Story Builder

Un constructeur d'*aventure dont vous êtes le héros*

Projet réalisé par

Jérémie LEYMARIE

Guillaume Thiebaud

Nabi Aries

Yanis Gherdane

Rihab Ammar

Introduction

Résumé

Story-Builder est une plateforme de création de jeu type *Histoire dont vous êtes le héros.* Le projet se décompose en deux parties : d'un côté la partie constructeur d'histoire, le *builder* et de l'autre le jeu en lui-même. L'idée générale est de permettre à la fois l'élaboration intégrale d'une histoire, la publication de celle-ci sur un store interne donnant accès aux joueurs à une bibliothèque communautaire.

Objectifs Techniques

Ce projet repose également sur deux ambitions techniques. D'une part, proposer un produit cross-platform : Mobile (IOS/Android), Web & Desktop, par le biais d'une *Progressive Web App (PWA).* D'autre part, le projet est pensé *offline-first*, c'est-à-dire que la connexion Internet ne doit être nécessaire que dans de rares cas (téléchargement d'histoires depuis le store, synchronisation entre appareils). L'essentiel de l'application doit pouvoir fonctionner entièrement en local.

Architecture technique du projet

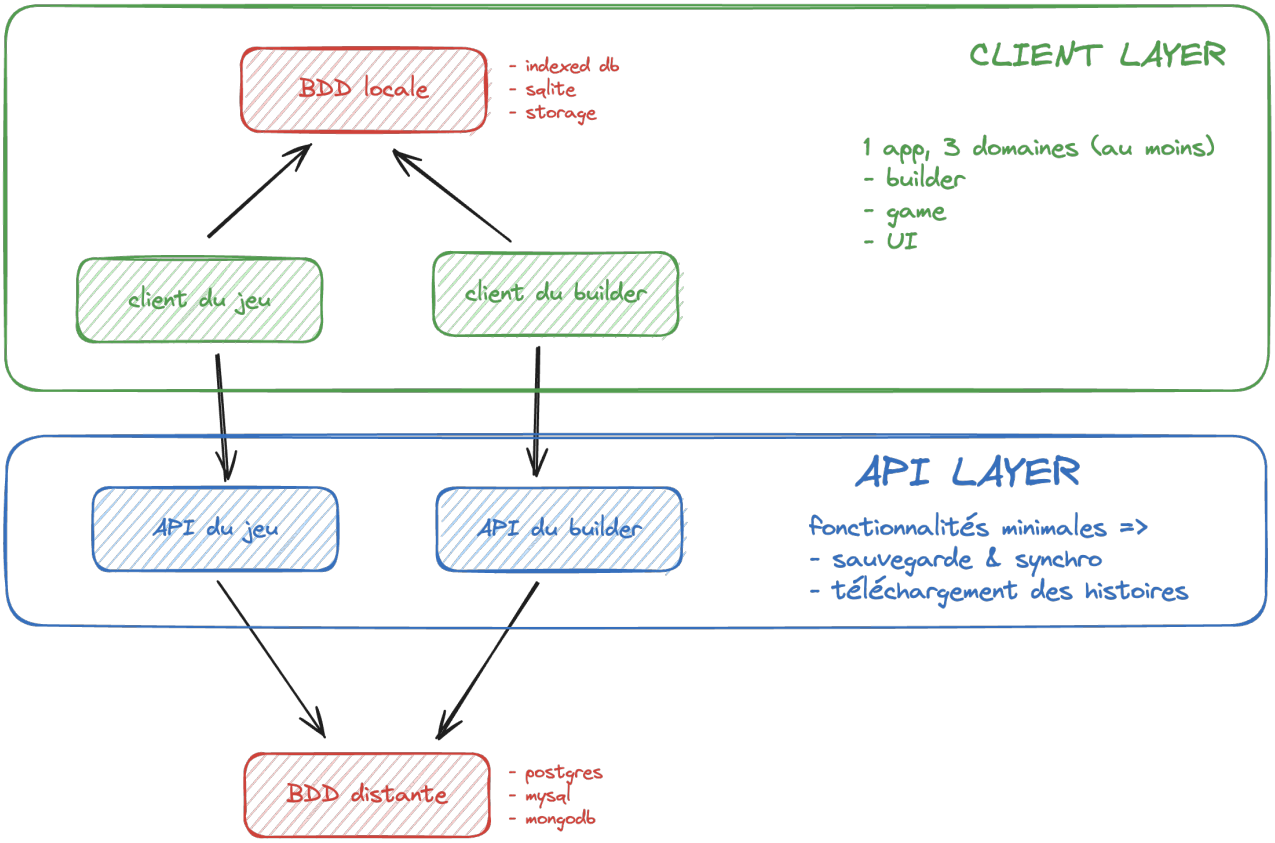

Fig 1. Schéma d'ensemble de Story Builder

Stack technique

Frontend

Le frontend est une application *React*, orchestrée par *Vite* et écrite en *Typescript*. La partie PWA est gérée par le plugin *vite-pwa.*
Quelques outils sont structurels pour le frontend :

- *Tanstack Router* gère le routing dans le frontend
- *TailwindCSS* est le moteur de style
- *ShadCN UI* est la librairie de composants graphiques qui forme la base de notre design-system
- *React Flow* permet de représenter des données sous forme de noeuds et de liens

Backend

Le backend est écrit en *Python*, en utilisant le framework web *Fast API.* La validation des requêtes et des réponses se fait via *Pydantic.* Le backend est typé via le mécanisme de type hints en python et la vérification statique des types se fait par *mypy.*

Bases de Données

Le project repose sur une base de donnée locale dans le navigateur, *IndexedDB.* On utilise un wrapper autour de l'API native, la libraire *Dexie.*

La base de donnée à distance est une base de donnée *MongoDB*, et les interactions avec elle se font par le driver python *pymongo.*

Liste de fonctionnalités

Builder

Page d'accueil

Tags : Front
Description :
Doit servir de Hub central à l'application. Donne accès à : builder, bibliothèque, parties en cours, partie actuelle.
Charge en J/H : ***0.5***
Etat : Fini

Initialiser la gestion du stockage local

Tags : Front
Description :
- Initialiser l'instance Dexie
- Créer les modèles de données (v1)
- Exposer un repository pour s'interfacer avec la base de données
- Ecrire un premier exemple d'utilisation

Charge en J/H : ***0.5***
Etat : Fini

Initialiser la gestion du stockage à distance

Tags : Back, Devops
Description :
- Dockerizer MongoDB pour faciliter l'utilisation à toute la team
- Initialiser le client Mongo dans l'app FastAPI
- Ecrire un premier exemple d'utilisation
- Avoir des petits helpers réutilisables liés à MongoDB (formatage des ids)

Charge en J/H : ***0.5***
Etat : Fini

Authentification (Back)

Tags : Back
Description :
- Route de connexion
- Route d'inscription
- Création/Vérification de token JWT
- Hashing de mot de passe
- Validation via pydantic

Charge en J/H : ***0.5***

Etat : Fini

Page de connexion (Front)

Tags : Front
Description :
- Création de la page */signin* et */signup*
- Formulairesavec validation utilisant *zod, react-hook-form* et le *design-system*
- Interaction avec l'API pour valider les actions
- Redirection vers la page d'accueil en cas de succès, affichage d'un message d'erreur sinon

Charge en J/H : ***0.5***
Etat : Fini

Page d'inscription (Front)

Tags : Front
Description :
- Création de la page */signup*
- Formulaire avec validation utilisant *zod, react-hook-form* et le *design-system*
- Interaction avec l'API pour valider les actions
- Redirection vers la page d'accueil en cas de succès, affichage d'un message d'erreur sinon

Charge en J/H : ***0.5***
Etat : Fini

Page de liste des histoires (Builder)

Tags : Front
Description : La page doit afficher la liste de toutes les histoires crées par un utilisateur.
- Récupère la liste des histoires depuis la base de données locale
- Affiche des cartes représentant les histoires d'un utilisateur
- Au clic, amène sur l'interface du Builder correspondant à l'histoire

Charge en J/H : ***0.5***
Etat : Fini

Création d'une nouvelle histoire

Tags : Front
Description :
- Dans la liste des histoires, la première carte de la liste propose de créer une histoire
- Formulaire dans une modale, avec validation et messages d'erreur
- Si la création est fructueuse, l'histoire est stockée dans la base de données locale
- Après validation de l'action, redirection vers le builder dans l'histoire nouvellement créée.

Charge en J/H : ***0.5***
Etat : Fini

Affichage des scènes dans l'espace de création

Tags : Front
Description :
- Dans le builder, les scènes enregistrées doivent être affichées
- Leurs données sont chargées depuis la base de données locale
- Leur position est enregistrée sauvegardée
- On peut interagir avec les noeuds, les relier, les déplacer

Charge en J/H : ***0.5***
Etat : Fini

Création/Modification d'une scène dans une histoire

Tags : Front
Description :
- Dans l'interface du builder, une toolbar à gauche permet de créer une scène
- Une modale s'ouvre, avec un formulaire validé par zod, permettant de créer une scène
- Dans sa v1, permet d'ajouter un titre et un contenu
- Au succès, la modale se ferme et le nouveau contenu est enregistré dans la base de données locale
- L'interface est mise à jour avec les nouvelles informations
- Quand une scène est déplacée dans l'interface, sa nouvelle position est enregistrée dans la base de données locale

Charge en J/H : ***0.5***
Etat : Fini

Espace *Whiteboard/Croquis*

Tags : Front
Description :
- Une page (v1, peut-être intégrée dans l'interface du Builder plus tard) doit donner à un espace whiteboard
- Sur cet espace, on a accès à l'outil *Excalidraw* qui permet de dessiner, faire des croquis à main levée et des schémas

Charge en J/H : ***0.5***
Etat : Fini

Ajouter/Modifier/Supprimer des boutons dans une scène

Tags : Front
Description :

- Le formulaire d'édition et de création d'une scène doit permettre d'ajouter/retirer/modifier des boutons
- Les boutons doivent s'afficher dans le builder
- Ils doivent être enregistrés dans la base de données locale

Charge en J/H : ***0.5***
Etat : Fini

Lier des boutons à des scènes

Tags : Front
Description :
- Dans le builder, il doit être possible de relier des boutons à une scène
- Un bouton ne peut pas se lier à sa propre scène
- Les liens doivent être enregistrés automatiquement dans la base de données locale

Charge en J/H : ***0.5***
Etat : Fini

Noeud de départ d'une histoire

Tags : Front
Description :
- Lors de la création d'une histoire, une noeud de départ doit être créé dans la base de données locale
- Le Builder doit afficher ce noeud de manière distinctive
- Il doit être possible de lier de noeud à une scène, mais rien ne peut être lié à ce noeud

Charge en J/H : ***0.5***
Etat : Fini

Bouton de synchronisation à distance

Tags : Back, Front
Description :
- Dans le builder, un bouton doit permettre de synchroniser l'histoire dans la base de données distante (ça demande de refactoriser la navbar)
- Une route API doit permettre de synchroniser l'histoire entre l'état de la base de données locale et la base de données distante

Charge en J/H : ***0.5***
Etat : Fini

Test de l'histoire dans le builder

Tags : Front
Description :
- Un.e créateur.rice d'histoire doit pouvoir tester son histoire avant de la publier
- Pour cela, un bouton doit apparaître dans la toolbar du builder

- Redirige vers le jeu, à la page initiale de l'histoire
- Cela doit créer un document StoryProgress dans la base de données locale, mais flaggé comme test

Charge en J/H : ***0.5***
Etat : Fini

Publication d'une histoire

Tags : Back, Front
Description :
- Dans le builder, un bouton doit permettre de publier l'histoire => passage de statut *draft* à status *published*
- La modification s'effectue d'abord dans la base de données distante via API, puis dans la base de données locale

Charge en J/H : ***1***
Etat : Fini

Sauvegarder plus d'informations sur les histoires

Tags : Front
Description :
- Ajouter un genre aux histoires (penser aux types de la base de données locale)
- Le genre doit pouvoir être renseigné dans le formulaire de création d'histoire
- On doit aussi pouvoir stocker la date de publication d'une histoire

Charge en J/H : ***0.5***
Etat : Fini

Supprimer des scènes

Tags : Front
Description :
- On doit pouvoir sélectionner une scène en cliquant
- On doit voir quel élément est sélectionné (bordure ?)
- En appuyant sur la touche de suppression, l'élément est supprimé

Charge en J/H : ***0.5***
Etat : Pas commencé

Export JSON d'une histoire

Tags : Front
Description :
- Dans le builder, on doit pouvoir exporter une histoire.
- Le bouton export ouvre une modale, qui montre le contenu du JSON (avec un scroll)
- Un bouton permet de copier dans le presse papier
- Un bouton permet de télécharger le fichier

- Un bouton permet d'annuler et ferme la modale

Charge en J/H : ***0.5***

Etat : Fini

Import JSON d'une histoire

Tags : Front

Description :

- Dans la page des histoires du builder, on doit pouvoir importer une histoire.
- La carte qui permet de créer une histoire à un autre bouton, qui propose d'importer une histoire
- Cela ouvre une modale qui propose une text Area qui permet de copier du JSON.
- Un bouton permet d'importer
- Un autre d'annuler

Charge en J/H : ***0.5***

Etat : Fini

Game

Page du store d'histoires

Tags : Front

Description :

- Une page, accessible depuis la page d'accueil donne accès à un store d'histoires
- Sur cette page, on affiche une liste de cartes, représentant chacune une histoire

Etat : Fini

Charge en J/H : ***0.5***

Route API pour le store

Tags : Back, Front

Description :

- Une route API permet de récupérer toutes les histories, dont le statut est *published*
- Seulement les informations essentielles doivent être récupérées (pas besoin des scènes)

Charge en J/H : ***0.5***

Etat : Fini

Télécharger une histoire (BACK)

Tags : Back

Description :

L'application fonctionnant essentiellement hors-ligne, le téléchargement d'une histoire est un pré-requis pour pouvoir jouer une partie.
Une route API doit permettre d'envoyer tout le contenu d'une histoire.
Points exploratoires à débroussailler :
- Le téléchargement peut être long, comment Fast API gère ce type de problématique (voir HTTP Polling)

Charge en J/H : ***0.5***
Etat : Fini

Télécharger une histoire (FRONT)

Tags : Front
Description :
L'application fonctionnant essentiellement hors-ligne, le téléchargement d'une histoire est un pré-requis pour pouvoir jouer une partie.
- Au clic sur une carte dans le store, une modale propose de lancer le téléchargement de l'histoire ou d'annuler.
- Le clic sur *Cancel* referme la modale
- Le clic sur *Download* lance le téléchargement via l'API
- La UI doit refléter la progression du téléchargement de l'histoire
- Une fois les données de l'histoire reçues, elles doivent être enregistrées dans la base de donnée locale.
- Après le téléchargement, l'interface propose de lancer la partie ou de revenir sur le store.
- Point de vigilance sur la lenteur du call API dans le cas d'une longue histoire => gérer le cas (HTTP polling, batching, à explorer)

Charge en J/H : ***0.5***
Etat : Fini

Page *Parties en cours*

Tags : Front
Description : La page doit afficher la liste de toutes les histoires téléchargées par un utilisateur.
- Récupère la liste des histoires depuis la base de données locale
- Affiche des cartes représentant les histoires téléchargées d'un utilisateur
- Au clic, redirige vers la page de détail d'une partie
- La page doit être pensée d'abord pour mobile, mais doit être 100% responsive

Charge en J/H : ***0.5***
Etat : Fini

~~Page *Parties en cours - Téléchargements*~~

~~Tags : Front~~
~~Description : La page doit également afficher les histoires commencées par un utilisateur, mais non disponible sur l'appareil en cours d'utilisation.~~

- ~~Pour toutes les histoires commencées, mais non téléchargées, l'utilisateur.rice doit pouvoir les télécharger sans perdre sa progression~~

~~*Charge en J/H :* ***0.5***~~
~~***Etat : OUT***~~

Page de détail d'une partie

Tags : Front
Description : La page doit détailler les informations d'une partie.
- On peut y lire des informations sur l'histoire : titre, résumé
- On affiche l'image de couverture de l'histoire (background)
- La page doit être pensée d'abord pour mobile, mais doit être 100% responsive
- On y trouve également l'endroit où le joueur s'est arrêté (au moins le titre de la scène)
- On peut imaginer d'autres infos (TDB), comme le pourcentage de progression dans l'histoire, les stats du personnage au moment où iel s'est arrêté.e, etc…
- Un bouton "Jouer" permet de rediriger vers la page de jeu, à l'endroit où le.a joueur.euse s'est arrêté.e
- Charge les informations depuis la base de données locale

Charge en J/H : ***0.5***
Etat : Fini

Page de jeu

Tags : Front
Description :
- C'est l'élément central du jeu : la page que le.a joueur.euse lit et avec laquelle iel interagit pour se déplacer dans l'histoire
- Elle affiche le titre de la scène, le contenu de la scène
- Si la scène a une image, elle est présente
- Affiche les boutons pour se déplacer vers une autre scène
- Au clic sur un bouton, on redirige vers la scène correspondante, et on met à jour l'état actuel de la partie dans la base de données locale
- La page doit être pensée d'abord pour mobile, mais doit être 100% responsive
- Charge les informations depuis la base de données locale

Charge en J/H : ***0.5***
Etat : Fini

Pouvoir finir une histoire

Tags : Front
Description :
- Le jeu doit être capable de gérer une fin d'histoire, c'est-à-dire une scène qui ne contient aucune action
- La page de jeu doit afficher le texte de fin et proposer de revenir à l'écran principal.
- On enregistre dans la base de données locale que le.a joueur.euse a fini l'histoire.

Charge en J/H : ***0.5***
Etat : Fini

Enregistrement de la progression du jeu pendant la partie

Tags : Front
Description :
- Chaque déplacement dans l'histoire doit être enregistré dans la base de données locale (StoryProgresses)
- L'implémentation doit être stable et sûre, pour que la progression de l'utilisateur.rice soit toujours à jour.

Charge en J/H : ***0.5***
Etat : Fini

Synchronisation des parties entre différents appareils

Tags : Front, Back
Description :
- Quand un.e utilisateur.rice ouvre l'app sur un deuxième appareil, ses parties précédentes doivent être chargées, ainsi que ses histoires dans le builder.
- Si l'appareil est hors connexion, l'application doit fonctionner normalement et afficher un message (permanent, toast ?)
- Point de vigilance très important => la gestion des ids des scènes pour enregistrer la progression dans l'histoire. Les ids originaux proviennent de la base de données locale du premier appareil, et ils ne correspondront pas nécessairement aux ids dans la base de données locale des autres appareils. Il faut donc trouver une manière fiable de retrouver des scènes entre les différentes bases de données.
- Point à déterminer => à quelle fréquence est enregistrée la progression dans la base de données remote

Charge en J/H : ***1***
Etat : En cours

Gérer la dernière histoire jouée

Tags : Front
Description :
- L'info doit être stockée dans StoryProgress
- Elle doit être updatée à chaque sauvegarde du jeu
- Une méthode du repository doit permettre de récupérer la dernière partie jouée
- Cette histoire est affichée sur la page d'accueil

Charge en J/H : ***0.5***
Etat : Fini

Améliorer les pages de listes (store, builder-stories et library)

Tags : Front
Description :
- Ces trois pages sont très dénudées et peu reconnaissables entre elles

- Il faudrait au moins pouvoir avoir un titre permettant de savoir sur quelle page on se trouve
- Découper les listes en plusieurs catégories, en séparant les histoires par genre, ou par date de parution par exemple
- Faire en sorte que ces pages paraissent moins vides

Charge en J/H : ***0.5***
Etat : Pas commencé

Technique

Typecheck du code python

Tags : Back, DevOps
Description :
- Ajouter mypy (static type checker) au projet
- Créer un script bash qui type check l'intégralité de la codebase
- Ajouter un hook de pre-commit qui vérifie le typage
- Ajouter une CI qui vérifie le typage (BONUS)

Charge en J/H : ***0.5***
Etat : Fini

Ajouter des règles de linting/formatage proches des défauts de l'industrie

Tags : Front, DevOps
Description :
- Ajouter plus de règles eslint (imports, unused-vars, etc…)
- Forcer l'application des règles dans un hook de pre-commit
- Ajouter le check en CI

Charge en J/H : ***0.5***
Etat : Fini

Etre capable de détecter dynamiquement le statut du réseau

Tags : Front
Description :
- Le front doit être capable de gérer les deux états : en ligne / hors ligne
- Un hook react doit pouvoir nous donner cette information dynamiquement
- Un exemple de composant qui change d'affichage en fonction de l'état du réseau (Store)

Charge en J/H : ***0.5***
Etat : Fini

~~Déploiement de l'application~~

~~Tags : DevOps~~
~~Description :~~
- ~~Dans le but de tester les fonctionnalités de compatibilité/synchronisation entre différents appareils, il faudra mettre en ligne l'application.~~
- ~~Solutions techniques plébiscitées => Vercel + MongoDB Atlas + *[solution cloud SQL gratuite]*~~
- ~~Il faudra voir si le déploiement via le repository epitech est possible, sinon il faudra mettre en place un repo perso pour gérer le déploiement~~

~~*Charge en J/H : **1***~~
~~***Etat : OUT***~~

Mettre en place l'infrastructure pour les tests unitaires

Tags : DevOps, Back
Description :
- Set up pytest (ou autre, si plus pertinent) dans les deux APIs et écrire des tests unitaires pour servir d'exemple.
- Ajouter les tests pythons dans la CI
- Set up vitest dans le client
- Ajouter les tests typescript dans la CI

*Charge en J/H : **0.5***
Etat : Fini

Utiliser poetry pour les APIs

Tags : DevOps, Back
Description :
- Venv & pipreqs sont un peu léger pour un projet qui a vocation à durer, il faut mettre en place quelque chose de plus robuste
- Les remplacer par poetry, pour la gestion des modules et de l'environnement

*Charge en J/H : **0.5***
Etat : Pas commencé

Utiliser des clés primaires comme source d'identification universelle

Tags : Front, Back
Description :
- Devoir gérer des ids différents dans les différentes base de données est un vrai challenge et pose des problèmes de clarté de code, nous expose à des risques de bugs

- La solution proposée est d'utiliser des clés, sous la forme d'uuid, qui sont utilisées comme clé principale dans toutes nos bases de données opérationnelles (et analytiques ?)
- La base de donnée locale doit être responsable de créer ces clés
- Elles doivent être systématiquement créées lors de la création d'une ressource dans la base de données locale
- Il faut mettre à jour toute la codebase pour retirer les occurrences d'id ou remoteId et les remplacer par l'utilisation de la nouvelle clé primaire

Charge en J/H : ***1***
Etat : Fin

DATA

Réaliser une étude sur les outils DATA à intégrer

Tags : Data
Description :
- Il faut étudier les différents outils qui constitueront la Stack DATA
- Un point essentiel est de gérer la compatibilité avec Mongo, qui est la base principale du projet.
- Plusieurs solutions à étudier : la possibilité d'avoir une DB SQL en plus, utiliser un ETL pour transformer la donnée vers du relationnel, ou pouvoir utiliser directement MongoDB dans les requêtes
- Etudier les différentes solutions de data-warehouse

Charge en J/H : ***2***
Etat : Fini

~~[OUT] Créer l'API DATA pour la gestion des données d'utilisation~~

~~Tags : Data, Back~~
~~Description :~~
- ~~Le traitement et l'analyse de la data doit être découplé des données opérationnelles et le stockage de ces données également.~~
- ~~Choisir une techno pertinente pour de la DATA (python ?)~~
- ~~Ajouter de la documentation pour expliquer comment lancer le serveur~~
- ~~L'API doit avoir une première version de l'arborescence des fichiers et une route servant d'exemple~~
- ~~La codebase doit être localisée au niveau de server/src, mais il faut la séparer de l'API de l'app~~
- ~~Il peut-être pertinent de mettre en commun les domaines et les repositories, de manière ça ce qu'ils soient réutilisables pour l'API DATA~~

~~*Charge en J/H :* ***0.5***~~
~~***Etat : OUT***~~

~~[OUT] Mettre en place une base de données SQL (à la place d'un outil BI)~~

~~Tags : Data, Back, DevOps~~
~~Description :~~
- ~~Pour stocker les données analytiques à dans un espace de stockage dédié, et fait pour l'analyse de données~~
- ~~Choisir la DB SQL idéale (postgres ?)~~
- ~~La base de données doit être dockerisée et documentée dans le README~~
- ~~Dans l'API DATA, il faut ajouter une connexion à la DB, et au moins un exemple de requête. Il faut notamment se poser la question de la gestion d'un ORM~~

~~Charge en J/H : **0.5**~~
~~Etat : OUT~~

~~[OUT] Intégration d'un outil de BI (à la place du DB SQL)~~

~~Tags : Data, Back, DevOps~~
~~Description:~~
- ~~L'api DATA doit avoir un endpoint d'extraction de données~~
- ~~Charge en J/H : **0.5**~~

~~Etat : OUT~~

Extraire les données Mongo vers du CSV

Tags : Data
Contexte : Pour pouvoir exploiter les données de l'app sur PowerBI, le format nécessaire est CSV.
Description :
- Créer un script bash d'export de la base de données Mongo vers un format CSV

*Charge en J/H : **0.5***
Etat : Fini

Générer des données pour l'analyse en local

Tags : Data
Contexte : Pour produire des Dashboard sur PowerBI il faut de la donnée en local.
Description :
- Créer via une utilisation de l'app une quantité suffisante de données pour pouvoir réaliser un dashboard
- Plusieurs utilisateurs et histoires, publications d'histoires dans le store, progressions dans des parties.

*Charge en J/H : **0.5***
Etat : En cours

Mettre en place un Dashboard DATA

Tags : Data, DevOps
Description :

- Choisir un outil gratuit pour afficher des reportings DATA (A déterminer => outil SaaS, outil low-code, lib de charts en python…)
- Le mettre en place :
  - Créer des accès pour l'équipe
  - Documenter son utilisation dans un README
  - Le configurer
  - Tenter de connecter des premières sources de données
- Créer des premiers graphiques

Charge en J/H : ***1***

Etat : Pas commencé

BONUS

A détailler

Rendre la toolbar rétrécissable
Créer un personnage dans l'histoire (BUILDER & GAME)
Avec des inputs utilisateurs (nom, etc…)
Ajouter des statistiques de personnages (BUILDER & GAME)
Par ex => PV, Intelligence, Charisme, etc…
Utiliser des variables dans le texte de l'histoire (BUILDER & GAME)
Gestion de la musique (BUILDER & GAME)
Connexion par SSO
Upload de fichiers
Gestion d'inventaire
Combat
Conditionner l'histoire en fonction de choix passées
Pouvoir personnaliser les couleurs
Pouvoir personnaliser les boutons
Pouvoir personnaliser la police
Ajouter un wiki à une histoire
Intégrer une documentation utilisateur in-app
Créer une histoire à partir d'une histoire existante
Page détaillée des histoires dans le store
Réutiliser les hooks pre-commit dans la CI
Pouvoir publier plusieurs versions d'une histoire
Migrer vers reactflow 12
Ajouter des tests unitaires dans l'API
Page de profil
Développer des outils internes pour faciliter la manipulation d'indexed-db en dev
Sécuriser les APIs
Etudier la question des images en mode hors-ligne (stockage, récupération, export JSON)
Retravailler l'UI du formulaire d'édition des scènes
Ajouter une command palette (BUILDER)

TÂCHES RÉALISÉES LORS DU SPRINT #2

25/06/24 - 08/07/2025 (4 jours travaillés)

- Noeud de départ d'une histoire
- Lier des boutons à des scènes
- Bouton de synchronisation à distance
- Création de compte/Connexion lors de la synchronisation du Builder
- Etre capable de détecter dynamiquement le statut du réseau
- Test de l'histoire dans le builder
- Page de jeu
- Page du store d'histoires
- Route API pour le store
- Réaliser une étude sur les outils DATA à intégrer
- Typecheck du code python (local + pre-commit + CI) + ESLINT
- Etude DATA en cours

TÂCHES ENGAGEES POUR LE SPRINT #3

- Avancer sur les pages de connexion/inscription
- Publication d'une histoire
- Sauvegarder plus d'informations sur les histoires
- Export JSON d'une histoire
- Télécharger une histoire (BACK)
- Télécharger une histoire (FRONT)
- Page *Parties en cours*
- Page *Parties en cours - Téléchargements*
- Pouvoir finir une histoire
- Enregistrement de la progression du jeu pendant la partie
- Réaliser une étude sur les outils DATA à intégrer
- [OUT] Créer l'API DATA pour la gestion des données d'utilisation
- [OUT] Mettre en place une base de données SQL *OU* Intégration d'un outil de BI

REALISEES HORS SPRINT =>
- Finition du téléchargement des histoires (front/back)
- Import JSON
- Gestion des clés uniques
- Gérer la dernière histoire jouée

TÂCHES ENGAGEES POUR LE SPRINT #4

DATA
- Extraire les données Mongo vers du CSV
- Générer des données pour l'analyse en local
- Mettre en place un Dashboard DATA (premiers graphs, etc…)

GAME
- Synchronisation entre appareils
- [OUT] Page *Parties en cours - Téléchargements*
- Pouvoir finir une histoire

- Améliorer les pages de listes (store, builder-stories et library)
- Création d’un bug-log, et début du dépilage
    - Gestion des erreurs API
    - Publier une histoire déjà synchro ne fonctionnait pas
    - Retirer le fichier généré par le routeur de la configuration Prettier

BUILDER
- Import/Export JSON

TECHNIQUE
- Mettre en place l’infrastructure pour les tests unitaires

TÂCHES ENGAGEES POUR LE SPRINT #5

BUILDER
- Supprimer des scènes
- Améliorer les pages de listes
- Commencer à étudier les possibilités de actions conditionnelles dans le BUILDER

TECHNIQUE
- Utiliser poetry pour l’API

BUGS
- Vider les inputs du formulaire d'ajout de scène après le submit
- Ne pas enregistrer de story progress en mode test
- StoryProgress.history est rempli à chaque refresh d'une page de jeu
- Description overflow dans story-card
- Afficher les liens au desus des cartes
- Retirer l'autheur dans l'import JSON d'une histoire
- Chasser les bugs restants