# Progressive Webapp

**Definition:**

Any application that delivers app-like experience in the browser and can be developed progressively (Can be developed in layers that are independent of each other)

**Characteristics:**

1. Instant loading 7. Linkable
2. Discoverable 8. Re-engageable(Push notification)
3. N/W independent 9. Works everywhere
4. Responsive 10. Fast
5. Installable
6. Secure

**Progressive Enhancement: Adding features incrementally in layers.**

1. It’s a website
2. Adds native installation
3. Adds Web push notifications
4. Add hardware & platform access

**PWAs and Hybrids:**

1. Big differences
2. Store distribution – Need to pay for Hybrid apps and need to package them
3. No packaging and signing for PWAs
4. No native plugins for PWA. This is where Hybrids take the lead**.**

**PWAs compatibility:** <http://gs.statcounter.com/>(Site for getting stats for browser usage)

1. Chrome on Android
2. Opera Mobile
3. Samsung Internet browser (Chromium based)
4. Firefox developer edition
5. Edge (future, IE +16)
6. Limited APIs available for iOS, so limited support. iOS 11 may be better performing in PWAs
7. Chrome iOS is not chrome. Check user-agent string for chrome.
8. Each instance of the PWA depends on the browser engine it has been installed. Eg: If we installed the same app from 2 different browsers, then we would have 2 separate instances of the app running independently. This is a problem to be addressed.
9. We could save the credentials of the user if there is only one engine that the app is being installed. Currently this is supported only in Android. But there is a credentials management API in the pipeline that helps the OS manage the user session.
10. PWA is ideal for the apps that are single page applications.
11. PWA supports web-sockets as the browsers supporting the PWAs do that.
12. PWA is great asset to the Physical web. (https://google.github.io/physical-web/)

**Limitations:**

1. Multiplatform support, eg: camera might different in different platform.
2. Responsive design – Taking into account network, UI rendering and other cases for better design.
3. PWAs do not get the privileges of First class citizen like the native apps.
4. Intents
5. Distribution, PWAs are not present in the store.
6. Social networking sites cannot be acting as a discovery mechanism for PWAs. The reason is that a lot of the time in the mobile devices the social networking apps open their own internal browser. These do not support PWAs.
7. Web app installers also could be ineffective as the user has to visit the website twice for getting the PWA install option.

Web App manifest: Eg: <https://www.flipkart.com/manifest.json> (Change to device mode in chrome debug tool)

* Should be based on W3C spec.
* It is a JSON file.
* Should be served as of type application/manifest.json
* name: The name shown on the splash screen.
* Start\_name: The name shown under the icon on the mobile home page.
* Start\_url : The url from where the app should or opened.
* Orientation : any, portrait [-primary | -secondary], landscape [-primary | -secondary], natural(Lets the device decide what is best suitable.)
* Display:
  + Browser : Not actually PWA
  + minimal-ui : Lite version of the app. Browser UI but with little options to edit the options say URL etc. Currently similar as standalone
  + **standalone**: Most web-apps use this for mobile app experience.
  + fullscreen: Mostly games where we miss the battery status, top bar of phone.
* background\_color:
* theme\_color: The color of the theme for the app.
* Icon: array of the icons with props
  + Src
  + Sizes: size of the icons for diff platforms.
  + Type

**Service workers:**

* One of the two important core APIs for the PWA apart from manifest.json
* It is a web worker
* It has one scope and abilities over that scope.
* It works detached from any tab on the browser. Has its own lifecycle.
  + **Web Worker**
    - A JS thread.
    - No access to UI or DOM
    - No document , no window.
    - Separate JS file.
  + **Scope**
    - Just an origin(protocol + host (domain) + port) and path
    - Default and Max scope
    - Can be overridden with “Service-Worker-Allowed” flag.
    - Eg: Scope: https://fluent.com/files
* Service worker can see all the resources that pages on the scope request, even cross-origin.
* It can respond for those resources synthesizing responses or fetching them. Acts like a proxy on client side.
* Service worker is independent of the website or the Tab. It has its own lifecycle. We can even debug the code in the service worker.
* Each website can have its service worker with a scope.
* Service worker can send messages to clients using the client connection API.
* The only way to have the browser check for new version of Service worker is to have no-cache tag in the HTTP header in the js file for the service worker.
* The service worker has a cache. The cache stores key value pairs of the HTTP request and response.
* The service worker cannot access any synchronous API for Eg: Local storage API.
* The service worker must be served only using TLS (HTTPS), localhost exception.
* Service worker is based on promises.
* The service worker must be in the root folder if it must access all the resources under the domain.
* Each service worker file will be installed only once.
* It is safer to use arrow functions within a service worker
* <https://github.com/GoogleChrome/sw-precache>
* <https://github.com/GoogleChrome/workbox> - Collection of libraries for Service worker.
* Clients API for SW to access the client. It is a part of the PWA running in the browser.
  + claim().then()
  + openWindow()
  + matchAll(options).then
    - includeuncontrolled
  + focus()
  + navigate(url).then()
  + Many other options.
* navigator.onLine = false means we are offline. If true then the OS is telling that we are online. But we cannot tell if we are connected to the internet. The best way is to try to ping to the server.
* For “activate” event to fire on a SW the previous version of the SW must be idle for some time defined by the browser. Then when we invoke the SW the activate event is fired.
* If any of the external CDN based or 3rd party urls are being used say JQuery, it is also better to add all the URLs and the dependency URLs injected into the cache. Just be sure of Offline access. In our example Google fonts might have URLs which we add.
* The PWA has a scope that is different from a SW scope. The scope of PWA defines how external URLs would be handled. Eg: If PWA scope is <https://fulent.com/pwa> and if we access <https://fluent.com/hybrid> the later is opened in another browser tab or window out of the scope of PWA.
* registration.update can be used to update the SW.
* a 404 is not a SW update.
* Manually unregister must be invoked to remove the existing SW.
* We can remove cookies.
* We can get only changed part of the file from the server using the SW
* We can add out Custom Headers to help the server . Eg: Client Hints
* Create Middlewares in SW
* Image progressive load. Showing the low resolution image first and then show the actual resolution image.
* BPG image form <img src=”image.bpg.png”>. SW will download the bpg file format and then serve the file as a png format.

**Use cases of Service worker:**

* Offline web. Similar to an Apache Server. Service worker would act as an proxy that handles the logic at the client end.
* PWA can be effectively built using service worker.
* Deliver assets immediately from cache. Or provide local resources faster to the user.
* Background update of resources and assets asynchronously.
* React to bad n/w connections. If no updated resources is returned then render the cached version.
* Better response to non-200 responses.
* Prefetching resources and assets before time.
* Local content generation.

**FetchAPI()** :

* Replaces XML/HTTP request.
* It is promise based.
* It can return responses in blob(), json(), arrayBuffer()
* fetch (url, options)
  + method: “GET” ,”POST”
  + headers[]
  + cors, no-cors
  + credentials: omit, include
  + cache: default, no-store, reload
  + redirect: manual, follow, error(throws catch)

**Background Sync:**

* Optimistic saves – Saving content in background. Gmail does not send large mails instantly, it goes to outbox and then sends it.
* Since SW has its own lifecycle, the saving can be done in background.
* Registration.sync.register(‘<unique label or name>Eg:save’).then(function () {

//Sync success

}

* In desktop Brower if we close the browser process then the background sync does not happen which is not the case in Android.
* SyncManager is the API for background sync in window.

**Replacing the SW:**

* app shell caching can be used in iOS for

**Web APK:**

* Chrome has a Web APK that allows the PWA to be installed as a native APP. In this PWA we could make use of Intents and other native features.
* This is how we could eliminate multiple instances of the PWA on the device.
* Currently a chrome Beta
* Permission and OS integration.
* Home screen and App launcher icons
* Browser will do this automatically.
* Microsoft has PWA Generator (<https://www.pwabuilder.com/>) for windows apps.

**Web Push Notifications:**

* Keep the user updated
* Stages
  + Web push Subscription
  + Web push delivery
* Firebase is the only tool that supports push notifications over web.
* Push Subscription
  + Push API
    - Latest spec: with Voluntary Application Server Identification(VAPID)
    - Old spec: With payload
    - Service Workers (HTTPS)
    - Visible notifications only
    - Has an endpoint & optional key to encrypt the payload data on server side.
  + Safari Push notification for website
    - Apple Dev account ($99/year)
    - HTTPS
    - Generate Certificate
* The client side is only responsible for requesting permission and subscribing to notifications. The server side would have to push notifications to the URL with the corresponding private key. Eg: <https://fcm.googleapis.com/fcm/send/dNVMjARTzNc:APA91bFccX_5WMRHUNZzlA-2lb6…SrLsauqlHcTfio7R5Fpi0g9s9XB8zMTtPJwzUix59vbqMrUaXqykyC2oUOhrub8BSzKX-zspPz>

Was the endpoint created in the demo.

**Offline Storage APIs**

* Web Storage(local/session): Persist 5 Mb-(2.5 MB) string only
* Web SQL: Chrome/ safari only < 50 MB – SQL DB(SQL lite)
* IndexedDB: persist < 50 MB – No SQL DBs, JS objects
* FileManager: Files in chrome
* Cache storage API (SW): no limits, request/responses key value pairs
* Application cache: 50 MB, responses.

## Web Performance

**Measuring the website performance:**

* Page load does not serve as an ideal performance measure.
* Speed index is better factor to web performance. It checks the progress of the website in loading the content from start to finish.
* If a PWA does not respond within 100ms then it is slow. The user sees it is a failure.
* If the wait time for any event is 1 sec, then the user is not engaged.
* RAIL metrics from Google is also a factor.
* Javascript will cause the browser to stop parsing the HTML.
* The CSS will cause the browser from rendering the HTML until it parses al the CSS files.

**Tips for better performance:**

* CSS must be provided ASAP to the browser.
* Javascript must be delayed as much as possible.
* Optimizing images by compressing them etc.
* Have a HTTP cache policy.
* Try serving the zip version of the files from the server.
* Avoiding redirects
* Working with TLS/HTTP2.
* APM and Facebook instant articles are frameworks for serving pages faster in Mobile app.

**Hack First Load: Initial load for Mobile view**

* Deliver Above the Fold(ATF), i.e. the content that fits in the phone view before scrolling, in 14Kb in 1 TCP packet.
* Embed all the CSS and JS needed.
* If space embed logo or low res images.
* Announce DNS queries ASAP or before time.
* Avoid http to https redirect. Instead use HSTS(HTTP strict Transport Security) in the Header. This will case the browser to remember your protocol.
* Opt in at hstspreload.org for whitelisting your URL redirect from HTTP to HTTPS
* Save 4-8% data-transfer using Zopfli. Takes time to compress, but that is on server side.
* Use Brotli to save 14% data transfer. Check encoding header to check support of Brotli.
* Create Low-res version by checking save-data Client Hint header. NetInfo or Performance training API could also be used.
* Use <picture> tag to have different version of images for different browsers.
* Decode images in the background.
* Use animated PNGs or animated WebP instead of GIFs. Or use muted videos( Can save 60% of size)

**Notes for iOS:**

* <meta name="apple-mobile-web-app-capable" content="yes"> add this for iOS devices.
* In iOS the PWAs are not safari processes. They are stored as process with extension “app”. iOS 11 would have better support.
* <meta name="apple-mobile-web-app-title" content="Fluent"> to add a name for the icon.
* <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent"> along with the header tag and css styles can be used to emulate the theme property of manifest.json file in iOS
* In iOS the PWAs are reloaded every time we open the app.
* <link rel="apple-touch-icon" href="icon\_ios.png"> to add the icon for PWA in iOS devices like iPhone and iPad.

Useful Links

Slides: <https://drive.google.com/file/d/0B5fmXjzTaalqS1cweTNyTkZKX28/view?usp=sharing>

PWA Generator

<https://www.pwabuilder.com/>

Web App Manifest Generators

<https://tomitm.github.io/appmanifest/>

<https://app-manifest.firebaseapp.com/>

<https://brucelawson.github.io/manifest/>

Icon Generators

<http://appicon.build/>

[https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html](https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html#foreground.type=clipart&foreground.clipart=android&foreground.space.trim=1&foreground.space.pad=0.25&foreColor=rgba(96%2C%20125%2C%20139%2C%200)&backColor=rgb(68%2C%20138%2C%20255)&crop=0&backgroundShape=square&effects=none&name=ic_launcher)

Service Workers Recipes

<https://serviceworke.rs>

<https://mdn.mozillademos.org/files/12638/sw101.png>

Web Push Demos

Book: <https://web-push-book.gauntface.com>

Web Push NPM library: <https://github.com/web-push-libs/web-push>

Firebase <https://github.com/firebase/quickstart-js/tree/master/messaging>

Code Snippets

### **iOS Meta Tags**

<meta name='viewport' content='width=device-width,initial-scale=1'>

<meta name="apple-mobile-web-app-capable" content="yes">

<meta name="apple-mobile-web-app-title" content="Fluent 2017">

<meta name="apple-mobile-web-app-status-bar-style"

     content="black-translucent">

<link rel="apple-touch-icon" href="icon\_ios.png">

### **Service Workers Snippets**

**Registration**

if ('serviceWorker' in navigator) {

   navigator.serviceWorker.register("serviceworker.js")

       .then(function(registration) {

   // Worker is registered

   }).catch(function(error) {

   // There was an error registering the SW

   });

}

**Basic Add to Cache**

var urls = [];

self.addEventListener("install", function(event) {

   console.log("The SW is now installed");

   event.waitUntil(caches.open("myAppCache").then(function(cache) {

       return cache.addAll(urls);

   }));

});

**Fetch**

self.addEventListener("fetch", function(event) {

   event.respondWith(caches.match(event.request)

       .then(function(response) {

           if (response) {

               // The request is in the cache

               return response;

           } else {

               // We need to go to the network

               return fetch(event.request);

           }

       })

   );

});

**Fetch: Stale while Revalidate**

self.addEventListener('fetch', function(event) {

   event.respondWith(

       caches.match(event.request)

           .then(function(response) {

               // Even if the response is in the cache, we fetch it

               // and update the cache for future usage

               var fetchPromise = fetch(event.request).then(

                   function(networkResponse) {

                       caches.put(event.request, networkResponse.clone());

                       return networkResponse;

                   });

               // We use the currently cached version if it's there

               return response || fetchPromise;

           })

       );

   });

**Fetch: Network first (server or connection errors)**

self.addEventListener('fetch', function(event) {

   event.respondWith(

     fetch(request).catch(function(error) {

       return caches.open(myAppCache).then(function(cache) {

         return cache.match(request);

       });

     })

   );

 });

**Deletion**

self.addEventListener('activate', function(event) {

 // Array of cache that we will use in this version

 var cacheWhitelist = ['pages-cache-v1', 'blog-posts-cache-v1'];

 event.waitUntil(

   caches.keys().then(function(cacheNames) {

     return Promise.all(

       cacheNames.map(function(cacheName) {

         if (cacheWhitelist.indexOf(cacheName) === -1) {

           // Deletes the cache because we won't use it here

           return caches.delete(cacheName);

         }

       })

     );

   })

 );

});

**Forcing activation and claiming clients**

self.addEventListener("install", function(event) {

   self.skipWaiting();

})

self.addEventListener("activate", function(event) {

   // oncontrollerchange will be triggered on the client(s)

   event.waitUntil(self.clients.claim());

});

**Emulating a Web Server**

self.addEventListener("fetch", function(event) {

   var url = event.request.url;

   var body = "";

   var mockResponse = new Response(body, {

     status: 200,

     statusText: 'OK',

     headers: {

       'Content-Type': 'text/html'

     }

   });

   event.respondWith(mockResponse);

 });

**Receiving messages from Service Workers in the Web Page**

navigator.serviceWorker.addEventListener("message",

     // General message received from the SW

     m => console.log(m.data)

);

**Sending messages to a Service Worker from the Web Page**

function sendMessageToSW(message){

   return new Promise(function(resolve, reject){

       // Create a Message Channel that will transport messages

//Channel has 2 ports for sending and receiving respectively.

       var channel = new MessageChannel();

       // Handler for receiving reply from service worker

       channel.port1.onmessage = function(event){

           if (event.data.error) {

               reject(event.data.error);

           } else {

               resolve(event.data);

           }

       };

       // Send message to service worker along with port for reply

       navigator.serviceWorker.controller.postMessage(message, [channel.port2]);

   });

}

**Receiving messages in the SW**

self.addEventListener("message", event => {

   // Sending a response back through the Channel received

   event.ports[0].postMessage("Message received: " + event.data);

});

**Broadcasting messages to all clients**

function sendMessage(client, message){

   return new Promise(function(resolve, reject){

       var channel = new MessageChannel();

       channel.port1.onmessage = function(event){

           if(event.data.error){

               reject(event.data.error);

           }else{

               resolve(event.data);

           }

       };

       client.postMessage(message, [channel.port2]);

   });

}

function broadcast(message){

   clients.matchAll().then(clients => {

       clients.forEach(client => {

           sendMessage(client, message);

       })

   })

}

**Check network status**

self.addEventListener('fetch', function(event) {

    var isOnline = navigator.onLine;

    // TODO: Check is navigator.connection is available as it might only work in Android.

    var isOnCellular = navigator.connection.type=="celullar";

    var is2G = navigator.connection.downlinkMax < 0.5;

    var is3G = navigator.connection.downlinkMax < 15;

    var is4G = navigator.connection.downlinkMax >= 15;

});

**Unregistering all Service Workers**

navigator.serviceWorker.getRegistrations().then(function(registrations) {

    for(let registration of registrations) {

        // You can check registration.active.scriptURL

    registration.unregister();

    }

});

**Background Sync Detection**

if ('SyncManager' in window) {

}

**Register a Background Sync**

 //Wait until the SW is ready. Once it is ready the then function is called.

navigator.serviceWorker.ready.then(function(registration) {  
 return registration.sync.register('mySync');  
});

## Executing a Background Sync

self.addEventListener('sync', function(event) {  
 if (event.tag == 'mySync') {  
   event.waitUntil(doSomeStuff());  
 }  
});

## **Web Push Snippets**

### **Get VAPID keys**

npm install web-push -g

web-push generate-vapid-keys

### **Client-side Web Push code**

**Detecting availability**

if (!('PushManager' in window)) {

  // Push not available

  return;

}

**Ask for Push Notification Permission**

function pushAskPermission() {

 return new Promise(function(resolve, reject) {

   const permissionResult = Notification.requestPermission(function(result) {

     resolve(result);

   });

   if (permissionResult) {

     permissionResult.then(resolve, reject);

   }

 })

 .then(function(permissionResult) {

   if (permissionResult !== 'granted') {

     throw new Error('We weren\'t granted permission.');

   }

 });

}

**Subscribe a User**

function pushSubscribeUser() {

 navigator.serviceWorker.getRegistration().then(

     function(registration) {

       const subscribeOptions = {

           userVisibleOnly: true,

           applicationServerKey: urlBase64ToUint8Array(

               '<REPLACE WITH PUBLIC KEY>'

           )

       };

       return registration.pushManager.subscribe(subscribeOptions);

 })

 .then(function(pushSubscription) {

   console.log('Push Subscription: ', JSON.stringify(pushSubscription));

   return pushSubscription;

 });

}

// Snippet from https://www.npmjs.com/package/web-push

function urlBase64ToUint8Array(base64String) {

 const padding = '='.repeat((4 - base64String.length % 4) % 4);

 const base64 = (base64String + padding)

   .replace(/\-/g, '+')

   .replace(/\_/g, '/');

 const rawData = window.atob(base64);

 const outputArray = new Uint8Array(rawData.length);

 for (let i = 0; i < rawData.length; ++i) {

   outputArray[i] = rawData.charCodeAt(i);

 }

 return outputArray;

}

**Receiving the Push Notification**

self.addEventListener("push", function(event) {

 if (event.data) {

   console.log('Push Data: ', event.data.text());

 } else {

   console.log('No data :(');

 }

 self.registration.showNotification("Push title", {

       body: event.data.text()

 });

});

**Detect if Actions are available**

if ('actions' in Notification.prototype) {

   const maxVisibleActions = Notification.maxActions;

}

**Notification Click detection**

self.addEventListener('notificationclick', function(event) {

 if (!event.action) {

   console.log('Notification Click with no action');

   return;

 } else {

   // event action has the action id

 }

 self.registration.showNotification("Push title", {

       body: event.data.text()

 });

 event.notification.close();

 event.waitUntil(doSomething);

});

**Notification with all options**

{

 body: "Message String",

 dir: "auto|rtl|ltr",

 actions: [{

         action: 'action-id',

         title: 'Action Title',

         icon: 'https://...48dp'

       }]

 icon: "https://...64dp",

 badge: "https://...24dp",

 image: "https://...",

 //Pattern for phone vibration on notifications

 vibrate: [100,0,100,0,100],

//Sound on notification

 sound: "https://",

 //To add to esxting notification or replace existing one

 tag: "notification-id-tag",

 data: "",

 requireInteraction: true,

//To send the same notification again

 renotify: true,

 silent: false,

}

**NODE.js - Setup Web Push**

const webpush = require('web-push');

const vapidKeys = {

 publicKey: '<YOUR PUBLIC KEY>',

 privateKey: '<YOUR PRIVATE KEY>'

};

webpush.setVapidDetails(

 'mailto:<YOUR EMAIL>',

 vapidKeys.publicKey,

 vapidKeys.privateKey

);

PWA HTML

<!doctype html>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width">

<link href="https://fonts.googleapis.com/icon?family=Material+Icons"

rel="stylesheet">

<link rel="stylesheet" href="style.css">

<link rel="manifest" href="manifest.json">

<script src="launcher.js"></script>

<script>

if ('serviceWorker' in navigator) {

//To register a service worker.

navigator.serviceWorker.register("serviceworker.js", {scope: '/'}) //Defining the scope on the doamin. If we gave /2017 then only the folder 2017 would be accessible under the SW

.then(function(registration) {

// Worker is registered

}).catch(function(error) {

// There was an error registering the SW

});

}

</script>

<title>Max's Activity Launcher</title>

<h1>Max's Activity Launcher</h1>

<ul>

<li><a href="https://maps.google.com/?q=Miami">

<i class="material-icons">map</i>

<p>Miami Map</p>

</a></li>

<li><a href="https://maps.google.com/?q=104+Market+St.+San+Francisco%2C+CA">

<i class="material-icons">map</i>

<p>Work Place Map</p>

</a></li>

<li><a href="https://maps.apple.com/?q=104+Market+St.+San+Francisco%2C+CA">

<i class="material-icons">map</i>

<p>Work Place Map</p>

</a></li>

<li><a href="https://maps.apple.com/?daddr=104+Market+St.+San+Francisco%2C+CA">

<i class="material-icons">directions</i>

<p>Drive to Work (AM)</p>

</a></li>

<li><a href="https://maps.google.com/?daddr=104+Market+St.+San+Francisco%2C+CA">

<i class="material-icons">directions</i>

<p>Drive to Work (GM)</p>

</a></li>

<li><a href="https://maps.google.com/?saddr=Big+Ben&daddr=Hyde+Park%2C+London&dirflg=w">

<i class="material-icons">directions\_walk</i>

<p>Walk to the Park</p>

</a></li>

<li><a href="https://maps.apple.com/?saddr=Big+Ben&daddr=Hyde+Park%2C+London&dirflg=r">

<i class="material-icons">directions\_transit</i>

<p>Transit to the Park</p>

</a></li>

<li><a href="https://m.uber.com/ul/?action=setPickup&pickup=my\_location">

<i class="material-icons">local\_taxi</i>

<p>Call an Uber</p>

</a></li>

<li><a href="sms:12345678?body=I%27m+late">

<i class="material-icons">message</i>

<p>Text Wife</p>

</a></li>

<li><a href="whatsapp://send?text=I%27m+late">

<i class="material-icons">message</i>

<p>Whatsapp Wife</p>

</a></li>

<li><a href="tg:msg?text=I%27m+late">

<i class="material-icons">message</i>

<p>Telegram Friend</p>

</a></li>

<li><a href="fb-messenger://share?link=http://">

<i class="material-icons">message</i>

<p>Send by Messenger</p>

</a></li>

<li><a href="tel:+541145454545">

<i class="material-icons">call</i>

<p>Call Mom</p>

</a></li>

<li><a href="facetime:user@domain.com">

<i class="material-icons">video\_call</i>

<p>Facetime John</p>

</a></li>

<li><a href="facetime-audio:user@domain.com">

<i class="material-icons">call</i>

<p>Audio Facetime John</p>

</a></li>

<li><a href="skype:mary89?call">

<i class="material-icons">call</i>

<p>Skype Mary</p>

</a></li>

<li><a href="skype:mary89?call&video=true">

<i class="material-icons">video\_call</i>

<p>Video Skype Mary</p>

</a></li>

<li><a href="https://twitter.com/intent/tweet">

<i class="material-icons">add</i>

<p>New Tweet</p>

</a></li>

<li><a href="snapchat://">

<i class="material-icons">camera</i>

<p>Start Snapchat</p>

</a></li>

</ul>

## **styles.css**

   body {

       font-family: Verdana, Geneva, Tahoma, sans-serif;

   }

   h1 {

       font-size: large;

       text-align: center;

   }

   ul {

       list-style: none;

       padding: 0;

       text-align: center;

   }

   li {

       margin: 10px;

       padding: 0px;

       background-color: #ddd;

       float: left;

   }

   .material-icons {

       font-size: 50px;

   }

   li p {

       font-size: small;

   }

   a {

       width: 70px;

       height: 120px;

       display: block;

       padding: 10px;

       text-decoration: none;

   }