

Mémoire de fin d'études

Master MIAGE
La triche sur un Serious Game



M. Minh-Huy LE
M2 Miage Classique
2016/2017

Maître de stage : **M. Henri DARMET**
Enseignants tuteurs : **M. Fabrice LEGOND-AUBRY**

Remerciements

Je tiens à remercier dans un premier temps M. Henri DARMET, pour mon affectation sur le projet I-Learning. En effet c'est au sein de l'entreprise VISEO que j'ai pu finaliser mes recherches sur mon sujet de mémoire. Ce projet m'a permis d'avoir d'une nouvelle expérience, dans les technologies et sur le plan humaine. Je remercie également toute l'équipe pédagogique et les intervenants professionnels, pour leurs enseignements durant ma formation [Master 2 \(M2\) Méthodes Informatiques Appliquées à la Gestion des Entreprises \(MIAGE\)](#). Je tiens également à remercier M. Fabrice LEGOND-AUBRY pour m'avoir aidé à choisir ce sujet et son suivi durant toute la durée de ce stage.

Contents

| | |
|---|-----------|
| Remerciements | 3 |
| Introduction | 6 |
| I Contexte et problématique | 7 |
| A Contexte | 7 |
| B Problématique | 7 |
| II Les attaques | 8 |
| A Attaque coté client | 8 |
| 1 Session hijacking | 9 |
| 2 Code inspection | 10 |
| 3 Code modification | 11 |
| 4 Objet modification | 14 |
| 5 Cross Site Scripting (XSS) | 14 |
| B Attaque coté serveur BDD | 15 |
| 1 NoSQL injection | 15 |
| III Protection | 17 |
| A Protection globale | 17 |
| 1 La mise à jour | 17 |
| 2 Identifier les vulnérabilité des composants | 17 |
| 3 Mauvaise configuration | 17 |
| 4 Les guides - Bonnes pratiques | 17 |
| B Protection coté client | 18 |
| 1 Offuscation | 18 |
| 2 Protection contre les modifications | 18 |
| C Protection coté serveur web | 19 |
| 1 Controle coté serveur | 19 |
| Bilan | 20 |
| Biographie | 21 |
| Glossaires | 22 |
| Annexes | 24 |
| A Debugguer avec chrome | 24 |
| B Snapshot memory avec chrome | 26 |
| C Event Listener Breakpoints | 28 |
| 1 Modification étape 1 | 28 |

2 Modification étape 2 31

Introduction

Dans le cadre de la formation [M2 MIAGE](#) en classique, les étudiants de l'université Paris Ouest Nanterre doivent effectuer un stage obligatoire de 5 mois et réaliser un mémoire afin de valider le [M2](#). C'est au sein de l'entreprise Viseo que j'ai décidé de réaliser mon stage sur le développement d'un jeu sérieux.

Le "jeu sérieux" plus connu sous le nom de "serious game", est en expansion. Présent déjà dans de nombreux secteurs (l'industrie aéronautique et de la défense, l'industrie automobile, l'éducation, l'énergie, ...), il touche de plus en plus de marchés, car ses méthodes sont efficaces [\[1, 2\]](#). Le concept du "serious game" a pour but d'enseigner de manière synthétique à travers des jeux. De cette façon, on rendrait l'enseignement plus amusant, moins formel, ce qui rendrait la formation plus attractive et qui serait un meilleur moyen pour s'adresser aux nouvelles générations "digitales natives" [\[3\]](#).

Ayant réalisé plusieurs stages assez techniques, j'ai souhaité continuer à élargir mes connaissances. C'est avec l'aide de mon tuteur d'enseignement que mes recherches porteront sur " La triche sur un serious game".

Ce mémoire sera présenté en plusieurs parties. Tout d'abord, on commencera à voir le contexte du sujet. On abordera ensuite les problématiques du projet, les triches qu'on peut réaliser. Ensuite, nous allons voir quels sont les moyens pour se protéger contre les tentatives de triche et on terminera avec le bilan.

I Contexte et problématique

A. Contexte

De plus en plus, les "jeux sérieux" s'intègrent dans les entreprises afin de former efficacement les collaborateurs. Ils sont d'une telle efficacité qu'on peut ranger ces jeux dans plusieurs différentes catégories. On peut même voir qu'en ce moment, on pousse encore plus loin ces "jeux sérieux" avec la réalité augmentée. Le but est de former efficacement tout en mélangeant la pédagogie et le plaisir à moins coûts.

A Viseo, le projet I-Learning a été imaginé par M. Henri DARMET et développé par les stagiaires de 2015/2016 en JS. Il porte le nom I-Learning comme Interactive-Learning, qui a pour but de former les collaborateurs ou stagiaires afin de limiter les formations longues et fatigantes. Mon équipe et moi avons repris le projet I-Learning dans le but d'implémenter de nouveaux jeux et améliorer les fonctionnalités existantes.



Le projet est codé en Full JS, sur le côté back-end, nous utilisons le NodeJS qui permet de faire du JS coté serveur. Le serveur utilise avec le module Express qui permet de gérer plus facilement les [Uniform Resource Locator \(URL\)](#). Les données sont stockées dans MongoDB qui permet d'avoir une souplesse de structure de données. Sur le côté front-end, nous utilisons le [Scalable Vector Graphics \(SVG\)](#) qui constitue le squelette de l'application Web. La décision du choix des technologies a été prise par les anciens stagiaires en fonction de la demande et des besoins du projet : ils ont étudié les technologies du moment et entre Canvas et [SVG](#), c'est [SVG](#) qui a été conservé car il permet de manipuler les [Document Object Model \(DOM\)](#).

B. Problématique

Comme tous les systèmes, certaines personnes se laissent à la tentation pour tricher. Parfois, on le fait involontairement, mais plus souvent cet acte est volontaire. La triche permet de se rassurer, pour être meilleur que les autres, obtenir le meilleur résultat, voir simplement contourner la sécurité du système. Ou encore d'autres personnes peuvent justifier une augmentation du salaire avec ces résultats dans le "serious game" si leurs compétences sont testées en entretien d'embauche par exemple. L'état du projet actuel présente peu de sécurités, il est toujours en développement, mais sans que la sécurité soit un aspect nécessaire. Ce qui donne beaucoup de possibilités de tricher lorsqu'on s'y intéresse de plus près.

Comment éviter la triche sur un serious game ? Nous essayerons de répondre le plus objectivement possible à cette question.

II Les attaques

Une attaque est l'origine de l'exploitation d'une faille d'un système informatique en attaquant l'intégralité, la confidentialité ou encore la disponibilité. Attaquer l'intégralité consiste à modifier le message/donnée originale, pour ce qui est de la confidentialité c'est obtenir les informations sans autorisations. Et pour ce qui est la disponibilité, cela consiste à nuire l'application/serveur afin qu'il ne réalise plus ces services comme il devrait. Pour contrer ces attaques, il faut connaître l'origine et le type d'attaque afin de mettre des mesures préventives. Il faut comprendre, que le pirate peut intervenir à n'importe quel maillon de la chaîne (de l'utilisateur au composant jusqu'à l'alimentations électrique) [4].

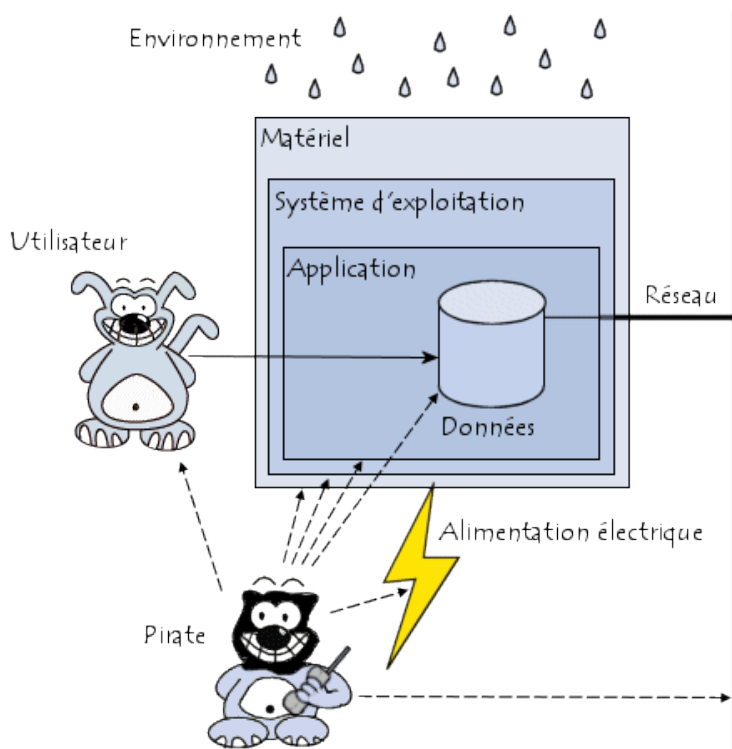


Figure II.1: Possibilités attaques

A. Attaque coté client

Dans cette partie, on va voir les attaques côté client qui sont les attaques qui visent l'utilisateur pendant l'utilisation de l'application. L'application peut être une application de type bureau, web ou mobile.

1. Session hijacking

La session hijacking est une attaque destinée à voler une session utilisateur. Comme le [Hypertext Transfer Protocol \(HTTP\)](#) est sans état, les applications Web se servent des cookies afin de maintenir les états des actions des utilisateurs par exemple la connexion d'un utilisateur.

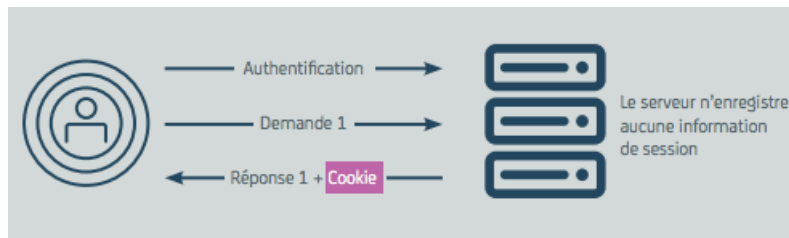


Figure II.2: Utilisation cookie

On peut voir un exemple d'utilisation de cookie avec la figure ci-dessus. Lorsque l'utilisateur se connecte, et pour éviter qu'on remette à chaque fois le login et le mot de passe, on se sert du cookie pour garder l'état de la connexion. On peut également se servir du cookie pour lorsque l'utilisateur ferme la page et reviens sur la page, il n'a pas à s'authentifier de nouveau et que la connexion se fait via le cookie.

Malheureusement ces cookies peuvent être exploités par les sessions hijacking, et il a plusieurs types de sessions hijacking :

- Interception (Man in the middle) : consiste à intercepter un cookie par écoute du réseau entre le serveur et le client. Il existe deux sorts d'écoutes réseau, le passif et l'actif. Le passif consiste simplement à écouter le réseau sans modifier les packages de données qui circulent. L'actif consiste à modifier les données afin de perturber, ou berner le serveur et le client. Avec l'interception, le pirate peut récupérer bien plus que le cookie comme les clés de chiffrements, les mots de passe, etc ... Lors de l'interception du cookie, l'attaquant pourra à la suite utiliser pour accéder à l'application web, et bénéficier des droits de l'utilisateur.
- Prédiction : deviner une session valide en analysant l'algorithme de génération qui se sert de certaines informations connues (le login, la date, le type de navigateur, ...). Plus l'application possède d'utilisateurs, plus cette attaque sera facile à appliquer.
- Force brute : est une attaque qui consiste à générer des tentatives massives et en exploitant sa prédictibilité afin de trouver une session valide.
- Fixation : le but de cette attaque est de pousser l'utilisateur à se connecter (souvent par phishing) sur une session créée par le pirate. L'utilisateur se connecte, la session se crée avec la valeur qui a été mise en place par l'attaquant. Il accède ensuite sur le site avec la valeur de la session piéger. Le pirate peut utiliser un outil requêtant l'application régulièrement afin d'éviter l'expiration de la session due à une période d'inactivité.
- Infection du navigateur (Man in the browser) : on fait en sorte d'implémenter un code qui permet d'envoyer la session de la victime.

Lorsque le pirate arrive à contourner une session, il peut se servir du compte de l'utilisateur afin de tricher. Si l'utilisateur est un administrateur, il peut voir ainsi toutes les bonnes, si l'utilisateur est un collaborateur le pirate peut consulter les jeux qui sont déjà effectués par la victime et ainsi éviter les mauvaises réponses [13, 14].

2. Code inspection

Actuellement, il y a aucun contrôle concernant les appels au serveur. À chaque action, le coté client fait appelle direct au serveur, et c'est au coté client qui va vérifier quelles sont les données à afficher.



Figure II.3: Question sur java basic

Voici un exemple de vue qu'on peut voir sur le côté collaborateur/stagiaire, on peut voir le nom de la formation "Java" et la partie "Basic". On voit également la question ainsi que les réponses proposées. Le problème, c'est que même si les données qui ne sont pas affichées, ils sont sur le côté client, et ces données on peut les consulter sur le côté client. Il y a plusieurs façons d'observer ces données :

- Lire le code source depuis le côté client (navigateur), mettre des breakpoints pour les observers.



Figure II.4: Debug sur le coté client

On peut voir sur la figure à côté, la variable label est la question, et si on fouille un peu plus on voit la variable rightAnswers qui nous permet de voir la bonne réponse "int". Pour éviter de suivre tout le code, il faut mettre les breakpoints à des endroits qui semblent intéressants pour trouver l'information au plus vite. Tous les détails concernant le debug coté client sont décrits en annexe "Debugguer avec chrome".

- Prend une photo de la mémoire de la page (JS Objet et tous les noeuds des DOM).

```

▼ correct :: system / Oddball @53
  ► map :: system / Map @319
    ► 2 :: "true" @655
    ► 4 :: "int" @379
  ► filled :: system / Oddball @53
  ► imageLoaded :: system / Oddball @53
  ► invalidLabelInput :: system / Oddball @55
  ► selected :: system / Oddball @55
  ► explanation :: @200475
  ► properties :: (object properties)[] @200477
  ► bgColor :: Array @169857
  ► colorBordure :: Array @169855
  ► label :: "int" @188579

```

Figure II.5: Snapshot memory

Avec le snapshot de la mémoire, on peut observer un ensemble d'objets avec le nom de la classe. Il comporte de légèrement différence avec la vue debug, mais on se repère assez rapidement. Et contrairement au débogueur, on n'a pas besoin de breakpoints. Ceci prend tous les objets en mémoire qui ont un lien avec le JS ou des objets DOM. Il faut ensuite chercher la donnée qui nous intéresse. On peut également enregistrer la mémoire avec toutes les actions de la page. Les détails sont décrits en annexe "Snapshot memory avec chrome".

3. Code modification

L'autre moyen de voir les bonnes réponses, c'est de modifier le code et de faire passer pour administrateur. Lors de la connexion, le côté front vérifie si on est administrateur ou collaborateur, il va ensuite charger les formations et instancier la vue correspondante à l'utilisateur.

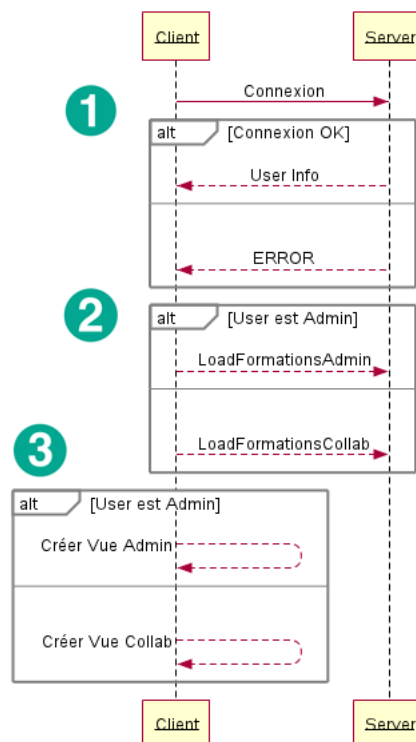


Figure II.6: SD Client - Connexion à chargement Dashboard

On peut voir sur la figure ci-dessus, le diagramme de séquence lorsqu'on clique sur connexion. Ce qu'on va essayer de faire dans le premier temps, c'est de voir si on peut se faire passer pour admin sans se connecter avec le login et le mot de passe. On va donc modifier le comportement de l'étape 1. Faire en sorte que si on clique sur connexion ça n'envoie pas de requête sur le serveur, mais il va passer directement en mode admin. Pour trouver où est le code de l'exécution du bouton connexion, on peut faire comme ce qu'on a vu précédemment avec le debug de manière plus ou moins aléatoire. Où on peut se servir de "Event listener breakpoints", pour s'arrêter à une action/événement particulier.

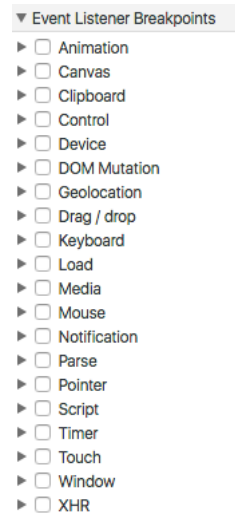


Figure II.7: Event Listener Breakpoints

Comme le bouton connexion enclenche l'évènement click, on va raisonner de manière logique, en tentant "Mouse: click". Lorsqu'on tente de faire un click, on s'arrête bien dans le code grâce au debug Event Listener Breakpoints. Mais le code où on est placé n'est pas celui qu'on souhaite avoir, on tombe sur les codes d'assez bas niveau. De plus il ne se passe absolument rien, on n'observe pas d'erreur comme "Veuillez remplir tous les champs" ou "Veuillez saisir une adresse email correcte" car on a laissé le champ vide. La raison est due au timeout, le fait qu'il se bloque sur le breakpoint n'a pas déclenché son évènement attendu.

Avec la première tentative, on a rien obtenu, ce qu'on va faire à la suite c'est étudier le comportement de connexion pour voir si on peut s'arrêter avec d'autres Event Listener Breakpoints. Et lorsqu'on étudie attentivement l'action du bouton connexion, on observe plusieurs comportements/événements :

- Lorsqu'on clique, un message apparaît si l'adresse email ou le mot de passe n'est pas bon, puis qui disparaît après un certain temps (Event responsable : Timer).
- On peut faire en sorte de faire connexion avec le bouton "Entrée" à la place d'un click (Event responsable : Keyboard).

Maintenant qu'on observe de nouveaux comportements du bouton connexion, on va tenter de s'arrêter avec ces events. Avec la tentative de l'évènement "Timer", on s'arrête au niveau où on définit un timeout pour enlever le message d'erreur qui se trouve exactement dans la fonction qu'on souhaite atterrir "connexionButtonHandler". Ensuite on va tenter de voir avec le second event "Keyboard" avec le bouton entrée. Lors de cette tentative, on atterrit sur un code un peu plus haut qui gère les touches claviers comme "Tab" ou "Entrée". On voit dans le management de la saisie de la touche "Entrée", que la fonction "connexionButtonHandler" est appelée.

Avec les deux événements précédents, on arrive bien à s'arrêter sur le code que nous voulons. À la suite de ceci, on modifie le code pour essayer de rentrer sans se connecter. Le résultat c'est qu'on a réussi à rentrer dans le "Dashboard Admin" avec les formations chargées. Les formations publiées et ceux qui ne sont pas encore publiés comme vous pouvez le voir sur la figure ci-dessous. Malgré le fait qu'on a réussi à charger la vue Dashboard Admin, les clics sur les formations ne font absolument rien, aucune erreur n'est affichée côté client. De même si on tente de recharger la page, le serveur ne répond plus. Si on regarde le côté serveur, on voit qu'il a crashé, car il a essayé de charger un token à partir d'un user. Et comme on ne s'est jamais identifié, on n'a donc pas de token. Les détails d'étape par étape sont décrits en annexe "Event Listener Breakpoints - 1".

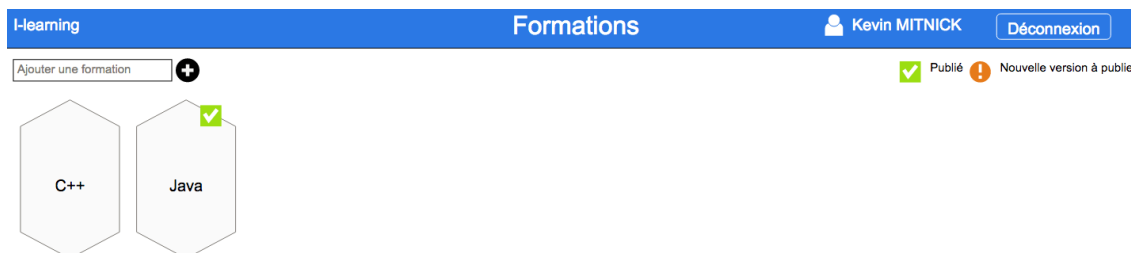


Figure II.8: Dashboard Admin

Avec la tentative de faire passer pour administrateur en modifiant le comportement numéro 1 (Figure SD Client), ne nous a pas rapporté de bonnes réponses, car on a fait crashé le serveur avec le code modification. Ce que nous allons faire maintenant c'est essayer de se modifier le comportement numéro 2. C'est à dire que nous allons connecter avec un vrai utilisateur non admin, et nous allons faire en sorte de modifier le code pour se connecter en tant qu'administrateur.

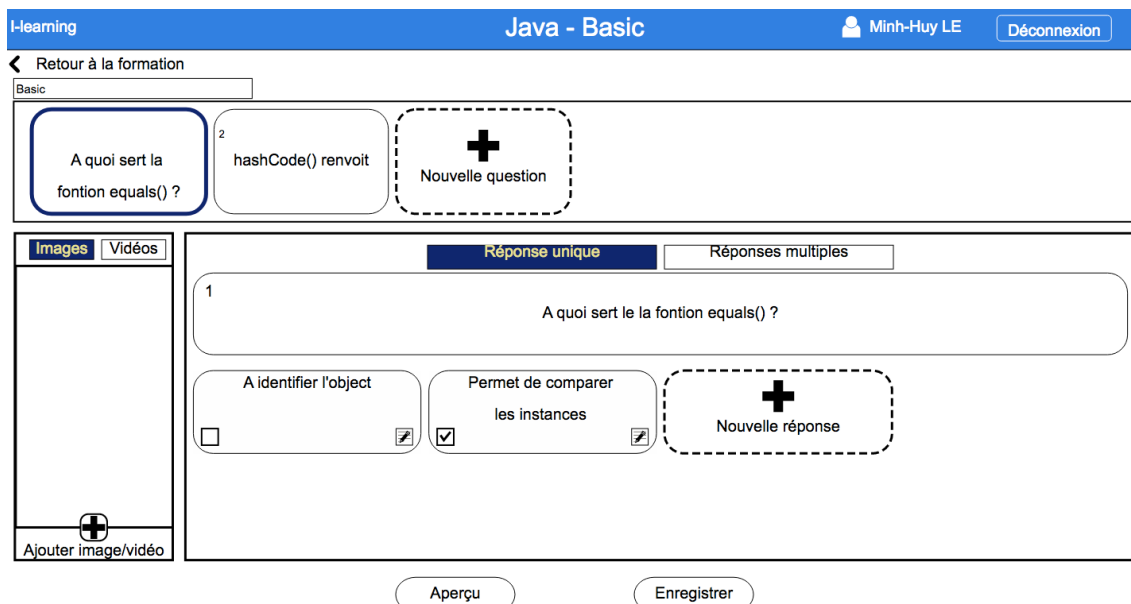


Figure II.9: Quiz Vue Admin

Résultat après la modification du code, on arrive bien à se connecter sur la vue Dashboard Admin avec les formations chargées. Contrairement à l'étape précédant cette fois-ci, on peut cliquer pour accéder à une formation et bien évidemment voir les bonnes réponses. On n'a

également pas de problème avec le token car on s'est bien authentifié, les étapes détaillées sont expliquées en annexe "Event Listener Breakpoints - 2". On peut également modifier le code pour faire en sorte d'afficher que les bonnes réponses.

4. Objet modification

Il y a également d'une autre manière de tricher avec le navigateur, c'est de modifier les données via la console du navigateur. Il suffit de chercher dans la variable Windows ou si on est passants on pourrait chercher les noms des variables en lisant le code.

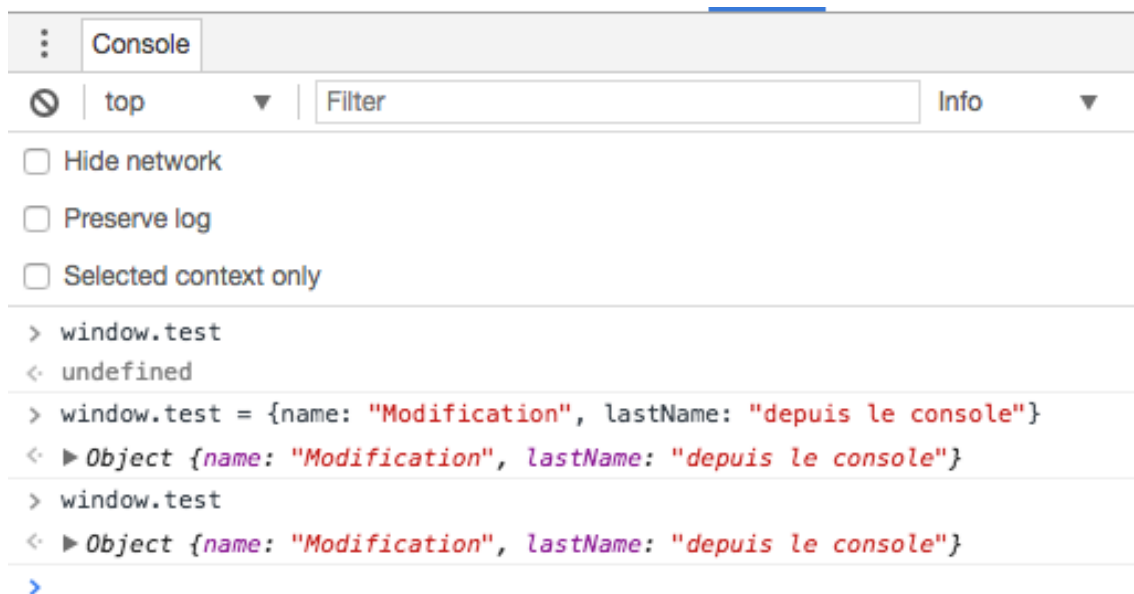


Figure II.10: Modification depuis le console

Si le pirate connaît bien la forme de donnée du jeu, il pourra modifier et envoyer comme données réelles sur le serveur.

5. Cross Site Scripting (XSS)

Le [Cross-Site Scripting \(XSS\)](#) est une attaque qui a pour but d'injecter du code malveillant pour que l'utilisateur exécute depuis son navigateur. Il existe deux sortes de types attaques en XSS, le réfléchi et le stocké. On appelle l'attaque réfléchi car il n'est pas stocké sur le serveur ou dans un fichier. La victime reçoit cette attaque via un mail qui utilise les liens contenant le script injecté (en paramètre de URL). Lorsque la victime clique sur ce lien, le code malveillant qui est non filtré par le serveur va être exécuté par le navigateur. L'attaque stockée consiste à injecter le code malicieux sur le serveur, par exemple dans un commentaire qui sera visible pour tout monde. Lorsque l'utilisateur charge sa page, ceci va exécuter le code qui peut effectuer des manoeuvres qui peuvent envoyer les cookies de l'utilisateur actuels. Ou encore, implémenter un "Keylogging" pour enregistrer ce que tape l'utilisateur.

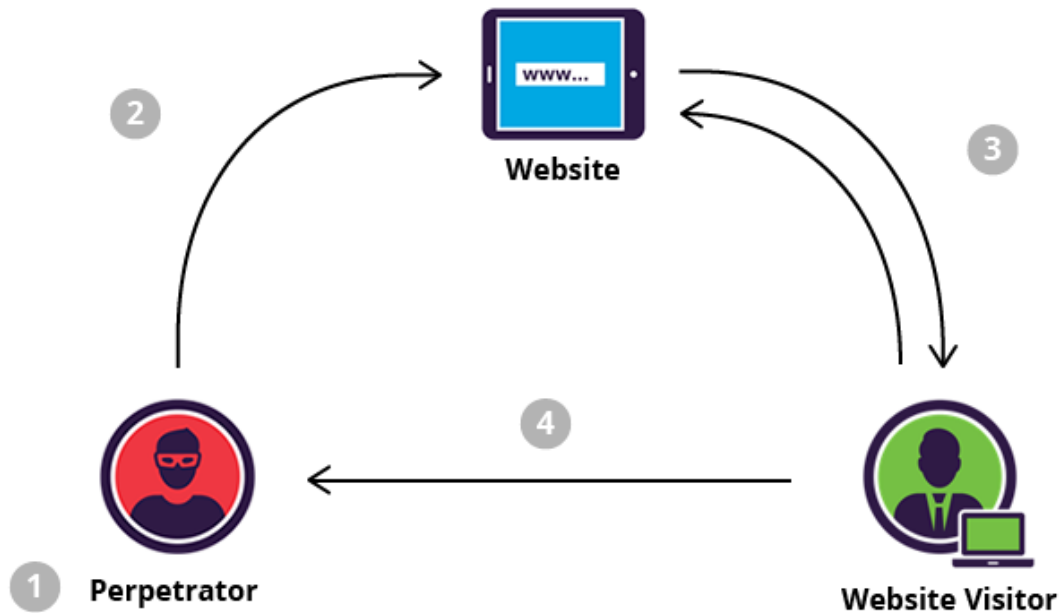


Figure II.11: Attaque XSS

1. Le pirate va chercher une faille dans le site, s'il peut injecter des données via [URL](#) paramètre ou encore des éléments [Hypertext Markup Language \(HTML\)](#). Il peut combiner avec l'attaque précédant "Code modification" pour rentrer avec les droits admin.
2. L'attaquant créer une formation tout en injectant son script afin de pouvoir voler les cookies ou les mots de passe des utilisateurs.
3. L'utilisateur se connecte, et charge la formation avec le code injecté.
4. Sans se rendre compte que l'utilisateur envoie des données confidentielles à son attaquant. Le pirate peut se connecter par le biais de la session ou avec les informations saisies au clavier.

Les attaques peuvent être écrites en n'importe quel langage, tant qu'il est exposé à [XSS](#) et compatible avec le navigateur [5, 6].

B. Attaque coté serveur BDD

1. [NoSQL injection](#)

La base de données utilisée dans le projet est le MongoDB, elle nous permet de stocker un document, récupérer via une seule clé et c'est un serveur [Not Only SQL \(NoSQL\)](#). Le terme [NoSQL](#), désigne les bases de données qui ne sont pas fondées sur l'architecture classique des bases de données relationnelles. Grâce à ceci, on n'est pas obligé d'avoir une structure similaire et qui nous évite de faire des jointures partout les tables. Par exemple on peut mettre dans la [Base de données \(BDD\)](#) un quiz avec trois questions qui et un autre avec dix questions. Il permet de répondre également à la cohérence, la haute disponibilité et la tolérance au partitionnement. Mais d'après le théorème de [Consistency Availability Partition tolerance \(CAP\)](#), les serveurs [NoSQL](#) ne peuvent avoirs seulement deux sur trois des caractéristiques [16].

Le No SQL, n'est pas signe de non injection, les serveurs tels que mongoDB utilisé derrière NodeJs n'assure pas la protection contre les attaques. Le mongoDB est dans le top 5 des serveurs les plus populaires, qui stocke ces données en [JavaScript Object Notation \(JSON\)](#). Le [JSON](#) est très utilisé entre les échanges de données avec les serveurs [17].

Imaginons pour se connecter, on a besoin d'un email et un mot de passe sous cette forme. Avec ces données correctes ci dessous lorsqu'on soumet on arrive bien à se connecter.

```
{
  "email" : "pentesterkunal@live.com",
  "password" : "scotch.io"
}
```

Figure II.12: Forme de données correctes en JSON

Maintenant si on va essayer de contourner le serveur afin de se logger sans à mettre une identification valide.

```
{
  "email" : {"$gt":""},
  "password" : {"$gt":""}
}
```

Figure II.13: Forme de données exploitées en JSON

En mettant ces données, on arrive à se connecter sans à avoir à mettre un vrai email. L'opération \$gt permet de dire a MongoDB de donner ce qui plus grand que des caractères vides. Ce qui retourne "true" et qui laisse l'attaquant à se logger sur le système [18, 19]. Le terme \$gt n'est pas la seule façon d'injecter, si on recherche plus en profondeur sur le langage, on peut voir d'autres par exemple \$ne qui signifie différent. Pour pallier à ce problème on peut convertir tous les valeurs entrants en une chaîne de caractères, pour que mongo ne l'exécute pas comme du code. Malgré le fait qu'on a mis les valeurs en string, le risque d'injection est encore là, avec l'opérateur \$where qui est très dangereux car elle permet les chaines de caractères à être exécuté. De plus, elle n'est pas performance car elle ne garde pas les indexes, le scope n'est pas accessible (en Javascript). Et de plus avec la version 2.1 de Node.js Mongo, l'opération bug si on lui donne en argument une fonction, ceci retourne tout [20].

III Protection

A. Protection globale

1. La mise à jour

Rester bien à jour pour bénéficier des derniers patches afin de protéger des attaques sur votre serveur/applications. On en découvre tout les jours des vulnérabilités, qui sont exploitées à des fins malicieuses.

2. Identifier les vulnérabilité des composants

Identifier tous les composants utilisés librairies, framework et composants. Éviter de les exposer, ne déployer pas les composants/librairies que vous n'utilisez pas. Il faut obfusquer les liens ou les chemins avec des mots clés qui permettraient d'identifier la librairie, on ne doit pas divulguer les informations sur [Operating System \(OS\)](#), le serveur, la [BDD](#), tous les maillons qui pourrai nuire à l'application. Il faut faire en sorte de rendre plus difficile les attaques, en cachant/supprimant tout informations qui pourrait identifier la librairie ou la technologie, c'est ce qu'on appelle la sécurité par l'obscurité. Il faut maintenir à jour les composants utilisés, englober les composants qui sont identifiés à risque [7].

3. Mauvaise configuration

La mauvaise configuration peut causer parfois de sérieux problèmes. Des paramètres par défaut qui ne sont pas changés par exemple les logins et les mots de passe de l'administrateur. Ou encore des applications qui sont installées avec le serveur et dans lequel on a oublié d'enlever les configurations par défauts [8].

4. Les guides - Bonnes pratiques

Il existe des guides de bonnes pratiques qui permettent d'éviter des erreurs débutants comme [Owasp.org](#) qui décrit quelles sont les recommandations à appliquer. D'autres sites qui permettent de rester à jour des failles découvertes [thehackernews.com](#). Il faut faire également en sorte l'application n'accepte que les mots de passe soient compliqués et assez long, l'OWASP conseille de fixer à 8 caractères minimum afin de minimiser les attaques. Il faudrait les composer de lettres majuscules, minuscules avec des chiffres et caractères spéciaux, c'est de cette façon qu'on ralentira la brute force. De même, il faut éviter de renvoyer toutes les erreurs en clair à l'utilisateur. Car si l'utilisateur connaît l'erreur, il gagnera du temps pour effectuer son attaque, alors que le but des protections et des préventions mises en place c'est pour le ralentir. Il faut également mettre en place des systèmes qui permettent de bannir l'utilisateur temporairement ou définitivement lors des tentatives d'accès à l'authentification ou encore les

autres web services. Il faut limiter les méthodes GET avec des paramètres, et il faut traiter les données sortant pour se prémunir des attaques [XSS](#).

B. Protection coté client

1. Offuscation

Offuscation est une méthode qui utilise la minification, qui permet de réduire tout caractère inutile comme les commentaires, les espaces et les indentations. La minification permet de rendre le code moins lisible et de plus elle augmente la performance, car on réduit le temps de chargement vu que la taille du fichier est réduit mais rend également plus dure à déboguer. L'offuscation utilise aussi le renommage des fonctions et des variables, ceux qui rendent le code incompréhensible. Il faut savoir que sur le côté client, s'il y a des failles on ne fait que ralentir l'attaquant. Il prendra du temps à comprendre, mais il pourra toujours attaquer [9].

Le problème est que si on met en place l'offuscation, c'est lorsqu'on souhaite déboguer qu'on se retrouve avec un code totalement incompréhensible. Il faut mettre en place une intégration seulement en production (lors du push sur les dépôts), comme ça lorsque qu'on est en développement on se retrouve pas à déboguer un code offusqué. On peut trouver plusieurs services d'après les recherches qui permettent d'offusquer les codes JavaScript et qui permet de geler le debug côté front, comme JScrambler ou encore javascript-obfuscator qui sont compatibles avec NodeJS. Bien évidemment, l'offuscation a une répercussion auprès de la performance, comme la taille du code peut être augmenté et certaines options qui peut ralentir un peu plus.

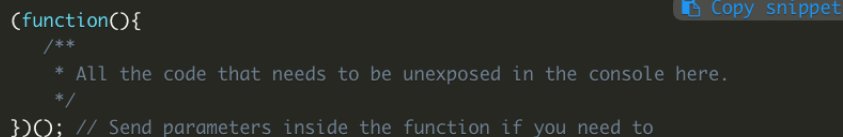
2. Protection contre les modifications

Comme on le sait, on ne peut faire confiance au client, car les utilisateurs malhonnêtes peuvent modifier le code source du côté client. Pour résoudre plus ou moins ce problème, on peut mettre en place le Mutation Observer. Elle nous permettra d'émettre des événements lorsque les [DOM](#) sont modifiés et donc réagit à l'action. On peut faire observer une modification des [DOM](#), qui déclenche un événement comme la déconnexion de l'utilisateur, encore supprimer la session utilisateur. Ou encore on peut tout effacer les objets [SVG](#), détacher tous les fichiers pour que le pirate puisse faire quoi ce soit. Il faut faire également attention à le désactiver lorsque ce sont nos actions qui sont la cause des modifications [10, 11].

Pour ce qui est des objets JavaScript, il y a plusieurs façons de rendre les objets immutables depuis la console du navigateur :

- `Object.freeze` : cette méthode permet de prémunir d'ajout de nouvelles propriétés. Ou encore d'effacement, et modification des propriétés. Problématique de cette méthode, est qu'elle applique seulement au parent, si on veut rendre l'objet enfant immutable il faut répéter l'action et de la même façon pour l'objet enfant de l'enfant.
- `Object.seal` : cette méthode est presque identique que celui de `Object.freeze`, sauf qu'on peut réaliser des modifications des propriétés déjà existants.
- `Object.preventExtensions` : comme son nom indique, cette méthode empêche seulement d'ajout des nouvelles propriétés.

Il faut bien évidemment choisir la bonne méthode selon le contexte et les besoins. Même si on a ces méthodes, on ne peut sceller tous les objets. Les objets qui ne sont pas scellés, sont accessibles depuis la console et donc sont modifiables. Pour éviter qu'il soit accessible depuis la console, il faut éviter les variables globales et réaliser des fonctions anonymes.



```
(function(){  
    /**  
     * All the code that needs to be unexposed in the console here.  
     */  
})(); // Send parameters inside the function if you need to
```

Figure III.1: Fonction anonyme

Malgré toutes ces protections, le code JavaScript est du côté du client, donc toujours accessible, et ça n'arrêtera pas le pirate. Cette méthode marchera peut être la première fois, mais les fois suivant ne fonctionnera plus avec. Ces objets restent modifiables, il suffit de le faire avant que les objets ne soient scellés ou par la modification d'une ou plusieurs parties du code [12].

C. Protection coté serveur web

1. Contrôle coté serveur

On a pu voir dans la partie les attaques, certaines sont réalisable car les validations sont réaliser par le client. De ce fait les données qu'on reçoit n'importe quoi, ce qui peuvent provoquer des graves dommages. Dans les règles de sécurités de bases, il ne faut jamais faire confiance au client comme on a pu voir la partie "code modification". Le client peut toujours modifier, ou désactiver les protections coté web, il faut donc s'assurer de filtrer ce qui vient du client afin d'assurer le bon fonctionnement de l'application.

Il faut voir la différence entre la validation des données coté client et côté serveur. Le contrôle coté client n'est pas fait pour sécuriser l'application, mais de filtrer un grand nombre de requêtes qui ne sont pas avec le bon format ou encore avec des données correctes. C'est une question d'optimiser les performances du serveurs, mais également une question d'esthétique. Il vaut mieux un message qui apparait pour nous prévenir que les données sont fausses, qu'une requête envoyé au serveur qui va la rejeter. C'est donc le serveur qui a la responsabilité de vérifier les formes de données, si elles sont correctes et de parser les données. Parser et contrôler les données permet de se protéger contre les injections qui peut nuire au serveur et à la BDD qui sont volontaires ou involontaires [15].

Bilan

Dans ce mémoire, nous avons étudié plusieurs types d'attaques pouvant être menées sur le projet d'I-Learning que mène VISEO afin de former ses collaborateurs/stagiaires. Cependant le panel de cyberattaques démontré n'est pas exhaustif, il existe une multitude d'autres attaques que nous n'avons pas pu aborder par manque évident de temps. Malgré les failles qui sont découvertes, la plupart des applications ne sont pas du tout sécurisées et beaucoup de développeurs n'y prêtent pas attention. Par exemple le contrôle des données qui sont effectués seulement coté front ou encore les injections sur les [BDD](#) relationnelles ou non relationnelles.

Nous avons pu aborder également la partie les protections contre les attaques, qui est concentrée sur la problématique "Comment éviter la triche sur un serious game ?". On a pu apercevoir que pour empêcher les attaques il faut tout d'abord étudier l'application. Identifier tous les maillons de la chaîne afin d'analyser les risques à tout niveau, dans quel langage est développée l'application, quel [BDD](#) intégrer, sous quel [OS](#) est l'application, etc. Il faut donc sensibiliser les développeurs afin qu'ils se posent des questions et mettent des mesures préventives pour minimiser les risques. Pour assurer une bonne sécurité, il vaut mieux combiner un ensemble de protections qu'appliquer une seule et surtout de rester vigilant par rapport aux failles. L'état du projet actuel ne présente très peu de sécurité, ce qui élargit les possibilités d'attaques sur le projet. Mais celui-ci n'est pas encore en production, on n'a pas pu évaluer l'impact de la sécurité ni disposer de données réelles afin d'analyser quelconque triche. Il ne faut pas s'attendre à ce que le projet soit attaqué pour mettre en place des outils de sécurité, et surtout bien avant la fin du développement. Cela peut représenter beaucoup de travaux, mais également un coût supplémentaire sur le projet.

On pourrait aller plus loin en mettant en pratique les protections nécessaires sur le projet, et tenter à nouveau de tricher. Si la triche est possible : comparer le temps nécessaire pour attaquer le projet avant et après. Trouver des moyens de détecter les tricheries afin de bannir ou bloquer l'utilisateur jusqu'à ce que l'administrateur vérifie les faits pour prendre une décision finale. Mais malheureusement, par manque de temps, les protections n'ont pas été implémentées. De plus le projet n'a pas cessé d'évoluer avec les patches, les nouvelles fonctionnalités pour arriver finalement à une refonte totale de l'application, principalement pour des problèmes d'architecture.

Biographie

- [1] *serious-game.fr*
<http://www.serious-game.fr/marche-serious-games-continue-bonne-croissance-2017/>
- [2] *infodsi.com*
<http://www.infodsi.com/articles/166234/point-evolution-serious-games-learning-mobile-learning-gamification-moocs-solutions-logiciels-rh-horizon-2016-2020.html>
- [3] *academia.edu*
http://www.academia.edu/5981159/Pertinence_et_efficacit  _des_serious_games._Enqu  te_de_r  ception_sur_neuf_serious_games
- [4] *commentcamarche.net*
<http://www.commentcamarche.net/contents/47-piratage-et-attaques-informatiques>
- [5] *excess-xss.com*
<https://excess-xss.com/#xss-overview>
- [6] *openclassrooms.com*
<https://openclassrooms.com/courses/protegez-vous-efficacement-contre-les-faillles-web/la-faille-xss-1>
- [7] *kemptechnologies.com*
<https://kemptechnologies.com/blog/owasp-top-ten-series-using-components-with-known-vulnerabilities/>
- [8] *owasp.org*
https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration
- [9] *petitchevalroux.net*
<http://dev.petitchevalroux.net/javascript/obfusquer-minifier-compresseur-javascript-javascript-308.html>
- [10] *hacks.mozilla.org*
<https://hacks.mozilla.org/2012/05/dom-mutationobserver-reacting-to-dom-changes-without-killing-browser-performance/>
- [11] *jsbin.com*
<http://jsbin.com/codopayivi/edit?html,css,output>
- [12] *ourcodeworld.com*
<http://ourcodeworld.com/articles/read/167/how-to-prevent-modification-of-an-object-in-javascript-and-prevent-them-from-being-accessible-in-the-console>

- [13] *ca.com*
<https://www.ca.com/content/dam/ca/fr/files/ebook/session-hijacking-a-new-method-of-prevention.pdf>
- [14] *developpez.com*
<http://cyberzoide.developpez.com/securite/session/#LD>
- [15] *net-informations.com*
<http://net-informations.com/faq/asp/validation.htm>
- [16] *neoxia.com*
<http://blog.neoxia.com/nosql-5-minutes-pour-comprendre/>
- [17] *advens.fr*
<https://www.advens.fr/ressources/blog/nosql-no-security>
- [18] *scotch.io*
<https://scotch.io/@kunalrehan/mongodb-injection-in-nodejs>
- [19] *websecurify.com*
<http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html>
- [20] *zanon.io*
<https://zanon.io/posts/nosql-injection-in-mongodb>

Glossaires

BDD Base de données. [15](#), [17](#), [19](#), [20](#)

CAP Consistency Availability Partition tolerance. [15](#)

DOM Document Object Model. [7](#), [11](#), [18](#)

HTML Hypertext Markup Language. [15](#)

HTTP Hypertext Transfer Protocol. [9](#)

JSON JavaScript Object Notation. [16](#)

M2 Master 2. [3](#), [6](#)

MIAGE Méthodes Informatiques Appliquées à la Gestion des Entreprises. [3](#), [6](#)

NoSQL Not Only SQL. [15](#)

OS Operating System. [17](#), [20](#)

SVG Scalable Vector Graphics. [7](#), [18](#)

URL Uniform Resource Locator. [7](#), [14](#), [15](#)

XSS Cross-Site Scripting. [14](#), [15](#), [18](#)

Annexes

A. Debugger avec chrome

La fenêtre ci-dessus est affichée lorsque vous faites inspecter sur chrome, on peut voir sur le top-bar la navigation sur la vue "Source". Avec lequel il y a trois blocs, celui de gauche un menu de fichiers, le central le fichier sélectionner, à droite le menu avec le débbugger et le scope qui permet d'observer les variables.

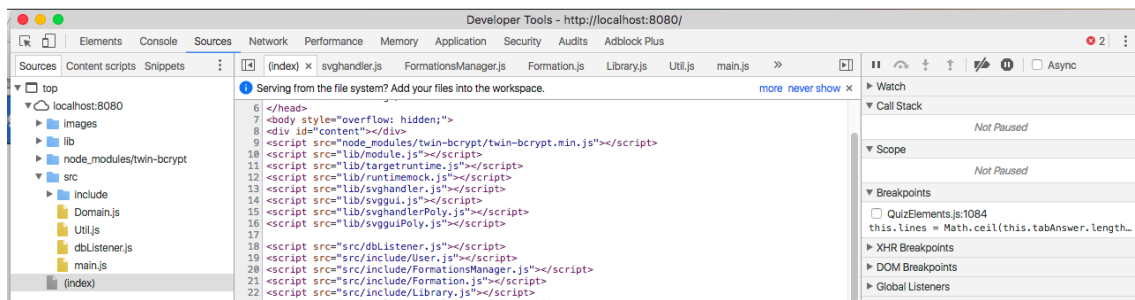


Figure 2: Developer Tools

On peut voir un le code du fichier sélectionner, avec quelque fichier js qui sont inclus dans la page.

```
<script src="src/Util.js"></script>
<script src="src/Domain.js"></script>
<script src="src/main.js"></script>
<script src="lib/enhancer.js"></script>
<script>
  const
    enhance = exports.Enhance(),
    targetRuntime = exports.targetRuntime(),
    SVG = exports.SVG,
    dbListener = new DbListener(true, false);
  var url = document.URL;
  var query = url.split('?')[1];
  var ID;
  var param;
  if (query) {
    param = query.split('=')[0];
  }
  var redirect = false;
  if (param == 'ID') {
    redirect = true;
    ID = query.split('=')[1];
  }
  main(SVG(targetRuntime), targetRuntime, dbListener, false, {redirect: redirect, ID: ID});
</script>
</body>
</html>
```

Figure 3: Observation du fichier index.html

Si on utilise le bloc de gestion de fichiers de gauche, on peut voir les fichiers qui sont chargés. On peut évidemment lire tous les codes de tous les fichiers, mais cela pourrait nous prendre énormément de temps.

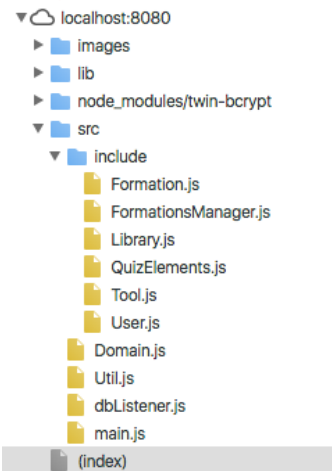


Figure 4: Block gestion fichier

On pourrait également chercher plus intelligemment dans le fichier les mots clés, comme ici dans ces fichiers nous avons des classes. On va essayer de repérer les noms des classes pour voir si on peut trouver quelque chose qui va nous être utile très rapidement, et on voit il y a deux noms de classes qui pourrait nous intéresser "Answer" et QuizManaverVue.

```

/////////MODEL////////////////////////////////////
/**
 * Réponse à un quiz. Cette réponse peut être correcte ou non. Une explication
 * @class
 */
class Answer {
  /**
   * @class
   */
  class QuizManagerVue extends Vue {
    /**
     * construit un quiz associé à une formation
     * @constructs
     * @param quiz - objet qui va contenir toutes les informations du quiz crée
     * @param formation - formation qui va contenir le quiz
     */
    // ...
  }
}

/**
 * @class
 */
class QuestionVue extends Vue {

```

Figure 5: Classes en javascript

Pour utiliser le debugger côté client, il faut se placer dans le bloc central sur la ligne que vous souhaitez arrêter et cliquer sur le bar de gauche du bloc. Lors de l'exécution du code si elle passe par votre ligne, l'exécution va se mettre en pause là où il y a le break point. Et vous pouvez ensuite exécuter ligne par ligne, et suivre toutes les modifications de l'état des objets.

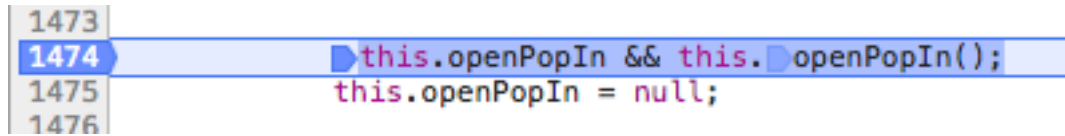


Figure 6: BreakPoint

On peut mettre des conditions d'arrêt sur le break point, lorsqu'on cherche à s'arrêter sur une valeur ou à un état particulier de l'objet. Pour mettre la condition, il suffit de faire un clic droit sur le breakpoint et l'éditer comme le figure ci-dessous.

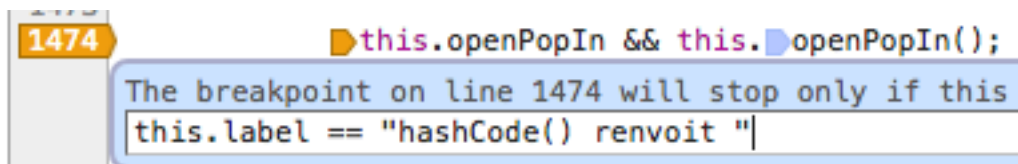


Figure 7: BreakPoint avec condition

B. Snapshot memory avec chrome

Il y a plusieurs types de photo mémoire, il y a celui qui prend une photo instantanée de la mémoire. Il y a celui qui enregistre tout, et le dernier qui enregistre par rapport à une chronologie.

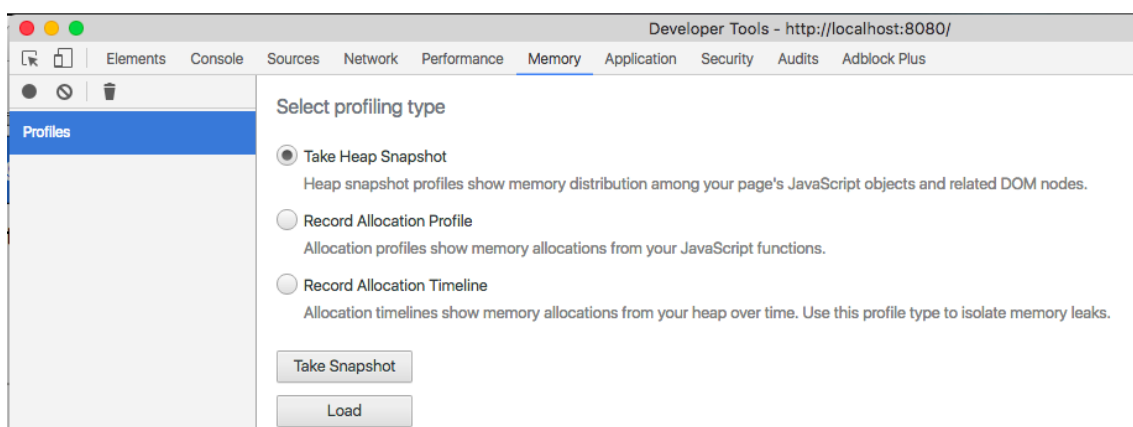


Figure 8: Developer Tools Memory

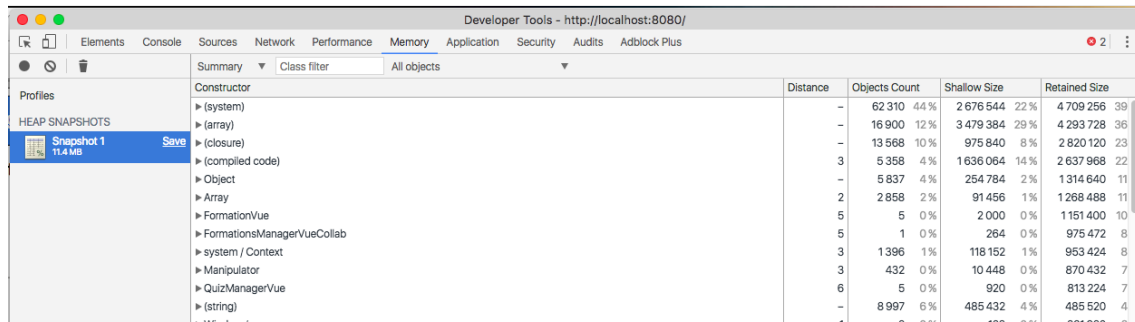


Figure 9: Snapshot memory

On peut voir un aperçu de la mémoire sur la figure ci-dessus, avec des précisions comme sa taille, le nombre de fois instancié, le nom de l'objet. On peut également essayer de fouiller dans tout les objets photographiés pour trouver ce qu'on cherche. Sinon on peut utiliser le filtre comme la figure ci-dessus.

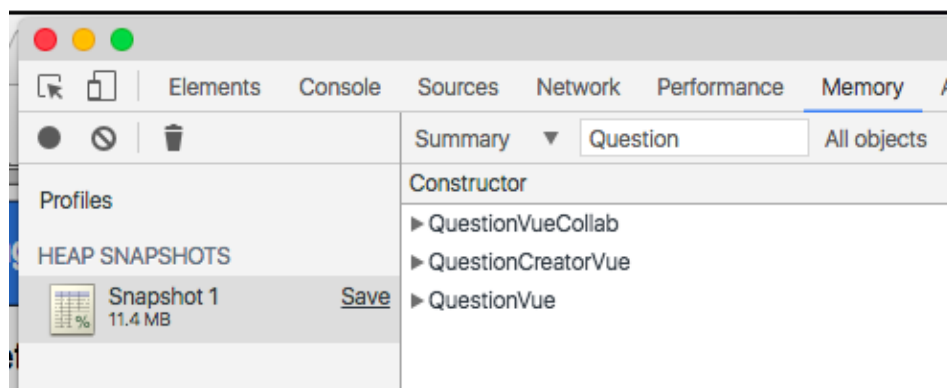


Figure 10: Snapshot memory filtrer

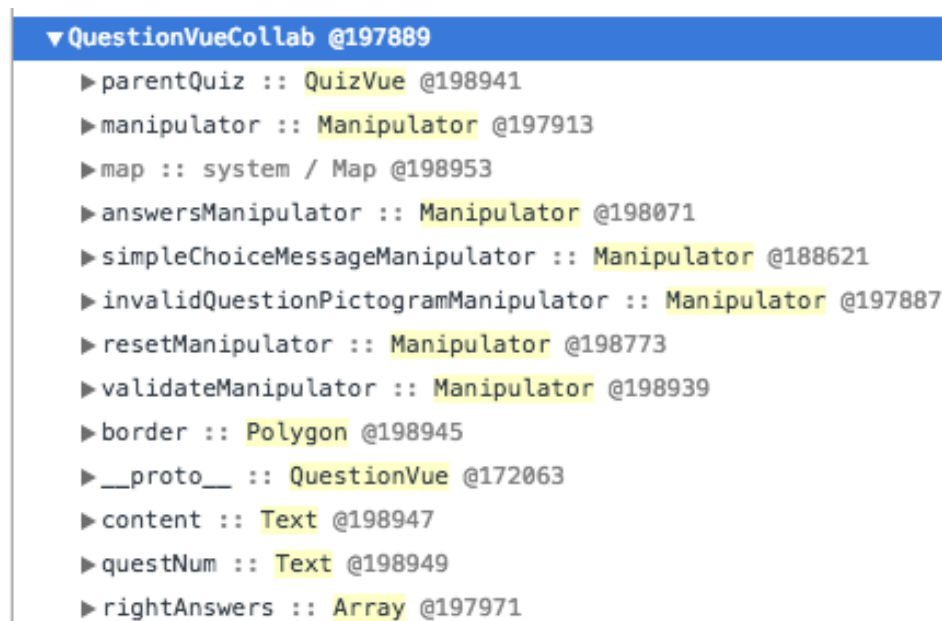


Figure 11: Aperçu de l'objet depuis Snapshot

C. Event Listener Breakpoints

1. Modification étape 1

Lors de la modification de l'étape 1, nous avons essayé de repérer avec event "mouse" qui ne nous a malheureusement pas amenés sur le code source que nous souhaitons.

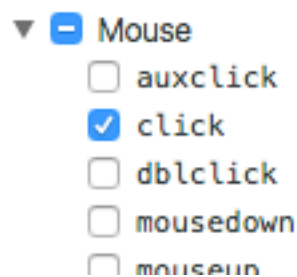


Figure 12: Event mouse

À la suite de ceci, nous avons étudié le comportement de l'étape 1 pour essayer de trouver d'autres événements qui pourraient nous aider à trouver le code que nous souhaitons avoir.

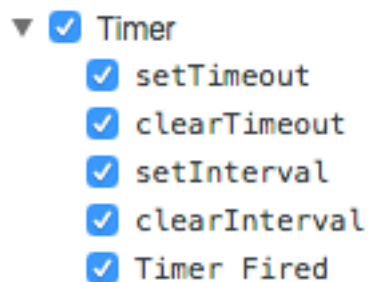


Figure 13: Event Timer

Lorsqu'on essaye d'utiliser l'événement timer, on s'arrête exactement dans la fonction que nous souhaitons modifier "la connexion". Et si on observe plus attentivement, il nous pointe sur la ligne où ç'a été défini l'événement "timeout".

```

connexionButtonHandler() {
  this.mailAddressField.input.hideControl() && this.passwordField.input.hideControl();
  let emptyAreas = this.tabForm.filter(field => field.input.textMessage === '');
  emptyAreas.forEach(emptyArea => {
    emptyArea.input.color(ERROR_INPUT);
  });

  if (emptyAreas.length > 0) {
    let message = new svg.Text(EMPTY_FIELD_ERROR)
      .dimension(INPUT_WIDTH, INPUT_HEIGHT)
      .position(0, -MARGIN - BUTTON_HEIGHT)
      .color(myColors.red)
      .font(FONT, FONT_SIZE_INPUT)
      .mark("msgFieldError");
    this.connexionButtonManipulator.set(1, message);

    svg.timeout(() => {
      this.connexionButtonManipulator.unset(1);
      emptyAreas.forEach(emptyArea => {
        emptyArea.input.color(COLORS);
      });
    }, 5000);
  } else {
    Server.connect(this.mailAddressField.input.textMessage, this.passwordField.input.textMessage,
      this.model.correct).then(data => {
        data = data && JSON.parse(data);
        if (data.ack === 'OK') {
          drawing.username = `${data.user.firstName} ${data.user.lastName}`;
          data.user.admin ? globalVariables.domain.adminGUI() : globalVariables.domain.learningGUI();
          let user = data.user;
        }
      });
  }
}

```

Figure 14: Break point avec l'événement timer

Lors de l'étude du comportement de l'étape 1, on a découvert également l'événement "Keyboard" qui pourrait nous amener à le code chercher.

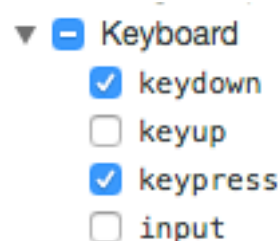


Figure 15: Event Keyboard

Avec l'événement keyboard, on s'arrête à un niveau au-dessus, on voit la gestion des touches claviers. Mais on voit également dans la gestion de la touche entrée, la fonction "connexionButtonHandler" qui est la même que celui nous a amené l'événement timer.

```

/**
 * handler global pour gérer les appuis sur les touches du clavier
 * @param event
 */
keyDownHandler(event) {
  if (event.keyCode === 9) { // TAB
    event.preventDefault();
    let index = this.tabForm.indexOf(this.focusedField);
    if (index !== -1) {
      event.shiftKey ? index-- : index++;
      if (index === this.tabForm.length) index = 0;
      if (index === -1) index = this.tabForm.length - 1;
      svg.event(this.tabForm[index].input.glass, "click");
      this.focusedField = this.tabForm[index];
    }
  } else if (event.keyCode === 13) { // Entrée
    event.preventDefault();
    this.focusedField && this.focusedField.input.hideControl();
    this.connexionButtonHandler();
  }
}
}

```

Figure 16: Break point avec l'événement keyboard

On modifie le code à en sort que lorsqu'on clique, aucun appel au serveur est envoyer, et on créer un utilisateur "Kevin MITNICK" en admin.

```

connexionButtonHandler() {
  this.mailAddressField.input.hideControl() && this.passwordField.input.hideControl();

  let data = {user:{firstName: "Kevin", lastName:"MITNICK", admin: "true"}}
  drawing.username = `${data.user.firstName} ${data.user.lastName}`;
  data.user.admin ? globalVariables.domain.adminGUI() : globalVariables.domain.adminGUI();
  let user = data.user;
  Server.getAllFormations().then(data => {
    let myFormations = JSON.parse(data).myCollection;
    globalVariables.formationManager = classContainer.createClass("FormationsManagerVue", myFormations);
    if (user && user.lastAction && user.lastAction.formation) {
      util.goDirectlyToLastAction(user.lastAction);
    } else {
      globalVariables.formationManager.display();
    }
  });
}
}

```

Figure 17: Code modifié pour l'étape 1

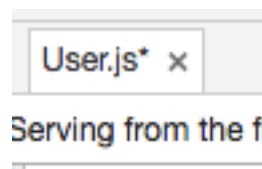


Figure 18: Modification fichier

Lorsqu'on modifie un fichier, on doit sauvegarder pour appliquer la modification et ne pas recharger la page. Sinon on risque de perd la modification, on peut voir également l'historique des modifications en faisant click droit puis "Local Modifications".

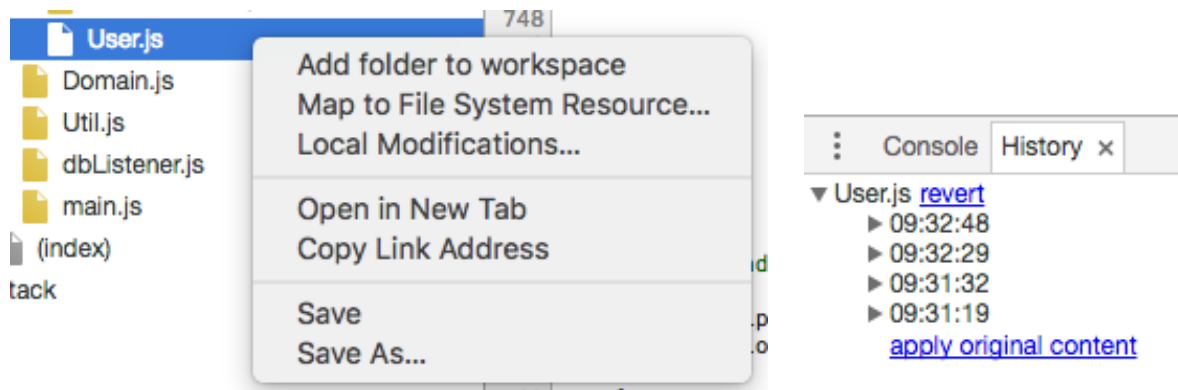


Figure 19: Historique des modifications du fichier

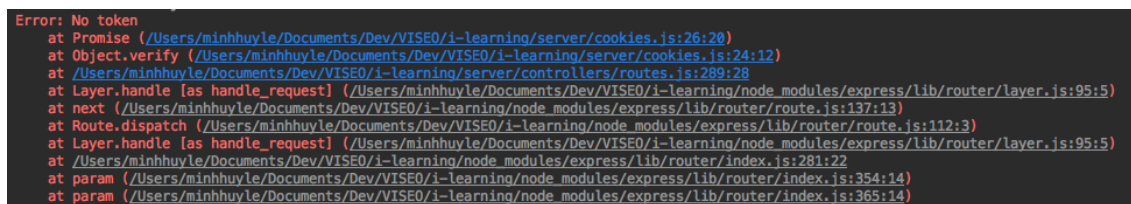


Figure 20: Erreur du serveur : No token

Après la modification du code de l'étape 1, on a fait planter le serveur. Car on a fait une requête sur le serveur, comme on ne s'est jamais authentifié, on n'a pas de token ce qui cause les dégâts cotés back-end.

2. Modification étape 2

Modification de l'étape 2, faire passer un utilisateur normal en utilisateur admin.

```
Server.connect(this.mailAddressField.input.textMessage, this.passwordField.input.textMessage,
  this.model.correct).then(data => {
    data = data && JSON.parse(data);
    if (data.ack === 'OK') {
      drawing.username = `${data.user.firstName} ${data.user.lastName}`;
      data.user.admin = true;
      data.user.admin ? globalVariables.domain.adminGUI() : globalVariables.domain.adminGUI();
      let user = data.user;
      Server.getAllFormations().then(data => {
```

Figure 21: Code modifier pour l'étape 2

Après la modification du code, on arrive à se connecter et parcourir tout en tant qu'admin.

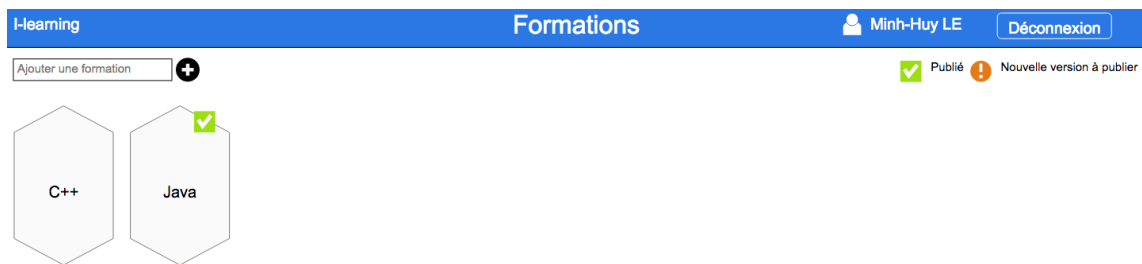


Figure 22: Vue dashboard admin

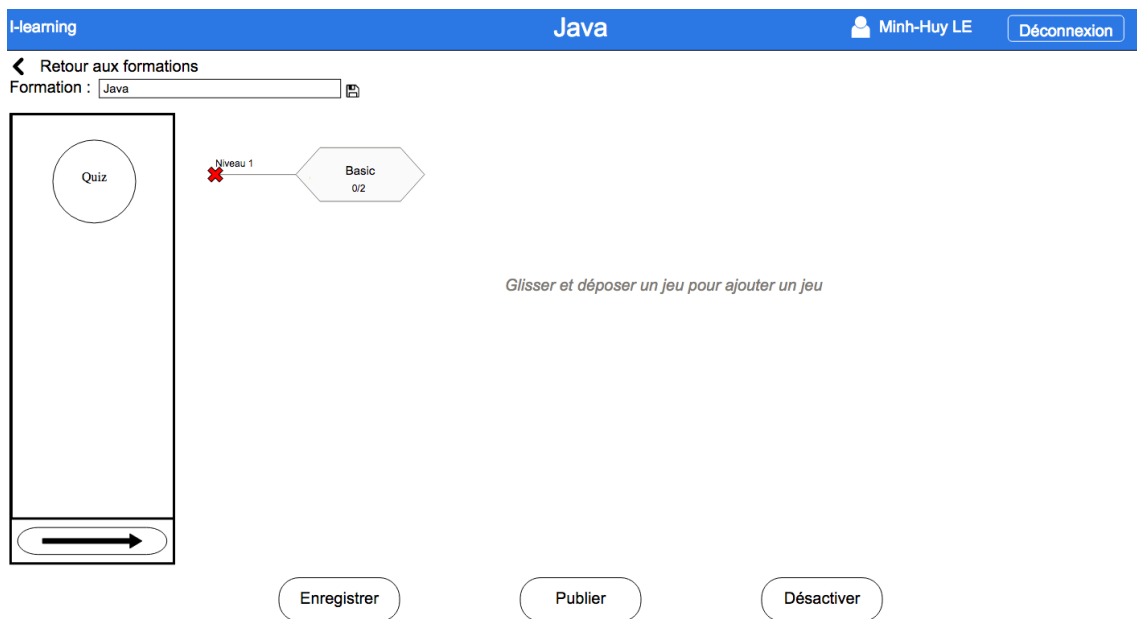


Figure 23: Vue formation admin

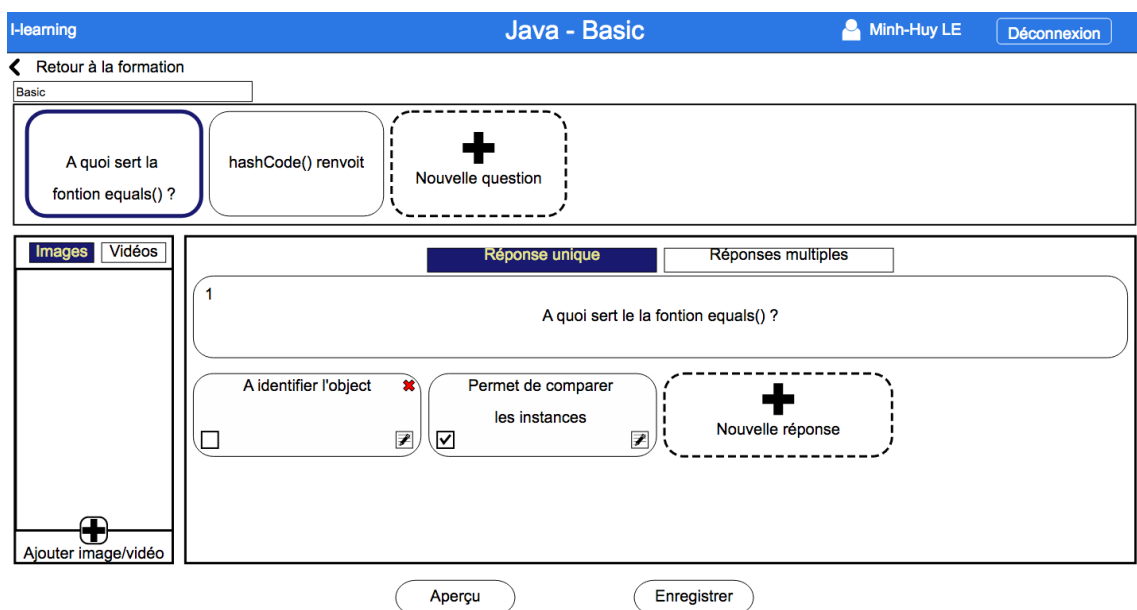


Figure 24: Vue création quiz admin

Lorsqu'on recharge la page, on perd la modification et on bascule donc en vue collaborateur (non admin).

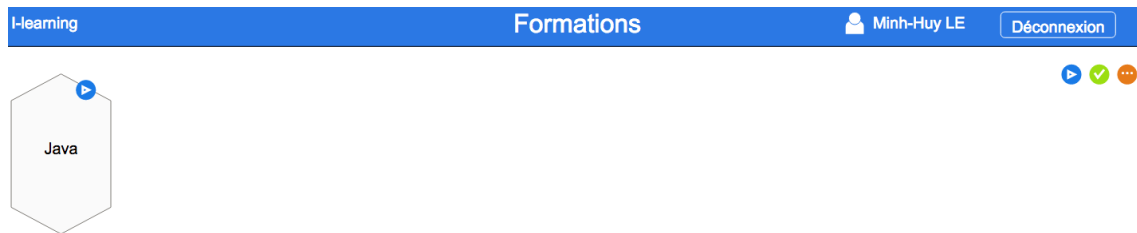


Figure 25: Vue dashboard collaborateur



Figure 26: Vue formation collaborateur

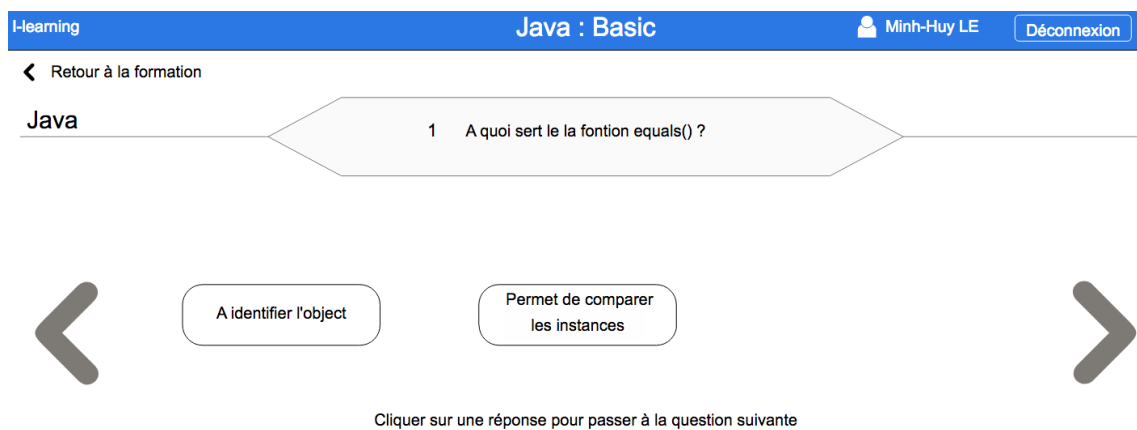


Figure 27: Vue quiz collaborateur