

BAB VI

PWA #2

6.1 Tujuan

1. Praktikan mengetahui apa itu *Progressive Web Application*.
2. Praktikan memahami kegunaan dari *Progressive Web Application*.
3. Praktikan mengetahui cara mengimplementasikan *Progressive Web Application*.
4. Praktikan dapat memahami cara kerja dari *Progressive Web Application*.
5. Praktikan dapat mengimplementasikan berbagai *API (Application Programming Interface)* pada *Progressive Web Application*.
6. Praktikan dapat mengetahui berbagai macam jenis *API (Application Programming Interface)* untuk berbagai *platform*.

6.2. Alat dan Bahan

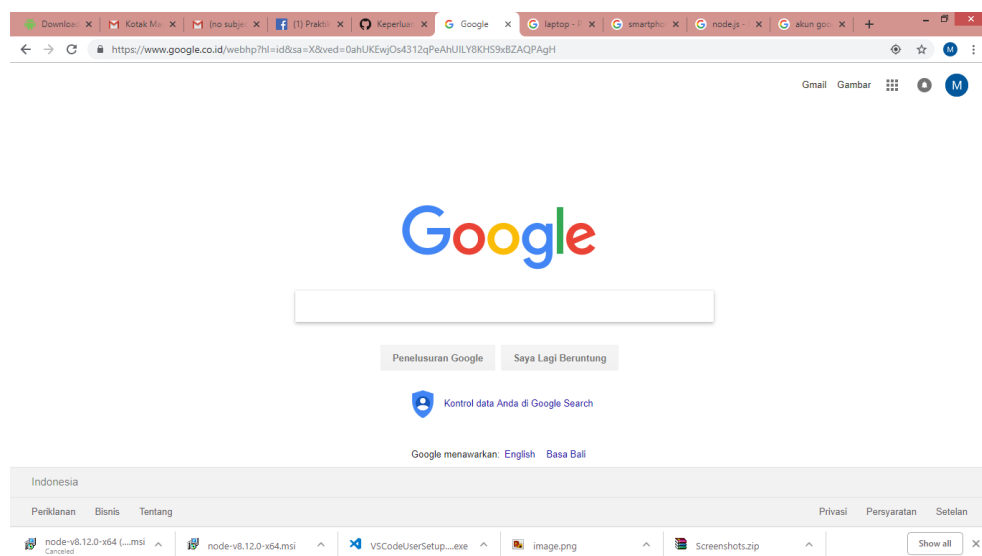
1. PC / Notebook



Gambar 6.1 Laptop

Laptop digunakan sebagai media dalam pengerjaan praktikum kali ini. Kita *install* dan melakukan kegiatan *programming* pada *laptop*.

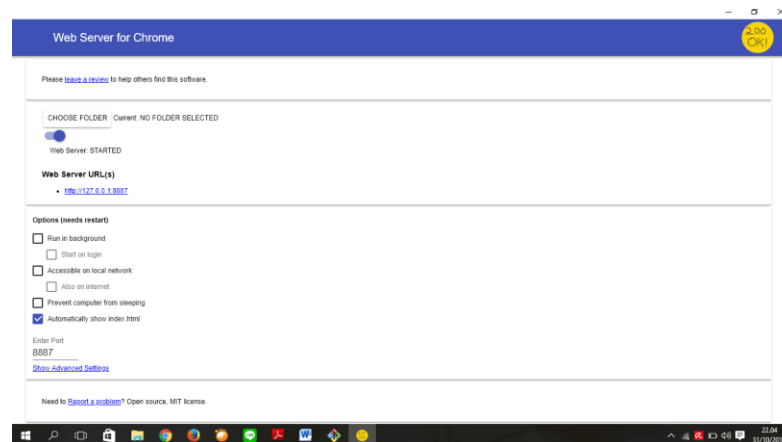
2. Google Chrome (ver 52 atau di atasnya)



Gambar 6.2 Google Chrome Pada Laptop

Google Chrome digunakan untuk kita membuka alamat *hosting* yang telah dibuat sebelumnya. Kita dapat melihat tampilan pada alamat *hosting* yang telah dibuat.

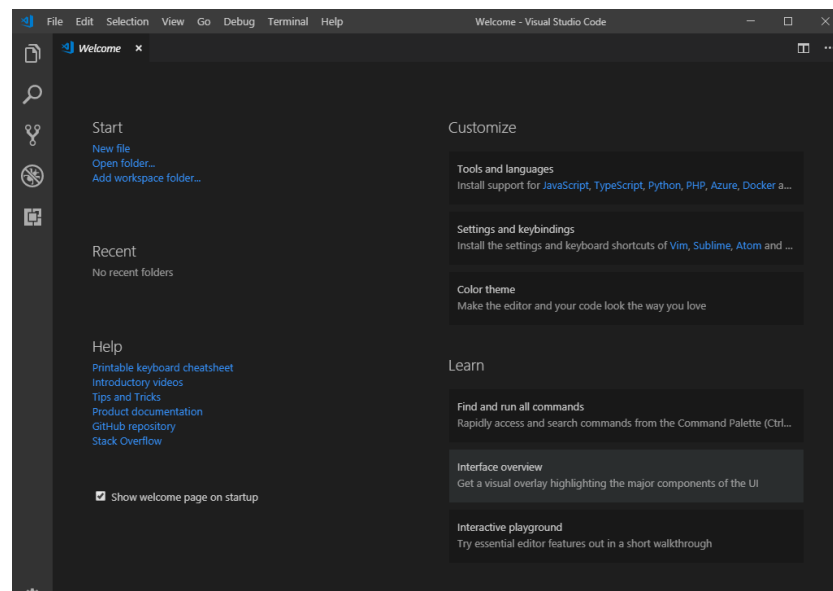
3. Web Server for Chrome (ekstensi google chrome)



Gambar 6.3 Web Server for Chrome

Pada praktikum kali ini digunakan *Web Server for Chrome* sebagai *software* yang memberikan layanan berbasis data dan berfungsi menerima permintaan dari HTTP atau HTTPS pada klien yang dikenal dan biasanya kita kenal dengan nama *web browser* (Mozilla Firefox, Google Chrome) dan untuk mengirimkan kembali yang hasilnya dalam bentuk beberapa halaman *web* dan pada umumnya akan berbentuk dokumen HTML.

4. Notepad++ / Sublime text 3/Visual Studio Code (Text editor)



Gambar 6.4 Visual Studio Code

Text Editor yang akan digunakan pada praktikum kali ini yaitu Visual Studio Code. Di sinilah kita akan melakukan kegiatan *coding program* untuk praktikum.

6.3 Dasar Teori

6.3.1 Progressive Web Application

Progressive Web App adalah pengalaman yang menggabungkan yang terbaik dari *web* dan yang terbaik dari aplikasi. Pengalaman ini bermanfaat untuk pengguna dari kunjungan pertamanya di *tab browser*, tanpa harus melakukan pemasangan. Pengguna secara progresif akan membangun hubungan dengan aplikasi, yang semakin lama semakin kuat. Aplikasi dimuat cepat, bahkan pada jaringan yang tidak stabil, mengirimkan pemberitahuan *push* yang relevan, memiliki ikon pada layar beranda, dan dimuat dengan pengalaman tingkat atas selayar penuh.

PWA memiliki sifat-sifat sebagai berikut :

- **Progresif** - Bekerja untuk setiap pengguna, apa pun pilihan *browser* mereka karena dibangun dengan peningkatan progresif sebagai konsep intinya.
- **Responsif** - Cocok dengan setiap faktor bentuk: perangkat *desktop*, seluler, tablet, atau apa saja yang muncul berikutnya.
- **Konektivitas independen** - Disempurnakan dengan *service worker* agar bisa bekerja offline atau pada jaringan berkualitas-rendah.
- **Seperti-Aplikasi** - Terasa seperti sebuah aplikasi untuk pengguna dengan interaksi dan navigasi bergaya-aplikasi karena mereka dibangun di atas model *shell* aplikasi.
- **Segar** - Selalu terkini berkat proses pembaruan *service worker*.
- **Aman** - Disediakan melalui HTTPS untuk mencegah *snooping* dan memastikan materi belum dirusak.
- **Dapat ditemukan** - Dapat diidentifikasi sebagai "aplikasi" berkat manifes W3C dan cakupan registrasi *service worker*, yang memungkinkan mesin telusur untuk menemukannya.
- **Bisa dilibatkan-kembali** - Kemudahan untuk dilibatkan-kembali dengan fitur seperti pemberitahuan *push*.
- **Dapat dipasang** - Memungkinkan pengguna untuk "menyimpan" aplikasi yang mereka anggap paling berguna di layar beranda tanpa kerumitan toko aplikasi.

- **Bisa ditautkan** - Dapat dengan mudah dibagikan melalui URL, tidak memerlukan pemasangan yang rumit.

Sumber: <https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/?hl=id>

6.3.2 Service Worker

Service worker adalah skrip yang dijalankan *browser* Anda di latar belakang, terpisah dari laman *web*, yang membuka pintu ke berbagai fitur yang tidak memerlukan laman *web* atau interaksi pengguna. Saat ini, *service worker* sudah menyertakan berbagai fitur seperti pemberitahuan *push* dan sinkronisasi latar belakang. Di masa mendatang, *service worker* akan mendukung hal-hal lainnya seperti sinkronisasi berkala atau *geofencing*. Fitur inti yang didiskusikan dalam tutorial adalah kemampuan mencegat dan menangani permintaan jaringan, termasuk mengelola *cache* respons lewat program.

Yang membuat API ini menarik adalah karena memungkinkan Anda mendukung pengalaman *offline*, yang memberikan *developer* kontrol penuh atas pengalaman.

Sebelum *service worker*, ada satu API lain yang memberi pengguna pengalaman *offline* di web, yang disebut AppCache. Masalah utama pada AppCache adalah jumlah *gotcha* yang ada serta fakta bahwa meskipun desain bekerja dengan sangat baik untuk laman aplikasi *web* tunggal, namun ternyata tidak begitu baik untuk situs multi-laman. *Service worker* telah didesain untuk menghindari titik-titik menyulitkan yang sudah umum ini.

Sumber: <https://developers.google.com/web/fundamentals/primers/service-workers/?hl=id>

6.3.3 Web Server

Server atau *Web server* adalah sebuah *software* yang memberikan layanan berbasis data dan berfungsi menerima permintaan dari HTTP atau HTTPS pada klien yang dikenal dan biasanya kita kenal dengan nama *web browser* (Mozilla Firefox, Google Chrome) dan untuk mengirimkan kembali yang hasilnya dalam bentuk beberapa halaman *web* dan pada umumnya akan berbentuk dokumen HTML.

Fungsi utama *Server* atau *Web server* adalah untuk melakukan atau akan mentransfer berkas permintaan pengguna melalui protokol komunikasi yang telah

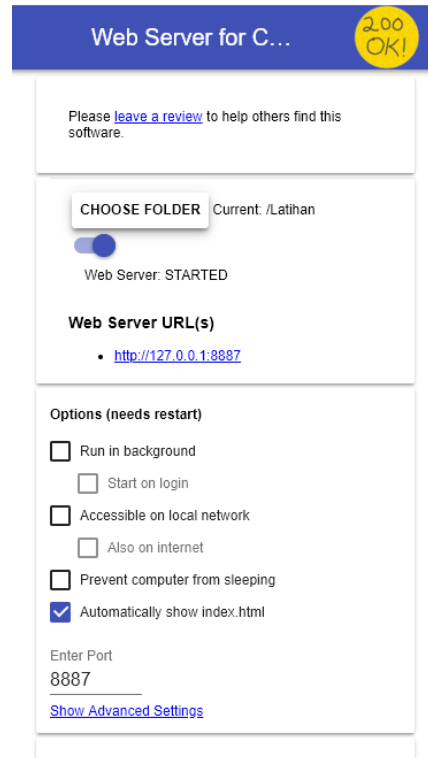
ditentukan sedemikian rupa. Halaman *web* yang diminta terdiri dari berkas teks, video, gambar, *file* dan banyak lagi. Pemanfaatan *web server* berfungsi untuk mentransfer seluruh aspek pemberkasan dalam sebuah halaman *web* termasuk yang di dalam berupa teks, video, gambar dan banyak lagi.

Salah satu contoh dari *Web Server* adalah Apache. Apache (*Apache Web Server – The HTTP Web Server*) merupakan *web server* yang paling banyak dipergunakan di Internet. Program ini pertama kali didesain untuk sistem operasi lingkungan UNIX. Apache mempunyai program pendukung yang cukup banyak. Hal ini memberikan layanan yang cukup lengkap bagi penggunaanya.

Sumber: <https://idcloudhost.com/pengertian-web-server-dan-fungsinya/>

6.4 Langkah Percobaan

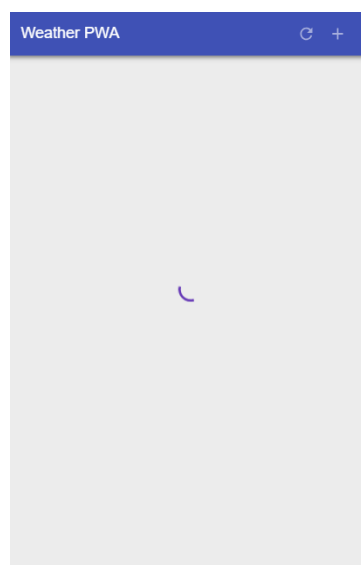
1. Menyalakan dan mengkonfigurasi Web Server for Chrome.



Gambar 6.5 Memilih *folder* Latihan pada *Web Server for Chrome*

Choose folder lalu pilih *folder* Latihan yang telah *download* sebelumnya, kemudian centang *Automatically show index.html*. Lalu nyalakan *web server*.

2. Buka link dari *web server* dan lihat tampilannya.



Gambar 6.6 Tampilan *link* dari *web server*

3. Untuk melihat bagaimana data cuaca palsu dirender, hilangkan tanda komentar pada baris berikut di bagian bawah file `index.html`

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

Berikutnya, hilangkan tanda komentar pada baris berikut di bagian bawah file `app.js`

```
// app.updateForecastCard(initialWeatherForecast);
```

Muat ulang aplikasi. Hasilnya akan menjadi kartu prakiraan cuaca yang terformat dengan baik (menggunakan data palsu yang telah dibuat) dengan spinner dinonaktifkan, seperti ini:



Gambar 6.7 Hasil tampilan *link* dari *web server* setelah diedit dan dimuat ulang

4. Tambahkan kode yang diperlukan untuk menyimpan preferensi pengguna. Temukan komentar `TODO` berikut dalam `app.js`

```
// TODO add saveSelectedCities function here
```

Dan tambahkan kode berikut di bawah komentar.

```
// Save list of cities to localStorage.
app.saveSelectedCities = function() {
  var selectedCities = JSON.stringify(app.selectedCities);
  localStorage.selectedCities = selectedCities;
};
```

```
// TODO add saveSelectedCities function here
// Save list of cities to localStorage.
app.saveSelectedCities = function() {
  var selectedCities = JSON.stringify(app.selectedCities);
  localStorage.selectedCities = selectedCities;
};
```

Gambar 6.8 Penambahan kode di bawah `// TODO add saveSelectedCities function here`

5. Berikutnya, tambahkan kode *startup* untuk memeriksa apakah pengguna memiliki kota yang disimpan dan merender kota tersebut, atau menggunakan data yang dimasukkan. Temukan komentar berikut:

```
// TODO add startup code here
```

Dan tambahkan kode berikut di bawah komentar.

```
app.selectedCities = localStorage.selectedCities;
if (app.selectedCities) {
  app.selectedCities = JSON.parse(app.selectedCities);
  app.selectedCities.forEach(function(city) {
    app.getForecast(city.key, city.label);
  });
} else {
  /* The user is using the app for the first time, or the
  user has not
  * saved any cities, so show the user some fake data. A
  real app in this
  * scenario could guess the user's location via IP
  lookup and then inject
  * that data into the page.
  */
  app.updateForecastCard(initialWeatherForecast);
  app.selectedCities = [
    {key: initialWeatherForecast.key, label:
initialWeatherForecast.label}
  ];
  app.saveSelectedCities();
}
```

```
// TODO add startup code here
app.selectedCities = localStorage.selectedCities;
if (app.selectedCities) {
  app.selectedCities = JSON.parse(app.selectedCities);
  app.selectedCities.forEach(function(city) {
    app.getForecast(city.key, city.label);
  });
} else {
  /* The user is using the app for the first time, or the user has not
  * saved any cities, so show the user some fake data. A real app in this
  * scenario could guess the user's location via IP lookup and then inject
  * that data into the page.
  */
  app.updateForecastCard(initialWeatherForecast);
  app.selectedCities = [
    {key: initialWeatherForecast.key, label: initialWeatherForecast.label}
  ];
  app.saveSelectedCities();
}
```

Gambar 6.9 Penambahan kode di bawah `// TODO add startup code here`

Kode *startup* memeriksa bila ada kota yang disimpan dalam penyimpanan lokal. Jika ada, itu akan mem-*parse* data penyimpanan lokal dan kemudian menampilkan kartu prakiraan untuk masing-masing kota yang disimpan. Jika tidak, kode *startup* akan menggunakan data prakiraan palsu dan menyimpannya sebagai kota *default*.

6. Mengubah penanganan tombol "*add city*" untuk menyimpan kota yang dipilih ke penyimpanan lokal. Cari `butAddCity` pada `app.js` lalu ubah menjadi seperti berikut.

```
document.getElementById('butAddCity').addEventListener('click', function() {
    // Add the newly selected city
    var select = document.getElementById('selectCityToAdd');
    var selected = select.options[selected.selectedIndex];
    var key = selected.value;
    var label = selected.textContent;
    if (!app.selectedCities) {
        app.selectedCities = [];
    }
    app.getForecast(key, label);
    app.selectedCities.push({key: key, label: label});
    app.saveSelectedCities();
    app.toggleAddDialog(false);
});
```

```
document.getElementById('butAddCity').addEventListener('click', function() {
    // Add the newly selected city
    var select = document.getElementById('selectCityToAdd');
    var selected = select.options[selected.selectedIndex];
    var key = selected.value;
    var label = selected.textContent;
    if (!app.selectedCities) {
        app.selectedCities = [];
    }
    app.getForecast(key, label);
    app.selectedCities.push({key: key, label: label});
    app.saveSelectedCities();
    app.toggleAddDialog(false);
});
```

Gambar 6.10 Mengubah kode untuk penanganan tombol *add city*

7. Selanjutnya mendaftarkan *service worker*. Temukan komentar berikut pada `app.js`

```
// TODO add service worker code here
```

Tambahkan kode berikut setelah komentar tersebut

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker
```

```

        .register('./service-worker.js')
        .then(function() { console.log('Service Worker
Registered'); });
    }

```

```

// TODO add service worker code here
if ('serviceWorker' in navigator) {
    navigator.serviceWorker
        .register('./service-worker.js')
        .then(function() { console.log('Service Worker Registered'); });
}
})();

```

Gambar 6.11 Penambahan kode di bawah // TODO add service worker code here

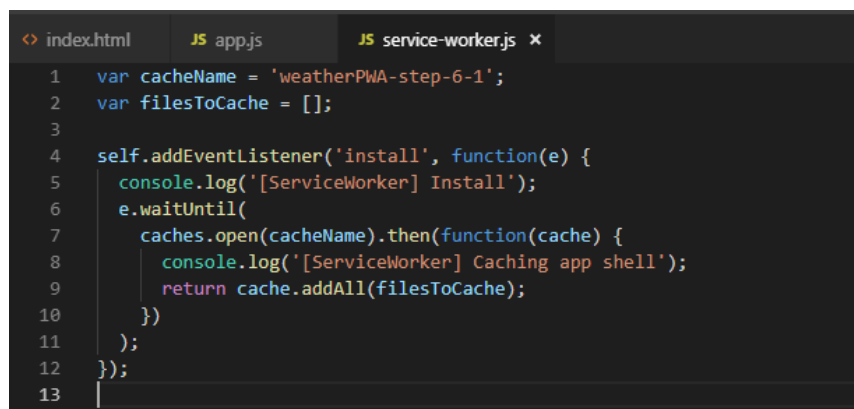
8. Buatlah *file* bernama `service-worker.js` pada *folder* latihan lalu tambahkan kode berikut.

```

var cacheName = 'weatherPWA-step-6-1';
var filesToCache = [];

self.addEventListener('install', function(e) {
    console.log('[ServiceWorker] Install');
    e.waitUntil(
        caches.open(cacheName).then(function(cache) {
            console.log('[ServiceWorker] Caching app shell');
            return cache.addAll(filesToCache);
        })
    );
});

```



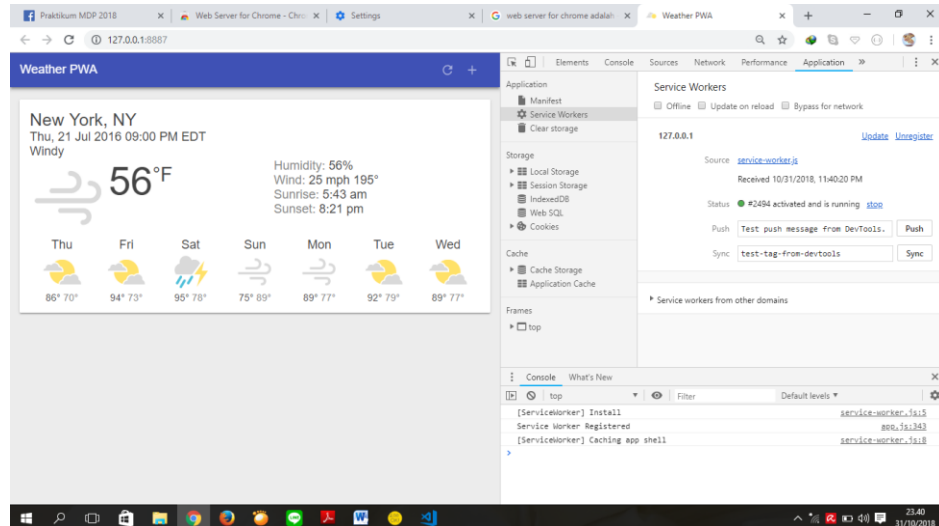
```

1  var cacheName = 'weatherPWA-step-6-1';
2  var filesToCache = [];
3
4  self.addEventListener('install', function(e) {
5      console.log('[ServiceWorker] Install');
6      e.waitUntil(
7          caches.open(cacheName).then(function(cache) {
8              console.log('[ServiceWorker] Caching app shell');
9              return cache.addAll(filesToCache);
10         })
11     );
12 });
13

```

Gambar 6.12 Membuat *file* `service-worker.js`

9. Buka Dev Tools pada Google chrome (Ctrl + Shift + i) saat membuka page latihan. Jika berhasil maka tampilan pada *Service Workers* akan seperti ini.



Gambar 6.13 Tampilan *Dev Tools* pada Google Chrome setelah ditambah kode pada program

10. Tambahkan *event listener* activate di atas *event listener* install dalam file `service-worker.js`

```
self.addEventListener('activate', function(e) {
  console.log('[ServiceWorker] Activate');
  e.waitUntil(
    caches.keys().then(function(keyList) {
      return Promise.all(keyList.map(function(key) {
        if (key !== cacheName) {
          console.log('[ServiceWorker] Removing old
cache', key);
          return caches.delete(key);
        }
      }));
    })
  );
  return self.clients.claim();
});
```

11. Pada bagian atas file `service-worker.js`, ganti `var filesToCache = [];` dengan kode di bawah ini:

```
var filesToCache = [
  '/',
  '/index.html',
  '/scripts/app.js',
  '/styles/inline.css',
  '/images/clear.png',
  '/images/cloudy-scattered-showers.png',
  '/images/cloudy.png',
  '/images/fog.png',
```

```

'/images/ic_add_white_24px.svg',
'/images/ic_refresh_white_24px.svg',
'/images/partly-cloudy.png',
'/images/rain.png',
'/images/scattered-showers.png',
'/images/sleet.png',
'/images/snow.png',
'/images/thunderstorm.png',
'/images/wind.png'
];

```

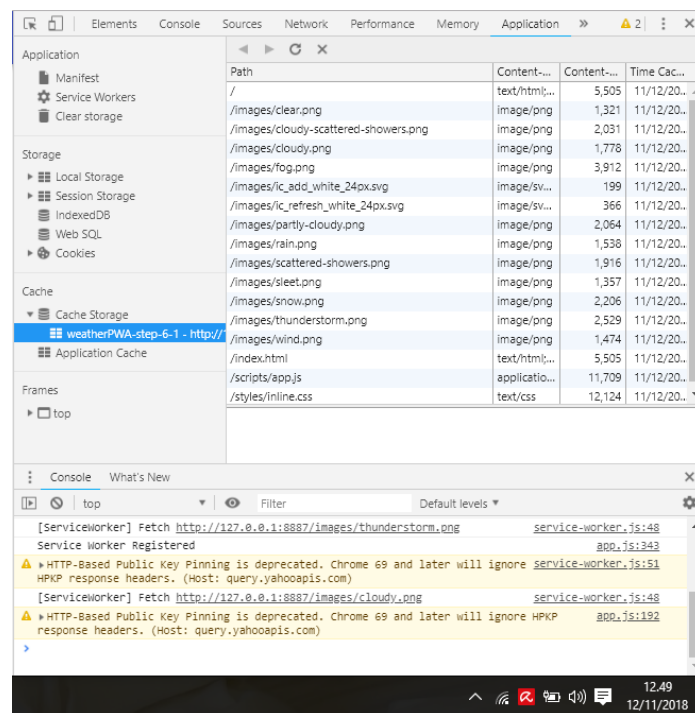
12. Sediakan *shell* aplikasi dari *cache*. Tambahkan kode berikut ke bagian bawah file `service-worker.js`

```

self.addEventListener('fetch', function(e) {
  console.log('[ServiceWorker] Fetch', e.request.url);
  e.respondWith(
    caches.match(e.request).then(function(response) {
      return response || fetch(e.request);
    })
  );
});

```

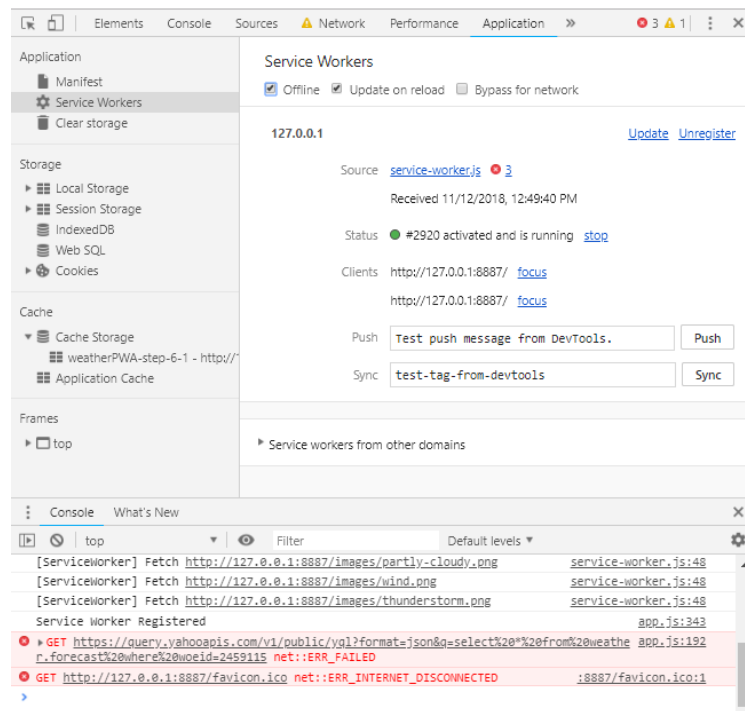
13. Muat ulang laman Anda lalu buka panel *Cache Storage* pada panel *Application* dari DevTools, lalu lihat apakah tampilan sudah seperti ini.



Gambar 6.14 Hasil pada *Cache Storage* setelah dimuat ulang

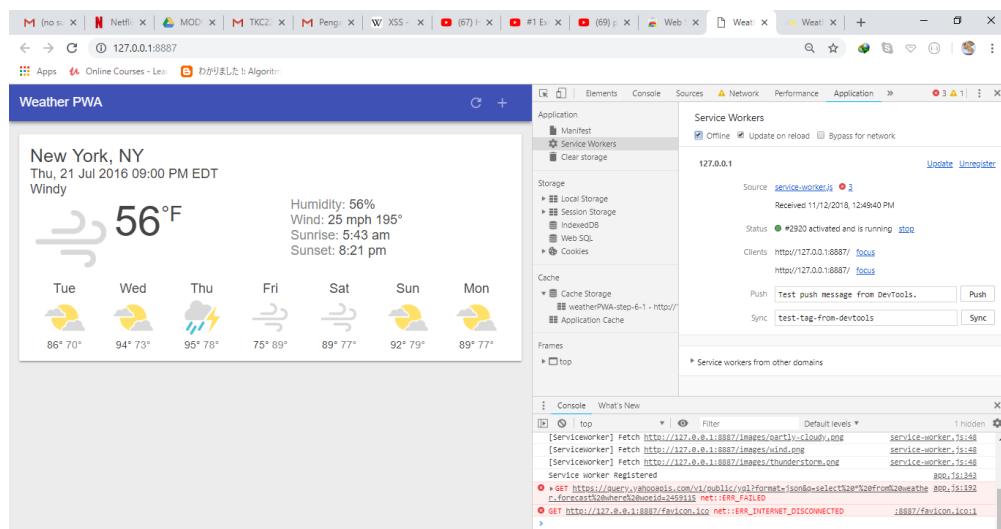
Jika sudah seperti itu maka *resource* yang digunakan telah berhasil di *cache*.

14. Sekarang menguji mode *offline*. Kembali ke panel *Service Worker* dari DevTools dan aktifkan kotak centang *Offline*. Setelah mengaktifkannya, Anda akan melihat ikon peringatan kecil berwarna kuning di sebelah *tab panel Network*. Ini menunjukkan bahwa kondisi telah *offline*.



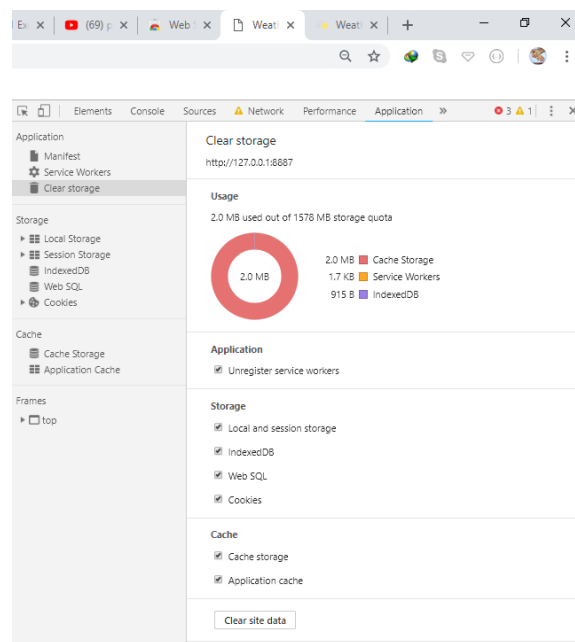
Gambar 6.15 Mengaktifkan mode *offline* pada *Service Worker*

15. *Reload* halaman dan lihat apakah yang akan ditampilkan dalam keadaan *offline*.



Gambar 6.16 Hasil yang ditampilkan pada mode *offline*

16. Selanjutnya adalah membuat *service worker* untuk meng-*cache* data prakiraan cuaca. Pertama hapus semua data yang tersimpan (*localStorage*, *data indexedDB*, *file cache*) dan membuang *service worker*, gunakan *panel Clear storage* di *tab Application* lalu klik *Clear site data*.



Gambar 6.17 Menghapus semua data yang tersimpan

17. Buka file `service-worker.js` lalu tambahkan kode berikut ke bagian atas dari file tersebut.

```
var dataCacheName = 'weatherData-v1';
```

18. Perbarui `if (key !== cacheName) {` dari event listener `activate` sehingga tidak menghapus *cache data* ketika membersihkan *cache shell* aplikasi.

```
if (key !== cacheName && key !== dataCacheName) {
```

19. Perbarui event listener `fetch` untuk menangani permintaan ke data API secara terpisah dari permintaan lainnya.

```
self.addEventListener('fetch', function(e) {
  console.log('[Service Worker] Fetch', e.request.url);
  var dataUrl = 'https://query.yahooapis.com/v1/public/yql';
  if (e.request.url.indexOf(dataUrl) > -1) {
    /*
     * When the request URL contains dataUrl, the app is
    asking for fresh
     * weather data. In this case, the service worker always
    goes to the
```

```

        * network and then caches the response. This is called
the "Cache then
        * network" strategy:
        */
        e.respondWith(
            caches.open(dataCacheName).then(function(cache) {
                return fetch(e.request).then(function(response) {
                    cache.put(e.request.url, response.clone());
                    return response;
                });
            })
        );
    } else {
        /*
        * The app is asking for app shell files. In this
scenario the app uses the
        * "Cache, falling back to the network" offline
strategy:
        */
        e.respondWith(
            caches.match(e.request).then(function(response) {
                return response || fetch(e.request);
            })
        );
    }
}
});

```

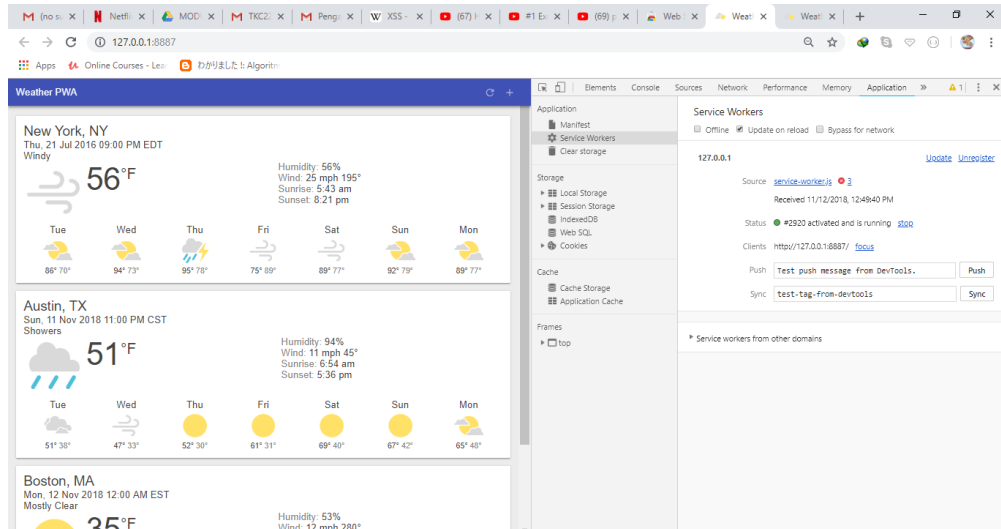
20. Berikutnya buka `app.js`, kita harus memeriksa apakah objek `caches` ada dan meminta data terbaru dari situ. Temukan komentar `TODO add cache logic here` di `app.getForecast()`, dan kemudian tambahkan kode berikut di bawah komentar tersebut.

```

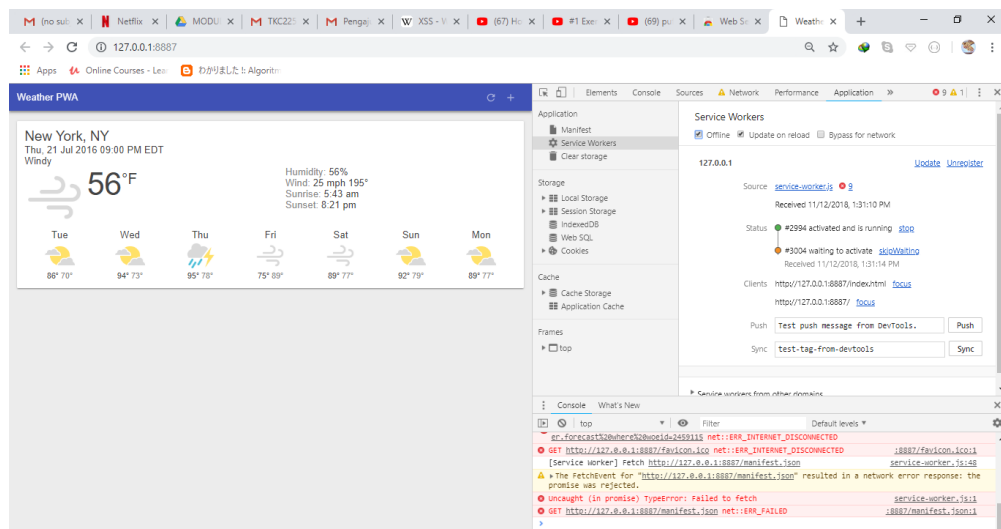
if ('caches' in window) {
    /*
        * Check if the service worker has already cached this
city's weather
        * data. If the service worker has the data, then
display the cached
        * data while the app fetches the latest data.
        */
        caches.match(url).then(function(response) {
            if (response) {
                response.json().then(function
updateFromCache(json) {
                    var results = json.query.results;
                    results.key = key;
                    results.label = label;
                    results.created = json.query.created;
                    app.updateForecastCard(results);
                });
            }
        });
}

```


21. Lakukan Pengujian dengan menambahkan kota baru pada prakiraan cuaca, lalu centang *Offline* pada *Service workers* di Dev Tools Google Chrome, lalu coba *reload* halaman tersebut.



Gambar 6.18 Menambahkan kota baru saat mode *Online*



Gambar 6.19 Tampilan saat mode *Offline*

22. Buat file bernama `manifest.json` di *folder* Latihan dan masukkan kode berikut.

```
{
  "name": "Weather",
  "short_name": "Weather",
  "icons": [{
    "src": "images/icons/icon-128x128.png",
    "sizes": "128x128",
    "type": "image/png"
  }], {
```

```

    "src": "images/icons/icon-144x144.png",
    "sizes": "144x144",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-152x152.png",
    "sizes": "152x152",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-192x192.png",
    "sizes": "192x192",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-256x256.png",
    "sizes": "256x256",
    "type": "image/png"
  }
],
"start_url": "/index.html",
"display": "standalone",
"background_color": "#3E4EB8",
"theme_color": "#2F3BA2"
}

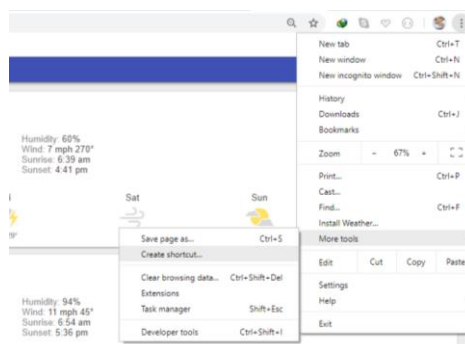
```

23. Sekarang tambahkan baris berikut ke bagian bawah elemen `<head>` dalam *file* `index.html`

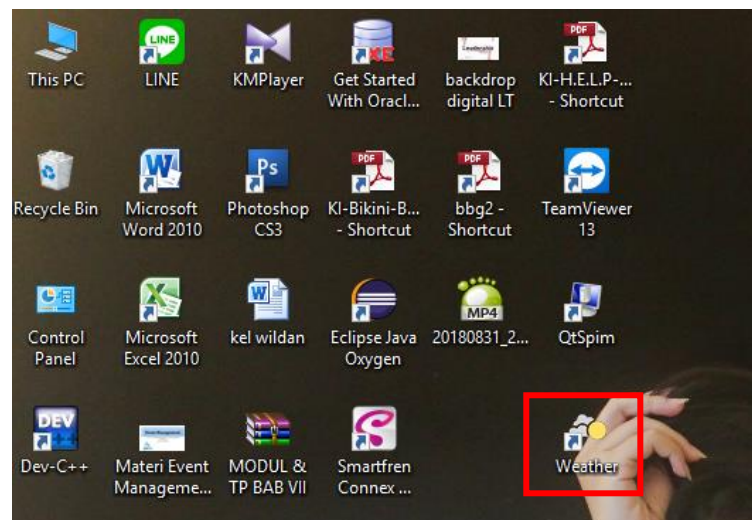
```
<link rel="manifest" href="/manifest.json">
```

24. *Run* aplikasi pada google chrome lalu buka Developer tools (Ctrl + Shift + i) lalu klik *tab application* dan klik *manifest*, pada bagian *identity* klik *Add to homescreen* lalu *reload page*, maka akan muncul *pop up* pada bagian atas google chrome, klik *Add* pada *pop up* tersebut dan lihat hasilnya.

Untuk beberapa kasus tidak dapat menambahkan dengan cara ini lakukan dengan cara memilih *Create Shortcut* pada menu *More tools* dalam *Customize and Control Google Chrome*.



Gambar 6.20 Membuat *shortcut* dari PWA cuaca



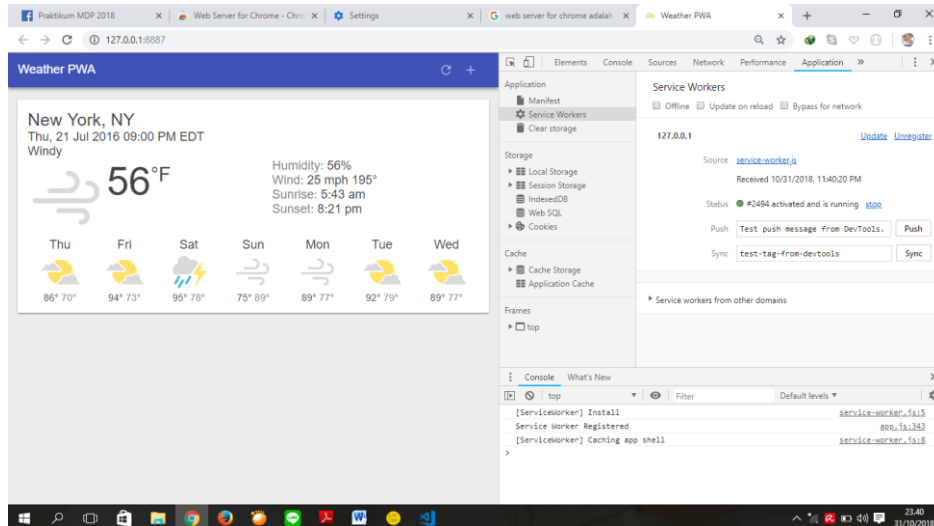
Gambar 6.21 *Shortcut pada homescreen*



Gambar 6.22 Tampilan aplikasi ketika dibuka

6.5 Analisa Percobaan

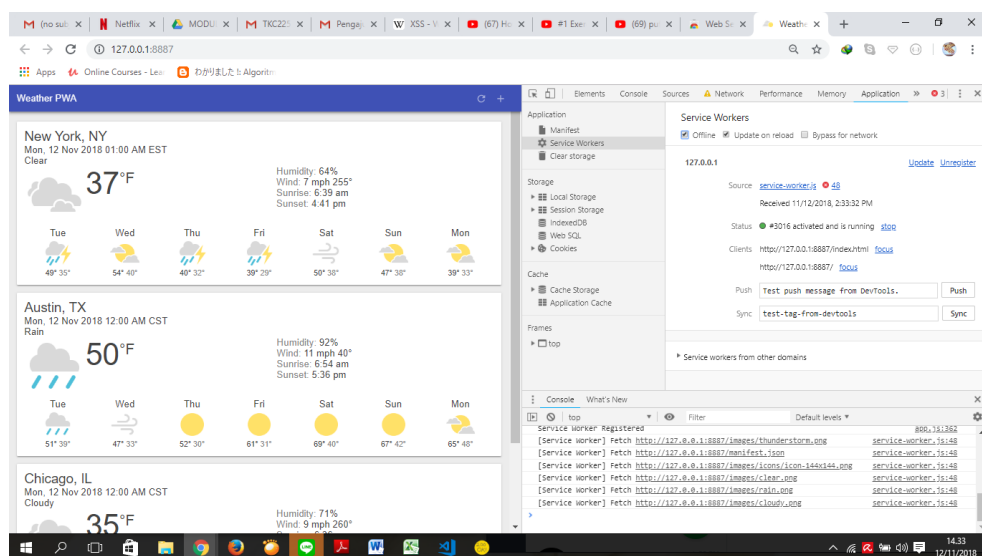
6.5.1 Percobaan menampilkan fake data saat offline



Gambar 6.23 Penampilan *fake data* pada saat *offline*

Pada percobaan ini, kita menggunakan *fake data*, atau data yang tidak *real* pada *data weather*, jadi data ini merupakan *data default* ketika *web server* tidak mempunyai *cache data* ketika mode *offline* diaktifkan dan *service-worker* akan mengarahkan pada *data default* yang dapat disetting melalui *scripts file app.js* yang dibuat dengan sebuah *variable* bernama *initialWeatherForecast*, disitu data berisikan pengaturan dalam membuat *data default fake* yang akan digunakan.

6.5.2 Percobaan menampilkan data terakhir saat offline



Gambar 6.24 Penampilan data terakhir pada saat *offline*

Pada percobaan kali ini, kita menggunakan data terakhir yang ditampilkan sebelum web server memasuki mode offline, jadi data yang terakhir ada pada weather yang merupakan data realtime, akan tetap bisa dilihat karena cache storage yang menyimpan data tersebut masih bisa digunakan ketika memasuki mode offline, hal ini juga merupakan fungsi dari cache storage yang mana dapat menyimpan serpihan data yang nantinya dapat digunakan lagi jikalau tidak terkoneksi pada internet.

6.5.3 Membuat manifest

```
{
  "name": "Weather",
  "short_name": "Weather",
  "icons": [{
    "src": "images/icons/icon-128x128.png",
    "sizes": "128x128",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-144x144.png",
    "sizes": "144x144",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-152x152.png",
    "sizes": "152x152",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-192x192.png",
    "sizes": "192x192",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-256x256.png",
    "sizes": "256x256",
    "type": "image/png"
  }],
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#3E4EB8",
  "theme_color": "#2F3BA2"
}
```

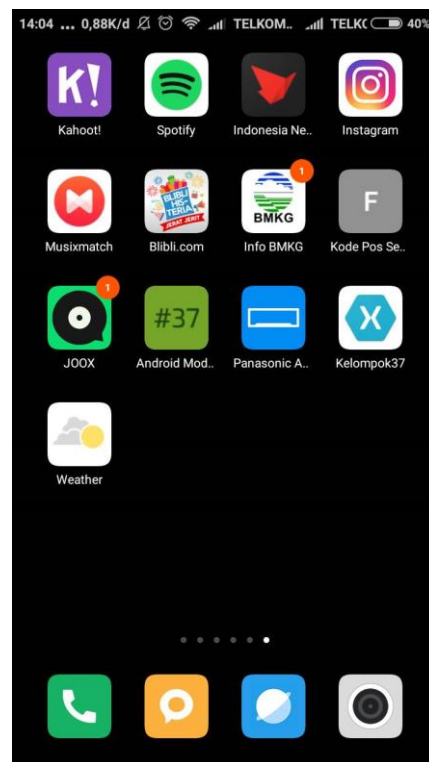
Manifest.json merupakan *file* berekstensi json, yaitu Bahasa pemrograman kekinian yang digunakan, dan butuh *library* penerjemah untuk menerjemahkan Bahasa json. *File* manifest.json pada proyek kali ini berfungsi sebagai *file* yang membuat suatu tampilan pada *mobile* saat *icon* pada *homescreen* di tekan. Karena pada dasarnya proyek ini merupakan proyek html yang diperuntukan untuk *web*,

namun agar tampilan pada *mobile* bisa lebih di tolerir, maka fungsi *manifest.json* ini lah yang berperan dalam pembuatan *splashscreen* awal program saat dibuka di *mobile* serta memberikan kemampuan untuk mengontrol bagaimana aplikasi terlihat oleh *user*.

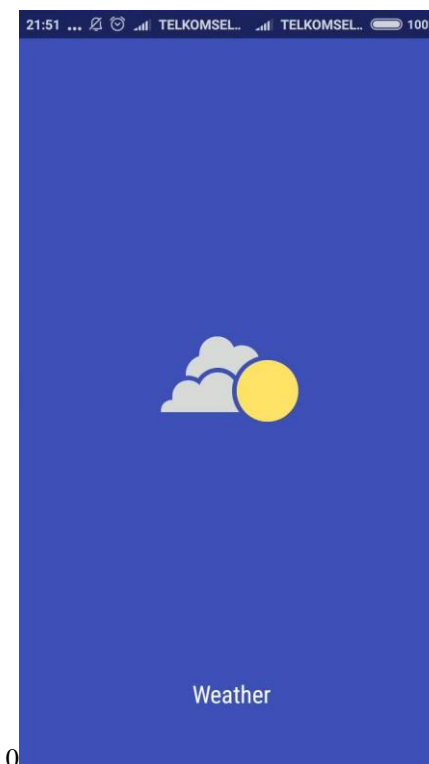
Pada *source code* diatas, terdapat beberapa pendefinisian, diawali dengan “*name*” yang mana berfungsi sebagai nama yang akan ditampilkan pada tampilan awal *splashscreen* program, kemudian “*short_name*” berfungsi sebagai nama yang diberikan pada *homescreen* program, “*theme_color*” berfungsi sebagai pemberi warna utama pada halaman dan untuk kode warna yang digunakan adalah #2F3BA2 yang mana berwarna biru dongker, dan “*background_color*” juga merupakan tampilan awal yang mana *backgroundnya* memiliki jenis *style color* atau berupa warna biru keputihan atau dengan kode warna #3E4EB8.

Untuk “*display*” diberikan jenis “*standalone*” agar jenis tampilan tidak seperti *web browser* dan lebih mengarah pada tampilan aplikasi *mobile* sungguhan, jika *user* ingin tampilan sesuai dengan *web browser*, bisa mengubah jenis *display* menjadi “*browser*”. Lalu “*start_url*” berfungsi sebagai halaman mana yang akan diakses pertama oleh aplikasi.

Dan yang terakhir adalah “*icons*” yang berfungsi untuk mensettings *icon* yang akan dipakai aplikasi, pada proyek ini masing-masing *icon* dipersiapkan dengan ditentukan “*src*” atau asal alamat *icon* tersebut, “*sizes*” untuk berapa besar ukuran *icon* yang digunakan, dan “*type*” merupakan tipe dari gambar yang digunakan.



Gambar 6.25 Hasil *deploy* aplikasi pada *icon homescreen*



Gambar 6.26 Tampilan *splash screen* aplikasi pada *mobile*



Gambar 6.27 Tampilan *index* aplikasi pada *mobile*

(Link Github Tugas: <https://github.com/saskiaugust/kel37-pwa2>)

6.6 Tugas

Pada percobaan tugas, diminta menambahkan *data weather* pada aplikasi dengan 3 buah kota yang ada di Indonesia dan memiliki data yang *realtime* atau tidak *fake*, kemudian diminta untuk membuat *footer* pada tampilan aplikasi, dan yang terakhir mengubah logo dan tampilan *splashscreen* pada *manifest*. Oleh karena itu dibutuhkan *source code* berikut pada *app.js*, *index.html*, dan *manifest.json*.

Adapun *source code* pada *app.js* adalah sebagai berikut:

```
(function() {
  'use strict';

  var app = {
    isLoading: true,
    visibleCards: {},
    selectedCities: [],
    spinner: document.querySelector('.loader'),
    cardTemplate: document.querySelector('.cardTemplate'),
    container: document.querySelector('.main'),
    addDialog: document.querySelector('.dialog-container'),
    daysOfWeek: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat',
'Sun']
  };

  document.getElementById('butRefresh').addEventListener('click',
function() {
  // Refresh all of the forecasts
  app.updateForecasts();
});

  document.getElementById('butAdd').addEventListener('click',
function() {
  // Open/show the add new city dialog
  app.toggleAddDialog(true);
});

  document.getElementById('butAddCity').addEventListener('click',
function() {
  // Add the newly selected city
  var select = document.getElementById('selectCityToAdd');
  var selected = select.options[select.selectedIndex];
  var key = selected.value;
  var label = selected.textContent;
  if (!app.selectedCities) {
    app.selectedCities = [];
  }
  app.getForecast(key, label);
  app.selectedCities.push({key: key, label: label});
  app.saveSelectedCities();
  app.toggleAddDialog(false);
});
});
```

```

});

document.getElementById('butAddCancel').addEventListener('click'
, function() {
    // Close the add new city dialog
    app.toggleAddDialog(false);
});

// Toggles the visibility of the add new city dialog.
app.toggleAddDialog = function(visible) {
    if (visible) {
        app.addDialog.classList.add('dialog-container--visible');
    } else {
        app.addDialog.classList.remove('dialog-container--
visible');
    }
};

app.toggleDelDialog = function(visible) {
    if (visible) {
        app.addDialog.classList.remove('dialog-container--
visible');
    } else {
        app.addDialog.classList.remove('dialog-container--
visible');
    }
};

// Updates a weather card with the latest weather forecast. If
the card
// doesn't already exist, it's cloned from the template.
app.updateForecastCard = function(data) {
    var dataLastUpdated = new Date(data.created);
    var sunrise = data.channel.astronomy.sunrise;
    var sunset = data.channel.astronomy.sunset;
    var current = data.channel.item.condition;
    var humidity = data.channel.atmosphere.humidity;
    var wind = data.channel.wind;

    var card = app.visibleCards[data.key];
    if (!card) {
        card = app.cardTemplate.cloneNode(true);
        card.classList.remove('cardTemplate');
        card.querySelector('.location').textContent = data.label;
        card.removeAttribute('hidden');
        app.container.appendChild(card);
        app.visibleCards[data.key] = card;
    }

    // Verifies the data provide is newer than what's already
visible
    // on the card, if it's not bail, if it is, continue and
update the
    // time saved in the card
    var cardLastUpdatedElem = card.querySelector('.card-last-

```

```

updated');
    var cardLastUpdated = cardLastUpdatedElem.textContent;
    if (cardLastUpdated) {
        cardLastUpdated = new Date(cardLastUpdated);
        // Bail if the card has more recent data than the data
        if (dataLastUpdated.getTime() < cardLastUpdated.getTime())
    {
        return;
    }
    }
    cardLastUpdatedElem.textContent = data.created;

    card.querySelector('.description').textContent =
current.text;
    card.querySelector('.date').textContent = current.date;
    card.querySelector('.current
.icon').classList.add(app.getIconClass(current.code));
    card.querySelector('.current .temperature
.value').textContent =
        Math.round(current.temp);
    card.querySelector('.current .sunrise').textContent =
sunrise;
    card.querySelector('.current .sunset').textContent = sunset;
    card.querySelector('.current .humidity').textContent =
        Math.round(humidity) + '%';
    card.querySelector('.current .wind .value').textContent =
        Math.round(wind.speed);
    card.querySelector('.current .wind .direction').textContent
= wind.direction;
    var nextDays = card.querySelectorAll('.future .oneday');
    var today = new Date();
    today = today.getDay();
    for (var i = 0; i < 7; i++) {
        var nextDay = nextDays[i];
        var daily = data.channel.item.forecast[i];
        if (daily && nextDay) {
            nextDay.querySelector('.date').textContent =
                app.daysOfWeek[(i + today) % 7];

nextDay.querySelector('.icon').classList.add(app.getIconClass(da
ily.code));
            nextDay.querySelector('.temp-high .value').textContent =
                Math.round(daily.high);
            nextDay.querySelector('.temp-low .value').textContent =
                Math.round(daily.low);
        }
    }
    if (app.isLoading) {
        app.spinner.setAttribute('hidden', true);
        app.container.removeAttribute('hidden');
        app.isLoading = false;
    }
};

/*
 * Gets a forecast for a specific city and updates the card

```

```

with the data.
    * getForecast() first checks if the weather data is in the
    cache. If so,
    * then it gets that data and populates the card with the
    cached data.
    * Then, getForecast() goes to the network for fresh data. If
    the network
    * request goes through, then the card gets updated a second
    time with the
    * freshest data.
    */
    app.getForecast = function(key, label) {
        var statement = 'select * from weather.forecast where
woeid=' + key;
        var url =
'https://query.yahooapis.com/v1/public/yql?format=json&q=' +
        statement;
        // TODO add cache logic here
        if ('caches' in window) {
            /*
            * Check if the service worker has already cached this
            city's weather
            * data. If the service worker has the data, then display
            the cached
            * data while the app fetches the latest data.
            */
            caches.match(url).then(function(response) {
                if (response) {
                    response.json().then(function updateFromCache(json) {
                        var results = json.query.results;
                        results.key = key;
                        results.label = label;
                        results.created = json.query.created;
                        app.updateForecastCard(results);
                    });
                }
            });
        }

        // Fetch the latest data.
        var request = new XMLHttpRequest();
        request.onreadystatechange = function() {
            if (request.readyState === XMLHttpRequest.DONE) {
                if (request.status === 200) {
                    var response = JSON.parse(request.response);
                    var results = response.query.results;
                    results.key = key;
                    results.label = label;
                    results.created = response.query.created;
                    app.updateForecastCard(results);
                }
            } else {
                // Return the initial weather forecast since no data is
                available.
                app.updateForecastCard(initialWeatherForecast);
            }
        }
    }

```

```

    };
    request.open('GET', url);
    request.send();
};

// Iterate all of the cards and attempt to get the latest
forecast data
app.updateForecasts = function() {
    var keys = Object.keys(app.visibleCards);
    keys.forEach(function(key) {
        app.getForecast(key);
    });
};

// TODO add saveSelectedCities function here
// Save list of cities to localStorage.
app.saveSelectedCities = function() {
    var selectedCities = JSON.stringify(app.selectedCities);
    localStorage.selectedCities = selectedCities;
};

app.getIconClass = function(weatherCode) {
    // Weather codes:
https://developer.yahoo.com/weather/documentation.html#codes
    weatherCode = parseInt(weatherCode);
    switch (weatherCode) {
        case 25: // cold
        case 32: // sunny
        case 33: // fair (night)
        case 34: // fair (day)
        case 36: // hot
        case 3200: // not available
            return 'clear-day';
        case 0: // tornado
        case 1: // tropical storm
        case 2: // hurricane
        case 6: // mixed rain and sleet
        case 8: // freezing drizzle
        case 9: // drizzle
        case 10: // freezing rain
        case 11: // showers
        case 12: // showers
        case 17: // hail
        case 35: // mixed rain and hail
        case 40: // scattered showers
            return 'rain';
        case 3: // severe thunderstorms
        case 4: // thunderstorms
        case 37: // isolated thunderstorms
        case 38: // scattered thunderstorms
        case 39: // scattered thunderstorms (not a typo)
        case 45: // thundershowers
        case 47: // isolated thundershowers
            return 'thunderstorms';
        case 5: // mixed rain and snow

```

```

        case 7: // mixed snow and sleet
        case 13: // snow flurries
        case 14: // light snow showers
        case 16: // snow
        case 18: // sleet
        case 41: // heavy snow
        case 42: // scattered snow showers
        case 43: // heavy snow
        case 46: // snow showers
            return 'snow';
        case 15: // blowing snow
        case 19: // dust
        case 20: // foggy
        case 21: // haze
        case 22: // smoky
            return 'fog';
        case 24: // windy
        case 23: // blustery
            return 'windy';
        case 26: // cloudy
        case 27: // mostly cloudy (night)
        case 28: // mostly cloudy (day)
        case 31: // clear (night)
            return 'cloudy';
        case 29: // partly cloudy (night)
        case 30: // partly cloudy (day)
        case 44: // partly cloudy
            return 'partly-cloudy-day';
    }
};

/*
 * Fake weather data that is presented when the user first
uses the app,
 * or when the user has not saved any cities. See startup code
for more
 * discussion.
 */
var initialWeatherForecast = {
    key: '2459115',
    label: 'New York, NY',
    created: '2016-07-22T01:00:00Z',
    channel: {
        astronomy: {
            sunrise: "5:43 am",
            sunset: "8:21 pm"
        },
        item: {
            condition: {
                text: "Windy",
                date: "Thu, 21 Jul 2016 09:00 PM EDT",
                temp: 56,
                code: 24
            },
            forecast: [
                {code: 44, high: 86, low: 70},

```

```

        {code: 44, high: 94, low: 73},
        {code: 4, high: 95, low: 78},
        {code: 24, high: 75, low: 89},
        {code: 24, high: 89, low: 77},
        {code: 44, high: 92, low: 79},
        {code: 44, high: 89, low: 77}
      ]
    },
    atmosphere: {
      humidity: 56
    },
    wind: {
      speed: 25,
      direction: 195
    }
  }
};
// TODO uncomment line below to test app with fake data
app.updateForecastCard(initialWeatherForecast);

// TODO add startup code here
app.selectedCities = localStorage.selectedCities;
if (app.selectedCities) {
  app.selectedCities = JSON.parse(app.selectedCities);
  app.selectedCities.forEach(function(city) {
    app.getForecast(city.key, city.label);
  });
} else {
  /* The user is using the app for the first time, or the user
has not
  * saved any cities, so show the user some fake data. A real
app in this
  * scenario could guess the user's location via IP lookup
and then inject
  * that data into the page.
  */
  app.updateForecastCard(initialWeatherForecast);
  app.selectedCities = [
    {key: initialWeatherForecast.key, label:
initialWeatherForecast.label}
  ];
  app.saveSelectedCities();
}

// TODO add service worker code here
if ('serviceWorker' in navigator) {
  navigator.serviceWorker
    .register('./service-worker.js')
    .then(function() { console.log('Service Worker
Registered'); });
}
})();

```

File `app.js` merupakan file tempat dimana segala fungsi dibuat dan diberikan deskripsi, pada file `app.js` juga terdapat banyak fungsi yang memiliki kegunaan masing masing, dapat dimisalkan mulai dari *updateForecasts* merupakan fungsi yang berguna untuk me *refresh* data *weather* yang *realtime*, kemudian *butAddCity* untuk fungsi menambah daftar kota yang hendak dilihat data *weathernya*, lalu ada *weather code* yang berfungsi sebagai pendeskripsian kode *weather* pada setiap kondisinya. Kemudian diakhir *source code* juga terdapat pengaturan *service-worker* yang digunakan saat *web server* memasuki mode *offline*. Kemudian dilanjutkan dengan *source index.html* sebagai berikut:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="manifest" href="/manifest.json">
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <link rel="canonical" href="https://weather-pwa-
sample.firebaseio.com/final/">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Weather PWA</title>
  <link rel="stylesheet" type="text/css"
href="styles/inline.css">

  <!-- TODO add manifest here -->

</head>
<body>

  <header class="header">
    <h1 class="header_title">Weather PWA</h1>
    <button id="butRefresh" class="headerButton" aria-
label="Refresh"></button>
    <button id="butAdd" class="headerButton" aria-
label="Add"></button>
  </header>

  <main class="main">

    <div class="card cardTemplate weather-forecast" hidden>
      <!-- <button id="butDel" class="cardButton" aria-
label="Delete"></button> -->
      <div class="city-key" hidden></div>
      <div class="card-last-updated" hidden></div>
      <div class="location"></div>
      <div class="date"></div>

      <div class="description"></div>
      <div class="current">
        <div class="visual">
```



```

        <div class="icon"></div>
        <div class="temperature">
            <span class="value"></span><span
class="scale">°F</span>
        </div>
    </div>
    <div class="description">
        <div class="humidity"></div>
        <div class="wind">
            <span class="value"></span>
            <span class="scale">mph</span>
            <span class="direction"></span>°
        </div>
        <div class="sunrise"></div>
        <div class="sunset"></div>
    </div>
</div>
<div class="future">
    <div class="oneday">
        <div class="date"></div>
        <div class="icon"></div>
        <div class="temp-high">
            <span class="value"></span>°
        </div>
        <div class="temp-low">
            <span class="value"></span>°
        </div>
    </div>
    <div class="oneday">
        <div class="date"></div>
        <div class="icon"></div>
        <div class="temp-high">
            <span class="value"></span>°
        </div>
        <div class="temp-low">
            <span class="value"></span>°
        </div>
    </div>
    <div class="oneday">
        <div class="date"></div>
        <div class="icon"></div>
        <div class="temp-high">
            <span class="value"></span>°
        </div>
        <div class="temp-low">
            <span class="value"></span>°
        </div>
    </div>
    <div class="oneday">
        <div class="date"></div>
        <div class="icon"></div>
        <div class="temp-high">
            <span class="value"></span>°
        </div>
        <div class="temp-low">
            <span class="value"></span>°
        </div>
    </div>

```

```

        </div>
    </div>
    <div class="oneday">
        <div class="date"></div>
        <div class="icon"></div>
        <div class="temp-high">
            <span class="value"></span>°
        </div>
        <div class="temp-low">
            <span class="value"></span>°
        </div>
    </div>
    <div class="oneday">
        <div class="date"></div>
        <div class="icon"></div>
        <div class="temp-high">
            <span class="value"></span>°
        </div>
        <div class="temp-low">
            <span class="value"></span>°
        </div>
    </div>
    <div class="oneday">
        <div class="date"></div>
        <div class="icon"></div>
        <div class="temp-high">
            <span class="value"></span>°
        </div>
        <div class="temp-low">
            <span class="value"></span>°
        </div>
    </div>
</div>
</div>
</main>

<div class="dialog-container">
    <div class="dialog">

        <div class="dialog-title">Add new city</div>
        <div class="dialog-body">
            <select id="selectCityToAdd">
                <!-- Values map to Yahoo Weather API Where On Earth
Identifiers (WOEIDs).
https://developer.yahoo.com/weather/documentation.html#req -->
                <option value="2357536">Austin, TX</option>
                <option value="2367105">Boston, MA</option>
                <option value="2379574">Chicago, IL</option>
                <option value="2459115">New York, NY</option>
                <option value="2475687">Portland, OR</option>
                <option value="2487956">San Francisco, CA</option>
                <option value="2490383">Seattle, WA</option>
                <option value="1047592">Kendal, INA</option>
                <option value="56000422">Tegal, INA</option>
                <option value="1048324">Semarang, INA</option>
            </select>
        </div>
    </div>
</div>

```

```

        </select>
    </div>
    <div class="dialog-buttons">
        <button id="butAddCity" class="button">Add</button>
        <button id="butAddCancel" class="button">Cancel</button>
    </div>
</div>
</div>

<div class="loader">
    <svg viewBox="0 0 32 32" width="32" height="32">
        <circle id="spinner" cx="16" cy="16" r="14"
fill="none"></circle>
    </svg>
</div>

<!-- Uncomment the line below when ready to test with fake
data -->
<script src="scripts/app.js" async></script>
<div id="footer">
    <center>
        Modul 5 Praktikum MDP 2018 - Created by : Agus dan
        Saskia (Kelompok 37)
        Copyright &copy; 2018

    </center>
</div>

</body>
</html>

```

Index.html merupakan file berjenis html yang berisikan pengaturan apa saja yang akan ditampilkan sebagai *interface* yang akan dilihat pertama kali oleh *user*, index.html pada percobaan kali ini berisikan tentang pengaturan *background*, gambar - gambar yang hendak ditampilkan, pengaturan tampilan data dari data yang diambil pada suatu sumber dan mengolahnya, dan pada percobaan tugas, diambil data dari *Yahoo Weather API* yang bersumber dari <https://developer.yahoo.com/weather/documentation.html#req> yang kemudian kita olah datanya dan menampilkannya dengan *template* yang sudah disetting sebelumnya, pada dasarnya penggunaan *weather API* ini diberikan kode lokasi untuk setiap kota di dunia, jadi pada saat diakses kode tersebut, maka lokasinya akan menyesuaikan dengan data *weather* lokasi *Yahoo*, kemudian terdapat *button delete* yang belum bisa difungsikan karena masih dalam tahap pengembangan, yang nantinya akan berfungsi sebagai penghapus data *weather* kota yang dipilih untuk dihapus pada halaman utama. Adapun juga membuat dan mengatur

tampilan footer di index.html ini yang pada percobaan kali ini digunakan sedikit fitur css yang mana file css yang dapat diinput pada folder *scripts*.

Kemudian pada index.html juga terdapat tampilan pada saat ingin menambahkan daftar kota yang hendak dilihat data *weathernya*, diketahui data yang terdapat pada index.html sebanyak 11 kota, adapun pada percobaan tugas diminta untuk menambahkan 3 kota yang ada di Indonesia, oleh karena itu ditambahkan Kendal, Tegal, dan Semarang yang masing-masing memiliki kode lokasi 1047592, 56000422, dan 1048324. Yang nantinya kode lokasi ini lah yang akan menjadi patokan bagi *API Weather Yahoo* dalam memberikan data weather suatu kota. Lalu ada juga source code dari manifest.json sebagai berikut:

```
{
  "name": "Kelompok 37",
  "short_name": "Weather",
  "icons": [{
    "src": "images/icons/icon-128x128.png",
    "sizes": "128x128",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-144x144.png",
    "sizes": "144x144",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-152x152.png",
    "sizes": "152x152",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-192x192.png",
    "sizes": "192x192",
    "type": "image/png"
  }, {
    "src": "images/icons/icon-256x256.png",
    "sizes": "256x256",
    "type": "image/png"
  }],
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#7f7fbf",
  "theme_color": "#ae5daf"
}
```

Manifest.json merupakan file berekstensi json, yaitu Bahasa pemrograman kekinian yang digunakan, dan butuh *library* penerjemah untuk menerjemahkan Bahasa json. File manifest.json pada proyek kali ini berfungsi sebagai file yang membuat suatu tampilan pada mobile saat *icon* pada *homescreen* di tekan. Karena pada dasarnya proyek ini merupakan proyek html yang diperuntukan untuk web,

namun agar tampilan pada mobile bisa lebih di tolerir, maka fungsi `manifest.json` ini lah yang berperan dalam pembuatan *splashscreen* awal program saat dibuka di *mobile* serta memberikan kemampuan untuk mengontrol bagaimana aplikasi terlihat oleh *user*.

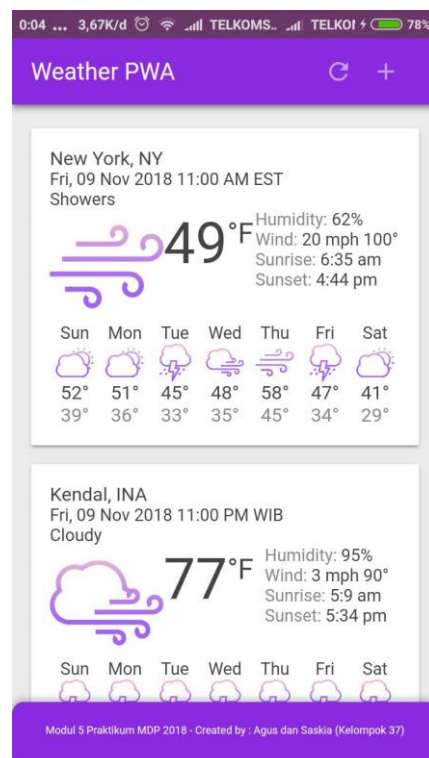
Pada *source code* diatas, terdapat beberapa pendefinisian, diawali dengan `"name"` yang mana berfungsi sebagai nama yang akan ditampilkan pada tampilan awal *splashscreen* program, kemudian `"short_name"` berfungsi sebagai nama yang diberikan pada *homescreen* program, `"theme_color"` berfungsi sebagai pemberi warna utama pada halaman dan untuk kode warna yang digunakan adalah `#ae5daf` yang mana berwarna werah pucat, dan `"background_color"` juga merupakan tampilan awal yang mana backgroundnya memiliki jenis *style color* atau berupa warna putih berkode `#7f7fbf`, yang pada percobaan kali ini digunakan warna sebagai background.

Untuk `"display"` diberikan jenis `"standalone"` agar jenis tampilan tidak seperti web browser dan lebih mengarah pada tampilan aplikasi *mobile* sungguhan, jika *user* ingin tampilan sesuai dengan web browser, bisa mengubah jenis *display* menjadi `"browser"`. Lalu `"start_url"` berfungsi sebagai halaman mana yang akan diakses pertama oleh aplikasi.

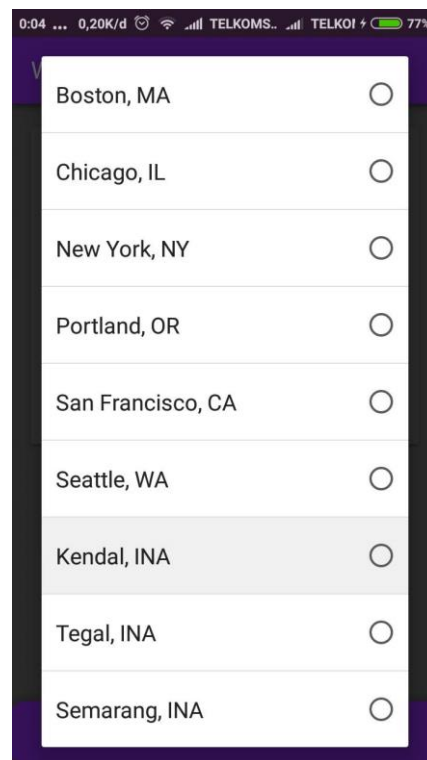
Dan yang terakhir adalah `"icons"` yang berfungsi untuk mensettings icon yang akan dipakai aplikasi, pada proyek ini masing-masing icon dipersiapkan dengan ditentukan `"src"` atau asal alamat icon tersebut, `"sizes"` untuk berapa besar ukuran icon yang digunakan, dan `"type"` merupakan tipe dari gambar yang digunakan.



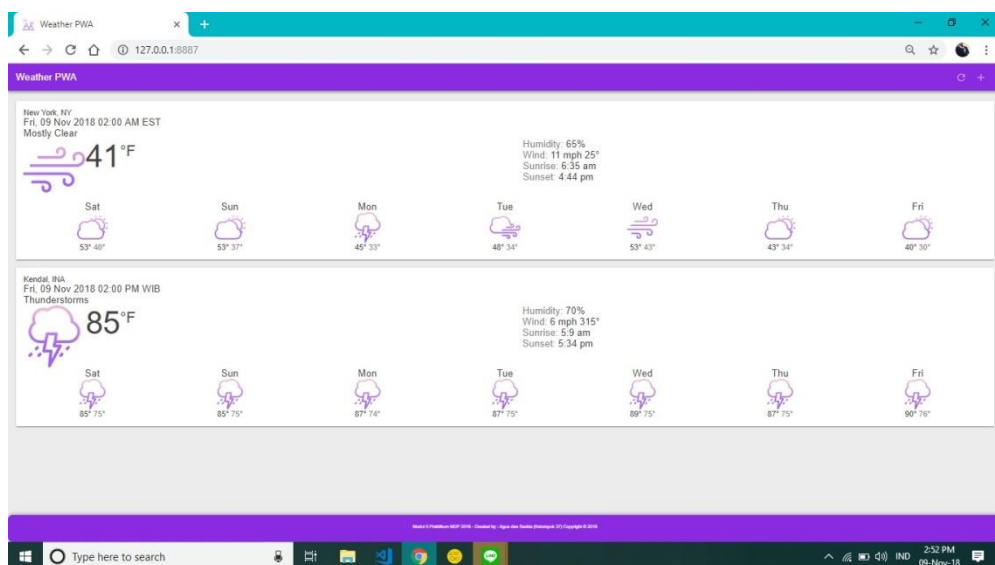
Gambar 6.28 Tampilan *splash screen* aplikasi pada *mobile*

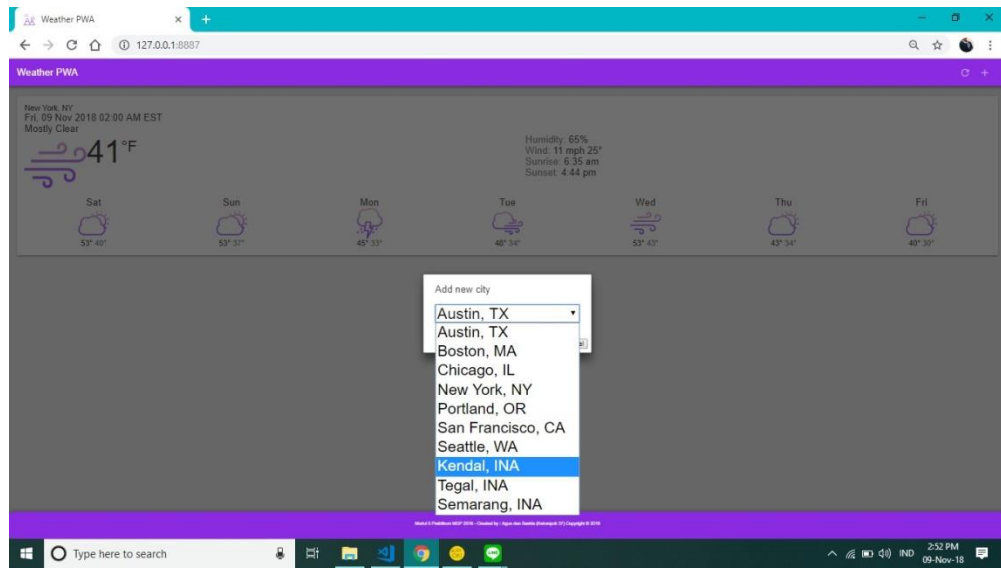


Gambar 6.29 Tampilan data *weather real time* beserta *footer*



Gambar 6.30 Tampilan pilihan kota

Gambar 6.31 Tampilan data *weather real time* pada *browser*



Gambar 6.32 Tampilan pilihan kota

(Link github tugas: <https://github.com/saskiaugust/kel37-pwa2>)

6.7 Kesimpulan

1. Index.html berfungsi sebagai file yang bertanggung jawab atas sebagai proses pembentukan *interface* utama.
2. Manifest.json merupakan file yang berfungsi membuat tampilan proyek pada PWA terlihat lebih baik dan tidak berbasis web ketika ditampilkan lewat *mobile*.
3. App.js merupakan file yang bertanggung jawab dalam teknis menampilkan data pada proyek PWA.
4. Penggunaan PWA dalam pembuatan aplikasi *mobile* tergolong lebih ringan dan dinamis.
5. Aplikasi berbasis PWA cenderung lebih mudah dalam pengelolaan data pada aplikasi PWA sama seperti basis web.
6. Dalam pembuatan aplikasi PWA, dapat menggunakan fitur *service-worker* jika hendak menjalankan aplikasi secara *offline*.
7. Kostumisasi aplikasi berbasis PWA cenderung lebih mudah karena menggunakan basis *Webpage* (php/html/dll).
8. *Cache Storage* dapat digunakan kembali dalam menampilkan *interface* jika aplikasi PWA sedang tidak terhubung koneksi internet.
9. Pengembangan aplikasi PWA saat ini sudah dapat diintegrasikan dengan berbagai jenis *API* (*Application Programming Interface*).
10. Aplikasi berbasis PWA didukung oleh *Google Firebase* dalam hal *deploying* terhadap *mobile*, sehingga dapat lebih membuat tampilan pada *mobile* terlihat baik.