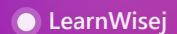


# Checklist de durcissement et projet de synthèse

Guide de la leçon · Guide du lab / de l'examen



## Guide de la leçon

HTTPS et en-têtes de sécurité, attributs des cookies, paramètres de session, mises à jour des dépendances, limitation du débit, durcissement du déploiement et revue de sécurité d'AccessOps. Ce que cela implique pour un développeur qui sécurise une application Wisej.NET de production — et comment aborder ce module.



### Enjeux en production

HTTPS et en-têtes de sécurité, attributs des cookies, paramètres de session, mises à jour des dépendances, limitation du débit, durcissement du déploiement et revue de sécurité d'AccessOps.

## 1. Le problème en production

AccessOps vérifie désormais les mots de passe par rapport à un hachage salé, résout les permissions dans un service, n'envoie les documents que par `DocumentService` et audite chaque approbation. Rien de tout cela n'aide si la première requête circule en clair, si un script peut lire le cookie de session, ou si le build embarque un package faisant l'objet d'un avis de sécurité publié. Le dernier kilomètre revient à l'hébergement : le certificat, les en-têtes de réponse, les attributs des cookies, les limites de requête, et la version de chaque élément du graphe de dépendances.

## 2. Concepts clés

| Concept                 | Pourquoi c'est important                                                                                                                                                                                                         |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HSTS                    | Un en-tête <code>Strict-Transport-Security</code> qui demande au navigateur de refuser le HTTP en clair pour cet hôte pendant une durée donnée, afin qu'une visite ultérieure ne soit jamais interceptée lors de la redirection. |
| Content-Security-Policy | Un en-tête qui liste les origines depuis lesquelles une page peut charger scripts, styles et frames ; il transforme un script injecté en requête bloquée au lieu d'un code qui s'exécute dans une session active.                |
| frame-ancestors         | La directive CSP ( <code>X-Frame-Options</code> en est l'ancien équivalent) qui empêche un autre site d'encadrer AccessOps pour pousser un manager à cliquer sur Approve.                                                        |
| attributs des cookies   | <code>Secure</code> tient le cookie de session à l'écart du HTTP en clair, <code>HttpOnly</code> le tient à l'écart de JavaScript, et <code>SameSite</code> l'empêche d'accompagner une requête lancée par un autre site.        |
| limites de requête      | Des plafonds, au niveau de l'hébergement, sur la taille du corps et des en-têtes, qui rejettent un upload surdimensionné avant qu'une seule ligne de code d'AccessOps ne s'exécute.                                              |
| token bucket            | Un compteur qui n'accorde une tentative que tant qu'il détient un jeton et qui se recharge à un rythme fixe, de sorte que le trafic peut connaître un léger pic puis ralentit fortement.                                         |
| audit des dépendances   | Une vérification reproductible de chaque package direct et transitif, et du runtime, au regard des avis de sécurité publiés — consignée avec une date et une décision.                                                           |

## 3. Déroulé de la leçon

### HTTPS est l'affaire de l'hébergement, HSTS celle du navigateur

TLS est terminé en dehors de votre C# : par IIS avec un certificat lié, par Kestrel configuré avec un certificat, ou par le reverse proxy placé devant eux. Votre rôle est de rendre le HTTP en clair inutile. Redirigez chaque requête HTTP au niveau de l'hôte, puis envoyez `Strict-Transport-Security` avec un `max-age` long pour que le navigateur cesse complètement de tenter le schéma non sécurisé. N'ajoutez `includeSubDomains` qu'une fois que chaque sous-domaine d'AccessOps sert réellement du TLS, et considérez `preload` comme une décision sur laquelle on ne revient pas rapidement. Un détail de déploiement piège souvent : derrière un proxy, la requête atteint l'application sans chiffrement, et sans gestion des forwarded headers l'application se croit non sécurisée et ne marque rien comme `Secure` .

Les en-têtes sont les contrôles les moins coûteux de ce cours, et ils se définissent une fois pour tout le pipeline plutôt qu'écran par écran. `X-Content-Type-Options: nosniff` empêche un navigateur de deviner qu'un document déposé est en réalité du HTML. `Referrer-Policy: strict-origin-when-cross-origin` empêche un identifiant de document de fuiter dans les logs d'un autre site. `frame-ancestors 'none'` refuse purement et simplement l'encadrement. Définissez-les là où passe chaque réponse, y compris le handler de lien de partage signé du Module 5 — une URL distincte, et celle que l'on oublie.

Content-Security-Policy demande ici de la prudence, car le navigateur exécute le client Wisej.NET. Ce client, les ressources de son thème et vos scripts de widgets viennent de la même origine, donc `default-`

`src 'self'` les couvre ; le moteur de mise en page écrit des styles inline, donc `style-src` nécessite généralement `'unsafe-inline'` ; et l'interop qui construit du code client à l'exécution vous oblige à élargir `script-src` . Déployez d'abord en `Content-Security-Policy-Report-Only` , parcourez chaque écran, lisez les violations, puis passez en mode bloquant.

```
app.UseForwardedHeaders();           // the proxy terminated TLS, not us
app.UseHttpsRedirection();
app.UseHsts();                       // Strict-Transport-Security
app.Use(async (ctx, next) =>
{
    var h = ctx.Response.Headers;
    h["X-Content-Type-Options"] = "nosniff";
    h["Referrer-Policy"] = "strict-origin-when-cross-origin";
    h["Content-Security-Policy-Report-Only"] = AccessOpsCsp;
    await next();
});
// the Wisej.NET handler runs after this, so every response carries the headers
```

## Cookies, limites de session, et un bouton qui n'est pas un endpoint

Une fois la connexion réussie, le cookie de session devient l'identifiant : il reçoit donc les trois attributs.

`Secure` signifie qu'il ne circule jamais sans TLS. `HttpOnly` signifie qu'une injection de script réussie dans un widget personnalisé ne peut toujours pas le lire. `SameSite` réglé sur `Lax` ou `Strict` signifie qu'une requête lancée par un autre site ne le transporte pas — la défense la moins coûteuse contre une approbation falsifiée. Sur ASP.NET Framework, ce sont des paramètres des éléments cookie et session-state de `Web.config` ; sur ASP.NET Core, ce sont des options de la politique de cookies. Vérifiez-les dans les outils de développement du navigateur, pas en relisant la configuration.

La durée de vie de session comporte deux moitiés. Wisej.NET lit ses propres paramètres dans le fichier de configuration JSON de l'application, à côté de la page de démarrage, et c'est là que se règle le délai d'expiration de session du framework : le filet de sécurité qui récupère une session abandonnée et son état côté serveur. Ce n'est pas la déconnexion pour inactivité construite au Module 2, car le client maintient sa connexion active et un navigateur abandonné peut ressembler à du trafic — l'inactivité doit être mesurée depuis le dernier véritable événement utilisateur, et la déconnexion explicite doit libérer le

`SessionContext` dans `Application.Session` pour que le bouton Précédent mène à une page de connexion. Les limites de requête sont le troisième paramètre : plafonnez la taille du corps et des en-têtes au niveau de l'hôte pour qu'un upload surdimensionné soit rejeté avant d'entrer dans `DocumentService` .

La limitation du débit se partage selon une ligne importante. Le handler de lien de partage est une vraie URL : un middleware peut donc le limiter directement — un limiteur token bucket partitionné par adresse client, qui répond 429 sans révéler si le lien existait. Le bouton de connexion n'est pas une URL : chaque événement Wisej.NET, du défilement d'une grille à un clic sur Sign in, atteint le même endpoint de l'application au sein d'une session existante, et un middleware ne peut pas les distinguer. La limitation des connexions vit donc dans le service, sous forme de bucket indexé par le nom d'utilisateur soumis et l'adresse client, et vérifié avant la vérification du hachage. Un refus renvoie le même message générique qu'un mauvais mot de passe, est audité avec son identifiant de corrélation, et est compté dans un stockage partagé plutôt que dans un champ statique lorsqu'AccessOps tourne sur plusieurs serveurs.

## À jour, et preuves à l'appui

Exécutez `dotnet list package --vulnerable --include-transitive` et lisez tout le résultat : le package que vous n'avez jamais choisi, tiré trois niveaux plus bas, est le constat habituel. Ajoutez `--outdated` et `--deprecated` pour ce qui est simplement périmé ou abandonné. Les SDK .NET récents vérifient aussi les avis de sécurité pendant la restauration, ce qui fait de l'audit une sortie du build plutôt qu'une tâche dont quelqu'un doit se souvenir. Intégrez-le au pipeline pour qu'un nouvel avis fasse échouer un build au lieu d'apparaître lors d'un test d'intrusion.

Les packages ne sont qu'une partie du graphe. Le runtime .NET, le serveur web et l'image de base du conteneur ont chacun besoin d'un rythme de mise à jour et d'une version consignée. Wisej.NET lui-même est une dépendance versionnée : figez sa version, lisez les notes de version avant de migrer, et retestez le thème et les widgets personnalisés après une mise à niveau majeure. Le JavaScript tiers présent dans la racine web est invisible pour le graphe de packages : listez chaque fichier à la main avec sa version et son origine. Puis consignez l'audit par écrit — date, outil, constats, une décision pour chacun (mettre à jour, atténuer, ou accepter avec une date d'expiration) et une date de prochaine revue.

## La revue avant mise en production, module par module

Le livrable du projet de synthèse est une revue, pas une fonctionnalité. Reprenez la carte de placement des contrôles du Module 1 et parcourez le cours dans l'ordre, en vérifiant l'application plutôt que l'intention, et testez en attaquant plutôt qu'en utilisateur : demandez le document d'un autre département en tant qu'employé ordinaire, rejouez un lien de partage après son heure de validité, appelez une méthode d'approbation en tant que quelqu'un qui n'est pas le manager, soumettez vingt mauvais mots de passe en dix secondes.

- Module 2 — verrouillage, hachage, identité vérifiée, délai d'inactivité, déconnexion qui libère l'état.
- Module 3 — chaque méthode de service protégée, règles par département et par manager sous l'UI, requête falsifiée refusée.
- Module 4 — PKCE, validation de l'ID token et du nonce, mappage des claims, déconnexion mettant fin aux deux sessions.
- Module 5 — validation des uploads, téléchargements via le service, liens signés à expiration, requêtes paramétrées, sorties encodées.
- Module 6 — lignes d'audit pour les commandes sensibles, identifiants de corrélation dans les logs, ni secrets ni données personnelles dans l'un ou l'autre.
- Module 7 — HTTPS et HSTS, jeu d'en-têtes, attributs des cookies, limites de session et de requête, limitation du débit, audit daté.

Chaque constat reçoit une gravité, un responsable et soit une correction, soit un risque accepté avec une date d'expiration ; une checklist comportant des cases vides inexpliquées n'est pas terminée. La validation finale signifie une seule chose : chaque frontière tracée au Module 1 dispose désormais d'un contrôle appliqué par un service, et non par un écran.



### Réflexe sécurité

Le navigateur n'est jamais digne de confiance. Chaque décision importante — qui est l'utilisateur, ce qu'il peut faire, quel enregistrement il peut voir — est prise sur le serveur, dans un service, et l'UI se contente de la refléter. Si une règle n'existe que sous la forme d'un bouton masqué, elle n'existe pas.



### Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Configurer HTTPS, HSTS, les attributs de sécurité des cookies et les en-têtes de sécurité pour un déploiement Wisej.NET.
- ✓ Ajuster les paramètres de session, les limites de requête et la limitation du débit pour résister aux abus.
- ✓ Maintenir les dépendances et le runtime à jour grâce à un processus reproductible.
- ✓ Mener une revue de sécurité avant mise en production à l'aide d'une checklist et livrer le projet de synthèse AccessOps.

## Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



### Lab pratique

Réalisez le projet de synthèse AccessOps : imposez HTTPS avec HSTS et définissez les en-têtes de sécurité, marquez les cookies Secure et HttpOnly avec une politique SameSite stricte, fixez le délai d'expiration de session et les limites de taille des requêtes, ajoutez une limitation du débit sur la connexion et les liens de partage, lancez un audit des dépendances et consignez les résultats, puis passez toute la checklist de ce cours sur l'application et corrigez chaque constat.

## Point de départ

Ouvrez la solution AccessOps telle qu'elle se présente après le Module 6 : une page de connexion avec verrouillage, un `SessionContext` portant l'identité vérifiée, un `PermissionService` qui protège `DocumentService` et `ApprovalService`, la connexion externe via OpenID Connect, une chaîne d'upload et de téléchargement avec un handler de lien de partage signé, et un `AuditService` qui écrit des lignes horodatées avec leur identifiant de corrélation. Rien n'a encore été touché dans la configuration de l'hébergement : l'application répond en HTTP en clair en développement, ses cookies ne portent aucun attribut, et aucun package n'a été vérifié au regard des avis de sécurité.

## Approche suggérée

- ✓ Liez un certificat et redirigez HTTP vers HTTPS au niveau de l'hôte, ajoutez `Strict-Transport-Security`, et vérifiez la gestion des forwarded headers si un reverse proxy se trouve devant AccessOps.
- ✓ Ajoutez dans le pipeline un endroit unique qui définit `X-Content-Type-Options`, `Referrer-Policy`, `frame-ancestors` et une Content-Security-Policy, en commençant en mode report-only jusqu'à ce que chaque écran d'AccessOps et le handler de lien de partage se chargent sans violation.
- ✓ Marquez le cookie de session `Secure` et `HttpOnly` avec une politique `SameSite` stricte, réglez le délai d'expiration de session Wisej.NET dans le fichier de configuration de l'application, et plafonnez la taille du corps des requêtes au niveau de l'hôte.
- ✓ Placez un token bucket indexé par nom d'utilisateur et adresse client dans le service de connexion, et un limiteur middleware sur l'URL des liens de partage, en auditant chaque refus avec son identifiant de corrélation.
- ✓ Exécutez `dotnet list package --vulnerable --include-transitive`, consignez les constats avec une décision et un responsable, puis passez la checklist des sept modules sur l'application en cours d'exécution et corrigez ce qu'elle révèle.

## Livrables attendus

- ✓ Configuration HTTPS, HSTS et en-têtes de sécurité
- ✓ Paramètres de durcissement des cookies et de la session

- ✓ Limitation du débit sur la connexion et les liens de partage
- ✓ Rapport d'audit des dépendances
- ✓ Checklist de revue de sécurité complétée avec les corrections

## Questions de révision

- ✓ Un middleware peut limiter le débit du handler de lien de partage, mais pas celui du bouton Sign in. Qu'est-ce qui distingue ces deux requêtes dans une application Wisej.NET, et où la limite de connexion doit-elle se trouver à la place ?
- ✓ Votre Content-Security-Policy est en mode bloquant et la grille s'affiche sans styles. Quelle directive en est probablement la cause, et comment le mode report-only vous l'aurait-il signalé avant que le déploiement ne le fasse ?
- ✓ L'audit des dépendances signale un package vulnérable que rien dans AccessOps ne référence directement. Comment est-il arrivé dans le build, et quelles sont les deux façons honnêtes de clore ce constat ?



### Réflexe sécurité

Le navigateur n'est jamais digne de confiance. Chaque décision importante — qui est l'utilisateur, ce qu'il peut faire, quel enregistrement il peut voir — est prise sur le serveur, dans un service, et l'UI se contente de la refléter. Si une règle n'existe que sous la forme d'un bouton masqué, elle n'existe pas.