FORMATION DEV WEB WEB MOBILE

CCP 2 - DÉVELOPPER LA PARTIE BACK-END D'UNE APPLICATION WEB

NODE.JS

JOUR 4 : AUTHENTIFICATION ET JWT

PRESENTE PAR NATACHA DESSE

AFEC

DEROULE MODULE

- Découverte de Node.JS
- Développer une API REST avec Express
- Gestion d’erreurs et middleware
- **Authentification et JWT**
- Routing et Accès
- Documentation dans un projet web

LES OBJECTIFS

Comprendre le principe d’authentification stateless avec JWT

Mettre en place un système de login sécurisé avec Express

Générer, signer, et vérifier des tokens JWT

Protéger des routes à l’aide d’un middleware

NUAGE DE MOTS

Quels principes de cybersécurité connaissez vous ?

https://www.menti.com/aldpp4773piz

CYBERSECURITE

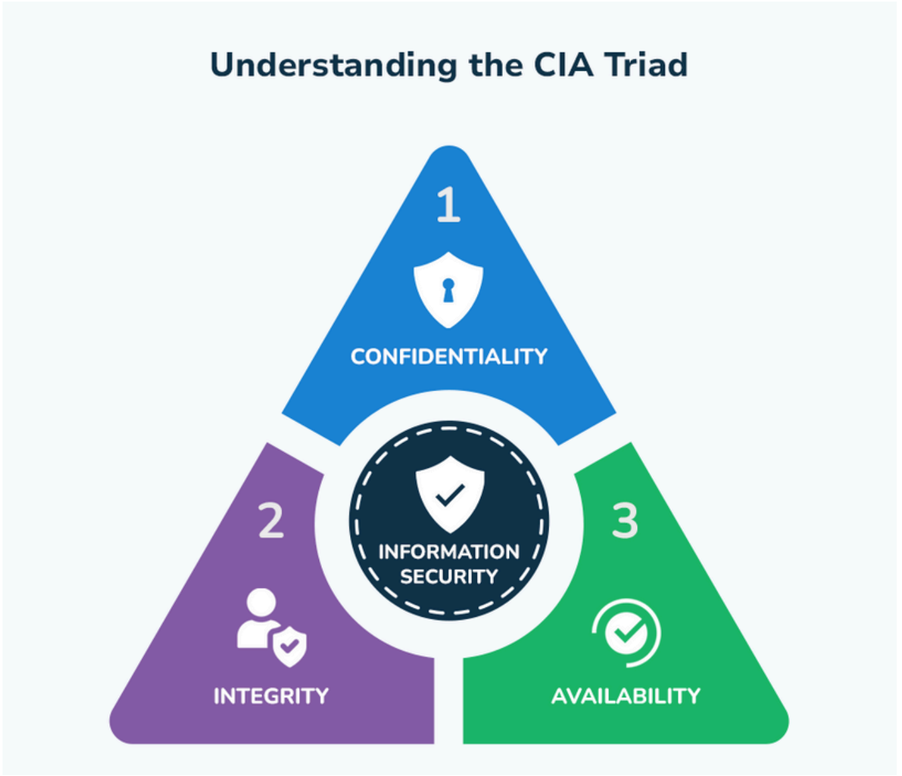
LES GRANDS PRINCIPES

La cybersécurité est un domaine essentiel qui englobe l'**ensemble** des **pratiques**, **technologies** et **processus** mis en œuvre pour **protéger** les systèmes informatiques, les réseaux et les données contre les cybermenaces.

Elle vise à garantir

- la **confidentialité**
- l'**intégrité**
- la **disponibilité** des informations

On parle souvent du trièdе CIA auquel on associe aussi le principe d'**authentification** et la **non répudiation** des données



LES 5 PRINCIPES FONDAMENTAUX

Confidentialité

But : Empêcher l'accès non autorisé à l'information

📌 Cela signifie que seules les personnes ou systèmes autorisés peuvent lire ou accéder à une donnée.

Exemples :
- Données chiffrées avec une clé privée
- Authentification par mot de passe
- Gestion des permissions (ACL, rôles)

Intégrité

But : Garantir que les données ne sont pas modifiées ou corrompues de manière non autorisée.

📌 Cela assure que l'information est fiable et complète.

Exemples :
- Hachage (ex: SHA-256)
- Signature numérique
- Contrôle d'intégrité des fichiers

Disponibilité

But : S'assurer que les données et services sont accessibles quand on en a besoin.

📌 Même en cas de panne, de surcharge ou d'attaque (DDoS), le service doit rester fonctionnel.

Exemples :
- Redondance des serveurs
- Systèmes de sauvegarde
- Répartition de charge (load balancing)

Non-répudiation

But : Empêcher qu'une personne nie avoir réalisé une action.

📌 Cela signifie qu'on peut prouver qu'un utilisateur a effectué une action précise (signature, envoi de message, etc.).

Exemples :
- Signature électronique
- Journalisation (logs) horodatée
- Accusé de réception

Authentification

But : Vérifier l'identité d'un utilisateur, service ou système.

📌 Savoir qui fait une action.

Exemples :
- Login/mot de passe
- Authentification à double facteur (2FA)
- Certificat numérique

OWASP

L'**Open Web Application Security Project**, ou **OWASP**, est une organisation **internationale** à but non lucratif qui se consacre à la **sécurité** des **applications** web.

L'un des principes fondamentaux de l'OWASP est que tout son contenu soit disponibles gratuitement et facilement accessibles sur son site web, ce qui permet à chacun d'améliorer la sécurité de ses propres applications web.

Le contenu qu'elle propose comprend de la documentation, des outils, des vidéos et des forums.

Leur projet le plus connu est le **Top 10 de l'OWASP**.

SECURITE DES DONNEES DANS UNE APPLI WEB

Les principales menaces :

Menace	Description	Exemple concret
🔍 Injection (SQL, NoSQL)	Entrée mal filtrée qui modifie une requête	' OR 1=1 --
🔐 Vol de mot de passe	Si mot de passe stocké sans hashage	Une base de données piratée
🕸️ XSS (Cross-Site Scripting)	Exécution de scripts malveillants dans le naviga	<script>alert('XSS')</script>
🔄 CSRF (Cross-Site Request Forgery)	Requête frauduleuse depuis un autre site	Formulaire caché qui envoie une action
👤 Usurpation d'identité	Piratage d’un token ou session	Cookie volé

LE RGPD

Le RGPD (Règlement Général sur la Protection des Données) est une loi européenne entrée en vigueur en mai 2018.

Son objectif : **Protéger** les **données personnelles** des citoyens européens.

DONNES PERSONNELLES

Qu’est ce qu’on qualifie de donnée personnelle ?

Toute information permettant d’**identifier directement** ou **indirectement** une **personne** physique.

Exemples :

- Nom, prénom
- Email, téléphone
- Adresse IP
- Identifiant utilisateur
- Cookies de session

OBLIGATIONS DU DEVELOPPEUR

🔑 Authentification

- Stocker les mots de passe hashés
- Avoir une politique de mot de passe solide

🔒 Stockage

- Chiffrer les données sensibles
- Sécuriser les bases de données avec accès restreint

🚫 Cookies & tracking

- Ne pas utiliser de cookies de suivi sans consentement
- Proposer un bandeau cookie clair

📦 Fichiers & logs

- Anonymiser ou pseudonymiser les logs utilisateurs
- Ne pas garder indéfiniment les journaux sensibles

CE QU'IL FAUT METTRE EN PLACE

01 Politique de confidentialité lisible

02 Page de gestion des données personnelles

03 Notification des failles de sécurité à la CNIL (sous 72h)

04 Accès limité aux données côté développeurs/admin

05 Sécurité serveur (mise à jour, firewall, backups)

CHIFFRER VS CRYPTER : UNE DIFFÉRENCE QUI PÈSE LOURD

Chiffrer : la science de l'inviolabilité
Chiffrer données, c'est les rendre illisibles aux intrus grâce à des algorithmes affûtés et des clés secrètes. Un rempart technique contre les cyberattaques, validé par des experts comme l'ANSSI. Chaque octet est verrouillé, chaque accès contrôlé. C'est la méthode des pros, et rien d'autre.

Crypter : un faux pas linguistique
"Crypter" vous semble familier ? C'est un piège. Emprunté à l'anglais "encrypt", ce terme est un intrus dans le français technique. Les puristes le bannissent, les experts le méprisent. Dire "je crypte mes données" sonne amateur là où "je chiffre mes données" impose le respect.

ETUDE DE CAS

Fuite de données chez NexaCorp

FUITE DE DONNEES CHEZ NEXACORP

Contexte :

NexaCorp est une entreprise de e-commerce spécialisée dans la vente de matériel médical. Elle héberge des données sensibles de ses clients :

- Nom, prénom
- Adresse
- Données bancaires
- Dossiers médicaux (ordonnances, diagnostics, etc.)

Scénario

📅 Lundi matin, 8h15.

Une alerte déclenchée par le système de surveillance détecte un pic anormal de trafic sortant depuis la base de données client, à destination d'une **IP étrangère**.

🛠️ L'équipe technique découvre que :

- Un compte **administrateur** a été **compromis**.
- Des fichiers contenant des données **clients non chiffrées** ont été exfiltrés.
- Des données semblent avoir été **modifiées** pendant la fuite.
- Les journaux d'accès sont **incomplets**.
- Le serveur est mis **hors-ligne** pour éviter que la fuite ne continue.

⚖️ La Direction est alertée. Le Responsable RGPD informe qu'une notification à la CNIL est obligatoire sous 72h, et que les clients concernés doivent être informés.

QUESTIONS À TRAITER

1. Quels principes fondamentaux de la cybersécurité ont été compromis ?

Cochez ceux qui s'appliquent :

- Confidentialité
- Intégrité
- Disponibilité
- Non-répudiation
- Traçabilité
- Conformité RGPD

2. Quelles erreurs ou négligences ont contribué à cette fuite de données ?

3. Quelles mesures correctives devraient être prises pour chaque principe compromis ?

Principe	Violation observée	Correctif à mettre en place
Confidentialité		
Intégrité		
Disponibilité		
Non-répudiation		
RGPD		

4. Que prévoit le RGPD en cas de violation de données personnelles ?

5. 💡 Bonus : Proposez un plan d'action pour renforcer la sécurité globale de NexaCorp.

L’AUTHENTI-FICATION

C’EST QUOI L’AUTHENTIFICATION ?

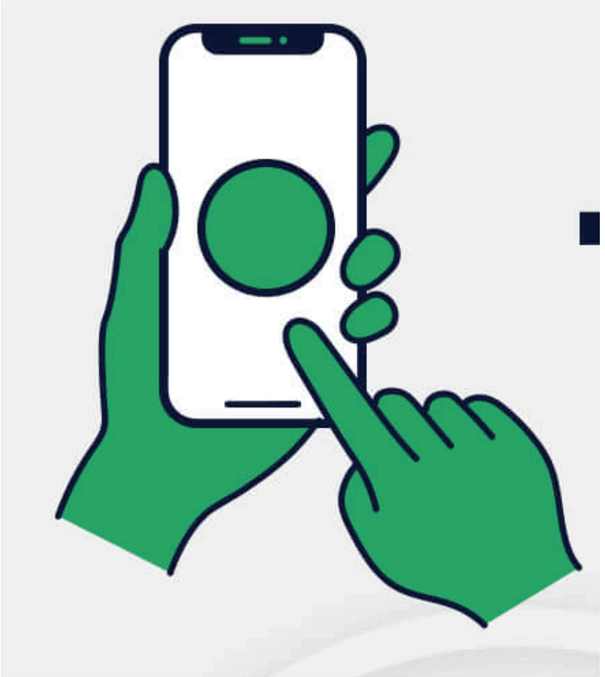
Définition

Processus de vérification de l'identité d'un utilisateur.

👉 Elle répond à la question : "Qui êtes-vous ?"

Identification vs Authentification vs Autorisation

- **Identification** dire qui je suis (e.g., login, empreinte)
- **Authentification** vérifier/prouver qui je suis (e.g., password)
- **Autorisation** vérifier si j’ai le droit (contrôle d’accès)



LES MÉTHODES D’AUTHENTIFICATION

Login/Password

- L’utilisateur fournit un identifiant + mot de passe
- Le serveur compare avec ce qui est stocké (hashé !)

Token (JWT)

- Après connexion, le serveur génère un token signé
- Ce token est envoyé par le client dans chaque requête suivante (dans les headers)

OAuth2 (Google, GitHub, etc.)

- Authentification déléguée à un fournisseur tiers
- L’utilisateur se connecte avec un compte Google par exemple

LES FACTEURS D'AUTHENTIFICATION

Les facteurs d'authentification peuvent être regroupés en 3 catégories principales

- Ce que l'utilisateur **sait** (par exemple, un mot de passe)
- Ce que l'utilisateur **possède** (par exemple, une clé de sécurité)
- Ce que l'utilisateur **est** (par exemple, une empreinte digitale)

L'utilisation combinée de plusieurs de ces facteurs constitue une authentification **multi-facteurs**, renforçant significativement la sécurité d'un système.

LA CRYPTOGRAPHIE

Terme	Définition
Cryptographie	Science de la protection de l'information par des techniques mathématiques.
Chiffrement	Opération qui transforme un message lisible (**texte clair**) en un message incompréhensible (**texte chiffré**).
Déchiffrement	Opération inverse qui permet de retrouver le message initial.

Hachage :

Transforme une donnée en une empreinte unique et fixe.
Utilisé pour vérifier l'intégrité ou stocker des mots de passe.
⚠️ Un hachage n'est pas réversible (≠ chiffrement).
Exemples : SHA-256, SHA-512, bcrypt, Argon2

Salt :

Transforme une donnée en une empreinte unique et fixe.
Utilisé pour vérifier l'intégrité ou stocker des mots de passe.
⚠️ Un hachage n'est pas réversible (≠ chiffrement).
Exemples : SHA-256, SHA-512, bcrypt, Argon2

POURQUOI LE SALT EST IMPORTANT ?

Si deux personnes utilisent le même mot de passe, elles auront la même valeur de hash.
Cela donne une indication à un potentiel hackeur.

Username	String to be hashed	Hashed value = SHA256
user1	**password123**	EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F
user2	**password123**	EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F

On introduit alors un SEL qui sera différent pour chaque mot de passe et haché par la suite.
Ainsi on s'assure que chaque Hash final soit bien identique.

Username	Salt value	String to be hashed	Hashed value = SHA256 (Password + Salt value)
user1	D;%yL9TS:5PalS/d	**password123**D;%yL9TS:5PalS/d	9C9B913EB1B6254F4737CE947EFD16F16E916F9D6EE5C1102A2002E48D4C88BD
user2	)<,-<U(jLezy4j>*	**password123**)<,-<U(jLezy4j>*	6058B4EB46BD6487298B59440EC8E70EAE482239FF2B4E7CA69950DFBD5532F2

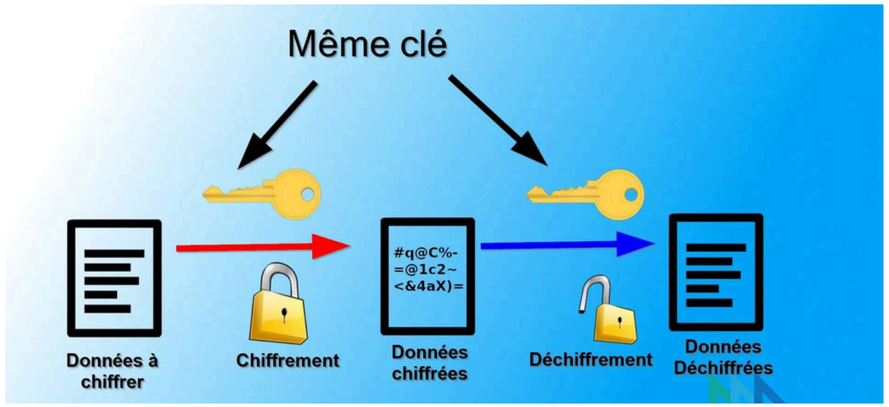
LES TYPES DE CHIFFREMENTS

Chiffrement symétrique

- Utilise une seule clé pour chiffrer et déchiffrer.
- Rapide, mais la distribution de la clé est un problème.

Exemples :

- AES (Advanced Encryption Standard)
- DES (obsolète aujourd'hui)

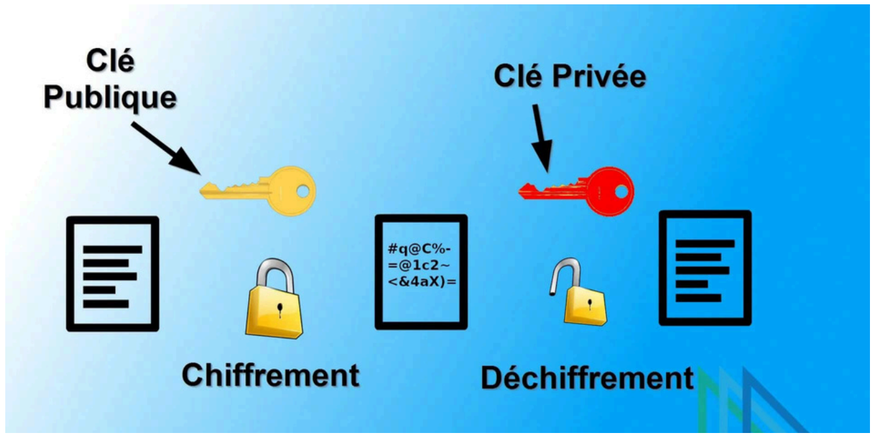


Chiffrement asymétrique

- Utilise deux clés : une publique pour chiffrer, une privée pour déchiffrer.
- Plus lent, mais plus sûr pour échanger des données.

Exemples :

- RSA
- ECC (Elliptic Curve Cryptography)



LES JWT

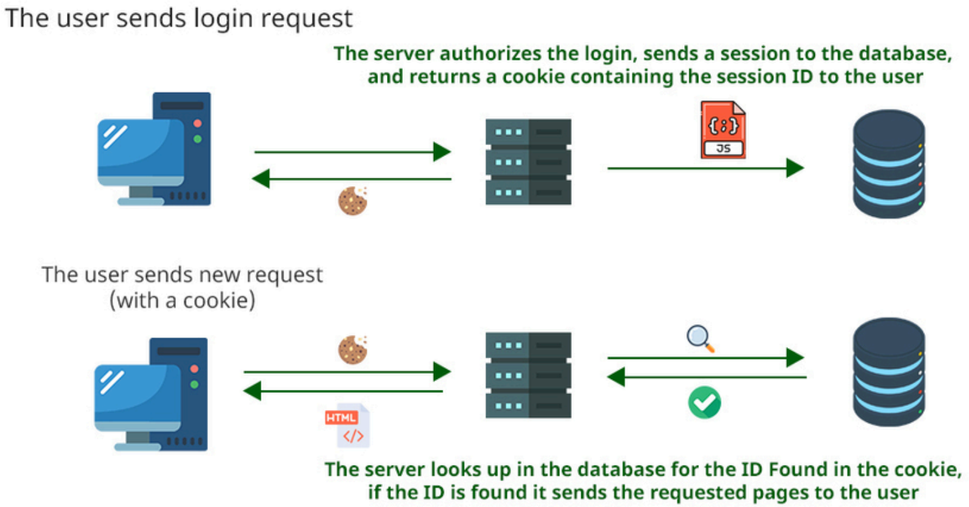
AUTHENTIFICATION PAR SESSION

Authentification par session (basée serveur)

Le serveur crée une session et la stocke (souvent via cookie)
L'utilisateur reçoit un identifiant de session
Chaque requête envoie ce cookie pour s'identifier

Inconvénients :

- Stockage côté serveur (mémoire, base de données)
- Moins scalable en microservices
- Besoin de synchroniser les sessions si plusieurs serveurs



AUTHENTIFICATION PAR JETON

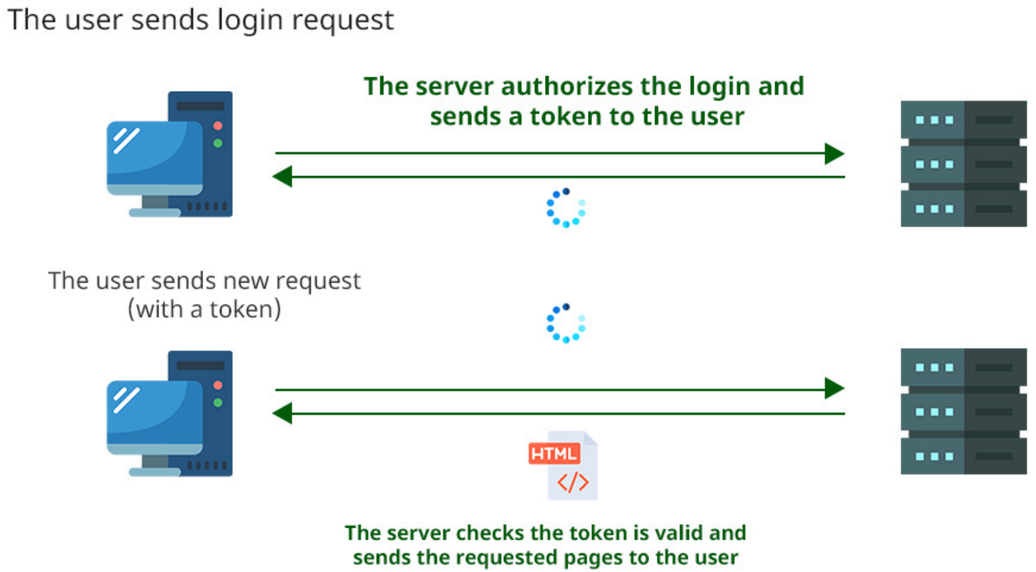
Authentification par token (stateless)

Le serveur génère un jeton (token) (ex : JWT)
L'utilisateur garde ce jeton et l'envoie à chaque requête

Le serveur ne stocke rien : il vérifie le token à chaque fois

Avantages :

- Plus scalable
- Fonctionne bien avec API REST
- Stateless (pas de session stockée côté serveur)



ETUDE DE CAS

Lire un JWT

LECTURE D’UN JWT

1. Ouvrez https://jwt.io

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiZW1haWwiOiJhbGl
jZUBleGFtcGxlLmNvbSIsInJvbGUiOiJhZG1pbiJ9.9FU_-
G82LdWqmwQfiGbn8ePthT7OBdi4V7B8Vzkg9B8

2. Lisez et analyser le jeton.

3. Que se passe-t-il si on change une valeur dans le payload ?

LES BONNES PRATIQUES DE JETON

- Ne jamais stocker d’informations sensibles dans le payload (pas de mot de passe !)
- Mettre une date d’expiration (exp)
- Transmettre via Authorization: Bearer <token>
- Toujours utiliser HTTPS

LES ALGORITHMES DE HACHAGE

QU'EST CE QU'UN ALGORITHME DE HACHAGE ?

Un algorithme de hachage est une fonction qui transforme n'importe quelle donnée (texte, fichier, etc.) en une empreinte numérique (appelée hash ou digest) de taille fixe.

Exemple :

```
SHA256("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e...
```

Propriétés principales :

- **Rapidité** : Calculé rapidement.
- **Taille fixe** : La sortie a toujours la même taille (ex : 256 bits).
- **Déterminisme** : Même entrée → même sortie.
- **Non réversible** : On ne peut pas retrouver l'entrée à partir du hash.
- **Résistance aux collisions** : Impossible (ou très difficile) de trouver deux entrées ayant le même hash.

LES DIFFERENTS ALGORITHMES DE HACHAGE

Nom	Longueur	Usage principal	Sécurité actuelle
MD5	128 bits	Vérification simple (checksum)	❌ A éviter (collisions faciles)
SHA-1	160 bits	Ancien standard, Git, certificats anciens	❌ A éviter (collisions démontrées)
SHA-256	256 bits	Standard moderne (blockchain, API)	✅ Solide
SHA-512	512 bits	Sécurité renforcée	✅ Solide
bcrypt	Variable	Hashage sécurisé des mots de passe	✅ Très robuste
argon2	Variable	Hashage ultra-sécurisé des mots de passe	✅ Recommandé (plus récent que bcrypt)

CAS D’USAGES

Cas d’usage	Algorithmes recommandés
Intégrité de fichiers	MD5, SHA-1, SHA-256 (checksum rapide)
Signature numérique	SHA-256, SHA-512
Mots de passe	bcrypt, argon2, scrypt

Les mots de passe nécessitent un hachage lent pour éviter les attaques par force brute.

❌ Stocker un mot de passe avec SHA256 → Beaucoup trop rapide, vulnérable aux attaques
✅ Stocker un mot de passe avec bcrypt ou argon2 → Lent volontairement + salage automatique

BCRYPT

POURQUOI BCRYT ?

bcrypt est l'un des algorithmes les plus utilisés car :

- Il est lent volontairement → rend les attaques bruteforce très coûteuses.
- Il génère un hash unique avec un sel automatique → même mot de passe, hash différent

Installation de bcrypt sous Node :

```
npm install bcrypt
```

1. HASHER UN MOT DE PASSE

Utilisation de bcrypt.hash() :

```
1 const bcrypt = require('bcrypt');
2
3 const plainPassword = 'motdepasse123';
4 const saltRounds = 10; // plus le chiffre est élevé, plus c'est sécurisé (et
  lent)
5
6 const hashedPassword = await bcrypt.hash(plainPassword, saltRounds);
7 console.log(hashedPassword); // -> une longue chaîne illisible
8
```

2. VERIFICATION MOT DE PASSE

Pour vérifier si un mot de passe correspond au hash stocké on utilise la méthode compare()

compare() :

- Prend le mot de passe saisi et le hash
- Applique le même hashage interne et compare le résultat

```
const isMatch = await bcrypt.compare(plainPassword, hashedPassword);

if (isMatch) {
    console.log('Mot de passe valide');
} else {
    console.log('Mot de passe invalide');
}

```

2. VERIFICATION MOT DE PASSE

Pour vérifier si un mot de passe correspond au hash stocké on utilise la méthode compare()

compare() :

- Prend le mot de passe saisi et le hash
- Applique le même hashage interne et compare le résultat

```
const isMatch = await bcrypt.compare(plainPassword, hashedPassword);

if (isMatch) {
    console.log('Mot de passe valide');
} else {
    console.log('Mot de passe invalide');
}

```

RESUME RAPIDE

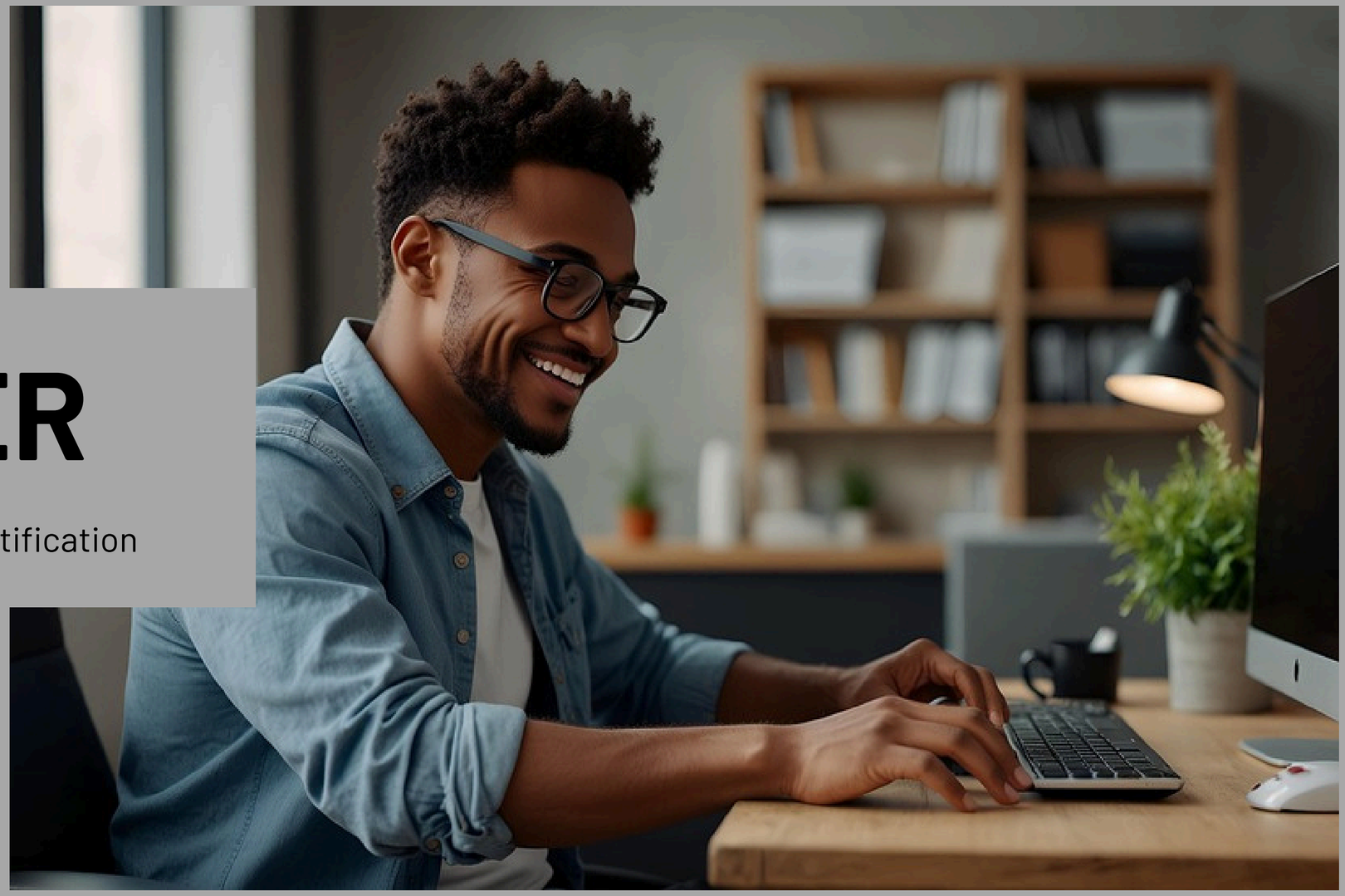
Action	Méthode bcrypt	Description
Hasher un mot de passe	bcrypt.hash()	Chiffre un mot de passe avec sel
Vérifier un mot de passe	bcrypt.compare()	Compare un mot de passe avec son hash
Salage automatique	✅ Intégré dans hash()	Pas besoin de gérer manuellement

EXEMPLE COMPLET

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

const app = express();
app.use(express.json());

let fakeDB = {}; // stockage simulé

app.post('/register', async (req, res) => {
    const { username, password } = req.body;
    const hash = await bcrypt.hash(password, 10);
    fakeDB[username] = hash;
    res.json({ message: 'Utilisateur créé' });
});

app.post('/login', async (req, res) => {
    const { username, password } = req.body;
    const hash = fakeDB[username];
    if (!hash) return res.status(404).json({ error: 'Utilisateur inconnu' });

    const valid = await bcrypt.compare(password, hash);
    if (!valid) return res.status(401).json({ error: 'Mot de passe invalide'
});

    res.json({ message: 'Connexion réussie' });
});

app.listen(3000, () => console.log('Serveur démarré sur http://
localhost:3000'));

```

ATELIER

Réaliser un système d’authentification



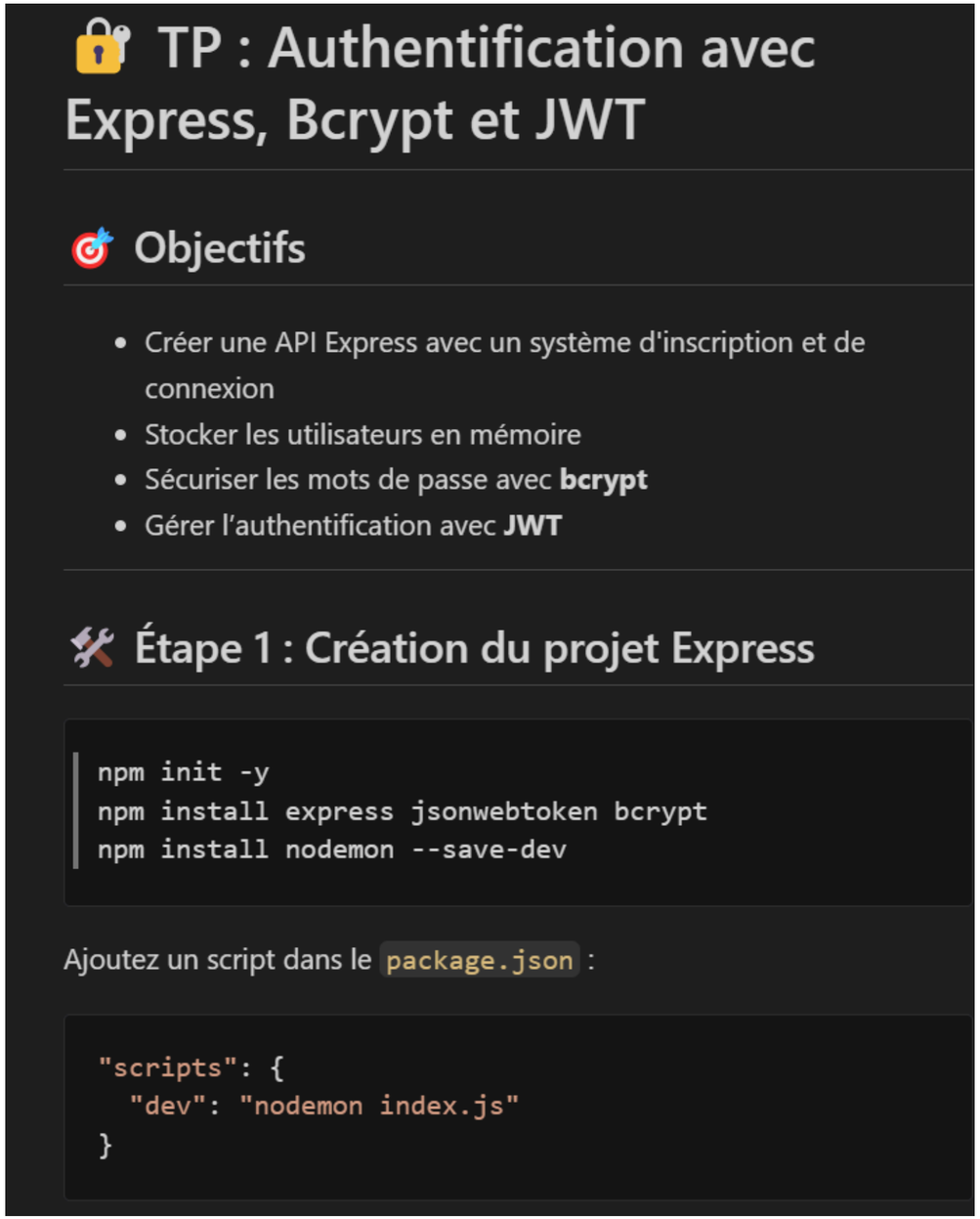
SUIVEZ LE GUIDE

https://github.com/dessna12/04-JWT

Nous allons mettre en place un système d'authentification à l'aide du package bcrypt() et renvoyer un token avec le package jsonwebtoken();

Nous allons créer un serveur Express qui retournera deux méthodes HTTP:

- POST / register pour enregistrer un nouvel utilisateur
- POST / login pour s'authentifier et retourner un JWT si l'utilisateur a le bon mot de passe.



🔐 TP : Authentification avec Express, Bcrypt et JWT

🎯 Objectifs

- Créer une API Express avec un système d'inscription et de connexion
- Stocker les utilisateurs en mémoire
- Sécuriser les mots de passe avec **bcrypt**
- Gérer l'authentification avec **JWT**

🛠️ Étape 1 : Création du projet Express

```
npm init -y
npm install express jsonwebtoken bcrypt
npm install nodemon --save-dev
```

Ajoutez un script dans le `package.json` :

```
"scripts": {
  "dev": "nodemon index.js"
}
```

QUIZZ