Rapport de projet

PatteD'Oie

Compte-rendu de projet

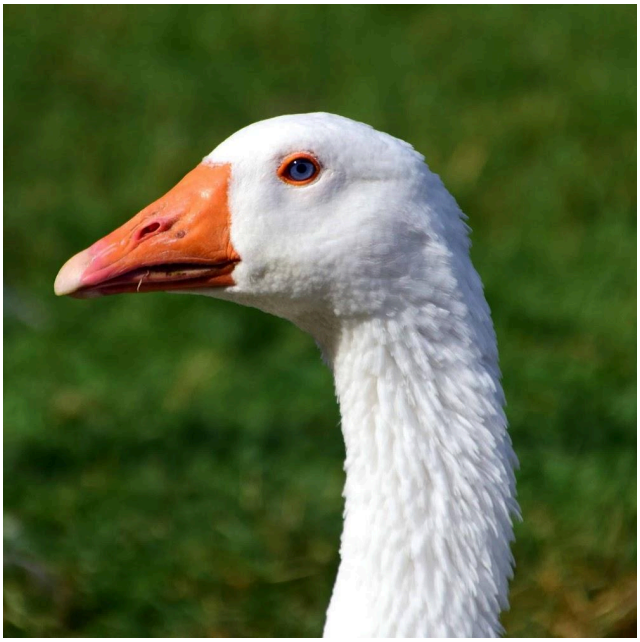
Insérez votre texte ici

OBJECTIF DU DOCUMENT

Ce document a pour objectif de présenter les fonctionnalités réalisées sur la plateforme de jeux PatteD'Oie ainsi que les difficultés rencontrées et la méthodologie de travail utilisée pour ce projet.

Chaque membre de l'équipe expliquera ainsi sa vision sur le projet.

> Seul on va plus vite, ensemble on va plus loin



PLAN DU DOCUMENT

- - - - X

- Présentation des fonctionnalités réalisées
- Explication des difficultés rencontrées
- Déroulement de la gestion de projet et la répartition des tâches
- Bilan sur le projet

PRÉSENTATION DES FONCTIONNALITÉS RÉALISÉES

- - - - X

Pour avoir accès à l'ensemble de notre projet en ligne, vous pouvez cliquer sur ce lien.

Il est tout d'abord à noter que nous avons choisi les fonctionnalités avec un poids de 12 (2 non-alternants, 3 alternants).
Nous avons donc choisi d'implanter les fonctionnalités suivantes :

- Mise en place des meilleurs scores pour les jeux (2 poids)
- Réalisation du Speed Typing (3 poids)
- Réalisation du Petit Bac (4 poids)
- Mise en place d'un mot de passe pour les parties privées (3 poids)
- Application responsive (2 poids)

Ce qui nous donne un total de 14 poids.

Passons maintenant en revue ce qu'il est possible de faire sur l'application:

Lorsque l'on arrive sur la plateforme, la première action demandée est de saisir un pseudo, sans quoi il sera impossible de jouer (même si on navigue sur les fenêtres du site, tenter de créer ou rejoindre une partie nous ramène à la page d'authentification).

Il est ensuite possible de lister l'ensemble des lobbies disponibles et de les filtrer selon le type de jeu (Speed Typing ou Petit bac dans notre cas), ou selon la visibilité des parties (publiques, privées).

Si on souhaite rejoindre une partie privée, un mot de passe sera demandé, si on souhaite rejoindre une partie publique, cliquer sur un bouton Join the lobby suffit.

Il est également possible de consulter le détail d'une partie. Seul le propriétaire du lobby peut lancer la partie.

Tout utilisateur peut créer une partie de Speed Typing ou Petit bac. A la création, il est proposé à l'utilisateur de créer une partie publique (auquel cas seul un nom de lobby sera demandé) ou une partie privée (un mot de passe non nul sera alors demandé en plus du nom de lobby).

Une fois dans une partie de Speed Typing, le chrono se déclenche immédiatement : les joueurs ont alors 1min maximum pour écrire 10 mots assez complexes. Ce jeu se joue alors au temps (la personne qui a le meilleur temps gagne).

Le mot courant est affiché en grand et gras pour mettre en évidence le mot à écrire. Le mot tapé est indiqué en rouge lorsqu'il n'est pas correct, ce qui indique en temps réel à l'utilisateur que son mot contient une erreur. A chaque saisie d'espace le mot est envoyé et analysé, s'il est correct le score est incrémenté. La barre de progression de chacun est également actualisée en temps réel.

Lorsque tous les joueurs ont fini de saisir leurs mots, ou que le timer est à 0, le classement est affiché avec le nombre de mots par minute calculé. Au bout d'une minute après cet affichage, l'utilisateur est automatiquement ramené à la page d'accueil et la partie est considérée comme terminée.

Du côté du Petit Bac, une fois sur la partie le joueur voit s'afficher la liste des catégories avec une lettre. Sous chaque catégorie, il dispose d'une zone de texte dans laquelle il peut saisir son mot. Pour chaque mot, s'il est indiqué en rouge il s'agit alors d'un mot erroné (ne commençant pas par la lettre indiquée). Une fois que le joueur a fini, il peut choisir d'envoyer ses mots et tous les joueurs passent alors en phase de vérification, pendant laquelle le propriétaire de la

partie choisit d’accepter ou refuser chaque mot des joueurs. Le score est affiché en temps réel et indique donc l’avancement des scores.

Selon les paramètres du lobby, il peut y avoir différents rounds, le gagnant sera alors le joueur qui possède le plus de points.

De même que le Speed Typing, une minute après la fin le joueur est redirigé sur la page d’accueil.

EXPLICATION DES DIFFICULTÉS RENCONTRÉES

- Une difficulté importante a été de trouver quelle architecture Front choisir. En effet, nous avons eu le choix entre plusieurs technologies différentes : Blazor WebApp, Blazor WebAssembly, Razor Pages, etc. Au début du projet, l'architecture par défaut était une architecture utilisant les pages Razor, avec donc un ensemble de contrôleurs retournant des fichiers cshtml.
  Cependant, cette architecture ne nous convenait pas car nous voulions utiliser les composants Blazor, voulant profiter de leur syntaxe claire et de la facilité d'intégration de plusieurs composants entre eux.
  Nous avons donc choisi de migrer vers une architecture utilisant Blazor WebApp. Pour cela, nous avons dû comprendre le fonctionnement de Blazor WebApp pour transformer l'ensemble de nos fichiers en composants razor tout en respectant l'architecture nécessaire (Layout, routes, fichier d'imports, etc).
  Nous avons choisi de partir sur le modèle Blazor WebApp car il s'agit du modèle apparu en .NET 8, combinant donc les forces de Blazor WebAssembly et de Blazor Server, avec énormément de nouvelles fonctionnalités et une compatibilité complète avec toutes les nouveautés et futures nouveautés de .NET.

- Une autre difficultée rencontrée fut la gestion de la synchronicité, majoritairement dans le jeu du SpeedTyping qui requiert du temps réel, notamment pour afficher en continu la progression des joueurs.
  Pour cela, nous avons mis en place SignalR, qui est une bibliothèque permettant d'envoyer, à tout un groupe de personnes en même temps, un message.
  Par exemple, au SpeedTyping il faut afficher en continu les barres de progression de tous les joueurs et qu'elles soient visibles à tous en temps réel. Pour cela, il fallait à chaque mot ajouté par un des joueurs, envoyer un message à tous les joueurs pour les prévenir de mettre à jour cet élément côté front en se re-synchronisant avec les données côté back.
  L'aspect réellement technique sur cette partie a été la configuration initiale et sa mise en place à travers le système de hubs. En effet, il a fallu tout bien configurer, notamment

côté certificat SSL car nous rencontrions des erreurs. Enfin, il fallait bien s'assurer que chaque joueur soit dans le bon groupe et que chaque joueur ait bonne réception de tous les messages.

- Une partie importante de la réflexion autour de l'architecture de la plateforme a été le système d'authentification. En effet, nous devions trouver un moyen d'authentifier les joueurs de façon unique sans pour autant dépendre d'un mot de passe.
  Nous avons donc opté pour un système utilisant les UUID : à l'arrivée sur la plateforme, l'utilisateur va choisir un nom et ce nom ainsi qu'un UUID aléatoire seront sauvegardés dans le stockage local du navigateur. Cet UUID servira par la suite à authentifier le joueur. L'idée ici est de se dire que l'UUID est suffisamment grand avec suffisamment de caractères pour que le même l'UUID ne puisse pas être généré pour deux utilisateurs différents. Il y a en effet une probabilité de 0.5^128 que le même UUID soit généré, ce qui est donc dans notre cas supposé impossible.
  La suite de la mise en place de l'authentification a donc été de faire en sorte qu'à la création d'un lobby, l'utilisateur puisse se connecter avec cet UUID ce qui va donc créer à ce moment-là un utilisateur spécifique au lobby. L'UUID servira par la suite de "mot de passe" secret pour l'utilisateur, le système utilisant celui-ci pour associer chaque navigateur à un joueur.
  La dernière étape a été de mettre en place des composants Razor de factorisation, représentant par exemple une page nécessitant de l'authentification : si le joueur n'a pas dans son local storage d'UUID ou de nom, celui-ci n'est pas authentifié et est ramené à la page d'authentification. Les pages suivantes n'avaient plus qu'à hériter de cette page d'authentification pour bloquer l'accès.

- Enfin, notre dernière difficulté majeure résidait dans la gestion de nos instances de DBContext. En effet, de par l'aspect synchronisé de notre application, et notamment le SpeedTyping, il a pu arriver que nous fassions face à des problèmes d'accès concurrents à notre Base de données.
  En effet, nous n'avions qu'une seule instance par service et, avec nos méthodes asynchrones, cela pouvait arriver que 2 méthodes fassent appel au même DBContext en même temps, ce qui provoquait des erreurs.
  Nous avons donc décidé de mettre en place un IDbContextFactory qui est une factory permettant d'instancier un objet de type DBContext valable seulement dans la portée de la méthode. Chacun

de ces objets gérant uniquement les modifications propres à sa méthode et réglant les problèmes d’accès concurrents.

DÉROULEMENT DE LA GESTION DE PROJET ET LA RÉPARTITION DES TÂCHES

Gestion du projet

Pour gérer notre projet de la manière la plus efficace possible, nous avons adopté diverses solutions.

Nous avons notamment créé, dès le premier jour du projet, un excel répertoriant les exigences et fonctionnalités que nous aimerions implanter sur notre application de jeux.

De cet excel ont pu découler 2 choses :

- Un trello, permettant de lister tous nos tickets à traiter. Nous avons pu découper chaque jeu en petites étapes, en User Stories, afin d'avoir des tâches courtes pour éviter les blocages.
- 3 documents d'architecture concernant le SpeedTyping, le petit Bac et la plateforme. Cela nous a permis de nous faire des premières idées sur le modèle de données que nous utiliserons.

Comme cela est souligné, nous ne nous sommes pas jetés sur le code tout de suite mais avons préféré une approche plus réfléchie et procéder à une analyse plus poussée de ce qui était demandé, pour ne pas nous retrouver bloqués.

Pour garantir un code de qualité, nous avons mis en place sur notre Github un système de branches suivant ce schéma :

- main : la branche de production
- develop : la branche de développement commune

- feature/xxx-yyy où xxx désigne le numéro du ticket trello et yyy désigne un résumé de la tâche

Chaque merge d’une feature dans develop est systématiquement validé auparavant à l’aide d’une Pull Request. Il en va de même pour les merge de develop vers main.

Le fait d’avoir un Trello avec des tâches bien découpées et bien unitaires, permettait notamment d’avoir des Pull Requests avec moins de commit et moins de lignes de code, donc plus facilement lisibles et commentables.

Enfin, nous étions répartis par petits groupes pour chacun des lots, cela permettait d’avancer plus rapidement, tout en se coordonnant facilement et rapidement avec son binôme. En plus du Trello, des points étaient régulièrement faits entre les membres des binômes pour lister ce qu’il restait à faire ou corriger. Ainsi, chaque membre du projet avait en vision l’avancée de chacun mais aussi savait les tâches sur lesquelles il allait pouvoir travailler.

Tout cela a permis d’amener à la gestion d’un projet de qualité et maîtrisé.

Répartition des tâches

Pour répondre à toutes les attentes de notre cahier des charges, nous nous sommes répartis le travail ainsi :

- Erwann : Je suis intervenu sur deux parties du projet. Tout d'abord j'ai commencé par la plateforme, en créant l'ensemble des entités. Concernant la partie métier de l'application je me suis occupé de :
    - Créer les lobbys publics
    - Créer les lobbys privés avec mot de passe chiffré, on ne stocke pas de mots de passes en clair
    - Lister l'ensemble des lobbys disponibles
    - Créer le front pour pouvoir se connecter
    - Créer le front pour accéder au différents lobbies
    - Créer le front pour accéder aux détails du lobby

  Concernant le petit bac, j'ai pu faire les choses suivantes :

    - Affichage du tableau des scores avec rafraîchissement pour tous les joueurs lors d'une mise à jour du classement
    - Gérer la vérification des mots de tous les joueurs par le créateur de la partie uniquement
    - Réaliser l'affichage des inputs du joueur lors de la phase de remplissage des mots
    - Réaliser l'affichage des inputs de l'ensemble des joueurs lors de la phase de validation
    - Mise en place de SignalR sur le jeu pour pouvoir avoir un rafraichissement pour l'ensemble des joueurs

  J'ai aussi participé à la rédaction du backlog initial en début de projet.

Enfin, j’ai participé à la gestion de projet en alimentant au fur et à mesure le tableau des tâches et en le mettant à jour.

- Hortense : Pour ma part j’ai réalisé le petit bac en coopération avec Erwann

    - Réalisation des entités du petit bac

    - Création d’une partie de petit bac

    - Méthode de création et d’initialisation d’un nouveau tour de jeu

    - Saisie et validation des mots du joueur en temps réel

    - participation à la création du formulaire d’entrée des mots des joueurs

    - Méthode pour vérifier que le joueur a rempli toutes ses catégories

    - Ajout des catégories de bases dans la base de données via une migration

    - Envoie des mots du joueur pour confirmation et vérification de la complétude des catégories

    - Affichage en rouge des mots invalides de manière dynamique

- Etienne : Je suis intervenu sur différents plans du projets, avec plusieurs étapes :
    - Réalisation de l’environnement DevOps du projet, avec l’initialisation du projet de base et de l’environnement docker
    - Mise en place des différentes solutions Visual Studio pour la création et l’application automatique des migrations, permettant de simplifier le développement
    - Mise en place de l’architecture Front du projet utilisant Blazor WebApp
    - Réalisation de la plateforme avec Erwann, comprenant :
        - la mise en place de l’authentification (sans mot de passe, via UUID)
        - gestion de la création des lobbys publics / privées
        - détail d’un lobby
        - recherche d’un lobby
        - choix et instanciation du jeu à la création d’un lobby
    - Gestion des autorisations cotée front avec une redirection de l’utilisateur si celui-ci n’a pas les accès nécessaires
    - Gestion des fichiers Razor de base
    - Aide à la mise en place de SignalR
    - Aide sur les bonnes pratiques .NET

- Axelle : J’ai pour ma part pu participer à différents “lots” du projet
    - La réalisation du Speed Typing, simultanément avec Florian
        - notamment avec la mise en place de SignalR (avec les configurations initiales et la mise en place complète sur l’application)
        - l’affichage de la progression des scores des joueurs
        - la saisie de mots par l’utilisateur
        - La mise en place du timer côté front sur le Speed Typing
    - La mise en forme du site, notamment le design et le choix des couleurs
        - La mise en forme sur le Speed Typing
        - La mise en forme sur le Petit Bac
        - La mise en forme sur les listes, détails, page d’accueil
    - La mise en place de l’arborescence initiale du projet avec ajout des entités du Speed Typing
    - J’ai pu résoudre différents bugs sur la plateforme
        - Liés à l’authentification
        - A des problèmes de synchronisation
        - A des problèmes visuels (de CSS, d’imbrications HTML)
    - J’ai pu ajouter quelques vérifications générale par exemple sur la vérification des joueurs dans un lobby, s’il s’agit d’un host, pour afficher ou non certaines options sur la plateforme
    - La mise en place du Trello avec le backlog initial, ainsi que le suivi des tâches et de la gestion de projet, afin d’assurer une bonne communication et un avancement continu.

- Florian : Je suis majoritairement intervenu sur le jeu du SpeedTyping en coopération avec Axelle
    - Aide à la création des entités du SpeedTyping
    - Création d’une partie de SpeedTyping
    - Affichage des mots à écrire
    - Mise en évidence du mot courant pour le joueur
    - Mise en évidence d’une erreur de saisie par le joueur
    - Vérification du mot saisi (s’il est égal au mot courant)
    - Gestion des scores finaux d’un joueur
    - Gestion de la fin de partie (si tous les joueurs ont fini ou si la partie a commencé depuis une minute)
    - Affichage du chronomètre à l’utilisateur
    - Gestion des 2 chronomètres côté backend (le premier gérant le fait de savoir quand la partie prend fin, le second gérant la suppression automatique des objets)
    - Alimentation des tables liées aux highscores du SpeedTyping si nécessaire
    - Affichage du classement à la fin de la partie
    - Redirection à la fin de la partie et suppression des objets liés à la partie
    - Côté plateforme : affichage front des highscores pour chacun des jeux
    - Gestion de projet : création des documents de gestion de projet et alimentation du Trello pour le backlog initial et mise à jour régulièrement pour les autres tâches. Suivi de l’avancée de chacun des membres pour garder une communication fluide et une vision claire du produit

BILAN SUR LE PROJET

- - - - X

La plus grosse difficulté que nous avons rencontré sur ce projet a été le temps et comment bien nous organiser pour réussir à faire efficacement cette application. Nous avions toutefois un bagage concernant les plateformes de jeux, en ayant déjà fait par le passé. Ce projet nous a donc motivé sur le fond, tout en nous permettant d'apprendre (ou approfondir) nos connaissances en .NET. Nous avons pu manipuler un ensemble d'outils, enrichissant ainsi notre palette de technologies.

Certains éléments sont perfectibles sur notre projet actuel, comme la proposition de tirage d'un nombre variable de catégories, que l'host peut choisir, pour le petit bac, ou encore de même pour le nombre de tours. Nous aurions pu également afficher un timer qui indique quand la partie se fermera automatiquement pour nous ramener à la page d'accueil.

Mais bien que le projet soit perfectible, nous ressortons grandi de cette expérience, et présentons fièrement ce que nous avons réussi à produire.