Building Your First Angular PWA from Scratch (Angular 6)

Coming with features such as offline functionality, deployment without a third-party prerequisite, and quite a bit more, the growing popularity of progressive web apps **isn't** a surprise.

Their simple integration and variety of use-cases have made them a **massive** contributing factor to a lot of online programs, and the growing trend should certainly come with an in-depth tutorial!

That's why, in today's lesson, we'll be covering everything there is to know about PWAs, including:

- Understanding how to create a PWA
- Mastering design customization
- Understanding offline functionality
- **And quite a bit more**

Then, towards the end, I'll also guide you through the process of building your very own "Chuck Norris" joke generator using PWAs!

Here's what the project would look like when you load it up:

(You can see a preview of this project here, and the GitHub repo can be accessed here.)

Aside from that, I've got nothing more to say, so let's get started!

Nailing Down Installation

As always, before we can dive into anything else, we need to nail down our installation prerequisites. I'll guide you through everything chronologically:

1. Be sure to verify that the Angular CLI installed by running this command (as usual):

```
> ng -v
```

2. Then, just start a new project. Since our project is centered around a joke generator, we'll name our project based on the theme, so you can just run this command:

```
> ng new jokes
```

3. Then, you can just cd directly into the project by running this command:

```
> cd jokes
```

4. After you've got all that, we'll create a service file that will be used for this project via this command:

```
> ng g s data
```

5. Now, we can finally dive into the code! If you have Visual Studio Code on your device, which is what Gary uses for the video tutorial, you can access it with all the files you just created by running this command:

```
> code .
```

Running Our Project

Very briefly, just to test out what we've got so far, we can open up our project by running these commands:

```
> ng build --prod
```

After that's run, cd into the new folder you created:

```
> cd dist\jokes
```

Now, just run this command to run a default project in the browser:

```
> http-server -o
```

After you've taken care of all that, your default browser should open up a window with the default Angular project, which looks like this:

If you've got this far, all the prerequisites are done, however, at the moment, this Angular project has no PWA functionality whatsoever.

That's why we'll now transition to experimenting with adding that extra layer to our app!

Playing Around with Offline Customization

One of the main benefits that having a PWA comes with is the **seamless** offline functionality, which could serve a **great** purpose both for the content creator and the content that a user would see.

In order to play with these features you just need to add PWA functionality to your app, which isn't too difficult of a process:

1. In the video, before running the PWA command, Gary enables offline functionality so we don't have to later. This can be done very easily from the Google developer tools, which can be accessed if you're using the Chrome browser. Just hit ctrl + shift + I, navigate to the 'application' tab, and at the very top, and select the 'offline' box, as I've shown below:

In this early phase of the process, you'd get your typical error once you hit offline and refresh, but we'll take care of this soon!

2. Now, we'll add the PWA functionality to see what happens! In order to do so, you'll first need to type in this cd command **twice**:

```
> cd..
```

Remember to do that twice before running the command below, which will add the PWA functionality to your app:

```
> ng add @angular/PWA
```

By running this command, you'll notice that your editor will add several new files that'll essentially give life to your apps PWA-specific functionality.

These files include:

- Manifest.json - This file lays out all the design and location specifications for your web app, including the color, name, URL, and icon size
- Nsgw-config.json - In the video tutorial, Gary refers to this file as the "heart" of what defines a PWA, and I'm sure that we could all agree. This file is like the control center for your PWA, and for this tutorial, it's what we'll be using to add the offline functionality
- *There are several others that we won't have to worry about right now*

Now, returning to our discussion on offline functionality, you'd then have to rerun the three commands I talked about earlier, which will update your Angular app:

```
> ng build --prod

> cd dist\jokes

> http-server -o
```

(Keep in mind that you'll have to rerun these lines every time you update your code for this particular project, so I'll tell you to return to this section very frequently)

Once you run that last command, a new window should open with an updated version of your project. Now, your app should run **despite** an offline environment:

As shown here, the offline option is checked, but the app still runs smoothly!

As one could assume, this feature has quite a bit of obvious benefits, including allowing your user to maintain sanity in the case of an offline environment, however, as Gary points out in the video version of this tutorial, it's not flawless.

Since the service workers for the Angular app are cached, element reloading on a site can be quite difficult and frustrating, which is what we'll be trying to resolve next...

Implementing Live Reloading Functionality

In order for us to add live reloading into our progressive web app, we'll need to go through a few extra steps:

1. First, we'll make a quick tweak to the code of our PWA just to use as a live reloading example. In the tutorial, Gary just went over to the app.component.html file and made a quick change to the text that's currently displayed at the top of our project. In order to replicate that, just replace the existing header code with this, which will change the statement from "welcome to app" to "welcome, app":

```
<h1>
Welcome, {{title}}!
</h1>
```

2. Now, we'll work towards adding live reloading so that our modification can be visible in the browser. Start by heading over to the pre-installed file in our project titled **app.component.ts**. In that file, reserve the second line to add this code, which will add something known as SwUpdate to your project:

```
> import {SwUpdate} from '@angular/ service-worker';
```

3. After you've got that done, we'll make a few more adjustments to our app.component.ts file. All you need to do is find this code below:

```
export class AppComponent {
  title = 'app';
```

Directly underneath that, we'll just add some info that will introduce SwUpdate to our application:

```
  update: boolean = false;
  joke: any;

  constructor(updates: SwUpdate, private data: DataService) {
    updates.available.subscribe(event => {

      updates.activateUpdate().then(() => document.location.reload());
```

4. Now, we'll set up a notification system that will notify us and the users of our app when an update has occurred. In order to do this, head over to the file titled app.compontent.html and reserve the third line for this code:

```
<span *ngIf="update">There's an update associated with your progressive web application!</span>
```

(You can type in any text you'd like. I just used this message as an example.)

5. Then, if everything goes accordingly, you should be able to rerun the project and see some results. Remember, to rerun the project, you'd first type **cd..** twice and then these three commands:

```
> ng build --prod

> cd dist\jokes

> http-server -o
```

Now, any changes you make in your code will be visible in your project when you rerun it, as shown with our "welcome, app" example:

As for the update notification that we set up in step 4, this can also be seen very briefly when we make an update, as Gary shows with his next modification of the app:

Creating Our Chuck Norris Joke Generator

If you've gone this far into the tutorial, congrats! It's now time to package everything into a mini project, which will just spit out random jokes that are somehow correlated with Chuck Norris!

Once again, I'll guide you through the process chronologically:

1. First, pull up the data.service.ts file. We'll import an Http client into our code that will help us run the project. You can do so by using this code:

```
import {HttpClient} from '@angular/common/http';
```

2. Then, in line 9, fill in the code with the following:

```
constructor(private http: HttpClient) {}
```

3. Then, we'll create a method right underneath this code, where we'll introduce an API from gimmeJokes:

```
gimmeJokes() {
  return this.http.get('https://api.chucknorris.io/jokes/random')
}
```

The API that we're using is derived from a site called chucknorris.io, which, as one may assume, offers an API that randomly generates Chuck Norris jokes. The site also has some other integrations, including one for a Slack app.

4. Followed by that, we'll make some adjustments in our app.module.ts file. Here, we'll import our HttpClient module and DataService by pasting this code into the editor:

```
import { HttpClientModule } from '@angular/common/http';
import { DataService } from './data.service';
```

After you have those implemented, make these adjustments to the code below to include them in your project:

5. Then, find this line from the **app.module.ts** file and paste it into line 3 of the **app.component.ts** file:

```
import {DataService} from './data.service';
```

6. Then, you'll need to make a few more adjustments to the app.component.ts. For this step, just make these adjustments to the code in the app.component.ts file:

```
constructor(updates: SwUpdate, private data: DataService) {
    updates.available.subscribe(event => {
      updates.activateUpdate().then(() => document.location.reload());

    })
  }

  ngOnInit() {
    this.data.gimmeJokes().subscribe(res => {
      this.joke = res;
    })
  }

}
```

*(Oh, and earlier on in this tutorial in one of the code snippets, I included the line **joke: any; ,** which was a contributing factor to this phase of the process. That line just tells us that there isn't any specific Chuck Norris joke that we want to summon when we reload our browser.)*

7. Followed by that, just head over to our app.component.html file and make some more customizations. In this case, you could essentially just remove all the code up to line 3 and write the following:

```
<h1>Jokes</h1>

<p *ngIf="joke">{{joke.value}}</p>
```

This code essentially specifies that every time we reload our PWA, a new joke will be displayed from the API we're using.

8. Now, just run the same three commands you were running the last several times you updated your application:

```
> ng build --prod

> cd dist\jokes

> http-server -o
```

Once that's done, you might still see your old project pop up in the browser. To fix that, just hit the reload button and embrace the beauty of your very own Chuck Norris joke generator:

Now, you have a partially-functioning version of the project, but this version still has two major issues that we'll have to tackle:

1. The design of the PWA is somewhat weak
2. When you reload the project in offline mode, a new joke does not show up

These issues are what we'll focus on next...

Fixing Bugs and Adding Customizations

We'll begin by going through the design customization step:

1. Let's say that we wanted to add a more visually appealing font to our project order to improve its overall design. In order to do so, we'd just use the typical HTML and CSS customization process. In this case, we'll test this out with the Montserrat font. In order to make this the default font, just open up the index.html file and run this code:

```
 <link href="https://fonts.googleapis.com/css?family=Montserrat" rel="stylesheet">
</head>
```

2. After you've got that taken care of, we'll add some standard CSS into our styles.css file to use as a rule set. You can just copy-paste this code to add it to your project:

```
body {
    background: brown;
    color: #fff;
    text-align: center;
    font-family: 'Montserrat';
    display: grid;
    align-items: center;
    font-size: 3em;
    padding: 2em;
}
```

3. Now, although it may be a bit tedious, if we want our font to be visible offline, we have to manually implement it. All you really need to do for this step is to head over to the **nsgw-config.json** file and add some code in one location.

- We'll add this code in order to allow our PWA to recognize the font in an offline environment:

```
"urls": [
        "https://fonts.googleapis.com/**"
      ]
    }
```

Now, if everything functions properly, a new font should be able to load on your app, but before we test that out, we have just one more thing to tackle, which is to allow offline functionality for new joke generation.

This process is **far** more straightforward than the design customization process, and everything is done through the **nsgw-config.json** file.

Starting in line 27, we'll create a location where our API endpoints will be stored by running the following code:

```
 }],
  "dataGroups": [
    {
      "name": "jokes-api",
      "urls": ["https://api.chucknorris.io/jokes/random"],
      "cacheConfig": {
        "strategy": "freshness",
        "maxSize": 20,
        "maxAge": "1h",
        "timeout": "5s"
      }
    }
  ]
}
```

The "name" and "urls" categories are quite straightforward in this code, however, the categories stored in cacheConfig definitely deserve a quick explanation:

- "strategy" defines the format in which caching will occur
- "maxSize" determines the number of responses that will be cached by our PWA
- "maxAge" determines the duration for which the cached response is valid
- "timeout" is used to determine when to display a 404 error message if a program doesn't run for some duration of time

Now, returning to our project, you will, once again, need to type in **cd..** and the three commands I mentioned before. Once you do that and hit the refresh button, you'll be presented with this in your browser running in offline mode:

(Keep in mind that, in offline mode, the PWA will only return the most recent result that was given when you were online, but in online mode, random jokes will appear every time you reload the page.)

Wrapping Everything Up

Whether it's the offline customization or the seamless deployment, PWAs are certainly among the most efficient ways to run a web app, and their popularity will continue to expand.

That's why, as always, you can feel free to refer to this article at any time in the future!

For now, however, if you've gone through the entire tutorial, you're all set to begin creating progressive web apps of your own!