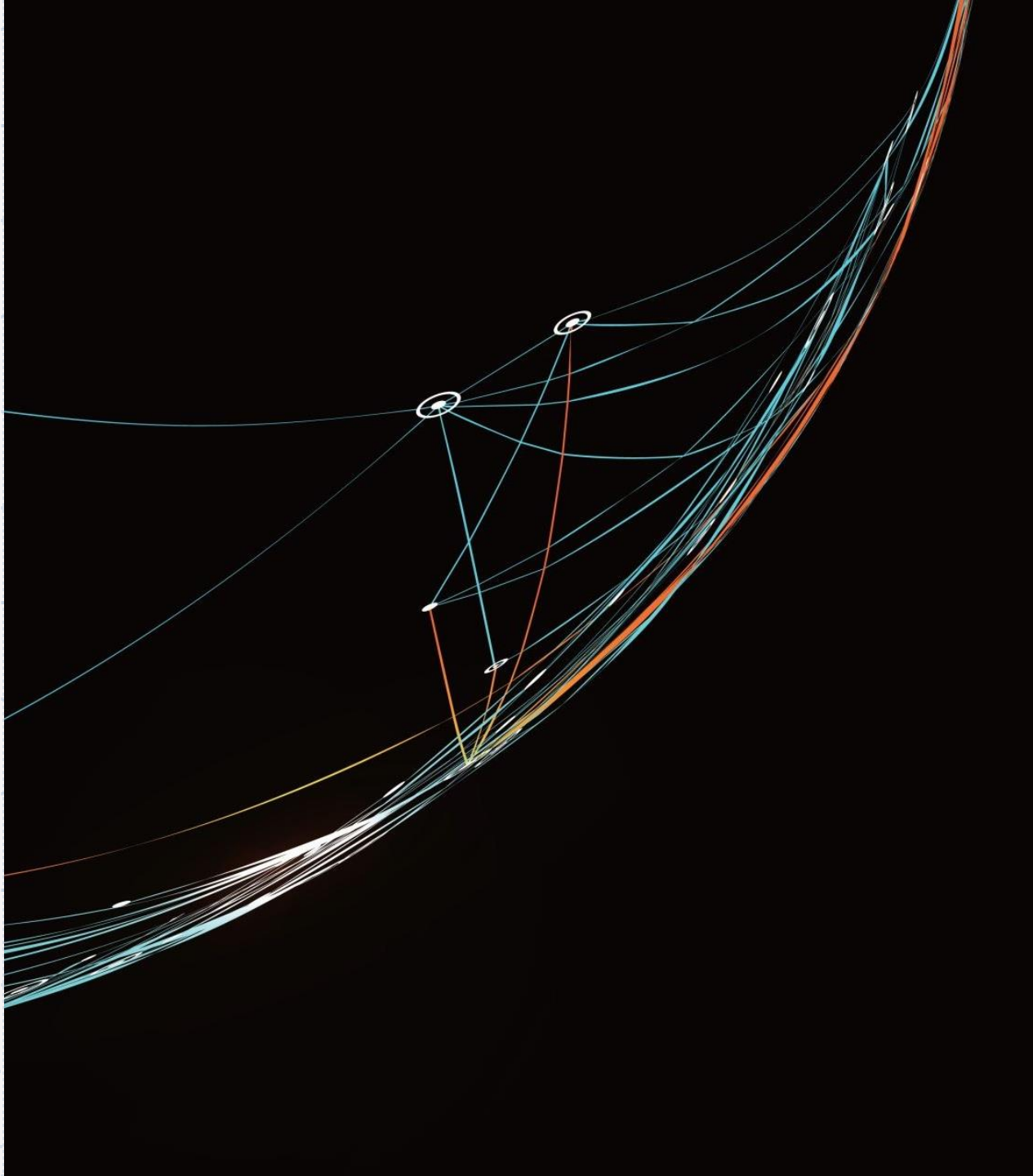
REST API

Bonnes pratiques et sécurité

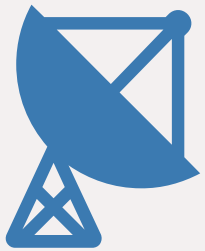


Introduction : Évolution des applications modernes et l'importance des API

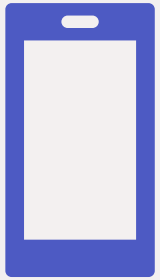
Objectifs du jour

- Comprendre l’évolution des applications modernes.
- Découvrir le rôle central des API.
- Bases de REST vs SOAP.
- Un peu de pratique !

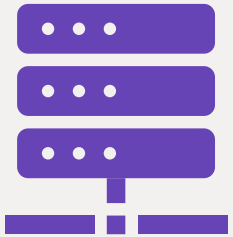
Les API dans les applications modernes



Définition des API : Interfaces permettant aux logiciels de communiquer.



Évolution : Des systèmes isolés aux applications interconnectées.



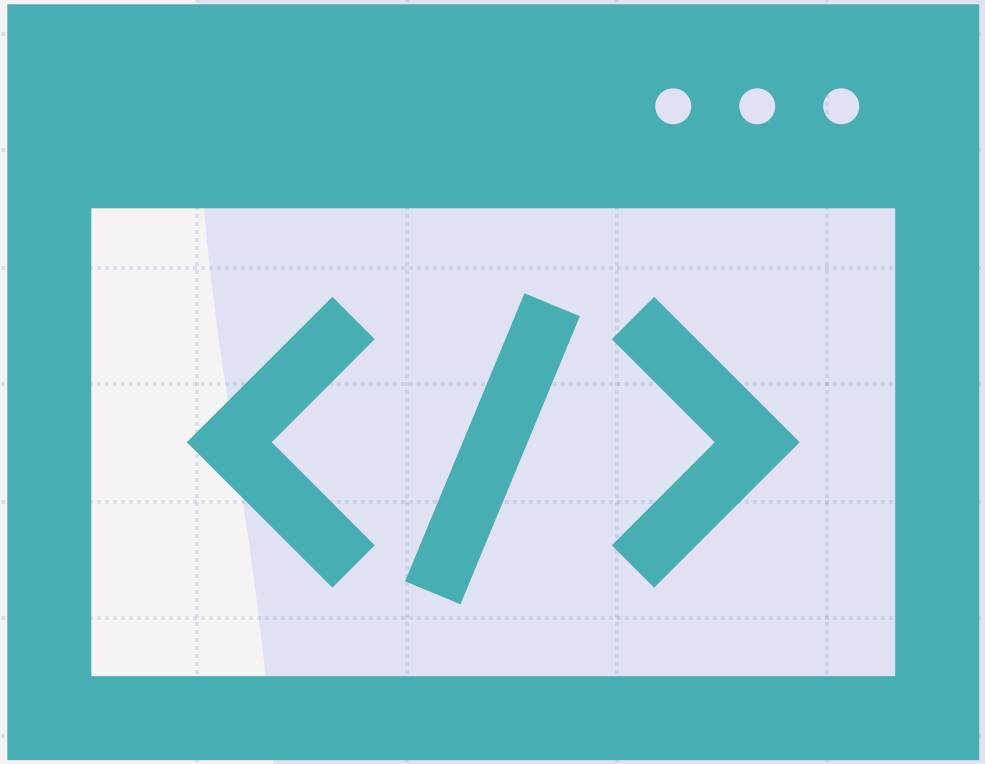
Rôle des API : Connecter des services, partager des données, favoriser l'interopérabilité.

Exemples d'API dans la vie quotidienne

- Exemples d'applications utilisant des API :
  - Applications météo (API météo).
  - Réseaux sociaux (API de partage).
  - E-commerce (API de paiement).
  - Cartographie (API Google Maps).

REST vs SOAP : Introduction aux architectures d'API

- **REST** (Representational State Transfer) :
- Conception sans état (stateless).
- Utilisation de JSON, structure légère.
- **SOAP** (Simple Object Access Protocol) :
- Basé sur des protocoles stricts (XML).
- Protocoles orientés sécurité, état possible.



REST vs SOAP : Avantages et inconvénients

REST

- Avantages : Flexibilité, simplicité, large adoption (services web modernes).
- Inconvénients : Moins rigide sur la sécurité.

SOAP

- Avantages : Sécurité, transactions complexes.
- Inconvénients : Plus lourd, nécessite XML.

Cas d’usage de REST vs SOAP

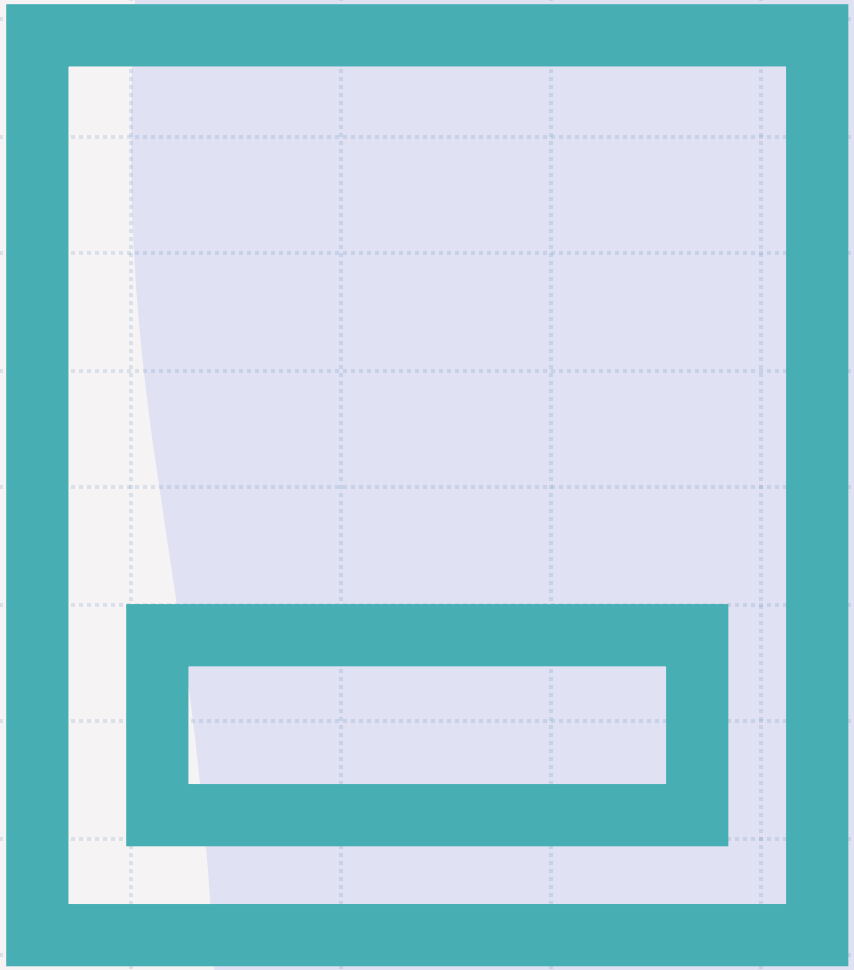
REST : IDÉAL POUR LES SERVICES WEB SIMPLES, FLEXIBLES ET RAPIDES.

SOAP : RECOMMANDÉ POUR LES APPLICATIONS NÉCESSITANT DES TRANSACTIONS SÛRES ET DES RÈGLES STRICTES.

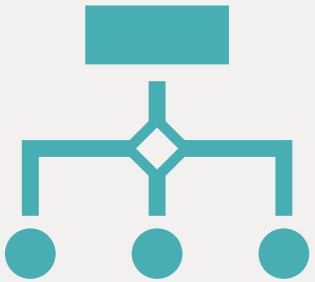
Requêtes HTTP vers une API : Méthodes courantes

- **GET** : Récupération de données.
- **POST** : Création de nouvelles données.
- **PUT** : Mise à jour de données existantes.
- **DELETE** : Suppression de données.

Et… Beaucoup d’autres.



Exercice pratique : Faire une requête GET vers une API publique



Objectif : Envoyer une requête GET vers OpenWeather API pour récupérer la météo.

Exemple de commande : curl -X GET "https://api.openweathermap.org/data/...".

Exemple de vulnérabilité : Exposition excessive des données

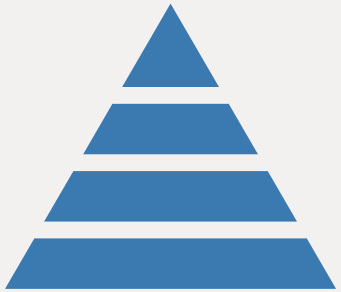
Problème : Une API expose des informations sensibles.

Solution : Limiter les informations renvoyées, filtrer les données sensibles.

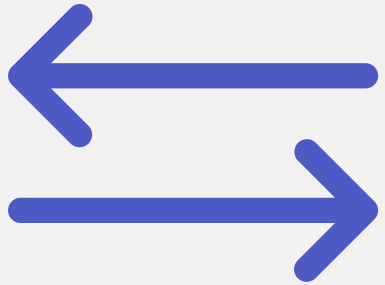
Conventions de nommage et structuration des routes

- Utiliser des noms clairs et explicites.
- Exemple : /api/users pour accéder aux utilisateurs.

Versionnement des API



Importance du versionnement pour la compatibilité (ex: /v1/api/users).



Pratiques : Numérotation par version, éviter les modifications de rupture.

Formats de réponse : JSON vs XML

JSON : Format léger, lisible, compatible avec REST.

XML : Utilisé par SOAP, plus formel et sécurisé.

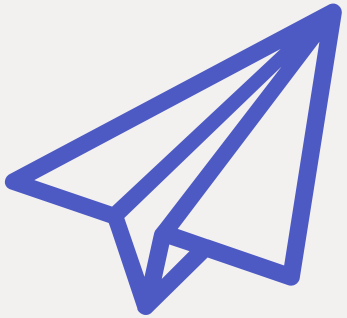
Bonnes pratiques : Gestion des erreurs et codes d'état HTTP

Codes d’état HTTP :		
200 : Succès.	400 : Mauvaise requête.	500 : Erreur serveur.

Importance de gérer les erreurs côté client et serveur.

Validation des données dans les API

Validation : Empêche les données incorrectes de passer.



Exemples : Exigence de format pour les e-mails, vérification de la longueur des mots de passe.

Documentation des API

Documentation : Indispensable pour l'utilisation et la maintenance.

Inclure les routes, paramètres, exemples de réponse.

Exemple de réponse JSON standardisée

Format type : { "status": "success", "data": { ... } }.

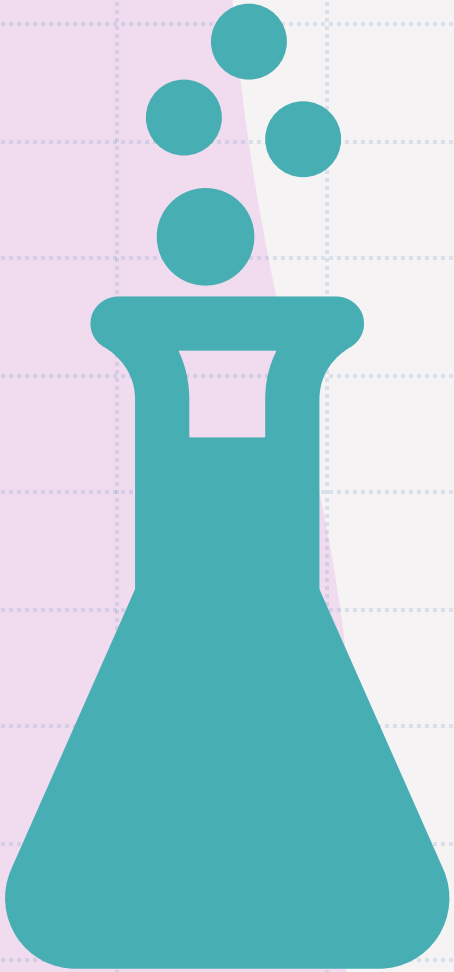
Modèles d'architecture : MVC et Repository Pattern

MVC : Structure en trois couches : Modèle, Vue, Contrôleur.

Repository Pattern : Gestion des données entre API et base de données.

Exemple

- Exemple : API de gestion des utilisateurs avec routes séparées pour chaque fonctionnalité.



Présentation des outils de développement d'API

Introduction aux outils utiles pour documenter, tester et simuler des API.

OpenAPI et Swagger : Documentation et spécification d'API

OpenAPI : Langage standard pour la documentation d'API.

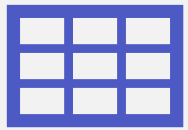
Swagger : Génère automatiquement la documentation à partir du code.

Postman : Test des requêtes et des réponses

Tester les endpoints, valider la conformité avec la documentation.

JSON Server et JSON Generator : Simuler des API

JSON Server : Crée des endpoints RESTful à partir de données JSON.



JSON Generator : Génère rapidement des données JSON.

Exercice pratique : Utilisation de Postman ou Insomnia pour tester une API

Objectif : Tester une requête GET pour vérifier la documentation et le format de réponse.

Exemple de requête avec Postman

- Étapes :
  - Ouvrir Postman et créer une nouvelle requête.
  - Sélectionner la méthode (GET, POST, etc.).
  - Entrer l'URL de l'API.
  - Envoyer la requête et observer la réponse.

Analyse des réponses dans Postman

Comment interpréter les codes d'état et le corps de réponse.

Importance de valider les données reçues.

Exemples de réponses valides et invalides.

Les menaces principales pour la sécurité des API

Attaques courantes :

- Injections (SQL, commande).
- Cross-Site Scripting (XSS).
- Cross-Site Request Forgery (CSRF).

CIDP:

Confidentialité

Intégrité

Disponibilité

Preuve

CIDP: Concepts clés de la sécurité

1

Confidentialité : Protéger les informations sensibles contre les accès non autorisés.

2

Intégrité : Garantir que les données ne sont ni modifiées ni détruites de manière non autorisée.

3

Disponibilité : Assurer que les services restent accessibles aux utilisateurs légitimes.

4

Preuve : Assurer le suivi, ou tracabilité de chaques actions, via des logs par exemple.

Top 10 OWASP API Security : Introduction

OWASP API Security Top 10 : Liste des vulnérabilités les plus courantes dans les API.

Importance de l'OWASP pour identifier les risques prioritaires et adopter des mesures de sécurité.

A ne pas confondre avec le Top 10 OWASP "Classique" plus adapté aux applications web.

Top 10 OWASP : API1 – Broken Object Level Authorization (BOLA)

Description : Accès non autorisé à des objets en raison de contrôles d'autorisation incorrects.

Exemple : Utilisateur accédant aux données d'un autre utilisateur sans autorisation.

Solution : Vérification des autorisations pour chaque objet (middleware).

Top 10 OWASP : API2 - Broken User Authentication

Description : Failles dans le processus d'authentification.

Exemple : Accès non autorisé dû à une authentification incomplète.

Solution : Implémenter une authentification sécurisée (ex. multifactorielle).

Autres risques OWASP API : Vue d'ensemble

API3 : Excessive Data Exposure

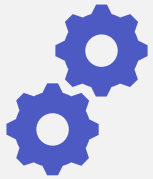
API4 : Lack of Resources & Rate Limiting

API5 : Broken Function Level Authorization

Chaque vulnérabilité peut exposer les données ou affecter les performances.

Exemple de vulnérabilité : Injection SQL dans une API

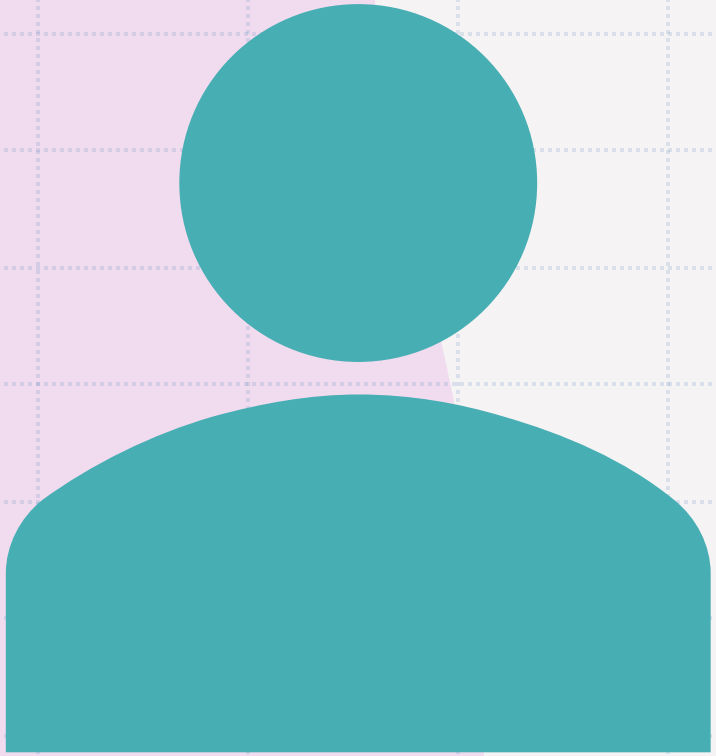
Problème : Injection SQL possible si les entrées utilisateur ne sont pas correctement validées.



Solution : Utilisation de requêtes préparées ou ORM pour éviter les injections SQL.

Authentification et gestion sécurisée des mots de passe

- Problèmes courants :
  - Stockage de mots de passe en clair.
  - Faiblesse des mots de passe.
- **Solutions** :
  - Hashing sécurisé (bcrypt, Argon2).
  - Limiter les tentatives de connexion.



Gestion des identifiants et autorisations d'objets

- **Rôles et permissions** : Limiter l’accès aux ressources en fonction du rôle utilisateur.
- **Exemple** : Un utilisateur ne doit pas pouvoir accéder aux informations d’un administrateur.

Exploitation des "business flows" : Identifier les failles dans la logique métier

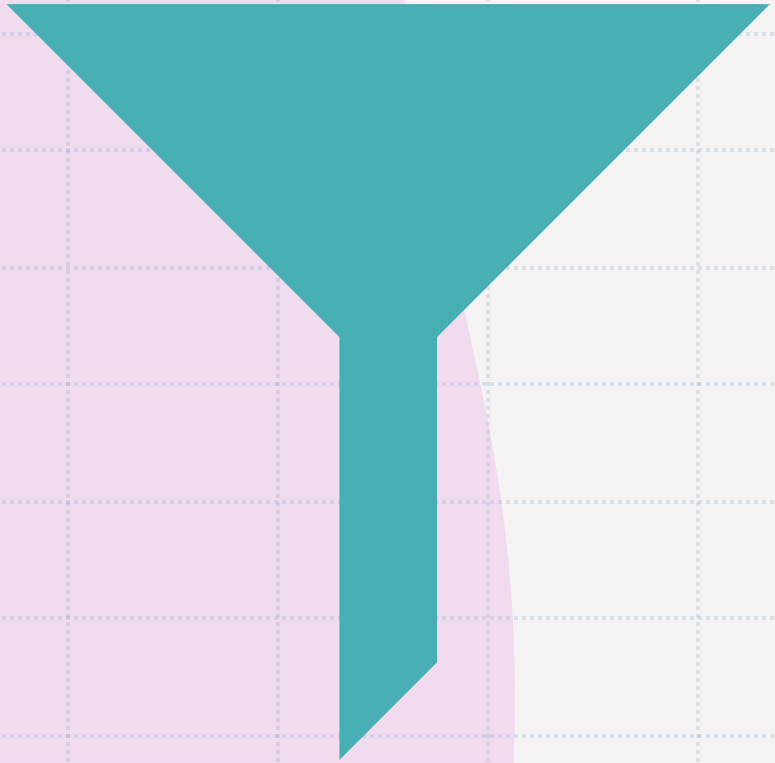
- **Exemple de problème** : Événements ou actions dans le mauvais ordre (ex : valider une transaction avant paiement).
- **Solution** : Valider la logique métier pour chaque endpoint.

Exemple de vulnérabilité : Faille d’authentification (Brute Force)

Problème : Tentatives répétées d'authentification pour deviner des identifiants.

Solution :

Implémentation de la limitation de taux (Rate Limiting).	Utilisation de CAPTCHAs.



Server-Side Request Forgery (SSRF) : Qu'est-ce que SSRF ?

- **Définition** : SSRF permet à un attaquant d'exploiter le serveur pour envoyer des requêtes non autorisées.
- **Exemple** : Un utilisateur exploite une API pour accéder aux données internes.

Protection contre les attaques SSRF

- **Solutions** :
  - Limiter les destinations autorisées par le serveur.
  - Mettre en place des filtres pour bloquer les requêtes internes.

Mauvaise configuration de sécurité : Exemples et risques

- **Exemple** :
  - Configurations par défaut exposant des informations sensibles.
  - Ports ouverts non sécurisés.
- **Solution** :
  - Désactiver les options non utilisées (ex. header X-Powered-By).

Exemple de vulnérabilité : Headers d’information serveur

- **Problème** : Exposer des informations de configuration peut être utilisé par les attaquants.
- **Solution** :
  - Masquer ou supprimer les headers comme X-Powered-By et Server.

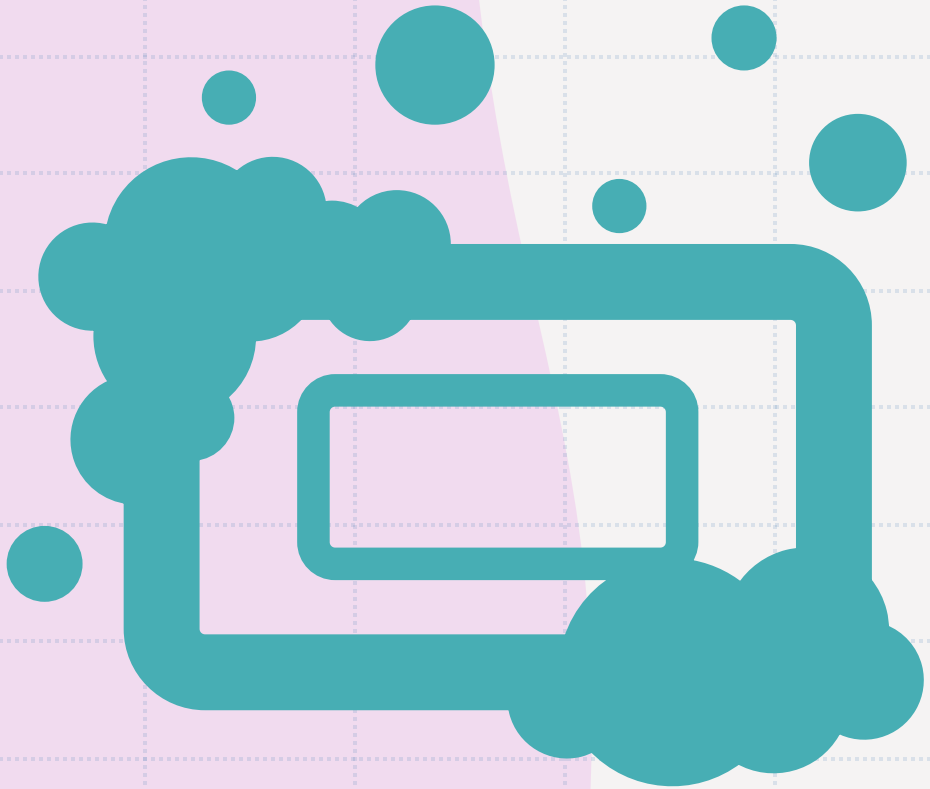
Introduction à l'exercice d’audit de sécurité

- **Objectif** : Appliquer les connaissances pour identifier les vulnérabilités sur une API non sécurisée.
- **Étapes de l’audit** : Tester les endpoints pour identifier les failles courantes.

Exemple de vulnérabilité : Absence de chiffrement (HTTP au lieu de HTTPS)

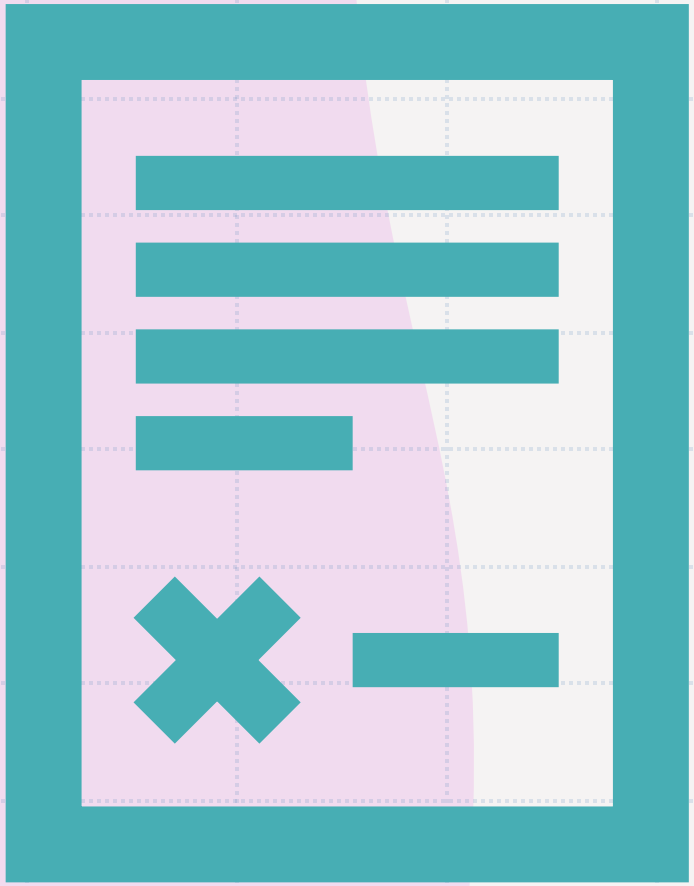
Problème : Transmission non sécurisée des données, rendant les informations sensibles vulnérables.

Solution : Utilisation de HTTPS pour chiffrer les communications.



Analyse des endpoints pour vérifier l'authentification et les permissions

- Vérifier les autorisations pour chaque action (lecture, écriture, modification).
- **Exemple** : Utilisateur non autorisé pouvant modifier des données sensibles.



Documentation des vulnérabilités trouvées et correctifs proposés

- **Rapport** : Identifier chaque vulnérabilité et proposer un correctif détaillé.
- Exemples de correctifs : Limitation des données exposées, validation des entrées, configurations de sécurité.

Différence entre Authentification et Autorisation

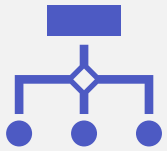
Authentification : Vérifie l'identité de l'utilisateur (ex. login).

Autorisation : Définit les permissions pour accéder à des ressources spécifiques.

Exemple : Authentifier un utilisateur admin et vérifier son autorisation pour accéder à des données sensibles.

Concepts clés d'authentification moderne : SSO/OAuth2 et OpenID Connect

OAuth2 : Protocole de délégation d'autorisation.



OpenID Connect (OIDC) : Extension OAuth2 pour l'authentification, fournissant des informations sur l'identité de l'utilisateur.

Les principaux flux OAuth2

Authorization Code Flow : Utilisé pour les applications serveur.

Implicit Flow : Utilisé pour les applications front-end.

Client Credentials Flow : Utilisé pour l'authentification serveur à serveur.

Mise en place de l'Authorization Code Flow avec OAuth2

- Explication des étapes :
  1. L'utilisateur est redirigé vers une page d'autorisation.
  2. L'utilisateur accorde les permissions et reçoit un code d'autorisation.
  3. L'application échange ce code contre un jeton d'accès.

Oauth: Démonstration

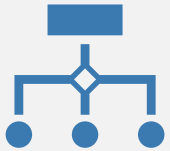
```
const passport = require('passport');
const OAuth2Strategy = require('passport-oauth2');

// Configuration de Passport avec OAuth2
passport.use(new OAuth2Strategy({
    authorizationURL: 'https://provider.com/oauth2/
authorize',
    tokenURL: 'https://provider.com/oauth2/token',
    clientID: 'CLIENT_ID',
    clientSecret: 'CLIENT_SECRET',
    callbackURL: 'https://example.com/auth/provider/
callback'
  },
  (accessToken, refreshToken, profile, cb) => {
    // Logique pour trouver ou créer l'utilisateur
    User.findOrCreate({ providerId: profile.id }, (err,
user) => {
      return cb(err, user);
    });
  }
));

// Endpoint pour déclencher l'authentification
pp.get('/auth/provider', passport.authenticate('oauth2'));

  Endpoint de callback après authentification
  .get('/auth/provider/callback',
  assport.authenticate('oauth2', { failureRedirect: '/
  n' }),
  eq, res) => {
  es.redirect('/');
```

Avantages et Limites d'OAuth2

Avantages : Permet la délégation sécurisée d'accès et protège les informations d'identification des utilisateurs.

Limites : Nécessite une configuration correcte pour éviter des failles de sécurité.

Rôles et Permissions : Pourquoi sont-ils importants ?



Rôles : Regroupements de permissions pour des utilisateurs (ex. admin, utilisateur, invité).

Permissions : Actions spécifiques que les utilisateurs peuvent effectuer.

Exemple d'API avec rôles et permissions en Node.js

- Création d’une API qui vérifie le rôle de l’utilisateur avant d’accorder l’accès.

JSON Web Token (JWT) : Présentation et Structure

JWT : Format de jeton sécurisé utilisé pour échanger des informations.

Structure : Header, Payload (claims), et Signature.

Risques et vulnérabilités des JWT

Expiration : Jetons expirés mais toujours utilisés.

Réutilisation : Jetons volés ou réutilisés après expiration.

Solution ? Rotation !

Ou ? Révocation !

Attention: Un token, qu'il soit d'accès à l'api ou JWT ne doit pas avoir une validité permanente !

Mise en place de la révocation des JWT

Liste noire : Enregistrement des jetons révoqués dans une liste pour éviter leur réutilisation.

Exemple de révocation des JWT avec une liste noire en mémoire

```
let blacklist = [];

app.post('/logout', authenticateToken, (req, res) => {
  blacklist.push(req.token);
  res.send('Déconnexion réussie');
});

function authenticateToken(req, res, next) {
  const token = req.headers['authorization'];
  if (!token || blacklist.includes(token)) return
res.status(401).send('Token révoqué');
  next();
}
```

Résumé des Problématiques liées à l’Authentification et à l’Autorisation

Risques fréquents :

- Réutilisation de jetons, mauvaises configurations de rôles, vulnérabilités d'accès, manque de suivi d'activité.

Bonnes pratiques :

- Validation des sessions, mise en place de journalisation, séparation stricte des rôles.

Risques de non-séparation des rôles

Escalade de privilèges : Des utilisateurs peuvent obtenir des permissions non autorisées sans validation stricte.

Correction : Vérifier systématiquement les rôles à chaque requête sensible.

Introduction à l'Audit de Sécurité d'une API

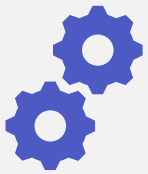
- **Objectifs de l'audit** :
  - Identifier les vulnérabilités de sécurité potentielles.
  - Évaluer la conformité aux bonnes pratiques.
  - Établir une stratégie de remédiation pour renforcer la sécurité de l'API.

Principales Vulnérabilités à Auditer

- **OWASP API Top 10** : Se concentrer sur les vulnérabilités les plus fréquentes :
  - Exposition excessive des données.
  - Manque de contrôle d’accès.
  - Injection de données malveillantes.
  - Problèmes d’authentification et de gestion de session.

Étapes Préliminaires : Collecte d’Information

Documentation : Examiner la documentation (Swagger/OpenAPI) pour repérer les endpoints exposés. (Si applicable)



Configuration des accès : Analyser les configurations réseau et de pare-feu pour voir si l’API est exposée inutilement.

Vérification des permissions : Confirmer les niveaux d'accès et permissions nécessaires pour chaque endpoint.

Outils d'Audit de Sécurité d'API

Postman / Insomnia : Pour tester manuellement les requêtes et réponses.

OWASP ZAP : Scanner de sécurité automatisé pour les API.

Burp Suite : Pour inspecter, modifier et rejouer les requêtes pour détecter des vulnérabilités.

Nmap : Vérification des ports et services exposés.

Exemple : Tester un Endpoint avec Insomnia / Postman

Configuration de Postman : Ajouter les en-têtes d'authentification si nécessaires.**Envoyer des requêtes**

GET/POST/DELETE : Vérifier les réponses et les codes d'état pour s'assurer que seuls les utilisateurs autorisés ont accès aux données.

Capturer les erreurs de validation : Observer les réponses pour détecter les éventuelles fuites de données.

Audit : Exposition Excessive des Données

Problème : Les API peuvent parfois exposer plus d'informations que nécessaire.

Test :

- Vérifier les champs dans la réponse JSON pour identifier les données sensibles.
- Envoyer des requêtes sans authentification et voir si des données sont divulguées.

Audit : Injection SQL dans une API

Problème : Les API acceptant des entrées de l'utilisateur non sécurisées peuvent être vulnérables aux injections SQL.

Test : Tenter d'injecter des caractères spéciaux dans les paramètres des requêtes.

Audit : Contrôle d'Accès Manquant

Problème : Les API qui ne contrôlent pas correctement les autorisations permettent aux utilisateurs d'accéder à des ressources non autorisées.

Test : Tenter d'accéder à des données avec un rôle d'utilisateur limité.

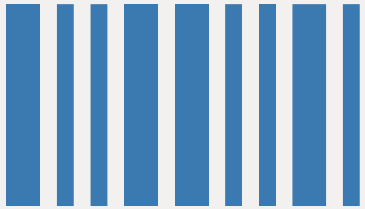
Exemple : Tester l'accès avec un utilisateur standard à un endpoint réservé aux administrateurs.

Audit : Mauvaise Gestion des Tokens JWT

Problème : L'absence de révocation de JWT permet à un jeton expiré ou volé d'accéder aux ressources.

Test : Vérifier la validité du JWT et s'assurer qu'il expire après déconnexion.

Audit : Test des Taux de Limite (Rate Limiting)



Problème : Sans limite de requêtes, l'API est vulnérable aux attaques de type DDoS.

Test : Effectuer plusieurs requêtes en un temps réduit pour vérifier si l'API bloque les requêtes après un certain seuil.

Audit : Contrôle des En-têtes HTTP et des CORS

Problème : Mauvaise configuration des en-têtes expose des informations sensibles et rend l'API vulnérable aux attaques CORS.

Test : Examiner les en-têtes Access-Control-Allow-Origin, X-Powered-By, et Server.

Correctif : Configuration des En-têtes Sécurisés avec Helmet en Node.js

```
const helmet = require('helmet');
app.use(helmet());
```

Audit : Utilisation du HTTPS et Sécurisation des Cookies

Problème : Transmission des données sensibles en HTTP sans chiffrement.

Test : Vérifier si l'API utilise HTTPS pour toutes les communications.

Conclusion de l'Audit de Sécurité

Évaluation : Consolider les résultats de chaque test, noter les points faibles et forts de l'API.

Remédiation : Proposer des solutions correctives pour les vulnérabilités détectées.

Documentation : Documenter tous les correctifs appliqués et créer un plan de maintenance pour garantir la sécurité future de l'API.

Merci !