

Hubert Samson

BARONNIES

Dossier de Synthèse

Introduction	3
Le Projet	3
Baronnies	4
Cahier des charges	4
RGPD	6
Spécifications techniques	7
Liste globale d'outils	7
Node JS	8
Frameworks	9
Express	9
Colyseus	10
Arborescence du projet	11
Structure de l'application	12
Réalisation	13
Sécurité	13
Une base de donnée NoSQL	20
Les Routes	24
Le Lobby et les fonctionnalités sociales	25
Adapter "Baronie"	27
Mini-Goban	32
Ressources, tests & veille	33
Choix et constats personnels	33
Tests de jeu	34
Ressources Graphiques / Sonores	36
Recherche Anglophone	38
Passage en Production	39
Veille & maintenance	40
Conclusion	40
ANNEXES	41
ANNEXE 1 - Charte Graphique	42
ANNEXE 2 - Traduction	43
ANNEXE 3 - Données Personnelles	45

Introduction

Ma première expérience de codeur est le modding pour jeux vidéo. Comme beaucoup de francophones, je découvre vers 13 ans le site du Zéro, et me lance dans les cours en C/C++. Depuis, J'ai pu expérimenter plusieurs outils, langages, approches.

Avant d'entamer les démarches Pour acquérir de titre professionnel de Développeur Web, je n'avais pas pensé pouvoir transformer ce savoir en un travail professionnel rémunéré correctement. Voir cette application aboutir est donc pour moi très grisant.

Le Projet

Cette application est un portage pour navigateurs web & web mobile du jeu de société "Barony" édité par la société Matagot. Il n'a sous la forme actuelle aucune vocation commerciale. Ce jeu se destine à une utilisation au sein d'un cercle personnel, mais sa structure permet d'implémenter rapidement de nouvelles règles voire de nouveaux jeux.

Le site doit offrir la possibilité de s'inscrire, d'établir des liens sociaux avec d'autres joueurs, de consulter et créer des salles de jeux, de conserver scores et sauvegardes, et enfin évidemment de jouer. Le pari est le suivant: après une formation en PHP-SQL, développer une application Node Js, en Javascript donc, dans un système impliquant nombreuses nouvelles spécificités techniques.

La particularité du jeu en ligne sur navigateur dans une application similaire, c'est d'obliger l'alliance de requêtes HTTP avec une communication bilatérale pour le temps réel, ainsi que l'usage de base de donnée, de créations de pages statiques et d'animations. L'avantage d'un portage est d'économiser le temps de création et d'équilibrage des règles de jeu, au profit de l'application elle même.

Référentiel du titre professionnel

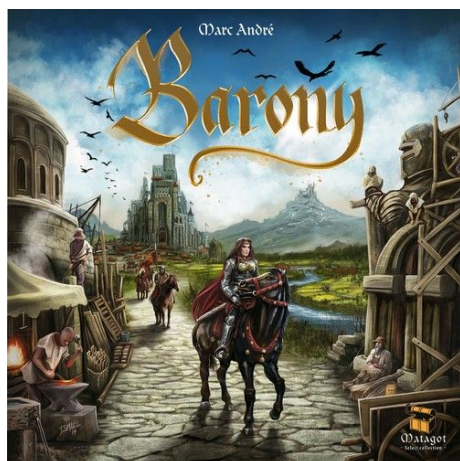
Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Maquetter une application
- Développer une interface utilisateur web dynamique
- Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile

Baronnies



Le jeu de plateau Baronnie est un jeu de stratégie-construction au tours par tours, où l'objectif est de se disputer le contrôle d'une région au travers des faveurs du suzerain contre jusqu'à trois autres Barons.

A mi-chemin entre une partie de Go et de Civilizations, le principe du jeu est de parvenir à anticiper au mieux les actions de l'adversaire. Les règles, nombreuses pour un jeu de société "casual", ainsi que l'absence de hasard, rendent les impératifs des joueurs assez prédictibles. Il n'y a donc pas lieu de négocier oralement avec son adversaire, et ne pas

l'avoir en face hôte peu au plaisir de jouer.

C'est ce raisonnement qui m'a poussé à choisir ce modèle de jeu pour mon application, la perte de contact "direct" du jeu de société étant compensée par les possibilités qu'offre le jeu en ligne.

Voici un bref aperçu du déroulement du jeu:

- Chaque joueur commence avec le même immuable nombre de pions
- Il doit choisir à chaque tour une seule action à effectuer, comme recruter, se mouvoir, construire un bâtiment, ou encore payer un tribut au Suzerain.
- Les joueurs gagnent de l'argent en construisant ou en attaquant leurs rivaux. Cet or est dépensé en titres de noblesses, et le premier joueur à atteindre le plus haut titre clos le jeu. C'est alors celui des joueurs qui possèdera le plus de points (titres, citées et or restant pris en compte) qui remportera la partie.

Cahier des charges

L'objectif est une solution de jeu multijoueurs, avec une interface sociale, un système de lobby, et une implémentation fonctionnelle de Baronnies. Les deux premiers points sont prévus comme indépendants du second: l'application doit être prête à accueillir d'autres jeux de type similaire (petits nombre de joueurs par salle). Enfin, le système de jeu de Baronnie doit être suffisamment souple pour implémenter/modifier rapidement les règles de jeux à l'avenir.

Interface Sociale
<ul style="list-style-type: none">- Enregistrement / connexion par couple mail/pseudo & mot de passe- Modification/Suppression des données personnelles- Possibilité de lier d'autres joueurs à son compte- Possibilité de gérer ces liens- Possibilité de consulter les autres profils

<ul style="list-style-type: none"> - Possibilité de leurs réserver des sièges dans une salle de jeu
Système de Lobby
<ul style="list-style-type: none"> - Interface pour lister/rejoindre les salles existantes en fonction du jeu - Possibilité de créer une salle en précisant les options offertes par le jeu - Les salles sont soit publiques, privées ou mixtes - Les salles publiques peuvent accueillir des joueurs anonymes - Les privées n'acceptent que les joueurs liés à l'hôte - Les mixtes peuvent se voir réserver des sièges nominatifs pour d'autres joueurs - Création d'un mini jeu de Go pour tester le caractère générique des méthodes ci-dessus
Jeu Baronnie
<ul style="list-style-type: none"> - Jeu 1 - 4 joueurs - Interface de jeu utilisant les éléments du DOM - Interface Mobile - Système de sauvegardes / scores pour les joueurs authentifiés - Adaptation de l'ensemble des éléments du plateau - Systèmes d'actions / règles / validateurs distincts
Sécurité
<ul style="list-style-type: none"> - Système de sessions persistantes en base de donnée - Système d'authentification par mot de passe - Système de rôles pour les administrateurs - Système de tokens JSON pour authentification sur Socket.io avec Colyseus - Protection contre la faille csrf
Hébergement
<ul style="list-style-type: none"> - Mise en place d'un serveur sous ubuntu-server sur un raspberry Pie - Solution Nginx + NodeJs + MongoDB server - Réservation d'un nom de domaine gratuit (Netlib.re) - Mise en place d'un certificat TLS gratuit (Let's encrypt) - Première mesures de protection contre les intrusions/attaques de force (Fail2Ban)

RGPD

Le Règlement Général sur la Protection des Données à "caractère Personnel" Européen, entré en vigueur en Mai 2018, est venu renforcer en France (notamment par des moyens financiers un peu plus coercitifs) l'encadrement déjà posé par la CNIL du traitement et du stockages des dites données.



Principes Généraux

- L'utilisateur doit être **informé clairement** de la **liste exhaustive** des données caractérisées comme personnelles le concernant traitées par le site
- La liste doit préciser ou mettre à disposition proche une **documentation** incluant tous les motifs, acteurs et natures des traitements de ces données
- Le **consentement** de l'utilisateur pour ce traitement est obligatoire et doit être établis par une **démarche active** de ce dernier
- L'utilisateur doit pouvoir à la demande récupérer ou supprimer ces données
- Enfin l'utilisateur doit être garanti que ses données ne serviront pas à des fins de **profilages** dont il ne serait pas informé ou qui seraient pas admises par la CNIL

Dans ce projet, seules ces données sont caractérisables comme personnelles au vu du droit français:

- 1) Les adresses Email des joueurs
- 2) Leurs liens intra-application
- 3) Leurs sauvegardes/Scores
- 4) Leurs historiques de connexions sur le serveur, dans le cadre de l'auto-hébergement

Mesures de mise en conformité

- Information à l'enregistrement, acceptée par une check-box, avec un lien vers une page de documentation
- Page de documentation quant aux natures et durées des stockages/traitements de données personnelles (cf Annexe n°)
- Mise à disposition d'outils simples de récupération/destruction des scores et sauvegardes de jeu

Spécifications techniques

Liste globale d'outils

Développement sur la machine

- Ubuntu 18.x (*Système d'exploitation*)
- VisualStudio Code (*IDE*)
- Node Js (*Plateforme Js*)
- Nodemon (*module node Js*)
- npm (*Gestionnaire de dépendances*)

Développement en local

- Raspberry-Pi 3
- Ubuntu 18.x Server (*Système d'exploitation*)
- OpenVPN (*tunnel vpn*) + OpenSSH (*tunnel ssh*) + SFTP (*tunnel de transmission de fichiers*)
- GitHub (*Stockage, suivi & export du code*)
- NGINX (*Server & reverse proxy*)
- UFW (*Pare-feu*)

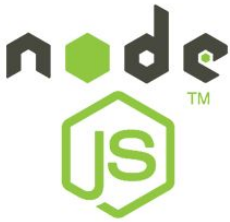
Tests en environnement de production

- Fail2ban (*contrôle du nombre de requêtes*)
- Netlib.re (*Non de domaine Gratuit*)
- LetsEncrypt (*Certificat pour protocole TLS Gratuit*)
- PM2 (*Gestionnaire de processus pour Node Js*)

Autres

- Draw.io (*Outil de schématisation*)
- Discord (*Plateforme de chat*)
- Gimp (*Bibliothèque de manipulation d'images*)

Node JS



“Node.js (Node) is an open source development platform for executing [JavaScript](#) code server-side.”

Cette plateforme permet d'exécuter du **JavaScript côté serveur** avec des bibliothèques utilisables aussi côté client.

Son modèle est **mono-thread et non-bloquant** (voir ci-dessous), ce qui lui permet de gérer des tâches asynchrone sur un modèle événementiel. Cela diffère des autres solutions web classiques type Apache-PHP. Ce modèle rend la plateforme plutôt adapté aux applications nécessitant une **connexion persistante du navigateur au serveur**, de type chatroom ou encore web push notification. Malgré cela, Node Js n'est pas La solution parfaite, et les possibilité qu'elle offre viennent en autres avec une charge de travail supplémentaire : L'application tourne sur un serveur HTTP dédié dont il faut coder les différentes couches. Cependant de nombreux modules et frameworks simplifient cette tâche.

Un système type Apache-PHP va traiter par thread (coeur physique ou virtuel alloué par le serveur pour traiter l'opération) une liste successive de tâches, dont chacune doit attendre l'exécution de la précédente sur une même liste.

Le système Node Js permet sur un seul thread d'exécuter de manière concurrente plusieurs événements, dont le traitement peut éventuellement être confié à un autre thread. C'est son aspect non bloquant: si une tâche prend plus de temps que celles qui la suivent, son exécution peut aboutir après ces autres tâches. Si cette exécution implique une autre tâche à sa suite directe, elle est appelée via un “call-back”, un appel imbriqué dans sa prédécessrice: ces ensembles de tâches chainées sont considérées comme asynchrones.

Enfin la structure de Node Js permet d'implémenter d'adapter facilement la puissance des systèmes de traitements de données en fonction des besoins en temps réel (scalability).

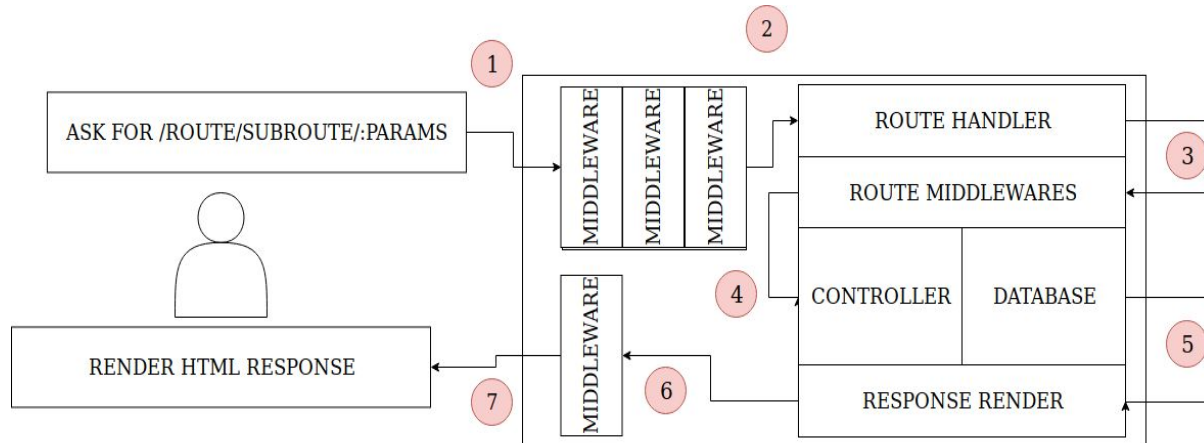
Sous licence libre (MIT), Node Js est fourni avec le gestionnaire de dépendances NPM.

Frameworks

Express

Express est un module Node Js multi-fonctions, mais ici il servira essentiellement de **Routeur**. Son cadre de travail permet de définir des routes simple, ainsi que les d'extraire, valider et transmettre les informations contenues dans les requêtes au travers de middlewares.

middlewares: méthodes préposées aux routes s'exécutant à chaque appel de ces dernières



- 1) Middlewares globaux (appelés sur chaque requête)
- 2) Routage de la requête
- 3) Middlewares de la route
- 4) appel éventuel aux contrôleurs
- 5) et à la base de donnée
- 6) construction de la réponse
- 7) Middlewares de réponse

exemple de middleware:

```
5  AuthStatus.isLoggedIn = (req, res, next) => {  
6  
7    if (req.isAuthenticated()) {  
8      return next();  
9    }  
10   res.redirect('/');  
11 }
```

Ce middleware très simple ouvre la requête pour s'assurer que le token qu'elle contient est valide, auquel cas la fonction `next()` est appelée (soit la fonction suivante dans la chaîne de middlewares). Sinon, le client est redirigé vers la racine du Site.

Son deuxième rôle sera de garder une trace des connexions en cours en base de données via son système de session. Chaque utilisateur reçoit un ID secret, qui doit correspondre à son pair en base de donnée, afin de faire persister ses informations

exemple contenu session:

```
{ "id" : "nFg0F8R_rrgxjjmb3eyq0AAkZe6312NI", "expires" : ISODate("2019-06-18T09:57:24.777Z"), "session" : "{\\"cookie\\":{\\"originalMaxAge\\":10800000,\\"expires\\":\\"2019-06-18T09:57:24.777Z\\",\\"httpOnly\\":true,\\"path\\":\\"/\\"},\\"flash\\":{}}" }
```

Cet extrait provient d'un objet session tel que stocké en base de donnée. On y trouve l'id, la date d'expiration, les réglages du cookie client, les messages flash, et plus tard, les informations d'identification Passport.

Enfin express apporte sont lot de validateurs, j'ai donc construit un middleware de validation pour les requêtes de type post (afin d'assainir ces requêtes et d'éviter les injections de code)

exemple: voir partie Sécurité

Colyseus

"Colyseus is a Authoritative Multiplayer Game Server for Node.js. The mission of this framework is to be the easiest solution for creating your own multiplayer games in JavaScript."

Colyseus est un projet open-source maintenu par une petite équipe, offrant un système de Lobby minimaliste pour gérer un serveur de jeu. Son système comprend:

- des salles de jeu
- des joueurs (avec ID unique persistant + ID par salle)
- des états de salle de jeu

L'approche est assez simple:

- 1) les joueurs demandes à rejoindre des salles, s'y inscrivent si l'autorisation leur est donnée, et envois dès lors leurs requêtes au serveur de jeu via socket.io
- 2) Si l'action est validée côté serveur, l'état de la salle est modifié
- 3) Lors de la modification d'un état, les joueurs inscrit à la salle correspondante sont notifiés en temps réel de cette évolution

Ce framework est "engine-agnostic", il peut s'adapter à n'importe quel moteur de création (type Unity ou encore Phaser) et s'implémente très bien au sein d'une application Express. Il permet ici de libérer une forte charge de travail sur un domaine certes passionnant mais pas spécifiquement lié à la création de sites Web, et donc pas essentiel à prendre en main dans le cadre de ce projet.



Arborescence du projet

éléments principaux

/config => fichiers de configuration incluant les réglages possibles, imposés et par défaut pour de nombreux pan de l'application

/controllers => Éléments chargés de manipuler certains objets en base de donnée selon leurs règles de validations et manipulations

/data => Base de données

/models => Schémas d'objets Mongoose pour la base de donnée

/modules (dépendances) => Modules faits pour le projet prêts à être exportés sur d'autres projets

/node_modules (dépendances) => Modules de la communauté Node Js

/rooms => Code des différents types de salle de jeu

- /rooms/Barony** => Jeu de stratégie Baronie
- /rooms/Go** => Mini jeu de go à des fins de test

/routes => Routes accessibles, définies pour routage par Express

- /routes/index** => routes principales
- /routes/user** => routes /user/x pour les requêtes relatives aux manipulations utilisateurs
- /routes/rooms** => routes /rooms/x pour les requêtes relatives aux salles de jeu

/static => Fichiers publics prêts à être servis au client à la demande

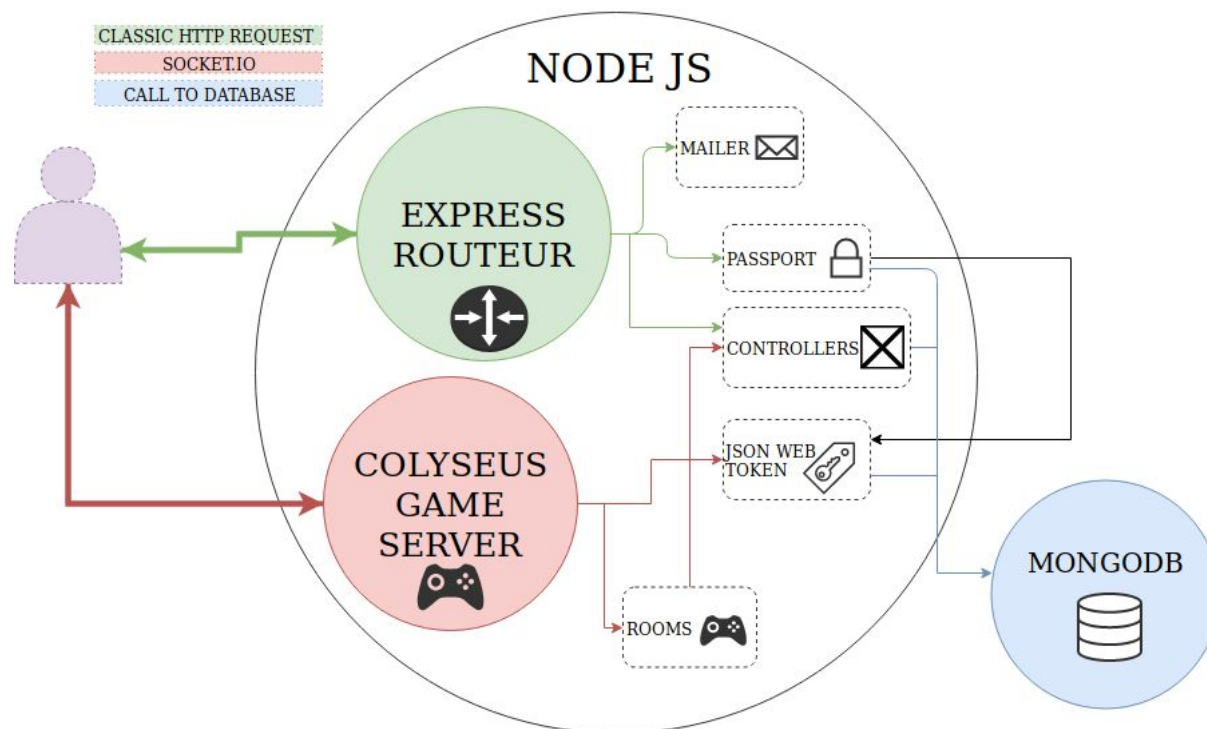
/views => Templates html

- /views/partials** => Navbar, footer, Header
- /views/user** => Vues relatives à l'utilisateur
- /views/rooms** => Vues relatives aux jeux

index.js => racine de l'application

package.json => liste du premier niveau de dépendances en modules de la communauté Node Js

Structure de l'application



La plateforme Node Js accueille le **serveur HTTP** au sein duquel s'exécute Express. A la création du serveur HTTP, un **serveur Socket.io** pour Colyseus est attaché au processus d'Express. Celui-ci délivre les **pages**, s'assure de l'identité de ses clients pour toute opération en base de donnée où avant d'accéder à une ressource protégée. Le client s'enregistre sur le socket souhaité une fois la page délivrée par Express. Colyseus se charge alors de le notifier de l'évolution de sa **salle de jeu**.

Les méthodes concernant les modifications en base de données sont enfermées aux seins des contrôleurs. Colyseus n'utilise la base de donnée que pour la consulter, et ne doit pas pouvoir y écrire. Les modules Passport, JWT et le UserController oeuvrent de concert pour assurer la sécurité du système. Les deux premiers communiquent avec la base de donnée au travers du UserController.

Axes d'amélioration

Une des améliorations importante de cette structure serait d'inclure Express et Colyseus dans des plateformes Node Js distinctes, ainsi que de déléguer la gestion des fichier statiques publiques à un autre serveur comme NGinx, afin d'alléger les charges respectives, et ne pas conditionner la bonne exécution d'Express à celle de Colyseus. Enfin cela permettrait d'utiliser Colyseus-proxy pour gérer de multiples instances de Colyseus derrière un proxy supplémentaire.

En effet ce dernier est plus sujet aux problèmes d'exécutions, la maintenance du code des jeux reposant uniquement sur moi. De plus, une erreur critique sur un des serveur de jeu n'entraînerait plus l'arrêt de toute la structure

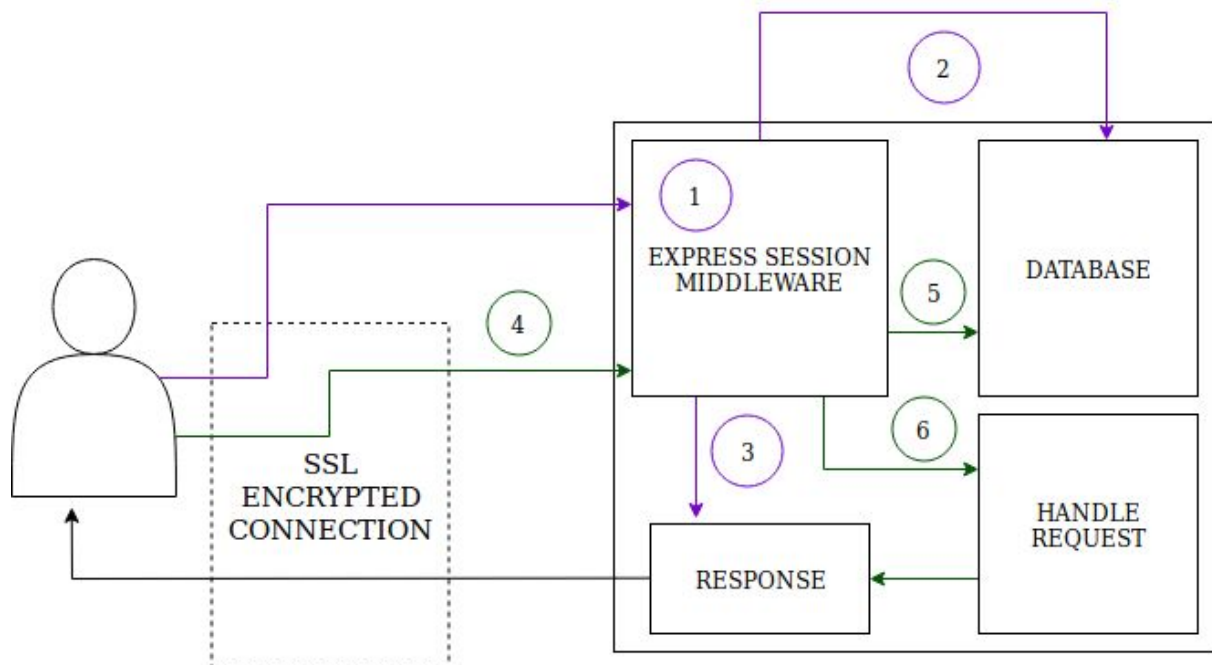
Réalisation

Sécurité

Sessions persistantes

Afin de permettre à l'utilisateur de conserver sur un temps court (quelques heures au plus) les éventuelles données temporaires liés à son passage sur le site dans le cas où il quitterait le site pour y revenir dans ce délai, un système de sessions persistantes en base de donnée est implémenté.

C'est le framework Express qui va apporter cette fonctionnalité:

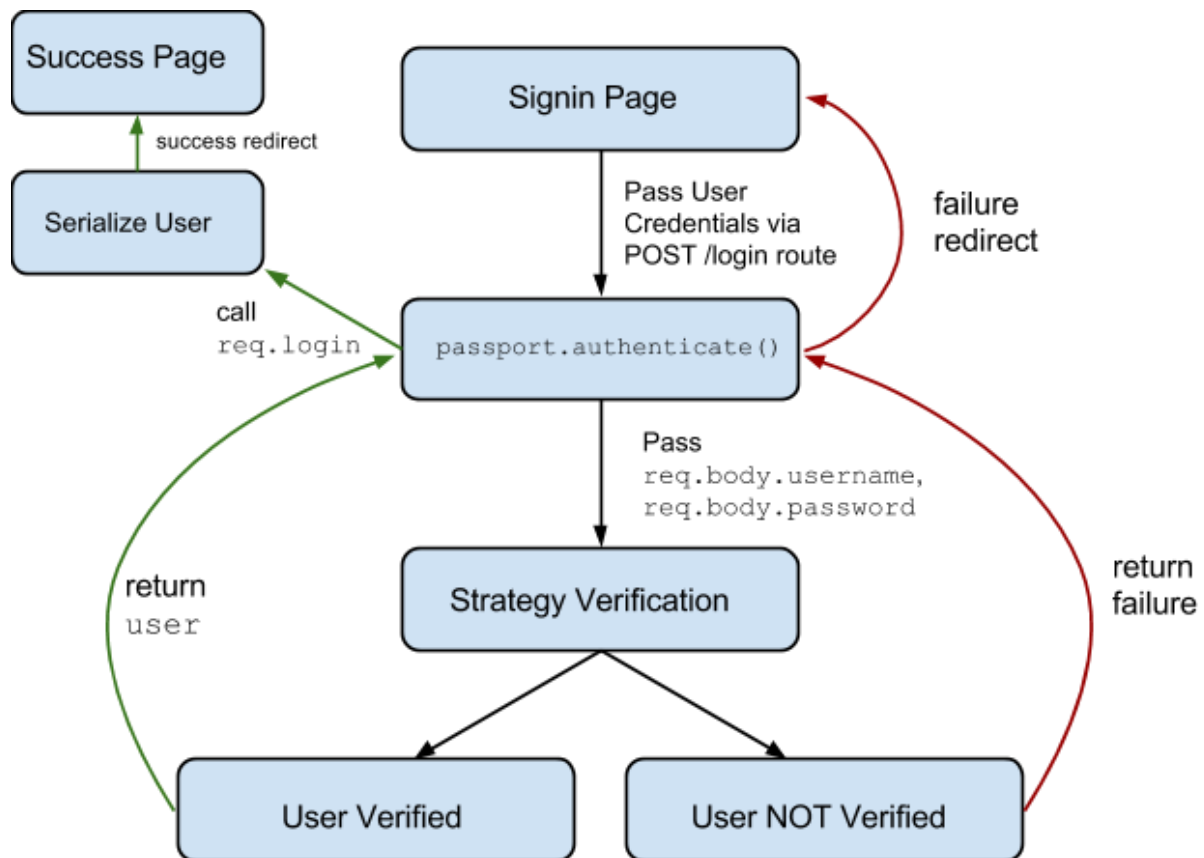


Première Requête d'une machine client

- 1) Génération d'un identifiant unique de session
- 2) Création d'un objet Session (ID, début, durée) et stockage de cet objet en base de donnée
- 3) Inclusions de l'ID Session dans un cookie dans la réponse client après traitement de la requête
- 4) Nouvelle requête du client
- 5) Vérification de la conformité du ticket en base de donnée
- 6) Traitement de la requête

Authentication & Rôles

Afin d'authentifier les utilisateurs lors de requêtes vers des routes restreintes, vers leurs informations personnelle ou vers une page d'administration par exemple, le module Node.js "Passport" apporte un middleware permettant de vérifier les informations fournies par l'utilisateur et d'inclure certaines de ces informations dans l'objet session créé plus tôt.



En plus de ce système chaque utilisateur se voit associer automatiquement **un rôle à l'inscription**, persistant en base de donnée, par défaut: **client**.

Les utilisateurs dont le rôle est configuré sur **administrateur** peuvent accéder aux routes de suppression d'autres comptes et à la page de gestion des salles de jeu.

Mail de confirmation

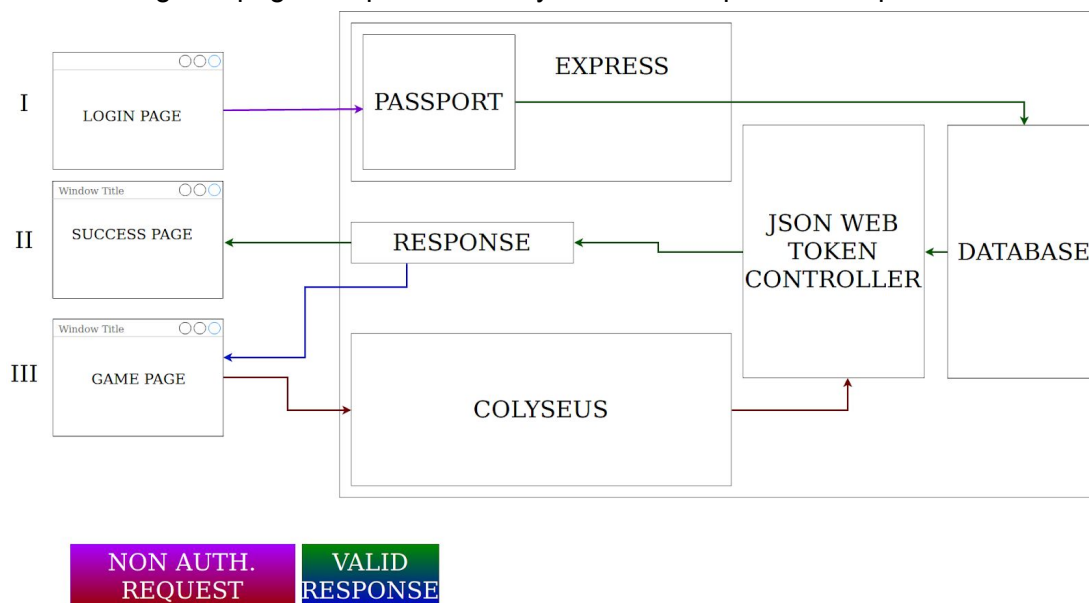
Pour permettre aux utilisateurs de modifier leur mot de passe et de récupérer leurs compte en cas d'oubli de ce dernier, une adresse mail leurs est demandée à l'inscription. A chaque enregistrement ou modification d'une adresse Mail, un ticket unique est généré, transmis par mail à l'adresse, et une route spéciale du site permet de renseigner le ticket dans l'URL. A l'appel de cette route, le ticket est détruit et l'adresse est validée. Un utilisateur ne confirmant pas son mail ne pourra pas récupérer son compte sans l'intervention d'un modérateur.

Token JSON & colyseus

Les différents middlewares cités ci-dessus étudient les requêtes standard faites au serveur par les clients. Cependant, le framework Colyseus n'écoute pas ce type de requête, seulement celles faites sur Socket.io.

“Socket.io : Socket.IO is a library that enables real-time, bidirectional and event-based communication between the browser and the server”

Ce protocole permet simplement d'envoyer des informations aux clients inscrits sur un socket sans attendre qu'ils n'en aient fait la requête. Essentiel pour le jeu, ce système ne peut donc pas établir l'authenticité des requêtes qui lui sont faites sans un mécanisme supplémentaire: l'authentification par **tickets Json**. En effet, l'ID de session créé par Express est stocké par le client dans un “Secure-Cookie”, un cookie accessible par le Javascript exécuté depuis le navigateur. Il faut donc un moyen de transmettre son identité sans recharger la page indépendamment du système de requête classique.



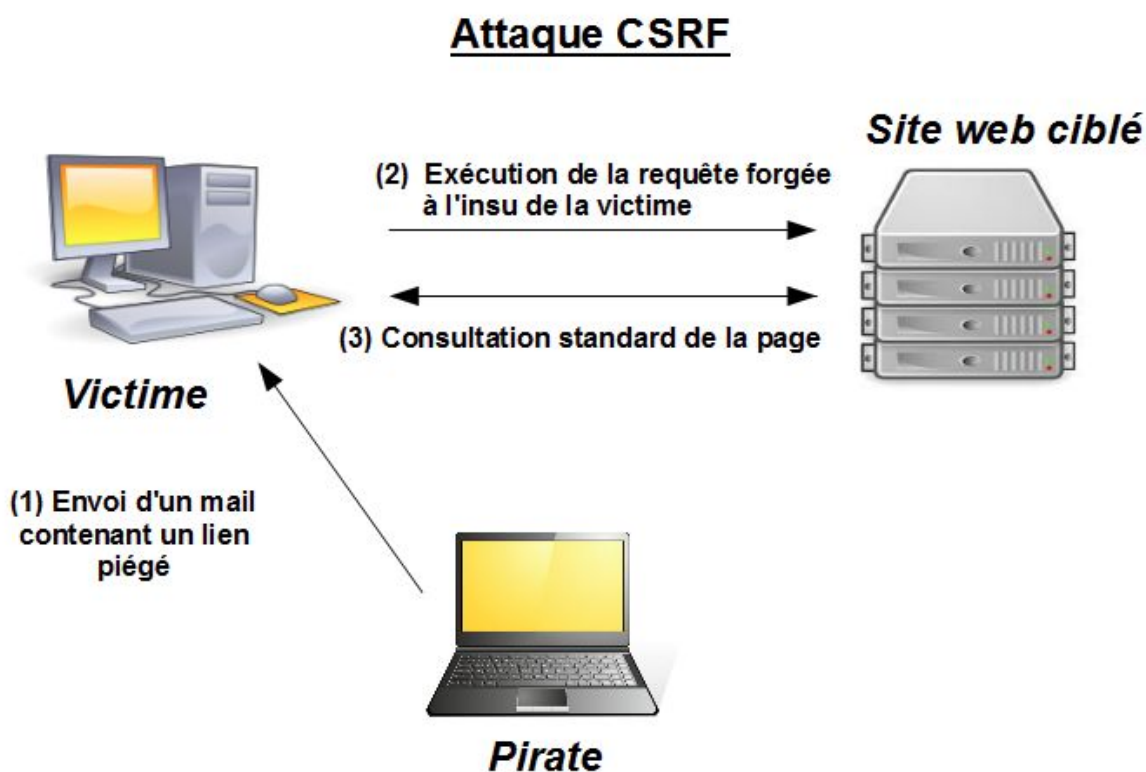
Sur le modèle précédent, l'utilisateur s'authentifie via un formulaire. Si l'identification est confirmée, un token Json est généré, et chiffré depuis un code composé de deux clés: une unique stockée pour le serveur et une pour le client. Le token est alors transmis dans un cookie. Sa date d'expiration est la même que la Session Express ouverte.

Une fois réceptionné, lors de la connexion au socket de jeu, le token est transmis à Colyseus, qui consulte les clés en mémoire pour ouvrir le token et assurer l'identité du joueur.

Cependant, les requêtes concernant les sauvegardes de jeu sont transmises sur le schéma de requêtes classiques et donc sécurisées par Express. La surface à sécuriser ici est donc réduite. Si l'utilisateur demande sa déconnexion, le token JSON est détruit. En bonus, la clé client est modifiée, si le cookie venait à ne pas être effectivement détruit.

Faible CSRF

L'attaque par exploitation de la faille Cross Sites Request-Forgery est relativement simple à mettre en place et ses conséquences peuvent être graves. Elle consiste à tromper l'utilisateur en l'amenant à consulter un lien, reposant donc sur le comportement faillible de l'utilisateur. Ce lien, dissimulable facilement dans de nombreux éléments des pages web, vers le site peut mener à une route exécutant n'importe-quelle tâche que l'utilisateur est autorisé à accomplir, en se servant de son identification préalablement établie.



Une des mesures simples pour se prémunir de ce type d'attaque est de conditionner toute les requêtes importantes à la transmission d'un formulaire, et lors de la délivrance de ce formulaire encore vierge, y lier un token unique. Si le formulaire une fois rempli et transmis par le client au serveur ne comporte pas le dit token, c'est qu'il n'a pas été délivré par le site, et est probablement transmis à l'insu de l'utilisateur.

Validations des requêtes

Dans le cadre de cette application, trois types de requêtes sont à distinguer:

- celles faite via POST ou GET au routeur Express
- celles faites via Socket.io à Colyseus
- enfin celles faites par Express ou Colyseus à la base de donnée

Afin de s'assurer de la conformité des informations transmises dans les requêtes avec les données attendue, une série de validateurs est déployé au sein de l'application.

Ci-dessous, une liste d'exemples de validations de chaque types:

1) Express-validator

```
14 Validator.verifyBody = (req, res, next) => {
15
16   let keys = Object.keys(req.body)
17
18   keys.forEach(input => {
19     switch (input) {
20       case 'email':
21         req.checkBody('email', 'Invalid/Blank E-mail').isEmail();
22         break;
23       case 'password':
24         req.checkBody('password', 'Password is too short').isLength({ min: 4 });
25         req.checkBody('confirmation', 'Password is too short').isLength({ min: 4 });
26         break;
27       case 'username':
28         req.checkBody('username', 'Invalid Username').notEmpty().trim().isLength({ min: 4 }).matches('^[a-zA-Z0-9_-]{4,15}$');
29         break;
30       case 'accept':
31         req.checkBody('accept', 'Wrong attempt').notEmpty().trim().isLength(1);
32         break;
33       case 'avatar':
34         req.checkBody('avatar', 'Wrong attempt').notEmpty().trim().isLength(7);
35         break;
36       case 'nbPlayers':
37         req.checkBody('avatar', 'Wrong attempt').notEmpty().trim().min(2).max(4);
38         break;
39       case 'invit':
40         req.checkBody('avatar', 'Wrong attempt').notEmpty().trim()
41         break;
42     }
43   });
44
45   var errors = req.validationErrors();
46   if (errors) {
47     var messages = [];
48     errors.forEach(function (error, index) {
49       messages.push(error.msg);
50     });
51     messages = removedDups(messages);
52     req.flash('error_messages', messages);
53     next();
54   } else {
55     next();
56   }
57 }
58
59 module.exports = Validator
```

Cette fonction est un middleware appelé sur toute les routes de type POST. Toute les valeurs transmises par formulaires sont vérifiées & validées ici ou ne sont pas utilisées du tout.

Une simple boucle vérifie le contenu de la requête, dont on va d'abords chercher la nature via le switch, puis valider par les validateurs Express. Enfin les éventuelles erreurs sont stockées par le module “flash” en mémoire ou en base de donnée jusqu'à leur affichage côté client.

Dans le détail d'un validateur:

`req.checkBody('username', 'Invalid) => champ et message d'erreur`
`.notEmpty() => n'est pas vide`
`.trim() => supprime les espaces`
`.isLength({ min: 4 }) => longueur minimale de 4 caractères`

.matches('^[a-zA-Z0-9_-]{4,15}\$'); => respecte la REGEX suivante

2) In-Game

```
1  const Cell = require('../cell.js')
2
3  /** RECRUIT .js */
4
5  module.exports = {
6    max: 2,
7    validate: (state, data) => {
8
9      let hex = Cell.get(state, data.map_click),
10         player = state.players[state.order[state.round]],
11         oldHex = Cell.get(state, Cell.StringToCoords(state.selection))
12
13      /** VALIDATOR */
14
15      if (hex == null ||
16          this.counteur > 1)
17         return false
18
19      /** RULES */
20
21      if ((oldHex != null && Cell.getDistance(oldHex, hex) > 1) ||
22          (oldHex != null && Cell.getDistance(oldHex, hex) == 0) ||
23          //(this.moved != null && Cell.getDistance(this.previous, hex) == 0 && ||
24          (Cell.isDefended(hex) != 5 && Cell.isDefended(hex) != state.round) ||
25          !Cell.forbiddenValues(hex, state.round, [5], [])) {
26         return false
27      }
28
29      return true
30    },
31  },
```

Ce pan de code est extrait de l'action de jeu "move", qu'un joueur appelle lorsqu'il déplace un chevalier. La partie Validator s'assure que l'hexagone pointée existe sur la carte et que le joueur dispose de suffisamment de points d'actions. La partie Rules s'assure dans l'ordre que:

- Les hexagones sélectionnée sont voisines et non identiques
- qu'il est possible au vu des règles de jeu de se déplacer dans la case, en étudiant si l'ennemi l'occupe (isDefended) ou que la valeur de la case ne soit pas un lac (forbiddenValues ...[5])

Si aucune de ces conditions d'invalidité n'est remplie, la fonction renvoi true et autorise l'exécution de l'action

3) Mongoose: voir partie Base de Données

Sécurité serveur

cette section tient un bref aperçu des mesures de sécurité prises en compte concernant l'hébergement du site, sachant que le titre ne les concernent pas directement et que leur application consiste en un métier tout à fait à part.

Let's Encrypt

“Let's Encrypt is a free, automated, and open certificate authority brought to you by the non-profit Internet Security Research Group (ISRG)”

Tout est dit. Ce dispositif met à disposition gratuitement des certificats permettant d'assurer la communications sur le protocole de chiffrement TLS, la rendant opaque aux observateurs extérieurs.

UFW - Uncomplicated Firewall

Pare feu minimaliste pour les distributions basées sous Debian. Permet d'assurer le contrôle des ports ouverts sur le serveur.

Fail2Ban

Fail2Ban permet d'établir des règles d'exclusions pour les IP dont les tentatives d'identifications échoueraient trop souvent dans un temps donné. L'IP est alors blacklistée pour un délai choisi. Ce dispositif permet d'éviter de nombreuses attaques de type brute-force, à savoir celles reposant sur un important débit de requêtes.

Une base de donnée NoSQL

No Sql ?

“**NoSQL** désigne une famille de [systèmes de gestion de base de données \(SGBD\)](#) qui s'écarte du [paradigme](#) classique des [bases relationnelles](#).”

Le système NoSql permet de créer des bases de donnée ne connaissant pas les contraintes relationnelles classiques. Le principe n'est pas de se débarrasser d'une contrainte utile, mais plutôt de permettre aux applications pouvant s'en passer de le faire au profit d'un important gain de vitesse. Comme son nom l'indique, son utilisation n'implique pas le langage SQL.

Plutôt que d'indexer et de vérifier à chaque traitement les relations et leurs clés d'identifications, un gestionnaire de base de données NoSql vas “grossièrement” imbriquer les entités par copie les unes dans les autres. on parlera de peuplement. Les avantages sont les suivants:

- *les requêtes sont plus courtes*
- *leur exécution est plus rapide parce que moins gourmande en calcul*
- *la structure des bases est très simple, même pour un important volume de données, et donc très souple, tant qu'elle est évidemment utilisée dans un cadre ou les relations sont simples et peu nombreuses.*

Pourquoi MongoDB



Le scénario d'un serveur de jeu, ou les relations joueurs-joueurs, joueurs-salles et joueurs-données sont peu nombreuses et directes, où les accès à la base de données peuvent amener à l'écriture rapide d'objets volumineux (comme les sauvegardes de jeux) à la structure variable, et où la vitesse d'exécution est capitale, le modèle NoSql m'a semblé adapté. En bonus, le choix du système MongoDB, permet d'économiser ici un temps de développement important, la documentation de MongoDB couvrant essentiellement des exemples en JavaScript. Ce système étant très utilisé avec Node Js, la communauté est réactive en cas de questions.

Mongoose dites-vous ?

L'ODM (Object Document Mapper) Mongoose va permettre d'ajouter un squelette aux objets sur le système un peu trop souple de MongoDB. Ce dernier permet d'établir des schémas d'objets, définissant les structures et les conditions de créations/modifications des objets. Cette couche d'abstraction va aussi permettre de protéger la base de donnée contre

une injection de code / entité dangereuse, en appliquant une série de validations en amont des requêtes.

exemple de schéma:

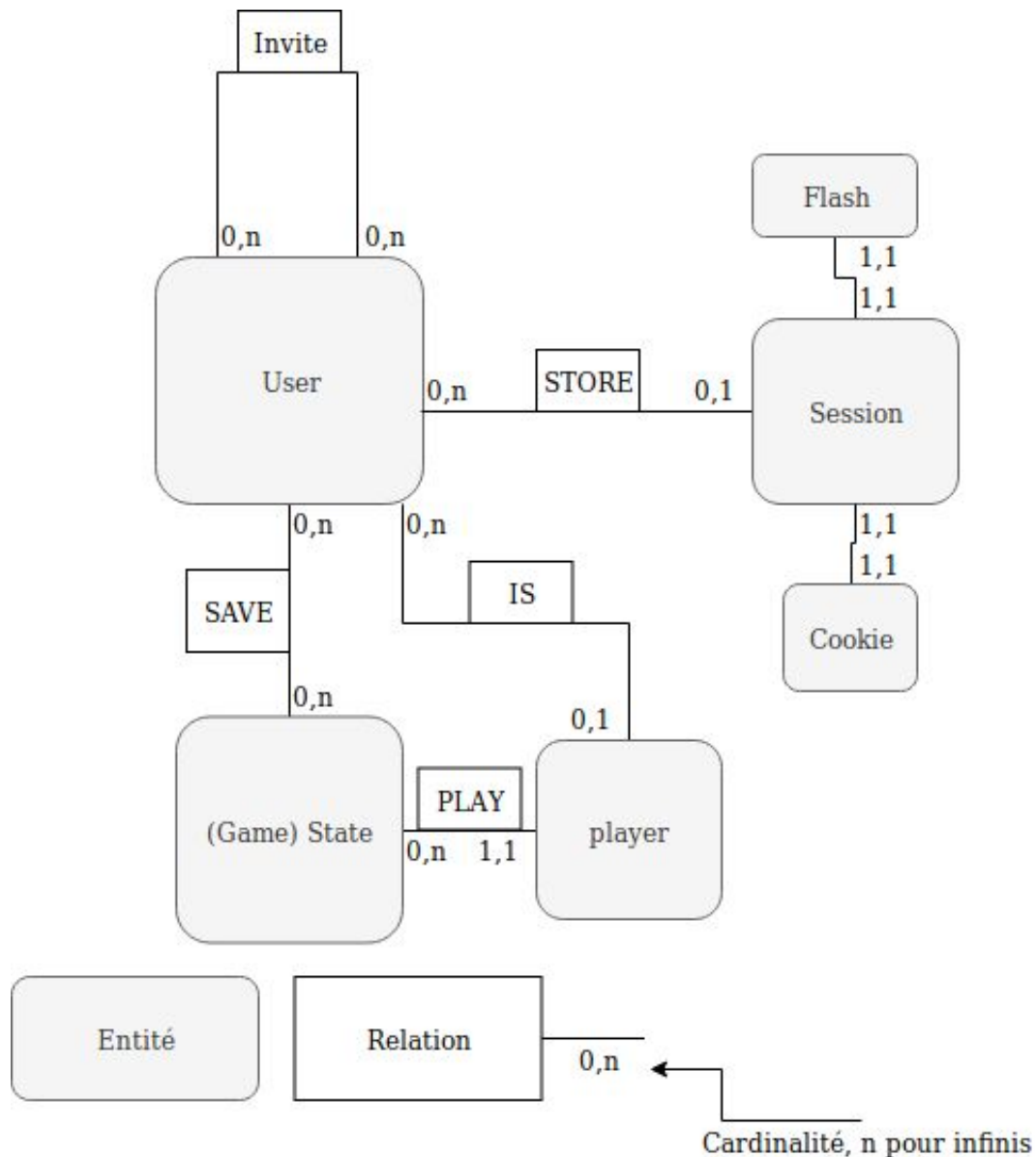
```
1  var mongoose = require('mongoose');
2  var Schema = mongoose.Schema;
3  const bcrypt = require('bcrypt-nodejs');
4  const Invite = require('./invite')
5
6
7  var userSchema = new Schema({
8    username: {type: String, require: true},
9    email: {type: String, require: true},
10   password: {type: String, require: true},
11   role: {type: String, require: true},
12   secret: {type: String, require: true},
13   valid: {type: Boolean, default: false},
14   friends: {type: [String]},
15   invites: {type: [Schema.Types.Invite]},
16   avatar: {type: String, default: 928},
17 });
18
19 userSchema.methods.encryptPassword = function (password) {
20   return bcrypt.hashSync(password, bcrypt.genSaltSync(5), null)
21 };
22
23 userSchema.methods.validPassword = function (password) {
24   return bcrypt.compareSync(password, this.password);
25 };
26
27 module.exports = mongoose.model('User', userSchema);
```

Voici le schéma d'un utilisateur tel qu'entendu par la base de donnée.

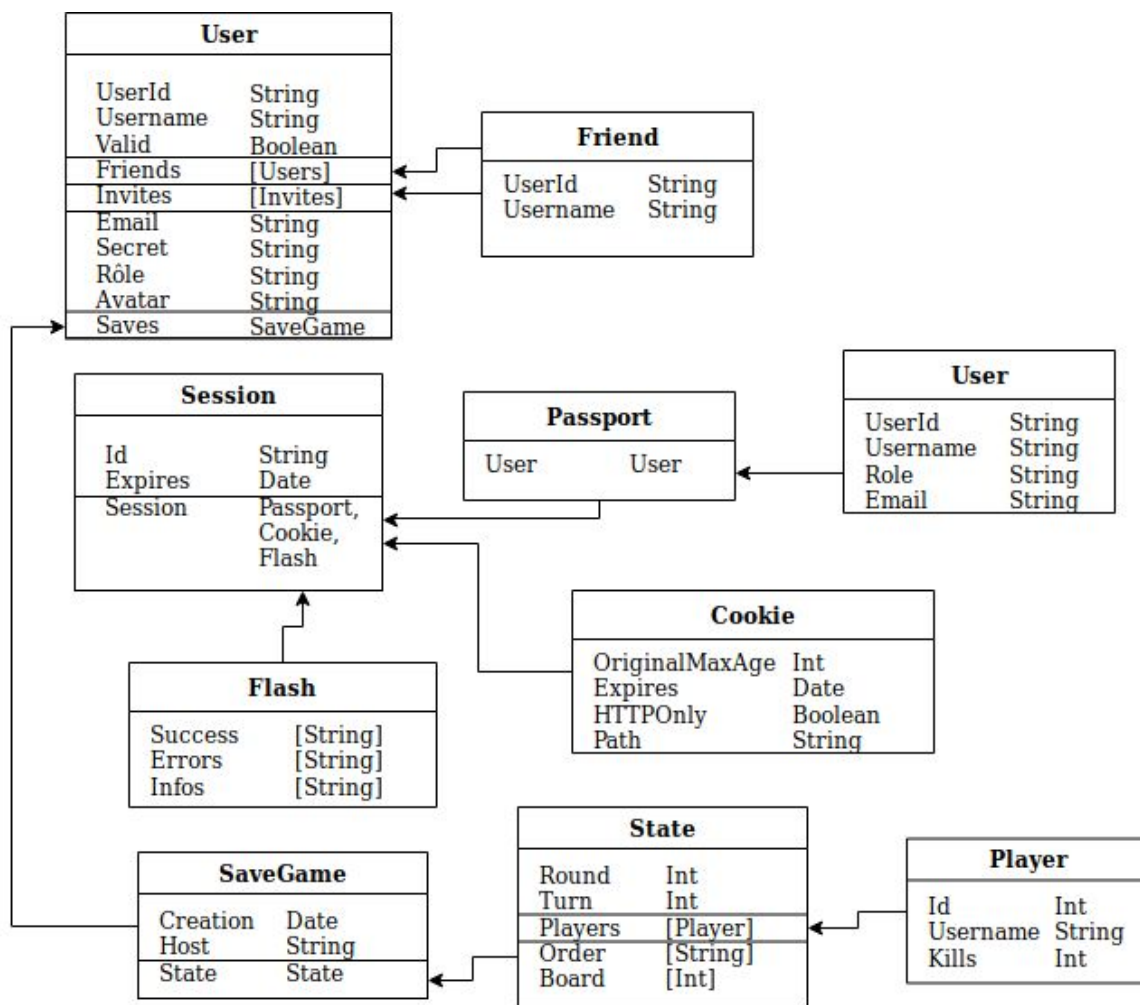
- Au sein de l'objet Schema sont les attributs de l'utilisateur. Leurs valeurs sont des validateurs: Ils indiquent au moins le type de l'attribut, ainsi que toutes ses caractéristiques au sein de l'objet (obligatoire? valeur par défaut?)
- Les fonctions appelées méthodes sont comme leurs noms l'indiquent les méthodes incluses dans l'objet, méthodes spécifiques à cette entité unique.

Remarquez la ligne "invites": son type est aussi un schéma Mongoose. En effet, le préciser permet de s'assurer que l'objet demandé soit de la bonne nature. Cependant, contrairement au système SQL de relations "classiques", les invitations ne seront pas stockées à part et renseignées par imbrications de clé. On dit qu'elles viendront "nourrir" l'utilisateur: ces valeurs seront directement incluses au sein de l'utilisateur concerné.

schéma de la base de donnée:



Ce Schéma représente les entités premières de la base de donnée. Il ne respecte pas les standards de représentations des modèles conceptuels de donnée, mais permet d'établir les **niveaux et natures des relations** qu'il va falloir mettre en place pour l'application. Dans ce modèle, l'utilisateur (User) n'est pas le coeur de jonction des informations. Les systèmes de jeux et de sessions sont indépendants et n'ont pas besoins d'un utilisateur enregistré pour fonctionner. La possibilité d'utiliser la base de donnée pour stocker une sauvegarde étant strictement réservée à un utilisateur inscrit, cette relation session-state n'est pas nécessaire.



Le schéma ci dessus n'est pas une modélisation logique classique du modèle de la base de données. Il représente l'ensemble des modèles (Schéma Mongoose) utilisés par l'application, et leurs arborescence d'inclusion. Les flèches représentées ces inclusions. Dans l'exemple suivant, les documents "STATE" et "PLAYER" sont ceux du **jeu de Go**. Leur structure est similaires pour **Baronnie**, cependant plus de valeurs sont nécessaires au jeu et leur représentation ici n'apporte rien à la compréhension de la dynamique de la base.

- Les relations aux cardinalités simples (1,1 - 1.1) montrées plus haut deviennent des inclusions simples: l'objet source vient peupler l'objet cible.
- Les relations dynamiques (0,n - 0,n) deviennent des documents à part entières. "Friend" pour lister les noms et ids des amis/futurs amis (nom + id suffisent pour nourrir toute les pages privée d'un utilisateur). "Savegame" pour représenter une sauvegarde unique pas un joueur unique.
- Les autres relations quant à elles sont matérialisées par inclusions partielle (notamment des Id) des éléments concernés dans leur cible. Seule les informations utiles à leurs cible y sont incluses.

Les Routes

L'application propose trois routes principales.

I	/	route menant vers la page principale	/
II	/user	routes menants aux actions traitant de(s) utilisateur(s)	/profile, /login, /logout, /change, etc...
III	/rooms	routes pour lister / créer / rejoindre des salles de jeux	/list, /:id, /create, /create-private

Exemple de route:

```
59
60 router.get('/profile', AuthStatus.isLoggedIn, function (req, res, next) {
61   const folder = './static/images/avatars';
62
63   let avatars = []
64   fs.readdirSync(folder).forEach(file => {
65     avatars.push(file)
66   });
67
68   UserController._model.findOne({ username: req.session.passport.user.name }, (err,result) => {
69     if(err || result == null )
70       console.log(err && result == null )
71
72     res.render('./user/profile', { title: 'User Profile',
73                                   invites: result.invites,
74                                   friends: result.friends,
75                                   avatars: avatars,
76                                   avatarCurrent: result.avatar,
77                                   csrfToken: req.csrfToken() });
78   })
79 });
80
```

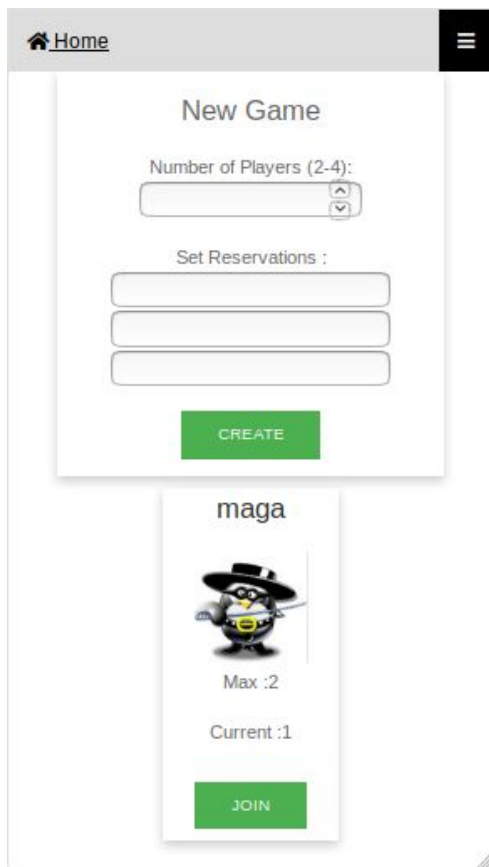
route "/user/profile"

Cette route est appelée via une requête de type GET.

- Le middleware "AuthStatus" s'assure que l'utilisateur est bien identifié avant de lui délivrer la page
- le module fs (pour files-system) va charger la liste des avatars disponibles sur le serveur
- Le contrôleur UserController est appelé pour trouver l'utilisateur de la session courante (contenu dans l'objet Express-session en mémoire serveur)
- Si l'utilisateur existe bien en base, la liste des informations désirées est transmise à EJS (moteur de templating), qui construit la page Web à inclure dans la réponse.

Le Lobby et les fonctionnalités sociales

Extrait de la page de sélection de salles (vue mobile):



Cette page permet de créer, lister et rejoindre les salles de jeu sur un serveur.

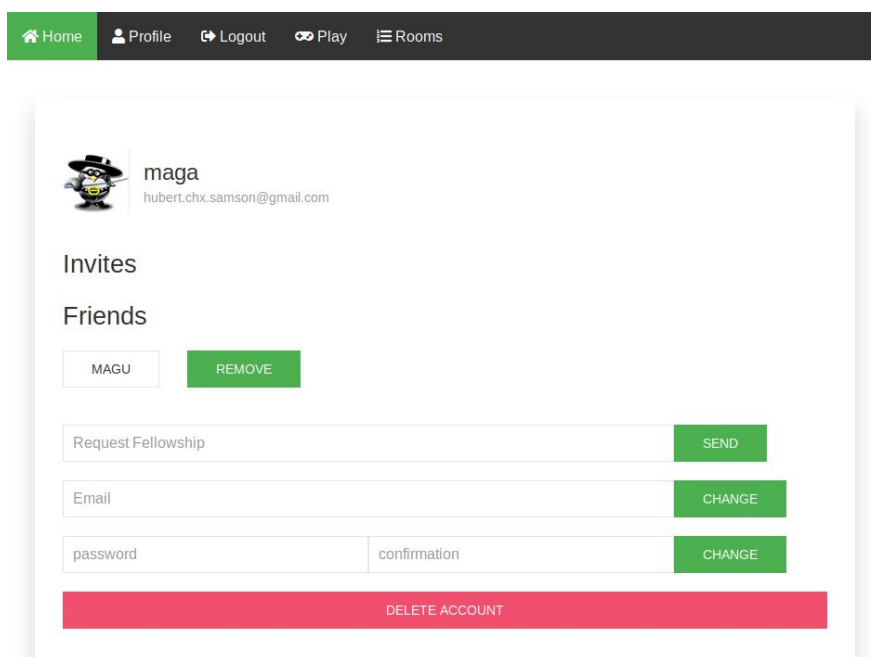
Il existe un exemplaire de cette page par type de jeux sur le site.

Cette vue est incomplète, les fonctionnalités permettant d'ouvrir une salle privée, de définir l'ordre de jeu, ou encore sa couleur ne sont pas encore implémentées correctement (seulement au niveau vue, les fonctionnalités sont prêtes mais le temps manque)

une salle peut être créée avec un nombre de joueurs, et éventuellement des sièges réservés

les Salles disponibles sont représentées sur une carte. Elles ont un Hôte, dont le pseudo/identifiant et éventuellement l'avatar est indiqué en haut de la carte, un nombre de joueurs maximum et un nombre de joueurs présents.

Extrait de la page profil (vue desktop):



Cette page relativement simple et incomplète elle aussi (il y manque l'option pour télécharger une sauvegarde, rendre publique l'adresse mail et pour désactiver temporairement le compte) liste:

- les informations du joueur au dessus
- la liste d'invitation
- la liste d'amis avec un lien vers leurs profils public

- Le formulaire pour inviter un joueur dans sa liste d'amis
- les formulaires pour changer l'adresse mail, le mot de passe ou supprimer le compte.

Exemple de manipulation du DOM:

```

dspList = () => {
  client.getAvailableRooms("barony", function(rooms, err) {
    if (err) console.error(err);
    var table = document.getElementById('room_list')

    /** check if template tag is handled by navigator */
    if ("content" in document.createElement("template")) {

      var template = document.querySelector("#room_template");
      var list = document.querySelector("#room_list");

      rooms.forEach(function (room) {

        let clone = document.importNode(template.content, true),
            title = clone.querySelectorAll("h3"),
            banner = clone.querySelectorAll("img"),
            p = clone.querySelectorAll("p");

        title[0].textContent = room.metadata.name;
        banner[0].src = '/images/avatars/' + room.metadata.avatar;
        p[1].textContent = 'Current :'+room.clients;
        p[0].textContent = 'Max :'+room.maxClients;

        let button = clone.querySelectorAll('button')

        //alert(""+room.roomId+"")
        join = (id) => {
          window.location.href = "/rooms/barony/"+id
        }
        button[0].setAttribute('onclick','join(""+room.roomId+"")' );

        list.appendChild(clone);
      });
    } else {
      /** Other method when tag not handled */
    }
  });
}

```

Cette fonction appartient au module chargé de gérer la page des dalles de jeu (le lobby). Elle utilise du Js natif pour extraire du tag html "template" un layout de carte représentant une salle, qu'elle vient nourrir avec les informations obtenues via client.getAvavailableRooms(Type) (fonction de Colyseus.js)

- 1) Si cette fonction ne renvoi pas d'erreur, on récupère le tableau contenant les cartes remplies
- 2) Si l'attribut template est pris en charge par le navigateur (au moins par les versions récentes de chrome et mozilla)
- 3) Alors on récupère le template, et on parcourt la liste des salles précédemment demandée
- 4) Pour chaque salle, on nourrit un template de carte avec nom/identifiant et avatar éventuel de l'hôte, et avec les informations numériques type nombre de sièges et sièges occupés.
- 5) Enfin, on insère le noeud créé au sein du tableau via appendChild()

Axes d'améliorations

Actuellement, les informations concernant les invitations, les amis, et les salles de jeux sont chargées à chaque appel de la page. Plutôt que d'obliger le re-chargement manuel de la page ou même de mettre en place un rafraîchissement des informations automatique en Ajax, il pourrait être utile d'implémenter un second serveur Socket.io pour gérer les changements du système social de l'application comme des notifications push.

Adapter "Baronie"

Composants de l'application

cell.js => module Client & Serveur contenant toute les fonctions de gestion de la carte de jeu

client.js => connexion à colyseus

draw.js => module de dessin sur canvas

events.js => liste des évènements DOM

game.js => objet de jeu (contient une copie mise à jour des valeurs de la salle de jeu)

geometry.js => module Client & Serveur contenant toute les fonctions mathématiques de manipulations d'hexagone et de carte hexagonales

images.js => module de chargements d'images par promesses

images-list.js => liste d'images à charger

listeners.js => "écouteurs" (événements) attachés aux variables synchronisés par la salle de jeu au client. Lors de la modification côté serveur d'une valeur écoutée, le code côté client est appelé dès que la salle diffusera l'information (tout les x millisecondes)

mouse.js => module pour gérer les mouvements de souris au seins des canvas

stack.js => fonctions de mise à jour des éléments informatifs du DOM

A l'exception du module **geometry** tout les composants furent bâtis par votre fidèle serviteur.

Voici l'ordre indicatif des tâches coté client à l'entrée dans une salle:

- 1) Le navigateur lance le chargement des balises externes et entame la construction du DOM
- 2) Dans un même temps, les scripts Js sont chargés à la suite.
- 3) Connexion à la salle
- 4) lancement du téléchargement des images de jeu
- 5) Si entrée accordée, récupération de l'état originel de la salle
- 6) ainsi que de l'historique de tous les changements survenus précédemment dans la salle
- 7) images chargées
- 8) Dom construit
- 9) Valeurs de la salle récupérée
- 10) Valeur du joueur récupérée

Lorsque ces quatre tâches (quelque soit leurs vitesse d'exécution) sont terminée, et seulement là, l'objet du jeu va dessiner le plateau de jeu au centre de la fenêtre, et commencer à écouter les éventuels actions de la souris ou du clavier. Cependant le serveur n'autorise les actions qu'une fois tout les sièges de la salle occupé.

L'interface

La création d'une interface intuitive est une tâche délicate. Le rendu visuel du site concerne des écrans de tailles très variés, et la charge d'informations à l'écran comme leurs répartitions comptent. Dans le cadre de la création de jeux vidéos, l'aspect intuitif et ergonomique de l'interface est essentiel. Même dans le cadre d'un jeu de stratégie ou le joueur est prêt à fournir un effort intellectuel pour s'immerger dans un jeu aux graphismes moins soigné et à l'interface parfois très fournie, tout plaisir risque d'être perdu si les boutons sont à chercher plusieurs secondes entre chaque action et que leurs conséquences n'apparaissent pas comme logique pour le joueur. Garder une interface "simple" permet donc d'éviter certains écueils. Les éléments principaux à prendre en compte sont la taille, le ratio d'affichage et le niveau de zoom. Le pari ici est d'avoir un site disponible sur écran de Télévision comme sur mobile. De plus, le projet part d'une version plateau.

Le postulat de départ est de tenter de reproduire cette univers plateau à l'écran, ce qui implique:

- 1) un plateau de jeu central
- 2) un "stack" avec une liste de pions personnels
- 3) un tableau de scores
- 4) une visibilité sur le stack des adversaires (le jeu est entièrement ouvert)

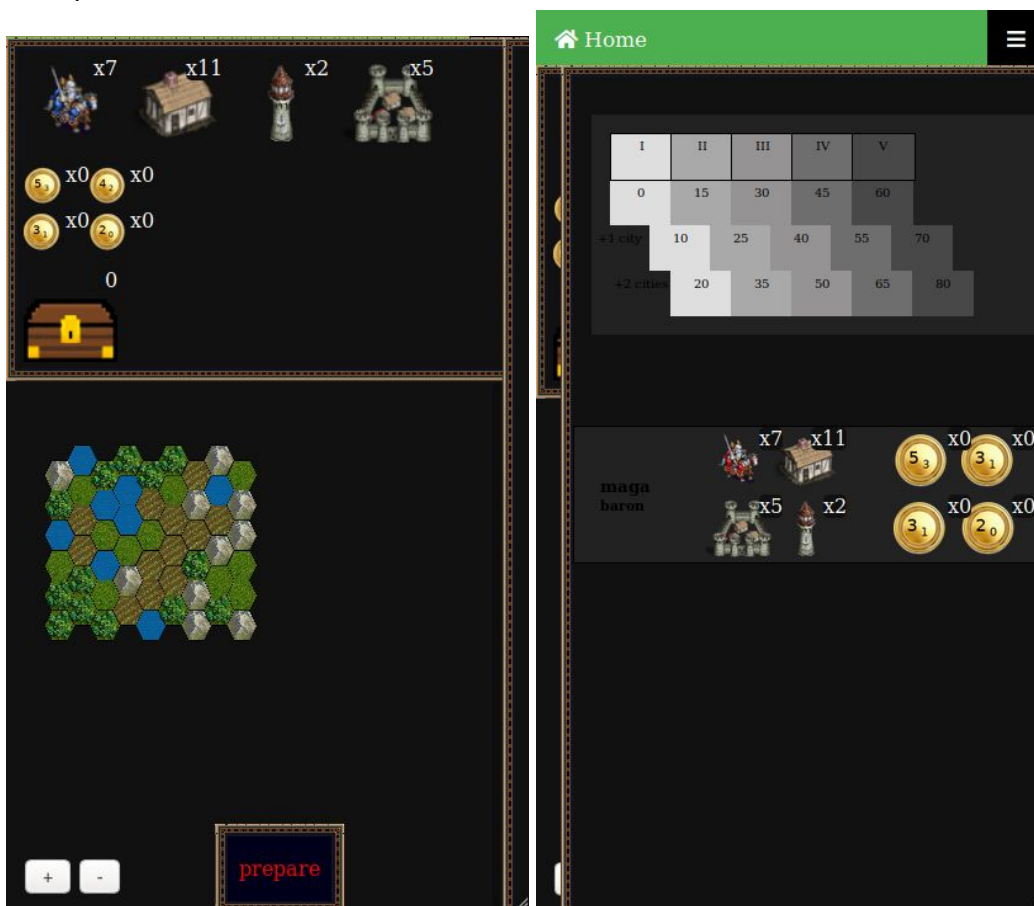
De plus le jeu doit laisser apparente la barre de navigation, et comporter un système pour initier/annuler/revenir sur/confirmer une action, afin de ne pas rendre tout clic définitif et d'éviter les erreurs utilisateurs.

exemple desktop:



Les boutons +/- assurent le niveau de zoom du canvas. Les boutons <-, V, et cancel servent à gérer l'action en cours, dont le nom est indiqué au dessus. les informations personnelles sont en haut, les informations générales regroupées sur la droite.

exemple mobile:



La vue de droite apparaît au clic sur la barre latérale

Le DOM

Le Document Object Model décrit les différents éléments de la page HTML, ainsi que sa structure

Le jeu est composé de trois éléments majeurs,

- 1) la division du haut contenant les pions
- 2) la division de droite contenant les autres informations
- 3) le Canvas central

En clair, deux méthodes sont utilisées pour rendre et manipuler le jeu:

- la manipulation classique en Javascript des éléments le dessin intra-canvas via des algorithmes de dessin

Les éléments du DOM sont manipulés par le Js via leurs propriétés Css ou leurs contenus. Des événements peuvent leurs être attaché. Ces éléments sont sensible au zoom, et si leurs taille est fixée en unités “viewport-height %” ou “viewport-width%” et restera donc proportionnellement fixe, le texte doit pouvoir être grossis pour des besoins de lecture. De plus le zoom natif du navigateur est périlleux à “désactiver” car dépendant des méthode d’un navigateur spécifique.

Au contraire, donner aux canvas une taille fixe et les redessiner à chaque tentative de zoom est simple et permet de se débarrasser du zoom natif (un client doit pouvoir zoomer sur la carte sans modifier l’affichage de toute la fenêtre) pour le remplacer par un niveau de zoom intra application, ne s’appliquant qu’au plateau de jeu.

Les canvas du jeu sont superposés sur plusieurs “layers”. Cela permet de ne dessiner que les parties utiles lors d’une modification de l’état du jeu, de laisser à part celle non concernée et de rafraîchir les layers ciblées.

Exemple d'événement Dom:

```
let validate_btn = document.getElementById('validate_btn')
validate_btn.addEventListener('click', () => {
  room.send({ validate_action: true })
})
```

Lors d’un clic sur le bouton “valider”, dont l’ID est “validate_btn”, un message de validation est envoyé à la salle de jeu.

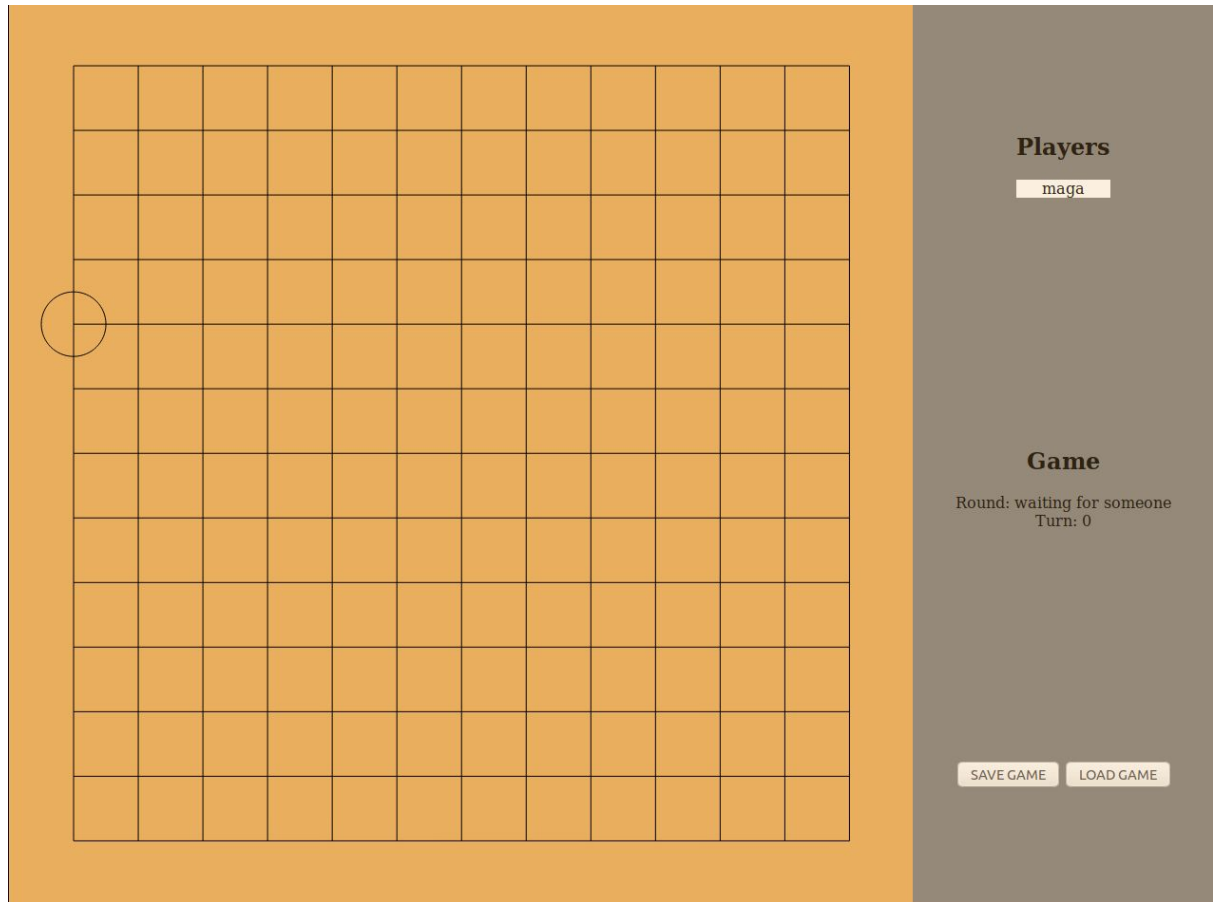
Exemple de fonction de dessin sur canvas:

```
147 Draw.cursor = () => {
148     const hex = Mouse.cell
149     canvas = Game.layers.cursor.canvas,
150     ctx = Game.layers.cursor.ctx,
151     dpi = window.devicePixelRatio,
152     layout = new Layout(Layout.flat, Game.hexSize, new Point(0, 0)),
153     width = canvas.clientWidth,
154     height = canvas.clientHeight,
155     size = ImagesHandler.getImage('battlefield').height,
156     coords = layout.hexToPixel(hex)
157
158     canvas.width = width * dpi
159     canvas.height = height * dpi
160
161     //ctx.scale(2, 2)
162     ctx.clearRect(0, 0, width, height)
163     ctx.translate(width / 2, height / 2);
164     ctx.drawImage(
165         ImagesHandler.getImage('brush'), // image
166         1 * size, // source x
167         0, // source y
168         size, // source width
169         size, // source height
170         coords.x - (size / 2), // target x
171         coords.y - (size / 2), // target y
172         size, // target width
173         size // target height
174     )
175     ctx.translate(-width / 2, -height / 2)
176 }
```

- 1) La fonction récupère d'abord:
 - le layer concerné
 - son contexte de dessin
 - le niveau de zoom actuel sur la fenêtre
 - la taille du canvas et de l'image à dessiner
 - le "layout" (grille de positionnement des hexagones)
 - la dernière coordonnée en pixel pointée par la souris, depuis les coordonnées 3D de l'hexagone.
- 2) La fonction ajuste ensuite la taille relative du canvas selon sa taille réelle, afin de corriger le zoom
- 3) La fonction nettoie le canvas, puis change le repère 0,0 px pour le placer au centre du canvas et se repérer sur le layout dont le point 0,0 est le centre
- 4) L'image est dessinée selon sa taille et ses coordonnées

Mini-Goban

rendu visuel du jeu:



La navbar n'est pas visible sur cette vue, elle apparaît au passage de la souris , sur le dessus de l'écran sur mobile, et sur le côté sur mobile en vue latérale. La partie de droite informe du nom de l'adversaire, du tours, du round, des coups joués. Les boutons save game et load game sont présents pour tester la fonctionnalité.

j'ai créé, en parallèle au jeu Baronie, une base de jeu de Go (toute les règles ne sont pas implémentées). Ce "sous" projet a deux intérêt:

développer les fonctionnalités de lobby/gestion de jeu sur un jeu moins important que baronnie, pas encore fonctionnel à ce moment, et tester toute ces fonctionnalités avec un deuxième jeu pour s'assurer de leur caractère générique.

Ressources, tests & veille

A partir d'ici, cette synthèse va prendre une forme moins académique pour me permettre d'explicitier un peu plus mon ressenti personnel au long du projet.

Choix et constats personnels

Tout d'abords, l'ordre dans lequel j'ai pensé et construit les différents éléments de l'application ne correspond pas du tout à l'ordre qu'il "aurait fallu" aborder pour clore ce projet même seul, dans un cadre où les technologies sont maîtrisées à minima et le cahier des charges imposé en amont.

Il me faut préciser deux choses ici: avant cette formation, je n'avais jamais écrit de scripts web. Et au début de ce projet, je n'avais encore jamais écrit une ligne de Javascript. Si le projet m'a tout de suite emballé, pour le plaisir comme pour la "prouesse" technique, je n'étais absolument pas sûr de voir autant aboutir l'application.

J'ai donc commencé par construire ce qui me semblait le plus inaccessible: la gestion de la carte hexagonale. Soit le rendu client, avant même l'architecture serveur. J'ai implémenté l'identification et la plupart des mesures de sécurités postérieurement à l'établissement des routes, et les autres exemples ne manquent pas. Je suis aussi revenu de nombreuses fois sur des composants complets dont la dynamique ou la syntaxe m'apparut par la suite inadaptée.

Certaines rencontres, entre dev, créateurs web ou simple mortels m'ont amené à optimiser voir réinventer certains pans de l'application.

Cependant je me suis attaché à bien respecter les points suivant:

- implémenter chaque fonctionnalité dans un produit toujours "prêt à l'emploi". A tout moment, le développement aurait pu s'arrêter en laissant une base utile pour d'autres projets
- documenter mon code proprement, plus pour structurer ma pensée que pour partager le dit code

De nombreuses heures de flous furent consacrées à trop insister sur des points d'incompréhension: ce fut aussi une école de patience et d'adaptation.

De plus plusieurs choix dans ce projet sont liés aux contraintes du diplôme: dans le cadre d'une application désireuse d'accueillir vite un grand nombre de joueurs, la fonctionnalité d sauvegarde en base de donnée par exemple n'est clairement pas, au vu de son coût serveur, une priorité sur par exemple un mécanisme de chat In-game. Cependant le diplôme comporte une importante part sur la gestion de bases de donnée et Socket.io est déjà utilisé au sein d'Express. Ce n'était donc pas une plus value pour le titre ici.

Tests de jeu

Le système de jeu est le pan de l'application qui à donné lieu au plus grand nombre de tests, et est plutôt bien représentatif du projet.

C'est aussi le morceau de code qui fut le plus sujet aux corrections et refontes.

Ces tests sont clairement établis comme tels puisqu'ils impliquent:

- une mise en condition précise de l'application au préalable
- un protocole d'action long et pré établis
- un résultat attendu
- et de nombreux résultats inattendu, aux sources identifiées rapidement (conception comme syntaxe)
-

Mise en situation

Pour résumer, chaque joueur peut effectuer une seule action par tour, elle même composée de plusieurs éléments. Par exemple, un joueur peut choisir d'utiliser son tour pour payer un tribut, l'action "PAY". Auquel cas il doit déposer autant de pièces d'or nécessaire dans le coffre commun pour atteindre au moins la valeur de 15. Si c'est le cas le joueur peut acheter son titre.

Lors de cette action, le nombre de pièces présentes dans la main du joueur et celles présentes dans le coffre seront modifiés. La fonctionnalité BACK (appui sur un bouton retour) doit permettre d'annuler un et un seul de ces couple de changement, sans réinitialiser l'action.

```
11
12 module.exports = {
13   max: 50,
14   validate: (state, data, handler) => {
15     let player = state.players[state.order[state.round]]
16
17     if(!data.selection &&
18       data.selection.split('-')[0] == 'gold' &&
19       data.selection.split('-')[1] < 0 ||
20       data.selection.split('-')[1] > 3) ||
21     player.golds[data.selection.split('-')[1]] == 0){
22       return false
23     }
24
25     console.log('action validated')
26     return true
27   },
28   execute: (state, data, handler) => {
29     let player = state.players[state.order[state.round]]
30
31     this.selected_coin = data.selection.split('-')[1]
32
33     /** Memorize */
34     handler.affected_players.push(JSON.parse(JSON.stringify(player)))
35     handler.affected_chest.push(JSON.parse(JSON.stringify({index: this.selected_coin, value: state.chest[this.selected_coin].current})))
36
37     console.log('chest', state.chest[this.selected_coin].current)
38     state.chest[this.selected_coin].current++
39
40     handler.counteur++
41   }
42 }
43
```

Une action (ici PAY) est composé d'un validateur, s'assurant de la conformité du coup, et d'un exécuter, chargé d'appliquer les modifications demandées. Ici, le validateur s'assure qu'une pièce est sélectionnée, que sa référence existe et que le joueur en possède encore

au moins une. L'exécuteur stocke dans un tableau les états précédant du coffre et du joueur concerné, puis les modifie

```
177     back(state) {
178         let done = false
179
180         if (this.affected_hexes.length > 0 && this.affected_hexes[0] != undefined) {
181             Cell.update(state.grid, this.affected_hexes[this.affected_hexes.length - 1])
182             this.affected_hexes.pop(1)
183             done = true
184
185             if (state.action == 'move') {
186                 Cell.update(state.grid, this.affected_hexes[this.affected_hexes.length - 1])
187                 this.affected_hexes.pop(1)
188             }
189         }
190         if (this.affected_players.length > 0 && this.affected_players[0] != undefined) {
191             let player = this.affected_players[this.affected_players.length - 1]
192             state.players[state.order[state.round]].switchPlayer(player)
193             this.affected_players.pop(1)
194             done = true
195         }
196         if (this.affected_chest.length > 0 && this.affected_chest[0] != undefined) {
197             let coin = this.affected_chest[this.affected_chest.length - 1]
198             state.chest[coin.index].current = coin.value
199             this.affected_chest.pop(1)
200         }
201         if (this.counteur > 0)
202             this.counteur--
203     }
```

La fonction BACK parcourt une suite de tableau-mémoires depuis leurs dernière entrée, met à jour les valeurs actuelles du jeu avec les valeurs précédentes, puis détruit l'entrée concernée.

Objectif: test de l'action PAY, ainsi que de la fonctionnalité back de l'ActionHandler au seins de l'action

Protocole (données entrées) :

- 1) Modification du nombre de pièces de départ des joueurs. Ces derniers partent avec 5 pièces de chaque au lieu de 0
- 2) Suppression de la contrainte d'ouverture: la première action obligatoire pour tous les joueurs est l'établissement de trois citées de départ. Au seins de ce test cette contrainte est désactivée car chronophage et non relative au système de paiement des tributs
- 3) Modifications des réglages pour imposer les salles deux joueurs sur la route de distribution automatique de salle
- 4) Redémarrage du serveur
- 5) Création d'une salle
- 6) connexion de deux joueurs anonymes
- 7) I) Dépense des pièces d'or < 15
- 8) II) Dépense des pièces d'or > 15
- 9) III) Dépense des pièces d'or puis retour avant validation
- 10) IV) Dépense des pièces d'or puis retour avant annulation

Données attendues:

- I) L'action doit rester en attente sans être validée ni annulée côté serveur
- II) L'action doit être validée, le nombre de pièces du joueurs et du coffre mis à jour en fonction des ,valeurs titre du joueur mis à jour à la hausse
- III) & IV) Les effets sont similaires: l'état du coffre et du joueur doivent revenir à celui précédant le dernier coup du client, et l'action doit pouvoir se valider si la contrainte > 15 est respectée.

Données obtenues:

- test I.
 - résultat: Erreur de conception. la structure du coffre coté client ne respecte pas celle coté serveur.
 - solution: Utilisation directe de l'objet chest depuis le serveur chez le client. Ce dernier est désormais synchronisé en temps réel en sens unique.
- test II.
 - résultat: Erreur de syntaxe.
 - solution: corriger la syntaxe...
- test III.
 - résultat: Erreur de conception. Si un joueur à quitté la salle en cour d'action et est remplacé depuis, le joueur subséquent n'a pas le même identifiant. Hors, lors de l'appel à la fonction "BACK", le script tente d'écrire les nouvelles valeurs en se servant du nom du joueur par qui elles furent établies/modifiées, nom qui n'est plus présent dans le tableau des joueurs actuels.
 - solution: Utiliser un attribut "order" pour retrouver l'index d'inclusion d'un joueur dans le tableau des joueurs, par rapport à la place que son prédécesseur occupait et non à son identifiant
- Test IV
 - résultat: Succès. Les données de fond sont bien à jour et l'interface graphique s'est modifiée en conséquence.

Ressources Graphiques / Sonores

Afin d'économiser un précieux temps de création de sons et d'images, je me suis généreusement servi, avec la bénédiction directe des auteurs que j'ai pu contacter, et la bénédiction tacite des autres, dans les ressources sonores et visuelles de l'application ci-dessous:



Le jeu BATTLE FOR WESNOTH est un jeu gratuit, sous licence libre, vieux d'une décennie maintenant. Il fut à l'origine créé et maintenu entièrement bénévolement.

Cette section est un remerciement à l'équipe de ce projet.

L'univers graphique (jeu médiéval tour par tour sur hexagone) se prête parfaitement au jeu Baronnie. Il est cependant évident que si le projet "Baronnies" venait à prendre une tournure plus professionnelle un travail de création personnelle serait nécessaire ne serait-ce que pour une question d'originalité.

Recherche Anglophone

La **quasi-totalité des recherches documentaires** et questions liés à ce projets furent menées **en anglais**: W3School, MDN Mozilla, StackOverflow, Colyseus, ainsi que toute les documentations spécifiques et petits blogs, pour ne citer que ca.

Etant d'un naturel mathématico-réfractaire, voici un problème que seul les mathématiques Euclidiens, et donc une solide et pédagogique documentation, pouvaient m'aider à résoudre: la **gestion de la carte hexagonale**. En effet, pour utiliser des algorithmes simples manipulant une telle carte un système de coordonnées axiales ou en 3 dimensions sont adaptés. Cependant, pour la stocker, un tableau à deux dimension de coordonnées dites "offset" est préférable.

Cette problématique ne se serait peut-être pas posée avec des hexagones reconnus comme éléments à part entières du DOM et non comme partie intégrante du dessin d'un canvas. Il s'agit alors de calculer "manuellement" la position de la souris sur la carte.

Le Graal:

<https://www.redblobgames.com/grids/hexagons/>

Cette documentation fait état de toute les techniques de manipulations et stockages des cartes hexagonales au seins d'un jeu. Il implémente aussi un générateur procédural de code permettant de produire à la volée des bibliothèques à jour dans le langage souhaité, ci Javascript. Merci pour le temps et la pédagogie de l'auteur.

traduction d'un morceau du site en ANNEXE 2

Passage en Production

Cette section s'attache à décrire brièvement la mise en environnement de production de l'application.

Machines

Une connexion fibre-optique, une Freebox, un Raspberry-Pi et un bon câble ethernet. Enfin une machine de contrôle avec interface graphique et logiciel de connexion ssh.

Serveur

Un nom de domaine sur Netlib.re, un certificat let's Encrypt, Un serveur NGINX, Node Js, MongoDB, UFW, Fail2Ban, Pm2

Protocole

1) Branche Github

La première étape consiste à créer une branche "stable" sur le projet Github, afin de séparer les phases de développement des phases de production. Cela permet de développer une fonctionnalité tout en la testant sans l'implémenter dans le produit final avant qu'elle ne soit complètement opérationnelle. La branche "Stable" n'admettra que des mises à jours de maintenance ou des blocs complètement fonctionnels.

2) Installation serveur

Installation d'une distribution orientée serveur, ici Ubuntu-server,
mise en place d'une connexion ssh unique avec identification par clé RSA,
Installation de Node Js, Pm2, et MongoDB
Installation et configuration de NGINX comme reverse proxy, dirigeant les requêtes vers le port occupé par le serveur HTTP Express

3) Configuration des ports

Configuration de UFW pour ouvrir les ports 22 (ssh), 443 et 80 (pages web)
Configuration de FreeOS sur la FreeBox afin d'établir une DMZ (demilitarized zone),
autorisant les requêtes depuis l'extérieurs sur les ports écoutés par Nginx, seulement en direction de ce dernier.

4) Installer & exécuter l'application

Usage des commandes git via ssh pour installer ou mettre à jour l'application sur le serveur depuis la branche stable.

Utilisation de PM2 pour lancer Node Js en tâche de fond et observer sa bonne exécution

5) Sécuriser

Installation et configuration de fail2Ban pour surveiller les occurrences de connexions vers les ports ouverts

Installation du certificat Let's Encrypt

6) Configuration DNS

Configuration sur netlib.re des entrées DNS menant vers le domaine ou les sous domaines du site en fonction des différents protocoles.

Hôpla! Un site près à l'emploi pour toute une série de test. Avec une bonne connexion internet et un Raspberry-Pi récent, ce dispositif peut déjà gérer quelques dizaines de clients simultanés, pour un coût minime.

Veille & maintenance

L'avantage premier d'une solution NodeJs + Express est sa rapidité de déploiement. L'un des principaux inconvénient de ce type de structure est la multiplicité et les cycles de développement très courts des modules nécessaires. Si le cadre de travail est très souple, le temps à accorder à la maintenance à chaque changement dans un module est très important.

Le domaine de la sécurité, quelque soit le type d'application ou de la technologie, est sujet à des évolutions constantes et une documentation la plus large possible sur le sujet n'est pas malvenue.

Enfin les réglementations quant au traitement des données personnelles font actuellement l'objet de nombreuses modifications dans le monde, l'approche la propriété intellectuelle en général vouée à évoluer rapidement avec les évolutions sociales. De plus, tout comme pour la sécurité en général, aucun système n'existe sans imperfections quant au traitement de ses données, et la prise d'information régulière est essentiel tant l'enjeu est grave, pour les clients comme pour le responsable de l'application.

Conclusion

Après une période de plus de cinq mois de développement & apprentissage sur cette application, force est d'admettre qu'il a été difficile de se retrouver dans un projet aussi gros, et que le choix de mois de fonctionnalités plus abouties m'aurait sans doute permis de passer moins de temps sur la théorie et plus sur la technique. Cependant, autant par la diversité des thèmes abordés que par la rigueur et l'organisation qui furent nécessaires, ce projet m'apparaît comme un choix adapté pour un développeur junior sans idées de spécialisation actuellement .

En bref, ce projet n'était pas la meilleure façon de couvrir tout les titres du référentiel au mieux pour le moindre effort.

Ce n'était pas non plus ma meilleure idée pour lancer un projet commercial

Ce n'était pas non plus le meilleurs moyen de prouver rapidement mes pratiques professionnelles ou mon investissement.

Mais c'était la meilleure conjugaison de ces points m'étant venue à l'esprit. Et le sentiment grisant d'appeler trois amis pour essayer mon jeu fraîchement hébergé justifiait largement l'effort.

Merci pour votre lecture

ANNEXES

ANNEXE 1 - Charte Graphique

Cette mini “Charte Graphique” fait état du jeu de couleur principal utilisé pour l’application.



Black, 0,0,0, #000000

Main Font color



Dark Gray, 51,51,51 , #333333

Navbar color



Pale gray, 204,204,204, #CCCCCC

**Navlinks + buttons on hover
or click**



Pale Gray, 221, 221, 221, #DDDDDD

Main Background



White, 255,255,255, #FFFFFF

**Font color on dark
background**



Opal Green, 76,175,80, #4CAF50

Active Nav link, Buttons

ANNEXE 2 - Traduction

Original

Hexagons are 6-sided polygons. *Regular* hexagons have all the sides the same length. I'll assume all the hexagons we're working with here are regular. The typical orientations for hex grids are vertical columns (flat topped) and horizontal rows (pointy topped).

Hexagons have 6 sides and 6 corners. Each side is shared by 2 hexagons. Each corner is shared by 3 hexagons. For more about centers, sides, and corners, see [my article on grid parts](#) (squares, hexagons, and triangles).

Angles#

In a regular hexagon the interior angles are 120° . There are six “wedges”, each an equilateral triangle with 60° angles inside. Each corner is size units away from the center. In code:

```
function pointy_hex_corner(center, size, i):  
    var angle_deg = 60 * i - 30°  
    var angle_rad = PI / 180 * angle_deg  
    return Point(center.x + size * cos(angle_rad),  
                 center.y + size * sin(angle_rad))
```

0°30°60°90°120°150°180°210°240°270°300°330°60°60°60°120°flat
pointy

To fill a hexagon, gather the polygon vertices at `hex_corner(..., 0)` through `hex_corner(..., 5)`. To draw a hexagon outline, use those vertices, and then draw a line back to `hex_corner(..., 0)`.

The difference between the two orientations is a rotation, and that causes the angles to change: flat topped angles are 0° , 60° , 120° , 180° , 240° , 300° and pointy topped angles are 30° , 90° , 150° , 210° , 270° , 330° . Note that the diagrams on this page use the y axis pointing *down* (angles increase clockwise); you may have to make some adjustments if your y axis points up (angles increase counterclockwise).

Traduction

Les hexagones sont des polygones à 6 côtés. Les Hexagones réguliers ont des côtés de même longueur. Je pars ici du principe que ceux que nous utiliserons sont réguliers. Les orientations classiques pour des grilles hexagonales sont des colonnes verticales (sommet plat) et des lignes horizontales (sommet pointu)

Les hexagones ont 6 cotés et six coins. Chaque côté est partagé par 2 hexagones, chaque coin est partagé par trois hexagones. Pour aller plus loin au sujet du centre, des côtes et coins, voir mon article sur les grilles (rondes, hexagonales ou triangulaires)

Dans un hexagone régulier les angles intérieurs sont de 120° . On peut compter 6 parties, chacune étant un triangle équilatéral d'angles à 60° . Chaque côté est situé à la même distance du centre, celle de la longueur d'un côté.

```
function pointy_hex_corner(center, size, i):  
    var angle_deg = 60 * i - 30°  
    var angle_rad = PI / 180 * angle_deg  
    return Point(center.x + size * cos(angle_rad),  
                center.y + size * sin(angle_rad))
```

Pour dessiner un hexagone plein, collectez les sommets du polygone de `hex_corner(..., 0)` à `hex_corner(..., 5)`. Pour dessiner les contours de l'hexagone, utilisez ces sommets puis revenez jusqu'à `hex_corner(..., 0)` en dessinant une ligne.

La différence entre les deux orientations est une rotation, et cela modifie les valeurs des angles : les angles d'hexagones plats sont 0° , 60° , 120° , 180° , 240° , 300° et ceux de pointus sont 30° , 90° , 150° , 210° , 270° , 330° .

Remarquez que les diagrammes sur cette page utilisent l'axe y orienté vers le bas (les angles augmentent dans le sens de la montre) ; vous pouvez avoir à faire quelques ajustements si l'axe y est orienté vers le haut, les angles augmentant dans le sens opposé

ANNEXE 3 - Données Personnelles

Cette documentation brève liste toute les données que nous conservons sur vous, les raisons et la nature de cette conservation.

Si vous avez une question ou une remarque, n'hésitez pas à me contacter:
hub@grauman.nelib.re

Informations personnelles collectées

- Adresse Mail
- Identifiant
- Liens d'invitations avec les autres joueurs
- Sauvegardes personnelles de jeu
- Adresses IP pour l'historique de connexion au serveur

Toute autre information concernant vos liens entre joueurs n'est pas traitée, sauf dans le cas de vos invitations aux parties, conservée tant que vous ne les validés ou rejetés pas, ou que votre compte est encore actif.

Nature du stockage

Ces informations sont conservées au sein de l'applications et ne seront **jamais** transmises à aucun tiers.

Leur durée de stockage est de **6 mois après votre dernière activité**, tant que l'application est active.

Dans le cas contraire, **vous serez notifié** en amont d'une solution **pour récupérer vos données** avant suppression.

Nature du traitement

L'adresse Mail est collectée dans le but de pouvoir vous permettre de récupérer votre compte en cas d'oubli de votre mot de passe, et de vous notifier en cas d'événement important concernant vos données. Le site ne vous transmettra pas de mails hors de ce scénario, et votre adresse mail n'est confiée à aucun un tiers.

Les liens entre joueurs servent exclusivement aux besoin du système de lobby de jeu. Il permettent au joueurs d'entrer directement en relation et d'organiser des parties de jeu. Il n'y a aucun système d matchmaking, aucun profilage, ces données ne servent qu'aux fonctionnalités suivantes:

- voir qui de vos lien est présent sur le serveur
- pouvoir les rejoindre en jeu ou leur réserver une place de votre salle.

Les sauvegardes personnelles de jeu ne sont exploitées que par vous seul.

les adresses IP sont conservées sur une période de plusieurs mois pour des raisons de sécurité, à des fins d'analyse pour vérifier à posteriori si une action frauduleuse a été commise. Ces adresses ne sont pas stockées avec vos identifiants personnels, et aucun lien n'est établi par l'application. Seul l'administrateur réseaux peut consulter ces données, et n'a accès aux identifiants d'un client que dans le cadre d'une plainte ou d'un dysfonctionnement concernant ce dernier.

Vos droits sur l'application

oubliez - moi

Sur votre page de profil se trouve un bouton "supprimer le compte". Ce dernier entraîne la suppression irréversible de vos données personnelle sur nos serveurs. Les traces que vous laisserez qui ne seraient pas détruites seront anonymisées.

Editez votre profil

Vous pouvez à tout moment depuis votre profil modifier toutes vos informations personnelles.

Restreindre votre profil

Vous pouvez demander la suspension de votre profil via l'interface de ce dernier. Il n'est alors plus visible par aucun joueur sauf vous. Sachez néanmoins que vos données seront automatiquement supprimées après une période de 6 mois d'inactivité

Exportez vos données

Si vous le souhaitez, vous pouvez à tout moment télécharger depuis votre profil vos données de sauvegardes de jeu.

Cependant, l'application ne garde actuellement aucune trace durable de vos historiques entre joueurs liés, c'est donc une information qu'elle ne peut pas vous fournir.

Anonymisation

Lors de votre passage sur le site, vous pouvez jouer de manière anonyme sans vous inscrire. Mais même via l'inscription, votre adresse Mail n'est transmise qu'à vos contacts sur votre accord, et vous n'avez aucune obligation de fournir un nom réel comme identifiant.

Le site n'étudie aucun de vos cookies autres que ceux qu'il délivre lui-même pour des raisons de sécurité. Aucune mesure de ciblage / traçage pour identifier vos actions hors site n'est en place.

Sécurité et chiffrement

Actuellement, votre mot de passe est hashé en base de donnée.

(https://fr.wikipedia.org/wiki/Fonction_de_hachage)

Vos autres données ne sont pas accessible en dehors de l'application et des règles qu'elle fixe. Si une faille venait à être exploitée et que l'équipe le découvre, vous serez notifié instamment de sa nature et des conséquences de cette exploitation.

Le site utilise le protocole TLS pour communiquer avec votre navigateur, un tunnel de chiffrement.

Recommandations de sécurité

Ce site, comme tous, n'est pas infailible, et celui ci est maintenu par une toute petite équipe. Ne stockez pas sur le site d'informations sensibles! cette application n'est pas prévue pour ca.

L'application ne vous enverra JAMAIS de mails autre que celui de confirmation au changement d'adresse et celui de récupération de compte si une demande à été faite.