system. Browsers do not directly interpret TypeScript, rather, TypeScript is transpilied into JavaScript.

Install TypeScript

```
sudo npm install -g typescript
```

```
vim ~/hello.ts
```

```
function msg(who) {
    return "Hello, " + who;
}

let who = "World";

document.body.innerHTML = msg(who);
```

```
tsc hello.ts
```

Additional Resources

- TypeScript

Tour of Heroes

We will start by completing Angular's Tour of Heros tutorial. Then modify the Tour of Heroes app to work with an API which we will also build into our ExpressJS application. Then we will create a CMS app which will serve as a drop in replacement for our vanilla js SPA.

NASA - Astronomy Picture of the Day

Create a Authentication App.

This app will allow a user to login and out of your ExpressJS app using an Angular application that connects to the API.

Install the angular shell

```
sudo npm install -g @angular/cli
cd ~
ng new ng-auth
cd ng-auth
ng serve --open
```

If your browser did not automatically open, open a browser an navigate to *http://localhost:4200* Open Atom and add *~/ng-auth* as a project in the Atom sidebar.

Note the *src* directory this is the directory that runs the local webserver, the *app* directory is the location of your application code. Open the file */src/index.html*, this is an template. All the code executed from with in */src/app* is injected into the `app-root` tags of this page. Inside *app* the main files to focus on are *app.component.html*, *app.component.css*, and *app.component.ts*.

- *app.component.html* - An HTML template for use by a component.
- *app.component.css* - Style definitions for use by a component.
- *app.component.ts* - In Angular, code is broken down into components; the business logic of your app.

Change the application's title

Open *src/app/app.component.ts* and change

```
//Change
export class AppComponent {
  title = 'app';
}

//to
export class AppComponent {
  title = 'User Authentication';
}
```

In the browser return to *http://localhost:4200* and you'll see the message *"Welcome to User Authentication!"*

Now open the file */src/app/app.component.html* and take note of the double curly braces `{{ title }}` double curly braces in an Angular template allows for variable injection. Instance variables set in the corresponding `.ts` file will be available for injection in the template.

```
<h1>
    Welcome to {{ title }}!
</h1>
```

The *component decorator* is where we create the relationship between components, styles and templates.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

Create a Users App

Create a CMS App

Labs

- Create an app that works with the NASA API.

Additional Resources

- Tour of Heros.
  - Try creating an API to work with the Tour of Heros application
- Angular Docs.

Build a CMS in Angular

In this section you build a CMS application that will interact with the API you built in the previous chapter.

Build a CMS

In Angular build a basic CRUD application for managing users via the users API provided by the *mean.example.com* project. Name this project *ng-cms*.

Before just dive into code stop and think about what you already know about the application you are about to build, this will help you find a starting point.

What do I know?

- I know I'm working with an API.
  - This tells me I will need to work with the HttpClient and with HttpHeaders
- I know I'm working with *http://localhost:3000/api/users*
  - This tells me I will need to define this URL someplace.
- I know I'm working with data
  - This tells me I will need a model (a service and a schema)
  - This tells me I will probably be working observables or promises
- I know the data I'm working with involves users (as the API is users)
  - This tells me I should create
    - a users component
    - a user schema
    - a user service

First things first, let's use the AngularCLI to create our project. Create then cd into the project. Adding the *--style-scss* flag will default AngularCLI generation commands to .scss instead of vanilla .css.

```
ng new ng-cms --style=scss
cd ng-cms
ng serve --open
```

At this point we will have a browser window running the default Angular application.

Next: UsersComponent

Users Component

We know we want to display a list of users, so let's start by creating a users component.

Create the users component

```
ng generate component users
```

Bring the users component into scope, to do this we will replace the contents of *src/app/users/app.component.html* with the following.

src/app/users/app.component.html

```
<app-users></app-users>
```

At this point navigating to https://localhost:4200 should resolve a page stating "users works!".

Commit your changes with the message *Generated the users component*.

```
git add .
git commit -a
```

Next: Schemas and Services

Schemas and Services

In Angular we can represent the data (or model) layer in the form of a service and a schema. A service is a class that will interact with the data in this case an API and schema is a class that defines the data.

The Schema

Build the user schema for now we can define this as an empty class, we can build this out later. Create the file *src/app/user.ts*.

src/app/user.ts

```
export class User {
  user: any;
  users: any;
  errors: any;
  success: boolean;
}
```

Commit your changes with the message *Added a User schema*.

```
git add .
git commit -a
```

HttpClientModule

Since we know we will be making API calls we should go ahead and load the *HttpClientModule*. Add the following to *src/app/app.module.ts*

src/app/app.module.ts

```
import { HttpClientModule } from '@angular/common/http';
...
imports: [
  BrowserModule,
  HttpClientModule
],
...
```

Commit your changes with the message *Import HttpClientModule*.

```
git add .
git commit src
```

The Service

The service is how we interact with an API. There are some things we already know that will help us set up the service for later use.

1. We are working with a web based API.
   - Import HttpClient from Angular.
   - inject HttpClient into the the service constructor.

2. We will post JSON data to the API.
   - Import HttpHeaders from Angular.
   - Create JSON headers for post requests.
3. We will be using observables.
   - Import Observable from the rxjs library.
4. We will be working with the User object.
   - Import our newly craeted User schema.
5. I know the URL of the API endpoints.
   - Set up URL variables accordingly.

We will use AngularCLI to generate the service and automatically import it into AppModule.

```
ng generate service user --module=app
```

Update *src/app/user.service.ts* as follows.

src/app/user.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable } from 'rxjs/Observable';

import { User } from './user';

const httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })
};

@Injectable()
export class UserService {

  //Set up the URL
  private url: string = 'http://localhost:3000/api/users';

  //Call the HttpClient in the Constructor
  constructor(private http: HttpClient) { }

  //Set up a simple observable.
  getUsers(): Observable<User> {
    return this.http.get<User>(this.url);
  }

}
```

To test our service we will add call a test method on initialization of UsersComponent.

src/app/users/users.component.ts

```
import { Component, OnInit } from '@angular/core';

// 1. Import the UserService
import { UserService } from '../user.service';

// 2. Import the User Object/Schema
import { User } from '../user';

@Component({
  selector: 'app-users',
  templateUrl: './users.component.html',
  styleUrls: ['./users.component.scss']
})
```

```
export class UsersComponent implements OnInit {

  // 3. Create a users property of type user
  users: User;

  // 4. Inject the UsersService into the constructor
  constructor(private userService: UserService) { }

  // 6. Make a call to the service on initialization
  ngOnInit() {
    this.getUsers();
  }

  // 5. Craete a local wrapper for
  getUsers(): void {
    this.userService.getUsers().subscribe(
      (response) => {
        this.users = response.users,
        console.log(this.users)
      }
    );
  }
}
```

Now in Chrome navigate to https://localhost:4200 and press [f12] then click on the console tab. You wil now see a JSON object that contains a list of all users on the server.

Commit your changes with the message *Implemented a user service*.

```
git add .
git commit src
```

Remove the `console.log()` from *src/app/users/users.component.ts* and loop through the list of users using ngForOf.

src/app/users/users.component.html

```
<h1>Users</h1>
<ul *ngIf="users">
  <li *ngFor="let user of users">
    <a>{{user.username}}</a>
  </li>
</ul>
```

Commit your changes with the message *Implemented a users UI*.

```
git add .
git commit src
```

Next: Routing

Routing

Routing allows the URL to control the flow of the site.

The following will generate the file *src/app/app-routing.module.ts*.

```
ng generate module app-routing --flat --module=app
```

Update *src/app/app-routing.module.ts* with the following.

src/app/app-routing.module.ts

```
import { NgModule }             from '@angular/core';

// 1. Routing Libraries
import { RouterModule, Routes } from '@angular/router';

// 2. Import the UserComponent
import { UsersComponent }   from './users/users.component';

// 3. Declare your routes
const routes: Routes = [
  // 4. The default route
  { path: '', redirectTo: '/users', pathMatch: 'full' },
  // 5. Map /users to the UsersComponent
  { path: 'users', component: UsersComponent }
];

@NgModule({
  imports: [ RouterModule.forRoot(routes) ],
  exports: [ RouterModule ]
})
export class AppRoutingModule {}
```

Update *src/app/app.component.html* with the following. This will provide an injection point for all components that accessible via routes.

```
<router-outlet></router-outlet>
```

Commit your changes with the message *Implemented basic routing*.

```
git add .
git commit -a
```

Next: User View Component

User View Component

Now we want to view a single user.

Create the user view component

```
ng generate component user-view
```

Bring the users component into scope, to do this we will replace the contents of *src/app/user-view/user-view.component.html* with the following.

Add the user-view component to the routing module. *src/app/app-routing.module.ts*

```
...
import { UserViewComponent }   from './user-view/user-view.component';
...
const routes: Routes = [
  { path: '', redirectTo: '/users', pathMatch: 'full' },
  { path: 'users', component: UsersComponent },
  { path: 'users/view/:id', component: UserViewComponent }
];
...
```

Now add a `routerLink` to your list of users. Use commas to concatenate strings and variables. Update *users/users.component.html* with the following.

users/users.component.html

```
<h1>Users</h1>
<ul *ngIf="users">
  <li *ngFor="let user of users">
    <a [routerLink]="['/users/view/', user._id]">{{user.username}}</a>
  </li>
</ul>
```

The list of users is now clickable, clicking on a user will return a view that says "user-view works!".

Commit your changes with the message *Generated and routed UserViewComponent*.

```
git add .
git commit -a
```

Add `getUser()` method the `UserService` this method will expect *id* as an argument.

src/app/user.service.ts

```
getUser(id: string): Observable<User> {
  return this.http.get<User>(this.url + `/view/${id}`);
}
```

Import ActivatedRoute in UserViewComponent then inject into the component via constructor injection. This will allow us to pull the id parameter out of the URL.

Create a local `getUser()` method that will expect *id* as an argument. This will serve as a wrapper for `UserService.getUser()` .

Call `this.getUser()` from `ngInit()` use ActivatedRoute to create the *id* that will be passed into `this.getUser()` .

user-view/user-view.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

import { UserService } from '../user.service';
import { User } from '../user';

@Component({
  selector: 'app-user-view',
  templateUrl: './user-view.component.html',
  styleUrls: ['./user-view.component.scss']
})
export class UserViewComponent implements OnInit {

  user: User;

  constructor(
    private route: ActivatedRoute,
    private userService: UserService
  ) { }

  ngOnInit() {
    const id = this.route.snapshot.paramMap.get('id');
    this.getUser(id);
  }

  getUser(id): void {
    this.userService.getUser(id).subscribe(
      (response) => {
        this.user = response.user
      }
    );
  }
}
```

Update *user-view/user-view.component.html* to show the details of a single user. Use `*ngIf` to make sure the user obejct is populated before calling it into the view.

user-view/user-view.component.html

```
<div *ngIf="user">
  <h1>{{user.first_name}} {{user.last_name}}</h1>
  <div><strong>Username:</strong> {{user.username}}</div>
  <div><strong>Email:</strong> {{user.email}}</div>
  <button>Edit</button>
  <button>Delete</button>
</div>
```

Commit your changes with the message *Implemented a user view*.

```
git add .
git commit -a
```

Next: Create User

User Create Component

We need to be able to add new users to the system. We will create a web form that will make a post request to the users API.

```
ng generate component user-create
```

Add a route

```
...
import { UserCreateComponent }   from './user-create/user-create.component';
...
const routes: Routes = [
  { path: '', redirectTo: '/users', pathMatch: 'full' },
  { path: 'users', component: UsersComponent },
  { path: 'users/view/:id', component: UserViewComponent },
  { path: 'users/create', component: UserCreateComponent }
];
...
```

Navigate to http://localhost:4200/users/create and you'll see the message "user-create works!"

Commit your changes with the message *Generated and routed UserCreateComponent*.

```
git add .
git commit -a
```

Since we are creating a user we will want to work with NgForms. Import the FormsModule into AppModule then import NgForm into the user-create component.

app.module.js

```
...
import { FormsModule }   from '@angular/forms';
...

  ...
  imports: [
    ...
    FormsModule
  ]
  ...
```

Commit your changes with the message *Imported FormsModule*.

```
git add .
git commit -a
```

user.service.ts

```
  createUser (user: User): Observable<User> {
    return this.http.post<User>(this.url + '/create',user, httpOptions);
  }
```

user-create/user-create.component.ts

```
import { NgForm } from '@angular/forms';
```

user-create/user-create.component.ts

```
import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Router } from "@angular/router";

import { UserService } from '../user.service';
import { User } from '../user';

@Component({
  selector: 'app-user-create',
  templateUrl: './user-create.component.html',
  styleUrls: ['./user-create.component.scss']
})
export class UserCreateComponent implements OnInit {

  user = new User();
  errors: Array<any> = [];
  errorMessage: string;

  constructor(
    private userService: UserService,
    private router: Router
  ) { }

  ngOnInit(): void{}

  response(response): void{
    if(response.success===false){
      this.errors = response.errors.errors;
      this.errorMessage = response.errors.message;
    }

    if(response.success===true){
      this.router.navigate(['/users/view/', response.user._id]);
    }
  }

  onSubmit(): void {
    this.userService.createUser(this.user).subscribe(
      (response) => {
        this.response(response)
      }
    );
  }
}
```

Then add a form to the user-create view. We want will bind the form the ngSubmit.

user-create/user-create.component.html

```
<h1>Create a New User</h1>

<form (ngSubmit)="onSubmit()" #createUser="ngForm">
  <div *ngIf="errorMessage" class="alert error">{{errorMessage}}</div>
  <div>
    <label for="username">Username</label>
    <input [(ngModel)]="user.username" type="text" id="username" [ngModelOptions]="{standalone: true}">
    <div class="error" *ngIf="errors.username">{{errors.username.message}}</div>
  </div>

  <div>
    <label for="email">Email</label>
    <input [(ngModel)]="user.email" type="text" id="email" [ngModelOptions]="{standalone: true}">
    <div class="error" *ngIf="errors.email">{{errors.email.message}}</div>
  </div>

  <div>
    <label for="first_name">First Name</label>
    <input [(ngModel)]="user.first_name" type="text" name="first_name" id="first_name" [ngModelOptions]="{stand
alone: true}">
    <div class="error" *ngIf="errors.first_name">{{errors.first_name.message}}</div>
  </div>

  <div>
    <label for="last_name">Last Name</label>
    <input [(ngModel)]="user.last_name" type="text" id="last_name" [ngModelOptions]="{standalone: true}">
    <div class="error" *ngIf="errors.last_name">{{errors.last_name.message}}</div>
  </div>
  <button type="submit">Submit</button>

</form>
```

Commit your changes with the message *Added the ability to create a user*.

```
git add .
git commit src
```

Next: Edit User

User Edit Component

```
ng generate component user-edit
```

app-routing.module.ts

```
...
import { UserEditComponent }   from './user-edit/user-edit.component';

const routes: Routes = [
  { path: '', redirectTo: '/users', pathMatch: 'full' },
  { path: 'users', component: UsersComponent },
  { path: 'users/view/:id', component: UserViewComponent },
  { path: 'users/create', component: UserCreateComponent },
  { path: 'users/edit/:id', component: UserEditComponent }
];
...
```

Navigate to http://localhost:4200/users/edit/5a5e285039084a199e4515f9 you will see a that says user-edit works!

Commit your changes with the message *Generated and routed UserEditView*.

```
git add .
git commit src
```

Add an `editUser()` method to *users.service.ts*.

users.service.ts

```
editUser (user: User): Observable<User> {
  return this.http.post<User>(this.url + '/edit',user, httpOptions);
}
```

user-view/user-edit.component.ts

```
import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Router } from "@angular/router";
import { ActivatedRoute } from '@angular/router';

import { UserService } from '../user.service';
import { User } from '../user';

@Component({
  selector: 'app-user-edit',
  templateUrl: './user-edit.component.html',
  styleUrls: ['./user-edit.component.scss']
})
export class UserEditComponent implements OnInit {

  user: User;
  errors: Array<any> = [];
  errorMessage: string;

  constructor(
    private userService: UserService,
    private route: ActivatedRoute,
    private router: Router
  ) { }

  ngOnInit(): void {
    const id = this.route.snapshot.paramMap.get('id');
    this.getUser(id);
  }

  getUser(id): void {
    this.userService.getUser(id).subscribe(
      user => this.user = user.user
    );
  }

  response(response): void{
    if(response.success===false){
      this.errors = response.error.errors;
      this.errorMessage = response.error.message;
    }

    if(response.success===true){
      this.router.navigate(['/users/view/', response.user._id]);
    }
  }

  onSubmit(): void {
    this.userService.editUser(this.user).subscribe(
      (response) => {
        this.response(response)
      }
    );
  }

}
```

user-view/user-edit.component.html

```
<h1 *ngIf="user">Edit {{user.first_name}} {{user.last_name}}</h1>

<form *ngIf="user" (ngSubmit)="onSubmit()" #editUser="ngForm">
  <div *ngIf="errorMessage" class="alert error">{{errorMessage}}</div>

  <input [(ngModel)]="user._id" type="hidden" id="_id" [ngModelOptions]="{standalone: true}">

  <div>
    <label for="username">Username</label>
    <input [(ngModel)]="user.username" type="text" id="username" [ngModelOptions]="{standalone: true}">
    <div class="error" *ngIf="errors.username">{{errors.username.message}}</div>
  </div>

  <div>
    <label for="email">Email</label>
    <input [(ngModel)]="user.email" type="text" id="email" [ngModelOptions]="{standalone: true}">
    <div class="error" *ngIf="errors.email">{{errors.email.message}}</div>
  </div>

  <div>
    <label for="first_name">First Name</label>
    <input [(ngModel)]="user.first_name" type="text" name="first_name" id="first_name" [ngModelOptions]="{stand
alone: true}">
    <div class="error" *ngIf="errors.first_name">{{errors.first_name.message}}</div>
  </div>

  <div>
    <label for="last_name">Last Name</label>
    <input [(ngModel)]="user.last_name" type="text" id="last_name" [ngModelOptions]="{standalone: true}">
    <div class="error" *ngIf="errors.last_name">{{errors.last_name.message}}</div>
  </div>
```

```
    <button type="submit">Submit</button>

  </form>
```

Commit your changes with the message *Added the ability to edit a user.

```
git add .
git commit src
```

Next: Delete User

Delete a User

Update *user.service.ts*

```
deleteUser (id: string): Observable<User> {
  return this.http.get<User>(this.url + `/delete/${id}`);
}
```

Import Router so that we can redirect after a delete.

user-view/user-view.component.ts

```
deleteUser(id: string): void {
  if(confirm("Are you sure to delete " + this.user.user.username)) {
    this.userService.deleteUser(id).subscribe(
      ()=>{this.router.navigate(['/users'])}
    );
  }
}
```

Update the delete button *user-view/user-view.component.html*

```
<button [routerLink]="['/users/edit/', user._id]">Edit</button>
<button (click)="deleteUser(user._id)">Delete</button>
```

Commit your changes with the message *Added the ability to delete a user*.

```
git add .
git commit src
```

Next: NextSteps

Next Steps

Now it's time to add some navigation and the components for creating content. We added the users components together. Now it's time to take what you've learned and build the Articles components on you own.

Navigation

Add some navigation to the app component.

src/app/app.component.html

```
<nav>
  <button [routerLink]="['/users']">Users</button>
  <button [routerLink]="['/users/create']">New User</button>
  <button>Articles</button>
  <button>New Article</button>
</nav>

<router-outlet></router-outlet>
```

ArticlesComponent

Generate ArticlesComponent

```
ng generate component articles
```

Add ArticlesComponent to routing.

```
import { ArticlesComponent }   from './articles/articles.component';

const routes: Routes = [
  { path: '', redirectTo: '/users', pathMatch: 'full' },
  { path: 'users', component: UsersComponent },
  { path: 'users/view/:id', component: UserViewComponent },
  { path: 'users/create', component: UserCreateComponent },
  { path: 'users/edit/:id', component: UserEditComponent },
  { path: 'articles', component: ArticlesComponent }
];
```

Create full CRUD for the articles API

At this point we have created the articles component and added it to routing.

1. Finish implementing the ArticlesComponent details.
2. Create a Article Schema - *src/app/article.ts*
3. Create ArticleService - *src/app/article.service.ts*
4. Create and route ArticleView - *src/app/article-view/**
5. Create and route ArticleCreate - *src/app/article-create/**
6. Create and route ArticleEdit - *src/app/article-edit/**

Chapter 15: Apache Cordova

- Convert the Nasa App to mobile.
- Install Ionic
- Build the users App

Apache Cordova

Apache Cordova is a free and open source environment that allows us to use web technologies to target multiple platforms with a single code base. This is used primarily for mobile development. While we are able to write the code using web technologies, we still need access to that platforms build environment. Cordova provides a mechanism to compile technology into native application code but it still needs the native environment to build and test that package. If you want to compile your application into an iOS app you will still need access to a MAC running the latest version of xCode. Android on the other hand can build on Windows, MAC or Linux as a result we will focus our build on Android. For the most part, anything you build in Cordova for Android should run on iOS save for the occasional tweaks.

Android Studio is the offical IDE (Integrated Dev Environment) for building Andorid applicators, this provides everything you will need to build Android applications. There are however a few dependencies.

In this lesson we will install

- Apache Cordova
- Java
- Gradle
- Android studio
- A few 32 bit binaries.

Install Apache Cordova

Cordova is built on top of Node; we will use npm to do a global install.

```
sudo npm install -g cordova
```

Install the Java SDK

Android runs on top of Java (and Java compatible APIs) we will need to install Java so we can compile our web based build into Java. We will use Oracle's JDK for this (there are rumors that Google will switch future build to Open-JDK).

Start by adding Oracle's PPA.

```
sudo add-apt-repository ppa:webupd8team/java
```

Update your apt repos list

```
sudo apt-get update
```

Install the JDK

```
sudo apt-get install oracle-java8-installer
```

Choose the desired installation Run the command `sudo update-alternatives --config java` and chose from the resulting menu, which should be similar to the following. In this case, I selected option 0 *auto mode*

```
  Selection    Path                                     Priority   Status
------------------------------------------------------------
```

```
  0            /usr/lib/jvm/java-8-oracle/jre/bin/java   1081      auto mode
* 1            /usr/lib/jvm/java-8-oracle/jre/bin/java   1081      manual mode
```

You will need to set your JAVA_HOME Environment Variable (so that running programs can find Java). To do this you will need to find your Java path; do this with the following command (the result of which will contain your Java path).

```
sudo update-alternatives --config java
```

Now set the path using your favorite editor. In my case the path is at */usr/lib/jvm/java-8-oracle/jre/bin/java* so I will add this line `JAVA_HOME="/usr/lib/jvm/java-8-oracle/jre/bin/java"` to the environment file.

```
sudo vim /etc/environment
```

Once you have added the that line, you will want to reload the file.

```
source /etc/environment
```

```
java -version
```

Install Gradle

In short Gradle is a the build system used by Android. Stack Overflow has a more detailed answer. You can install this using Apt, but the Ubuntu repos are a little behind on this one, so it's better to install it manually.

Download and Unpack Gradle

```
cd ~/Downloads
wget https://services.gradle.org/distributions/gradle-4.7-bin.zip
sudo mkdir /opt/gradle
sudo unzip -d /opt/gradle gradle-4.7-bin.zip
```

Add an Environmental Variable on Startup

Open the *environment* file

```
sudo vim /etc/environment
```

add the following lines, the first is for Gradle, the others you will need later so add them now. Replace *YOUR_USER_NAME* with the user name you use to login to your system.

```
export PATH=$PATH:/opt/gradle/gradle-4.5/bin
export ANDROID_HOME=/home/YOUR_USER_NAME/Android/Sdk
export PATH=${PATH}:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
```

Restart environment

```
source /etc/environment
```

Install additional 32 bit libraries

```
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-1.0:i386
```

Install the Android SDK

If you were to build an Android application from scratch, this is what you would use. Cordova needs access to the SDK for it's builds and we need access to the emulators. You will use Cordova to write the code, Cordova will use the Android SDK to build your applications and you will use Android Studio to build your emulators.

Download Android Studio

```
cd ~/Downloads
sudo unzip android-studio-ide-*-linux.zip -d /usr/local
cd /usr/local/android-studio/bin
./studio.sh
```

Follow the prompts and keep choosing next.

_ At some point you will be asked to create or import a new project. This is required but we will not use it. When prompted to do so, go ahead and create a project. Create a new project called Hello World

- Choose create with no activity _

You should now have a running instance of Android Studio. Add a desktop entry *Tools* > *Create desktop entry..*. and lock Android Studio to your launcher.

Now it's time to create an (AVD (Android Virtual Device))[https://developer.android.com/studio/run/managing-avds.html]

Tools > Android > AVD Manager Click the *Create Virtual Device* button Choose Nexus 5x Click on the *x86 images* tab and choose *Nougat 25 x86_64* (Download if required) Choose the default options from the AVD screen and click *Finish* From the *Your Virtual Devices* dialog click the green arrow beside our new device.

Hello World

Now let's get started with Cordova. We will start with the classic Hello World example. We will create our Hello World application is a package called hello. This will create a starter package with a few lines of source code to get your started.

```
cd ~
cordova create hello com.example.hello HelloWorld

cd hello
```

Check your list of platforms

```
cordova platform ls
```

Add the Android platform to your project.

```
cordova platform add android
```

Build and Android package from source code.

```
cordova build android
```

Start the emulator

```
cordova emulate android
```

Close the emulator.

Debugging with Logcat.

Logcat is the default android debugger, we can use this for tracking down issues in our web code. To do this we will want to add the console plugin to our Cordova app.

```
cordova plugin add cordova-plugin-console
```

Start your emulator and in another terminal start logcat

```
cordova emulate android

adb logcat
```

Everything that happens in the emulator is logged. Logcat `adb` is installed with android studio and lets us read the logs in real time. In a new console, run the logcat command.

```
adb logcat
```

You'll notice straight away that this is far too verbose to be useful, so your will want to run it with filters. We will use the regex filter to only return messages that have contain the string *INFO:CONSOLE*. This will limit Logcat's output primarily to `console.log()` calls. Making it far less verbose and allowing us focus on messages that matter.

```
adb logcat ActivityManager:I Cam:V -e INFO:CONSOLE*
```

Camera Plugin https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-camera/

Additional Resources

Use SQLite In Ionic 2 Instead Of Local Storage

Chapter 16: Ionic

- Sign up for Ionic Pro.
- Choose the free kickstarter plan.
- Go to the Geetting Started guide.
- Install Ionic and create your first app

```
sudo npm install -g ionic
ionic start myApp tabs
```

- Review the file structure and draw comaprisons to Angualar.
- Exit this program and create the CMS app

  ```
  ionic start ionic-cms sidemenu
  ionic serve
  ```

Create a users page and wire it into navigation.

1. Import the UserProvider (aka Service in Angular)
2. Import the User schema/model
3. Declare users as an Array containing user objects
4. Inject the UserProvider
5. Create a wrapper for the users provider
6. call the getUsers() wrapper

```
ionic generate page users
```

</> code Declare UsersPage in app.module.ts

</> code Add the UserPage to your side menu.

Create a data provider to connect to the users API.

```
ionic generate provider user
```

UserProvider will give us access to a users API. We will create a getUsers() method that will return a list of users; this is the goal of UsersPage. To allow UsersPage to access UserProvider we will need to import and inject it into UsersPage.

</> code Import the UserProvider and inject it into UserPage.

At this point your application should crash when attempting to access the UsersPage. This is because the data provider (UserProvider) is a calling HttpClient but HttpClientModule is not getting loaded. To resolve this issue we will load it into app.module.ts.

</> code Add HttpClientModule

It's always good to build one piece at a time. We want to make sure the connection between our provider and controller (UsersPage) is working. While we may not yet be ready to implement each provider method it's a good practice to make sure we've accounted for everything we will need to work with the API. In this case we will be dealing with basic CRUD (Create, Read, Update, Delete) logic. This tranlates in to

- get a user `getUser()`

- get many users `getUsers()`
- create a user `createUser()`
- update a user `updateUser()`
- delete a user `deleteUser()`

</> code Stub out the user provider with the soon to be used methods. Start with a having each method execute simple `console.log()` .

We will want to create a user data object, other paradigms may refer to this as a model or a schema. For the sake of argument we will call it a model. For now we will just define the class. Create a models dirctory and user model *models/user/user.ts*

</> code Stub a User object.

</> code Import the User object (model) into UserProvider.

Before we begining implementing UserProvider against the API lets create a wrapper in out UserPage and test the connection to the provider. To do this we will create a private method that calls the `getUsers()` method in UserProvider. We will call that method from the UserPage constructor. Opening the JS console in the dev tools panel then navigationg to the users page in our app will now display "Get Users". At this point we know we have a good connection and we can focus on implementing the `getUsers()` logic.

</> code Implement a basic wrapper for `getUsers()` .

Now that we have a basic connection between UserProvider and UserPage we will want to implement an API call against `UserProvider.getUsers()` . This will require the following steps.

1. Import Observable from rxjs
2. Set the base URL inside of UserProvider
3. Create an Observable of type User and implemnet the get logic
4. Subscribe to the observable and catch the response.

</> code Implement the `getUsers()` API call.

Opening the JS console in the dev tools panel then navigationg to the users page in our app will now display a JSON object containing a an Array of users.

Make the uses list public so that it will be accessible by the view.

1. Create a public instance varaible to hold the list of users. This will hold an Array of user objects.
2. Set the list of users the users instance variable. This will give view access to the list of users.
3. Log the current state of the users instance.

</> code Add the users data to a public variable.

At this point the `console.log()` will show a deliniated list of users.

Our view now has access to the public users instance. We can display dispay a list of users in the view using Angular's ngForOf directive to build an Ionic list. You can now remove the `console.log()` from UsesPage.

</> code Implement the users view.

Add a Loader

Our data comes from a web API. This means any network latency can make a page load feel sluggish or even broken. Using a loader is a good way to ease the pain ofa slow page load.

1. Import LoadingController
2. Inject the LoadingController into the constructor
3. Create a space in memory to hold a loader (an instance variable)

4. Create a method and display a loader
5. Call the loaded when requesting user data
6. Dismiss the loader after the HTTP request has completed

</> code Add a loader.

</> bug fix The previous commit was missing an import.

</> bug fix Typing issues brought to the surface byt the previous fix.

</> code Remove ListPage, it was only used for demo purposes.

Create UserPage and Display a Single User

Implement a page to display a single user.

1. Generate a user page
2. Add UserPage to AppModule
3. Add navigation between UsersPages and UserPage
4. Import the UserProvider (aka Service in Angular)
5. Import the User model
6. Declare a public user variable
7. Create a wrapper for the UserProvider.getUser()
8. Call the getUsers() wrapper
9. Build the view

```
ionic generate page user
```

</> code Add UserPage to NgModule.entryComponents

</> code Provide navigation between UsersPage and UserPage.

</> code Rough UserPage implementation - API test via console.log() (steps 4-8)

</> code - Added basic user details.

Generate CreateUserPage

Add the functionality to create a new user.

1. Generate UserCreatePage </> code
2. Add UserCreatePage to AppModule </> code
3. Link from UsersPage to UserCreatePage </> code
4. Import the User model </> code
5. Import the UserProvider </> code
6. Declare a public user variable instanitaed as a new User object </> code
7. Create a wrapper for the UserProvider.createUser() </> code
8. Implement UserProvider.createUser() </> code
9. Build a basic user form and implement a form submit </> code
10. Redirect after submit </> code

```
ionic generate page user-create
```

LAB - Edit and Delete Users

- Implement an edit users page.
- Add the ability to delete a user.
    - This may be it's own page or a method built into another component.

Glossary

Address

1. A storage location in memory.
2. The location of a given server on the internet. An IP address either IPv4 or IPv6.

Fullstack

A reference to both the front and backend of a web site, this generally inclusive the presentation, data and logic layers. Some "stacks" such as LAMP include OS and server software while others such as MEAN focuses on higher level tech.

The MEAN stack is also said to be Fullstack JavaScript in that all critical layers of the tech are written in JavaScript.

A fullstack developer is generally someone who is well versed in the presentation, data and logic layers.

Hybrid Mobile

Mobile applications that are written using web technologies (HTML, CSS and JavaScript) and compiled into native iOS and Android binaries. Common hybrid mobile toolkits include Apache Cordova, PhoneGap, Ionic and React Native.

LAMP

A technology stack consiting of Linux, Apache, PHP and MySQL. Some defintions my include Perl and/or Python.

MEAN

A technology stack consiting of MongoDB, ExpressJS, Angular and NodeJS.

MVC

MVC is a common architecture/pattern consiting pf three layers.

```
* Model - The data layer. This could provide connections to an API, database, csv file,etc.
* Controller - Buisness/control logic.
* View - The presentation layer. This could be a web page, json output, a pdf, spreadsheet, etc.
```

URI

Uniform Resource Identifier is a string of charaters that unambiguously identifies resources on a network. This may or may include an access mechanism. https://www.microtrain.net and www.microtain.net are both examples of URIs. The former is also an example of a URL while the later is not. All URLs are URIs but not all URLs are URIs.

URL

Uniform Resource Locator a type of URI to which an access mechanism has been prepended. https://www.microtrain.net is a URL while www.microtrain.net is not a URL. The later would be of the broader URI class. All URLs are URIs but not all URLs are URIs.

Variable

A reference to a location in memory (an address) which has an indentifer, a value, and a data type.

Appdendix

Cloud Server

In this unit we will launch our cloud infrastructure. This will include creating a thrid party database, registering a domain name, spinning up a web server and creating a valid TLS connection against a real CA.

Amazon Web Services (AWS)

AWS is an auxiliary of Amazon.com that provides on-demand cloud computing platforms such as IaaS, PaaS and SaaS. AWS offers compute power, database storage, content delivery and other functionality to help businesses scale and grow.

Terminology

Cloud Computing

Term used for allowing convenient, on-demand access from anywhere, to computing resources.

5 Charactics of Cloud Computing

- On-Demand Service
- Broad Network Acess
- Resource pooling
- Rapid Elasticty
- Metering

These includes servers, storage, networking, applications and services. That can be easy revised and released.

Software as a Service

On-Demand Pay Per Use of Application. Popular SaaS Providers are:

- Google Systems, Microsoft, and Slack.

Platform as a Service

Encapsulated environments where users can build, compile and run programs without infrastructure worries.

- Elastic Beanstalk
- Windows Azure

- ### Infrastructure as a Service

  Offering of computer resources to deploy virtual machines and etc.

- Amazon EC2; Virtual Environments

AWS Free Tier

AWS Free Tier offers 12 months of semi free service and AWS products. Notable services in the Free Tier include:

Services	Storage	Hours/Month

Amazon S3	5GB	
Amazon EC2		750
Amazon RDS		750
DynamoDB	25GB	
SNS		
CloudFront	50GB	

Register an AWS Account

Head over to AWS hompage and signup.

1. Create AWS Account with credentials.
2. Provide contact information for selected Personal Account.
3. Enter payment information - even though AWS offers free services, they require a vaild payment type to register.
4. Enter code for Identity Verification.
5. Select Support Plan - Basic(Free).
6. Sign in to the Console.

Route 53

https://aws.amazon.com/route53/

Amazon Route 53 is scalable cloud Domain Name System (DNS). This is amazons preferred way of connecting user request to infrastructures running in AWS. For Domain purchase, I reccomend Hover; use this referral link and get $2 off your domain name.

Amazon S3

https://aws.amazon.com/s3/?nc2=h_m1

S3 is object storage built to store and retrieve any amount of data from anywhere. Its a secure, durable and scalable infrastructure. Amazon Free Tier provides free services of S3 up to 5GB of standard storage, 20,000 Get Requests and 2,000 Put Requests.

S3 Setup

1. Navigate Services tab for Amazon S3.
2. Create Bucket.
3. Set Bucket Index Page, Properties and Permissions.
4. Start storing data such as static files(html, javascript, css), photos, videos and etc.
5. Access/Manage objects in bucket.

A bucket is a flat container of objects and doesn't provide hierarchical organization. You can create hierarchy by using object key names that imply a folder structure. When you store data, you assign a unique object key that can later be used to retrieve the data. Object names are prefixed with folder names. Every folder created must have an index document at each level.

Amazon EC2

https://aws.amazon.com/ec2/?nc2=h_m1

EC2 web service interface provides you with complete control of your computing resources. This provides virtual servers known as instances to host web applications. Amazon Free Tier provides up to 750 Hours per month of EC2 usages before additional fees apply. This applies to t2.micro instance types!

Amazon EC2 Install

1. Navigate Services tab for Amazon EC2.
2. Launch Instance under Create Instance Header.
3. Chose an (AMI) - Ubuntu Server 16.04 Free Tier 64-bit.
4. Choose an Instance Type - t2.micro Free Tier.
5. Configure Instance Details (Default).
6. Add Storage (Default).
7. Add Tags - Add a name tag and value.
8. Configure Security Group - Create a new security group and add rule with Type HTTP.
9. Review Instance Launch.
10. Create a new key pair - *This is the only time you can download the key file.
11. Launch Instance

Accessing server instance from terminal

To access AWS instance the public IP and downloaded key.pem file are required. First move the .pem file to your .ssh directory. Next we need to check/configure our permissions.

```
mv Downloads/my-aws-serverKey.pem ~/.ssh/
chmod 600 .ssh/my-aws-serverKey.pem
ssh ubuntu@52.25.your.Public.IP -i .ssh/my-aws-serverKey.pem
```

Now that your server is accessible you might want to install Apache2.

```
sudo apt-get update
sudo apt-get install apache2
```

RDS

https://aws.amazon.com/rds/?nc2=h_m1

Rational Database Service for MySQL, PostgreeSQL, MariaDB, Oracle BYOL or SQL Server. Amazon Free Tier includes 750 Hours of RDS usage per month of db.t2.micro database usage. RDS Free Tier allows up to 20GB of storage.

DynamoDB

https://aws.amazon.com/dynamodb/?nc2=h_m1

Similar to RDS, DynamoDB is a fully managed NoSQL database service. This Free Tier service allows up to 25GB of storage and suppose to handle up to 200M request per month.

Other AWS Free Tier Services

SNS

https://aws.amazon.com/sns/

Free service provided by AWS Free Tier. Amazon Simple Notification Service is a flexible and fully managed messaging and mobile notification service for subscribing endpoints and clients.

Elastic Beanstalk

https://aws.amazon.com/elasticbeanstalk/?nc2=h_m1

Free Tier services that is an easy-to-use services for deploying and scalling web applications without much installation and management of application. You can upload your code and Elastic Beanstalk automatically handles deployment with management containers for environments such as Java, .NET, PHP, NODE.js, and Python on familiar servers. Elastic Beanstalk leverages other AWS Services such as EC2, S3, and SNS.

Elastic Block Storage

https://aws.amazon.com/ebs/?nc2=h_m1

Service for additional EC2 instance storage. EBS is a storage volume for an EC2 instance. These blocks can be attached to any running instance to add, transfer or copy data.

CloudFront

https://aws.amazon.com/cloudfront/

Web service to distribute content to users with high data transfer speeds. Up to 50GB of data transfer total.

Cloud Server

In this unit we will launch our cloud infrastructure. This will include creating a thrid party database, registering a domain name, spinning up a web server and creating a valid TLS connection against a real CA.

Create a MongoDB Atlas Sandbox

https://www.mongodb.com/cloud/atlas

1. Create an account
2. Choose the free sandbox account
3. Create a cluster name
4. Generate a password

Update Your Local App to Connect to the Sandbox

Add a configuration file, since the repository is public create this outside of the repository. We will use Filezilla to transfer this file to the server.

~/config.prod.js

```
var config = {};
config.session = {};
config.cookie = {};

config.mongodb = 'mongodb://USERNAME:PASSWORD@cluster0-shard-00-00-XXXXX.mongodb.net:27017,cluster0-shard-00-01
-XXXXX.mongodb.net:27017,cluster0-shard-00-02-XXXXX.mongodb.net:27017/mean-cms?ssl=true&replicaSet=Cluster0-sha
rd-0&authSource=admin';

//Create a high entropy secret, I would recomend something from https://www.grc.com/passwords.htm
config.session.secret = ';7x,^SpXNq6*X9{9V4\?h:M3?tN96;lhuX_K@WM:]v~~@8V]KYgQ1[.T<uV0B)6';

config.cookie.domain = 'your-new-domain.tld';

module.exports = config;
```

Require the config file in app.js

```
//Call the config file
var config = require('../config.prod');
```

Load a config file base on enviromental variables

```
if(process.env.NODE_ENV==='production'){
  var config = require('../config.prod');
}else{
  var config = require('./config.dev');
}
```

Purchase a Domain

I reccomend Hover use this referal link - https://hover.com/2WvTmBun and get a $2 off your domain name.

Purchase a Cloud Based Web Server

Digital Ocean use this referal link - https://m.do.co/c/7d5ded2774f3 and get a $10 credit.

Set Up Your Droplet

Login to your droplet over SSH. You will be prompted to change your password. Once you have updated your password update apt and run any upgrades.

```
apt-get update
apt-get upgrade
```

Now create a less privileged user. For the sake of this article we will call that user production. Give the production user a good strong password and follow the prompts, you may leave these questions blank.

```
adduser production
```

Git should already be installed. You can verify this by running `git --version`.

Switch to the production user and follow create an ssh key

```
su production
```

- Create an ssh key and add it to your GitHub account.

Start by switching back to the root user

```
su root
```

Install the LAMP stack and the modules needed to run a reverse proxy.

```
apt-get install lamp-server^
a2enmod ssl proxy rewrite headers proxy_http
```

Restart Apache

```
service apache2 restart
```

Test Apache by entering your domain name into a browser window. If you see a page that says *Apache2 Ubuntu Default Page* then your Apache web server is working.

> Due to firewall settings this may not yet work in the classroom.

Assign ownership of the web directory to the production user.

```
chown production:production /var/www -fR
```

Reboot your droplet

```
reboot -n
```

Log back into the server and install NodeJs

```
ssh root@YOUR-IP
```

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
apt-get install -y nodejs
apt-get install -y build-essential

apt-get update
apt-get upgrade
```

Be sure npm is setup globally

```
npm install npm -g
```

Install pm2

```
npm -g install pm2
```

Install your website on the */var/www* path from GitHub.

```
su production
```

Add the production variable to .bashrc

```
vim ~/.bashrc
```

Press [shift] + [g] to go to the bottom of the file. Enter insert mode by pressing [i] and move the cursor to a new blank line at the bottom of the page. Add the following.

```
export NODE_ENV=production
```

Save your changes with [esc] followed by [shift] + [:] then [x] and [enter]. Reload .bashrc with the follwing.

```
cd ~
. .bashrc
```

Clone the repo, install the production packages and start the application.

```
cd ~ && git clone git@github.com:YOUR-GITHUB-ACCOUNT/mean.example.com.git
cd mean.example.com && npm install --production
npm start
```

Test that the site is running by opening a browser and entering your new doamin name followed by port :3000, *YOUR-DOMAIN.TLD:3000*.

Set up pm2

1. Login as production
2. Start the pm2 process

   ```
   cd /var/www/mean.example.com
   pm2 start process.yml
   ```

3. switch to root

   ```
   su root
   ```

4. Add production to the sudoers list

   ```
   usermod -aG sudo production
   ```

5. Write the current pm2 state to production home.

   ```
   sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2 startup systemd -u production --hp /home/pro
   duction
   ```

6. Save the current state

   ```
   pm2 save
   ```

Test that pm2 is working by rebooting your server. Then open a browser and entering you domain name against port 3000. If you see your webiste pm2 has taken effect.

Apache Configuration

```
cd /etc/apache2/sites-available
cp default-ssl mean.example.com
```

comment out the `DocumentRoot` directive.

```
## DocumentRoot /var/www
```

Add a `ServerName` directive, replace MYDOAMIN.TLD with the lowercase version of your doamin name.

```
ServerName MYDOMAIN.TLD
```

Finally set up you reverse proxy.

```
<Proxy *>
    Order deny,allow
    Allow from all
</Proxy>

ProxyRequests Off
ProxyPreserveHost On
ProxyPass / http://localhost:3000/
ProxyPassReverse / http://localhost:3000/
```

Force NON-SSL to an SSL connection, add the following to the top of the VHOST file.

```
<VirtualHost *:80>
    RewriteEngine On
    RewriteCond %{HTTPS} !=on
    RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI} [R=301,L]
</VirtualHost>
```

Force non-www

```
RewriteEngine On
RewriteCond %{HTTP_HOST} ^www\.jasonsnider\.net [NC]
RewriteRule ^(.*)$ https://jasonsnider.net$1 [L,R=301]
```

Restart Apache

```
a2dissite *
service apache2 restart
a2ensite mean*
service apache2 restart
```

LetsEncypt

Create an install a cert from LetsEncrypt

```
## Install Certbot
```

sudo apt-get install software-properties-common sudo add-apt-repository ppa:certbot/certbot sudo apt-get update sudo apt-get install python-certbot-apache ```

Next: Install a Valid SSL Cert

Let's Encrypt

Let's Encrypt

Let's Encrypt provides free and automatic domain validation (DV) certs. Certbot is popular utility for installing and maintaining Let's Encrypt certificates.

Install Certbot

```
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:certbot/certbot
sudo apt-get update
sudo apt-get install python-certbot-apache
```

Install the Certificate

Run Certbot with the Apache option and follow the prompts.

```
certbot --apache
```

Certificate Renewal

Use a crontab to auto renew your cert.

```
crontab -e
```

Add the following line. This will attempt a certificate renewal once a month at 10pm. The certificate will only only renew if it is close to expiry. Replace 15 with today's date minus one, this however, should be no higher than 27 to account for the shortest possible month.

```
0 22 15 * * certbot renew
```

Additional Resources

Install Certbot on Ubuntu

CloudFlare

ZOHO Productivity Suite

Security

This appendix contains information and resources pertinent to information security. While InfoSec is a field in and of itself; it is a cross-functional field. An understanding of security principles and best practices is crucial for long term success in any tecnical field. I highly reccomend for anyone serious about development to delve deeper into this topic.

Additional Resources

- The Security Now podcast.
- Sans NewsBites
- Sans Webcasts

Templates

- HTML5 Template

A Basic HTML Template

```
<!DOCTYPE html>
<html lang="en">
    <head>
      <meta charset="UTF-8">
      <title>Contact</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
       <h1>Contact</h1>
    </body>
</html>
```

Search Engine Optimization

This appendix contains information and resources pertinent to information search engine optimization (SEO).

Samples

Downloadable resources and code files available here

Syllabus (8/13/18 - 10/15/2018)

- Class Hours: Monday-Friday, 9am-5pm
- First Day: Monday, August 13, 2018
- Last Day: Monday, October 15, 2018
- Class will **NOT** meet on
    - Monday September, 3, 2018

Contact

- Join the Slack Channel
- Email bootcamp@microtrain.net

Description

The class is 40 hours a week for 10 weeks. It is comprised of a series of units with each unit corrosponding to a chapter in the courseware. A unit or chapter is compirsed pf working lectures, exercises, labs and additional resources. The goal of this class is to provide a working knowledge of web and mobile application development. This course is not a deep dive on any one topic, rather it is intended to provide a working foundation that will allow you to hit the ground running and give you the knowledge base upon which to build a career. In most cases you will get a light introduction to a topic; typically with in a specific context. We will build on these topics as we progress through the course. The syllabus is tentative and may be adjusted to the pace of the class as a whole. The last two to three weeks of class are open and are intended for to allow each stundet to complete a single or multiple projects to add to their portfolio.

Expectations

- Class starts promptly at 9am, you are expected to have your systems booted up and to be ready to go by 9am.
- Please limit cell phone use during lecture.

Unit Composition

A Lecture

An overview of the topic(s) at hand.

Exercises

Guided application of the topic(s) at hand.

Coding Challenges

Planned and impromptu inclass assignments that will present new and old concepts as challenges to be solved in code.

Labs

The work you are expected to do on your own. Lectures and and exercises should take roughly 40% - 50% of the class time with the rest of the time being spent in labs. If you get ahead it is expected tha

Additional Resources

This section provides links to additional reading, conversation, ebooks, videos online courses ect. While not required it is encouraged and expected that you will spend some time reviewing this material.

Udemy

We use Udemy as supplemental material. While it is not required it is encouraged to at least review the courses provided in the Additional Resources section. You will have access to Udemy for one year from today so event after this course is finished you can continue to learn in a structured environment.

Khan Academy

This provides free online learning for various topics. This is mostly included as a review of Algebra, Geometry, etc. We do not get to heavy with these topics but any one wanting a review can do so through the links provided.

TENTATIVE SCHEDULE

Week 1 (Dev, Bash, HTML, CSS)

Build and Understand a Development Environment ~1 Day

- Install Ubuntu Linux
- Linux basics
- Install dev tools
- Install the LAMP stack
- Configure the Apache Web Server
- NPM
- Git

Programming Basics ~1 day

- Shell Scripting with Bash

Introduction to Web ~3-5 days

- The Anatomy of a URL
- HTML
    - Intorduction to HTML
    - GitHub Pages (Basic Website)
    - Web Forms
    - HTML Resume
- CSS/Preprocessors
    - CSS and Basic Styles
    - SASS/SCSS and Gulp Processes
    - CSS Layouts (floating grids, CSS grids, flexbox)
    - Responsive CSS

Weeks 2-3 (Server Scripting, PHP, JavaScript)

Server Scripiting with PHP ~1-3 days

- Introduction to Templates
  - Build a template in PHP
  - PHP Web Site
  - Form Processing in PHP

JavaScript, jQuery, AJAX and Bootstrap ~5-7 days

- Programming Basics with JavaScript
- Walking the DOM
- Catch Events
- Progamming the Canvas
- jQuery Basics
- Ajax
- Introduction to Bootstrap

Weeks 3-4 (SQL, NoSQL, Web Frameworks)

MySQL and CakePHP ~3 days

- MySQL
- CakePHP
- Unit Testing
- Automated Code Review with Code Climate.

MongoDB, Express ~5 days

- MongoDB
- Express
- REST API
- Web Sockets

Introduction to the Cloud 1-2 days

- Launch a production site to Digital Ocean on your preferred tech stack (MEAN or LAMP).

Output Your personal production site on Digital Ocean on your preferred tech stack (MEAN or LAMP).

Week 5 (Angular)

Angular ~5 days

- Tour of Heroes Tutorial
- Interact with your REST API

Week 6-7 (Hybrid Mobile)

Apache Cordova and Ionic ~10 days

- Learn how to build mobile device emulators.
- Learn how to emulate mobile devices using the Chrome browser and Dev Tools.
- Run exisiting web code as a mobile app
- Ionic Applications

Weeks 8-10

If applicable week 8 may serve as catchup, review or overflow time for any topics of which there are issues.

Projects

Whatever you want to build. Start thinking of a project sooner than later