OWASP:
Atelier 0 &Atelier 1

SABRI OMAIMA

Atelier 0 :

1. Créez un dossier dans votre espace de travail : …/formationOWASP.

1. Dedans, clonez le dépôt Juice Shop : « git clone https://github.com/juice-shop/juice-shop.git »

```
C:\Users\pc\Desktop\GI2-S1\Securité\Tps\FormationOWASP>git clone https://github.com/juice-shop/juice-shop.git
```

2. Dans le dossier juice-shop, supprimer le dossier .git (pour ne plus être dépendant du dépôt)

le .git supprimé

3. Installez les dépendances : « npm install ».

```
C:\Users\pc\Desktop\GI2-S1\Securité\Tps\FormationOWASP\juice-shop>npm install
```

4. Lancez le projet : « npm start »

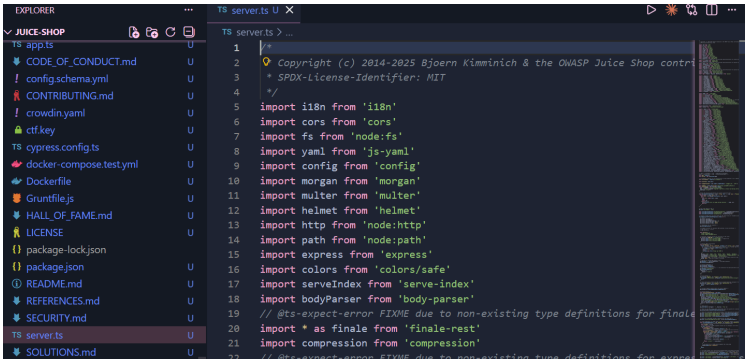
```
C:\Users\pc\Desktop\GI2-S1\Securité\Tps\FormationOWASP\juice-shop>npm start
```

5. Vérifiez que le projet est lancé

6. Ouvrir Visual Studio Code : « code . »

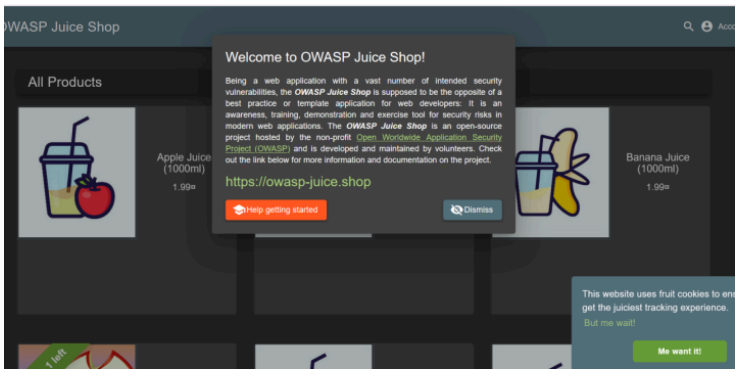
7. Ouvrir l'éditeur Visuel Studio Code (ou un autre à votre préférence), et analyser le code source.

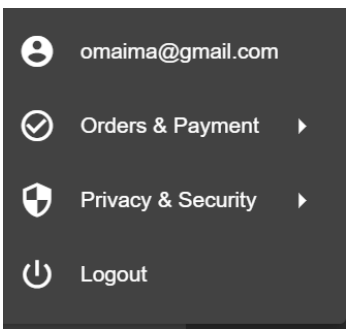
```
C:\Users\pc\Desktop\GI2-S1\Securité\Tps\FormationOWASP\juice-shop>npm start

> juice-shop@19.0.0 start
> node build/app

info: Detected Node.js version v22.14.0 (OK)
info: Detected OS win32 (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Entity models 20 of 20 are initialized (OK)
info: Required file server.js is present (OK)
info: Required file tutorial.js is present (OK)
info: Required file index.html is present (OK)
info: Required file styles.css is present (OK)
info: Required file runtime.js is present (OK)
info: Required file main.js is present (OK)
info: Required file vendor.js is present (OK)
info: Port 3000 is available (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Domain https://www.alchemy.com/ is reachable (OK)
info: Server listening on port 3000
```



8. Visitez la page : http://localhost:3000/#/score-board et découvrez son contenu

9. Créez un utilisateur, loggez vous avec et découvrez les pages.

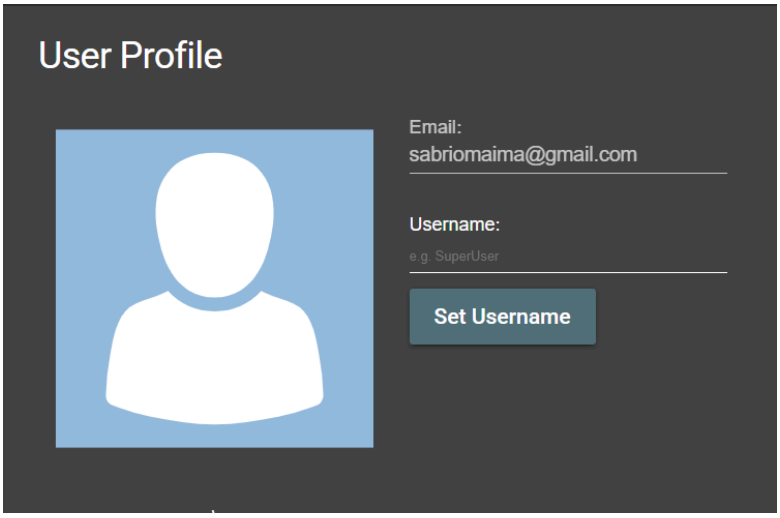




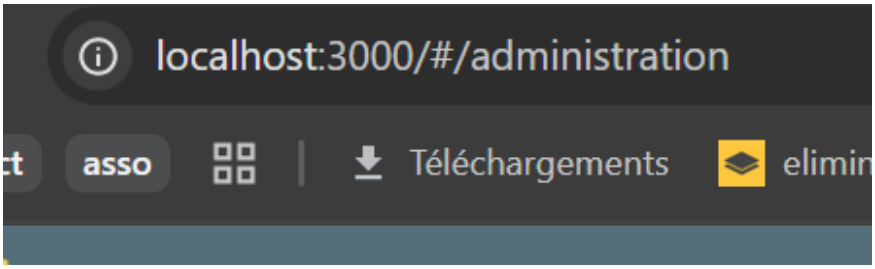
Atelier 1 :

1. Force Browsing (Accès Direct aux Routes Administratives) :

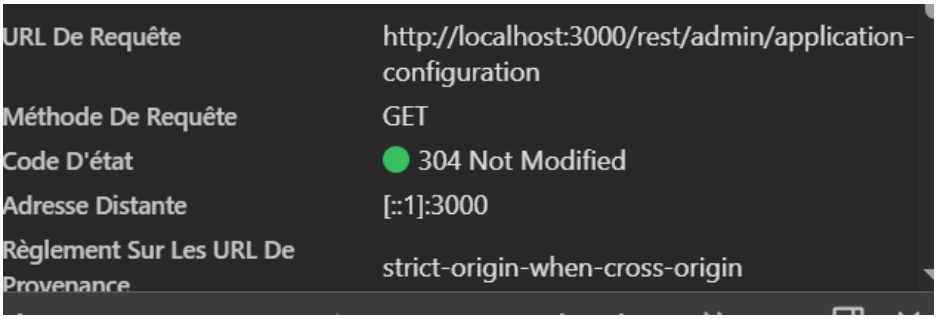
1. Créez un compte utilisateur standard.



2. Connectez-vous et naviguez manuellement vers http://localhost:3000/#/administration.



3. Observez que le Frontend bloque l'affichage mais que les appels réseau aux APIs Backend passent (vérifiez dans l'onglet Network des DevTools).



Mot "admin" dans l'URL La vulnérabilité est CONFIRMÉE
utilisateur "customer"
↓
Navigue vers /#/administration
↓
Frontend Angular Guard vérifie le token
↓
Voit role = "customer"
↓
❌ BLOQUE l'affichage de la page
MAIS EN MÊME TEMPS...
↓
Le JavaScript fait des appels au Backend :
✅ GET /rest/user/whoami → 304 (succès)
✅ GET /rest/admin/application-configuration → 304 (succès)
🚨 Le Backend RÉPOND sans vérifier les permissions !

4. Dans le fichier « app.routing.ts », cherchez le path : « administration »

```
// vuln-code-snippet start adminSectionChallenge scoreBoardChallenge web3SandboxChallenge
const routes: Routes = [
  { // vuln-code-snippet neutral-line adminSectionChallenge
    path: 'administration', // vuln-code-snippet vuln-line adminSectionChallenge
    component: AdministrationComponent, // vuln-code-snippet neutral-line adminSectionChallenge
    canActivate: [AdminGuard] // vuln-code-snippet neutral-line adminSectionChallenge
  }, // vuln-code-snippet neutral-line adminSectionChallenge
```

LE PROBLÈME : Ce Guard s'exécute dans le navigateur (côté client) !
L'utilisateur contrôle son navigateur
Il peut modifier le code JavaScript
Il peut changer le token dans localStorage

5. Dans le fichier « app.guards.ts », localiser la fonction « canActive » de la class « AdminGuard » et remarquez qu'elle se base uniquement sur le « role admin » dans le token :

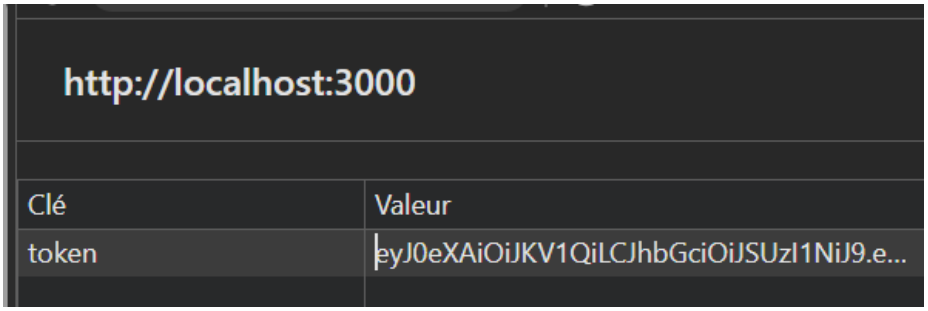
```
canActivate () {
  const payload = this.loginGuard.tokenDecode()
  if (payload?.data && payload.data.role === roles.admin) {
    return true
  } else {
    this.loginGuard.forbidRoute()
    return false
  }
}
```

Le Guard se fie uniquement au contenu du token
- Il ne demande pas confirmation au serveur
- Il ne vérifie pas si vous êtes vraiment admin dans la base de données

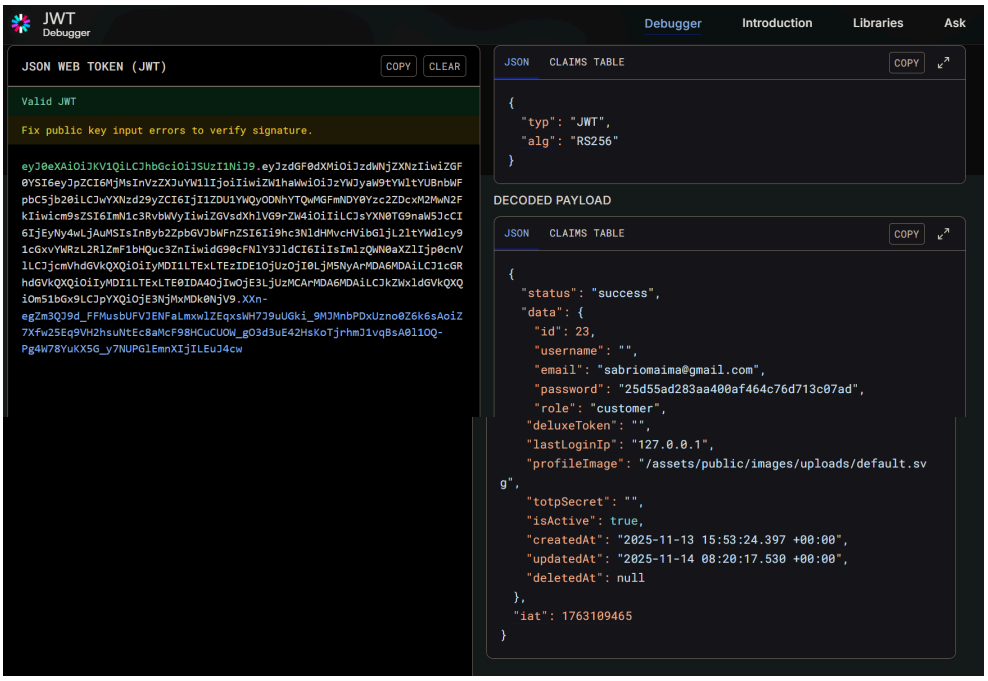
2. JWT Tampering (Élévation de Privilèges)

1. Connectez-vous avec un compte standard.

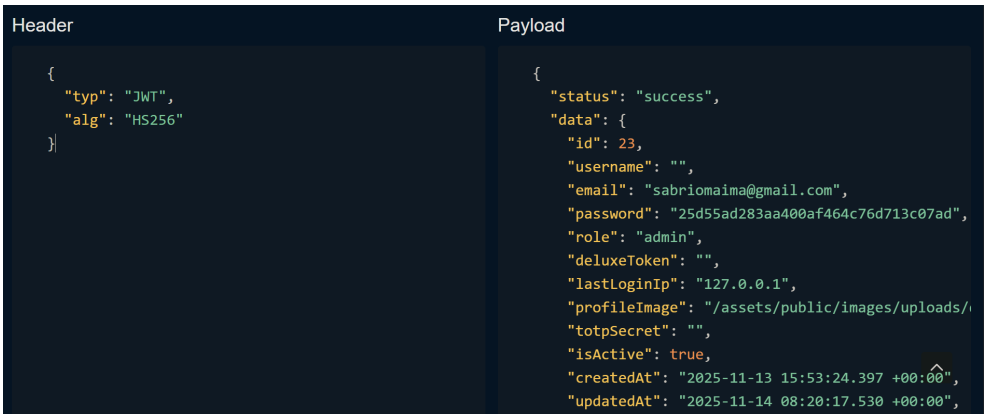
2. Dans les DevTools, onglet Application, copiez le « token » depuis le « Local Storage ».



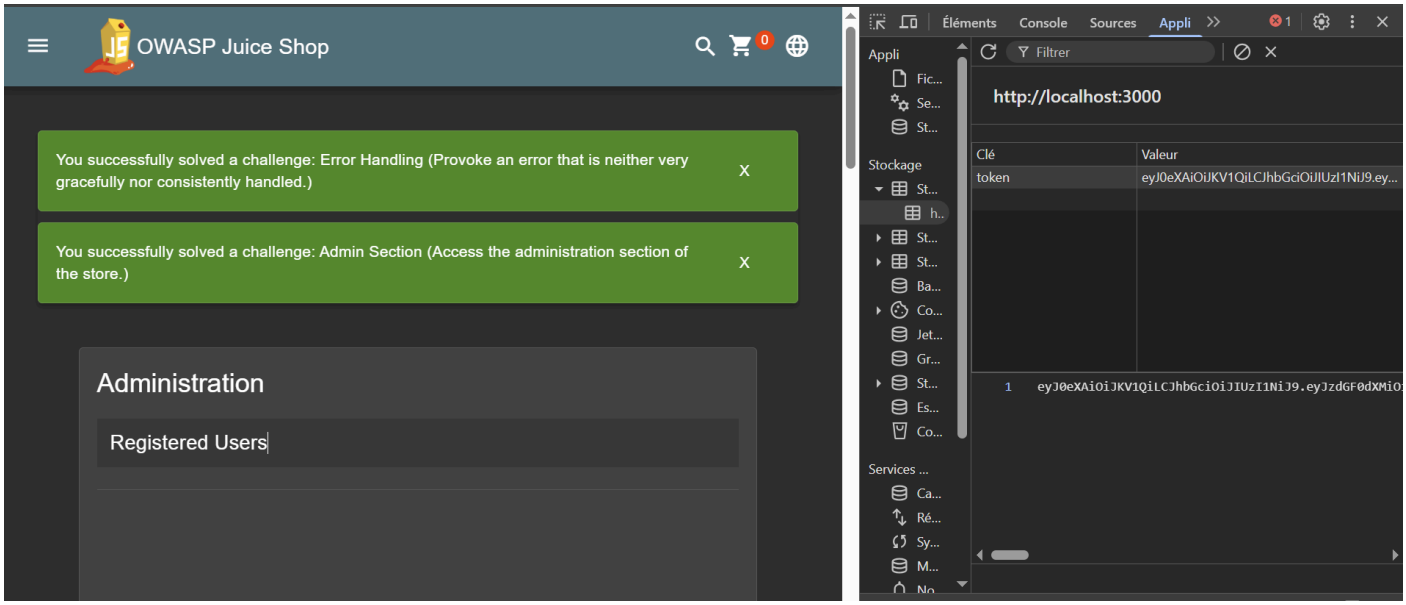
3. Collez-le sur jwt.io.



4. Modifiez le « role » : « user » en « admin », changez l'algorithme en HS256 (c'est pour illustrer une faiblesse, pas une bonne pratique)



5. Copiez le token modifié et remplacez-le dans le Local Storage.
6. Rafraîchissez la page et naviguez vers /#/administration.



Correction :

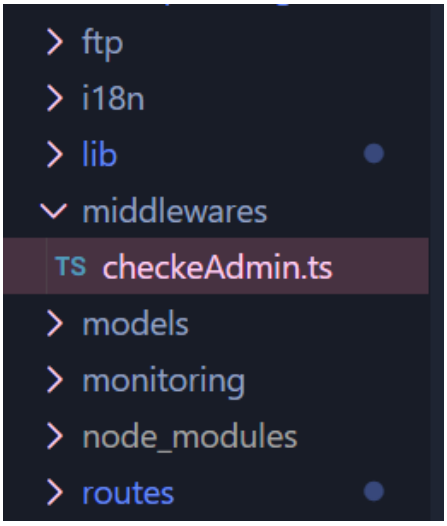
```
function jwtChallenge (challenge: Challenge, req: Request, algorithm: string, email: string | RegExp) {
  const token = utils.jwtFrom(req)
  if (token) {
    const decoded = jws.decode(token) ? jwt.decode(token) : null

    if (decoded === null || typeof decoded === 'string') {
      return
    }

    jwt.verify(token, security.publicKey, { algorithms: ['RS256'] }, (err: jwt.VerifyErrors | null) => {
      if (err === null) {
        challengeUtils.solveIf(challenge, () => {
          return hasAlgorithm(token, algorithm) && hasEmail(decoded as { data: { email: string } }, email)
        })
      }
    })
  }
}
```

Dans la premiere version on ne précise pas quel algorithme est autorisé. Donc un attaquant peut essayer de :

- mettre alg: "none"
- forger un token HS256(comme on a fait).
- d'autres attaques de substitution d'algorithmes

Mais apres la correction on n'accepte que l'algorithme RS256.
Cela empêche :

- l'attaque alg=none
- l'attaque où on remplace RS256 par HS256
- toute substitution d'algorithme
- l'utilisation d'un algorithme faible



Ona aussi ajouter cette fonction checkAdmin est un middleware Express.

- Un middleware est une fonction qui intercepte les requêtes HTTP avant qu'elles atteignent les routes finales. Ici, le rôle est de vérifier que l'utilisateur est un administrateur avant de laisser passer la requête.

```
import type { Request, Response, NextFunction } from 'express'
import * as utils from '../lib/utils'
import jws from 'jws'
import { UserModel } from '../models/user'

export const checkAdmin = (req: Request, res: Response, next: NextFunction) => {
  const token = utils.jwtFrom(req)

  if (token) {
    const decoded = jws.decode(token) ? jwt.decode(token) : null

    if (decoded === null || typeof decoded === 'string') {
      return
    }

    UserModel.findByPk(decoded.data.id)
      .then(user => {
        if (user && user.role === 'admin') {
          next()
        } else {
          res.status(403).send('Accès refusé : privilèges insuffisants')
        }
      })
      .catch(e => {
        console.error('Erreur DB:', e)
        res.status(500).send('Erreur interne serveur')
      })
  }
}
```

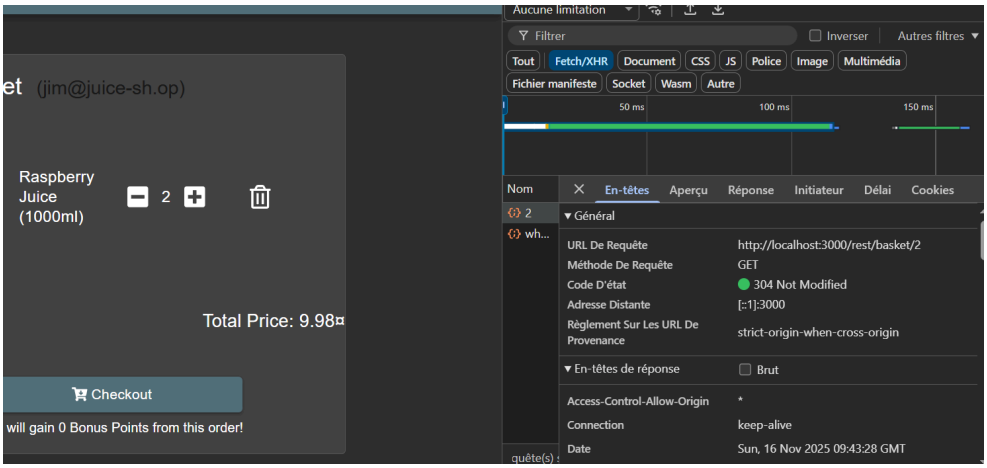
3. IDOR (Insecure Direct Object References)

C'est quoi une faille IDOR ?
C'est un bug où :

- un utilisateur peut changer un identifiant dans l'URL
- et accéder aux ressources d'un autre utilisateur
- parce que le serveur ne vérifie pas si c'est autorisé

1. Connectez-vous avec le compte jim@juice-sh.op (mot de passe: ncc-1701).

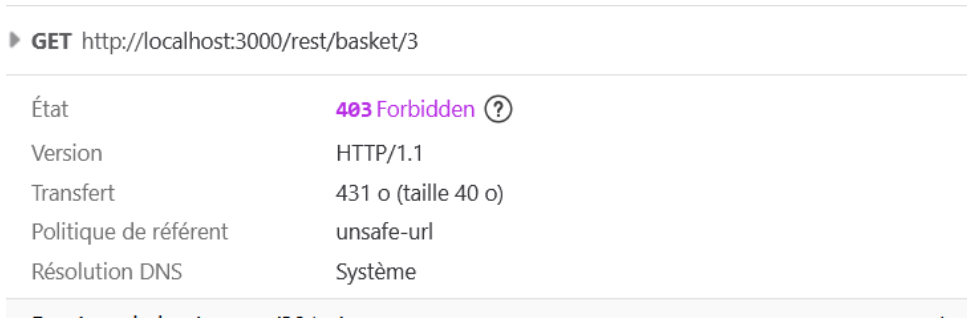
1. Ouvrez les DevTools, onglet Network.
2. Allez dans votre panier et repérez (dans DevTools) l'appel à GET /rest/basket/2.



3. Rejouez (edit and resend) la requête en modifiant l'ID : GET /rest/basket/3.
   (j'ai utilisé le nav Firefox au lieu de chrome pour simplifier la tâche)
4. Observez que le serveur renvoie le contenu du panier d'un autre utilisateur.

Correction :

```
const id = req.params.id
const userId = 2 // L'ID utilisateur est extrait du token valide, à dynamise
BasketModel.findByPk(id).then(basket => {
  // Vérification que la ressource appartient à l'utilisateur
  if (!basket || basket.UserId !== userId) {
    return res.status(403).json({ error: 'Accès interdit à ce panier' })
  }
}).catch((error: Error) => {
  next(error)
})
```



Excellent ! Le code 403 Forbidden confirme que le contrôle d'accès fonctionne correctement. L'utilisateur avec userId = 2 tente d'accéder au panier /basket/3, mais ce panier appartient à un autre utilisateur. Le serveur refuse légitimement l'accès avec l'erreur "Accès interdit à ce panier".

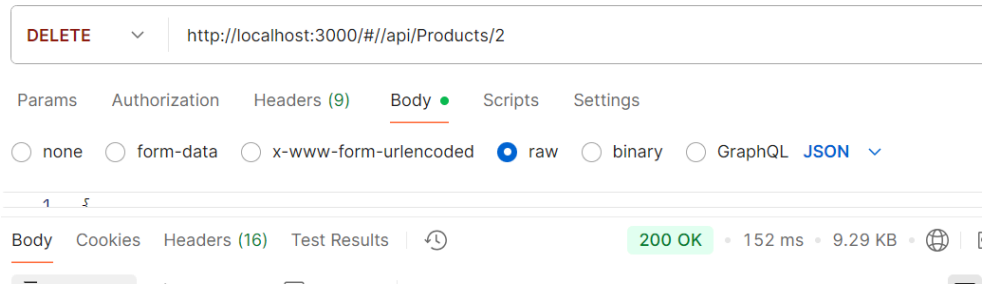
C'est exactement le comportement attendu pour protéger contre les attaques IDOR (Insecure Direct Object Reference). Sans cette vérification, n'importe quel utilisateur pourrait accéder aux paniers des autres en changeant simplement l'ID dans l'URL. La sécurité est maintenant renforcée : chaque utilisateur ne peut consulter que ses propres ressources.

4. Accès API Non Protégés (HTTP Methods)

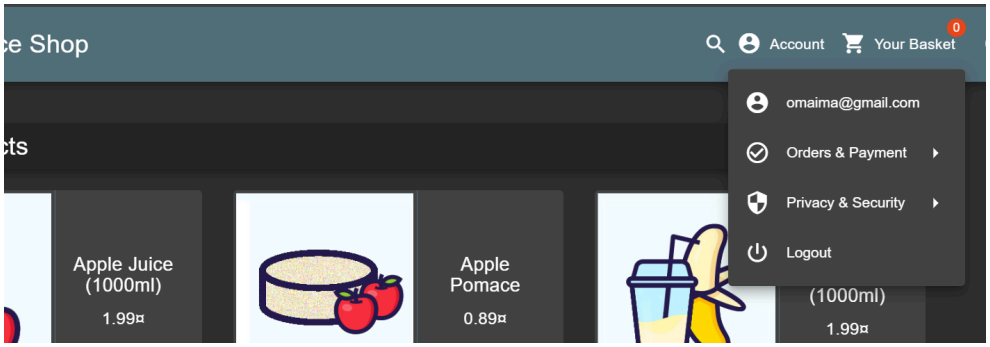
1. Identifiez une endpoint de suppression (méthodes DELETE), par exemple pour un produit (/api/Products/{id}).

```
app.delete('/api/Products/:id', security.denyAll())
```

2. Avec un outil comme « Postman », « curl » ou le « DevTools » du navigateur, envoyez une requête DELETE sur cette endpoint en étant connecté avec un compte standard.



3. Observez si l'action est exécutée sans vérification du rôle.



Lors du test, j'ai envoyé une requête : **DELETE /api/Products/2**
Alors que j'étais connecté avec un utilisateur différent, qui n'est pas propriétaire du produit ayant l'ID 2.
Malgré cela, le serveur a répondu :

- Status : 200 OK
- L'action de suppression a été exécutée
- Aucun contrôle d'identité ni de permission n'a été effectué

Correction :

```
app.delete('/api/Products/:id', checkAdmin, deleteProduit())
```

Problème observé :

- Avant la correction, un utilisateur standard (non administrateur) pouvait envoyer une requête DELETE sur n'importe quel produit et obtenir un statut 200 OK, même si le produit n'était pas lié à son compte.
- Cela signifie que l'accès n'était pas protégé, et tout utilisateur pouvait supprimer arbitrairement des ressources.

Cause :

- Le backend initial utilisait security.denyAll() ou n'avait pas de middleware de contrôle de rôle approprié. Il n'existait pas de fonction spécifique de suppression protégée (deleteProduit) avec vérification de rôle.

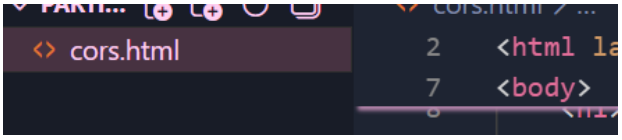
Résultat après correction :

- Les utilisateurs non administrateurs ne peuvent plus supprimer les produits → réponse 403 Forbidden.
- Seuls les administrateurs peuvent accéder à l'action de suppression.
- Cette correction applique le principe de "deny by default", garantissant que chaque endpoint est explicitement sécurisé.

5. Mauvaise Configuration CORS

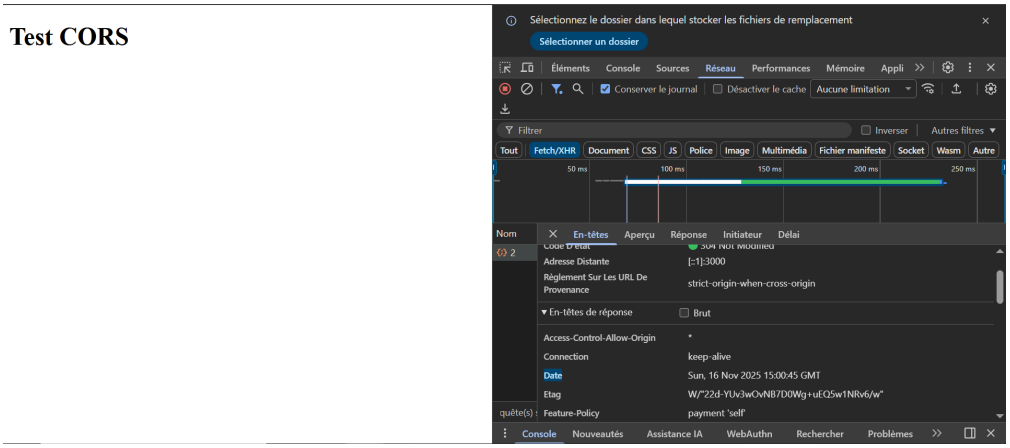
CORS = Cross-Origin Resource Sharing
(en français : partage de ressources entre origines différentes)
C'est un mécanisme de sécurité des navigateurs qui contrôle quelles ressources web peuvent être demandées depuis un autre domaine.

1. Créez une page HTML simple hébergée sur un autre domaine (ou localhost sur un autre port).
2. Dans un script, tentez une requête AJAX vers http://localhost:3000/rest/basket/2 en incluant le cookie ou token valide (qui peut avoir été volé par l'attaquant).

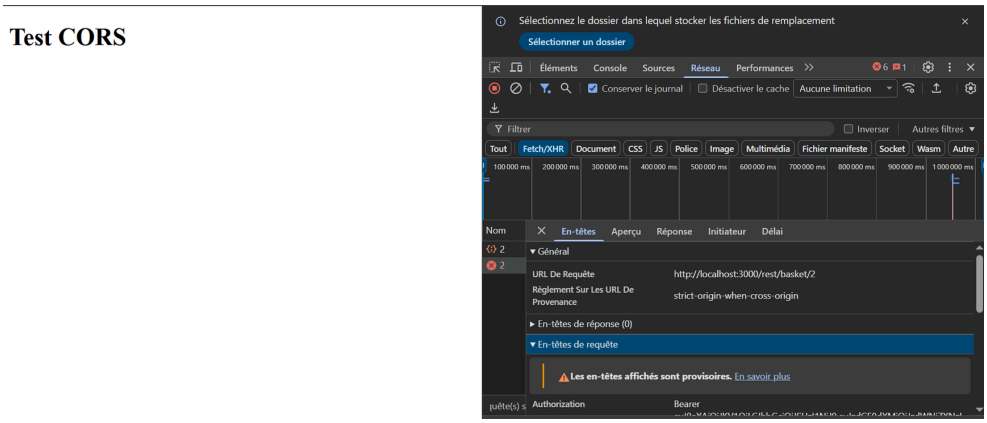



```
<!DOCTYPE html>
<html>
<body>
  <h1>Test CORS </h1>
  <script>
    const token = 'eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI

    const userId = 2;

  fetch(`http://localhost:3000/rest/basket/${userId}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${token}`,
    'Content-Type': 'application/json'
  },
})
.then(res => res.json())
.then(data => console.log(data))
.catch(err => console.error('Erreur :', err));
  </script>
</body>
</html>
```

3. Si le serveur répond avec Access-Control-Allow-Origin: * ou autorise votre domaine, les données seront accessibles au script malveillant.



Correction :



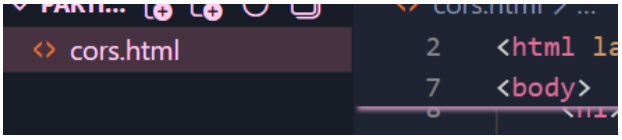
Après avoir appliqué la correction, j'ai relancé le test CORS depuis la meme page HTML.
Lorsque la requête AJAX tente d'accéder à :
http://localhost:3000/rest/basket/2

le navigateur bloque totalement la réponse pour des raisons de sécurité liées au CORS.
Dans l'onglet Network, on voit :
✓ La requête part bien depuis le navigateur
– URL correcte
– Méthode GET
– Le token ou cookie a été envoyé (on voit un header Authorization).
✗ Mais AUCUNE réponse n'est visible dans "Response Headers"
– La section En-têtes de réponse (0) est vide
– Le navigateur affiche : "Les en-têtes affichés sont provisoires"
👉 Conclusion : le navigateur a bloqué la réponse à cause du CORS.

5. Mauvaise Configuration CORS

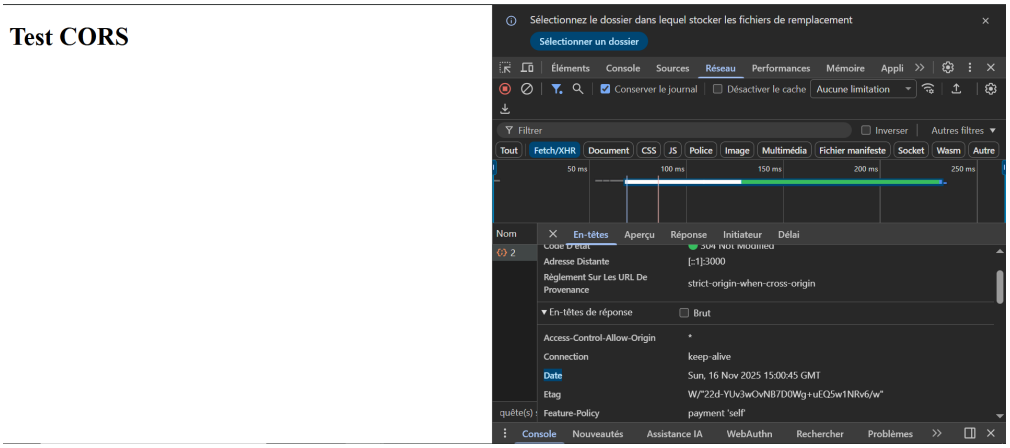
CORS = Cross-Origin Resource Sharing
(en français : partage de ressources entre origines différentes)
C'est un mécanisme de sécurité des navigateurs qui contrôle quelles ressources web peuvent être demandées depuis un autre domaine.

1. Créez une page HTML simple hébergée sur un autre domaine (ou localhost sur un autre port).
2. Dans un script, tentez une requête AJAX vers http://localhost:3000/rest/basket/2 en incluant le cookie ou token valide (qui peut avoir été volé par l'attaquant).

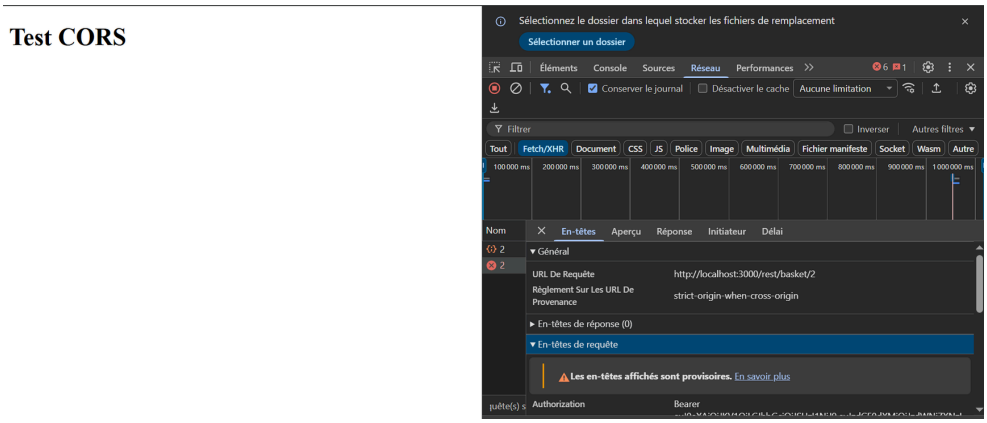



```
<!DOCTYPE html>
<html>
<body>
  <h1>Test CORS </h1>
  <script>
    const token = 'eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI

    const userId = 2;

  fetch(`http://localhost:3000/rest/basket/${userId}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${token}`,
    'Content-Type': 'application/json'
  },
})
.then(res => res.json())
.then(data => console.log(data))
.catch(err => console.error('Erreur :', err));
  </script>
</body>
</html>
```

3. Si le serveur répond avec Access-Control-Allow-Origin: * ou autorise votre domaine, les données seront accessibles au script malveillant.



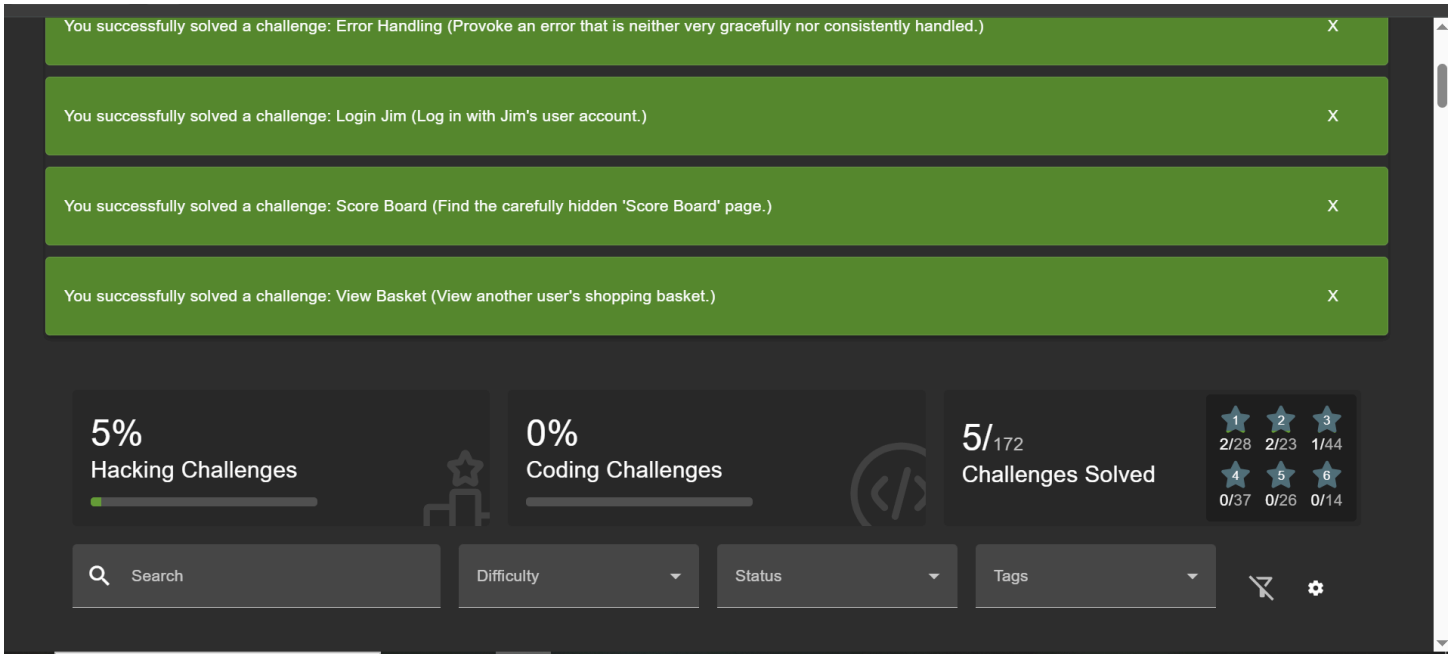
Correction :



Après avoir appliqué la correction, j'ai relancé le test CORS depuis la meme page HTML.
Lorsque la requête AJAX tente d'accéder à :
http://localhost:3000/rest/basket/2

le navigateur bloque totalement la réponse pour des raisons de sécurité liées au CORS.
Dans l'onglet Network, on voit :
✓ La requête part bien depuis le navigateur
– URL correcte
– Méthode GET
– Le token ou cookie a été envoyé (on voit un header Authorization).
✗ Mais AUCUNE réponse n'est visible dans "Response Headers"
– La section En-têtes de réponse (0) est vide
– Le navigateur affiche : "Les en-têtes affichés sont provisoires"
👉 Conclusion : le navigateur a bloqué la réponse à cause du CORS.

Resultat :


You successfully solved a challenge: Error Handling (Provoke an error that is neither very gracefully nor consistently handled.)
X
You successfully solved a challenge: Login Jim (Log in with Jim's user account.)
X
You successfully solved a challenge: Score Board (Find the carefully hidden 'Score Board' page.)
X
You successfully solved a challenge: View Basket (View another user's shopping basket.)
X
5%
Hacking Challenges
0%
Coding Challenges
5/172
Challenges Solved
1 2/28
2 2/23
3 1/44
4 0/37
5 0/26
6 0/14
Search
Difficulty
Status
Tags