**PROGRESSIVE WEB PAGE:**

**INDEX.HTML**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sky-High Airport Arrivals</title>

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

<link href="https://fonts.googleapis.com/css?family=Roboto:300,600,300italic,600italic" rel="stylesheet" type="text/css">

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

</head>

<body>

<header>

<div class="content">

<h3>Arrivals</h3>

</div>

</header>

<div class="container">

<div id="main" class="content">

<ul class="arrivals-list" data-bind="foreach: arrivals">

<li class="item">

<span class="title" data-bind="html: title"></span>

<span class="status" data-bind="html: status"></span>

<span class="time" data-bind="html: time"></span>

</li>

</ul>

</div>

</div>

<script src="./js/build/vendor.min.js"></script>

<script src="./js/build/script.min.js"></script>

</body>

</html>

**PAGE.JS**

(var Page = (function() {

// declare the view model used within the page

function ViewModel() {

var self = this;

self.arrivals = ko.observableArray([]);

}

// expose the view model through the Page module

return {

vm: new ViewModel(),

hideOfflineWarning: function() {

// enable the live data

document.querySelector(".arrivals-list").classList.remove('loading')

// remove the offline message

document.getElementById("offline").remove();

// load the live data

},

showOfflineWarning: function() {

// disable the live data

document.querySelector(".arrivals-list").classList.add('loading')

// load html template informing the user they are offline

var request = new XMLHttpRequest();

request.open('GET', './offline.html', true);

request.onload = function() {

if (request.status === 200) {

// success

// create offline element with HTML loaded from offline.html template

var offlineMessageElement = document.createElement("div");

offlineMessageElement.setAttribute("id", "offline");

offlineMessageElement.innerHTML = request.responseText;

document.getElementById("main").appendChild(offlineMessageElement);

} else {

// error retrieving file

console.warn('Error retrieving offline.html');

}

};

request.onerror = function() {

// network errors

console.error('Connection error');

};

request.send();

}

}

})();

**WEB PAGE MANIFEST:**

**MANIFEST.JSON :**

{

"short\_name": "Arrivals",

"name": "Arrivals at Sky High",

"description": "Progressive web application demonstration",

"icons": [

{

"src": "launcher-icon.png",

"sizes": "48x48",

"type": "image/png"

},

{

"src": "launcher-icon-96.png",

"sizes": "96x96",

"type": "image/png"

},

{

"src": "launcher-icon-144.png",

"sizes": "144x144",

"type": "image/png"

},

{

"src": "launcher-icon-192.png",

"sizes": "192x192",

"type": "image/png"

},

{

"src": "launcher-icon-256.png",

"sizes": "256x256",

"type": "image/png"

}

],

"start\_url": "./?utm\_source=web\_app\_manifest",

"display": "standalone",

"orientation": "portrait",

"theme\_color": "#29BDBB",

"background\_color": "#29BDBB"

}

**SW.JS**

// Use a cacheName for cache versioning

var cacheName = 'v1:static';

// During the installation phase, you'll usually want to cache static assets.

self.addEventListener('install', function(e) {

// Once the service worker is installed, go ahead and fetch the resources to make this work offline.

e.waitUntil(

caches.open(cacheName).then(function(cache) {

return cache.addAll([

'./',

'./css/style.css',

'./js/build/script.min.js',

'./js/build/vendor.min.js',

'./css/fonts/roboto.woff',

'./offline.html'

]).then(function() {

self.skipWaiting();

});

})

);

});

// when the browser fetches a URL…

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

// … either respond with the cached object or go ahead and fetch the actual URL

event.respondWith(

caches.match(event.request).then(function(response) {

if (response) {

// retrieve from cache

return response;

}

// fetch as normal

return fetch(event.request);

})

);

});

**REGISTER IN MAIN.TS**

// Register the service worker if available.

if ('serviceWorker' in navigator) {

navigator.serviceWorker.register('./sw.js').then(function(reg) {

console.log('Successfully registered service worker', reg);

}).catch(function(err) {

console.warn('Error whilst registering service worker', err);

});

}

window.addEventListener('online', function(e) {

// Resync data with server.

console.log("You are online");

Page.hideOfflineWarning();

Arrivals.loadData();

}, false);

window.addEventListener('offline', function(e) {

// Queue up events for server.

console.log("You are offline");

Page.showOfflineWarning();

}, false);

// Check if the user is connected.

if (navigator.onLine) {

Arrivals.loadData();

} else {

// Show offline message

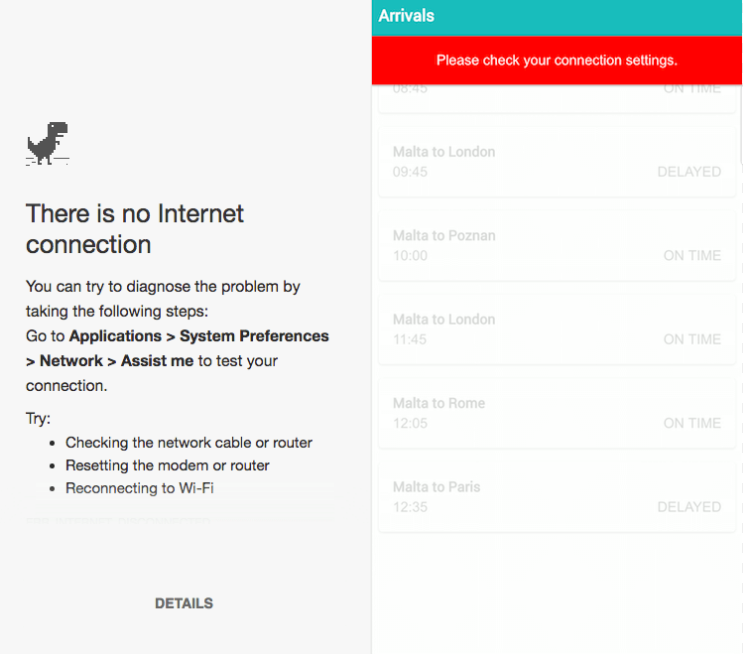
Page.showOfflineWarning();

}

// Set Knockout view model bindings.

ko.applyBindings(Page.vm);

**OFFLINE TEST:**

****