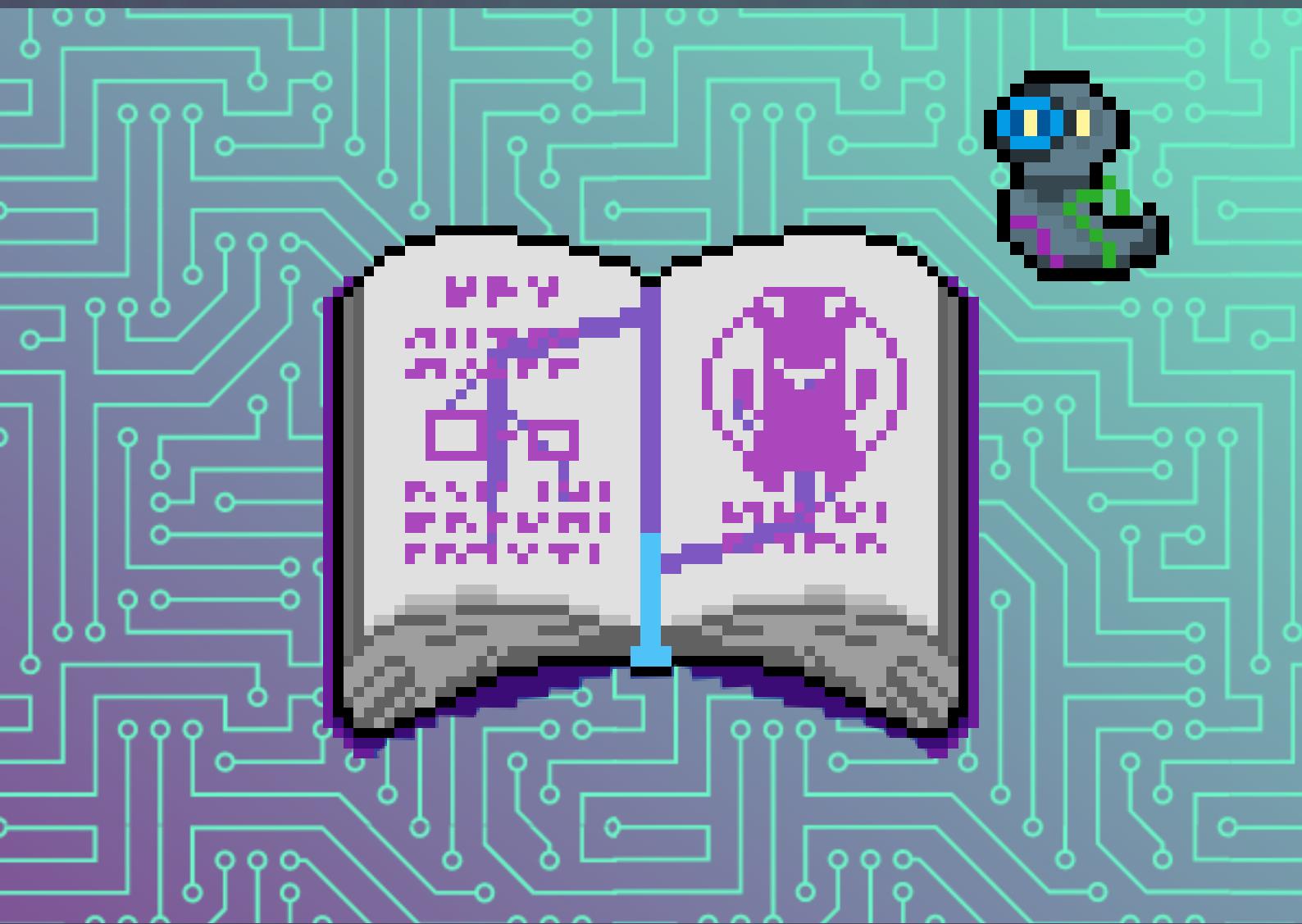


BOOKWORM IDLE GAME



Dossier de Projet

Agathe Pons

Promotion Zagreus



2021 - 2022

SOMMAIRE

A - INTRO	5
B - COMPÉTENCES DU RÉFÉRENTIEL.....	6
1/ Préambule	6
2/ Type 1 : Développer la partie front-end d'une application....	6
CP1 : Maquetter une application	6
CP2 : Réaliser une interface utilisateur web statique et adaptable.....	7
CP3 : Développer une interface utilisateur web dynamique	8
3/ Type 2 : Développer la partie back-end d'une application..	12
CP5 : Créer une base de données.....	12
CP6 : Développer les composants d'accès aux données.....	13
CP7 : Développer la partie backend d'une application web ou web mobile.....	15
C - RÉSUMÉ DE PROJET	17
D - CAHIER DES CHARGES	18
1/ Présentation du projet.....	18
2/ La conceptualisation de l'application.....	19
Les fonctionnalité du Minimum Viable Product	19
Les fonctionnalités envisagées pour une V2	21
Les rôles.....	21
Les User Stories du MVP	21
3/ La mise en place de l'application	23
Les wireframes	23
Les moodboards	25
La maquette	25
MCD - MLD - Dictionnaire des données	25
Liste des routes back (endpoints)	26
4/ La roadmap	27
E - SPÉCIFICATIONS TECHNIQUES	29

1/ Le versionning	29
2/ Les technologies du front	29
3/ Les technologies du back	30
4/ Sécurité	31
5/ Autres services utilisés	32
F - ORGANISATION DE L'ÉQUIPE.....	33
1/ Présentation de l'équipe	33
2/ Répartition des rôles et détails	33
3/ Les outils utilisés	34
4/ Organisation du travail	35
5/ Programme des sprints	37
G - PRODUCTIONS.....	40
1/ Réalisations personnelles.....	40
Service de build de save json	40
Calcul de gains hors ligne	46
SCSS	49
2/ Jeu d'essai	55
H - VEILLE ET TROUBLESHOOTING.....	58
1/ Veille	58
Cas 1 : Les mathématiques dans les idle games	58
Traduction d'une ressource en anglais	58
Cas 2 : Les JWT (Json Web Token).....	60
2/ Résolution de problèmes	60
Configuration du Swagger.....	60
Grands nombres, et limite du type integer	62
I - CONCLUSION	64



ANNEXES 65

User Stories.....	65
Wireframes.....	67
Moodboards.....	72
Maquette	74
Assets graphiques.....	75
Screenshots.....	76
Liste des endpoints.....	82
MCD.....	83
MLD	85
Dictionnaire des données.....	85
Trello.....	88
Requêtes.....	90



A - INTRO

Après cinq années passées dans l'Entreprise de Service Numérique **CGI**, tout d'abord comme **alternante** au sein du studio graphique d'une unité de digitalisation et développement d'application web, puis en CDI auprès du client Airbus comme **designeuse d'interface et intégratrice front HTML/CSS**, j'ai eu envie de franchir le pas et de devenir **développeuse web**.

C'est dans cette perspective que j'ai quitté CGI et que j'ai profité de mon chômage pour suivre la formation proposée par O'clock, **développeur fullstack JS**.

Cette formation intense de six mois prépare au passage de titre **Développeur Web et Web Mobile**, et se termine en apothéose sur la réalisation d'un **projet en groupe**. C'est ce projet que je présente aujourd'hui dans le cadre du passage de titre.



B - COMPÉTENCES DU RÉFÉRENTIEL

1/ Préambule

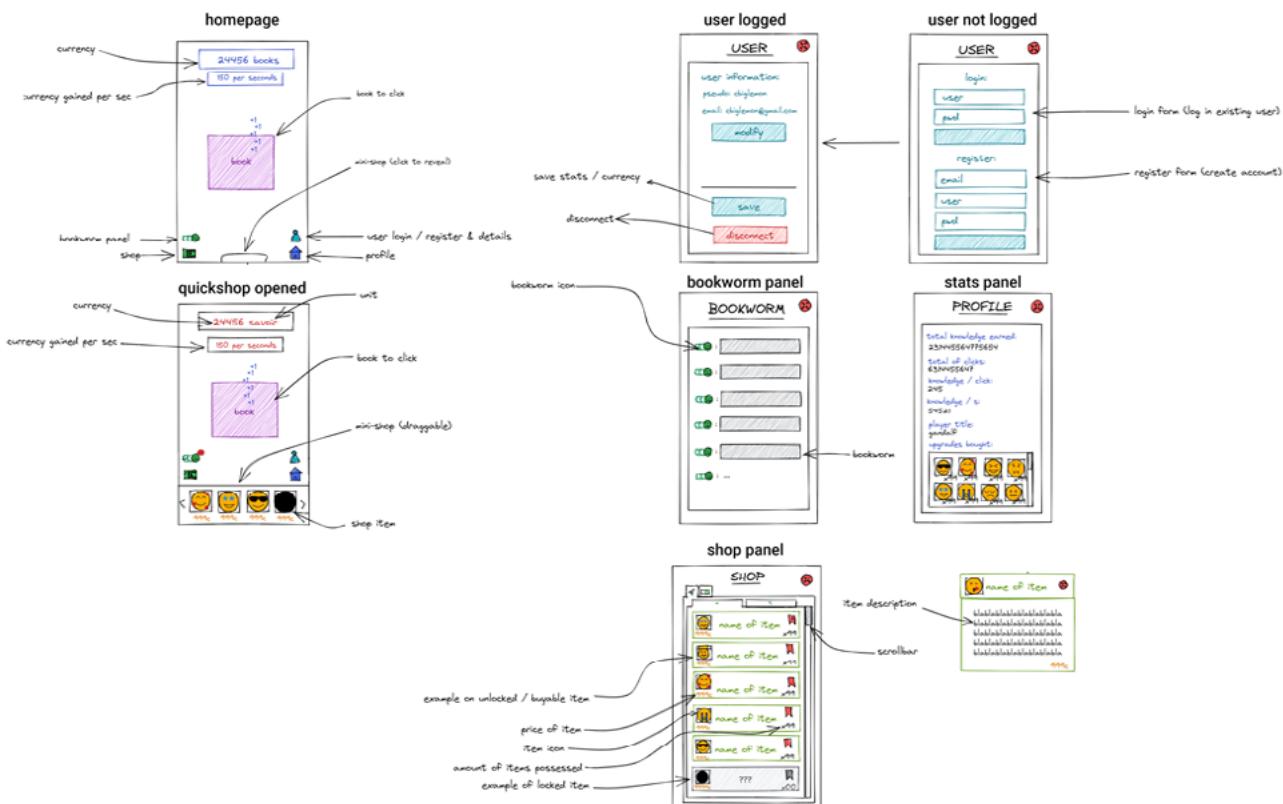
Comme il sera décrit dans les chapitres suivants, le projet **Bookworm Idle Game** couvre les compétences attendues. En voici donc un petit résumé en avant-goût, je reviendrai sur chaque point plus en détails tout au long des différents chapitres. De plus, certains documents seront fournis dans leur intégralité en annexes de ce dossier.

2/ Type 1 : Développer la partie front-end d'une application

CP1 : Maquetter une application

Le **sprint 0** du projet était un sprint de **conception**, durant lequel nous avons écrit le cahier des charges du projet Bookworm.

Nous avons donc listé les fonctionnalités de notre **MVP** (Minimum Viable Product), puis nous avons décliné ces fonctionnalités sous forme de **User Stories**. Cette étape s'est faite sous forme de discussions et de prise de décisions en groupe. Un grand nombre de fonctionnalités ont finalement été écartées (abandonnées, ou listées pour une éventuelle version 2), pour ne garder que les fonctionnalités qui nous ont semblées essentielles, et possibles à mettre en place dans le temps imparti.



Dans un second temps, nous avons réalisé plusieurs propositions de **wireframes** en version mobile, que nous avons corrigées et enrichies de manière itérative, jusqu'à validation. Puis nous avons décliné les wireframes en version desktop, en adaptant certains éléments d'interface. Les wireframes ont été réalisés avec Excalidraw.

Concernant les choix et l'orientation graphique, nous avons proposé plusieurs **moodboards** (planches de tendance) en format libre, que nous avons commenté et discuté, jusqu'à aboutir à un choix graphique. À partir de ce choix, une **maquette graphique** de l'écran principal a été réalisée en version mobile, pour donner une vision concrète de l'application, avec des éléments d'interface designés. Cependant, nous n'avons pas décliné tous les écrans wireframes en maquettes, par manque de temps. La maquette a été réalisée avec Lunacy (une alternative gratuite et compatible windows/Linux à Sketch).

Voir les annexes pour voir la totalité des documents de maquettage.

CP2 : Réaliser une interface utilisateur web statique et adaptable

L'application front est de type **SPA** (Single Page Application). L'intégration **HTML/CSS** s'est faite en **mobile-first**, c'est-à-dire décrite pour un device de type mobile, puis adaptée pour un device de type desktop grâce à l'utilisation de **media queries**.

Comme l'application front est développée avec le framework JS **React**, le HTML a été écrit dans des **composants** (des morceaux indépendants et réutilisables de code, sous forme de fonctions et qui renvoient du HTML) sous forme de **JSX** (Javascript Xml).

```
● ● ●
1  return (
2    <div className="box-position">
3      <div className={isOpen ? 'modal' : 'modal modal-hidden'}>
4        <div className="modal_header">
5          <h2 className="modal_header_title">{title}</h2>
6          <button type="button" className="modal_header_closeBtn" onClick={handleClick}>
7            <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 23.684 23.684"><path d="M23.684 1.974V5.92H21.71v1.975h-1.973v1.974v1.974h1.974v1.973v1.973h1.974v5.92H21.71v1.974h-3.946V21.71h-1.975v-1.973h-1.973v-1.974h-1.974v1.974H9.868v1.973H7.896v1.974H1.974V21.71H0v-3.946h1.974v-1.975h1.973v-1.973H5.92v-1.974H3.947V9.868H1.974V7.896H0V1.974H1.974V5.92h1.973V3.949H5.92V1.974h1.975v1.975h1.973V5.92h1.974V3.949h1.974V1.974h1.973V0h5.921v1.974h1.974zm-21.71 0V0H5.92V1.974H1.974z" /></svg>
8        </button>
9      </div>
10     <div className="modal_content">
11       <Bookworm isOpen={title === 'bookworm'} />
12       <Stats isOpen={title === 'stats'} />
13       { isDesktop ? <Info isOpen={title === 'info'} /> : <Shop isOpen={title === 'shop'} /> }
14       <User isOpen={title === 'user'} />
15     </div>
16   </div>
17 </div>
18 );
```

Concernant le **CSS**, il a été écrit en **SCSS**, un langage de script préprocesseur qui est compilé en CSS. Le SCSS permet l'utilisation d'une **syntaxe nested** (ou imbriquée), et qui pro-



pose de nombreuses fonctionnalités comme les variables, des fonctions intégrées (comme `lighten()` et `darken()` pour éclaircir ou assombrir une couleur), ou encore des mixins (des ensembles de règles réutilisables, auxquels on peut passer des arguments).

```
1  @media screen and (min-width: 1366px) {  
2      .modal {  
3          position: absolute;  
4          width: 33%;  
5          top: 1rem;  
6          bottom: 1rem;  
7          left: auto;  
8          right: 6rem;  
9          .modal_content {  
10              height: calc(100% - 70px - 3rem);  
11          }  
12      }  
13  }
```

Nous avons fait le choix de ne pas utiliser de librairie de CSS (type Bootstrap, ou MUI), le SCSS a donc entièrement été écrit par nous-même.

CP3 : Développer une interface utilisateur web dynamique

L'application front de Bookworm a été réalisée avec la librairie **React** qui a pour objectif de faciliter la création d'applications web de type **SPA** (Single Page Application).

React fonctionne avec un système de **composants** (components) qui correspondent à un composant de la page en HTML et qui est généralement écrit en **JSX** (JavaScript Xml). Le JSX est une extension React de la syntaxe javascript qui permet de structurer les composants à l'aide d'une syntaxe familière, celle du HTML.

Les composants correspondent à un élément de la page, avec son état propre, et ses actions associées, tout ça géré à l'intérieur du composant lui-même.

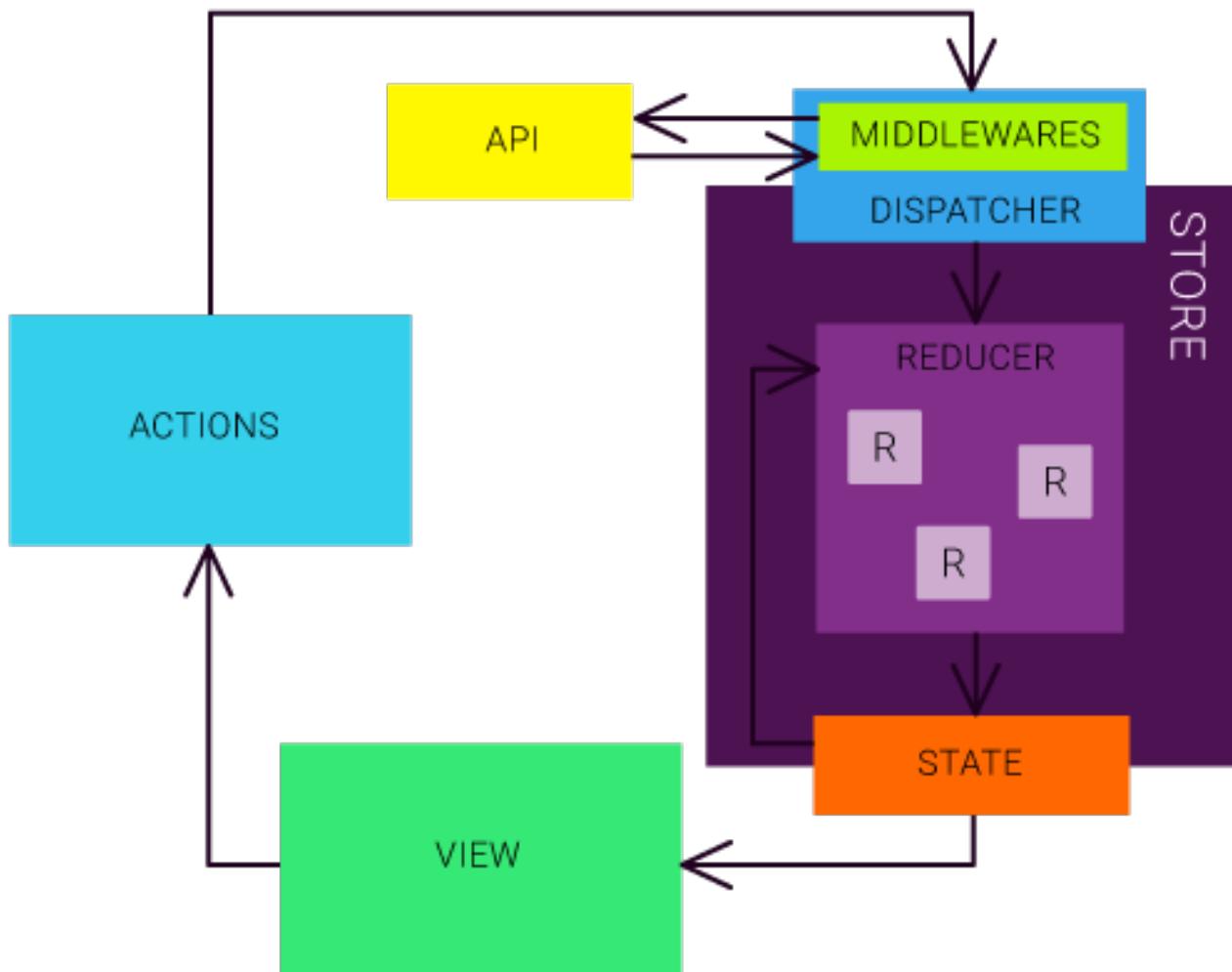
Les composants ont leur propre état, leurs propres données, sont indépendants les uns des autres, et peuvent contenir d'autres composants.

L'un des avantages de React est qu'il utilise un **DOM virtuel** pour rafraîchir uniquement les composants qui ont été modifiés et non toute la page.

Pour faciliter la gestion des **états** (states) et permettre l'utilisation de middlewares, la bibliothèque **Redux** est utilisée.



On pourrait schématiser le fonctionnement de l'application front ainsi :



Lorsqu'une action est effectuée, on passe éventuellement dans un **middleware**. C'est dans les middlewares que sont gérés les **fetch à l'API back**.

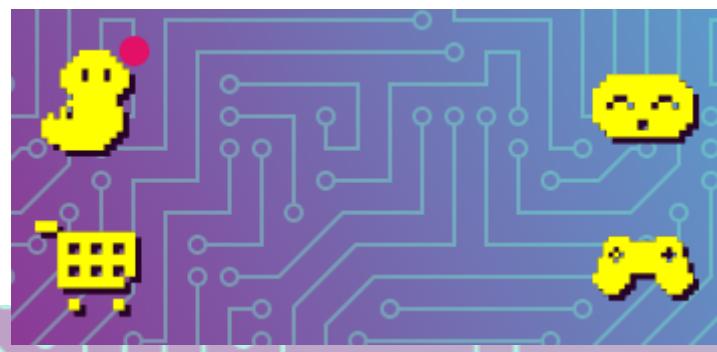
Les middleware peuvent eux-mêmes déclencher des actions.

Ensuite, le **Dispatcher** dispatche des **actions** qui pourront modifier les **Reducers**.

Chaque **Reducer** a un **état initial**, et en fonction des actions qui sont dispatchées, cet état peut changer.

Enfin, le state est renvoyé à la **vue** (view) qui affiche le résultat, avec les informations à jour, pour l'utilisateur.

Je vais prendre un exemple très simple, mais qui illustre bien cette circulation des informations.



Lorsque l'utilisateur arrive sur la page, sans même avoir besoin d'être connecté, il a une notification de message non lu du bookworm sous la forme d'une petite **pastille rouge accolée au bouton** de menu qui ouvre les messages du Bookworm.



```
1 export const initialState = {
2   messages: [],
3   newNotification: true,
4 };
```

Si l'on regarde dans les reducers, l'initialState du Bookworm initie **newNotification** à true.



```
1 <button onClick={handleClick}
2   type="button"
3   className={
4     newNotification ? 'buttons__a notification' : 'buttons__a'
5   }
6   value="bookworm">
```

Comme on peut le voir dans le **JSX du composant** qui correspond au bloc de boutons de menu, on a ici le bouton du Bookworm et on lui définit des classes avec une **ternaire** (opérateur conditionnel utilisé en raccourci de l'instruction if / else).

On regarde si newNotification est true, et si la condition est vérifiée, on applique une classe supplémentaire, la **classe notification**. C'est cette classe notification qui est utilisée comme **sélecteur css** pour afficher un **pseudo-élément** css :after qui prend la forme d'une pastille rouge.

Si newNotification est false, la classe notification n'est pas sur le bouton, la pastille rouge n'apparaît pas.

Comme l'initialState de Bookworm passe newNotification à true, **la pastille est visible lorsque l'utilisateur arrive sur l'application**.

On souhaiterait maintenant que cette pastille **disparaisse lorsque l'utilisateur ouvre la modale** du Bookworm et affiche le ou les messages de celui-ci. Sur notre bouton, on a également le **gestionnaire d'événement onClick** qui déclenche la **fonction handleClick** lorsque le bouton est cliqué.



```
1 function handleClick(e) {  
2     setTitle(e.currentTarget.value);  
3     setIsOpen(true);  
4     if (title === 'bookworm') {  
5         dispatch(readNewNotification());  
6     }  
7 }
```

Cette **fonction** est définie dans la logique du composant Buttons.

Dans un premier temps, elle met en titre de l'élément cliqué sa value, puis elle passe setI-sOpen à true ce qui aura pour effet d'afficher la modale correspondant au bouton cliqué, et enfin, si le titre de l'élément cliqué (donc sa value) est égal à bookworm alors on **dispatch** l'action **readNewNotification**.

```
1 export const readNewNotification = () => ({  
2     type: READ_NEW_NOTIFICATION,  
3 });
```

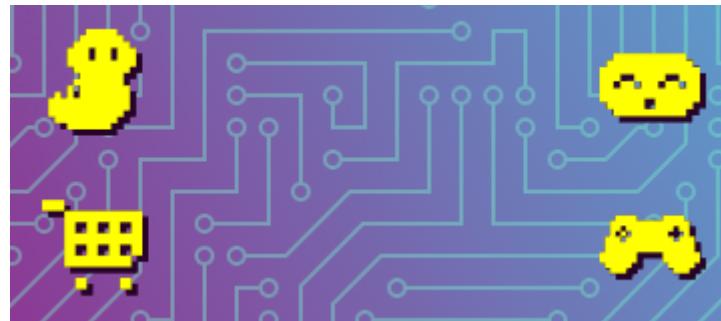
Cette action est définie dans les actions du bookworm, et c'est dans le Reducer du bookworm qu'elle va prendre effet.

```
1 const reducer = (state = initialState, action = {}) => {  
2     switch (action.type) {  
3         case READ_NEW_NOTIFICATION: {  
4             return {  
5                 ...state,  
6                 newNotification: false,  
7             };  
8         }  
9         case SAVE_FIRST_SENTENCE: {  
10            ...  
11        }  
12         case SAVE_RANDOM_SENTENCE: {  
13            ...  
14        }  
15         case RESET_BOOKWORM_STATE: {  
16            ...  
17        }  
18         default:  
19             return state;  
20     }  
21 };
```



Par l'utilisation d'un **switch javascript**, on vérifie les actions qui ont été déclenchées, ici, on tombe dans le cas READ_NEW_NOTIFICATION, qui passe newNotification à false.

Comme on l'a vu dans le composant Buttons, lorsque newNotification n'est pas true, le bouton du Bookworm n'a pas la classe notification, et donc **il n'a plus la pastille**.

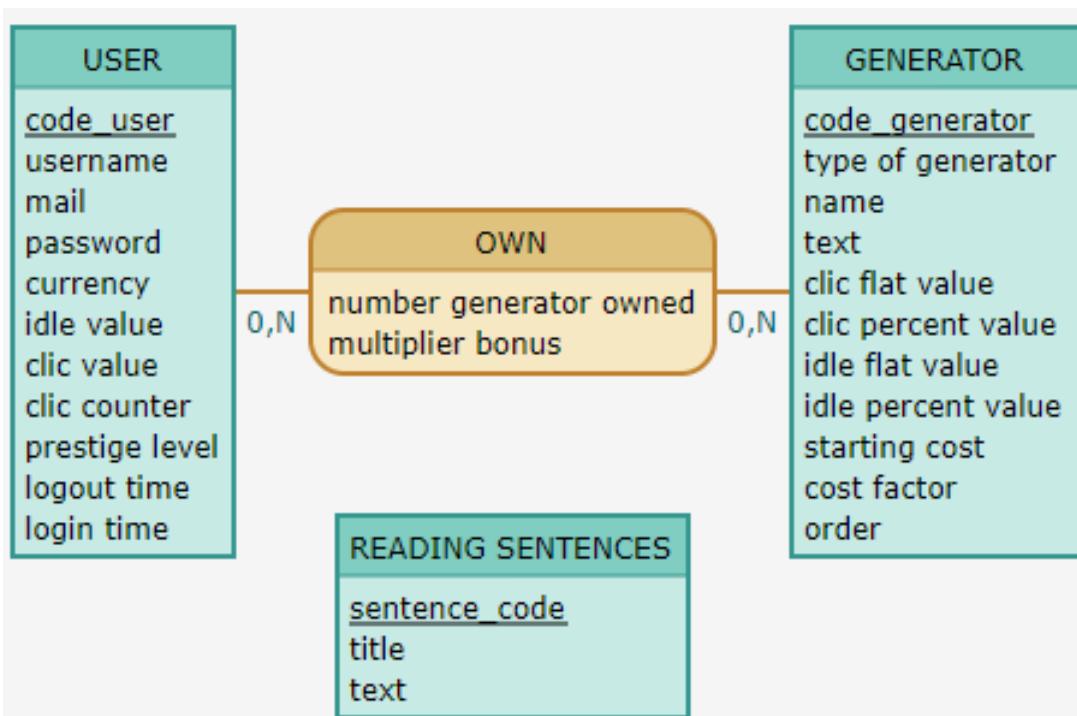


3/ Type 2 : Développer la partie back-end d'une application

CP5 : Créer une base de données

Durant le Sprint 0 du projet (phase de conception), nous avons réalisé une première version de **MCD** (Modèle Conceptuel de Données), puis après réflexion et discussions, une deuxième version, différente, beaucoup plus épurée et tout aussi fonctionnelle.

Dans la première version, nous avions une entité par type d'amélioration achetée, soit quatre entités distinctes, finalement très similaires. Nous avons fait le choix de regrouper ces quatre entités en une seule, et de différencier les items relevant de cette unique entité en fonction des informations qu'ils contiennent.



Comme on peut le voir dans cette version définitive du MCD, l'**association** entre USER et GENERATOR dispose de plusieurs **attributs**. En particulier, l'attribut number generator owned, qui sera très important dans le fonctionnement de l'application, car il permettra de savoir combien de fois le joueur a déjà acheté cet item, et de calculer par la suite le coût du prochain, car le coût d'achat des items évolue en fonction du nombre acheté.

Nous avons ensuite écrit le **MLD** (Modèle Logique de Données), et enfin, un **dictionnaire des données**. Cela nous a été très utile ensuite, lorsque nous avons écrit le script d'initialisation de la base de données pour créer le **MPD** (Modèle Physique des Données).

Pour la mise en place de la base de données, nous avons mis en place **Sqitch**, un outil de versionning pour gérer les migrations sur la base de données de type **DDL** (Data Definition Language). Sqitch permet de déployer des migrations suivant un plan, de vérifier le bon déroulement du déploiement, et de revenir sur une migration.

Cela nous a été très utile, car nous avons réalisé plusieurs **migrations** durant la phase de développement, pour mettre en place des domaines, et réaliser des modifications mineures, comme changer le type d'une colonne par exemple.

Enfin, nous avons écrit dans un fichier SQL à part, un **script de seeding** pour insérer les données initiales du jeu.

Voir les annexes pour voir la totalité des documents de conception de la base de données.

CP6 : Développer les composants d'accès aux données

L'**API back** du projet Bookworm Idle Game est développée selon le design **pattern MVC** (Model View Controller), l'aspect View n'étant pas géré ici vu qu'il s'agit d'une API qui met à disposition les données, mais qui ne gère pas la présentation de ces données (cet aspect étant géré dans l'application front en React).

```
 1  const { Pool } = require('pg');
 2
 3  const config = {
 4    connectionString: process.env.DATABASE_URL,
 5  };
 6
 7  const pool = new Pool(config);
 8
 9  module.exports = {
10   originalClient: pool,
11   async query(...params) {
12     debug(...params);
13     return this.originalClient.query(...params);
14   },
15 }
```



La connexion à la base de données **PostgreSQL** se fait dans un fichier de configuration. Nous commençons par importer le module Pool de pg, on l'initialise en lui passant une string de connexion à la base de donnée en config. Pour des raisons de **sécurité**, cette string est définie comme **variable d'environnement** dans un fichier .env .

Dans un dossier models, plusieurs fichiers décrivent différentes **méthodes** qui lancent des requêtes sur la base de données. Par l'utilisation du pattern **Data Mapper**, nous avons écrit les requêtes SQL très spécifiques dont nous avions besoin.

Pour le cas des sentences, qui est le plus simple et le plus basique, nous avons donc deux méthodes, une pour retourner toutes les sentences, sauf celle ayant l'id 1, et une méthode pour retourner une sentence par son id. De plus, nous avons écrit dans chaque fichier, pour chaque modèle, de la **JSDoc**, utile pour le **Swagger** que nous avons mis en place.



```
1 const sentenceDataMapper = {
2   async findAll() {
3     debug('findAll called');
4     const result = await client.query('SELECT * FROM sentences WHERE sentences.id NOT IN (1)');
5     return result.rows;
6   },
7   async findOne(id) {
8     debug(`findOne called for id ${id}`);
9     const result = await client.query('SELECT * FROM sentences WHERE id = $1', [id]);
10    if (result.rowCount === 0) {
11      return null;
12    }
13    return result.rows[0];
14  },
15};
```

Ces **modèles** sont exportés, et utilisés dans des **contrôleurs**.



```
1 async getRandom(_req, res) {
2   debug('getRandom called');
3   const sentences = await sentenceDataMapper.findAll();
4   const randomSentence = sentences[Math.floor(Math.random() * sentences.length)];
5   return res.json(randomSentence);
6 },
7 async getFirst(_req, res) {
8   debug('getFirst called');
9   const firstSentence = await sentenceDataMapper.findOne(1);
10  return res.json(firstSentence);
11},
```



Pour reprendre l'exemple des sentences, nous avons ici différentes **méthodes de contrôleur** qui font appel au modèle de sentence pour retourner au format json :

- Une sentence aléatoire parmi toutes les sentences sauf la première
- La première sentence

CP7 : Développer la partie backend d'une application web ou web mobile

Pour l'**API Back** du projet Bookworm, nous sommes sur un design pattern **MVC**, comme expliqué dans la partie précédente.

Pour commencer, nous avons un dossier routers, dans lequel différents fichiers vont définir les différentes **routes back**, listées dans le cahier des charges.

```
● ● ●
1 router
2   .route('/')
3   /**
4     * GET /api/sentence
5     * @summary Get a random sentence
6     * @tags Sentence
7     * @security BearerAuth
8     * @return {Sentence} 200 - success response - application/json
9   */
10  .get(controllerHandler(checkLogin.checkLogin), controllerHandler(controller.getRandom));
11
12 router
13   .route('/first')
14   /**
15     * GET /api/sentence/first
16     * @summary Get the first sentence
17     * @tags Sentence
18     * @return {Sentence} 200 - success response - application/json
19   */
20  .get(controllerHandler(controller.getFirst));
```

Par exemple, voici les routes concernant les sentences.

La première route passe tout d'abord dans un **middleware** checkLogin, qui vérifie la présence et la validité d'un **JWT** (Json Web Token), car il s'agit d'une route protégée, et l'utilisateur doit être identifié pour y accéder. Une fois le middleware passé, la route appelle la méthode de contrôleur getRandom, qui elle-même va chercher dans les modèles la méthode



findAll qui est une requête sur la base de données et qui retourne des sentences (comme expliqué plus en détail dans le paragraphe précédent).

Comme on peut le voir, le middleware et la méthode de contrôleur sont passés en argument du controllerHandler, qui est un **helper** qui nous permet de wrapper middlewares et contrôleurs dans un try / catch. Enfin, on peut trouver ici aussi de la **JSDoc**, utile pour décrire les routes que l'on souhaite mettre à disposition dans le **Swagger**.

Enfin, on peut trouver dans le dossier services un fichier buildSave, qui est un **service** appelé dans plusieurs méthodes de contrôleur. Ce service sert à récupérer via les requêtes définies dans les modèles les données relatives à la partie d'un joueur, à les manipuler et effectuer différents calculs, pour finalement renvoyer un Json complex qui fait office de "**fichier de sauvegarde de partie**" contenant toutes les informations dont le front à besoin pour permettre au joueur de jouer au jeu et d'effectuer des actions.



C - RÉSUMÉ DE PROJET

Bookworm Idle Game est un projet dont j'ai eu l'idée, et que j'ai proposé dans le cadre de la fin de ma formation. Il a été retenu, et un groupe a été formé par l'équipe pédagogique, en fonction des souhaits exprimés par chacun.

C'est un projet qui regroupe deux de mes passions : le **jeu vidéo** et la **lecture**.

Il se divise en **deux sous-projets** :

Une **API back** qui propose une documentation qui respecte le standard **OpenAPI** sous la forme d'un **Swagger** et qui met à disposition les différentes routes back du jeu.

Une **application front** en **React** supportée par **Redux**, et qui communique avec l'API via **Axios**.

Le but de ce projet était de se **confronter au développement d'un projet** complet et complexe sur une courte période (quatre semaines, phase de conception et présentation finale comprises), de **s'organiser en groupe**, de **mettre en pratique** les compétences techniques abordées durant la formation, de **se confronter à des problématiques** non abordées en cours et de **les solutionner** par nous-même, et bien sûr, de nous faire **plaisir** sur un projet choisi et qui nous plaît !



D - CAHIER DES CHARGES

1/ Présentation du projet

Le projet

Développer un **idle game** (jeu incrémental) sur la thématique du **livre**.

Idle game, kesako?

Le jeu incrémental, ou *idle game* (littéralement « jeu inactif »), ou encore *clicker game* est un genre de jeux vidéo dont la mécanique de jeu principale consiste en de simples actions à l'interactivité volontairement limitée : il peut s'agir de cliquer à répétition ou bien de simplement laisser le jeu s'exécuter, le « joueur » n'ayant alors rien à faire. Le plus souvent, cette mécanique de jeu entraîne une progression monétaire virtuelle permettant d'améliorer certaines caractéristiques afin d'accélérer encore une fois l'acquisition monétaire. Ce genre, apparu au début des années 2000, s'est popularisé avec le succès de *Cookie Clicker* en 2013 et s'est largement répandu au sein des jeux sur navigateur.

[Définition de Wikipédia]

Thématique

Dans un idle game, on doit accumuler le plus possible d'une unité (\$, or, cookies, ...), ici, on veut lire des livres, et on a un «**bookworm**» (rat de bibliothèque) qui va lire pour nous, et donc on va accumuler les livres lus. Ces livres seront convertis en savoir qui pourra être réinvesti pour acheter des améliorations.

Dans un idle game, les interactions sont en fait très limitées, il s'agit généralement d'attendre que le jeu «joue tout seul» en auto-incrémantation.

Alors pendant que notre bookworm dévore les livres à toute vitesse, pourquoi ne pas en faire autant de votre côté ? Le jeu proposera à minima quelques incentives à la lecture (livres, bd, mangas, ...), voire plus selon les features retenues.

L'idée est de faire se rencontrer deux univers dont on aurait tendance à penser que tout oppose, et de les mettre l'un au service de l'autre.

Le besoin

Au quotidien, nous sommes nombreux à passer du temps sur notre smartphone, à scrolller sur les réseaux sociaux, à jouer à des petits jeux sans grand intérêt... Parfois sans se rendre compte du temps passé (et perdu ?) ainsi.



L'objectif

L'idée du projet et de s'approprier le média du jeu incrémental, et de le mettre au service de notre cause :

Inciter l'utilisateur à laisser son smartphone de côté quelque temps pour prendre un livre, car le jeu travaillera pour lui en son absence. Les quelques minutes de lecture qu'il se sera accordées seront récompensées lorsqu'il reviendra sur le jeu plus tard.

La cible

Le jeu s'adresse aux personnes qui aiment jouer sur smartphone ou navigateur, en particulier ceux qui ont tendance à y passer trop de temps.

Le jeu cible toutes les personnes qui aiment passer un peu de temps sur un jeu qui s'annonce chronophage, mais qui donne la possibilité de ne pas l'être. La thématique se veut accessible et ouverte à tous les joueurs souhaitant jouer un peu, mais ne pas y passer trop de temps avec la satisfaction à la reconnexion d'une progression constante et efficace.

2/ La conceptualisation de l'application

Durant la phase de **brainstorming** pour décider des fonctionnalités, un très grand nombre d'idées intéressantes ont été proposées. Malheureusement, les délais de réalisation de ce projet étant très courts, nous avons dû sélectionner les fonctionnalités qui nous semblaient essentielles pour que le jeu soit à la fois fonctionnel, mais aussi le plus agréable et amusant possible.

Nous avons donc listé les fonctionnalités du **MVP** (Minimum Viable Product), et nous avons retenu une liste de fonctionnalités pour une V2 potentielle.

Les fonctionnalités du Minimum Viable Product

Pour le visiteur et le joueur

Afficher des informations (présentation de l'équipe, remerciements) → version desktop uniquement

Afficher les valeurs des deux caractéristiques principales :

- Combien de savoir l'utilisateur possède
- Combien l'utilisateur gagne de savoir par seconde (à zéro pour le visiteur)

Pour le visiteur

Afficher le message du Bookworm invitant le visiteur à se connecter



Afficher un formulaire permettant au visiteur de s'inscrire ou de se connecter

Afficher les informations de partie, qui sont à 0

Afficher un shop vide

Pour le joueur

Voir les informations de compte

Sauvegarder manuellement sa partie

Se déconnecter

Supprimer son compte

Clic : cliquer sur le livre pour incrémenter son savoir de x

Idle : incrémenter son savoir automatiquement de x/s

Acheter des améliorations pour augmenter la valeur du clic de x (flat)

Acheter des améliorations pour augmenter la valeur du clic de x%

Acheter des améliorations pour augmenter la valeur du idle de x (flat)

Acheter des améliorations pour augmenter la valeur du idle de x%

Afficher le shop, avec les améliorations réparties selon leur type :

- Amélioration du clic en flat (+x)
- Amélioration du clic en pourcentage (+x%)
- Amélioration du idle en flat (+x)
- Amélioration du idle en pourcentage (+x%)

Afficher les améliorations dans le shop selon l'ordre de dévoilement et selon si le joueur a suffisamment de savoir pour les acheter

Afficher les informations de partie :

- Nombre de clics effectués depuis le début de la partie
- Quantité de savoir gagnée par clic
- Quantité de savoir gagnée par seconde
- Liste des différentes améliorations achetées, et leur nombre

Afficher les détails d'une amélioration dans le shop et dans les informations de partie :

- Son nom et sa description
- Son coût de base, et son coût actuel (selon le nombre déjà possédé)
- La valeur de bonus de base, et les bonus déjà appliqués (selon le nombre déjà possédé)

Afficher le *quickshop*, une version alternative du shop adaptée aux petits écrans, et qui n'affiche que les améliorations actuellement achetables → version mobile uniquement

Afficher les messages du Bookworm (messages invitant le joueur à aller lire des livres ou message mystère dévoilant le konami code)

Faire une sauvegarde automatique de partie toutes les 60 secondes



Exécuter un **Konami Code** (combinaison de touches) pour avoir une surprise (animation colorée à l'écran) ➔ version desktop uniquement

Les fonctionnalités envisagées pour une V2

Les fonctionnalités de idle game :

- Pouvoir améliorer les **améliorations** (augmenter les valeurs de bonus accordées)
- Pouvoir avoir plusieurs **Bookworms** (multiplier la valeur du idle)
- Pouvoir acheter plusieurs **Clics** (multiplier la valeur du clic)
- Mettre en place la mécanique **New Game+** : La partie est entièrement remise à 0, mais avec des bonus permanents appliqués à ses valeurs clic et de idle, permettant d'aller encore plus vite encore plus loin dans la nouvelle partie
- Validation d'**achievements** : Faire gagner des badges d'achievement selon les milestones atteintes au cours de la partie

Les fonctionnalités expérience utilisateur :

- Pouvoir exporter / importer un fichier de sauvegarde
- Permettre au visiteur non inscrit de commencer à jouer et de s'inscrire plus tard, sans perdre la progression accomplie avant l'inscription

Les rôles

Le visiteur (utilisateur non connecté)

- Peut cliquer pour lire un livre
- Reçoit un message du Bookworm qui l'informe qu'il doit se connecter ou s'inscrire pour profiter des fonctionnalités et sauvegarder sa partie
- Peut créer un compte
- Peut se connecter

Le joueur (utilisateur connecté)

- Peut profiter de toutes les fonctionnalités du jeu

Les User Stories du MVP



En tant que	Je veux pouvoir	Afin de
guest / player	afficher le jeu sur mobile ou bureau	varier les plateformes sur lesquelles je veux m'amuser
guest / player	afficher des informations sur le projet	lire la présentation de l'équipe, les remerciements
guest / player	voir les stats principales	savoir combien je gagne de savoir et combien j'en ai accumulé.
guest	accéder au jeu sur la page d'accueil	pouvoir cliquer pour lire des livres et accumuler du savoir (limité à 1)
guest	cliquer sur un bouton "register" pour avoir un formulaire d'inscription	créer un compte
guest	cliquer sur un bouton "login" pour avoir un formulaire de connexion	me connecter à mon compte et reprendre ma partie
guest	voir un message m'avertissant que ma partie sera limitée et ne sera pas sauvegardée si je ne suis pas connecté	ne pas être pris au dépourvu, pouvoir sauvegarder ma partie et avoir accès à tout le jeu
player	sauvegarder manuellement	enregistrer ma progression
player	cliquer sur un bouton "account"	visualiser les informations de mon compte
player	cliquer sur un bouton "logout"	me déconnecter de mon compte
player	cliquer sur un bouton "delete account"	supprimer mon compte
player	cliquer pour accumuler du savoir	accumuler du savoir à dépenser pour le shop
player	accumuler du savoir en automatique	accumuler du savoir à dépenser pour le shop
player	cliquer sur un bouton "shop" (mobile)	pouvoir afficher les améliorations disponibles
player	voir facilement quels items sont achetables	prioriser mes achats
player	savoir de quel type d'item il s'agit (idle/ click / flat/ pourcent)	ne pas avoir à lire les détails d'un item pour savoir de quel type il s'agit
player	Voir apparaître en grisé avec des ??? le prochain item débloquable avec son coût	avoir un objectif et donner l'envie



En tant que	Je veux pouvoir	Afin de
player	cliquer sur un item du shop	pour acheter l'item et améliorer mes stats, accumuler plus de savoir
player	avoir un shop rapide disponible à l'écran principal avec un carrousel des achats disponibles (mobile)	acheter rapidement des items sans avoir à ouvrir la fenêtre du shop
player	recevoir du savoir lorsque je me reconnecte	recevoir le total des savoir gagné pendant le temps de déconnexion
player	cliquer sur un bouton statistiques	afficher les statistiques de ma partie (nombre de clics, savoir gagné, items achetés...)
player	cliquer sur un bouton pour voir les détails d'un item depuis le shop	connaître les effets de l'item et voir la description de l'item
player	cliquer sur un bouton pour voir les détails d'un item depuis les statistiques	connaître les effets de l'item et voir la description de l'item
player	recevoir des messages d'incitation à la lecture	quitter le jeu pour lire un livre
player	avoir ma partie sauvegardée automatiquement régulièrement	ne pas perdre ma progression si je n'ai pas sauvegardé manuellement
player	avoir ma partie sauvegardée à chaque fois que je fais une action importante (achat d'un item dans le shop)	ne pas perdre ma progression si je n'ai pas sauvegardé manuellement
player	exécuter une combinaison de touches de type Konami Code	avoir une surprise rigolote à l'écran (animation colorée)

3/ La mise en place de l'application

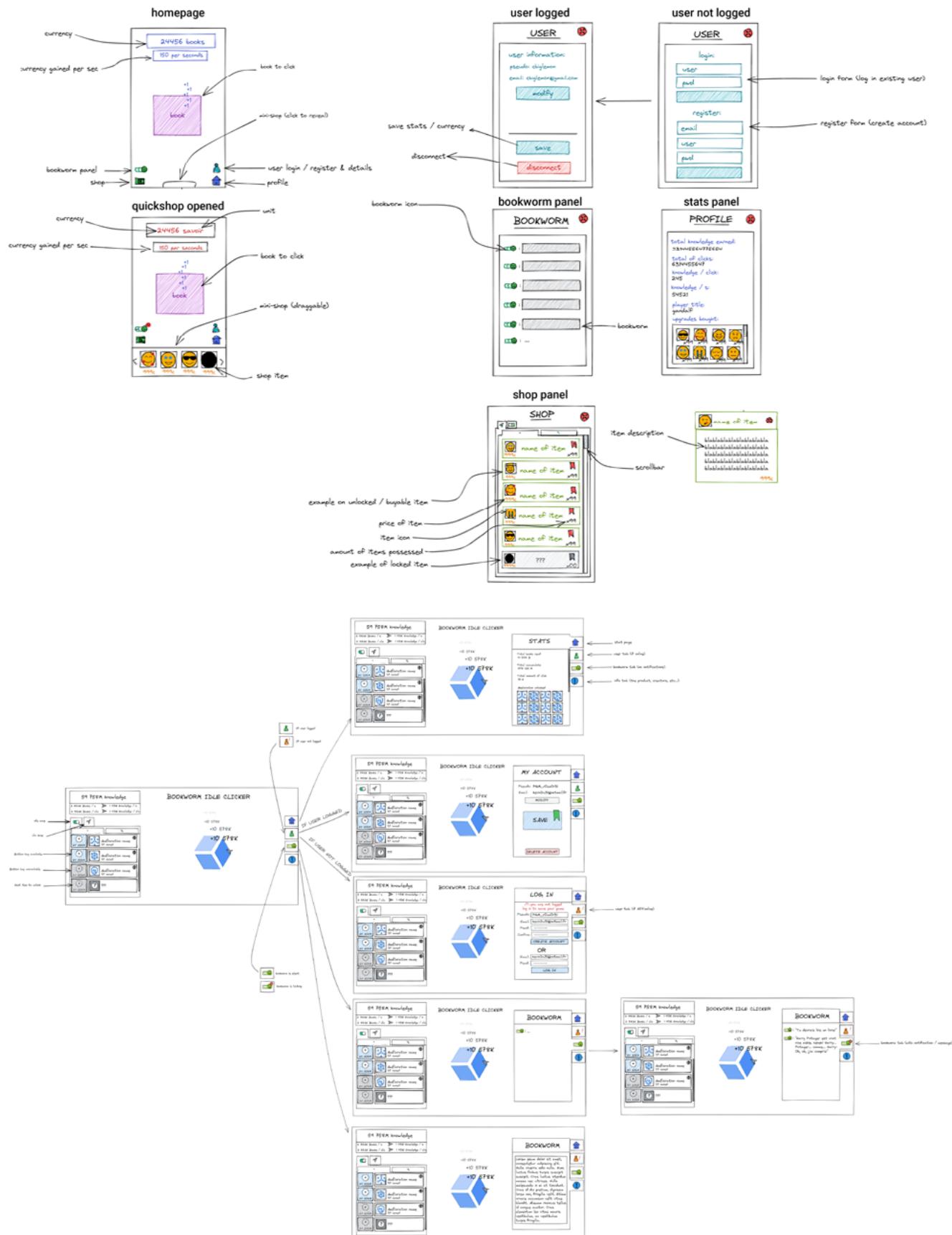
Les wireframes

Le jeu Bookworm Idle Game étant une **SPA** (Single Page Application), nous n'avons pas fait **d'arborescence**, et nous n'avions donc qu'un écran à maquetter, mais avec de nombreuses variations en fonction des différents éléments d'interface affichés ou non à l'écran.

Nous avons travaillé en **mobile-first** dès la phase de conception du projet, c'est pourquoi nous avons d'abord réalisé tous les wireframes de la version mobile. Une fois les wireframes



mobiles validés, nous les avons décliné en version desktop.

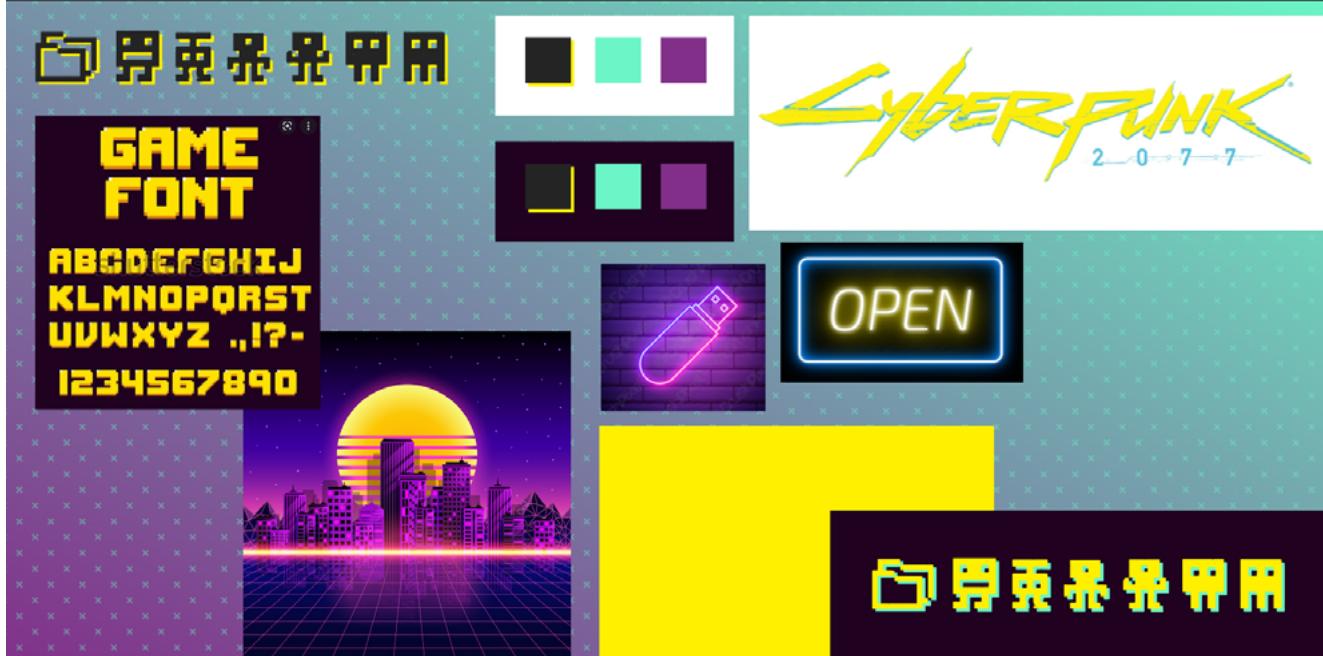


Voir les annexes pour voir la totalité des wireframes.



Les moodboards

Une fois les wireframes validés et avant de passer au maquettage graphique, nous avons fait le choix de proposer quelques **moodboards**, ou planches d'ambiance, pour voir et comparer nos idées et envies en termes d'ambiance graphique.



Voir les annexes pour voir la totalité des moodboards.

La maquette

Après avoir validé une orientation graphique grâce aux moodboards, nous avons réalisé une maquette graphique de l'écran d'accueil pour un utilisateur connecté, pour nous donner une vision concrète de ce à quoi ressemblera le jeu. Cependant, par manque de temps et de compétences en graphisme dans l'équipe, nous n'avons pas décliné tous les wireframes en maquettes graphiques.

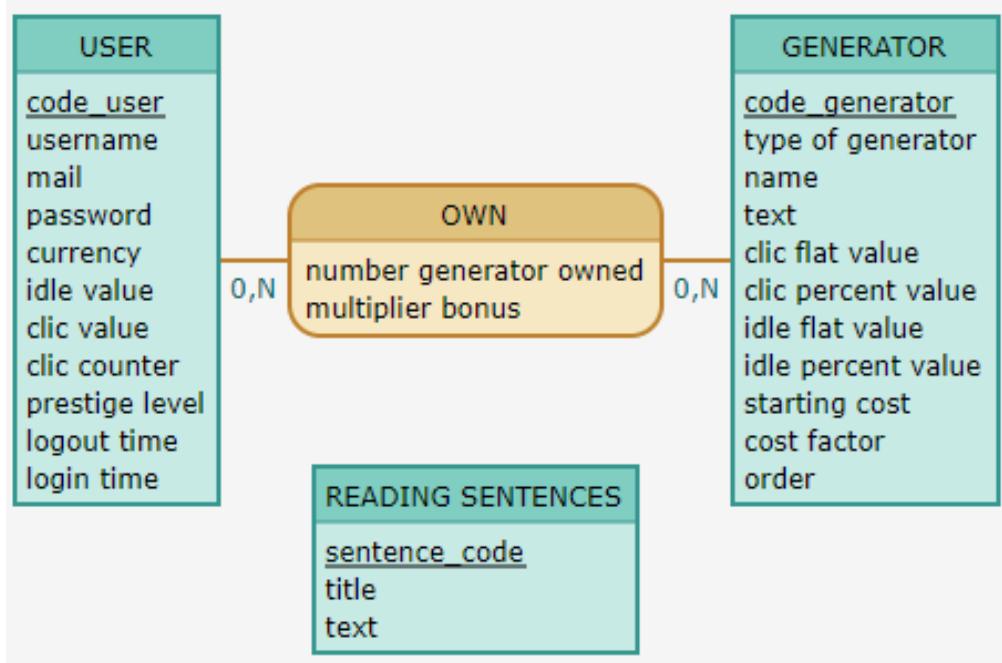
MCD - MLD - Dictionnaire des données

Dans un document annexe du cahier des charges, nous avons réalisé les différentes étapes de la **conceptualisation de la base de données**.

Nous avons réalisé une première version de **MCD** (Modèle Conceptuel de Données), puis après réflexion et discussions, une deuxième version différente, beaucoup plus épurée et tout aussi fonctionnelle.

Une fois le **MCD** validé, nous nous sommes appuyés dessus pour écrire le **MLD**, puis le **dictionnaire des données**.





Voir les annexes pour voir ces différents documents dans leur entièreté.

Liste des routes back (endpoints)

En nous appuyant sur la liste des fonctionnalités et des User Stories, nous avons enfin listé tous les **endpoints** (les routes back) qu'il nous faudrait développer dans l'**API back** pour répondre aux besoins de l'application.

Endpoint	Type	Data
endpoint/api/playerAccount	POST	Formulaire d'inscription : username, email, password et password confirmation, retourne le json de partie (nouvelle partie) du player
endpoint/api/playerAccount	PATCH	Formulaire de modification de compte : username, et/ou email, et/ou password et password confirmation, retourne le username et le mail Route protégée par bearer authentification
endpoint/api/playerAccount	DELETE	Suppression du compte utilisateur (table d'association player_owns_generator et table player) Route protégée par bearer authentification
endpoint/api/login	POST	Formulaire de login : email et password, retourne le json de partie du player



Endpoint	Type	Data
endpoint/api/disconnect	PATCH	Bouton disconnect : Envoie les informations du user mises à jour en front (currency + clic counter) Détruire le token Route protégée par bearer authentification
endpoint/api/save/	PATCH	Envoie les informations du user mises à jour en front (currency + clic counter), retourne le json de partie du player Route protégée par bearer authentification
endpoint/api/save/item/:id	POST	Si item pas déjà possédé : ajout de l'item dans la table d'association + update sur currency et clic counter, retourne le json de partie du player Route protégée par bearer authentification
endpoint/api/save/item/:id	PATCH	Si item déjà possédé au moins une fois, mise à jour du nombre d'items + update sur currency et clic counter, retourne le json de partie du player Route protégée par bearer authentification
endpoint/api/sentence	GET	Envoie une sentence au hasard parmi toutes sauf la première
endpoint/api/sentence/first	GET	Envoie la première sentence

4/ La roadmap

Pour avoir une vision du déroulement du projet et situer les grandes **milestones** (jalons) du projet, nous avons réalisé une roadmap sous la forme d'un petit tableau.



Sprint 0	Sprint 1
<p>Conception</p> <p>rédaction cahier des charges</p> <p>docs de conception de la base de données</p> <p>wireframes + identité graphique</p> <p>Mise en place des repo github</p>	<p>Développement</p> <p>Front :</p> <ul style="list-style-type: none"> Mise en place React app Création premiers composants Création interface statique Implémentation fonctionnalités clic et idle <p>Back :</p> <ul style="list-style-type: none"> Mise en place architecture API MVC Mise en place Swagger Écriture d'au moins une première route (sentences) Écriture des requêtes SQL Faire un POC pour les JWT <p>Front/back :</p> <ul style="list-style-type: none"> Connecter front et back Tester un fetch sur une route

Sprint 2	Sprint 3
<p>Développement</p> <p>Priorité si pas déjà fait :</p> <ul style="list-style-type: none"> Connecter front et back Tester un fetch sur une route <p>Front :</p> <ul style="list-style-type: none"> Finir d'implémenter les fonctionnalités en front Faire le SCSS Faire le RWD <p>Back :</p> <ul style="list-style-type: none"> Finir d'implémenter toutes les routes Implémenter les JWT Finir la JSDoc 	<p>Ajustement / Présentation</p> <p>Tests</p> <p>Fix des bugs</p> <p>Déploiement</p> <p>Oral blanc</p> <p>Présentation</p>



E - SPÉCIFICATIONS TECHNIQUES

1/ Le versionning



Afin de conserver un historique du développement de notre jeu, nous avons utilisé le **VCS** (logiciel de gestion de versions, ou Version Control System) **Git**, et le service d'hébergement et de versionning **GitHub**, pour héberger nos repos (repository, ou dépôts).

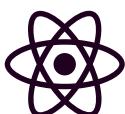
Comme le projet se divise en deux applications, nous avons fait le choix de créer deux repos distincts : **bookworm-back-API** pour l'API back, et **bookworm-front-app** pour le front.

Pour les deux repos, nous avons mis en place un process de travail identique :

- Une **branche master** correspondant à la version de production, c'est-à-dire une version fonctionnelle de l'application, et qui est la version déployée et accessible aux utilisateurs une fois le premier déploiement effectué.
- Une **branche develop** qui correspond à la version de développement, et qui n'est pas déployée. La branche develop doit toujours pouvoir se lancer sans planter, mais elle peut se trouver dans un état qui contient des bugs ou des features pas parfaitement et entièrement implémentées. Lorsque la branche develop est fonctionnelle et qu'un certain nombre de features ont été implémentées et testées, l'équipe se regroupe pour valider le merge de develop sur master pour passer la version de développement en production (généralement en fin de sprint).
- Des **branches de features** qui sont tirées depuis develop par un développeur pour travailler. Si un avancement notable, ou si le développement de la feature est terminé, la branche est mergée sur develop, et supprimée si elle n'a plus lieu d'être.

2/ Les technologies du front

Les principales technologies utilisées pour la réalisation de l'**application front** sont les suivantes :



React comme **framework** Javascript dont le but principal est de faciliter la création de **SPA** (Single Page Application). C'est une librairie Javascript libre développée par Facebook, qui utilise un **DOM** virtuel et permet de créer et de réutiliser des **composants** (des morceaux indépendants et réutilisables de code, sous forme de fonctions et qui renvoient du HTML).





Redux est utilisé avec React. C'est une librairie Javascript open-source de **gestion d'état** (state) pour des applications web.



Axios est utilisé comme **client HTTP** pour gérer les **fetchs** et la communication avec l'API back. Il est basé sur les **promesses** (promises). Il fonctionne avec tous les navigateurs ainsi qu'avec NodeJS.



Babel nous permet d'assurer la **compatibilité du code** Javascript avec tous les navigateurs. C'est un **trans-compiler** de Javascript. La rédaction du code sous React se faisant sous **ECMAScript**, Babel traduit ce code en un code Javascript compréhensible par tous les navigateurs.



Webpack nous permet de faciliter les processus de travail sur l'application. C'est un **module-bundler** (groupeur de modules) capable de réaliser un certain nombre de tâches, comme la gestion des dépendances, la compilation de code, ou encore faciliter le déploiement.



Sass est utilisé pour la gestion des feuilles de style **CSS**. C'est un langage de script préprocesseur qui est compilé en CSS et qui nous permet d'écrire en **SCSS**, ce qui nous donne accès à une **syntaxe nested** (ou imbriquée) et de nombreuses fonctionnalités comme des fonctions intégrées ou des mixins.

3/ Les technologies du back

Les principales technologies utilisées pour la réalisation de l'**API back** sont les suivantes :



PostgreSQL pour la **base de données relationnelles** (SGBDR, Système de Gestion de Base de Données Relationnelles). C'est un outil libre, comparable à d'autres systèmes de gestion de base de données comme Oracle ou MySQL.



Sqitch pour le versioning et la gestion des migrations sur la base de données. Sqitch est un outil de versioning pour gérer les migrations sur la base de données de type **DDL** (Data Definition Language). Sqitch permet de déployer des migrations suivant un plan, de vérifier le bon déroulement du déploiement, et de revenir sur une migration.



Node.js comme plateforme de serveur web permettant l'exécution de Javascript côté serveur



Express comme framework pour le développement de serveur en NodeJS. C'est le framework standard pour construire des applications web basées sur NodeJS.





JSDoc-Swagger pour **documenter l'API** et mettre à disposition une interface permettant de tester les différents endpoints de l'application. Swagger est un langage descriptif qui permet de décrire des APIs RESTful exprimées en Json. JSDoc-Swagger extrait les informations contenues dans la JSDoc et génère une interface disponible au endpoint passé en configuration, qui documente l'API et permet de tester les routes. Il dispose de nombreuses options de configuration, notamment la possibilité de gérer différents types d'authentification pour pouvoir tester les routes même si celles-ci sont protégées.

4/ Sécurité

Concernant la **sécurité**, plusieurs outils et technologies ont été mis en place.



Étant donné que **la plupart des routes de l'API sont protégées** et nécessite que l'utilisateur soit authentifié comme joueur sur l'application, nous avons utilisé les **JWT** (Json Web Token) pour permettre l'échange sécurisé de **tokens** (jetons) uniques entre front et back pour authentifier le joueur, et lui renvoyer les informations de compte et de partie correspondant à son identité. Les JWT reposent sur l'authentification **Bearer**, et le token est exposé via le **header** Authorization.



Le stockage des **mots de passe** des utilisateurs dans la base de données ne se fait pas en clair, pour des raisons de sécurité évidentes. C'est pourquoi nous avons utilisé le package **Bcrypt**, qui permet d'utiliser la fonction de hachage (hash) bcrypt pour chiffrer les mots de passe.

Pour éviter que des **data non conformes**, voire **malveillantes** soient envoyées à la base de données, risquant de la compromettre, plusieurs **sécurités** et **vérifications** ont été mises en place.

Tout d'abord, afin de d'empêcher une **injection SQL**, les requêtes sont **préparées** : les variables ne sont pas directement écrites dans la requête, elles sont stockées dans un array, et sont appelées avec la notation de substitution \$1, \$2 ...dans la requête.

De plus, une **première vérification** de la conformité des informations saisies par l'utilisateur est effectuée dans l'API back (avant requête sur la base de donnée), via le package **Joi**, qui permet de décrire dans des **schemas** les différentes **règles de validation** entre autres grâce à l'utilisation de **regex** (expressions régulières).

Enfin, une **deuxième vérification** est effectuée du côté de la base de données (après requête grâce à la mise en place de **domains** et de check sur la base de données). Si les datas à insérer ou modifier ne correspondent pas au format attendu, la requête est rejetée.

Enfin, les **informations sensibles** qui pourraient apparaître dans le code (en particulier les



informations de connexion à la base de données) ne sont pas écrites en dur dans le code, mais stockées sous forme de **variables d'environnement** dans un fichier `.env`. Les fichiers contenant des informations sensibles comme le `.env` ou encore le `sqitch.conf` ne sont pas envoyés sur le remote repo (le dépôt distant sur GitHub) grâce à l'utilisation du fichier `.gitignore` permettant à Git d'ignorer certains dossiers ou fichiers.

5/ Autres services utilisés



Concernant notre **environnement de travail**, nous étions tous sous **Ubuntu**, le système d'exploitation **GNU/Linux** basé sur la distribution **Debian**, ce qui nous a donné accès à la **ligne de commande** de Linux, très efficace pour les développeurs. Nous avons également utilisé l'**IDE** (Integrated Development Environment ou environnement de développement) **Visual Studio Code** avec différents plugins, en particulier le plugin Live Share, qui nous a été très utile pour réaliser des sessions de **pair programming**.



De plus, pour maintenir une **cohérence**, une **homogénéité** et une **propreté** dans le style d'écriture du code, nous avons mis en place sur chaque sous-projet (front et back) **ESLint**, **Prettier** et **editorConfig**, que nous avons configuré à partir de la config partagée par Airbnb qui regroupe un grand nombre de **bonnes pratiques**.



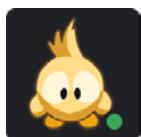
Enfin, pour le **déploiement** et la **mise en ligne** du jeu, nous avons utilisé un **serveur** privé hébergé par le service d'hébergement **OVH**.



F - ORGANISATION DE L'ÉQUIPE

1/ Présentation de l'équipe

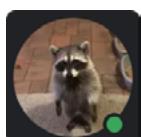
La composition de l'équipe a été décidée par notre encadrant pédagogique chez O'clock, en prenant en compte les souhaits exprimés par chacun. Étant moi-même la porteuse du projet, j'étais d'office positionnée dessus. Les trois collègues qui m'ont rejoint sur le projet avaient tous choisi le projet Bookworm en premier vœu, et nous étions tous heureux et motivés pour travailler sur ce projet.



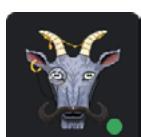
Agathe : Déjà un pied dans l'informatique, après quelques années en ESN comme designeuse d'interface et intégratrice HTML / CSS, c'est avec motivation et bonheur que je me lance pour apprendre le développement.



Yoan : Anciennement directeur de magasin, Yoan n'a jamais eu l'occasion de faire des études après une scolarité difficile. Il a fait le choix courageux d'une totale reconversion dans le développement, ce fut un plongeon dans le grand bain pour lui !



Philémon : Le plus jeune de l'équipe, avec ses 21 bougies soufflées pendant le projet ! Il s'est lancé dans l'aventure après un DUT MMI (Métiers du Multimédia et de l'Internet). Son caractère proactif et motivé ont été d'une grande aide tout au long du projet



Thibaud : Après avoir fait ses premiers pas dans la programmation sur Game Maker Studio en autodidacte, lui aussi s'est décidé à franchir le pas. Il est le calme et la force tranquille de l'équipe, on l'entend peu, mais quand on l'entend, c'est toujours pertinent !

2/ Répartition des rôles et détails

La répartition des rôles s'est faite sur la base du volontariat et après concertation de toute l'équipe.

On trouve dans les rôles des **Lead dev** les personnes se sentant les plus en confiance sur les technologies pour prendre des décisions techniques, le **Product Owner** est le porteur du projet, car c'est lui qui a la vision du projet final d'un point de vue fonctionnel, le poste de **Scrum Master** convient bien à une personne motivée et motivante, qui se sent à l'aise pour gérer



une dynamique de groupe, et enfin le poste de **référent technique** convient bien à quelqu'un qui aime faire de la veille, et lire de la documentation.

Agathe

Product Owner : C'est en quelque sorte le chef de projet en mode agile. Il s'assure que le projet en cours de réalisation correspond bien aux attentes et aux besoins exprimés. Il est responsable de la partie fonctionnelle, c'est pourquoi en cas de besoin, c'est lui qui aura le dernier mot lors d'une prise de décision sur un aspect fonctionnel du projet.

Lead Dev Back : Il est responsable de la partie back du développement. Il donne des recommandations sur l'architecture et les moyens à mettre en place, il est également amené à faire des choix techniques.

Dev Back

Yoan

Scrum Master : Il supervise l'avancement global du projet, et des tâches effectuées. Il a en charge la gestion du kanban avec l'avancement des différentes tâches et User Stories, et il anime les différents rituels agiles mis en place dans le projet.

Dev Back

Philémon

Lead Dev Front : Il est responsable de la partie front du développement. Il donne des recommandations sur l'architecture et les moyens à mettre en place, il est également amené à faire des choix techniques.

Dev Front

Thibaud

Référent Technique : Il apporte conseil et appui technique à l'équipe, en réalisant une veille sur les technologies qui sont, ou qui pourraient être, utilisées par le projet. Il peut être amené à lire de la documentation, réaliser des POC (Proof Of Concept), et transmettre sa connaissance à son équipe.

Dev Front

3/ Les outils utilisés

Communication

Slack était notre outil principal de communication écrite tout au long de la formation. Du-



rant la période de projet, nous avons continué à l'utiliser pour échanger et s'entraider entre groupes de projets au sein de notre promotion, et nous avons utilisé un salon dédié à notre équipe pour communiquer entre nous et avec nos deux encadrants pédagogiques.

Discord était notre outil de communication principal. Nous avons utilisé les salons vocaux tout au long du projet comme "open-spaces virtuels", pour échanger à l'oral, mais aussi partager nos écrans, échanger à l'écrit, partager des liens ou des fichiers lors de discussions.

Partage de fichiers et cloud

Nous avons utilisé **Google Drive**, et ses différents outils mis à disposition (Doc, Sheets...) pour rédiger et stocker certains fichiers, en particulier toutes la documentation du projet (cahier des charges, documentation de conception de la base de données etc.). La possibilité de lire et d'écrire sur un même document à plusieurs personnes en simultané nous a été très utile.

Gestion de projet

Trello était notre outil de gestion de projet en ligne, il est inspiré de la méthode Kanban qui repose sur une organisation des projets en listes de cartes, chaque carte représentant une tâche à accomplir.

Versionning

Pour le versioning, nous avons utilisé **Git** et **GitHub**, cet aspect ayant déjà été abordé plus en détails précédemment, je ne reviendrai pas dessus.

Développement

Pour le développement, nous avons utilisé l'IDE **Visual Studio Code**, open-source, très complet, et personnalisable grâce à l'installation de nombreuses extensions développées par la communauté.

4/ Organisation du travail

Journée de travail standard

La journée commençait entre **8h30** et **9h30** en fonction des préférences ou des impératifs de chacun, elle se terminait généralement vers **17h** ou **17h30**, parfois plus tard pour certains mais sans obligation.



Un **daily meeting** avait lieu dans la matinée une fois toute l'équipe connectée, pour que chacun expose en une phrase ou deux ce qu'il a fait la veille, ce qu'il va faire ce jour, et les éventuels problèmes auxquels il est confronté. Ce petit meeting est donc l'occasion de passer en revue les tâches du Trello pour les passer de Terminé à Validé.

Une **mise à jour quotidienne** était faite sur les **journaux de bord**, le journal d'équipe étant géré par le Scrum Master, et les journaux individuels par chacun.

En général, les équipes (front et back) s'installaient dans **deux salons Discord voisins**, avec des visites ponctuelles des uns chez les autres pour poser une question, clarifier un point, annoncer une modification pouvant avoir des répercussions sur l'autre équipe, ou encore annoncer un merge.

Les deux équipes se réunissaient sur **un salon Discord** en certaines occasions pour les- quelles il n'était pas opportun de se séparer (par exemple pour le déploiement).

En dehors des journées standard de travail, **deux sessions de code** ont eu lieu sur le week-end pour les volontaires uniquement, et une **session de détente et team building** a été organisée sur un jeu vidéo coopératif un week-end.

Le projet étant géré en **mode agile**, nous avions des phases de travail itératives sous forme de **Sprint**.

À chaque **début de sprint**, nous faisions un passage sur le backlog pour voir les tâches à effectuer, et éventuellement modifier, ajouter ou supprimer certaines tâches.

À chaque **fin de sprint**, nous faisions un meeting pour décider qui présenterait le projet à la **démo** (basé sur le volontariat), faire le point sur les milestones atteintes et les features implémentées et fonctionnelles, et basculer les tâches Trello du sprint validées dans des archives de sprint.

Organisation sur Trello

Quatre listes correspondant au sprint en cours

à faire : les tâches à réaliser durant le sprint, qui n'ont pas encore été commencées

En cours : les tâches en cours, assignées à une ou plusieurs personnes

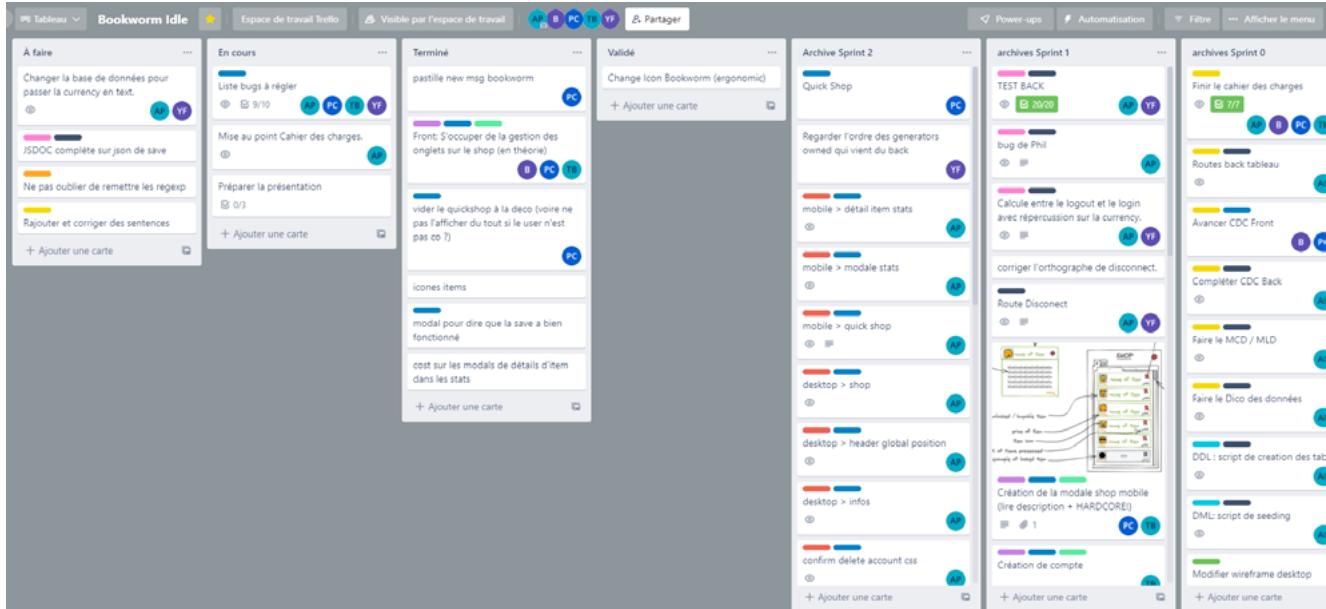
Terminé : les tâches dont les personnes qui en avaient la responsabilité estiment avoir terminées

Validé : les tâches pour lesquelles l'ensemble de l'équipe estime qu'elles sont terminées

Une **liste archive par sprint** passé, pour garder un aperçu des tâches des sprints précédents

Les tâches étaient **taguées** selon les thématiques, grâce au système d'étiquettes.





5/ Programme des sprints

Sprint 0

Conception	Mise en place
<p>présentation, prise en main du projet</p> <p>répartition des rôles</p> <p>définition des fonctionnalités et du MVP</p> <p>écriture du cahier des charges</p> <p>wireframes, moodboards, maquette</p> <p>MCD, MLD, dictionnaire des données</p>	<p>création des repos Git</p> <p>mise en place architecture de l'app front</p> <p>mise en place architecture de l'API back</p> <p>mise en place de Sqitch et de la BDD (init et seeding)</p> <p>travail sur les requêtes SQL</p>



Sprint 1

Front	Back	Connexion front/back
création des premiers composants mise en place (statique) des différentes modales mise en place (statique) du quickshop mise en place de Redux (passage de typescript no Redux à Redux no Typescript) mise en place de l'incrémation mise en place de l'affichage spécial des très grands nombres mise en place des formulaires mise en place des JWT côté front	création du service de build de save mise en place du CRUD player migrations sur la BDD (domains) mise en place des schémas de validation des saisies utilisateur mise en place des routes back mise en place du calcul des gains hors connexion mise en place des JWT côté back configuration Bearer authentication du jsdoc-swagger	premier fetch API modale bookworm fonctionnelle modale player fonctionnelle save manuelle fonctionnelle modale stats fonctionnelle modale shop WIP



Sprint 2

Organisation	Front	Back
passage d'Agathe en backup sur le front pour réaliser le css	mise en place de sass intégration du style en mobile-first et déclinaison desktop pour être RWD finalisation du shop finalisation du quickshop mise en place de la modal de reconnexion mise en place des modales de détails d'item mise en place des messages d'erreur de saisie côté front mise en place des icônes	fix d'une régression en back ajout de contenu dans le fichier de seeding migration sur la bdd pour ajout des icônes

Sprint 3

Derniers fix	Mise en production	Présentation
corrections de bugs derniers ajustements de css	déploiement du back et du front sur serveur OVH	préparation de la présentation oral blanc présentation live Youtube



G - PRODUCTIONS

1/ Réalisations personnelles

Service de build de save json

L'une de mes réalisations personnelles les plus intéressantes et significatives est le **service de build de save Json**.

C'est un **service** qui est appelé dans différents **contrôleurs**, eux-mêmes appelés par différents **endpoints** de l'application. Ce service nommé buildSave récupère dans la base de données toutes les informations nécessaires pour construire la sauvegarde du joueur. Ces données sont ensuite manipulées, et différents calculs sont effectués, pour finalement renvoyer au front un Json complexe contenant toutes les informations de partie du joueur, et dont l'application front a besoin pour fonctionner.

Ce **service** est appelé à chaque fois qu'une action qui nécessite un recalculation des informations de partie est effectuée par le joueur, c'est-à-dire lorsque le joueur achète un nouveau générateur qu'il ne possédait pas déjà, ou lorsque le joueur achète un générateur dont il avait déjà au moins un exemplaire.

Ce **service** est également appelé lorsqu'une sauvegarde est effectuée (excepté la sauvegarde déclenchée par une action de déconnexion).

Pour bien comprendre les différentes actions qui vont être décrites, il est important d'expliquer le **fonctionnement de base des générateurs**, qui sont les items que le joueur peut acheter dans le shop.

Cette mécanique de **game design** (conception et mise au point des règles et éléments constitutifs d'un jeu) fait partie des mécaniques de base que l'on retrouve dans quasiment tous les idle games parfois déclinées de différentes manières.

Dans Bookworm Idle Game, les générateurs existent en **quatre types différents**, chacun donnant un type de bonus différent.

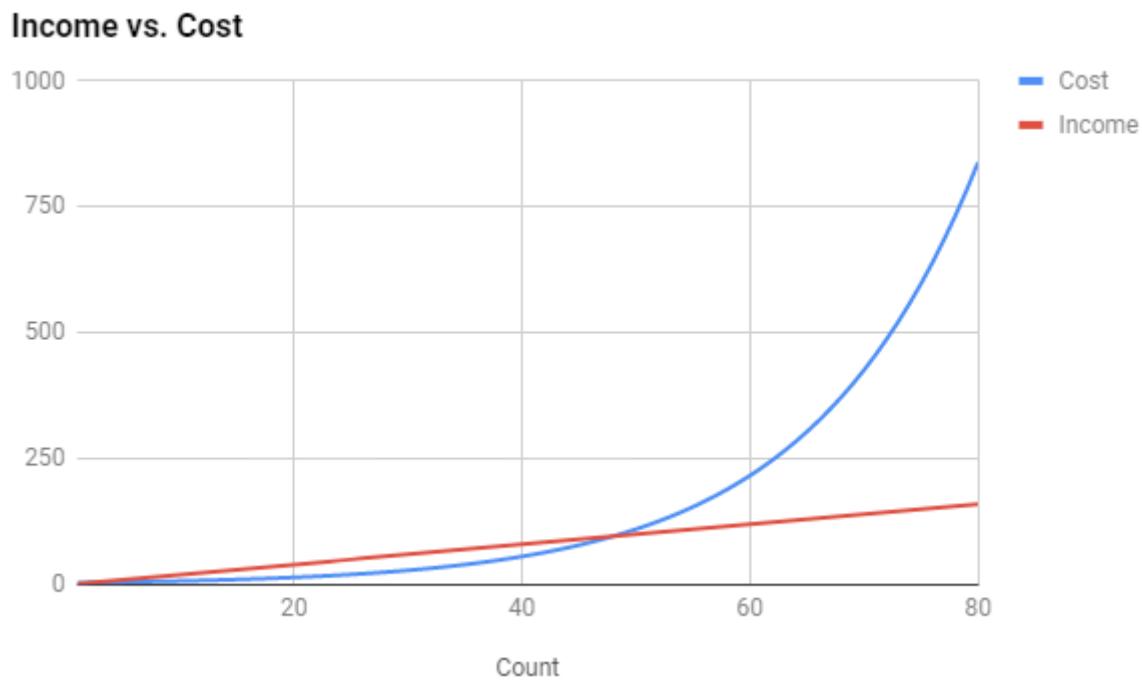
- Les générateurs dits **clickFlat** donnent un bonus flat (+x) à la valeur du clic
- Les générateurs dits **clickPercent** donnent un bonus en pourcentage (+x%) à la valeur du clic une fois les bonus flats appliqués
- Les générateurs dits **IdleFlat** donnent un bonus flat (+x) à la valeur du idle
- Les générateurs dits **IdlePercent** donnent un bonus en pourcentage (+x%) à la valeur du clic une fois les bonus flats appliqués



Tous les générateurs peuvent être achetés en plusieurs exemplaires, les bonus étant additionnés.

Les bonus donnés par un générateur sont fixes, un générateur qui donne un bonus de +5 à la valeur du idle lorsqu'on l'achète pour la première fois, donnera encore +5 si on l'achète une deuxième, puis une troisième fois.

En revanche, **le coût d'achat d'un générateur n'est pas fixe**. Pour inciter le joueur à acheter d'autres générateurs, et pour que sa courbe de progression atteigne un plateau s'il se contente d'acheter toujours le même générateur, les générateurs ont un coût de base (le coût lors du premier achat), et chaque fois que le joueur va acheter ce générateur, le coût du prochain achat du même générateur va augmenter, suivant une **courbe exponentielle**.



Comme on peut le voir sur le graphique ci-dessus, l'**income**, c'est-à-dire le bonus donné par le générateur est **linéaire**, tandis que le **cost**, c'est-à-dire le coût de chaque générateur est **exponentiel**.

Ainsi, dans un premier temps, il est intéressant pour le joueur d'acheter plusieurs fois ce même générateur, pour additionner les bonus et gagner plus de monnaie, et ainsi acheter encore des générateurs. Mais au bout d'un moment, le coût du générateur va devenir trop important et l'**écart coût/bonus rendra l'achat de ce générateur peu attractif**, incitant le joueur à varier ses achats.



Voyons maintenant comment tout cela se traduit dans buildSave.

```
● ● ●
1 async buildSave(playerId) {
2   debug(`buildSave called for player id ${playerId}`);
3   // player object
4   const playerJson = await dataMapper.getOneUserJson(playerId);
5   const { player } = playerJson;
6   delete player.password;
7   // generators owned array of objects
8   const generatorsOwnedJson = await dataMapper.getGeneratorsOwned(playerId);
9   const generatorsOwned = generatorsOwnedJson.generators;
10  // player not owned generator array of object
11  const playerNotOwnsGeneratorJson = await dataMapper.getPlayerNotOwnsGenerator(playerId);
12  const playerNotOwnsGenerator = playerNotOwnsGeneratorJson.generators;
13  // player_owns_generator array of objects
14  const playerOwnsGeneratorJson = await dataMapper.getPlayerOwnsGenerator(playerId);
15  const playerOwnsGenerator = playerOwnsGeneratorJson.playerownsgenerator;
```

Dans un premier temps, les **méthodes du data mapper** me permettent de récupérer les Json suivants :

- Un objet contenant les informations de la table player pour un id de joueur donné
- Un tableau contenant sous forme d'objets les générateurs que le joueur possède au moins une fois, on y trouve pour chaque générateur, les informations contenues dans la table generator.
- Un tableau contenant sous forme d'objets les générateurs que le joueur ne possède pas, on y trouve pour chaque générateur, les informations contenues dans la table generator.
- Un tableau contenant sous forme d'objets toutes les informations d'une ligne de la table d'association entre player et generator pour un joueur donné.

Voir les annexes pour voir le détail des différentes requêtes.

```
● ● ●
1 // init player bonus
2 const playerBonus = {
3   clic_flat_bonus: 0,
4   clic_percent_bonus: 1,
5   idle_flat_bonus: 0,
6   idle_percent_bonus: 1,
7};
```

Dans un second temps, j'**initie les valeur de base du joueur à 0**, c'est à dire les bonus flat à 0 (+0), et les bonus percent à 1 (*1).





```
1 // build generatorsOwned + calc bonus
2 for (let i = 0; i < generatorsOwned.length; i += 1) {
3     const generatorInfo = playerOwnsGenerator.find((element) => element.generator_id === generatorsOwned[i].id);
4     // add number owned
5     generatorsOwned[i].number_owned = generatorInfo.number_owned;
6     // calc next cost
7     generatorsOwned[i].next_cost = Math.floor(
8         generatorsOwned[i].starting_cost * generatorsOwned[i].cost_factor ** generatorsOwned[i].number_owned,
9     );
10    // calc total value for each generator
11    generatorsOwned[i].total_clic_flat = generatorsOwned[i].clic_flat_value * generatorsOwned[i].number_owned;
12    generatorsOwned[i].total_clic_percent = generatorsOwned[i].clic_percent_value * generatorsOwned[i].number_owned;
13    generatorsOwned[i].total_idle_flat = generatorsOwned[i].idle_flat_value * generatorsOwned[i].number_owned;
14    generatorsOwned[i].total_idle_percent = generatorsOwned[i].idle_percent_value * generatorsOwned[i].number_owned;
15    // calc total bonus
16    playerBonus.clic_flat_bonus += generatorsOwned[i].total_clic_flat;
17    playerBonus.clic_percent_bonus += generatorsOwned[i].total_clic_percent;
18    playerBonus.idle_flat_bonus += generatorsOwned[i].total_idle_flat;
19    playerBonus.idle_percent_bonus += generatorsOwned[i].total_idle_percent;
20 }
```

Je vais ensuite faire une **boucle sur la liste des générateurs possédés** par le joueur, pour **enrichir** chaque générateur d'**informations supplémentaires** et **calculer** les valeurs de playerBonus.

Pour chaque générateur possédé, je commence par aller **chercher les informations de la table d'association** que je stocke dans la variable generatorInfo.

Ensuite, j'**enrichis les informations de ce générateur** en y ajoutant un ensemble clé/valeur que je nomme `number_owned` et qui prend comme valeur le nombre de ce générateur que le joueur possède.

Grâce à cette information, je peux commencer mes **calculs**.

Je commence par ajouter un nouvel ensemble clé/valeur à mon générateur, que je nomme `next_cost` et qui calcule le **coût du prochain achat** de ce générateur en fonction du **coût de base**, du **nombre déjà possédé** et d'un **facteur multiplicateur** permettant de faire varier l'exponentielle du coût du générateur.

Le calcul est le suivant :

prochain coût = coût de base * facteur multiplicateur ^ nombre possédé

C'est ce calcul qui me permet d'obtenir une courbe des coûts d'un générateur qui soit **exponentielle**.

Ensuite, je vais **calculer les valeurs de bonus donnés par le générateur**, en fonction du nombre possédé, il s'agit d'une simple multiplication. Ces valeurs viennent elles aussi enrichir le générateur pour y ajouter pour chaque générateur possédé, les bonus obtenus en fonction du nombre possédé.



Et enfin, pour chaque type de bonus, je vais **ajouter les bonus précédemment calculés à playerBonus**, ce qui fait qu'en fin de boucle, les valeurs dans playerBonus **prendront en compte tous les bonus cumulés** par le joueur grâce à ses générateurs.



```
1 // split owned generators into 4 arrays of generators by type
2 const clickFlat = generatorsOwned.filter((element) => element.type === 1);
3 const clickPercent = generatorsOwned.filter((element) => element.type === 2);
4 const idleFlat = generatorsOwned.filter((element) => element.type === 3);
5 const idlePercent = generatorsOwned.filter((element) => element.type === 4);
6 // add generators owned arrays into player object
7 player.generatorsOwned = {
8   clickFlat,
9   clickPercent,
10  idleFlat,
11  idlePercent,
12};
```

Maintenant que generatorsOwned contient tous les générateurs, avec toutes les informations calculées, je sépare mes générateurs que je répartis **selon leur type**, et enfin, **j'enrichis l'objet player** récupéré au début, pour lui ajouter un ensemble clé/valeurs generatorsOwned qui est un objet qui contient un tableau pour chaque type de générateur, qui lui-même contient les générateurs du type en question, avec toutes les informations issues de la base de données et les informations calculées à l'étape précédente.





```
1 // split owned generators into 4 arrays of generators by type
2     //! debug('playerNotOwnsGenerator', playerNotOwnsGenerator);
3 // TODO refacto for better code ?
4 if (playerNotOwnsGenerator !== null) {
5     const clickFlatNot = playerNotOwnsGenerator.filter((element) => element.type === 1);
6     const clickPercentNot = playerNotOwnsGenerator.filter((element) => element.type === 2);
7     const idleFlatNot = playerNotOwnsGenerator.filter((element) => element.type === 3);
8     const idlePercentNot = playerNotOwnsGenerator.filter((element) => element.type === 4);
9     // add generators owned arrays into player object
10    player.generatorsNotOwned = {
11        clickFlatNot,
12        clickPercentNot,
13        idleFlatNot,
14        idlePercentNot,
15    };
16 } else {
17     const clickFlatNot = [];
18     const clickPercentNot = [];
19     const idleFlatNot = [];
20     const idlePercentNot = [];
21     // add generators owned arrays into player object
22     player.generatorsNotOwned = {
23         clickFlatNot,
24         clickPercentNot,
25         idleFlatNot,
26         idlePercentNot,
27     };
28 }
```

Je réalise la même opération pour les **générateurs non possédés** (pas de calcul ou d'information supplémentaire pour ceux-ci, seulement les informations issues de la table generator), cependant, je dois créer un tableau vide pour le cas où tous les générateurs de ce type soient achetés, sinon la valeur est undefined.



```
1 debug('-----START FINAL CALC-----');
2     player.idle_value += Math.floor(playerBonus.idle_flat_bonus * playerBonus.idle_percent_bonus);
3     player.click_value += Math.floor(playerBonus.click_flat_bonus * playerBonus.click_percent_bonus);
4     debug('bonus:', playerBonus);
5     debug('player idle value', player.idle_value);
6     debug('player click value', player.click_value);
7     return player;
```

Enfin, je réalise un dernier calcul, pour **calculer les valeurs finales du clic et du idle du**



joueur. J'ajoute à la valeur de base issue des informations de la table player le bonus flat multiplié par le bonus en pourcentage.

Je peux retourner player, qui est maintenant un **objet complexe, contenant toutes les informations de partie du joueur**. Toutes les manipulations de datas ont été faites pour envoyer au front un objet dans lequel il suffit de naviguer pour trouver toutes les informations nécessaires.

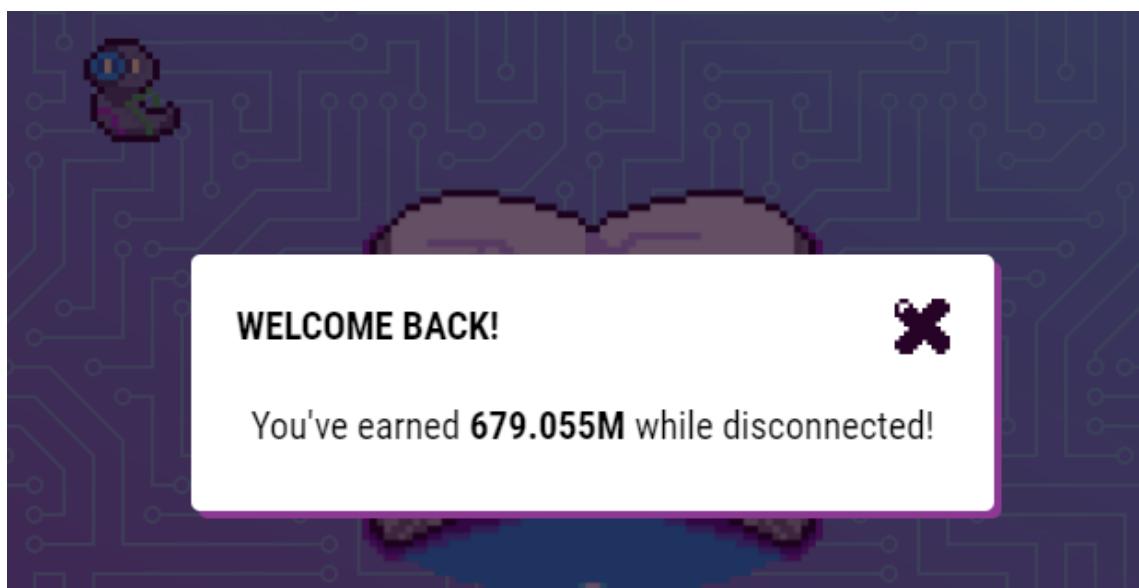
Une version allégée du Json de save envoyé au front est disponible en annexes. La version proposée en annexes est limitée à 17 générateurs à titre d'exemple, le jeu complet en contenant 40 au total.

Calcul de gains hors ligne

Une autre **mécanique de game design propre aux idle games** est la notion de **continuité des gains idle lorsque le joueur est déconnecté**. L'idée étant de donner l'illusion au joueur que "le jeu continue de tourner" alors que celui-ci est déconnecté. Ainsi, à sa reconnexion, le joueur recevra la monnaie qu'il a gagné alors qu'il était déconnecté.

Cette mécanique est très importante pour le projet Bookworm Idle Game, étant donné que l'idée du jeu est de proposer au joueur de faire une pause, de se déconnecter, et d'aller lire un bon livre.

Durant le jeu, le bookworm envoie, sous forme de petits messages, des incentives à quitter le jeu et aller découvrir tel ou tel livre. À sa reconnexion, le joueur est récompensé d'avoir quitté le jeu par **une modale qui lui annonce combien de savoir le bookworm a gagné pour lui pendant son absence**.



Cette feature se déroule en **deux étapes**.



La première étape a lieu à la déconnexion du joueur, et consiste à récupérer sous forme de **timestamp** (horodatage) **la date et l'heure de déconnexion** du joueur, et de l'enregistrer dans la base de données.

```
● ● ●  
1 router  
2   .route('/')
```

3 **/****
4 * PATCH /api/disconnect/
5 * @summary disconnected player
6 * @tags Player account
7 * @security BearerAuth
8 * @param {updateSave} request.body.required - json object with input fields values from front
9 */
10 .patch(controllerHandler(checkLogin.checkLogin), controllerHandler(controller.disconnectAndSave));

Lorsque le joueur clique sur le **bouton de déconnexion**, ou qu'il **ferme l'onglet** de navigateur de l'application, cela appelle le **endpoint** `/api/disconnect/` qui appelle la **méthode de contrôleur** `disconnectAndSave`.

```
● ● ●  
1 async disconnectAndSave(req, res) {  
2   debug(`updateSave for player: ${reqdecoded.id} ${reqdecoded.username}`);  
3   const playerId = reqdecoded.id;  
4   const { currency, clickCounter } = reqbody;  
5   debug(`currency: ${currency}, click_counter: ${clickCounter}`);  
6   // update currency and click_counter value and logoutTime  
7   await playerSavedataMapper.updateCurrencyClick(playerId, currency, clickCounter);  
8   await playerAccountDataMapper.updateLogoutTime(playerId);  
9   return res.status(200).json({ logged: false });  
10 }
```

Le **contrôleur** récupère du front les valeurs actuelles de **savoir** (`currency`) et de **compteur de clic** (`clickCounter`) et les enregistre dans la base de données via la **méthode** `updateCurrencyClick` **du data mapper**, puis enregistre dans la base de données **la date et l'heure de logout** (déconnexion) du joueur.

```
● ● ●  
1 async updateLogoutTime(id) {  
2   debug('updateLogoutTime ', id);  
3   const result = await client.query('UPDATE player SET logout_time = CURRENT_TIMESTAMP WHERE player.id = $1', [id]);  
4  
5   return !!result.rowCount;  
6 },
```



La **deuxième étape** a lieu à la **reconnexion** du joueur, où l'on va de nouveau **récupérer sous forme de timestamp la date et l'heure de connexion** du joueur.

```
● ● ●  
1 router  
2   .route('/')  
3   /**  
4     * POST /api/login/  
5     * @summary login to player account  
6     * @tags Player account  
7     * @param {playerLogin} request.body.required - json object with input fields values from front  
8     */  
9   .post(validate('body', loginSchema), controllerHandler(controller.login));
```

Lorsque le joueur clique sur le **bouton de login** (connexion), le **endpoint** `/api/login/` est appelé et déclenche la **méthode de contrôleur** `login`.

La **méthode** `login` est une méthode assez longue et complexe car elle commence par vérifier les informations de connexion saisies par l'utilisateur, et créer le token JWT d'authentification du joueur.

Ce sont les lignes suivantes qui vont nous intéresser :

```
● ● ●  
1 // calc earned currency  
2 const afkSc = (await playerAccountDataMapper.countAfkSc(player.id)).timepersecond;  
3 const playerSave = await save.buildSave(player.id);  
4 if (!playerSave) {  
5   throw ApiError('PlayerSave build error', { statusCode: 500 });  
6 }  
7 debug('idle value', playerSave.idle_value);  
8 const currencyAfk = afkSc * playerSave.idle_value;  
9 const newCurrency = playerSave.currency + currencyAfk;  
10 debug(`currency: ${playerSave.currency} (${typeof playerSave.currency}), afk sec: ${afkSc}, currency won: ${currencyAfk} (${typeof currencyAfk}), total currency: ${newCurrency} (${typeof newCurrency})`);  
11 playerSave.currency = newCurrency;  
12 return res.status(200).json({  
13   logged: true,  
14   token,  
15   logoutCurrency: currencyAfk,  
16   playerSave,  
17 });
```

Je commence par **stocker dans une variable le temps de déconnexion** du joueur en secondes.



C'est la **méthode** de data mapper countAfkSc qui me retourne cette donnée :

```
1  async countAfkSc(id) {
2      debug(`countAfkCurrency called for id ${id}`);
3      const result = await client.query(`
4          SELECT round(extract ('epoch' from login_time - logout_time)) as timepersecond
5          FROM player
6          WHERE player.id = $1;
7      `, [id]);
8      return result.rows[0];
9  },
```

La **requête** qui est faite ici sélectionne dans la table player, et pour le player ayant l'id 1, la période en seconde (epoch) qui est le résultat de la soustraction du timestamp de logout au timestamp de login, et l'arrondi pour obtenir un entier.

Ensuite, j'appelle le **service** buildSave, qui a été vu en détail dans la réalisation précédente. Cela me permet de **récupérer la valeur de idle du joueur** une fois tous les bonus de générateurs appliqués. Je peux alors **calculer la monnaie** (currency) **gagnée** en multipliant le nombre de secondes de déconnexion par la valeur de idle du joueur. Je calcule la nouvelle valeur de monnaie du joueur en ajoutant ce qu'il a gagné pendant sa période de déconnexion à ce qu'il avait déjà, et je modifie la valeur de currency de playerSave en y mettant cette nouvelle valeur (newCurrency).

Enfin, j'envoie la **réponse au format json**, qui contient entre autres l'ensemble clé/valeur logOutCurrency, pour pouvoir afficher au joueur la modale qui l'informe de la quantité de savoir gagné pendant sa déconnexion.

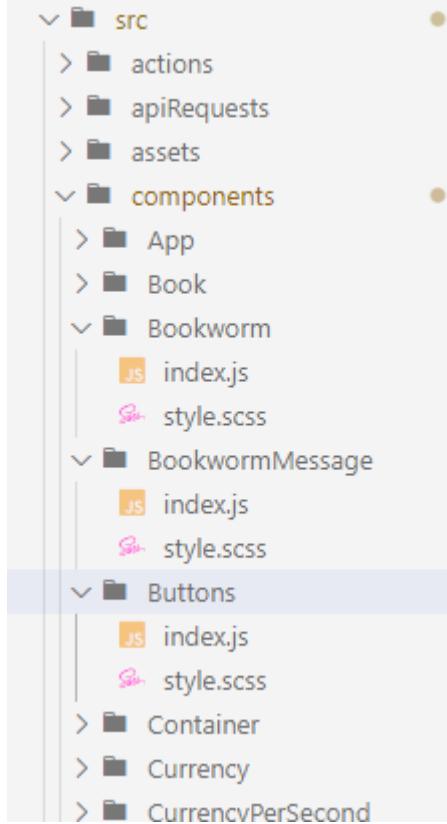
SCSS

Durant le **Sprint 2**, le développement de l'API back ayant pris de l'avance, il ne restait plus qu'à tester, récupérer les retours du front pour corriger d'éventuels bugs et compléter le seeding de la base de données pour augmenter la durée de vie du jeu.

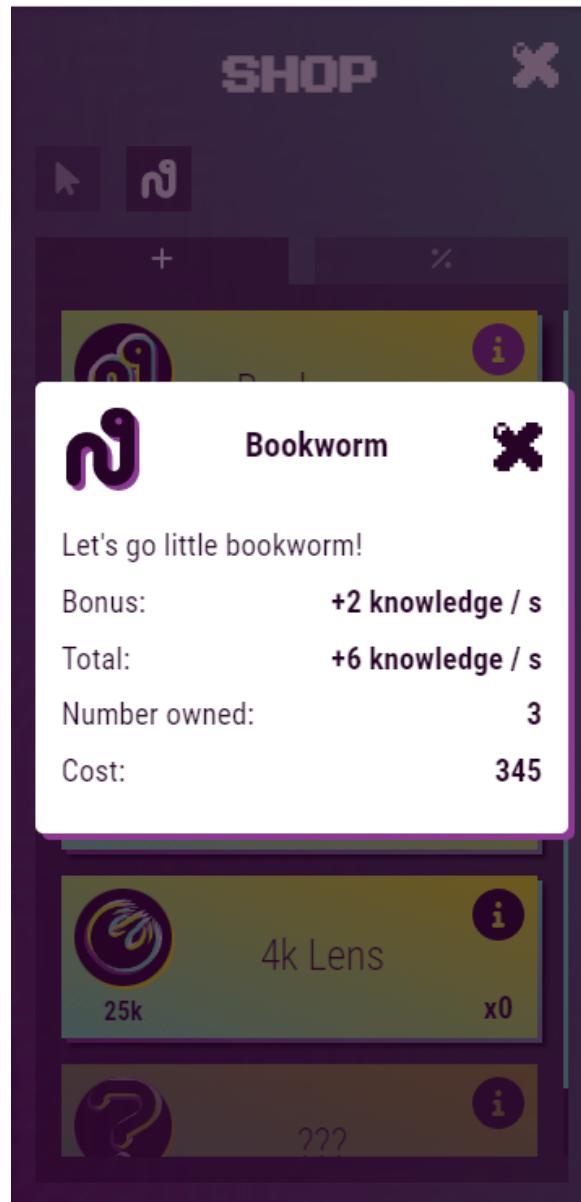
J'ai donc profité de cette avance en back pour venir en **renfort sur l'application front**, et je suis intervenue sur la **mise en place du style css**, qui avait été laissé en standby par l'équipe front pour prioriser le développement des fonctionnalités du MVP.

J'ai fait le choix de travailler en **SCSS** pour profiter de la **syntaxe nested** (imbriquée).





- Comme nous étions dans un **contexte d'application React**, les différents éléments constitutifs de l'interface étaient répartis en différents **composants** (components), chacun disposant de son propre fichier `style.scss` pour gérer le CSS.



Pour illustrer mon intervention sur le **SCSS**, je vais donc prendre l'exemple du **composant ShopItemModal**, qui est une modale qui s'ouvre par-dessus le shop, lorsqu'on clique sur le bouton d'information d'un item du shop, pour en voir les détails.



Structure du composants et ajout de classes



```
1  return (
2      <div className="small-modal" onClick={handleClick}>
3          <div className="small-modal-container">
4              <div className="shopitemdetail">
5                  <div className="shopitemdetail__header">
6                      <div className="shopitemdetail__header__img">
7                          <img src={icônes[icon]} alt="icônes" />
8                      </div>
9                      <p>{name}</p>
10                     <button className="shopitemdetail__header__btn" type="button" onClick={handleClick}>
11                         <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 23.684 23.684"><path d="M23.684 1.974V5.92H21.71v1.975h-1.973v1.973h-1.974v1.974h1.974v1.974h1.973v1.973h1.974v5.921H21.71v1.974h-3.946V21.71h-1.975v-1.973h-1.973v-1.974h-1.974v1.974H9.868v1.973H7.896v1.974H1.974V21.71H0v-3.946h1.974v-1.975h1.973v-1.973H5.92v-1.974H3.947V9.868H1.974V7.896H0V1.974h1.974V5.92h1.973V3.949H5.92V1.974h1.975v1.975h1.973V5.92h1.974V3.949h1.974V1.974h1.973V0h5.921v1.974h1.974zm-21.71 0V0H5.92v1.974H1.974z" /></svg>
12                     </button>
13                 </div>
14                 <div className="shopitemdetail__body">
15                     <p className="shopitemdetail__body__description">{text}</p>
16                     <p className="shopitemdetail__body__bonus">
17                         <span className="bonus--pre">Bonus:</span>
18                         <span className="bonus--post">{handleBonus()}</span>
19                     </p>
20                     <p className="shopitemdetail__body__bonus">
21                         <span className="bonus--pre">Total:</span>
22                         <span className="bonus--post">{handleBonusTotal()}</span>
23                     </p>
24                     <p className="shopitemdetail__body__bonus">
25                         <span className="bonus--pre">Number owned:</span>
26                         <span className="bonus--post">{number}</span>
27                     </p>
28                     <p className="shopitemdetail__body__bonus">
29                         <span className="bonus--pre">Cost:</span>
30                         <span className="bonus--post">{handleCost && convertToReadable(handleCost())}</span>
31                     </p>
32                 </div>
33             </div>
34         </div>
35     </div>
36 );
```

Pour commencer, j'ai **organisé la structure HTML** du composant dans le fichier javascript correspondant, pour pouvoir ensuite sélectionner et agir sur les différents éléments dans le SCSS. J'ai également **ajouté des classes** aux éléments, que j'ai nommées en respectant la convention de nommage **BEM** (Blocks-Elements-Modifiers).



Écriture du SCSS

J'ai ensuite utilisé de la **syntaxe nested** pour reproduire côté SCSS la **logique de structure** de l'élément.

```
1 .small-modal {  
2   position: absolute;  
3   background-color: rgba(32, 2, 32, .8);  
4   top: 0;  
5   right: 0;  
6   left: 0;  
7   bottom: 0;  
8   z-index: 2;  
9  
10  .small-modal-container {  
11    height: 100%;  
12    display: flex;  
13    justify-content: center;  
14    align-items: center;  
15  
16    .shopitemdetail {  
17      position: relative;  
18      background-color: #fff;  
19      filter: drop-shadow(4px 4px #822F8A);  
20      border-radius: 5px;  
21      width: 100%;  
22      padding: 1rem;  
23      margin: 0 1rem;  
24      color: #200220;
```

J'ai donc commencé par décrire le style des **éléments "conteneurs"**, c'est à dire le **layer** (calque) qui se met par-dessus le shop et obscurcit celui-ci pour mettre en évidence la modale (**.small-modal**), et le bloc qui constitue la base de la modal (**.shopitemdetail**).

Le contenu de la modale se divise en **deux parties** :

Un **header** contenant l'illustration et le titre de l'item, et un bouton close pour fermer la modale et revenir au shop.

Un **body** qui contient le texte descriptif de l'item, le bonus de base et le total de bonus cumulés donné par cet item, le nombre de cet item possédé par le joueur, et le coût d'achat de cet item (calculé en fonction du nombre possédé).

J'ai donc décrit **dans un bloc le header**, avec les différents éléments contenus dedans, et **dans un second bloc le body**.



```
1 .shopitemdetail {
2     position: relative;
3     background-color: #fff;
4     filter: drop-shadow(4px 4px #822F8A);
5     border-radius: 5px;
6     width: 100%;
7     padding: 1rem;
8     margin: 0 1rem;
9     color: #200220;
10
11    .shopitemdetail__header {
12        display: flex;
13        align-items: center;
14        justify-content: space-between;
15
16        .shopitemdetail__header__img {
17            width: 3rem;
18            height: 3rem;
19            display: flex;
20            justify-content: center;
21            align-items: center;
22            &>svg, &>img {
23                fill: #200220;
24                filter: drop-shadow(3px 3px #822F8A);
25                width: 100%;
26                height: 100%;
27            }
28        }
29        &>p {
30            font-weight: 700;
31            font-size: 1.3rem;
32        }
33        .shopitemdetail__header__btn {
34            width: 30px;
35            height: 30px;
36            padding: 0;
37            border: none;
38            background: none;
39            transform: rotate(0deg);
40            transition: all .7s cubic-bezier(0.17, 0.84, 0.44, 1);
41            &>svg {
42                fill: #200220;
43            }
44            &:hover {
45                transform: rotate(180deg);
46            }
47        }
48    }
```



```
● ● ●  
1  .shopitemdetail__body {  
2      padding: 1.5rem 0 0 0;  
3      [class^='shopitemdetail__body__'] {  
4          margin-bottom: .5rem;  
5          text-align: left;  
6          font-size: 1.2rem;  
7          line-height: 1.4;  
8          display: flex;  
9          justify-content: space-between;  
10         .bonus--post {  
11             font-weight: 700;  
12         }  
13     }  
14 }
```



2/ Jeu d'essai

Scénario : Déconnexion - Reconnexion

Dans ce **scénario**, nous allons **tester le bon déroulement d'une déconnexion suivie d'une reconnexion** sur le même compte, ce qui nous permettra de vérifier que la **fonctionnalité de gain hors ligne** (détaillée précédemment dans les réalisations personnelles) **fonctionne correctement** et renvoie les résultats attendus.

Nous allons pour cela utiliser un compte qui vient d'être créé, et sur lequel nous avons acheté quelques items, pour que la valeur de **idle** soit égale à **20**.

id	click_value	click_counter	prestige_level	logout_time	login_time
integer	integer	integer	integer	timestamp with time zone	timestamp with time zone
0	1	235	1	[null]	[null]

Comme on peut le voir dans la **base de donnée**, le compte vient d'être créé, les valeurs de **logout_time** (timestamp de déconnexion) et **login_time** (timestamp de connexion) sont null.

```
middleware:checkLogin token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MywidXNlcj5hbWUiOiJ0ZXN0cGxheWV0amMVndQrQzqTtzzNHE +40s
middleware:checkLogin decoded { id: 3, username: 'testplayer3', iat: 1655727592, exp: 1655813992 } +0ms
controller:playerSave updateSave for player: 3 testplayer3 +40s
controller:playerSave currency: 10580, click_counter: 235 +0ms
model:player_save updateCurrencyClick called for id 3 +40s
SQL:log {
SQL:log   text: 'UPDATE player SET currency=$1, click_counter=$2 WHERE id=$3;',
SQL:log   values: [ 10580, 235, 3 ]
SQL:log } +40s
model:player_save return: undefined +41ms
model:player_account updateLogoutTime 3 +10m
SQL:log UPDATE player SET logout_time = CURRENT_TIMESTAMP WHERE player.id = $1 [ 3 ] +41ms
```

Grâce au **package debug** et aux nombreux **logs** qui s'affichent en **mode dev**, on peut suivre dans la console le déroulement de la déconnexion.

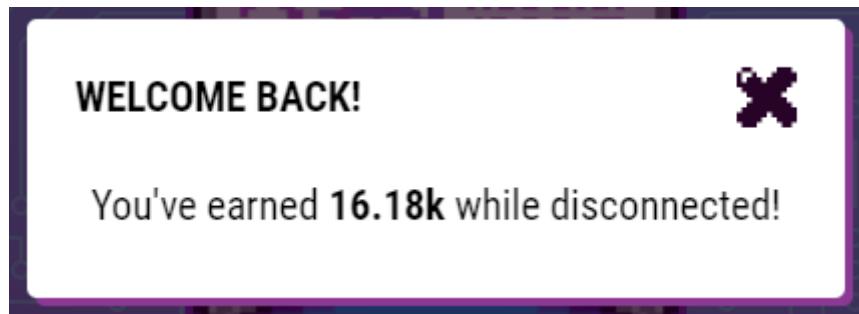
On peut donc voir que la **méthode updateSave** est appelée pour sauvegarder la partie, puis la **méthode updateLogoutTime**, qui update la valeur de **logout_time** dans la base de données.

On peut le vérifier directement dans la **base de données** :

e_value	click_value	click_counter	prestige_level	logout_time	login_time	cur big
eger	integer	integer	integer	timestamp with time zone	timestamp with time zone	cur big
0	1	235	1	2022-06-20 14:29:27.467...	[null]	



Nous allons maintenant **reconnecter le compte**, et vérifier que le gain obtenu est bien celui attendu.



Le message annonce **16,18k**, soit **16180**.

On va commencer par vérifier dans la base de données :

click_value	click_counter	prestige_level	logout_time	login_time	current
integer	integer	integer	timestamp with time zone	timestamp with time zone	begin
1	235	1	2022-06-20 14:29:27.467...	2022-06-20 14:42:56.079668+...	

Puis on va regarder dans la **console** le **déroulé des actions** à la connexion.

1 : La **méthode updateLoginTime** est appelée, et met à jour la valeur de `login_time` dans la base de données

2 : La **méthode countAfkSc** est appelée, et renvoie un nombre correspondant aux **secondes de déconnexion**

3 : Le **service buildSave** est appelé pour récupérer les **informations du joueurs**, effectuer les différents calculs et renvoyer un objet correspondant à la **sauvegarde de partie** du joueur

4 : Cela nous permet de **récupérer la valeur du idle**, **20**

5 : De retour dans le **contrôleur**, les informations récupérées aux étapes précédentes sont **manipulées** (pour plus de détails, voir la partie réalisations personnelles) et l'on obtient les **valeurs suivantes** :

currency (valeurs enregistrée à la déconnexion) : **10580**

afk sec (temps de déconnexion en secondes) : **809**

currency won (savoir gagné pendant la déconnexion) : **16180**

total currency (savoir du joueur après ajout du savoir gagné pendant la déconnexion) : **26760**



```

Validator:log { mail: 'testplayer3@mail.com', password: 'aaa' } +23ms
model:player_account findByMail called for mail testplayer3@mail.com +13ms
SQL:log SELECT * FROM player WHERE mail = $1 [ 'testplayer3@mail.com' ] +13ms
controller:playerAccount token: eyJhbGciOiJUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0ZXN0cGxhZWMyIiImlhdCI6MTY1NTcyO
UzV4bwOw0opATKXCjME7b Xkc +23ms
model:player_account updateLoginTime 3 +3ms
SQL:log UPDATE player SET login_time = CURRENT_TIMESTAMP WHERE player.id = $1 [ 3 ] +39ms
model:player_account countAfks called for id 3 +15ms
SQL:log SELECT round(extract ( epoch from login_time - logout_time)) as timepersecond
SQL:log FROM player
SQL:log WHERE player.id = $1: [ 3 ] +15ms
service:buildSave buildSave called for player id 3 +13ms
model:player_save getOneUserJson called for id 3 +13ms
SQL:log {
SQL:log   text: 'select row_to_json(player.*) as player from player where id=$1;',
SQL:log   values: [ 3 ]
SQL:log } +1ms
model:player_save getGeneratorsOwned called for id 3 +1ms
SQL:log {
SQL:log   text: 'SELECT json_agg(generator.* ORDER BY generator."order") as generators\n' +
SQL:log   '   FROM player_owns_generator\n' +
SQL:log   '   JOIN player ON player_owns_generator.player_id=player.id\n' +
SQL:log   '   JOIN generator ON player_owns_generator.generator_id=generator.id\n' +
SQL:log   '   WHERE player_owns_generator.player_id=$1\n' +
SQL:log   '   GROUP BY player;',
SQL:log   values: [ 3 ]
SQL:log } +1ms
model:player_save getPlayerNotOwnsGenerator called for id 3 +2ms
SQL:log {
SQL:log   text: 'SELECT json_agg(generator.* ORDER BY generator."order" ) as generators\n' +
SQL:log   '   FROM generator WHERE generator.id NOT IN (\n' +
SQL:log   '       SELECT player_owns_generator.generator_id FROM player_owns_generator\n' +
SQL:log   '       WHERE player_owns_generator.player_id=$1);',
SQL:log   values: [ 3 ]
SQL:log } +2ms
model:player_save getPlayerOwnsGenerator called for id 3 +1ms
SQL:log {
SQL:log   text: 'SELECT json_agg(player_owns_generator.*) as playerOwnsGenerator\n' +
SQL:log   '   FROM player_owns_generator WHERE player_id=$1;',
SQL:log   values: [ 3 ]
SQL:log } +2ms
service:buildSave -----START FINAL CALC----- +6ms
service:buildSave bonus: {
clic_flat_bonus: 37,
clic_percent_bonus: 1,
idle_flat_bonus: 20,
idle_percent_bonus: 1
} +0ms
service:buildSave player idle value 20 +0ms
service:buildSave player click value 38 +0ms
controller:playerAccount idle value 20 +27ms
controller:playerAccount currency: 10580 (number), afk sec: 809, currency won: 16180 (number), total currency: 26760 (number) +0ms

```

5

Pour vérifier que les calculs sont bons, on va commencer par vérifier le nombre de secondes de déconnexion en partant des informations de la base de données.

`logout_time : 14:29:27`

`login_time : 14:42:56`

On obtient **13 minutes et 29 secondes**, soit **809 secondes**.

La valeur de **idle** est de **20**, soit $809 \times 20 = 16180$

La valeur annoncée à la reconnexion de 16,18k est donc correcte.

Le joueur possédait 10580 savoir, soit $10580 + 16180 = 26760$

Le gain de déconnexion a donc correctement été ajouté au savoir du joueur.

On peut donc dire que **le scénario s'est déroulé avec succès** et que **la fonctionnalité effectue correctement les actions attendues**.



H - VEILLE ET TROUBLESHOOTING

1/ Veille

Cas 1 : Les mathématiques dans les idle games

Lorsque j'ai eu l'idée du projet de idle game, je connaissais les **principales mécaniques de jeu** car j'avais déjà joué à quelques jeux de ce genre. En revanche, je ne savais pas comment traduire certaines de ces mécaniques en code. C'était particulièrement le cas pour la **mécanique des générateurs, dont le prix doit augmenter à chaque achat**, ce qui permet d'instaurer une **courbe de progression** particulière dans la partie du joueur.

J'ai donc effectué quelques recherches, et j'ai trouvé une **série d'articles sur le Developers Blog de Kongregate** intitulée *The Math of Idle Games*. C'est la première partie (<https://blog.kongregate.com/the-math-of-idle-games-part-i/>) qui m'a été la plus utile, car c'est dans celle-ci que l'auteur détaille le **fonctionnement des générateurs**, et qu'il fournit la **formule mathématique pour calculer le prix du prochain générateur en fonction du nombre possédé**, de façon à obtenir une **courbe de prix exponentielle**.

Traduction d'une ressource en anglais

Voici une traduction de la première moitié de l'article, qui traite du sujet des générateurs :

J'ai donné quelques conférences par le passé sur l'attrait et les mécaniques principales des idle games, mais si vous vouliez concrètement en faire un ? La théorie et les modèles, c'est bien, mais il y a quelques calculs compliqués derrière. Et comment diable peut-on équilibrer un jeu dont les nombres sont incroyablement grands ?

Cet article est la première partie de ce qui sera une série en trois parties, détaillant les sujets abordés dans ma récente conférence. Cette partie 1 traite des idées fondamentales de croissance, coût, prestige, et équilibrage des générateurs. La partie 2 examinera des méthodes de croissances alternatives (en particulier basées sur les dérivées). La partie 3 examinera les cycles et l'équilibrage du prestige.

Commençons par préciser quelques termes pour permettre d'en parler plus facilement.

Monnaie principale : il s'agit du nombre principal qui est incrémenté, généralement le but du jeu étant d'en avoir le plus possible. Il s'agit en général d'une forme de



monnaie.

Générateur : ce sont les objets du jeu qui produisent cette monnaie principale. Le taux de production, ou de revenu, est mesuré en monnaie par seconde.

Monnaie d'échange principale : Dans certains cas, les générateurs produisent une monnaie différente qui est échangée contre la monnaie principale. Par exemple, dans Clicker Heroes, les générateurs produisent du DPS (dégâts par seconde), qui est ensuite converti en or en tuant des monstres. Cette couche de séparation peut donner un peu plus de contrôle sur la croissance de la monnaie principale du jeu, puisqu'on peut modifier le "taux de change" au fil du temps.

Multiplicateur : Une des nombreuses améliorations qui multiplie la puissance du générateur. Il peut s'agir d'augmentations explicites, basées sur le nombre de générateurs possédés, etc. Leur but est d'amener (temporairement) la valeur de production plus proche, voire plus importante, que la valeur des coûts.

Prestige : Une réinitialisation de la plupart des éléments du jeu (en particulier les générateurs et les multiplicateurs), mais en gagnant une monnaie spéciale (monnaie de prestige) et/ou des bonus multiplicateurs persistants pour accélérer la partie suivante. C'est similaire à un "New Game+".

Au niveau le plus basique, les idle gmes sont une balance entre taux de production et coûts. Au début d'une partie, votre production dépassera les coûts, puis les coûts finiront par devenir prohibitifs. Cela est généralement rendu possible en faisant croître les coûts de manière exponentielle, tandis que la production augmente de manière linéaire ou polynomiale. Pour poser cela en formules :

coût du prochain = coût de base * facteur multiplicateur \wedge nombre possédé

taux de production = production de base * nombre possédé * multiplicateur

Dans le jeu Adventure Capitalist, pour le stand de limonade, le facteur multiplicateur vaut 1,07, le coût de base vaut 4, la production de base vaut 1,67 / sec. Donc si on possède 10 stands de limonade :

coût du prochain = $4 \times 1,07^{10} = 7,87$

tandis que le taux de production est :

taux de production = $1,67 \times 10 \times 1 = 16,7 / \text{sec}$



Cas 2 : Les JWT (Json Web Token)

Lors de la phase de **conception**, nous nous sommes posé la question de l'**authentification** pour **protéger nos routes back**. Le but étant d'une part d'être capables d'authentifier un joueur pour lui envoyer les informations de partie correspondantes, mais aussi de sécuriser l'application en empêchant un joueur d'accéder aux informations et à la partie de quelqu'un d'autre.

Nous avons donc effectué des **recherches**, et avons **échangé avec d'autres groupes** de projet. Cela nous a lancé sur la **piste des JWT**, et après avoir lu quelques **articles théoriques** sur le sujet, nous avons décidé de réaliser un **POC** (Proof Of Concept).

Nous avons donc créé un repo GitHub et nous avons mis en place une **authentification par JWT** sur une application très simple pour tester l'accès à une route renvoyant des informations issues d'une base de données, en passant par un **middleware qui vérifie la présence et la validité d'un token**.

Cela nous a été très utile, non seulement pour **vérifier la faisabilité et la pertinence** de cette solution, mais aussi pour **expérimenter et bien comprendre** le fonctionnement de ce type d'authentification, et également pour être plus à l'aise par la suite lors de la mise en place de cette solution dans le contexte de notre projet, qui était plus complexe.

2/ Résolution de problèmes

Durant la phase de développement, nous avons été confrontés à de nombreuses reprises à des **problématiques techniques**, qui ont nécessité des recherches et des tests pour être résolues.

Configuration du Swagger

Un premier exemple est celui du **Swagger** que nous avions configuré et mis en place en début de projet. Lorsque nous avons mis en place quelques jours plus tard, l'**authentification par JWT, les routes back qui étaient accessibles sont devenues des routes protégées**, qui vérifient la présence et la validité d'un token dans l'Authorization du header, et donc **les routes sont devenue inaccessible via le Swagger**.

Nous avons effectué des recherches pour voir s'il était possible de **configurer le Swagger** pour que celui-ci permette de s'authentifier et nous avons appris que cela était possible.

Nous avons alors cherché dans la **documentation de Swagger** et nous avons trouvé une section dédiée au **paramétrage de l'authentification**.



[Swagger Inspector](#)
[Open Source Tools](#)
[OpenAPI Guide](#)
[What Is OpenAPI?](#)
[Basic Structure](#)
[API Server and Base Path](#)
[Media Types](#)
[Paths and Operations](#)
[Describing Parameters](#)
[Parameter Serialization](#)
[Describing Request Body](#)
[Describing Responses](#)
[Data Models \(Schemas\)](#)
[Adding Examples](#)
[Authentication](#)
[Links](#)
[Callbacks](#)
[Components Section](#)
[Using `sref`](#)
[API General Info](#)
[Grouping Operations With Tags](#)
[OpenAPI Extensions](#)
[2.0](#)
[What Is Swagger](#)
[Basic Structure](#)
[API Host and Base Path](#)
[MIME Types](#)
[Paths and Operations](#)
[Describing Parameters](#)
[Describing Request Body](#)
[File Upload](#)
[Describing Responses](#)
[Authentication](#)
[API Keys](#)

Authentication

Swagger 2.0 lets you define the following authentication types for an API:

- [Basic authentication](#)
- [API key \(as a header or a query string parameter\)](#)
- [OAuth 2 common flows \(authorization code, implicit, resource owner password credentials, client credentials\)](#)

Follow the links above for examples specific to these authentication types, or continue reading to learn how to describe authentication in general.

Authentication is described by using the `securityDefinitions` and `security` keywords. You use `securityDefinitions` to define all authentication types supported by the API, then use `security` to apply specific authentication types to the whole API or individual operations.

The `securityDefinitions` section is used to define all security schemes (authentication types) supported by the API. It is a name->definition map that maps arbitrary names to the security scheme definitions. Here, the API supports three security schemes named `BasicAuth`, `ApiKeyAuth` and `OAuth2`, and these names will be used to refer to these security schemes from elsewhere:

```
1. securityDefinitions:
2.   BasicAuth:
3.     type: basic
4.   ApiKeyAuth:
5.     type: apiKey
6.     in: header
7.     name: X-APT-Key
8.   OAuth2:
9.     type: oauth2
10.    flow: accessToken
11.    authorizationUrl: https://example.com/oauth/authorize
12.    tokenUrl: https://example.com/oauth/token
13.    scopes:
14.      read: Grants read access
15.      write: Grants write access
16.      admin: Grants read and write access to administrative information
```

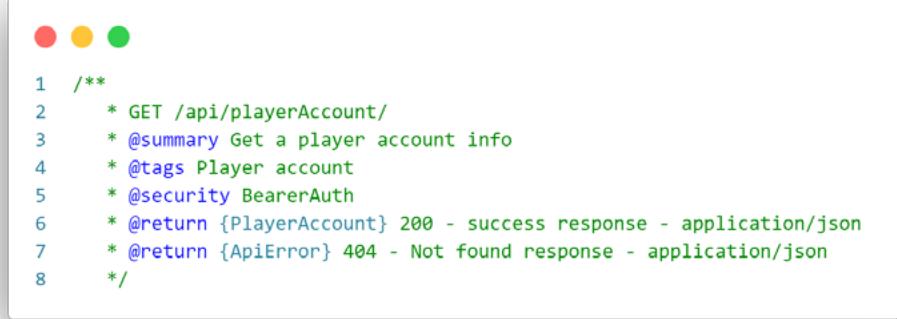
Each security scheme can be of `type`:

- [basic](#) for Basic authentication
- [oauth2](#) for an API key

Cependant, après différents tests, cela ne fonctionnait toujours pas. C'est en consultant des pages de **Stackoverflow** que nous avons compris notre **erreur**. Nous avions cherché sur la documentation de Swagger, mais nous utilisions **Swagger via un package spécial pour NodeJs et Express**, nommé **express-jsdoc-swagger**. Nous avons consulté la **documentation et les exemples fournis sur le GitHub du package**, et c'est comme ça que nous avons trouvé les **bonnes options de configuration** à passer à notre Swagger.



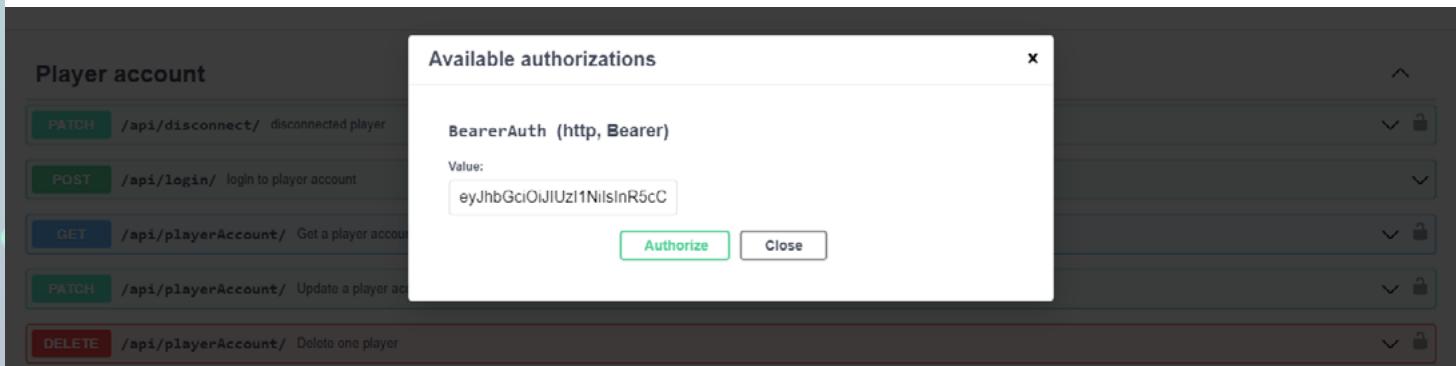
```
1 security: {
2   BearerAuth: {
3     type: 'http',
4     scheme: 'bearer',
5   },
6 }
```



```
1 /**
2  * GET /api/playerAccount/
3  * @summary Get a player account info
4  * @tags Player account
5  * @security BearerAuth
6  * @return {PlayerAccount} 200 - success response - application/json
7  * @return {ApiError} 404 - Not found response - application/json
8 */
```



Nous avons alors pu constater que des boutons et icônes étaient apparus sur les routes protégées de notre Swagger, permettant de s'authentifier en entrant un JWT.



Grands nombres, et limite du type integer

Un second exemple intéressant est celui des **grands nombres**. En effet, les idle games sont des jeux dans lesquels on est vite amené à **manipuler des très grands nombres**, vu que les **valeurs de revenus ne cessent d'augmenter**, et que les **valeurs de coûts augmentent de manière exponentielle**.

Lorsque nous avions conceptualisé la base de données durant le Sprint 0, dans le **dictionnaire des données**, nous avions défini nos colonnes qui devaient recevoir des entiers en **type INTEGER**, un type de **PostgreSQL** correspondant à un entier.

Cependant, lorsque nous avons commencé à tester l'API back, nous avons été confronté au **message d'erreur** suivant :

```
"value \"100000000000\" is out of range for type integer"
```

En effet, le type **INTEGER** de **PostgreSQL** dispose d'un espace de stockage de **4 bytes**, et permet de stocker un entier dans l'intervalle $[-2147483648, +2147483647]$, or la valeur qui avait été entrée lors du test était supérieure à 2147483647.

En consultant la **documentation de PostgreSQL**, nous avons appris qu'il existait un type **BIGINT**, disposant d'un espace de stockage de 8 bytes, soit deux fois plus que le type **INTEGER** et permettant de stocker un entier dans l'intervalle $[-9223372036854775808, +9223372036854775807]$.

Nous avons donc effectué une **migration sur la base de données** pour utiliser le **BIGINT** à la place du **INTEGER** sur les colonnes susceptibles de stocker de telles valeurs.

Mais **cette solution n'est pas parfaite**. Comme le jeu n'a pas de fin, et que les valeurs ne cessent d'augmenter, **il est théoriquement possible d'atteindre la limite du type BIGINT**, et donc d'avoir un message d'erreur du même genre que celui vu précédemment.



Nous avons **exploré d'autres pistes**, une première consistant à **décomposer nos grandes valeurs en plusieurs colonnes** de différents ordres de grandeur (dans l'idée d'une colonne pour les unités, une autre pour les dizaines, et les centaines...), une seconde consistant à **convertir la valeur reçue en string, et la stocker en type TEXT**, éliminant définitivement la limite.

Finalement, après discussions, nous avons conclu que le jeu, avec son équilibrage actuel, ne permettait pas d'atteindre la limite du BIGINT (cela est possible dans la théorie, mais il aurait fallu un temps de jeu incroyablement long), et donc **nous avons choisi, pour cette V.1, de laisser le BIGINT en place**.



I - CONCLUSION

Cette expérience de projet en groupe a été pour moi une **période intense et enrichissante**.

Aussi bien d'un point de vue **technique**, car cela m'a permis de mettre en pratique le fruit de 4 mois intenses de formation, mais aussi d'appréhender de nouvelles technologies, de me confronter à des problématiques et de les résoudre.

Mais également d'un point de vue **humain**, car le travail en équipe nous a demandé coordination, organisation, et communication. Il a fallu faire des concessions, et cela n'a pas toujours été facile, mais nous avons réussi à maintenir une cohésion et une bonne entente tout au long du projet. Nous avons vu ensemble notre jeu prendre forme grâce au travail accompli par chacun, et c'est quelque chose que je ne suis pas prête d'oublier !

Au final, j'en ressors avec un sentiment de fierté, pour avoir accompli autant en si peu de temps, mais également un grand sentiment d'humilité, de me rendre compte à quel point le monde du développeur est vaste et les connaissances infinies, et que je n'ai finalement fait que franchir la porte, reste maintenant un long chemin à parcourir, celui de l'expérience.



ANNEXES

User Stories

En tant que	Je veux pouvoir	Afin de
guest / player	afficher le jeu sur mobile ou bureau	varier les plateformes sur lesquelles je veux m'amuser
guest / player	afficher des informations sur le projet	lire la présentation de l'équipe, les remerciements
guest / player	voir les stats principales	savoir combien je gagne de savoir et combien j'en ai accumulé.
guest	accéder au jeu sur la page d'accueil	pouvoir cliquer pour lire des livres et accumuler du savoir (limité à 1)
guest	cliquer sur un bouton "register" pour avoir un formulaire d'inscription	créer un compte
guest	cliquer sur un bouton "login" pour avoir un formulaire de connexion	me connecter à mon compte et reprendre ma partie
guest	voir un message m'avertissant que ma partie sera limitée et ne sera pas sauvegardée si je ne suis pas connecté	ne pas être pris au dépourvu, pouvoir sauvegarder ma partie et avoir accès à tout le jeu
player	sauvegarder manuellement	enregistrer ma progression
player	cliquer sur un bouton "account"	visualiser les informations de mon compte
player	cliquer sur un bouton "logout"	me déconnecter de mon compte
player	cliquer sur un bouton "delete account"	supprimer mon compte
player	cliquer pour accumuler du savoir	accumuler du savoir à dépenser pour le shop
player	accumuler du savoir en automatique	accumuler du savoir à dépenser pour le shop
player	cliquer sur un bouton "shop" (mobile)	pouvoir afficher les améliorations disponibles
player	voir facilement quels items sont achetables	prioriser mes achats

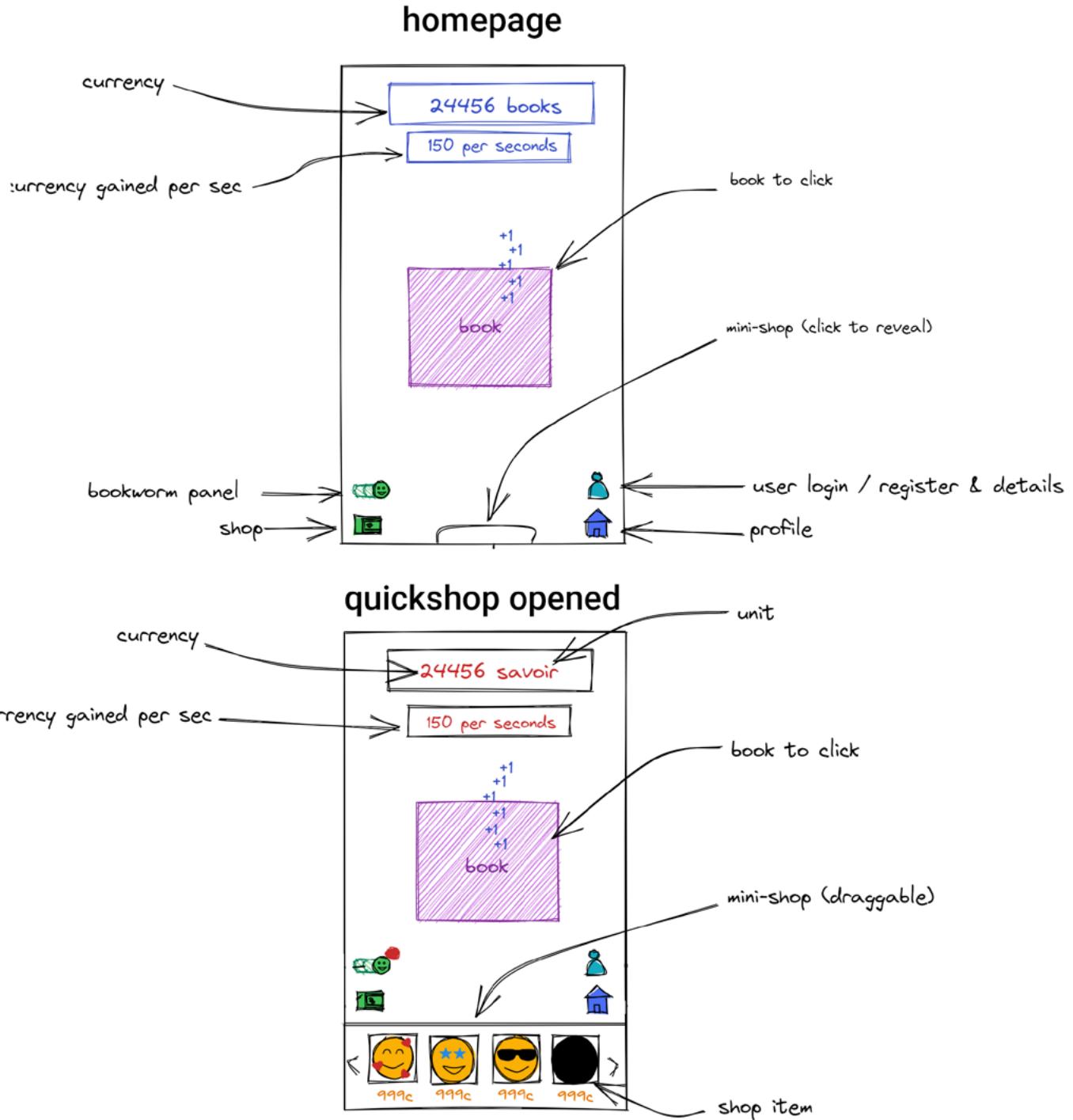


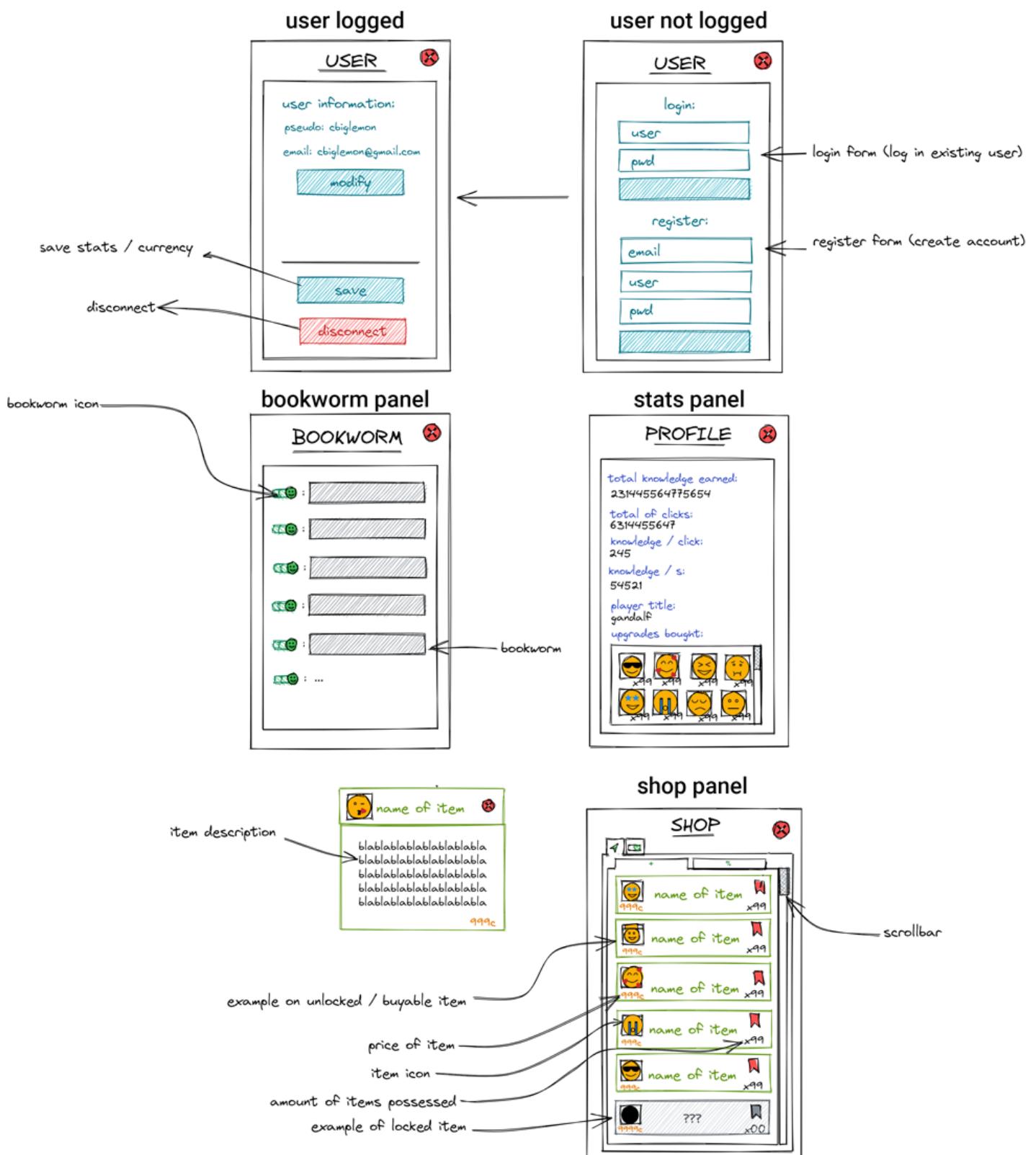
En tant que	Je veux pouvoir	Afin de
player	savoir de quel type d'item il s'agit (idle/ click / flat/ pourcent)	ne pas avoir à lire les détails d'un item pour savoir de quel type il s'agit
player	Voir apparaître en grisé avec des ??? le prochain item débloquable avec son coût	avoir un objectif et donner l'envie
player	cliquer sur un item du shop	pour acheter l'item et améliorer mes stats, accumuler plus de savoir
player	avoir un shop rapide disponible à l'écran principal avec un carrousel des achats disponibles (mobile)	acheter rapidement des items sans avoir à ouvrir la fenêtre du shop
player	recevoir de la savoir lorsque je me reconnecte	recevoir le total de savoir gagnée pendant le temps de déconnexion
player	cliquer sur un bouton statistiques	afficher les statistiques de ma partie (nombre de clics, savoir gagné, items achetés...)
player	cliquer sur un bouton pour voir les détails d'un item depuis le shop	connaître les effets de l'item et voir la description de l'item
player	cliquer sur un bouton pour voir les détails d'un item depuis les statistiques	connaître les effets de l'item et voir la description de l'item
player	recevoir des messages d'incitation à la lecture	quitter le jeu pour lire un livre
player	avoir ma partie sauvegardée automatiquement régulièrement	ne pas perdre ma progression si je n'ai pas sauvegardé manuellement
player	avoir ma partie sauvegardée à chaque fois que je fais une action importante (achat d'un item dans le shop)	ne pas perdre ma progression si je n'ai pas sauvegardé manuellement
player	exécuter une combinaison de touches de type Konami Code	avoir une surprise rigolote à l'écran (animation colorée)



Wireframes

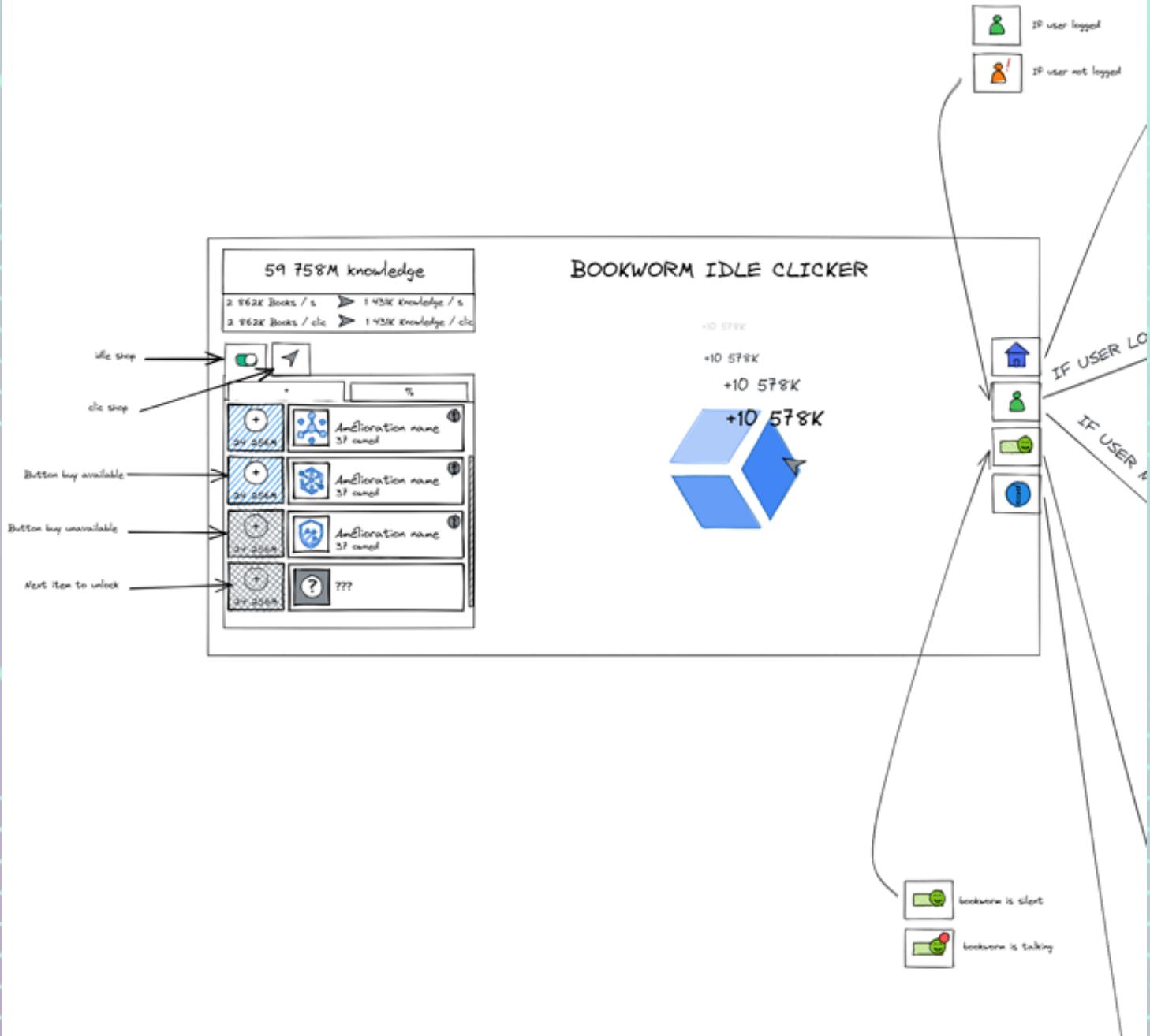
Mobile





Desktop

Accueil



Stats panel

BOOKWORM IDLE CLICKER

+10 578K
+10 578K
+10 578K

STATS

Total books read: 14 586 B
Total accumulate: 878 125 M
Total amount of click: 18 K
Amplification unlocked:

stat page ←
user tab (if online) ←
bookworm tab (no notifications) ←
info tab (the product, creators, etc...) ←

Account panel if logged

BOOKWORM IDLE CLICKER

+10 578K
+10 578K
+10 578K

MY ACCOUNT

Pseudo: PGMLxxOr31
Email: kevindu31@hotmail.fr
MODIFY
SAVE
DELETE ACCOUNT

Account panel if not logged

BOOKWORM IDLE CLICKER

+10 578K
+10 578K
+10 578K

LOG IN

You are not logged
log in to save your game

Pseudo: PGMLxxOr31
Email: kevindu31@hotmail.fr
Pswd:
Confirm:
CREATE ACCOUNT

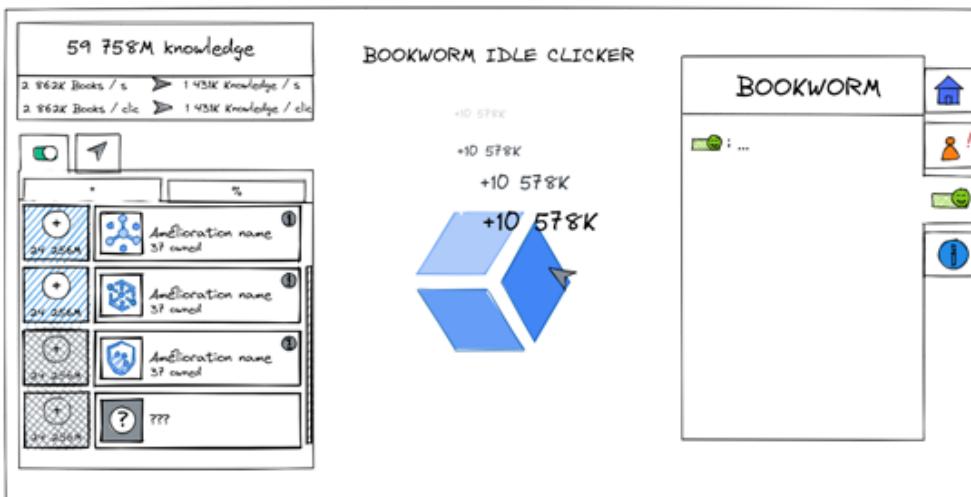
OR

Email: kevindu31@hotmail.fr
Pswd:
LOG IN

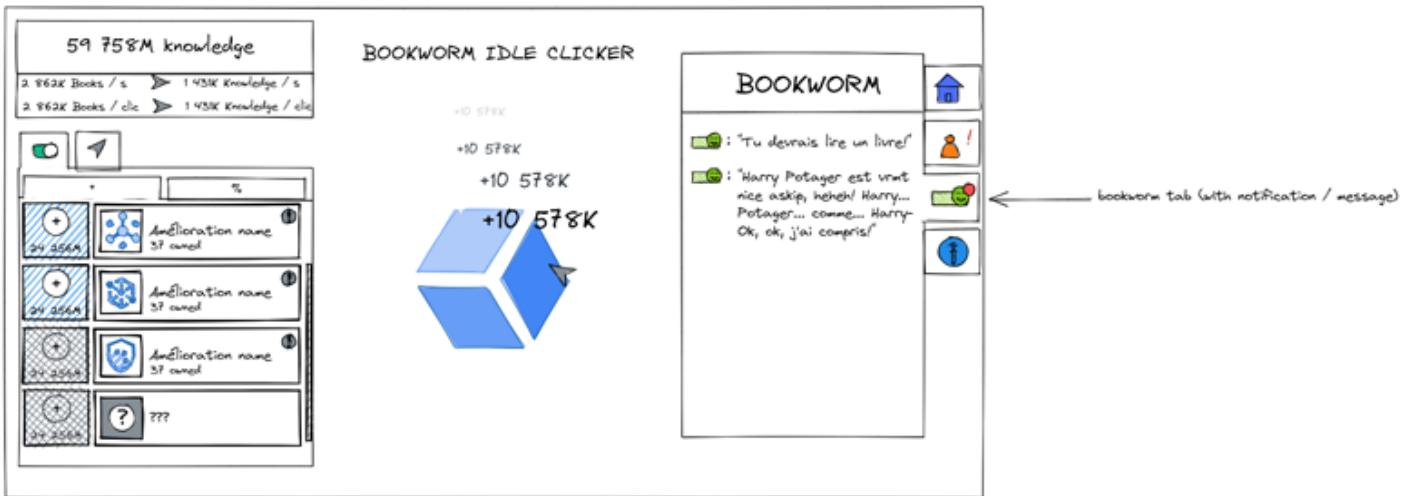
user tab (if NOTonline) ←



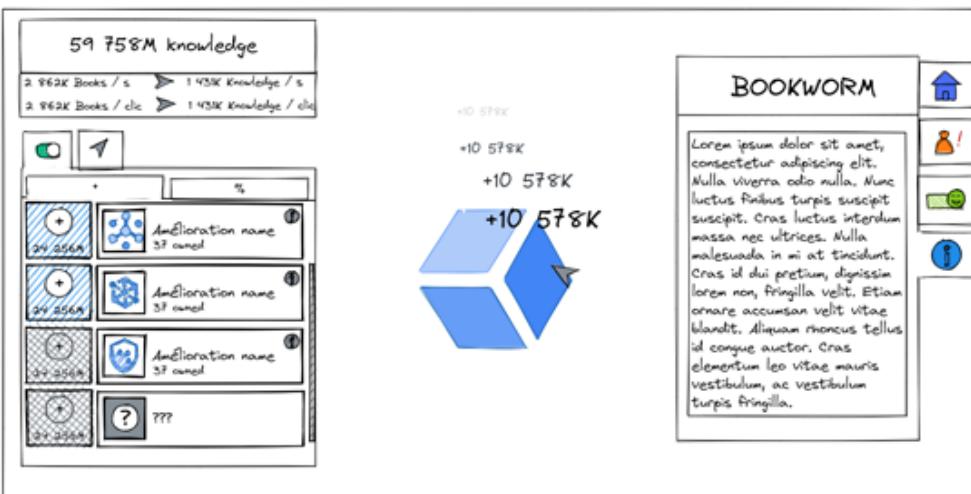
Bookworm panel



Bookworm notif



Infos panel

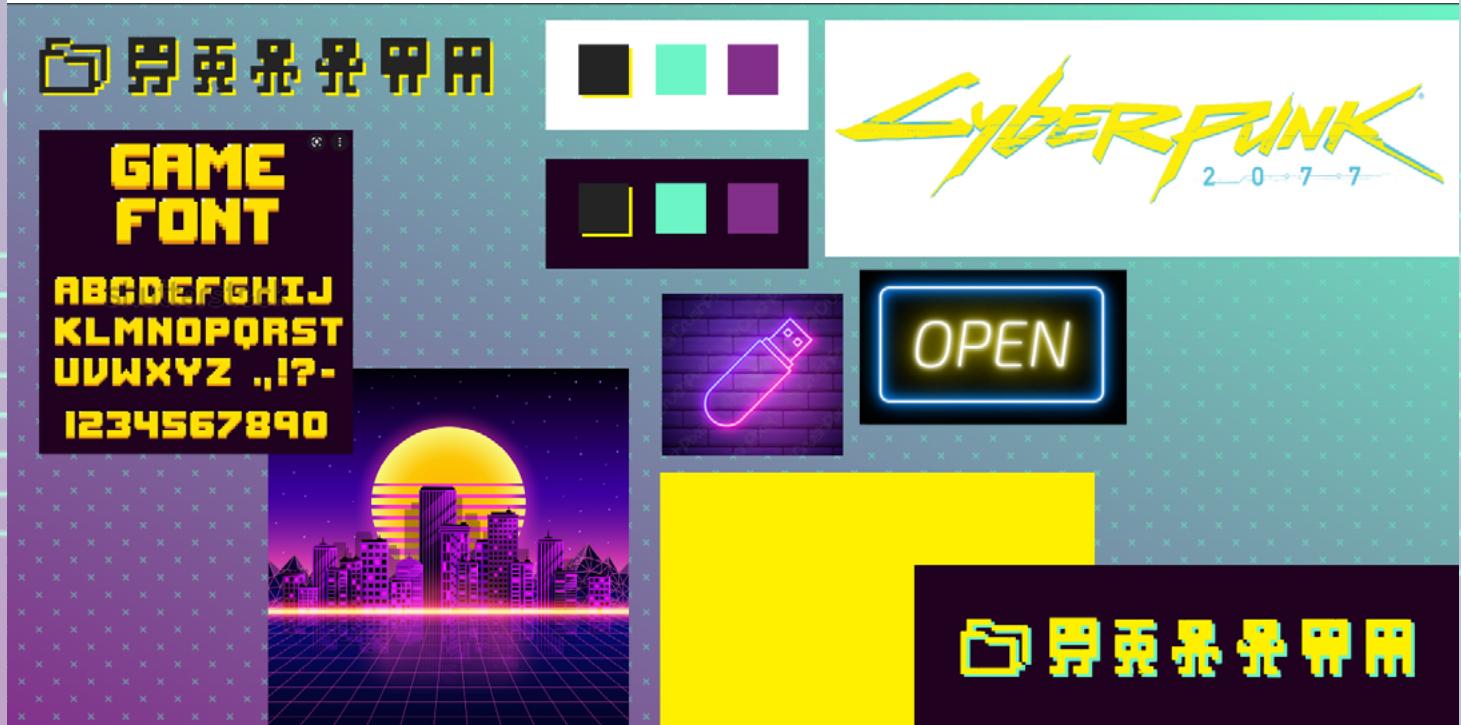


Moodboards

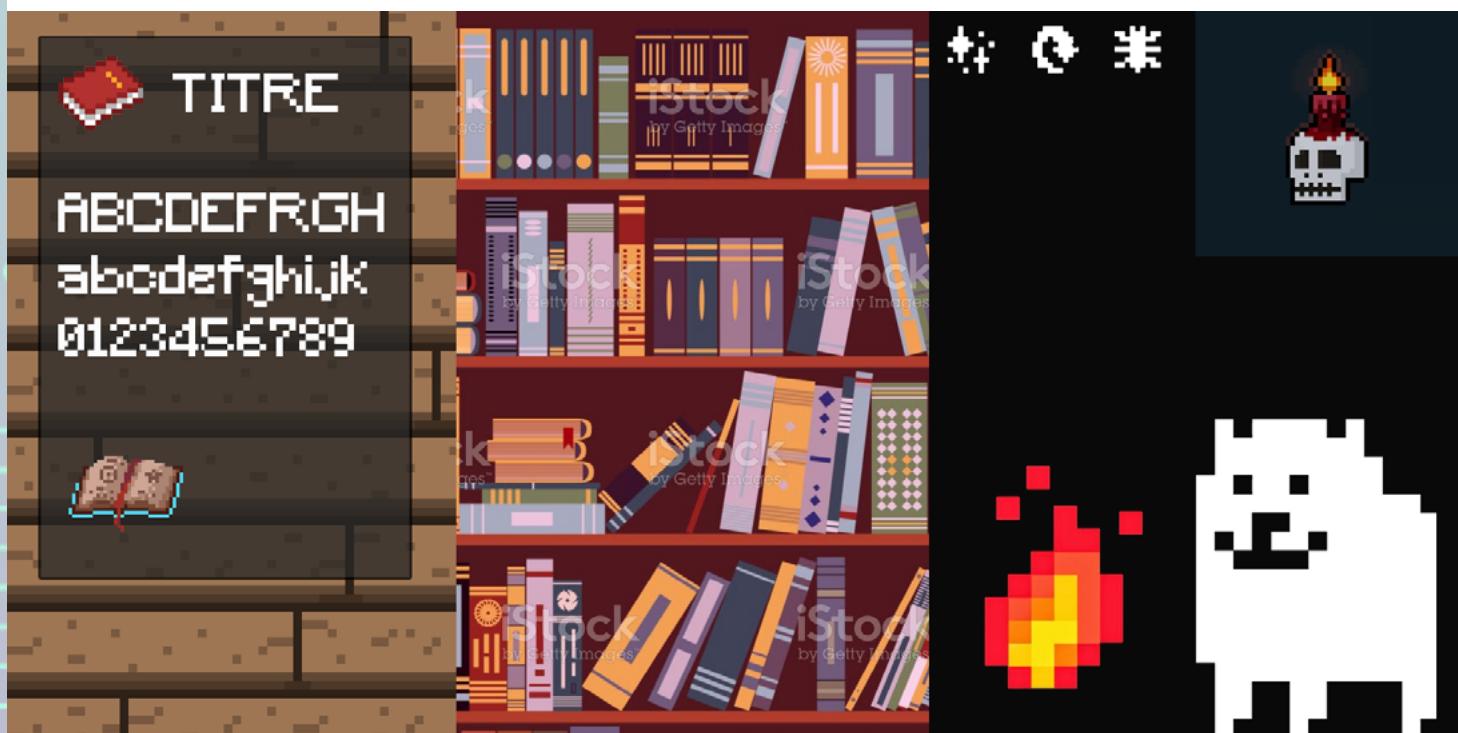
Moodboard 1

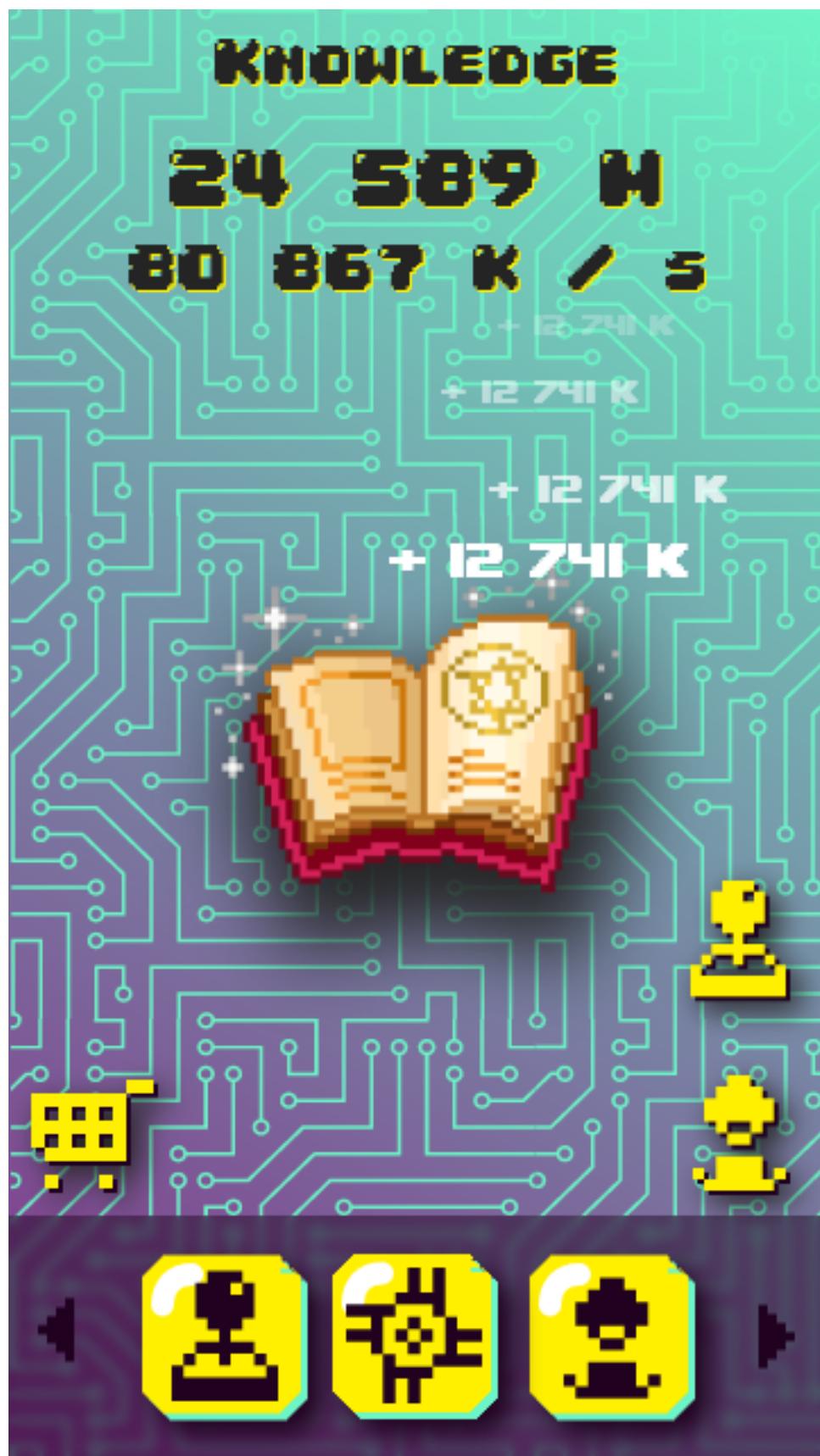


Moodboard 2



Moodboard 3

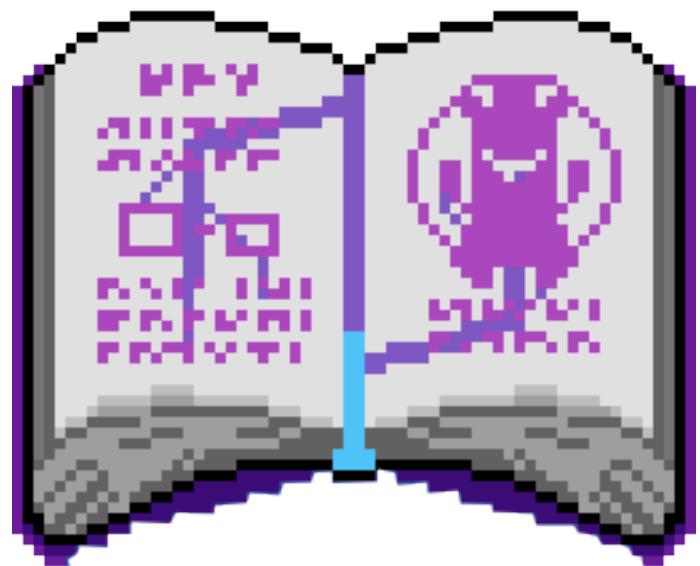




Assets graphiques

Assets graphiques réalisés par une ami pour le projet Bookworm sous forme de gif animé

Book



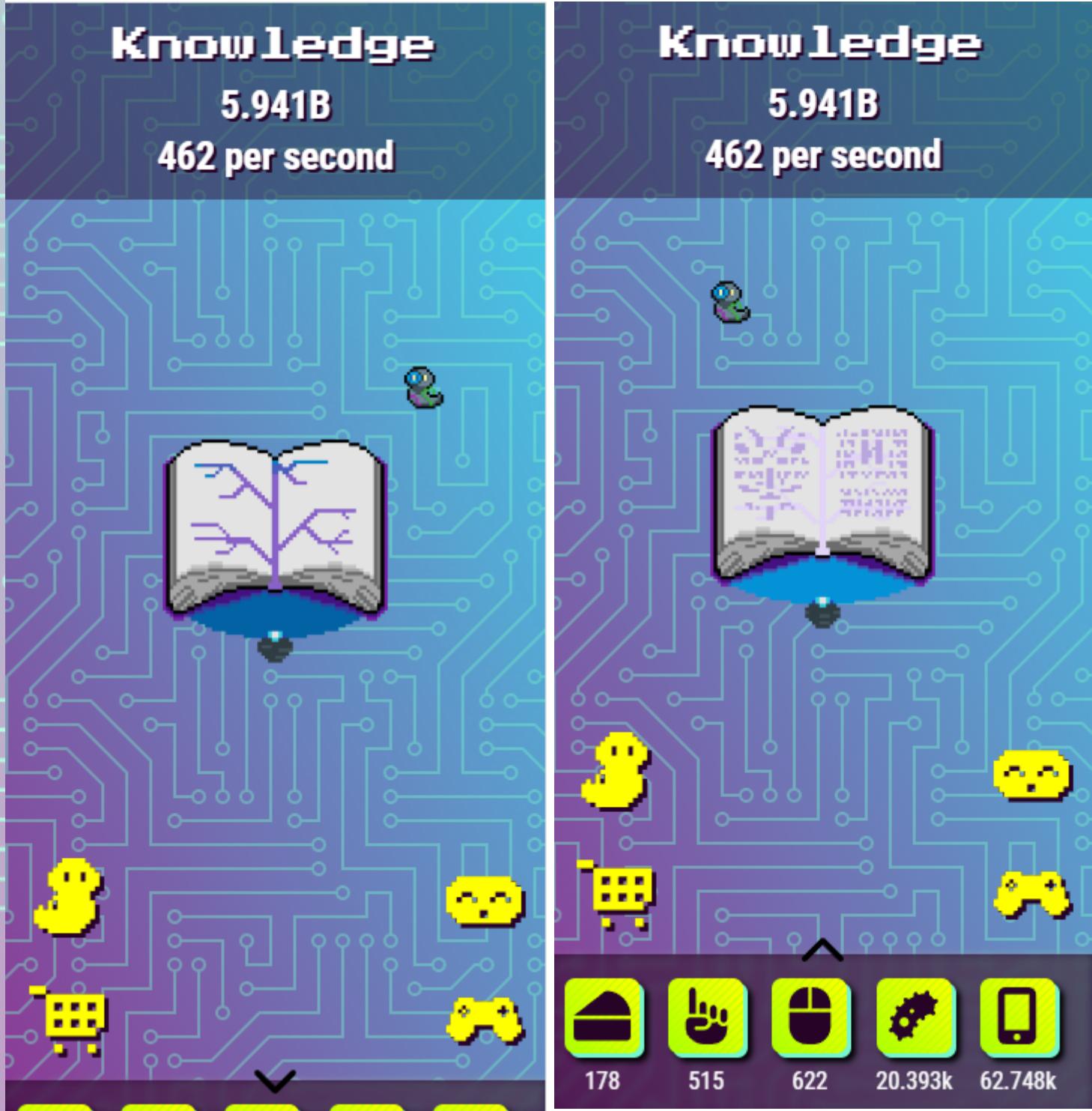
Bookworm



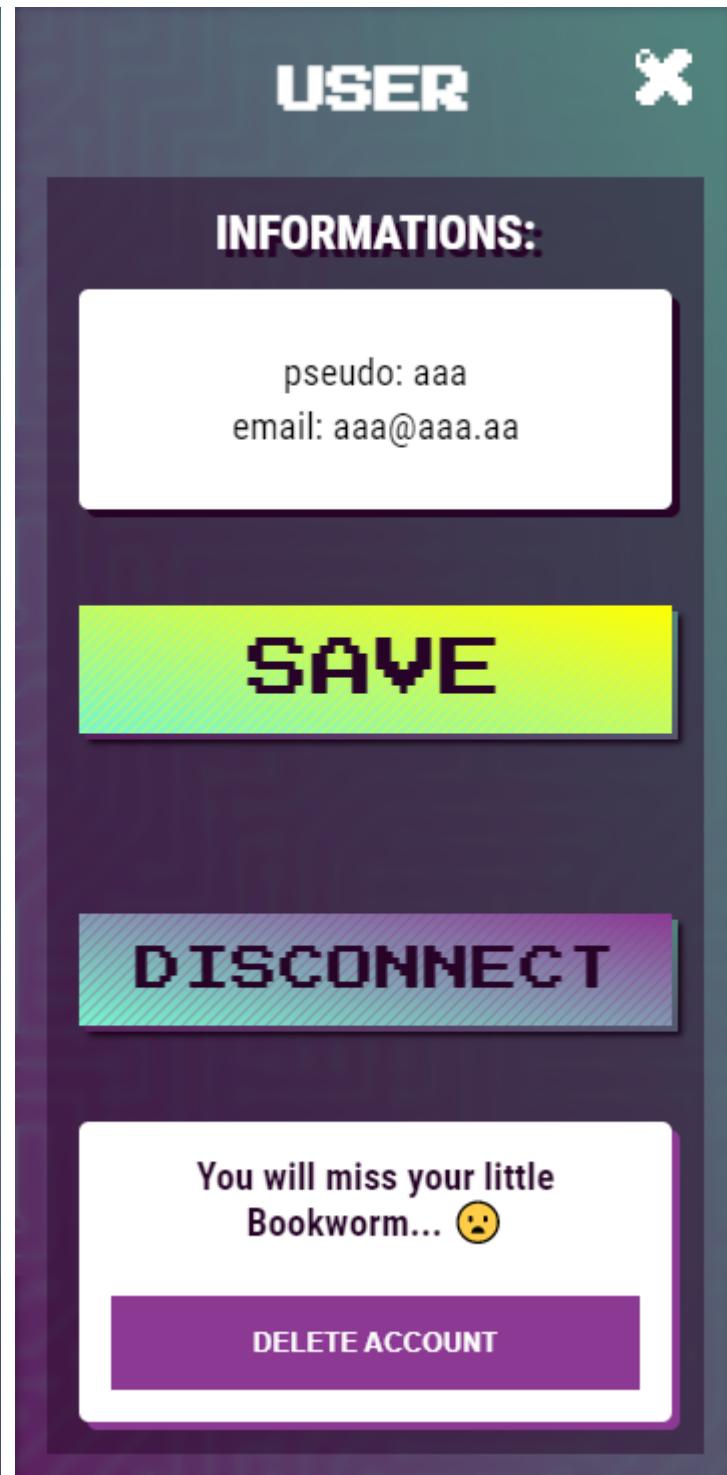
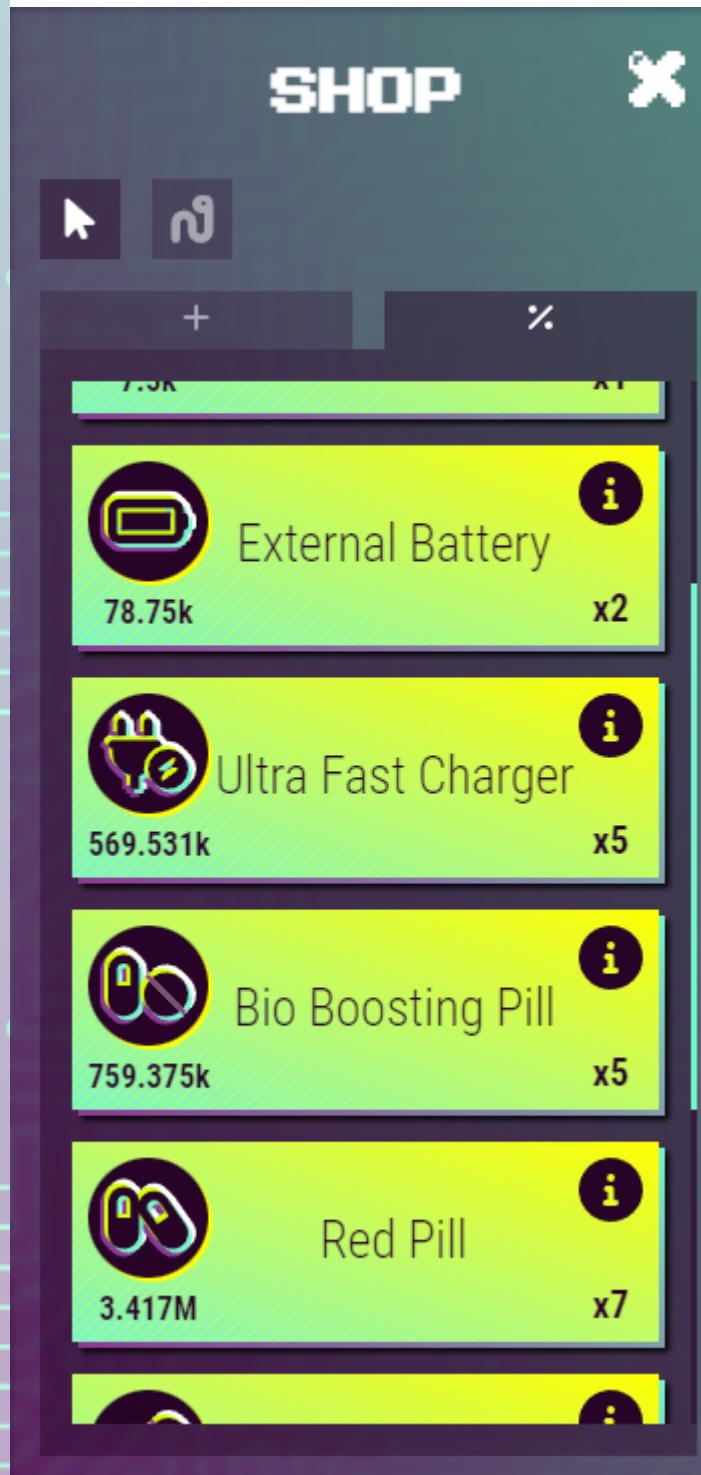
Screenshots

Mobile

home + quickshop opened



shop panel + user panel



stats panel + bookworm panel

STATS X

Knowledge owned:

5.941B

Total clicks:

547

Knowledge per second:

462

Knowledge per click:

15.642k

Upgrades bought:

12

9

5

8



7

3

1

1



1

2

5

5



7

7

8

6



BOOKWORM X



This is the story of a wrongly convicted man whose sudden wealth will make the bad fortune of his enemies. You get it ? Go read it !

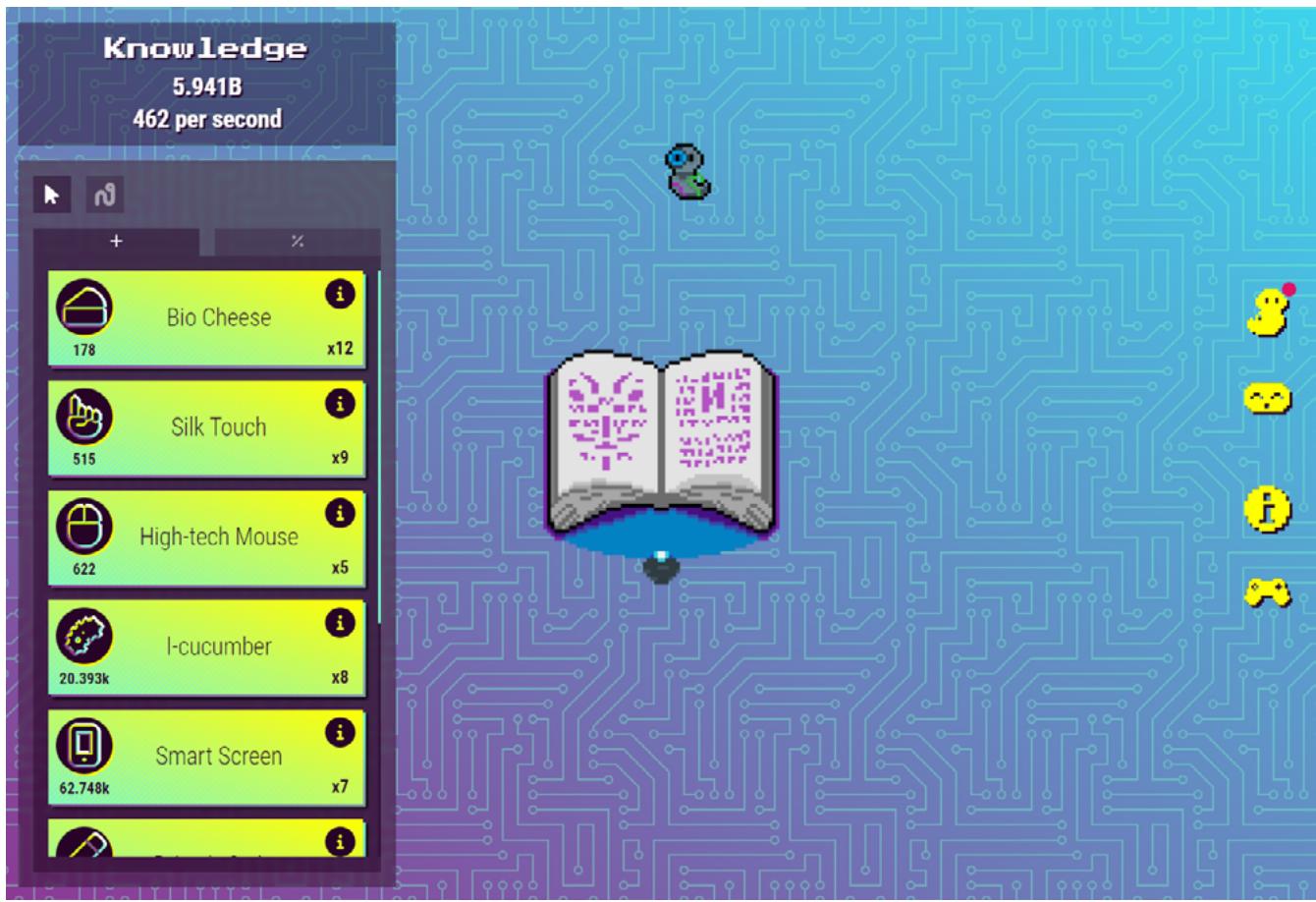


This is the story of a man who cannot see himself in paint. You get it ? Go read it !

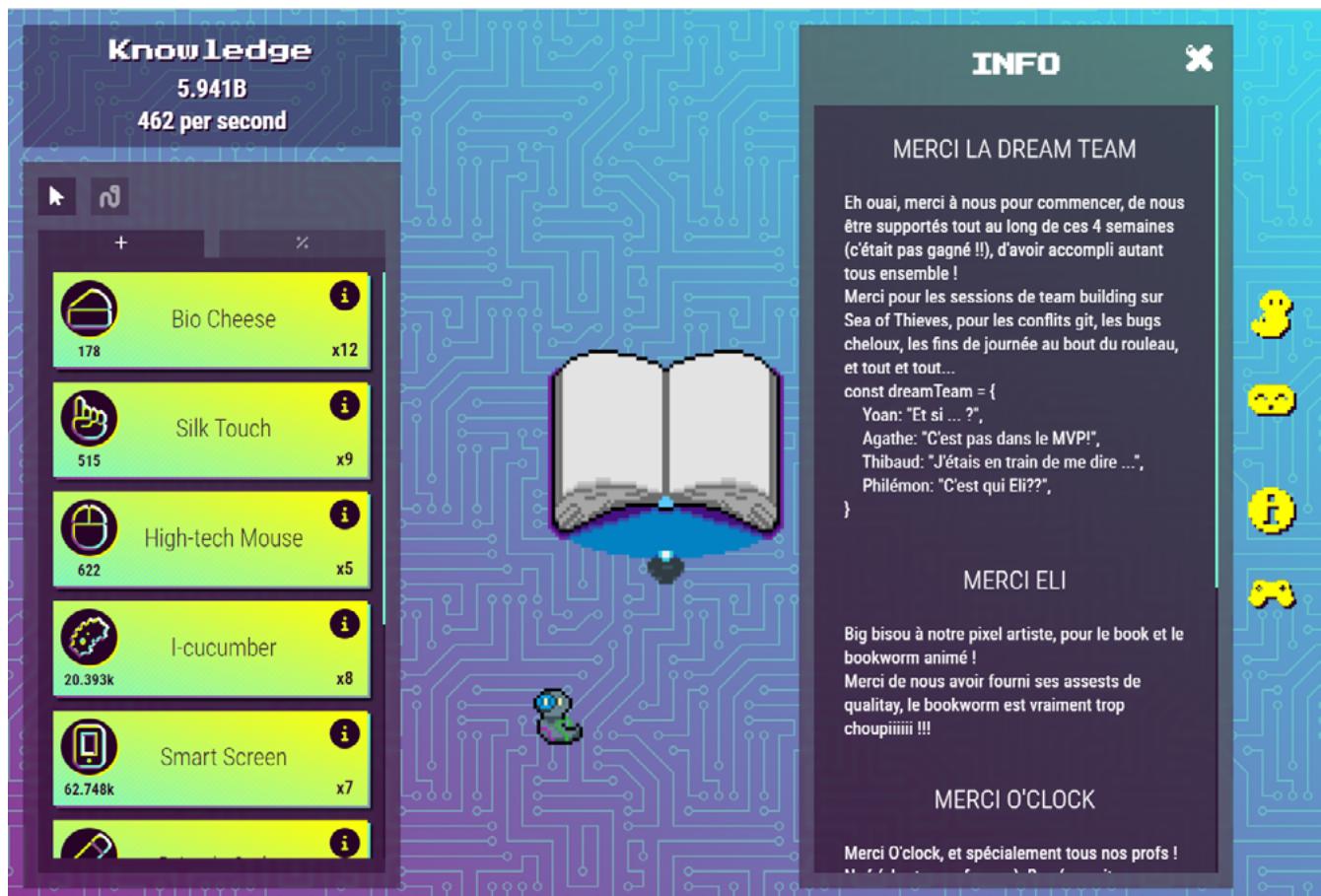


Desktop

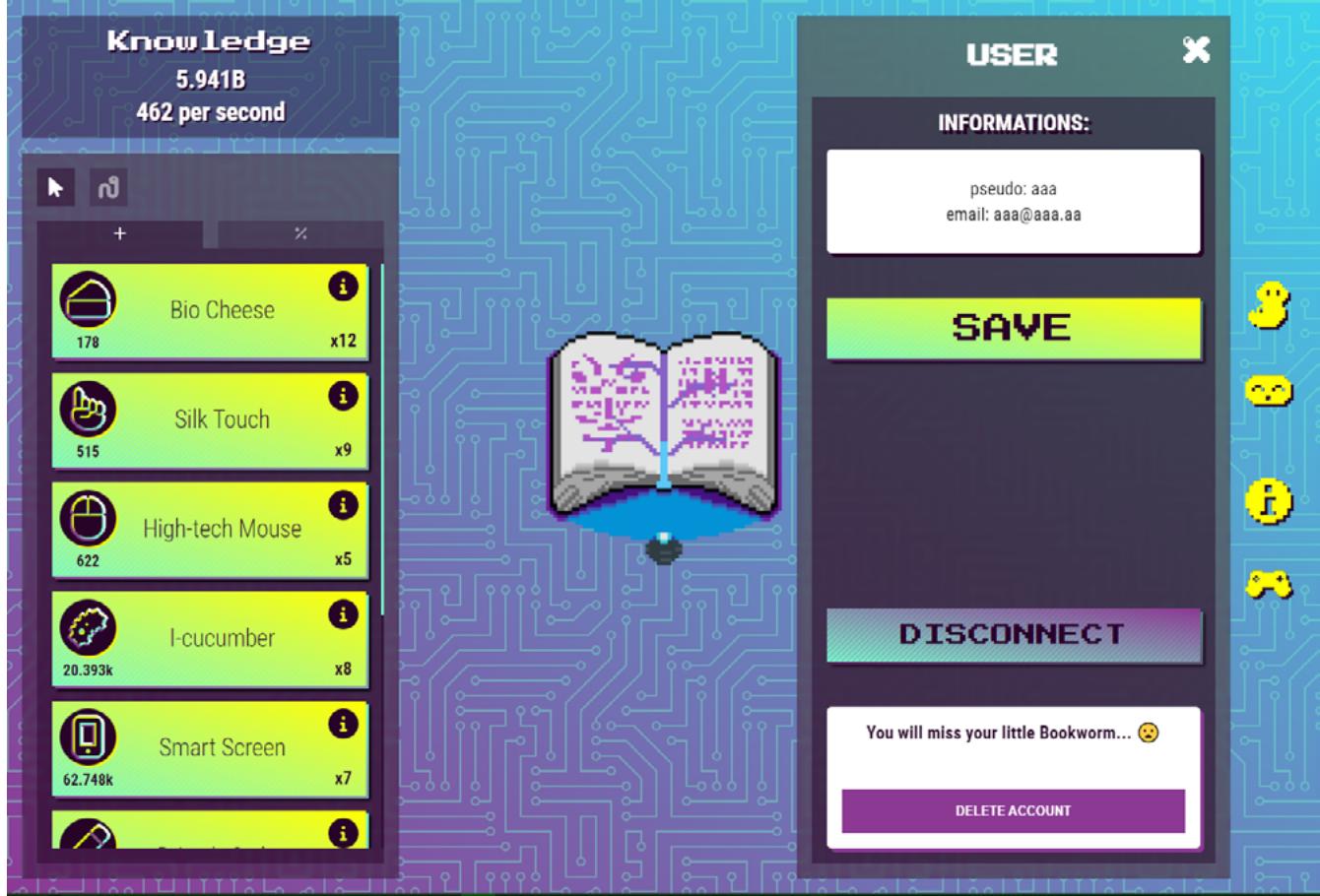
home



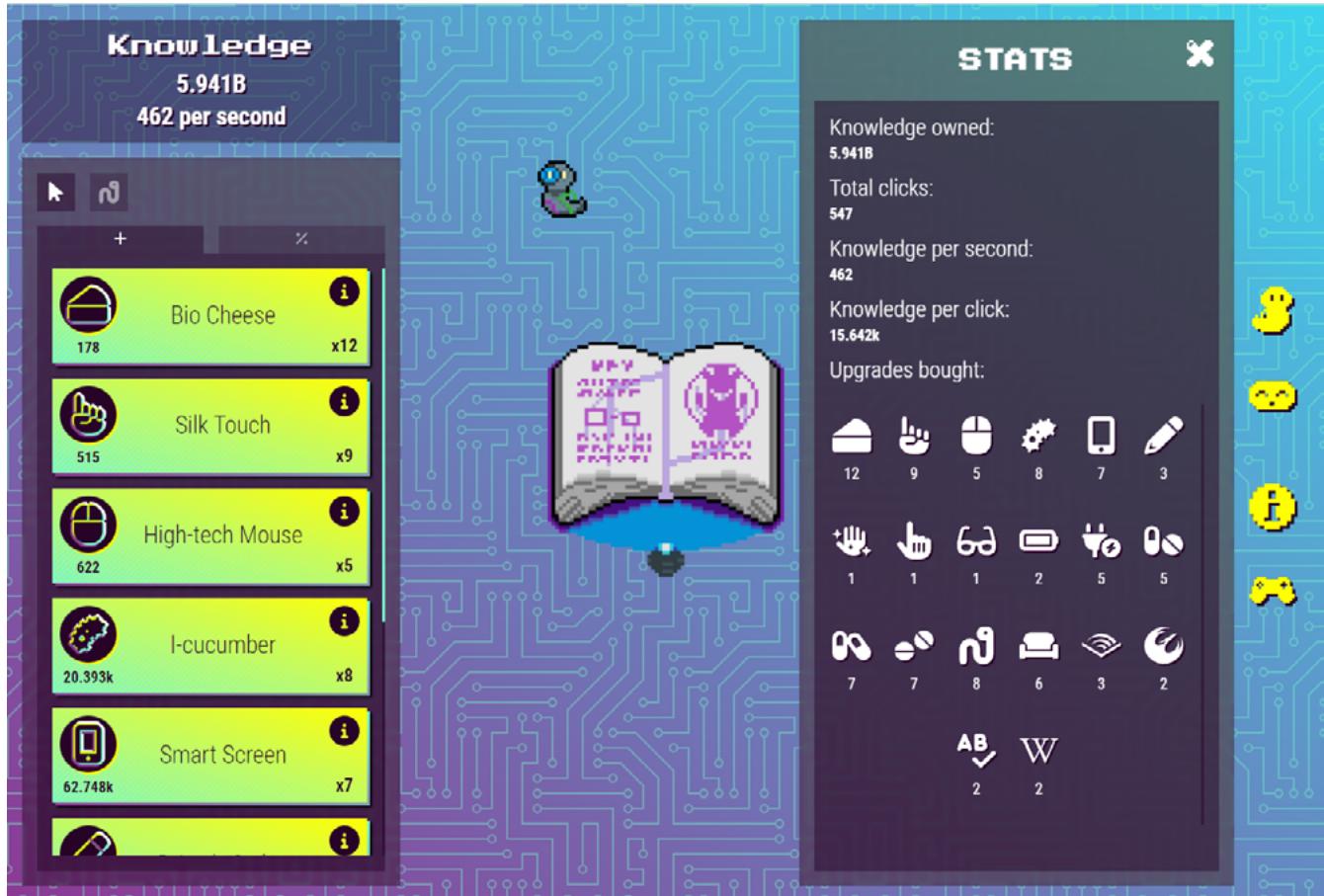
infos panel



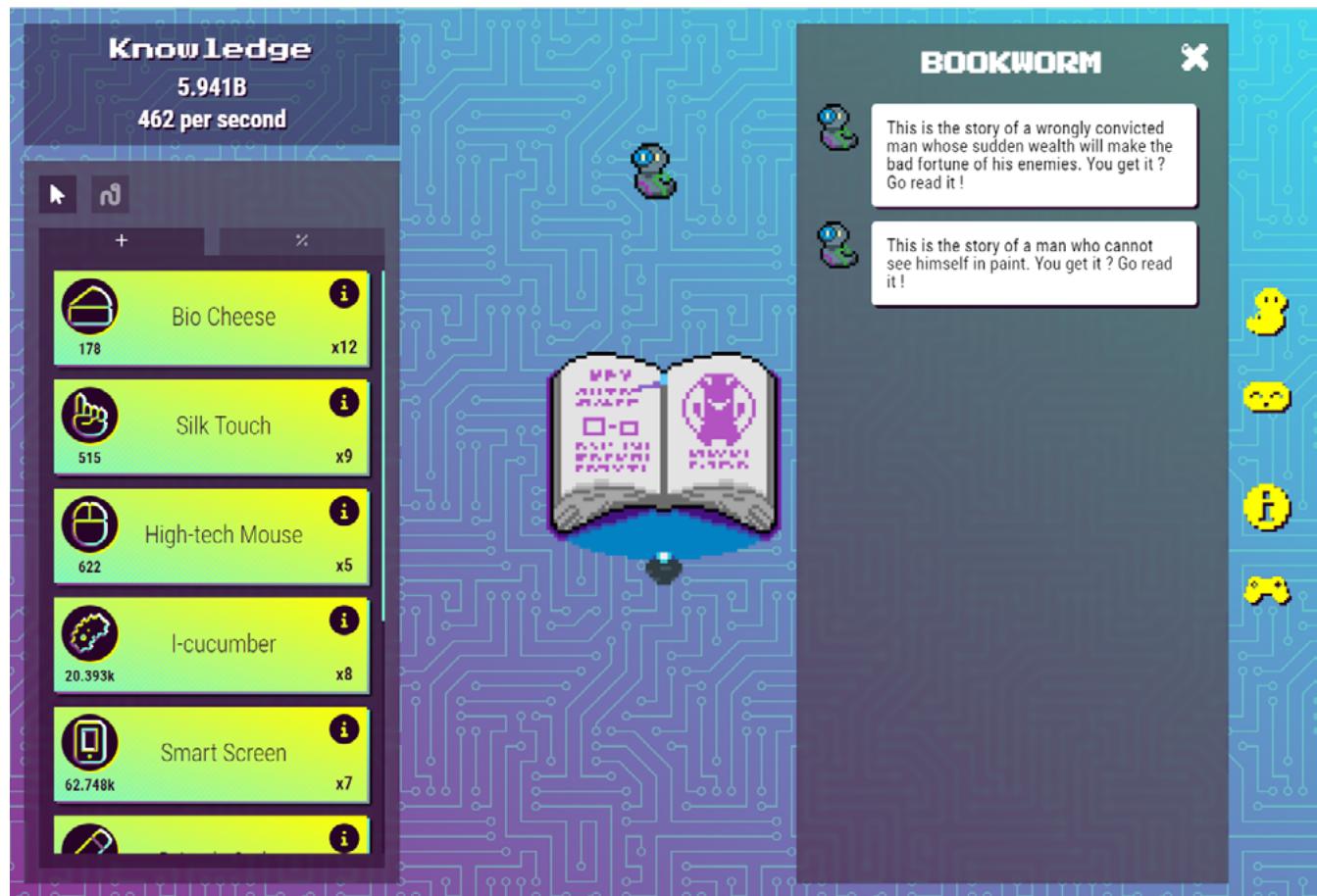
user panel



stats panel



bookworm panel



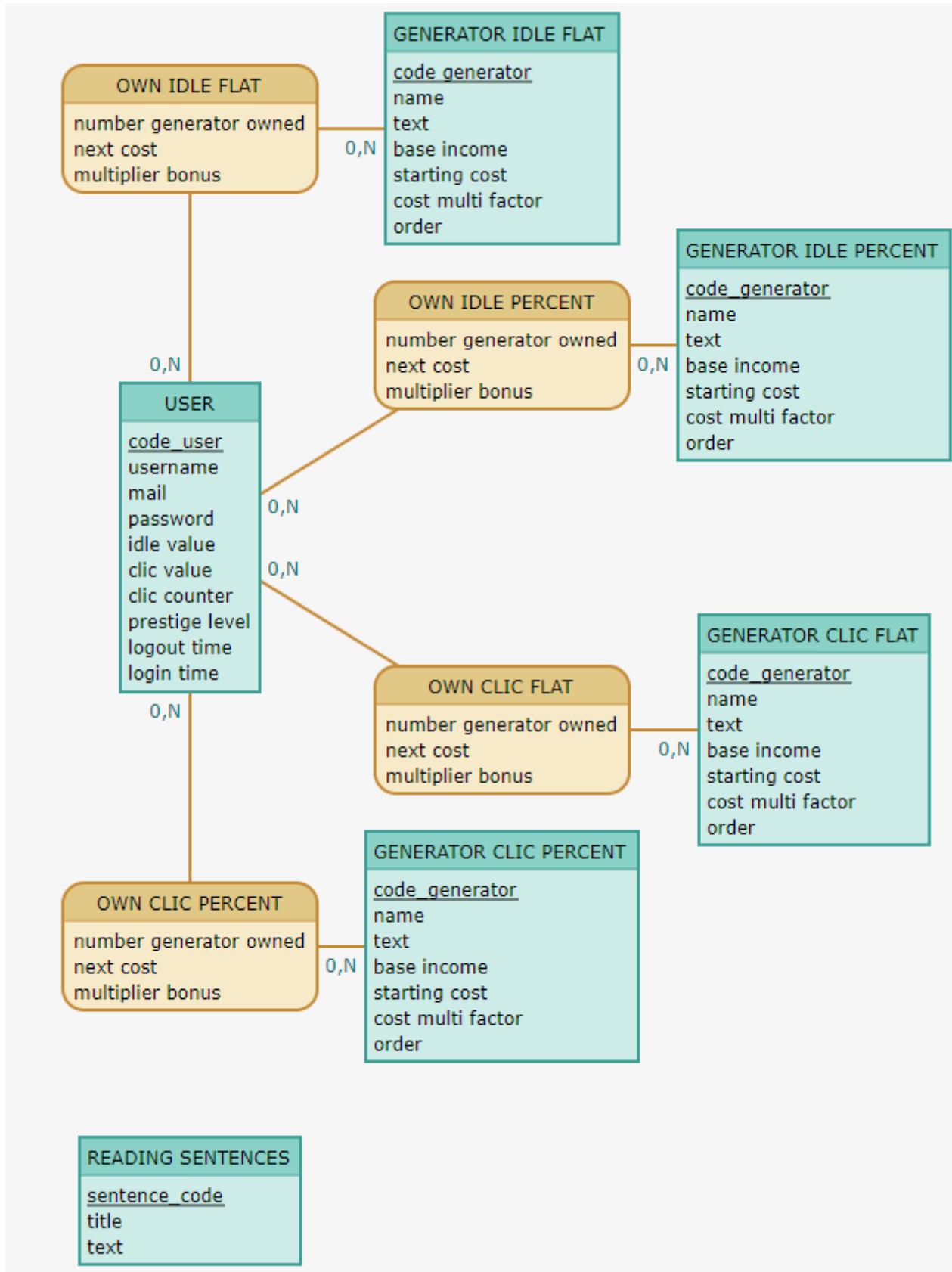
Liste des endpoints

Endpoint	Type	Data
endpoint/api/playerAccount	POST	Formulaire d'inscription : username, email, password et password confirmation, retourne le json de partie (nouvelle partie) du player
endpoint/api/playerAccount	PATCH	Formulaire de modification de compte : username, et/ou email, et/ou password et password confirmation, retourne le username et le mail Route protégée par bearer authentification
endpoint/api/playerAccount	DELETE	Suppression du compte utilisateur (table d'association player_owns_generator et table player) Route protégée par bearer authentification
endpoint/api/login	POST	Formulaire de login : mail et password, retourne le json de partie du player
endpoint/api/disconnect	PATCH	Bouton disconnect : Envoie les informations du user mises à jour en front (currency + clic counter) Détruire le token Route protégée par bearer authentification
endpoint/api/save/	PATCH	Envoie les informations du user mises à jour en front (currency + clic counter), retourne le json de partie du player Route protégée par bearer authentification
endpoint/api/save/item/:id	POST	Si item pas déjà possédé : ajout de l'item dans la table d'association + update sur currency et clic counter, retourne le json de partie du player Route protégée par bearer authentification
endpoint/api/save/item/:id	PATCH	Si item déjà possédé au moins une fois, mise à jour du nombre d'items + update sur currency et clic counter, retourne le json de partie du player Route protégée par bearer authentification
endpoint/api/sentence	GET	Envoie une sentence au hasard parmi toutes sauf la première
endpoint/api/sentence/first	GET	Envoie la première sentence

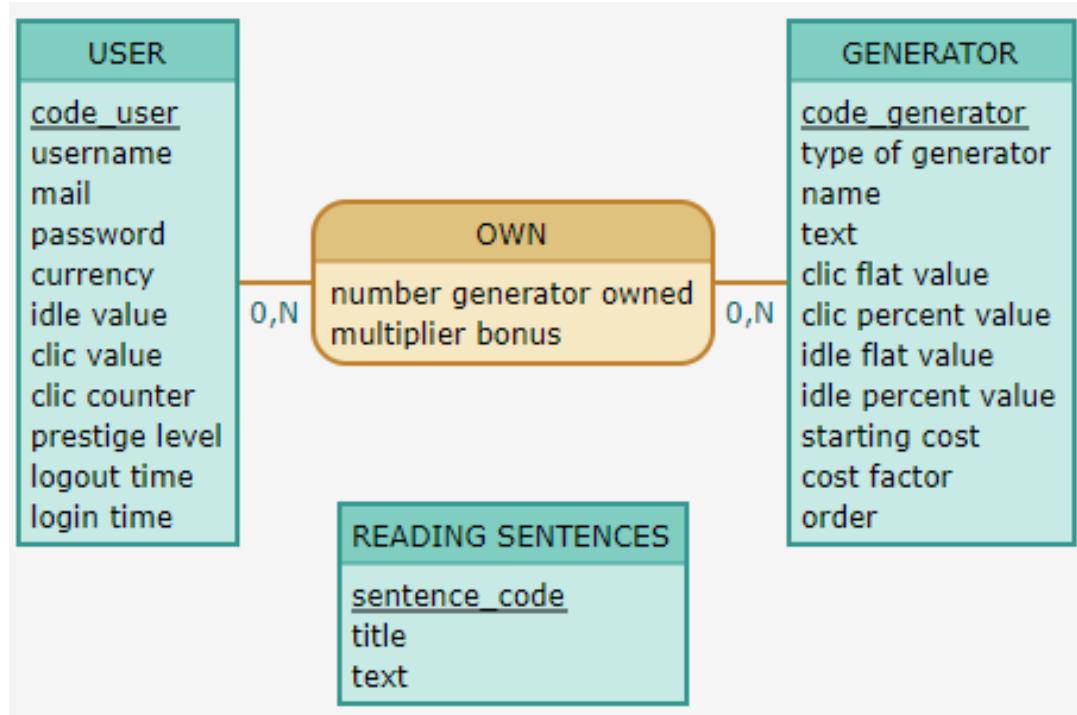


MCD

Version 1



Version 2



MLD

MLD V3 - Fonctionnel

- player (codePlayer, Username, Email, Password, currency, IdleValue, ClickValue, clickCounter, PrestigeLevel, LogoutTime, LoginTime)
- generatorIdlePercent (codeGenerator, type of generator, name, text, clic flat value, clic percent value, idle flat value, idle percent value, starting cost, cost factor, order)
- sentence(code_sentence, title, text)
- OWN (Player(#codePlayer), GeneratorIdleFlat(#codeGenerator), numberOwned, multiplierBonus)

Dictionnaire des données

Table player

Table player

Nom de la colonne	Description	Type
id	PK, identifiant unique du player	INTEGER
username	pseudo du player	TEXT
mail	adresse mail du player	TEXT
password	mot de passe du player	TEXT
currency	total de knowledge possédé par le player	INTEGER
idle_value	valeur d'auto-incrémantation	BIGINT
click_value	valeur du clic	BIGINT
click_counter	nombre de clic total depuis début de partie	INTEGER
prestige_level	multiplicateur qui s'applique à idleValue et clickValue en cas de new game +	INTEGER
logout_time	date et heure de déconnexion du player	TIMESTAMPTZ
login_time	date et heure de reconnexion du player	TIMESTAMPTZ



Table generator

Table generator

Nom de la colonne	Description	Type
id	PK, identifiant unique du generator	INTEGER
type	1 si generator clic flat 2 si generator clic percent 3 si generator idle flat 4 si generator clic percent	INTEGER
name	nom du generator	TEXT
text	text description du generator	TEXT
clic_flat_value	valeur du bonus de base appliquée au bonus total du clic flat	INTEGER
clic_percent_value	valeur du bonus de base appliquée au bonus total du clic en pourcentage	INTEGER
idle_flat_value	valeur du bonus de base appliquée au bonus total du idle flat	INTEGER
idle_percent_value	valeur du bonus de base appliquée au bonus total du idle en pourcentage	INTEGER
starting_cost	coût du premier achat du generator	INTEGER
cost_factor	facteur de multiplication à appliquer pour calculer le coût du prochain generator	INTEGER
order	nombre qui définit l'ordre d'apparition des generator	INTEGER



Table sentence

Table Sentences

Nom de la colonne	Description	Type
id	PK, identifiant unique du generator	INTEGER
title	titre du message	TEXT
text	message à l'attention du joueur	TEXT

Table d'association

Table d'association OWN

Nom de la colonne	Description	Type
player_id	id du player	INTEGER
generator_id	id du generator	INTEGER
number_owned	nombre de générateurs possédés	INTEGER
next_cost	Prochain coût	BIGINT
multiplier_bonus	Bonus multiplicateur.	INTEGER



Trello

Overview

The screenshot shows a Trello board titled "Bookworm Idle". The board is organized into several lists:

- À faire:**
 - Changer la base de données pour passer la currency en text.
 - JSDOC complète sur json de save
 - Ne pas oublier de remettre les regexp
 - Rajouter et corriger des sentences
- En cours:**
 - Liste bugs à régler (9/10)
 - Mise au point Cahier des charges.
 - Préparer la présentation
- Terminé:**
 - pastille new msg bookworm
 - Front: S'occuper de la gestion des onglets sur le shop (en théorie)
 - vider le quickshop à la deco (voire ne pas l'afficher du tout si le user n'est pas co ?)
 - icones items
 - modal pour dire que la save a bien fonctionné
 - cost sur les modals de détails d'item dans les stats
- Validé:**
 - + Ajouter une carte
- Archive Sprint 2:**
 - Quick Shop
 - Regarder l'ordre des generators owned qui vient du back
 - mobile > détail item stats
 - mobile > modale stats
 - mobile > quick shop
 - desktop > shop
 - desktop > header global position
 - desktop > infos
 - confirm delete account css
- archives Sprint 1:**
 - TEST BACK (20/20)
 - bug de Phil
 - Calcul entre le logout et le login avec répercussion sur la currency.
 - corriger l'orthographe de disconnect.
 - Route Disconnect
 - Création de la modale shop mobile (lire description + HARDCORE!)
 - Création de compte
- archives Sprint 0:**
 - Finir le cahier des charges
 - Routes back tableau
 - Avancer CDC Front
 - Compléter CDC Back
 - Faire le MCD / MLD
 - Faire le Dico des données
 - DDL : script de création des tab
 - DML : script de seeding
 - Modifier wireframe desktop

A sidebar on the right shows a tree view of other boards:

- archives Sprint 0
 - Finir le cahier des charges
 - Routes back tableau
 - Avancer CDC Front
 - Compléter CDC Back
 - Faire le MCD / MLD
 - Faire le Dico des données
 - DDL : script de création des tab
 - DML : script de seeding
 - Modifier wireframe desktop



Card

Finir le cahier des charges
Dans la liste [archives Sprint 0](#)

Membres Étiquettes Ajouter à la carte

AP B PC TB YF + Conception Réécriture + Membres

Description Ajouter une description plus détaillée...

to do Masquer les tâches cochées Supprimer

100%

- user stories compléter les messages lecture
- wireframe compléter message lecture
- spec fonctionnelles produit final, préciser
- user stories produit final
- Définition de la cible rédiger propre
- Navigateurs prendre décision
- breakpoints prendre décision

Ajouter un élément

Activité Afficher les détails

AP Écrivez un commentaire...

Ajoutez des menus déroulants, des champs de texte, des dates et plus encore à vos cartes.

[Commencer l'essai gratuit](#)

Power-Ups + Ajouter des Power...

Automatisation + Ajouter un bouton

Actions → Déplacer

Copier

Créer un modèle

Suivre



Requêtes

getOneUser



```
1  async getOneUserJson(id) {
2      debug(`getOneUserJson called for id ${id}`);
3      const query = {
4          text: `select row_to_json(player.*) as player from player where id=$1;`,
5          values: [id],
6      };
7      const player = (await client.query(query)).rows[0];
8      return player;
9  },
```

getGeneratorsOwned



```
1  async getGeneratorsOwned(id) {
2      debug(`getGeneratorsOwned called for id ${id}`);
3      const query = {
4          text: `SELECT json_agg(generator.* ORDER BY generator."order") as generators
5              FROM player_owns_generator
6              JOIN player ON player_owns_generator.player_id=player.id
7              JOIN generator ON player_owns_generator.generator_id=generator.id
8              WHERE player_owns_generator.player_id=$1
9              GROUP BY player;`,
10         values: [id],
11     };
12     const generatorsOwned = (await client.query(query)).rows[0];
13     if (!generatorsOwned) {
14         debug('return empty array');
15         const generatorsOwnedZero = {
16             generators: [],
17         };
18         return generatorsOwnedZero;
19     }
20     return generatorsOwned;
21 },
```



getPlayerNotOwnsGenerator



```
1  async getPlayerNotOwnsGenerator(id) {
2      debug(`getPlayerNotOwnsGenerator called for id ${id}`);
3      const query = {
4          text: `SELECT json_agg(generator.* ORDER BY generator."order" ) as generators
5              FROM generator WHERE generator.id NOT IN (
6                  SELECT player_owns_generator.generator_id FROM player_owns_generator
7                      WHERE player_owns_generator.player_id=$1)`,
8          values: [id],
9      };
10     const playerNotOwnsGenerator = (await client.query(query)).rows[0];
11     return playerNotOwnsGenerator;
12 },
```

get playerOwnsGenerator



```
1  async getPlayerOwnsGenerator(id) {
2      debug(`getPlayerOwnsGenerator called for id ${id}`);
3      const query = {
4          text: `SELECT json_agg(player_owns_generator.*) as playerOwnsGenerator
5              FROM player_owns_generator WHERE player_id=$1;`,
6          values: [id],
7      };
8      const playerOwnsGenerator = (await client.query(query)).rows[0];
9      return playerOwnsGenerator;
10 },
```



MERCI !

