Agenda

1. Progressive Web App
2. What is Progressive Web App?
3. Why Progressive Web App?
4. Core Tenets – Progressive Web App
5. First Progressive Web App

Introduction

“A Progressive Web App uses modern web capabilities to deliver an app-like user experience” - Progressiv Apps

Progressive Web Apps are a way to experience the combined best of the ***web*** and ***apps*** services

Useful to users from very the first visit, ***no installation*** is required

Features of a Progressive Web App:

- Loads quickly
- Sends relevant push notifications
- Icon is present on the home screen
- Full screen experience

What is Progressive Web App?

A Progressive Web App is:

I. **Progressive**: The browser choice doesn't matter, works for every user

II. **Responsive:** Fits into any form, desktop, mobile, tablet, etc.

III. **Connectivity Independent:** Can work offline or on low quality networks

IV. **App-like:** Feels very easy to use as an App

V. **Re-engageable:** Features like push notifications makes re-engagement easy

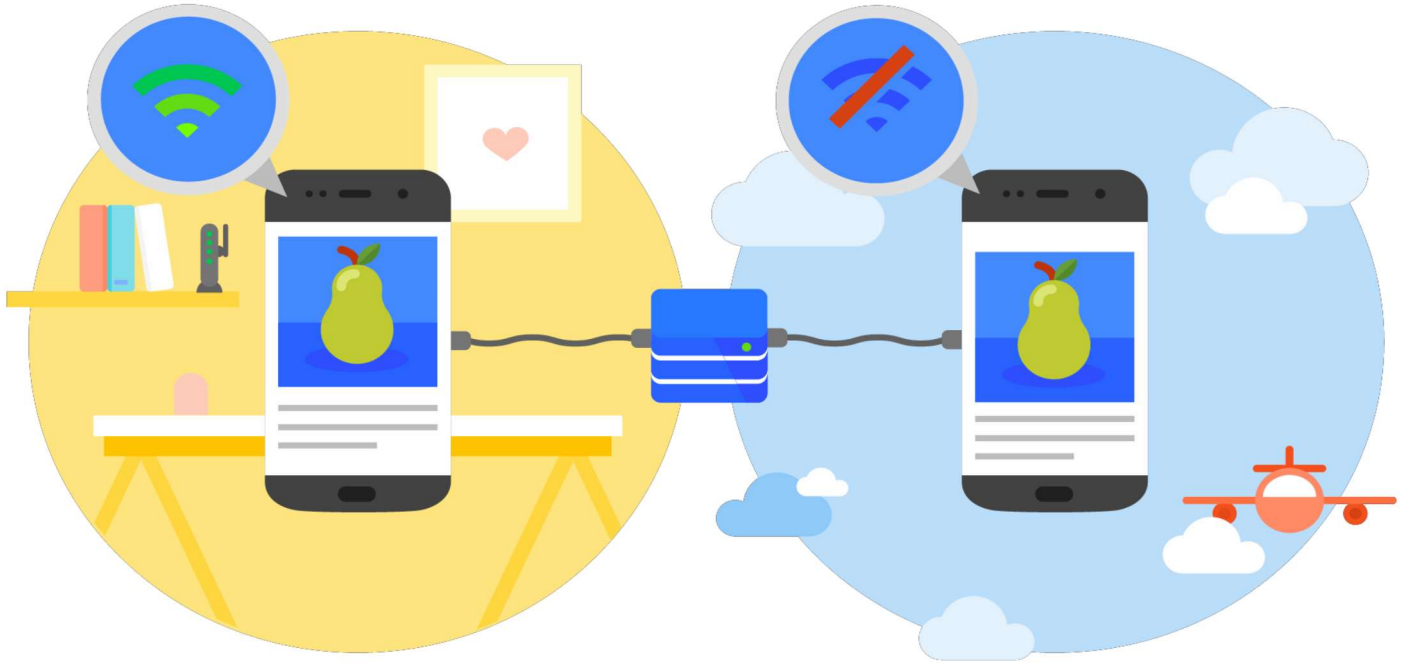
VI. **Installable:** No hassle of App Stores, allows users to keep the most useful apps on home screen

Why Progressive Web Apps?

- Reliable
- Fast
- Engaging

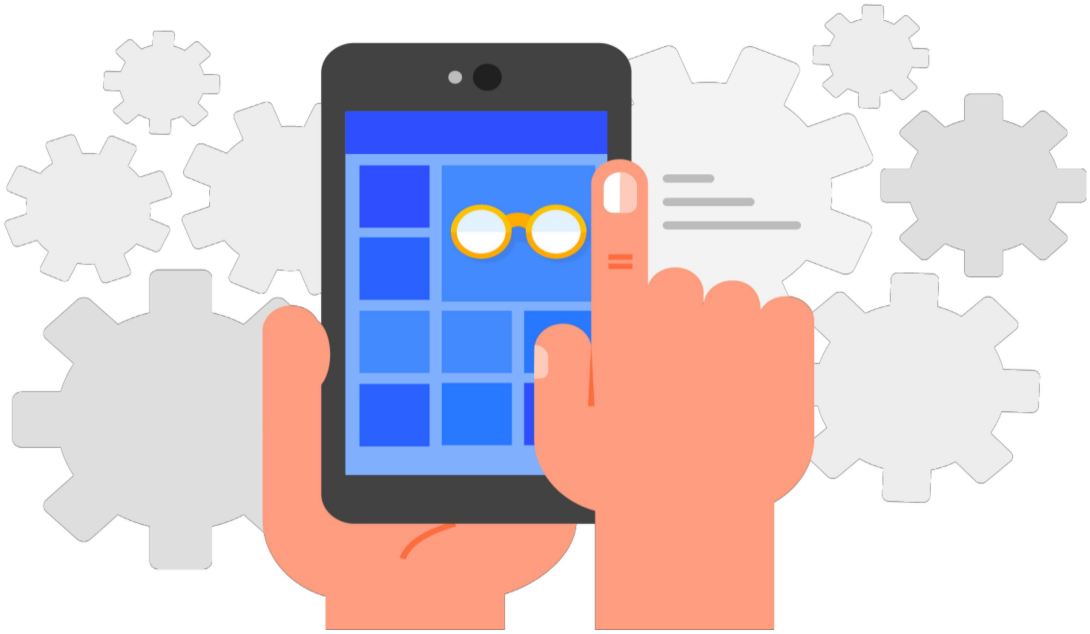
Why Progressive Web Apps? (cont.)

- **Reliable**: Loads instantly, even in ***uncertain*** or ***slow network*** conditions
- Can be launched from a user's home screen, service workers enable a Progressive Web App to load ins
  regardless of the network state



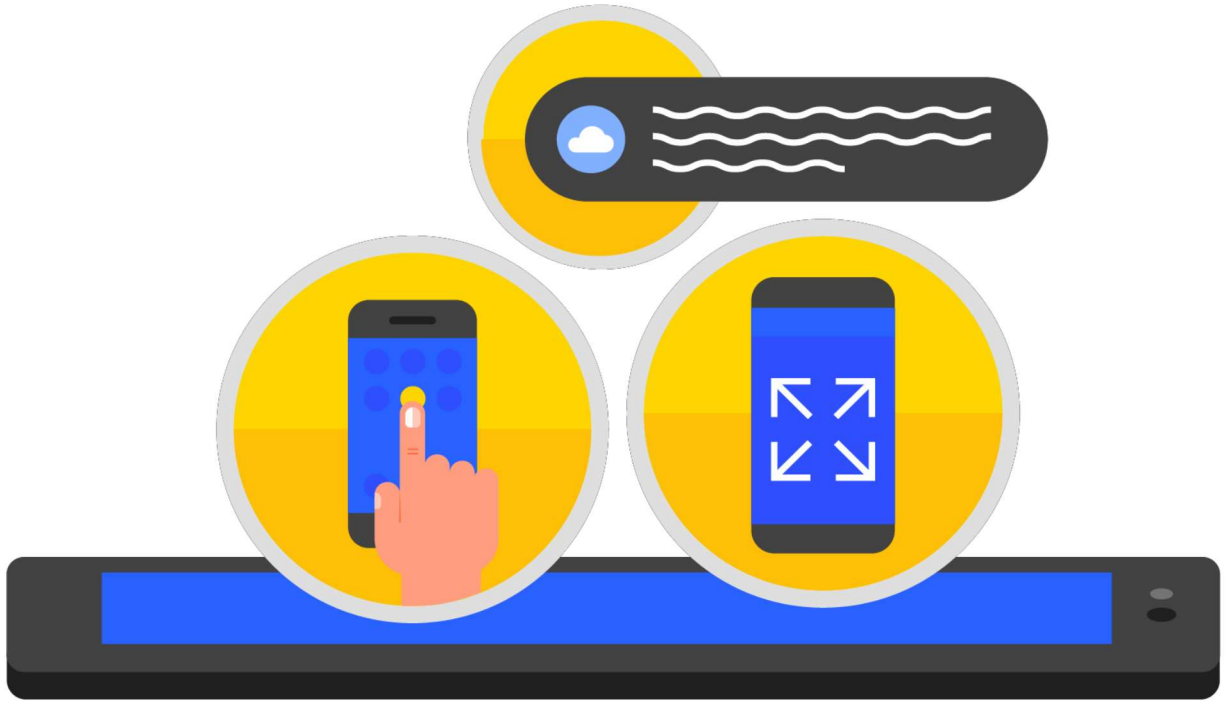
Why Progressive Web Apps? (cont.)

- **Fast:** Most of the users will abandon a site if it takes ***longer to load!*** And once loaded, they expect be fast; it should not be ***slow to respond***



Why Progressive Web Apps? (cont.)

- **Engaging:** Progressive Web Apps are installable and live on users' ***home screens***, offer an imme screen experience with use of ***web app manifest*** and with ***web push notifications***



Core Tenets of Progressive Web App

- Some core tenets of a progressive web app are:
  - *Service Workers*
  - *App Shell*
  - *Instalability and App Manifest*

Service Workers

- Incredibly powerful technology behind a Progressive Web App

- Powers ***offline functionality***, ***background content updating***, ***content caching***, and
- a lot more

- Service worker is a worker script that works behind the scenes, independent of your app and responses to events like network requests, push notifications, connectivity changes, etc.

- Service workers can be described as **"proxy,"** events, like **fetch** happens anytime a network reques We can handle that event with ***full control***, ***checking for cached data and returning immedi allowing the request to continue to a remote server***

- Our script acts as a proxy, or middleware for the request

Service Workers (cont.)

- Service workers due to ***flexibility*** are very complicated
- Generally developers use pre-made recipes for common service workers such as the ***offline mode***

***Note**: See the code section for offline mode service workers code example

Conclusion:

Service workers are just a ***JavaScript*** file

Running in background and triggered via events

Up to the developer for writing code to handle caching, push notifications, content fetching, etc.

Likely to be used existing recipes

App Shell

- It is a design approach, chosen by developers to adhere to what is enhanced by caching abilities of ***service workers***
- It's a pretty ***straightforward***, ***obvious approach***, made more dramatic by a ***buzzword***
- With the App Shell model, we focus on keeping the shell of our app ***UI and the content inside of it separate***
- App Shell is cached such that it loads as quickly as possible when a user visits and returns at a later date
- As ***shell*** and the ***content*** load separately, it improves the ***performance*** and ***usability*** of the app

Instalability and App Manifest

- ***Problem Statement: Mobile web apps*** were not installed like an
- ***app*** to the ***home screen***
-
  - ***Solution***
    - ***Chrome on Android*** added support for installing web apps to the home screen with a nati banner, just like native app banners
    - In order to tell Chrome our mobile website is installable as an app, we need to ***manifest.json*** file and link to it from the ***main HTML page***

Let’s Code

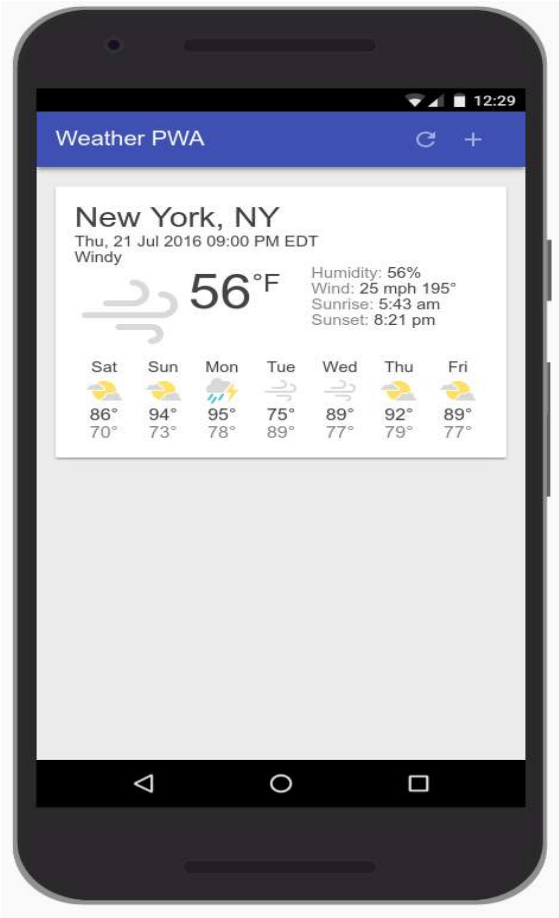
- Create your first Progressive Web App:

A Weather Progressive Web App

The Weather Progressive Web App - Introduction

- To simulate ***temperature as per location***
- **Objectives:**
  - ***Injecting the weather forecast data***: The app shows weather reports
- based on the **IP address geo-location** of the user as soon as he/she logs in
  - ***Differentiating the first run:*** On subsequent visits, the app shows the information as per the us current locations. This is not necessarily for the first location they ever looked up
- **Languages used:**
  - **HTML 5**
  - **Java Script**

The Weather Progressive Web App



Install and Verify the Web Server

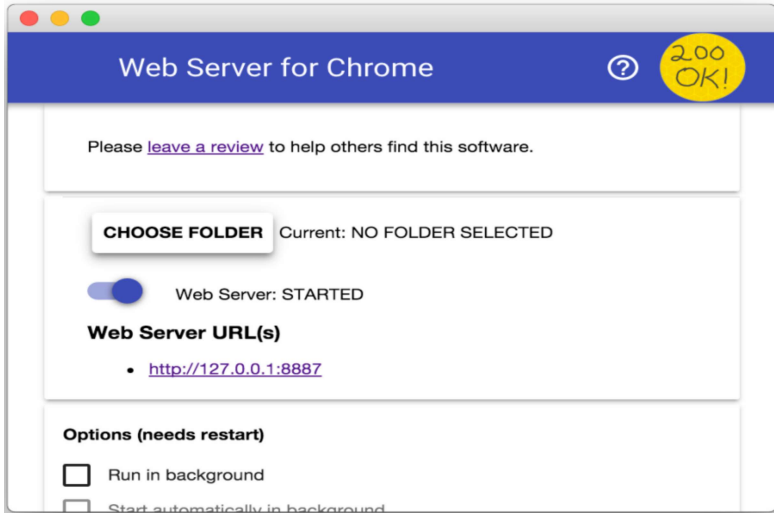
- Required Web Server: ***Chrome Web Server***
- ***Download Link:*** *https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb?hl=en*

After installing the ***Web Server for Chrome App***, click on the Apps shortcut on the bookmarks ba

In the window that pops up next, click on the Web Server icon:

Install and Verify the Web Server (cont.)

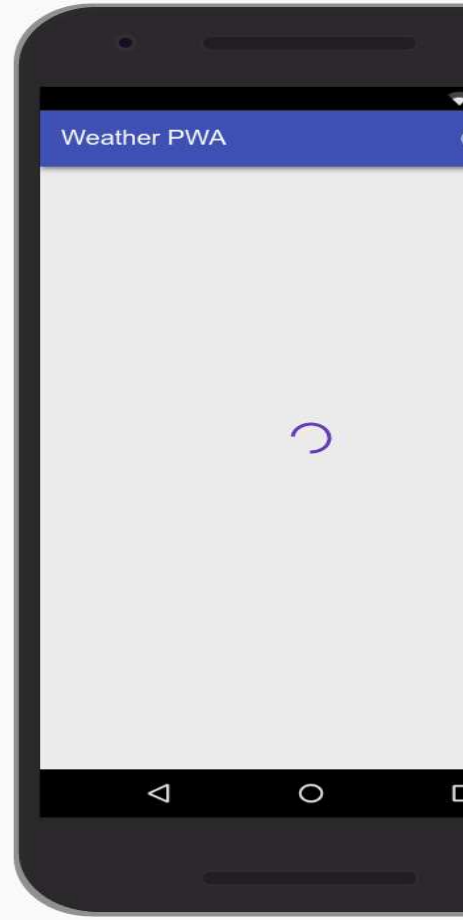
- A dialog box will appear next that allows us to configure the local web server:



- Click the ***choose folder*** button, and select the ***work*** folder
- This enables you to serve your work in progress via the URL highlighted in the web server dialog
- Under Options, check the box present next to ***"Automatically show index.html"***
- Then ***stop and restart the server*** by sliding the toggle labeled ***"Web Server:STARTED" to the left then back to the right.***

Install and Verify the Web Server (cont.)

- Now visit the work site in a web browser (***Click on the highlighted Web Server URL***) and the en
  new page that will look like this:



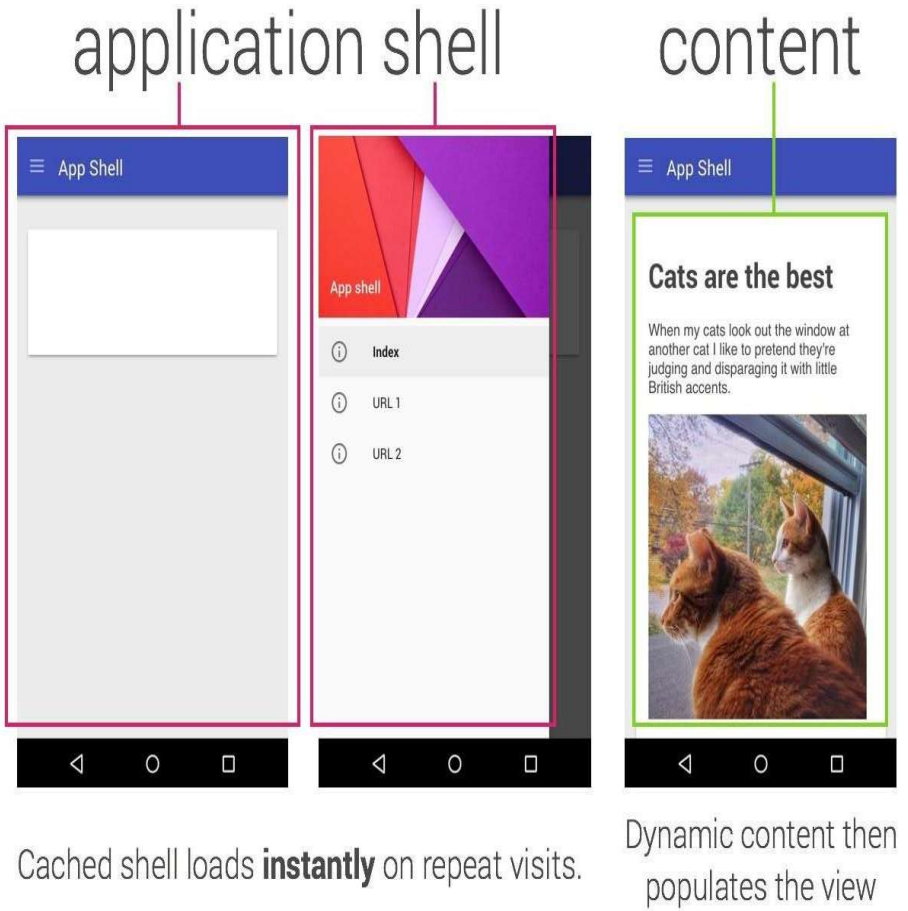
- The app is still not functional, it's just a minimal

skeleton with a ***Spinner*** to verify web server functionalities and UI features

Architect of the App Shell

- App Shell: ***HTML, CSS, and JavaScript*** are required to power the user interface of a ***Progressiv App***
- It’s first load should be extremely ***quick and immediately cached***
- ***"Cached"*** means that the shell files are loaded once they are over the network and then saved t local device
- Every subsequent time that the user opens the app, the shell files are loaded from the local device's that results in ***blazing- fast startup times***

Architect of the App Shell (cont.)



Implement Your App Shell

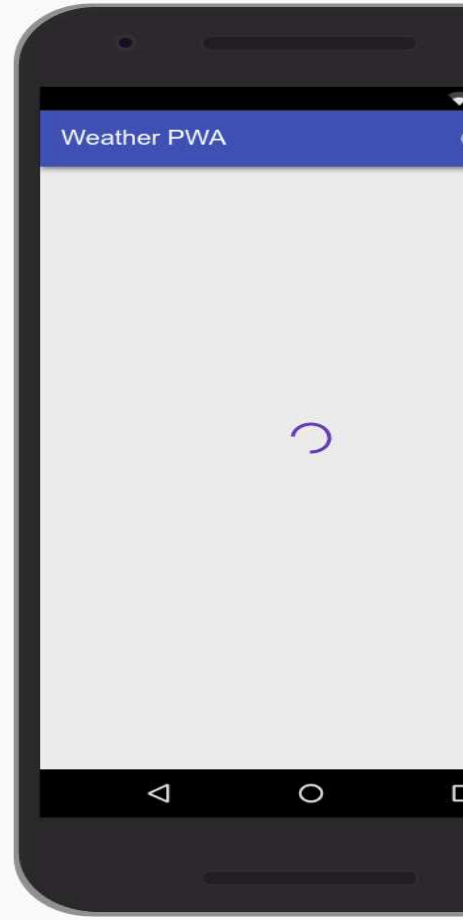
- **HTML for an App Shell:** The key components of an App Shell for our application consists of

  - ***Header with a title and add/refresh buttons***
  - ***Container for forecast cards***
  - ***Forecast card template***
  - ***Dialog for adding new cities***
  - ***Loading indicator***

- ***Note:***
  - ***Please refer to index.html file under work directory for full code of html page***
  - ***For Stylesheet file also check the work directory.***

Install and Verify the Web Server (cont.)

- Now visit the work site in a web browser (***Click on the highlighted Web Server URL***) and the en
  new page that will look like this:



- The app is still not functional, it's just a minimal skeleton with a ***Spinner*** to verify web server functionalities and UI features

JavaScript App Code

- As of now we have most of the UI ready, it's time to start hooking up the code to make eve work

- In our ***script/app.js*** it should include:
  - An ***app object*** that contains some of the key information necessary for the app

  - The ***event listeners*** for all of the buttons in the header (***add/refresh***) and in the ***add dialog (add/cancel)***

  - A method to add or update forecast cards (***app.updateForecastCard***)

  - A method to get the latest weather forecast data from the Firebase Public Weather AP (***app.getForecast***)

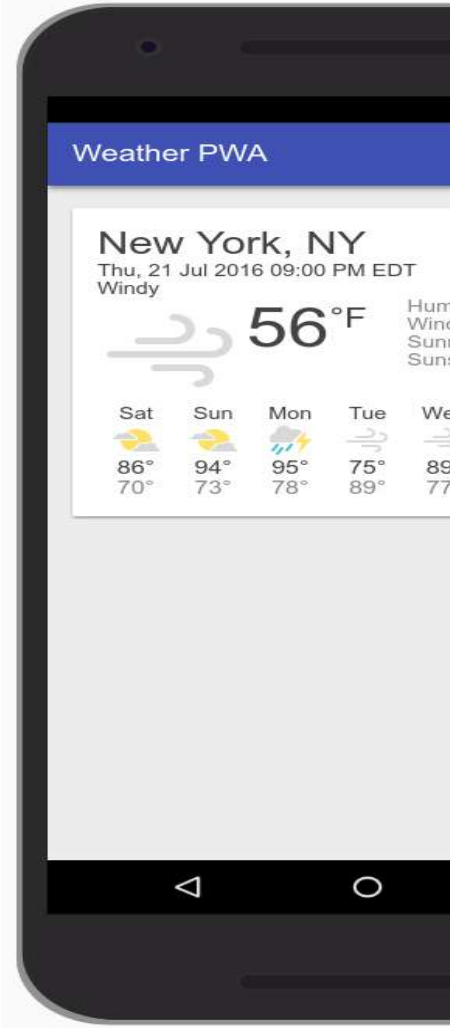
  - A method to iterate the current cards and call app.getForecast to get the latest foreca

- (***app.updateForecasts)***

  - Some fake data (***initialWeatherForecast)*** you can use to quickly test how things rend

JavaScript App Code

- ***Test Out:*** Now that we have done with ***Core HTML, STYLES, and JAVASCRIPT,*** we can test the a
  see how the fake weather data is rendered



Note:** ***Uncomment the following line at the bottom of index.html file:

<!--<script src="scripts/app.js" async></script>-->

- ***Next, uncomment the following line at the bottom of your app.js file:***

// app.updateForecastCard(initialWeatherForecast);

- ***Reload the app: The result should be a nicely formatted forecast card with fake data***