📆 Hari 3 (Rabu) – Hashing Password dengan Bcrypt

🎯 Tujuan Pembelajaran

- Memahami pentingnya hashing password untuk keamanan data pengguna.
- Mampu mengimplementasikan proses hashing password menggunakan library bcrypt.
- Mengintegrasikan hashing password ke dalam alur register dan login.

🧠 Materi Inti (2 Jam)

1. Pentingnya Hashing Password

- **Risiko Menyimpan Password Plain Text:** Jika database mengalami kebocoran data, semua password pengguna akan terekspos dan dapat disalahgunakan.
- **Tujuan Hashing:** Mengubah password asli menjadi string acak yang tidak dapat dikembalikan ke bentuk semula (one-way function). Ini melindungi password pengguna meskipun data hash password bocor.
- **Salt:** String acak unik yang ditambahkan ke password sebelum di-hash. Salt memastikan bahwa dua pengguna dengan password yang sama akan memiliki hash yang berbeda. Ini melindungi dari serangan rainbow table.

2. Menggunakan Bcrypt untuk Hashing

- **Bcrypt:** Algoritma hashing password yang dirancang untuk menjadi lambat secara komputasi, membuatnya lebih tahan terhadap serangan brute-force dibandingkan algoritma hashing cepat seperti MD5 atau SHA-256.
- **Instalasi:** `npm install bcrypt`
- **Hashing:** Menggunakan fungsi `bcrypt.hash(password, saltRounds, callback)` atau `bcrypt.hashSync(password, saltRounds)`. `saltRounds` menentukan kompleksitas hashing (semakin tinggi, semakin aman tapi semakin lambat).

```
const bcrypt = require('bcrypt');
const saltRounds = 10; // Nilai yang umum digunakan

bcrypt.hash('password123', saltRounds, (err, hash) => {
  if (err) {
    // Tangani error
  } else {
    // Simpan hash ke database
    console.log(hash);
  }
});

// Atau secara sinkron (hati-hati di aplikasi web karena bisa
memblokir event loop)
const hashSync = bcrypt.hashSync('password123', saltRounds);
console.log(hashSync);
```

- **Verifikasi:** Menggunakan fungsi `bcrypt.compare(passwordInput, hashDatabase, callback)` atau `bcrypt.compareSync(passwordInput, hashDatabase)`. Fungsi ini membandingkan password yang diinput pengguna dengan hash yang tersimpan di database.

```
const bcrypt = require('bcrypt');

// Saat login, ambil password input dan hash dari database
const passwordInput = 'password123';
const hashDatabase =
'$2b$10$abcdefghijklmnopqrstuvwxy.abcdefghijklmnopqrstuvwxy'; //
Contoh hash

bcrypt.compare(passwordInput, hashDatabase, (err, result) => {
  if (err) {
    // Tangani error
  } else if (result) {
    // Password cocok, lanjutkan proses login
    console.log('Password cocok!');
  } else {
    // Password tidak cocok
    console.log('Password salah!');
  }
});

// Atau secara sinkron
const resultSync = bcrypt.compareSync(passwordInput, hashDatabase);
console.log(resultSync);
```

3. Mengintegrasikan Bcrypt ke Alur Autentikasi

- **Register:** Saat pengguna mendaftar, hash password yang mereka berikan sebelum menyimpannya ke database.
- **Login:** Saat pengguna mencoba login, ambil hash password dari database berdasarkan username/email, lalu gunakan `bcrypt.compare()` untuk memverifikasi password yang diinput.

📝 Praktik Mandiri (8 Jam)

1. **Refactor Register:** Modifikasi endpoint register (`POST /api/register`) yang sudah dibuat di Hari 1. Pastikan password di-hash menggunakan `bcrypt.hash()` sebelum disimpan ke database.
2. **Refactor Login:** Modifikasi endpoint login (`POST /api/login`). Saat memverifikasi kredensial, gunakan `bcrypt.compare()` untuk membandingkan password yang diinput dengan hash yang diambil dari database.
3. **Uji Kembali Alur Autentikasi:**
    - Daftar pengguna baru.
    - Cek database, pastikan password tersimpan dalam bentuk hash.
    - Coba login dengan password yang benar. Pastikan berhasil dan mendapatkan JWT.

    - Coba login dengan password yang salah. Pastikan gagal dan mendapatkan response error.
4. **Eksplorasi Salt Rounds:** Coba ubah nilai `saltRounds` saat hashing dan perhatikan perbedaannya (misalnya, 12 atau 14). Rasakan dampaknya terhadap waktu proses hashing.

💡 Tips untuk Pemula

- Gunakan versi asynchronous dari `bcrypt.hash()` dan `bcrypt.compare()` di aplikasi web/server untuk menghindari blocking event loop, terutama jika jumlah request tinggi.
- Pilih nilai `saltRounds` yang seimbang antara keamanan dan performa. Nilai 10-12 umumnya sudah cukup baik.
- Jangan lupa menangani error yang mungkin terjadi selama proses hashing atau perbandingan.

📚 Referensi

- Bcrypt Docs (npm)
- OWASP Password Cheatsheet
- Penjelasan Salt dan Pepper