Progressive Web Application

V0.9

Ben Jones

15/8/2023

[Introduction to PWA: 2](#_Toc142980309)

[Setup your development environment: 2](#_Toc142980310)

[Initialize NodeJs application 3](#_Toc142980311)

[Setup Files and folders for your project: 3](#_Toc142980312)

[Build and test query your database 4](#_Toc142980313)

[Setup your Express Webserver 4](#_Toc142980314)

# Introduction to PWA:

A [Progressive Web Apps (PWAs)](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps?ref=arctype.com) is an app that's built using web platform technologies, but that provides a user experience like that of a platform-specific app.

Like a website, a PWA can run on multiple platforms and devices from a single codebase. Like a platform-specific app, it can be installed on the device, can operate while offline and in the background, and can integrate with the device and with other installed apps.

## Technical features of PWAs

Because PWAs are websites, they have the same basic features as any other website: at least one HTML page, which very probably loads some CSS and JavaScript. Like a normal website, the JavaScript loaded by the page has a global Window object and can access all the Web APIs that are available through that object.

### Beyond that, a PWA has some additional features:

* A web app manifest file, which, at a minimum, provides information that the browser needs to install the PWA, such as the app name and icon.
* A service worker, which, at a minimum, provides a basic offline experience.

Read more about how a PWA is different to a traditional website or a platform-specific website [here](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/What_is_a_progressive_web_app).

# Setup your development environment:

The development process for this example PWA has been specifically designed to demonstrate specific content points from the NESA Software Engineering 11-12 Syllabus specifically:

* Model elements that form a web development system
* Investigate cascading style sheets (CSS) and its impact on the design of a web application
* Investigate methods to support and manage the load times of web pages/applications
* Observe and describe the back-end process used to manage a web request
* Develop a web application using an appropriate scripting language with shell scripts to make files and directories, and searching for text in a text file
* Apply a web-based database and construct script that executes SQL
* Design, develop and implement a progressive web app (PWA)

The following applications are required as a minimum to complete this task.

|  |
| --- |
| Install Visual Studio Code |
| Install Node.Js   1. https://nodejs.org/en/download |
| Install VSCode Extensions   1. <https://marketplace.visualstudio.com/items?itemName=alexcvzz.vscode-sqlite> 2. <https://marketplace.visualstudio.com/items?itemName=McCarter.start-git-bash>   Window users only:   1. <https://marketplace.visualstudio.com/items?itemName=medo64.render-crlf>   If you are new to VSCode or don’t know how to install extensions use [this guide](https://code.visualstudio.com/docs/introvideos/basics). |

*Note: VSCode & Node.JS are available in the eT4L software catalogue*

# Establish a working directory

1. Start by selecting an appropriate folder location that is either version controlled or backed up to the cloud.
2. Open VSCode Ctrl + K Ctrl + O
3. Navigate to and open your working folder

# Initialize Node.Js application

To get started, we'll create our project folder with the #bash command below. Make sure you open a new terminal Ctrl + ` and choose Git Bash from the menu option in the top right of the terminal shell.

A screenshot of a computer

Description automatically generated

Start by making a folder with the following #BASH commands

mkdir myPWA && cd myPWA

Then, we'll initialize a Node.js application with the commands below:

npm init -y

The above command creates and configures a package.json file for the application.

# Setup Files and folders for your project:

|  |  |
| --- | --- |
| Create a directory for development files and documentation  mkdir .workingDocuments  Create a license file  touch LICENSE  Edit the LICENSE document and paste the text from <https://www.gnu.org/licenses/gpl-3.0.txt> in to it.  Create a directory for your database and SQL files  mkdir .database  Create a public directory for your front-end files with the #bash command below:  mkdir public && cd public  Now using #BASH commands and scripts setup a template folder structure for a PWA  touch files.txt  touch folders.txt  Inside the file list the files you want to routinely generate:   1. style.css 2. index.html 3. index.js 4. app.js 5. serviceWorker.js   Then create a file called folders.txt and in the file list the folders you want to routinely generate:   1. css 2. icons 3. images 4. js   Write a simple bash script to read the folders.txt document and create a directory for each line  while read -r line; do  echo $line  mkdir -p $line  done < folders.txt  Write a simple bash script to read the files.txt document and create a file for each line  while read -r line; do  echo $line  touch $line  done < files.txt | Watch this video for more #BASH knowledge: <https://www.youtube.com/watch?v=tK9Oc6AEnR4>  Common #BASH commands: <https://www.educative.io/blog/bash-shell-command-cheat-sheet>  Notes:   1. *The dot .file instructs the webserver to not serve the folder or contained files securing the data from being accessed via http adding an additional layer of secure architecture.* 2. *The Blank line is required to give the last line a line ending and allow the script below to read all lines.* 3. *If you are using a DOS (windows) environment with a bash emulator in VSCode the line ending will be incorrect you will need to use an online converter to replace the DOS line endings with Unix line endings such as* [*https://app.execeratics.com/LFandCRLFonline/?l=en*](https://app.execeratics.com/LFandCRLFonline/?l=en) *you can see your line endings using the optional extension listed in the setting up your environment section.* |

# Build and test query your database

|  |  |
| --- | --- |
| Install SQLite3 via NPM at the #BASH command  npm install --save sqlite3  Go to the .database folder and create your database files  touch datasource.db  touch myQuery.sql  Edit the myQuery.sql file and write the query to create the table  CREATE TABLE extension(extID INTERGER NOT NULL PRIMARY KEY,name TEXT NOT NULL, hyperlink TEXT NOT NULL,about TEXT NOT NULL,image TEXT NOT NULL,language TEXT NOT NULL);  Run the query by right click over the query then choose “Run Selected Query” and choose datasource.db when ask this will associate all queries in myQuery.sql with the database datasource.db.  Write a template for inserting values  -- INSERT INTO extension(extID,name,hyperlink,about,image,language) VALUES (X,"","","","",""); -- Insert template  Then copy and paste it filling in the insert values to manually add content to your database  INSERT INTO extension(extID,name,hyperlink,about,image,language) VALUES (1,"Live Server","https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer","Launch a development local Server with live reload feature for static & dynamic pages","https://ritwickdey.gallerycdn.vsassets.io/extensions/ritwickdey/liveserver/5.7.9/1661914858952/Microsoft.VisualStudio.Services.Icons.Default","HTML CSS JS");  Once you have built your Database test it with some different QUERYs for example:  SELECT \* FROM extension;  SELECT \* FROM extension WHERE language LIKE '#BASH'; | Watch this video to see how SQL Queries and SQLite3 works in VSCode: <https://www.youtube.com/watch?v=JrAiefGNUq8>  *Note: after each SQL query, students should comment out -- the query so there is a running log of SQL queries.* |

# Setup your Express Webserver

|  |  |
| --- | --- |
| In a BASH terminal setup a node.js file that will hold all the back-end js code, this file should be located in your application directory root.  cd .. && touch index.js  Use BASH to install the Express webserver via npm  npm install express | Note:  cd.. is assuming you are working in the pubic folder  Read the Node.js Express documentation [here](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/development_environment). |
| Add the following JS to your index.js file  const express = require("express");  const path = require("path");  const app = express();  app.use(express.static(path.join(\_\_dirname, "public")));  app.get("/", function (req, res) {  res.sendFile(path.join(\_\_dirname, "public/index.html"));  });  app.listen(8000, () => console.log("Server is running on Port 8000, visit <http://localhost:8000/> or <http://127.0.0.1:8000> to access your website") ); | In this script, we import express to create our server and the path module. We configured our app to render our static files using the express static method, which takes the path to the static folder (public), we created the root route of our application and rendered the index.html file. Then we configured the app to listen to port 8000. |

Extract your SQL in the backend to a JSON in the front end

|  |  |
| --- | --- |
| Add the following JS to your index file above the express webserver script. So it is executed before the weber is started.  const sqlite3 = require('sqlite3').verbose();  const db = new sqlite3.Database('.database/dataSource.db');  let myString = '[\n';  db.all("SELECT \* FROM extension", function(err, rows) {  let myCounter = 0;  rows.forEach(function (row) {  // console.log(row.extID + ": " + row.name + ": " + row.hyperlink + ": " + row.about + ": " + row.image + ": " + row.language);  myString = myString + '{\n"extID":' + row.extID + ',\n"name":"' + row.name + '",\n"hyperlink":"' + row.hyperlink + '",\n"about":"' + row.about + '",\n"image":"' + row.image + '",\n"language":"' + row.language;  myCounter++;  if (myCounter == rows.length) {  myString = myString + '"\n}\n';  } else {  myString = myString + '"\n},\n';  }  });  // console.log(myString);  var fs = require('fs');  fs.writeFile("public/frontEndData.json", myString + "]", function(err) {  if (err) {  console.log(err);  }  });  }); | Students should play with writing algorithms to construct a JSON data set from a database. You can test your algorithm with a JSON validation tool like https://jsonlint.com/ |

Render your front end

|  |  |
| --- | --- |
| Design a simple square 1080px X 1080px logo: using Canva or Photoshop  Design simplified icon 1080px X 1080px  Export both files as a PNG  Save design and original export files to the .workingfiles directory  Optimize your logo using: <https://tinypng.com/> and save the optimised versions to public/images | |
| Core HTML | <!DOCTYPE html>  <html lang="en">    <head>      <meta charset="UTF-8" />      <meta name="viewport" content="width=device-width, initial-scale=1.0" />      <meta http-equiv="X-UA-Compatible" content="ie=edge" />      <meta http-equiv="Content-Security-Policy" content="script-src 'self';">      <link rel="stylesheet" href="css/style.css" />      <title>VSCode Extension Catalogue</title>      <link rel="manifest" href="manifest.json" />      <link rel="icon" type="image/x-icon" href="/images/favicon.png">    </head>    <body>      <script src="js/app.js"></script>    </body>  </html>  Note the metatag to prevent cross site scripting. |
| Style the core | @import url("https://fonts.googleapis.com/css?family=Nunito:400,700&display=swap");  \* {  margin: 0;  padding: 0;  box-sizing: border-box;  }  body {  background: #fdfdfd;  font-family: "Nunito", sans-serif;  font-size: 1rem;  }  main {  max-width: 900px;  margin: auto;  padding: 0.5rem;  text-align: center;  } |
| Add Navigation | <!DOCTYPE html>  <html lang="en">    <head>      <meta charset="UTF-8" />      <meta name="viewport" content="width=device-width, initial-scale=1.0" />      <meta http-equiv="X-UA-Compatible" content="ie=edge" />      <link rel="stylesheet" href="css/style.css" />      <title>VSCode Extension Catalogue</title>      <link rel="manifest" href="manifest.json" />      <link rel="icon" type="image/x-icon" href="/images/favicon.png">    </head>    <body>      <main>        <nav>          <img src="images\logo.png"/>          <h1>VSCode Extensions</h1>          <ul class="topnav">            <li><a href="#">Home</a></li>            <li><a href="about.html">About</a></li>            <li><a href="suggest.html">Suggest</a></li>          </ul>        </nav>      </main>      <script src="js/app.js"></script>    </body>  </html> |
| Style navigation | nav {      display: flex;      justify-content: space-between;      align-items: center;  }  nav img {      height:100px;  }  nav ul {      list-style: none;      display: flex;  }  nav li {      margin-right: 1rem;    }  nav ul li a {      text-decoration-line:none;      text-transform: uppercase;      color: #393b45;    }  nav ul li a:hover {      color: #14E6DD;  }  nav h1 {      color: #106D69;      margin-bottom: 0.5rem;  } |
| Add the div container that will be scripted into | <!DOCTYPE html>  <html lang="en">    <head>     <meta charset="UTF-8" />     <meta name="viewport" content="width=device-width, initial-scale=1.0" />     <meta http-equiv="X-UA-Compatible" content="ie=edge" />     <meta http-equiv="Content-Security-Policy" content="script-src 'self';" />     <link rel="stylesheet" href="css/style.css" />     <title>VSCode Extension Catalogue</title>     <link rel="manifest" href="manifest.json" />     <link rel="icon" type="image/x-icon" href="images\favicon.png" />     <meta name="theme-color" content="#14E6DD" />    </head>    <body>      <main>        <nav>          <img src="images\logo.png"/>          <h1>VSCode Extensions</h1>          <ul class="topnav">            <li><a href="#">Home</a></li>            <li><a href="about.html">About</a></li>            <li><a href="suggest.html">Suggest</a></li>          </ul>        </nav>        <div class="container">          </div>      </main>      <script src="js/app.js"></script>    </body>  </html> |
| Style the grid | .container {  display: grid;  grid-template-columns: repeat(auto-fit, minmax(15rem, 1fr));  grid-gap: 1rem;  justify-content: center;  align-items: center;  margin: auto;  padding: 1rem 0;  }  .card {  display: flex;  align-items: center;  flex-direction: column;  width: 15rem auto;  height: 15rem;  background: #fff;  box-shadow: 0 10px 20px rgba(0, 0, 0, 0.19), 0 6px 6px rgba(0, 0, 0, 0.23);  border-radius: 10px;  margin: auto;  overflow: hidden;  }  .card--avatar {  width: 100%;  height: 10rem;  object-fit: cover;  }  .card--title {  color: #222;  font-weight: 700;  text-transform: capitalize;  font-size: 1.1rem;  margin-top: 0.5rem;  }  .card--link {  text-decoration: none;  background: #db4938;  color: #fff;  padding: 0.3rem 1rem;  border-radius: 20px;  } |
| Construct the JSON as HTML in the JS/app.js | let result = "";  fetch('./frontEndData.json')    .then(function (response) {      return response.json();    })    .then(function (data) {      appendData(data);    })    .catch(function (err) {      console.log('error: ' + err);    });    function appendData(data) {      data.forEach(({ name, image, hyperlink, about, language } = rows) => {        result += `         <div class="card">              <img class="card-image" src="${image}" alt="Product image for the ${name} VSCode extension."/>              <h1 class="card-name">${name}</h1>              <p class="card-about">${about}</p>              <a class="card-link" href="${hyperlink}"><button class="btn">Read More</button></a>          </div>         `;      });      document.querySelector(".container").innerHTML = result;    } |

Finish PWA Core

|  |  |
| --- | --- |
|  |  |
| Construct the manifest.json which describes a PWA, customize its appearance, and more deeply integrate it into the operating system.  Read more about the manifest [here in the documentation](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps#web_app_manifest).  The manifest will obviously list all the icon sets and it can be manually written or student can use node.js or python to construct it based on files located in public/icons. | {      "name": "VSCode Extension Catalogue",      "short\_name": "vscodeextcat",      "start\_url": "/",      "display": "standalone",      "background\_color": "#fdfdfd",      "theme\_color": "#14E6DD",      "orientation": "portrait-primary",      "icons": [        {          "src": "/icons/icon-128x128.png",          "type": "image/png",          "sizes": "128x128",          "purpose": "maskable"        },        {          "src": "/icons/icon-128x128.png",          "type": "image/png",          "sizes": "128x128",          "purpose": "any"        },        {          "src": "/icons/icon-192x192.png",          "type": "image/png",          "sizes": "192x192",          "purpose": "maskable"        },        {          "src": "/icons/icon-192x192.png",          "type": "image/png",          "sizes": "192x192",          "purpose": "any"        },        {          "src": "/icons/icon-384x384.png",          "type": "image/png",          "sizes": "384x384",          "purpose": "maskable"        },        {          "src": "/icons/icon-384x384.png",          "type": "image/png",          "sizes": "384x384",          "purpose": "any"        },        {          "src": "/icons/icon-512x512.png",          "type": "image/png",          "sizes": "512x512",          "purpose": "maskable"        },        {          "src": "/icons/icon-512x512.png",          "type": "image/png",          "sizes": "512x512",          "purpose": "any"        }      ]    } |
| Construct the service worker which is the brains of a PWA, it handles caching, API interfacing and any offline functionality. | const assets = [      "/",      "css/style.css",      "js/app.js",      "/images/logo.png",      "/images/blog2.jpg",      "/images/favicon.jpg",      "/icons/icon-128x128.png",      "/icons/icon-192x192.png",      "/icons/icon-384x384.png",      "/icons/icon-512x512.png"    ];  const CATALOGUE\_ASSETS = "catalogue-assets";  self.addEventListener("install", (installEvt) => {    installEvt.waitUntil(      caches        .open(CATALOGUE\_ASSETS)        .then((cache) => {          console.log(cache)          cache.addAll(assets);        })        .then(self.skipWaiting())        .catch((e) => {          console.log(e);        })    );  });  self.addEventListener("activate", function (evt) {    evt.waitUntil(      caches        .keys()        .then((keyList) => {          return Promise.all(            keyList.map((key) => {              if (key === CATALOGUE\_ASSETS) {                console.log("Removed old cache from", key);                return caches.delete(key);              }            })          );        })        .then(() => self.clients.claim())    );  });  self.addEventListener("fetch", function (evt) {    evt.respondWith(      fetch(evt.request).catch(() => {        return caches.open(CATALOGUE\_ASSETS).then((cache) => {          return cache.match(evt.request);        });      })    );  }) |

Test your PWA

|  |  |
| --- | --- |
| 1. Open Google Chrome 2. Open Developer tools 3. Choose Lighthouse report 4. Run the report   Note: You are note expected to address all diagnostic issues in your PWA. |  |

|  |
| --- |
| 1. Open Google Chrome 2. Open Developer tools 3. Choose Lighthouse report 4. Run the report |