# Secure storage of passwords in database:

When creating a login system, the password needs to be stored effectively. An insecure and naive way to do is using plaintext. The password is stored as plaintext, and during login, the row which matches the inputted username is extracted and then the inputted password is compared with the password from that row for the user.

Regarding storage of passwords, a better way to do so is using encryption. The idea is the encryption of password using any generated key and then storing it into the database. The inputted password has to be encrypted using the same key upon login which will be compared to the one stored in the database. If a hacker hacks his way into the database, he would only get to see the password field in ciphertext. However, this method is still insecure because if the hacker finds the encryption key through trial and error, he gets to know all the passwords!

Another way to store passwords is hashing. We can apply a hashing function before storing the password and effectively resolve the issue introduced when using encryption. Benefits of hashing are:

1. It’s not possible to reconstruct the input from the output after hashing.
2. Smallest change made in the input will result in a completely different hashing output.
3. The hashing output is of fixed length irrespective of the length of input.
4. Same input will produce the same output each time which means that the hashing algorithm is a function.

A hashing function can be called as collision resistant if no two inputs exist that map to the same hashing output. Examples of collision resistant hashing algorithms are bcrypt, scrypt etc. However, even after applying the best way of hashing, the password will still be insecure.

The best way right now to protect against the security implications mentioned before, is by applying a salt to the password before performing hashing. A salt is a large, random string which is concatenated with the password. This salt is generated whenever a new user is created and is stored with the username and the hashed password. PHP uses password\_hash() and password\_verify() methods for salt-hashing.

# Using Angular to read sensor.json:

Since angular 6+ supports typescript, with the help of resolveJsonModule a local JSON files can be imported just like any other module. Eg: import SampleJson from '../../assets/SampleJson.json';

The application will crash unless we add “resolveJsonModule” and ”esModuleInterop” configurations and set their values to true:

{ "compilerOptions": { "resolveJsonModule": true, "esModuleInterop": true } }

Once the values are read, we can directly use them in our application. For example:

var app = angular.module("myApp", []);

app.controller("myCtrl", function($scope) {

$scope.records = SampleJson;

});

Inside the html, we can use ng-repeat to iterate through the JSON data. This is applicable for all versions of angular.

In case of angular 5 or 4, HttpClient from ‘@angular/common/http’ and Observable from ‘rxjs/Observable’ must be used. An example is shown below:

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

import { Observable } from 'rxjs';

export class ReadAComponent implements OnInit {

private url = 'assets/SampleJson.json';

constructor(private http: HttpClient) {

this.getJSON().subscribe(dat => {

console.log(dat);

});

}

public getJSON(): Observable<any> {

return this.http.get(this.url);

}

}

In case of angular version 4.3 below and angular 2 +, Http from “@angular/http” must be used to import JSON files.

import { Component, Input } from '@angular/core';

import { Injectable } from '@angular/core';

import { Http, Response, Headers, RequestOptions } from '@angular/http';

import {Observable} from 'rxjs/Rx';

import 'rxjs/add/operator/map';

import 'rxjs/add/operator/catch';

@Injectable()

export class ReadService{

private url = '../assets/SampleJson.json';

constructor(private http: Http) {

var obj;

this.getJSON().subscribe(dat => obj=data, error => console.log(error));

}

public getJSON(): Observable<any> {

return this.http.get(url)

.map((response:any) => response.json())

.catch((error:any) => console.log(error));

}

}

If the application is offline, angular Http will stop working. In that case the alternate way of importing JSON files is by adding a new file json-typings.d.ts in app folder along with index.html file.

declare module "\*.json" {

const value: any;

export default value;

}

After this small step, we can simply import JSON files like latest versions of Angular (6/7).

# References:

1. Simon LH, Jan 2019, How (not) to store passwords, Online - <https://itnext.io/how-not-to-store-passwords-4955569e6e84>
2. Arunkumar Gudelli, May 2019, How To Read Local JSON Files In Angular, Onlline - <https://www.angularjswiki.com/angular/how-to-read-local-json-files-in-angular/>