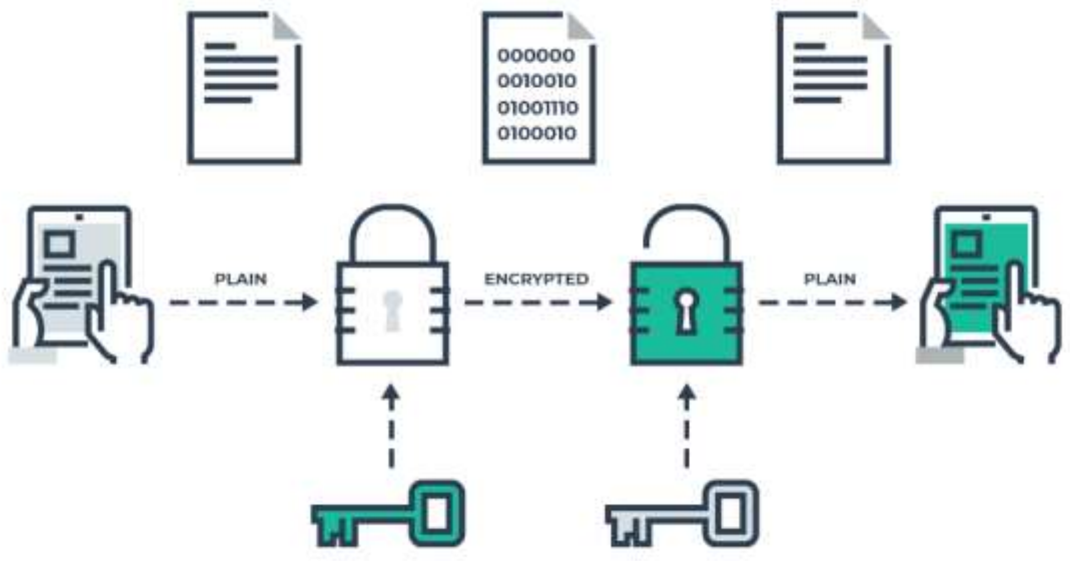
SECURE PASSWORDS IN A DATABASE:

Validating and authenticating users of an application is one of the major areas in the field of security. Storing passwords, the right way is of critical importance, yet it is something that many developers still fail in this day and age. In this section of the report, we will be looking at the different methods you can store your passwords without having them compromised.

Without a doubt, the single worst way you can store a password is by using plaintext. This is a tremendously bad practice and should be avoided at all costs. This approach simply stores the password as it is and so when the inevitable happens, anyone who now has access to your database has access to every one of your users' password. What's worse is that many people use the same password for all their accounts, so this means not only do they have their account in your website/application compromised but also others.

1. ENCRYPT PASSWORDS:

This is first approach in adding a layer of security on top of the passwords in your database. A quick illustration is displayed below:

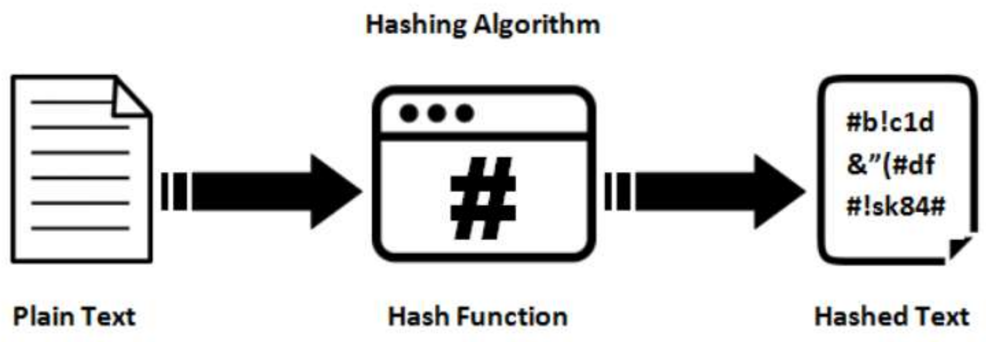


A key is generated first which will allow the users to encrypt messages. This will encrypt plaintext messages and turn them to ciphertext. After that, the same key can be used to decrypt the ciphertext and convert it into the original text, which is the password.

This method may be better than storing your passwords as plaintext, but it still a very bad practice for the sole reason that about 86% of all passwords are terrible. All it takes is for someone motivated enough to guess one of the passwords right and through that they will be able to derive the encryption key.

2. HASHING:

The second method is to hash your passwords on creation, instead of using encryption.

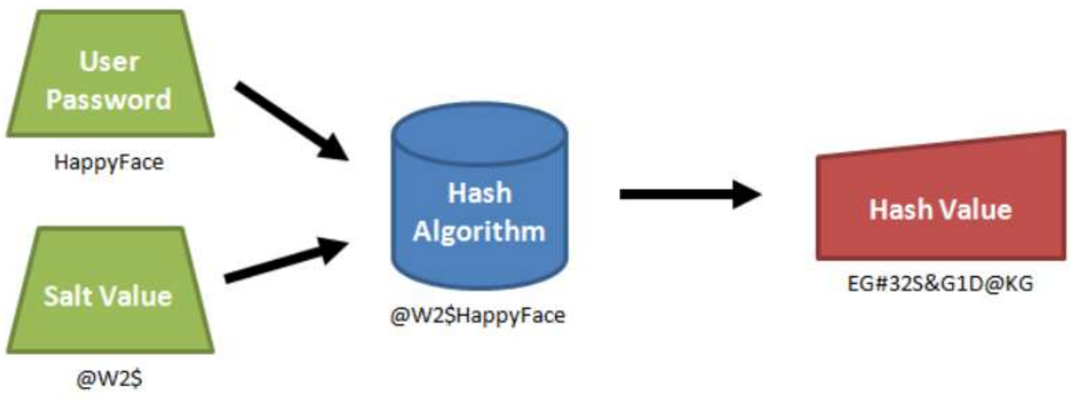


Its advantage is that it's non-invertible, meaning it is not computationally realistic to reconstruct the input from the output hash. The output hash is of the same length, being independent from the input hash. But the smallest of changes in the input can result in a completely different output.

The consequence of using this function is that there is a strong possibility of collision, because of two inputs being mapped to the same output hash. A cryptographic hashing function might just do the trick as it is generally considered to be collision free. A known example of this is the MD5, which used to be industry standard until 1996. The current industry standards are PBKDF2, scrypt and bcrypt.

3. HASHING + SALTING:

A salt is a long, randomly generated string that is concatenated with the password. This salt is generated upon every new user creation and is stored with the username and the hashed password.

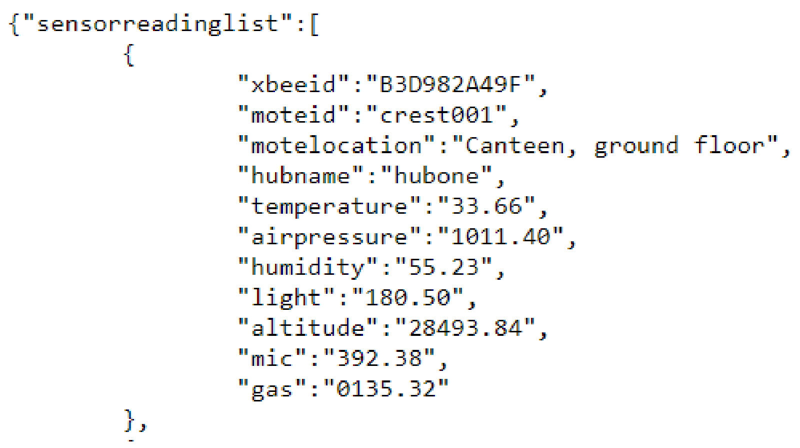


In this method, the password would generate a completely different hash every time it is used to create a new user. It would not match that of the same password created by a different user in the past. The only way of getting past this is by using a brute-force attack.

USING ANGULAR TO READ A JSON FILE:

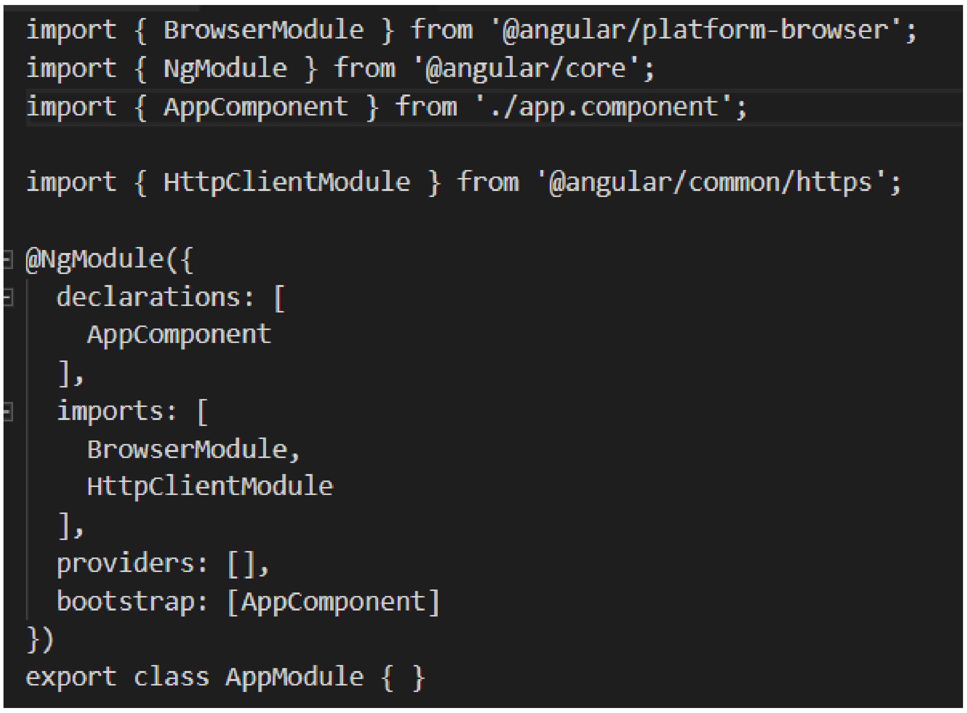
JSON or JavaScript Object Notation, as you know is a data format that allows you to conveniently store and share data using any medium. JSON is language independent and can be easily bind with any kind of application.

The screenshot below shows the JSON file which we need to display using our Angular application in a HTML table:


```
{"sensorreadinglist":[
        {
                "xbeeid":"B3D982A49F",
                "moteid":"crest001",
                "motelocation":"Canteen, ground floor",
                "hubname":"hubone",
                "temperature":"33.66",
                "airpressure":"1011.40",
                "humidity":"55.23",
                "light":"180.50",
                "altitude":"28493.84",
                "mic":"392.38",
                "gas":"0135.32"
        },
```

This is just an excerpt from the overall sensor.json file, and I have used it for reference.

Once we have setup Angular, downloaded the JSON file and saved it in an assets folder, we have to import the *HttpClientModule* to the project:

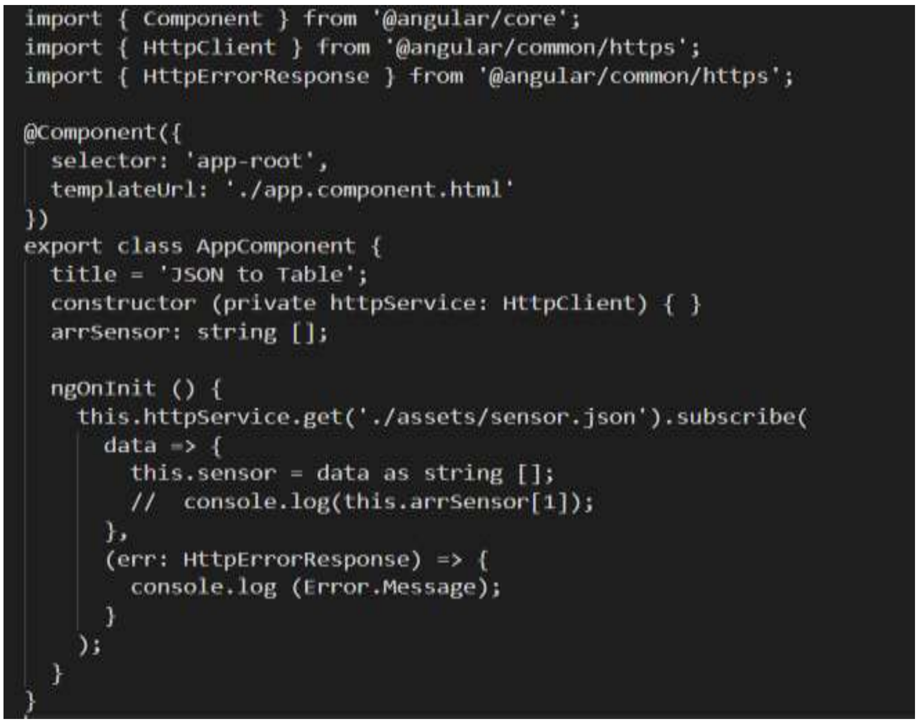


```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

import { HttpClientModule } from '@angular/common/https';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Next, we'll create our component, where we'll add/import HttpClient service. Adding this service to our project will ensure that we have access to the Get() method and its properties, which we need to access files in the server. Open app.component.ts file and add the below code:



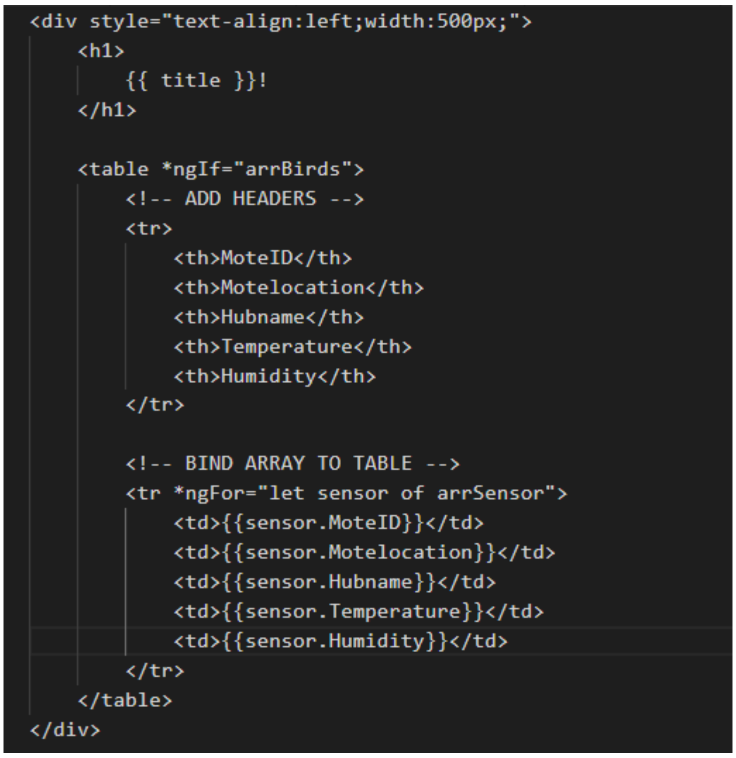
```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/https';
import { HttpErrorResponse } from '@angular/common/https';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  title = 'JSON to Table';
  constructor (private httpService: HttpClient) { }
  arrSensor: string [];

  ngOnInit () {
    this.httpService.get('./assets/sensor.json').subscribe(
      data => {
        this.sensor = data as string [];
        //  console.log(this.arrSensor[1]);
      },
      (err: HttpErrorResponse) => {
        console.log (Error.Message);
      }
    );
  }
}
```

We have declared an array named arrSensor of type string and added the JSON data extracted from the sensor.json file into an array, which will later be bound with a table using *ngFor* directive.

We will now create the application template, which is where we'll add the HTML <table> element and bind the array to the table:



```html
<div style="text-align:left;width:500px;">
    <h1>
        {{ title }}!
    </h1>

    <table *ngIf="arrBirds">
        <!-- ADD HEADERS -->
        <tr>
            <th>MoteID</th>
            <th>Motelocation</th>
            <th>Hubname</th>
            <th>Temperature</th>
            <th>Humidity</th>
        </tr>

        <!-- BIND ARRAY TO TABLE -->
        <tr *ngFor="let sensor of arrSensor">
            <td>{{sensor.MoteID}}</td>
            <td>{{sensor.Motelocation}}</td>
            <td>{{sensor.Hubname}}</td>
            <td>{{sensor.Temperature}}</td>
            <td>{{sensor.Humidity}}</td>
        </tr>
    </table>
</div>
```

I have only added a few elements from the sensor.json file here, but we get the overall picture from it.

We then save the file and go to the browser to check the output.

References:

Anon., N/A. *Encodedna.* [Online]
Available at: https://www.encodedna.com/angular/read-an-external-json-file-in-angular-4-and-convert-data-to-table.htm
[Accessed 30 May 2019].

LH, S., 2019. *ITNext.* [Online]
Available at: https://itnext.io/how-not-to-store-passwords-4955569e6e84
[Accessed 27 May 2019].