Corso di Programmazione Web e Mobile
A.A. 2021-2022

Password Manager

◖ Nikola, Marku, 910433 ◗

Autore: Microsoft Office User
Ultima modifica: 25 giugno 2015 - versione .....
Prima modifica: 14 maggio 2015

Password manager

1. Introduzione

Password manager è una single page application (SPA) utilizzabile da qualsiasi dispositivo dotato di browser. Permette agli utenti di gestire in maniera centralizzata e sicura le credenziali di siti e servizi. L'idea è quella di permettere ad un utente di accedere a tutte le sue credenziali mediante una singola password.

1.1. Breve analisi dei requisiti

1.1.1. Destinatari

Capacità e possibilità tecniche.

L'applicazione è rivolta a qualsiasi tipologia di utente, non è richiesto nessun livello di esperienza. Si presenta con un'interfaccia diretta e di facile intuizione, motivo per cui l'utente non ha bisogno di essere guidato per poterla utilizzare.

La banda necessaria richiesta per l'utilizzo dell'applicazione è minima. Nello specifico, l'utilizzo della banda si limita al caricamento iniziale ed alle funzioni di import/export dei dati.

Password Manager si può utilizzare da qualsiasi dispositivo dotato di browser.

1.1.2. Modello di valore

Il valore dell'applicazione risiede nel fatto che offre una funzionalità molto utile nella società attuale in quanto i siti, le applicazioni o i servizi a cui ci registriamo crescono in continuazione e doversi ricordare le credenziali, qualora non si utilizzino le stesse per ciascun sito, diventa un problema gestirle.

Il servizio che da più valore all'applicazione è la sua funzionalità principale, utilizzare una singola password per gestire l'insieme delle credenziali.

1.1.3. Flusso dei dati

Archiviare e organizzare i contenuti

I dati vengono cifrati e memorizzati nello storage locale del browser. Il formato è il seguente:

```
[
   {
       "name": "google.it",
       "username": "nikola.marku@studenti.unimi.it",
       "password": "XIaunkV53xv3Eus94WLZv0zdFSc="
   },
   {
       "name": "linkedin.com",
```

```
        "username": "nikola.marku2@studenti.unimi.it",
        "password": "XIaunm6MP5bYy8hkrtv1czTzGgT3U9lAZ+fO8g=="
    }
]
```


1.1.4. Aspetti tecnologici

Il front-end si basa sulle seguenti tecnologie:

- jQuery
- Bootstrap

Per soddisfare il requisito del progetto di poter essere utilizzato in modalità offline ho utilizzato i Service Workers, in quanto la feature di Cache Manifest è attualmente deprecata. La gestione della sicurezza, ovvero memorizzare in modo sicuro le password degli utenti nello storage locale, è stata sviluppata utilizzando le API standard Web Cryptography.

Il back-end si basa sul seguente stack:

- Apache2
- Node.js
- Redis

I contenuti statici vengono serviti dal web server apache. Per gestire la migrazione delle credenziali di un utente sono state sviluppate delle API REST con Node.js. Tali API fanno utilizzo di Redis, un database in-memory, per conservare temporaneamente le credenziali di un utente durante il processo di migrazione di esse.

Interfacce

L'interfaccia principale è un pannello di gestione in cui si può accedere a tutte le funzionalità offerte dal sistema e inoltre visualizzare la lista delle credenziali.

Lo stile di base dell'applicazione è quello offerto da Bootstrap. È presente un foglio di stile aggiuntivo che mette a punto alcuni dettagli.

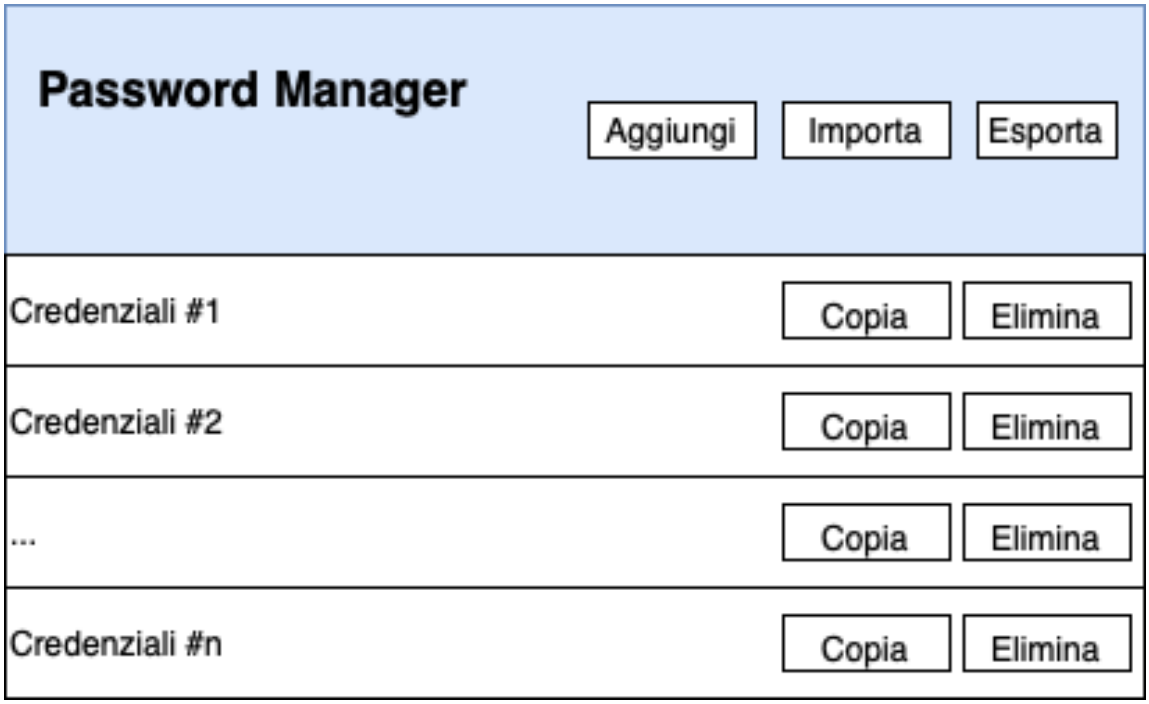


Figura 1: Interfaccia principale

Di seguito illustrate le interfacce delle finestre principali

Figura 2: Interfacce finestre

2. Architettura

2.1. Diagramma dell'ordine gerarchico delle risorse

Di seguito l'architettura interna del client

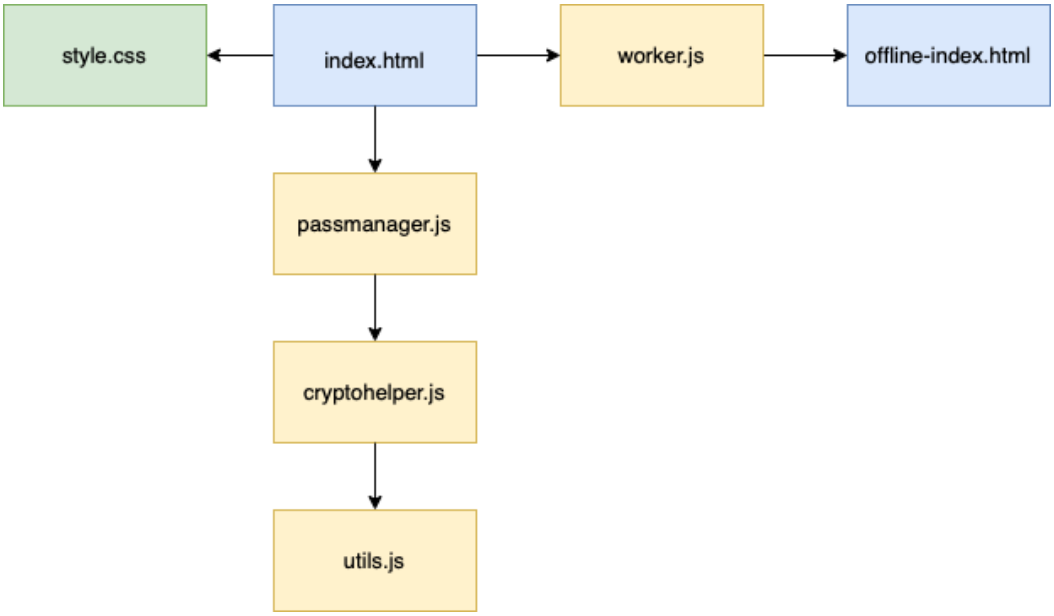


Figura 3: Architettura del sito

Questa invece è l'architettura di sistema

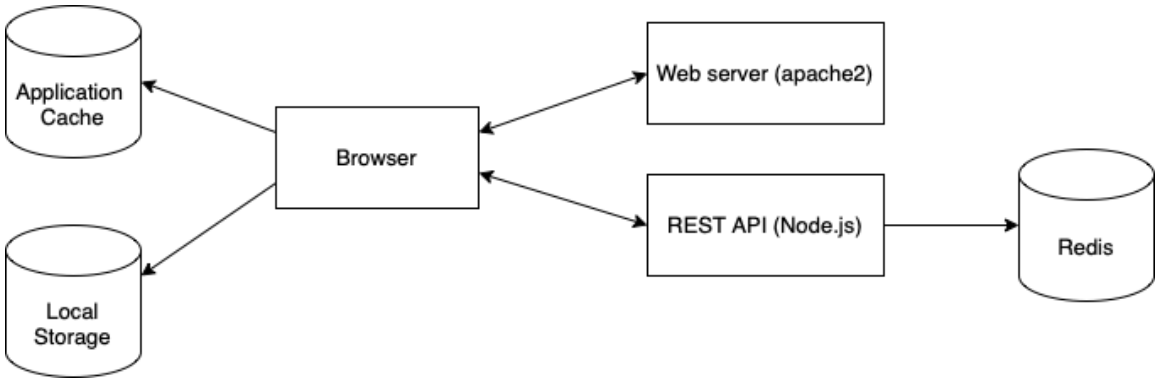
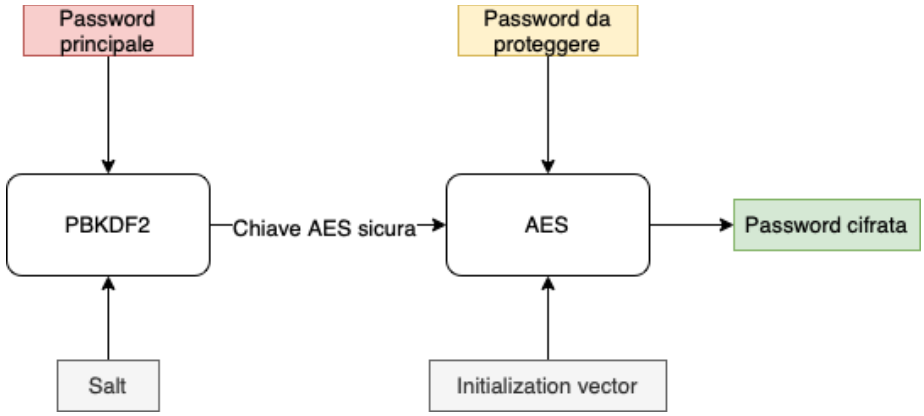


Figura 4: Architettura del sistema

2.2. Altri diagrammi

Di seguito viene illustrato il diagramma di attività che riguarda il processo di cifratura delle password delle credenziali da salvare.



Inizialmente viene applicata la funzione “PBKDF2” assieme ad un “salt” (una stringa random) sulla password principale, in modo da generare una chiave per l’algoritmo crittografico “AES”. PBKDF2 è una funzione di derivazione di una chiave volutamente lenta, in altre parole, è una funzione che genera una chiave sicura a partire da un testo generalmente debole (bassa entropia) e corto, come può essere una password. Utilizzare AES senza PBKDF2 è pericoloso in quanto solitamente un utente sceglie una password corta che permetterebbe ad un attaccante di effettuare un attacco di tipo brute force in tempi accettabili. Ipotizziamo che l’utente utilizzi una password di 5 caratteri su un alfabeto di 60 caratteri (per semplicità), l’ attaccante sarebbe in grado di attaccare mediante brute-force in 60^5 tentativi, che equivale a un costo computazionale di 2^{30} circa. Con la potenza delle attuali macchine un attacco del genere andrebbe a termine in poco tempo. Inoltre, l’attaccante ci metterebbe ancora meno tempo con un attacco di tipo dizionario, qualora la password principale scelta sia una parola relativamente comune. Utilizzando PBKDF2 ci assicuriamo di avere una chiave con alta entropia e lunga esattamente quanto richiesto da AES.

3. Codice

3.1. JavaScript

```
PasswordManager.prototype.storeCredentials = async function (
    name,
    username,
    password
) {
    let storage = window.localStorage;
    let encryptedPassword = await this.cryptoHelper.encrypt(password);
    this.data.push({
        name: name,
        username: username,
        password: encryptedPassword,
    });
    storage.setItem("user-data", JSON.stringify(this.data));
};
```

```
async function encrypt(key, iv, input) {
    let encodedInput = new TextEncoder().encode(input);
    return await window.crypto.subtle.encrypt(
        {
            name: "AES-GCM",
            iv: iv,
        },
        key,
        encodedInput
    );
}
```

```
function CryptoHelper(password) {
    let params = {
        salt: getUint8ArrayFromLocalStorage("salt"),
        iv: getUint8ArrayFromLocalStorage("iv")
    };
    //usamo chiusura per nascondere la password
    return {
        //parametro: stringa in chiaro
        //return: ct in base64
        encrypt: async (plaintext) =>  {
            let importedKey = await importKey(password);
            let aesKey = await getAESKey(importedKey, params.salt);
            let arrayBuffer = await encrypt(aesKey, params.iv, plaintext);
            return arrToBase64(new Uint8Array(arrayBuffer));
```

```
        },
        //Parametro: ct in base64
        //Return: stringa in chiaro
        decrypt: async (ciphertext) => {
            let importedKey = await importKey(password);
            let aesKey = await getAESKey(importedKey, params.salt);
            return await decrypt(base64toArr(ciphertext), params.iv, aesKey);
        }
    };
}
```

Nota: Sfrutto il concetto di chiusura nella funzione `CryptoHelper` per nascondere la password in chiaro dall'oggetto. Se avessi scritto la funzione come un normale costruttore avrei dovuto memorizzare `this.password = password` che rende insicura l'applicazione perché un eventuale attaccante che ha accesso al browser potrebbe stampare in console il contenuto del dell'oggetto e vedere la password in chiaro.

3.2. Node.js

```
app.get("/import/:id", async (req, res) => {
    const val = await redisClient.get(req.params.id);
    if(val === null)
        res.send({valid: false, message: 'Non trovato'});
    else{
        await redisClient.del(req.params.id);
        res.send({valid:true, data: JSON.parse(val)});
    }
});

app.post("/export", async (req, res) => {
    var token = null;
    do{
        token = util.randomString(5);
        var val = await redisClient.get(token)
    }while(val !== null);
    await redisClient.set(token,JSON.stringify(req.body))
    res.send({valid: true, token: token});
});
```

4. Conclusioni

L’applicazione è stata deployata su un server e per farla funzionare ho dovuto impostare server apache e node.js in modo che lavorassero su HTTPS perché le API standard Web Cryptography non vengono rese disponibili dal browser quando si trova in una pagina HTTP. Inoltre per permettere il funzionamento offline dell’applicazione ho riscontrato difficolta con il manifest per gestire la cache perché ho scoperto che tale funzionalità è deprecata in favore dei Service Workers. Ho cercato di sfruttare al meglio le mie conoscenze di crittografia per cercare di rendere la gestione dei dati il più sicura possibile.