Développement d'une version mobile du jeu de société Boggle
Projet de programmation 2

Bernardon Vincent - Duban Mathis - Gonzalez O. Gilles - Hurel Jeremy - Khelifa Chanez
L3 informatique
Faculté des Sciences
Université de Montpellier.

2023-2024

Remerciements

Nous tenons tout d'abord à remercier nos encadrants Vincent Boudet et Clémentine Nebut de nous avoir accompagné et guidé tout au long du développement du projet au fil de ce semestre.

Nous tenons également à remercier tous les enseignants qui, par leurs enseignements nous ont permis d'acquérir les compétences techniques nécessaires pour le bon déroulement du projet.

Résumé

Le rapport présenté ici porte sur le projet réalisé dans le cadre de l'UE HAI606I : Projet de Programmation 2. L'objectif de ce projet était de développer une version mobile du célèbre jeu de société Boggle qui consiste à trouver le maximum de mots possibles dans une grille de lettres. Tout au long du semestre, nous avons travaillé sur la conception et l'implémentation de cette application, que ce soit la mise en place de l'environnement dans lequel le jeu évolue ou encore l'implémentation d'un système permettant de jouer parties multijoueurs en ligne. Ce rapport résume nos efforts de développement, les défis rencontrés, solutions apportées, ainsi que les fonctionnalités finales de l'application résultante. Dans la première partie de ce rapport, nous présenterons le sujet du projet afin de mettre en place les besoins de ce dernier. Dans un second temps, nous verrons les différentes technologies que nous avons utilisées tout au long de ce projet. Dans une troisième partie, nous vous présenterons les différents points clés du projet, notamment leur conception, modélisation et leurs implémentations. Par la suite, nous traiterons de la manière dont nous nous sommes organisés tout au long du projet afin de le mener à bien. Et pour finir, nous établirons un bilan du travail qui a été réalisé au fil du temps, ainsi que les améliorations possibles pour le projet.

The report presented here relates to the project carried out within the course HAI606I : Programming project 2. The aim of this project was to develop a mobile version of the popular board game Boggle, which involves finding as many words as possible in a grid of letters. All throughout the semester, we worked on the design and implementation of this application, from setting up the environment in which the game evolves, to the implementation of a system of an online multiplayer system. This report summarizes our development efforts, the challenges we faced, the solutions we came up with, as well as the resulting application. In the first part of this report, we introduce the project's subject matter in order to set out the project's requirements. Secondly, we'll look at the various technologies used throughout the project. In the third part, we'll key points of the project, including design, modeling and implementation. We'll then look at how we organized ourselves throughout the project to bring the project to a successful conclusion. And finally, we'll take stock of the work that's been and possible improvements for the project.

Table des matières

Table des figures

1 Présentation du Sujet

1.1 Contexte et problèmes du sujet

Le jeu de Société "Boggle" est un jeu de lettre conçu par Alan Turoff et distribué par la société "Parker Brothers" (une branche de Hasbros) en 1972. Le jeu met à l'épreuve la capacité des joueurs à former des mots à partir d'une grille de lettres dans un temps limité.

Le jeu se déroule sur une grille de jeu 4 x 4 sur laquelle sont rangés sur chaque case un dé. Chaque face des dés possède une lettre spécifique à l'exception de la lettre Q qui est associée au U pour former la combinaison de lettres QU (pour la langue française).

Avant le début de chaque partie, la grille contenant les dés va être secouée afin de mélanger les différentes lettres, une fois cette étape réalisée, la grille sera prête, le jeu peut commencer.

À chaque tour, les joueurs vont devoir trouver le maximum de mots possibles présents sur la grille en un temps imparti (matérialisé par un sablier de 3 minutes). Pour qu'un mot soit valide, les différentes conditions suivantes doivent être réunies :

- Les lettres formant un mot doivent être adjacentes, que ce soit horizontalement, verticalement ou dans les diagonales inférieures ou supérieures.

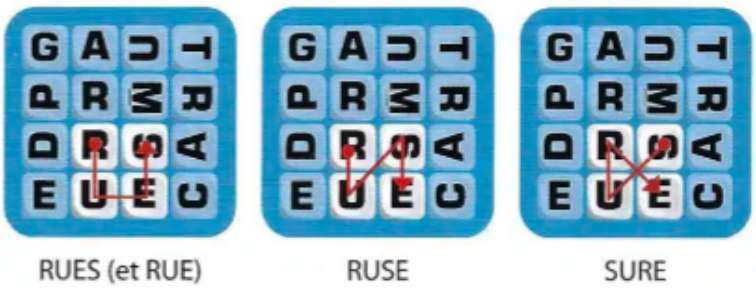


FIGURE 1 – Exemple de déplacements valides dans le jeu Boggle.

— Le mot obtenu doit être composé de trois lettres au minimum

— La réutilisation des lettres au sein du même mot est interdite

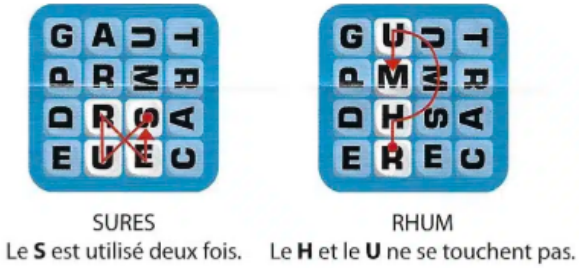


FIGURE 2 – Exemples de déplacements non valides dans le jeu Boggle.

Pour qu'un mot soit valide, il faut que ce dernier fasse partie du dictionnaire de mots de la langue française (les verbes conjugués et les mots au pluriel sont également autorisés). Les noms propres, acronymes et noms abrégés sont exclus des mots valides.

Une fois le tour des joueurs passé, ces derniers vont comparer leur liste de mots trouvés afin de calculer leurs scores respectifs :

— les mots composés de 3 ou 4 lettres valent 1 point.
— les mots composés de 5 lettres valent 2 points.
— les mots composés de 6 lettres valent 3 points.
— les mots composés de 7 lettres valent 5 points.
— les mots composés de 8 lettres ou plus valent 11 points ou plus.
— si les joueurs ont trouvé 1 mot en commun, ce dernier doit être retiré chez chacun des joueurs.

Le partie est alors terminée, le joueur ayant remporté le plus de point remporte la partie.

Le jeu n'étant pas aussi populaire que d'autres variantes de jeux de lettres (Scrabble, mots fléchés), il n'existe pas beaucoup de possibilités pour jouer au Boggle en ligne. C'est dans ce contexte que la réalisation du jeu est pertinente, le projet s'ancre dans l'objectif de développer une version mobile du jeu Boggle.

1.2 Cahier des charges

Le cahier des charges pour le développement de l'application Boggle se concentre sur plusieurs fonctionnalités clés, visant à créer une version mobile du jeu Boggle identique à sa version physique.

Dans sa version initiale, l'application devra :

— générer aléatoirement une grille de lettres conformément aux règles du Boggle.
— permettre aux joueurs de proposer des mots en effectuant des gestes sur l'écran tactile.
— vérifier la validité des mots proposés en les comparant à un dictionnaire intégré et en s'assurant que les lettres sont adjacentes dans la grille.
— calculer le total des points obtenus par chaque joueur dans le temps imparti pour la partie.
— afficher l'ensemble des mots possibles sur la grille à la fin de la partie et permettre aux joueurs de demander les positions spécifiques de ces mots.

Dans la deuxième phase de développement du projet, l'application sera étendue pour inclure une version multijoueur avec un serveur de jeu. Cette extension comprend :

— la mise en place d'un serveur de jeu pour gérer les parties multijoueurs, les connexions et les échanges de données entre les joueurs.
— l'ajout d'une fonctionnalité d'affichage en direct des scores des participants, permettant aux joueurs de suivre l'évolution des scores de leurs adversaires en temps réel.
— un système permettant de vérifier que les règles du Boggle sont correctement appliquées pour tous les joueurs présents, notamment en ce qui concerne la validité des mots proposés et le calcul des points.

2 Choix des technologies

Langages et Frameworks

Le projet étant un jeu mobile, il était nécessaire d'avoir un environnement qui nous permettait de compiler pour au moins une distribution android. Les distributions iOS n'étaient pas prioritaires, car nous aurions eu besoin d'un ordinateur Apple pour compiler, et aucun membre du groupe n'en possédait un.

Nous avons donc orienté nos recherches vers un framework pour les applications mobiles. Deux options se sont démarquées : Flutter et React Native. Bien que nous ayons également envisagé la possibilité de rechercher un framework basé sur Kotlin, cette option n'a pas été explorée en raison d'un manque d'affinité avec le langage. Ce manque d'affinité est principalement dû aux nombreux problèmes de fuites de mémoire associés à Java, le langage vers lequel Kotlin est compilé, ainsi qu'à la réputation de lenteur de Java, notamment en raison de l'utilisation d'une machine virtuelle pour fonctionner. Ainsi nous avons dû choisir entre le framework React Native et Flutter.

Le premier langage que nous avons envisagé est **JavaScript**, avec le framework React Native. React Native offre la possibilité de créer des applications multiplateformes pour Android et iOS, ce qui en a fait un candidat attrayant. De plus, sa popularité et sa large communauté de développeurs ont été des points forts. Cependant, une partie du groupe ayant déjà travaillé avec ce langage, nous avons pris en compte le désir d'explorer de nouvelles technologies et d'acquérir de l'expérience sur un autre langage. Cela nous a conduits à rechercher d'autres options pour notre projet mobile.

Finalement, notre choix s'est porté sur **Flutter**. Flutter est un framework open-source développé par Google, conçu pour créer des applications multiplateformes (Android et IOS) avec une seule base de code. Il utilise le langage de programmation Dart. Flutter se distingue par sa performance élevée et sa capacité à produire simplement des interfaces graphiques via des composants encapsulés (portions de code). Flutter possède également une grande variété d'outils disponibles gratuitement sur le site officiel des développeurs ce qui permet d'incorporer facilement certaines fonctionnalités complexes. Etant un des langages les plus populaires en développement mobile dans l'industrie et ayant une communauté grandissante, nous avons choisi de partir avec Flutter afin de découvrir ce langage tout au long de ce projet afin d'acquérir de l'expérience sur ce nouveau langage.

La seconde technologie que nous avons choisie pour la gestion des données et le backend est l'outil **Firebase**. Firebase est un Saas (Software as a Service), un outil permettant d'utiliser plus facilement les outils du Google cloud tout en proposant une interface supplémentaire afin de rendre plus simple son utilisation. Les avantages de l'utilisation de ce service sont multiples. Tout d'abord, son système de facturation à l'usage, ce qui signifie que pour un faible trafic de données, les coûts sont minimes, souvent proches de zéro. De plus, l'absence de gestion de la montée en charge permet de supporter jusqu'à 100 000 utilisateurs simultanés pour la base de données en temps réel, ce qui sera particulièrement utile pour le mode multijoueur.

Un dernier langage a été utilisé pour concevoir l'application, **Golang**. Go est un langage avec une syntaxe minimaliste et qui promet des performances assez élevées pour tout ce qui est opération système. De plus son système de typage et de "goroutine" fait de lui choix plus pertinent que Python pour la conception d'un outil interne permettant de convertir un dictionnaire sous forme de liste de mots en dictionnaire sous forme d'arbre de recherche, mais aussi pour décoder l'arbre.

On retrouvera aussi du Golang dans l'application pour certaines opérations de manipulation de bits, dart n'étant pas conçu pour ce genre d'opération.

Outils

Pour l'outil de Gestionnaire de Version (VCS), il a été décidé de travailler sur **GitHub**. Contrairement à GitLab, GitHub met à disposition des runners de pipeline en ligne. Cette fonctionnalité est particulièrement avantageuse car elle nous permet d'automatiser la phase de test de notre projet, déployer en ligne une version compilée et s'assurer que la version finale déployée est bien compatible avec Android et IOS afin que l'application fonctionne chez le plus grand nombre d'utilisateur.

Concernant la planification des tâches, nous avons choisi **Notion**. Notion est une plateforme centralisée qui permet de planifier et attribuer des tâches pour chacun des membres du groupe.

Dans le cadre de la sauvegarde des données et la mise en place des parties multijoueurs, il fallait un outil permettant de stocker et récupérer les données en ligne. Pour cela, nous avons choisi l'outil gratuit le plus utilisé pour sa simplicité et comptabilité avec les fonctionnalités Google : **Firebase**.

3 Développements Logiciel : Conception, Modélisation, Implémentation

3.1 Présentation des modules principaux

Au fil de l'avancement de notre projet, nous avons pris des mesures pour accélérer la phase de développement et de livraison de l'application tout en minimisant le risque d'erreur qui pourrait mettre à mal le developpement du projet. Nous avons décidé de suivre les principes clés du développement DevOps, ce qui a grandement facilité notre processus de développement.

En génie logiciel, le DevOps est une pratique informatique visant à accélérer et à automatiser le processus de développement d'un projet, tout en cherchant à minimiser les éventuels problèmes. En raison de sa nature itérative, le DevOps est symbolisé par une boucle continue qui illustre la manière dont ses différentes phases sont interconnectées dans le cycle de vie du projet.

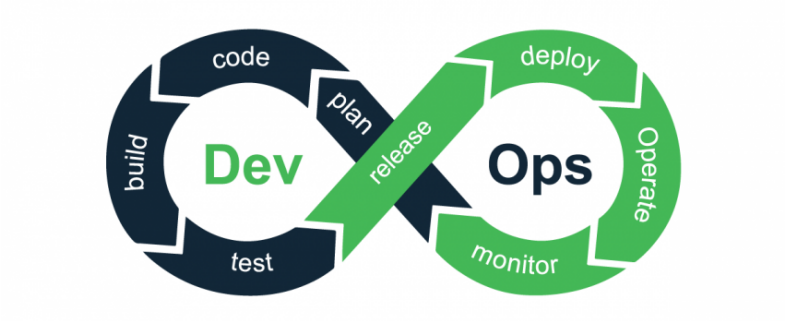


FIGURE 3 – Boucle DevOps

Planification des tâches (plan)

La partie **plan** de la boucle du DevOps représente la phase de planification des tâches à laquelle les différents membres de l'équipe sont affectés au fur et à mesure du développement du projet. Dans notre cas, nous avons utilisé l'outil **Notion**. Notion est une plateforme de productivité permettant l'intégration de tâches planifiées dans un environnement dédié, la création de diagrammes de Gantt ou encore la mise en avant de document pour les membres de l'équipe.

Dans notre cas, nous avons décidé d'utiliser cet outil de 2 façons différentes :

— Afin de distribuer les différentes tâches, nous y avons créé un espace dédié. Chaque tâche comprend plusieurs informations : le sprint dans lequel la tâche évolue, son descriptif, sa catégorie (*to do* pour les tâches à faire, *in progress* pour les tâches déjà commencées, *done* pour les tâches finies et *tentative* pour les essais facultatifs à apporter) ainsi que la personne qui y est affectée.

— Pour centraliser toutes les ressources du groupe telles que les règles officielles, les schémas de conception ou encore les dépôts GitHub.

Gestionnaire de Version (code)

Dans le contexte de la boucle DevOps, la partie **code** implique la gestion du code source, son développement et sa collaboration. Nous avons opté pour une approche collaborative entre les membres de l'équipe afin d'avoir un code partagé par tous afin d'éviter les éventuels problèmes découlant des différentes versions du code. C'est le rôle de **GitHub**, notre outil VCS (service de contrôleur de Version), permet de développer simultanément différentes fonctionnalités situées sur plusieurs branches pour de ne pas avoir de conflit. Github nous permet également de fusionner le travail de chacun pour sortir des versions compilées et testées plus facilement. Nous utilisons l'outil GitHub pour :

— Créer un système de hiérarchisation de branches. La branche principale **main** ne contient que la version la plus complète du projet qui est testée et validée. La branche **dev** qui contient le code fusionné des différentes fonctionnalités développées. Pour le reste, chaque fonctionnalité en cours de conception a sa propre branche.

— Garder un historique des changements réalisés avec un descriptif à chaque modification.

— Permettre d'appliquer des modifications dans notre code facilement grâce à une branche "hotfix".

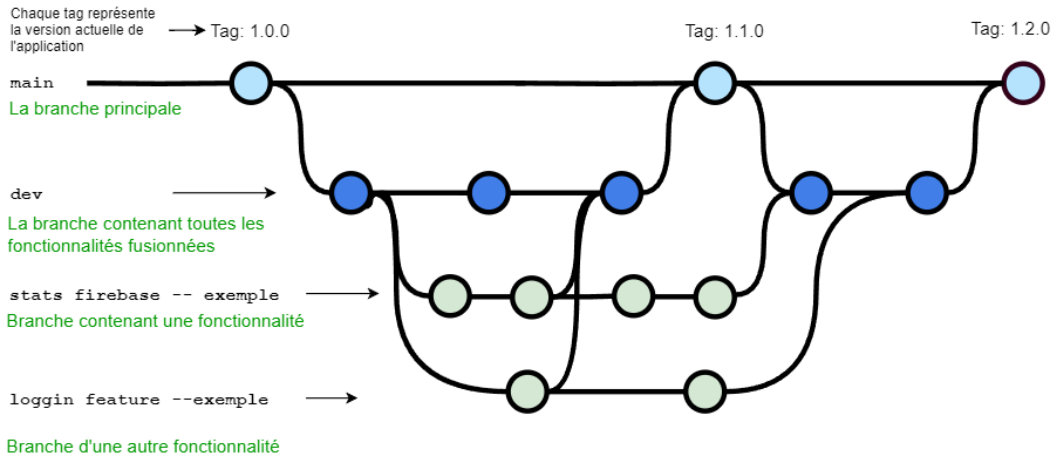


FIGURE 4 – Schéma représentant le système de branches du projet

Pipeline Github (build)

L'étape build dans la boucle DevOps correspond à la phase où le code source de l'application est transformé en un format exécutable (.apk) ou en artéfact (une version compilée stockée dans la pipeline) pour être par la suite testé.

Après avoir atteint cette étape, l'objectif était de mettre en place un système de pipeline directement intégré depuis GitHub afin d'automatiser toutes les étapes restantes de la boucle. Une pipeline GitHub est essentiellement une série d'actions automatisées personnalisables permettant d'effectuer différentes opérations **(jobs)** à travers plusieurs étapes **(stage)**. À l'intérieur d'un stage, plusieurs étapes peuvent s'exécuter en parallèle pour économiser du temps d'exécution, ou bien en série si le job nécessite des actions spécifiques préalables. Chaque pipeline fonctionne grâce à un **runner** en ligne, et dans notre cas, nous utilisons les runners proposés par GitHub gratuitement. Un **runner** est un outil qui tourne sur un serveur utilisé pour exécuter les tâches d'une pipeline.

Le comportement de la pipeline est décrit dans un fichier .yaml. Ce fichier est essentiellement une série de paires clé-valeur contenant des mots-clés spécifiques ainsi que des commandes bash associées. En d'autres termes, il s'agit d'un fichier qui organise les différentes actions à effectuer lors de l'exécution du pipeline.

— La première partie du pipeline est dédidée à son nom avec la balise **name**.

— La deuxième partie indique les conditions dans lesquels les jobs doivent se lancer avec le mot clé **on**. Par exemple, le mot clé **push : branches : -main** est indiqué, cela signifie que les jobs s'exécuteront lorsqu'un push sera effectué sur la branche main

— La dernière partie énumère tous les jobs présents dans la pipeline et les lignes de code Bash qui y seront exécutées.

Voici un extrait du code de notre pipeline contenant le code pour build le projet :

```yaml
build_apk:
  name: Build the APK Version
  runs-on: macos-latest
  needs: setup_and_test # après le succès du job setup_and_test
  steps:
    - name: Checkout code # récupération du code
      uses: actions/checkout@v3

    - name: Set up Java # installation de l'environnement Java dans la pipeline
      uses: actions/setup-java@v1
      with:
        java-version: '11.x'

    - name: Set up Flutter # installation de l'environnement Flutter dans la pipeline
      uses: subosito/flutter-action@v2
      with:
        flutter-version: '3.16.7'

    - name: Build APK # Création d'un fichier exécutable APK de l'application
      run: flutter build apk

    - name: Archive APK # Archivage du fichier APK dans l'espace artefacts dedié dans la pipeline
      uses: actions/upload-artifact@v2
      with:
        name: release-apk
        path: build/app/outputs/flutter-apk/
```

Sur ce code, notre pipeline va exécuter le job nommé build_apk, ce job utilise un runner (parmi ceux proposés en ligne), et réalise différentes étapes (steps) après que le job setup_and_test soit correctement exécuté :

— Le code va être récupéré.

— L'environnement virtuel Java va être instancié dans cette étape de la pipeline.

— L'environnement virtuel Flutter va être instancié pour cette étape également.

— Et enfin une version compilée (.apk) va être générée puis envoyée vers un artéfact virtuel.

Pipeline Github (test)

Dans la boucle DevOps, la partie test correspond à la réalisation de tests unitaires afin de tester que les nouvelles fonctionnalités développées ne nuisent pas au fonctionnement global de l'application.

Pour cela, nous avons utilisé le package Flutter **flutter_test package** directement intégré dans Flutter afin de réaliser la série de tests unitaires. Nous avons donc codé chaque test unitaire dans un fichier spécifique afin de tester les fonctionnalités principales du jeu (connexion, parties, options). Voici un exemple de code de test unitaire réalisé en Flutter :

```dart
test('grille contient un bon nombre de E', () {
    List<String> letters = listeDe.roll();
    int count = 0;
    for (String letter in letters) {
      if (letter == 'E') {
        count++;
      }
    }

    logger.i('Nombre de E: $count');
    //au maximum 12 fois la lettre E
    expect(count, lessThanOrEqualTo(countLetterInDiceSet(listeDe, 'E')));
  });
```

Afin d'automatiser nos tests unitaires, nous avons développé un job spécifique pour les tests unitaires lors de la phase pre-build :

```yaml
jobs:
  setup_and_test:
    name: Setup, Test & Cache
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code # récupération du code
        uses: actions/checkout@v3

      - name: Set up Java # installation de l'environnement Java dans la pipeline
        uses: actions/setup-java@v1
        with:
          java-version: '11.x'

      - name: Set up Flutter installation de l'environnement Flutter dans la pipeline
        uses: subosito/flutter-action@v2
        with:
          flutter-version: '3.16.7'

      - name: Cache Flutter packages # mise en cache des packages Flutter récupérés
        uses: actions/cache@v2
        with:
          path: | # chemin du dossier contenant les packages
            ~/.pub-cache
          key: ${{ runner.os }}-flutter-${{ hashFiles('**/pubspec.yaml') }}

      - name: Install Dependencies # installation des dépendences récupérées
        run: flutter pub get

      - name: Run Tests # lancement des tests
        run: flutter test
```

Sur ce code, le pipeline va réaliser le job **setup_and_test**. Tout d'abord les premières étapes du job (steps) vont mettre en place tout l'écosystème nécessaire pour la mise en place des tests unitaires tel que l'environnement Java, Flutter et la mise en cache des fichiers Flutter générés (afin de gagner en temps d'éxécution dans une étape ultérieure). On va ensuite installer les dépendances utilisées par le projet et enfin, l'exécution des tests unitaires.

Pipeline Github (release)

Dans la boucle DevOps, la partie "release" représente la phase où les nouvelles fonctionnalités ou les mises à jour du logiciel sont déployées dans l'environnement de production après avoir été testées et validées dans des environnements de test ou de pré-production. Ici, nous avons automatisé ce processus à l'aide de notre pipeline :

```yaml
push_to_release:
    runs-on: macos-latest
    needs: build_apk
    steps:
    - name: Download APK artifact # récupération du fichier APK présent dans les artéfacts

      uses: actions/download-artifact@v2
      with:
        name: release-apk
        path: build/app/outputs/flutter-apk/

    - name: Create GitHub Release # création d'une nouvelle release GitHub pour l'archivage

      uses: softprops/action-gh-release@v1
      with:
        files: |
          build/app/outputs/flutter-apk/*.apk
        token: ${{ secrets.PIPELINE_SECRET_BOOGLE_L3 }}
        tag_name: v1.0.${{ github.run_number }} # nom du tag associé
        release_name: Release v1.0.${{ github.run_number }} # nom de la release
      env:
        GITHUB_TOKEN: ${{ secrets.PIPELINE_SECRET_BOOGLE_L3 }} # utilisation du token
```

Ici, notre pipeline réalise le job **push_to_release**. Ce job va tout d'abord récupérer le fichier .apk que nous avons intégré dans un artéfact lors de la phase de build (ainsi nous n'avons pas besoin de le regénérer). Ensuite, le pipeline va créer une **release** GitHub du fichier .apk avec l'identifiant d'exécution actuel pour le stocker dans l'onglet release du projet GitHub. Sur GitHub, une release est un espace permettant de télécharger certains fichier directement depuis la plateforme. Une fois le fichier .apk déployé dans l'onglet release, il est possible de récupérer le fichier source d'installation pour le déployer sur son portable.

En appliquant les principes du DevOps, nous avons pu créer un pipeline de développement robuste et efficace, favorisant une livraison continue et itérative des fonctionnalités. Cette approche a non seulement accéléré notre rythme de développement, mais elle a également amélioré la qualité et la stabilité de notre application, grâce à des tests automatisés rigoureux et des déploiements fiables. Voici à quoi ressemble l'exécution globale du pipeline :

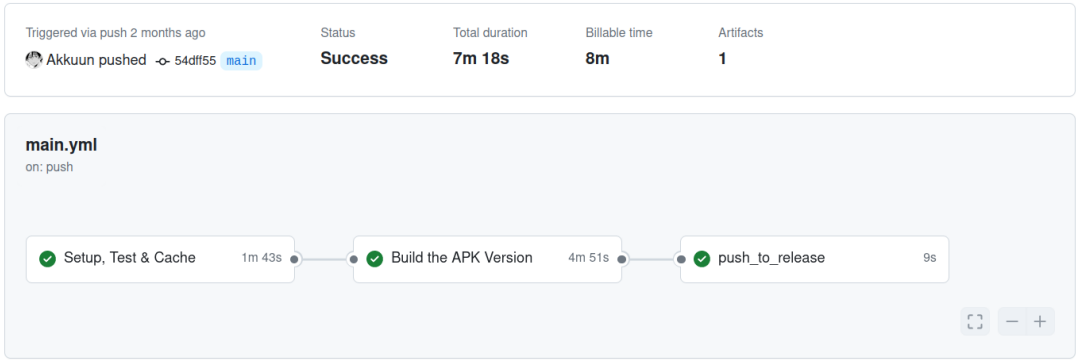


FIGURE 5 – Extrait de l'exécution du pipeline dans l'onglet Action

A chaque exécution, nous voyons bien le statut **Sucess** du pipeline qui signifie que tous les jobs se sont correctement exécutés lors du push sur la branche main. Tous les jobs se suivent correctement, et mettent en artéfact les fichiers pré-compilés pour un total de 7 minutes et 18 secondes (ce qui peut paraître long, mais cela s'explique par le fait que nous utilisions les runners gratuits de GitHub qui ne sont pas les plus performants, surtout pour des tâches complexes comme la compilation des fichiers .apk). A ce stade, notre pipeline permet de certifier que le code délivré est testé, compilé et qu'il est prêt à être déployé sur des téléphones clients.

3.2 Partie algorithmique du projet : exploration des mots possibles sur une grille

Besoins pour la réalisation du Boggle

Dans le cahier des charges de l'application, il était question de pouvoir vérifier si pour chaque mot, ce dernier était bien présent dans le dictionnaire ainsi que la possibilité d'afficher tous les mots trouvables dans une grille.

Afin de donner la possibilité aux joueurs de jouer en mode hors-ligne, il a été décidé de stocker les différents dictionnaires dans l'exécutable. Cela nous a alors apporté deux nouvelles problématiques :

— les dictionnaires ne devaient pas être trop volumineux afin de ne pas restreindre les performances de l'application.

— les dictionnaires devaient également être dans une forme permettant de récupérer dans un temps minimum n'importe quel élément présent dans le dictionnaire.

Format des données

Le premier choix que nous avons dû faire concerne la représentation des données, nous avons opté pour une représentation sous **forme d'arbres de recherche**. Cette représentation permet de savoir rapidement si une suite de lettres peut aboutir à un mot valide, ce qui est très utile pour la mise en place d'algorithmes exhaustifs.

Ensuite, nous nous sommes penchés sur le format des données présentes dans notre arbre, nous avons décidé de partir avec une représentation via le **type** de Golang.

Le type de Golang est une représentation permettant de créer ses propres types accompagné d'attributs contenant des données, nous avons alors imaginé le type "Node" qui serait présent dans notre arbre contenant plusieurs informations telle que la lettre actuelle du parcours du mot, un booléen pour spécifier si cette dernière est terminale et la suite du mot via le Node suivant :

```
1 //definition d'un type en Go
2 type Node struct {
3   Letter string // lettre de longueur 1 dans notre cas
4   DoEndWord bool
5   Next []Node
6 }
```

Cependant même si le type “Node” ci-dessus est très pratique pour programmer, celui-ci prend énormément de place quand on l'encode dans un fichier JSON. En effet, lorsque l'on utilise cette représentation pour générer un arbre, nous arrivons à un fichier de plus de 50 Mo qui est beaucuoup plus lourd que sa représentation sous forme de liste pesant 4.32 Mo.

Si nous voulions générer notre arbre de recherche avec ce type Node, nous devions régler ce problème de stockage. La première approche que nous avons tenté a été de voir s'il n'était pas possible de trouver le sous-ensemble des mots réalisables pour toutes les grilles de boggle. Cette approche pose un certain nombres de problèmes car il n'est pas possible de trouver cet ensemble de manière exhaustive car cela comporte plus de $\pm 6^{16}$ grilles possibles.

Notre deuxième approche a été de mettre en place un algorithme dynamique mettant en relation le nombre de lettres sélectionnées et le nombre de lettres disponibles. Malheureusement ce dernier ne validait pas la capacité de l'algorithme à fournir toutes les solutions possibles pour le problème donné, et il n'y avait pas une baisse significative dans le poids final de l'arbre.

La dernière approche, et celle que l'on détaillera, consista à réduire au maximum le nombre de caractères écrits dans le JSON sans perdre d'information et sans étape de compression. Ceci est un bon compromis entre rapidité d'accès vers la donnée et la possibilité d'optimisation de la RAM utilisée et le poids de l'application.

Etapes pour réduire le poids de la donnée

Avant d'optimiser notre façon d'encoder l'information, il nous faut tout d'abord connaître les limites des restrictions du format JSON imposé. Lorsque l'on encode une structure en JSON, l'information va prendre la forme suivante :

```
1 {
2   "key" : value,
3 }
```

En réalité, la donnée JSON est une chaîne de caractères qui est codée sur 32 ou 64 bits. Dans l'exemple ci-dessus la clé est une chaîne de caractère de longueur 3 mais encodée en 5 caractères à cause des guillemets. Afin de gagner de la place dans notre représentation, on peut réduire l'attribut "key" à une chaîne de caractères de longueur 1 ainsi afin de représenter un objet nous n'aurons besoin que de 24 bits au minimum pour le coder.

Il n'existe pas de moyen d'encoder un caractère seul sur 8 bits, car si on stocke le code ASCII de la donnée on aura un entier de 32 bits. Une des idées retenue a été de stocker une chaîne de caractères de longueur 1 ce qui nous ramène à 24 bits (avec les guillemets). Les caractères étant codés sur 8 bits il est possible de combiner l'information "DoEndWord" et "Letter" sur 32 bits.

Ainsi, on passe d'une donnée prenant en moyenne $3 \times 24bits + 40bits + 2 * 8bits = 128bits$ (2 clés + un caractère + la valeur "false" + les 2 " :") à une donnée qui prend $24bits + 32bits + 8bits = 64bits$. (une clé + l'information combinée)

Avec ces modifications, on arrive à générer un fichier de 11,78 Mo pour la langue française. Cependant si la donnée réelle prend $32bits$ cette dernière représentera uniquement la clé de la valeur. Cependant, il est possible de se passer des clés de valeur, en effet une structure n'est qu'une représentation labélisée d'un tableau, ainsi on peut réécrire le type original ("Node") de cette manière :

```
1 type Node [2]interface{} //interface est l'  quivalent   de void en C
2 // Node[0] = valeur
3 // Node[1] = []Node
```

Avec ce nouveau format on supprime toutes les clés de valeur ce qui fait économiser $32bits$ de données par clés plus les " :". Egalement, au lieu d'avoir le mot-clé *null* pour informer que nous sommes à une feuille de l'arbre, nous pouvons uniquement encoder la valeur ce qui fait gagner $16bits$ par feuille de l'arbre, ce qui est non négligable lorsque l'on encode plus de 400 000 mots.

Après toutes ces transformations on arrive à obtenir des dictionnaires d'environ $4,6Mo$ ce qui est convenable en comparaison avec le fichier texte original (A noter que c'est une version customisée du format BSON qui a été implémentée puisque celui-ci garde une trace du format de donnée et transforme les structures en tableau pour économiser de l'espace).

Validité d'un mot et recherche de tous les mots

Dans cette section, nous cherchons un moyen de trouver l'ensemble des mots réalisables dans une grille donnée en utilisant uniquement l'association de lettres adjacentes. Chaque lettre de la grille peut

être vue comme un point de coordonnées (i,j). Lorsque nous avons une grille de Boggle générée, il est en réalité assez simple de vérifier si ce mot est bien valide ou non. Afin de vérifier la validité du mot, nous avons mis en place différentes fonctions afin d'extraire la valeur des caractères et de vérifier si notre mot obtenu est correct :

```
func EndOfWord(v C.int) bool { //l'information endword est encodee sur le 9ieme bits
  return (v & 0b100000000) != 0
}

func GetChar(v C.int) C.char { //utile pour l'implementation en dart
  return C.char(v)
}


func IsSameKey(ps *C.char, key C.int) bool { //utile pour l'implementation en dart
  s := C.GoString(ps) //conversion vers le type string de golang
  return s[0] == byte(key)
}

func (this *Node) GetChild(value rune) (*Node, error) {

  for _, node := range this.Children {

    if SameKey(node.Value, int32(value)) { // similaire a IsSameKey mais pour go
      return node, nil
    }
  }

  return this, errors.New("not found")
}
```

Une fois les fonctions d'extraction écrites, nous avons mis en place nos fonctions de vérifications :

```
    func CheckWord(w string,root *Node) bool {
  next := root
  if len(w) < 3 {
    return false
  }

  for _, l := range w { // pour chaque lettre du mot

    if temp, err := next.GetChild(l); err == nil { // si on a un enfant qui correspond a
        la lettre en cours
      next = temp //l'enfant est observe
    } else { //chaine non finie et pas d'enfant possible donc pas un mot du dictionnaire
      return false
    }
  }

  return EndOfWord(next.Value) // verifie si le noeud terminal forme un mot
}
```

Avant d'exposer en détail l'algorithme de recherche, justifions le choix d'utiliser un arbre comme structure de données. Bien que la tentation d'utiliser une table de hachage pour vérifier la présence d'un mot dans le dictionnaire puisse surgir, cela ne se traduit pas par une amélioration de performance significative par rapport à l'utilisation d'un arbre.

La fonction de hachage appliquée à une clé de type "string", comme celle que l'on retrouve dans le code source de Golang, montre que pour les chaînes de caractères de taille réduite, le parcours de la chaîne et la descente le long des branches de l'arbre sont comparables en termes d'opérations nécessaires à la génération d'une clé de hachage à partir d'une chaîne de caractères.

Cependant, nous constaterons que l'utilisation d'un arbre offre un véritable avantage en terme de performance lors de la recherche de tous les mots d'une grille, en évitant d'explorer toutes les combinaisons possibles de 16 lettres.

Recherche de tous les mots

Dans cette section, nous cherchons un moyen de trouver l'ensemble des mots réalisables dans une grille donnée en utilisant uniquement l'association de lettres adjacentes.

En ayant cette problématique en tête, la mise en place d'un algorithme de type retour sur trace semble être cohérent afin d'explorer toutes les possibilités possibles.

Nous avons mis en place la fonction "appendFromPoint" pour rétroagir sur une trace où le choix de la prochaine lettre se fait parmi les voisins du point de coordonnées (i, j). Lorsqu'une coordonnée (i, j) est valide, c'est-à-dire qu'elle n'a pas été utilisée et qu'il existe un enfant dont la clé correspond à la valeur de la grille au même point de coordonnées, nous transmettons uniquement à l'appel récursif le sous-arbre dont l'enfant a une clé correspondant à la valeur du choix, comme illustré ci-dessous :

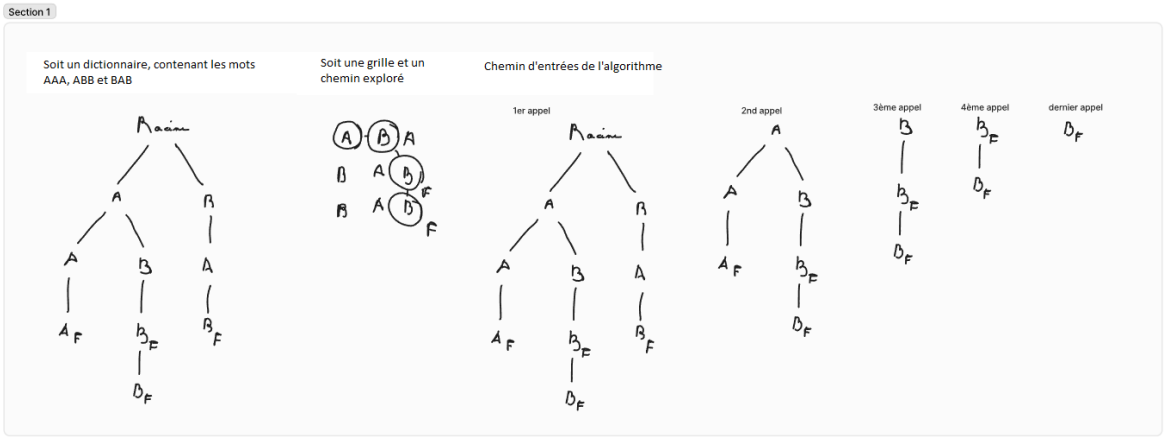


FIGURE 6 – Représentation de l'évolution de l'arbre utilisé par la fonction appendFromPoint

On appelle la fonction "appendFromPoint" pour chaque point de la grille et ainsi on obtient l'ensemble des mots de la grille. La complexité dans le pire des cas pour cet algorithme n'est pas très pertinente. Étant donné qu'il s'agit d'un backtrack, elle devrait être de l'ordre de $O(c^n)$ avec $c \geq 2$. Cependant, en moyenne, on trouve 87 mots avec un écart-type de 50 sur un échantillon de 100 000 grilles. Si l'on considère qu'en moyenne, il faut 10 appels récursifs par mot trouvé, tout en sachant que la plupart des mots ont une longueur de 3 à 5 lettres et que l'on prends en compte des appels supplémentaires pour des chaînes qui ne peuvent pas être complétées en un mot. Nous avons donc alors une complexité moyenne d'environ 870 pour une grille de taille 16. Ceci est bien inférieur à c^{16}.

3.3 Implémentation du système de parties multijoueurs avec FireBase

Dans cette partie de l'application, il est question de permettre aux joueurs de créer des parties en ligne et de jouer avec d'autres joueurs. Dans ces parties, le score des joueurs est mis à jour en temps réel et les joueurs peuvent voir les scores des autres joueurs. Enfin, il est nécessaire de sécuriser les envois de données afin d'éviter par exemple qu'un joueur ne triche.

Conception de la base de données en temps réel

Afin de mettre en place cette fonctionnalité, nous disposions de la base de données en temps réel de Firebase dans laquelle nous créons et mettons à jour les parties. Nous avons donc mis en place une architecture client-serveur pour gérer les parties en ligne.

Le serveur gère les connexions des joueurs et les parties, et est la seule entité autorisée à écrire dans la base de données en temps réel. Les joueurs envoient des requêtes telles que 'créer une partie', 'rejoindre une partie', 'envoyer un mot' au serveur qui les traite, met à jour la base de données en conséquence et envoie une réponse sous forme d'un code de retour. Enfin, chaque client dispose d'un accès en lecture seule à la base de données pour afficher les scores des autres joueurs à chaque fois qu'elle est modifiée.

Voici un schéma résumant l'architecture mise en place (voir annexe pour le schéma complet) :

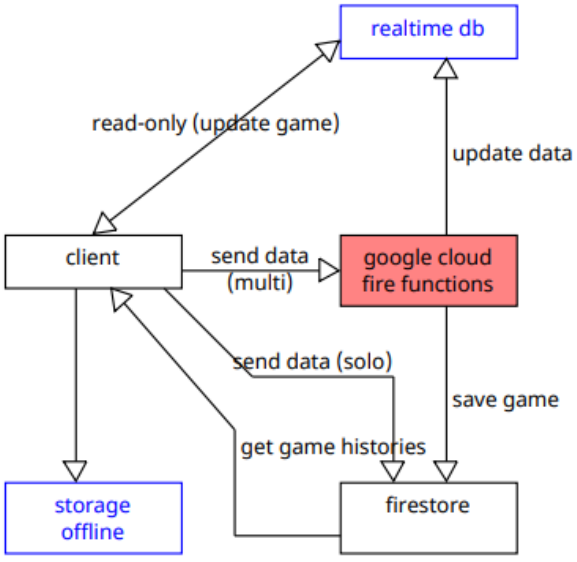


FIGURE 7 – Représentation de l'architecture client-serveur

Pour créer l'architecture associée à nos besoins, nous utilisons 3 bases de données fournies par Firebase pour ce projet, **Firestore Database**, **Realtime Database** et **Storage**.

Firestore Database est notre base de données principale. C'est une base de données NoSQL qui stocke des données pouvant être représentées sous la forme d'un objet JSON. Elle nous permet de stocker les informations liées à l'utilisateur en jeu (historique de partie, points totaux gagnés...). Cette même base de données sera également utilisée pour stocker toutes les données liées au profil de l'utilisateur. Elle s'organise comme une liste de collections dans lesquelles chaque valeur est associée à une clé.

La seconde base de données, la Realtime Database, nous permet de synchroniser les joueurs d'une même partie. Tout comme la base de données précédente, c'est une base de données NoSQL, mais qui ne contient qu'un seul objet JSON sur lequel il est possible d'observer les changements des différents champs.

Le Frontend de l'application l'observe donc afin de mettre à jour les différentes d'information sur la partie. A fin de faciliter le développement du multijoueur qui a été divisé en 2 entre l'aspect backend et frontend nous avons fait une description de l'objet JSON contenu dans la Realtime Database.

Voici comment la Realtime database est structurée :

```
{
    {
    "player_ingame" : {
        "[id]" : "[game_id] @string"
    },
    "games" : {
        "[game_id]" : {
            "letters" : "@string",
            "startedAt" : "@date",
            "end_time" : "@date",
            "lang" : "@int",
            "players" : {
                "[id]" : {
                    "(name)" : "@string",
                    "(email)" : "@string",
                    "score" : "@int",
                    "leader" : "@bool"
                }
            },
            "status" : "@enumGameStatus"
        }
    }
}
```

Les champs entre crochets correspondent à des données qui sont évolutives, ceux entre parenthèses des données qui sont optionnelles dans l'objet et pour finir la présence d'un "@" indique qu'il s'agit du type de la donnée.
Ainsi avec une visionneuse JSON, on obtient le schéma suivant.

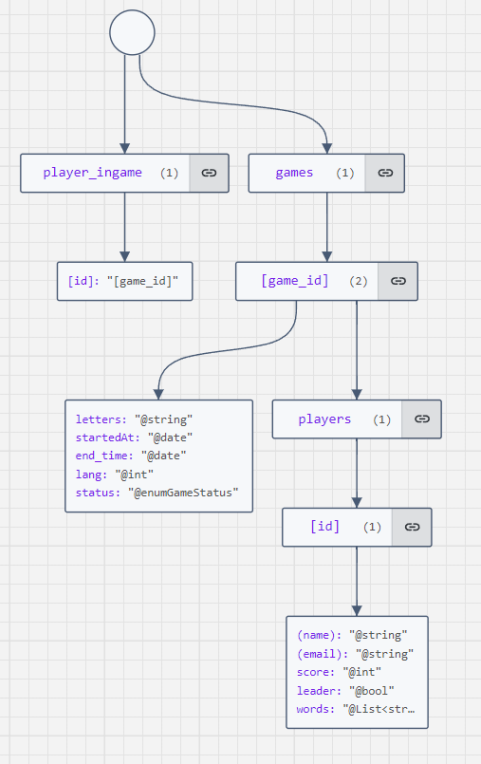


FIGURE 8 – Représentation de la Realtime database

Pour finir nous utilisons un système de stockage de fichiers dans le cloud pour enregistrer les fichiers les plus volumineux comme les dictionnaires.

L'objet game

Comme spécifié dans la structure, une partie a la liste des lettres utilisées dans la partie afin de permettre au serveur de vérifier que le mot envoyé par un client est bien un mot qu'il est possible de créer avec cette grille de Boggle. En reconstruisant la grille à partir de cette liste, le serveur peut vérifier que le mot envoyé sous forme de coordonnées est bien un mot valide.

Le statut de la partie permet de savoir dans quel état elle est. Les états sont utilisés par le serveur et le client pour diverses vérifications, par exemple, un joueur ne peut pas rejoindre une partie qui a déjà commencé.

Les états possibles sont définis dans une énumération :

```
1 export enum GameState {
2     InProgress = 0,
3     Finished = 1,
4     Cancelled = 2,
5     NotStarted = 3,
6 }
```

L'attribut lang permet de savoir dans quelle langue la partie a été créée, et donc quel dictionnaire utiliser pour vérifier les mots. Elle est initialisée à la valeur de la langue choisie par le joueur qui crée la partie, et sert à la vérification de mots côté serveur.

Enfin, startedAt et end_time permettent de savoir quand la partie a commencé et quand elle doit se terminer. Ces attributs sont utilisés pour vérifier si la partie est terminée ou non, afin de valider des mots uniquement si la partie est en cours ou bien pour synchroniser la durée restante de la partie des clients.

L'objet player

Chaque joueur a un score, un nom, un email, et une liste de mots. Le score est mis à jour en temps réel par le serveur, car bien qu'il soit calculable à partir des mots envoyés par le joueur, il est moins coûteux de le mettre à jour directement après avoir vérifié un mot en y ajoutant la valeur du mot.

L'attribut leader permet de savoir si le joueur est le créateur de la partie, et donc s'il a le droit de la démarrer ou de l'annuler. Il est transféré à un autre joueur si le créateur quitte la partie.

Words est une liste de mots valides envoyés par le joueur. Les mots sont envoyés par le client sous forme de liste de coordonnées au sein de la grille afin de vérifier l'appartenance du mot à celle-ci comme énoncé précédemment. Une fois le mot reconstruit et vérifié par le serveur, il est ajouté à la liste des mots du joueur sous sa forme chaîne de caractères et son score est mis à jour.

La liste player_ingame

Cet objet permet de contrôler rapidement les accès aux parties, chaque joueur ne pouvant être dans une seule partie à la fois. Il lie un identifiant de joueur à un identifiant de partie, ce qui permet de vérifier rapidement si un joueur est déjà dans une partie avant de lui permettre de rejoindre une autre partie.

Implémentation du serveur

Côte serveur, un serveur Google Cloud Functions est proposé par Google Cloud, offrant un support direct avec Firebase. Nous avons choisi de l'implémenter en Typescript et nous y avons donc déployé les fonctions appelées par le client pour gérer les parties en ligne.

Ce serveur dispose d'un routeur qui appelle la bonne fonction pour chaque requête spécifiée :

```
1     export const CreateGame = onCall(create_game);
2     export const StartGame = onCall(start_game);
3     export const JoinGame = onCall(join_game);
4     export const LeaveGame = onCall(leave_game);
5     export const CancelGame = onCall(cancel_game);
6     export const SendWord = onCall(check_word(dictionariesHandler, DictionaryConfig));
```

Ces fonctions retournent un objet contenant un code de retour et un message d'erreur si nécessaire. Voici par exemple la liste des codes de retour possible pour la fonction JoinGame :

```
1     export enum JoinGameReturn {
2     SUCCESS = 0,
3     GAME_FULL = 1,
4     GAME_STARTED = 2,
5     GAME_NOT_FOUND = 3,
6     ALREADY_IN_GAME = 4,
7     INVALID_PASSWORD = 5,
8     WRONG_GAME_ID = 6
9 }
```

Enfin, pour gérer les éventuelles parties qui seraient toujours dans la base de données mais inaccessibles (par exemple quand les joueurs d'une partie quittent brusquement l'application), nous avons mis en place un système de nettoyage des parties inactives à l'aide d'un cron job qui les supprime tous les jours à

5h du matin (un cron job est une exécution périodique automatique d'un script dans un environnement Unix).

Ajout de la feature au client

Côté client, afin de communiquer avec le serveur, nous utilisons la fonction httpsCallable fournie par Firebase qui permet d'envoyer des requêtes au serveur. Par exemple, pour demander au serveur de rejoindre une partie, il suffit de faire :

```
1   response = await Firebase.functions().httpsCallable('JoinGame')({userId: userid,
        gameId: idgame, email: email, name: name});
```

En fonction du code de retour donné par le serveur, nous pouvons afficher un message d'erreur ou rediriger l'utilisateur vers la partie. Concernant la vérification des mots, bien qu'elle soit effectuée côté client, elle est également vérifiée côté serveur pour éviter toute triche. Nous gardons les deux vérifications pour des raisons de performance, la vérification côté client permet de trier les mots valides des mots invalides avant de les envoyer au serveur, ce qui permet de réduire le nombre de requêtes au serveur.

Pour mettre à jour les scores, la librairie Firebase fournit des outils pour écouter les modifications d'une base de données en temps réel. Afin d'écouter la partie qui nous intéresse, il suffit de spécifier le chemin de la ressource voulue. Par exemple, pour écouter les modifications de la partie avec l'identifiant 'idgame', il suffit de faire :

```
1     Firebase.database().ref('game/' + idgame).on('value', (snapshot) => {
2     // mise a jour des scores, synchronisation de la duree restante de la partie, etc.
3 });
```

En fin de partie, l'état de la partie retourné par la fonction d'écoute de la base de données est sauvegardé dans une variable afin d'afficher les détails de la partie au joueur pendant tout le temps qu'il souhaite les consulter, même si la partie est supprimée de la base de données en temps réel par le serveur car elle est finie.

3.4 Description de l'interface graphique

Page home

La première page sur laquelle l'utilisateur arrive est la page **Home**. La page permet notamment de rediriger le joueur vers :

— les règles du jeu en cliquant sur le bouton lire.
— une partie solo en cliquant sur Partie Solo.
— une partie multijoueur en cliquant sur Partie multijoueurs.
— la page de connexion en cliquant sur Connexion.
— les options en cliquant sur les rouages dans le menu du bas.
— les statistiques en cliquant sur l'icône de la courbe.

FIGURE 9 – Capture d'écran de la page Home

Page de jeu

En cliquant sur partie solo, le joueur va pouvoir lancer une partie en cliquant sur un bouton ou bien en secouant son téléphone (grâce à l'accéléromètre programmé). Une fois la partie solo lancée, le joueur se retrouve sur cette page :

FIGURE 10 – Capture d'écran de la grille de jeu

C'est sur cette page que le joueur va chercher des mots pour gagner des points. Depuis cette page, le joueur peut :

- faire pause à tout moment en cliquant sur l'icone pause.
- afficher les détails avancées de la partie sur la page pause.
- sélectionner sur l'écran tactile les différentes lettres pour former les mots.
- obtenir un indice d'un mot en cliquant sur l'icone " ?", ce qui lui affichera en couleur la première lettre de ce dernier.
- observer le score obtenu dans la partie en cours.
- voir son nombre d'erreurs.
- voir le nombre total de mots restants.
- voir la longueur du mot le plus long restant.

Page détails

Si le joueur décide d'aller sur la page détails, il pourra alors prendre connaissance de plusieurs éléments :

— il aura la possibilité de consulter toutes les combinaisons de lettres possibles.
— pour chaque mot trouvable, son ordre d'éxécution possible représenté par un chemin de couleur (couleurs chaude pour le début du mot vers des couleurs froide pour la fin du mot).
— si l'utilisateur souhaite connaitre la définition d'un mot, il pourra également cliquer sur l'icône de dictionnaire afin d'être redirigé vers sa page Wikipedia.

FIGURE 11 – Capture d'écran de la page détail

Page de connexion

Pour se connecter aux services en ligne afin de sauvegarder ses données ou encore de rejoindre des parties multijoueurs, le joueur doit passer par la page de connexion suivante :

FIGURE 12 – Capture d'écran de la page de connexion

Partie mutltijoueurs

Pour les parties mutltijoueurs, il suffit de rejoindre une partie en insérant le code d'une partie pour arriver dans une session en ligne, le joueur va alors arriver sur une page de lobby en ligne :

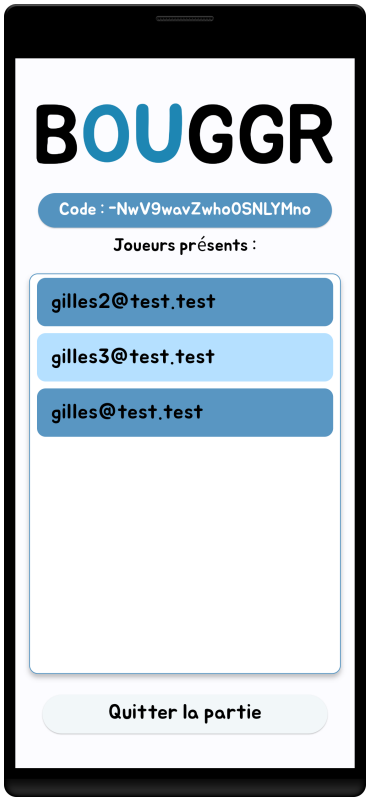


FIGURE 13 – Capture d'écran du lobby d'attente

Nous avons deux cas de figure :

— soit l'utilisateur est l'hôte de la partie, il peut alors lancer la partie dès qu'il le souhaite.
— soit l'utilisateur a rejoint la partie, il peut alors attendre que le lancement de la part de l'hôte soit lancé.

Voici à quoi ressemble une partie multijoueurs en temps réel :

FIGURE 14 – Capture d'écran d'une partie en ligne

Le haut de la fenêtre est réservé aux informations concernant le top 3 des joueurs de la partie en cours :

— la position du joueur.
— le nombre de points obtenus.
— l'adresse mail du joueur.

4 Gestion du Projet

4.1 Planification des tâches

Afin d'avancer efficacement dans notre projet, nous avons décidé de travailler en méthode agile (Scrum). Cette méthode consiste à diviser le projet en itérations appelées "sprints" de courte durée, généralement de deux à quatre semaines, au cours desquelles l'équipe se concentre sur la livraison de fonctionnalités prioritaires. Cette méthode permet d'apporter des changements fréquemment tout en ayant un retour sur le travail effectué dans les différents sprints. Dans la méthode Agile Scrum, plusieurs rôles clé sont associés. Le Product Owner est chargé de représenter les besoins des clients (rôle attribué à Jérémy) et de prioriser le travail à réaliser. Le Scrum Master (rôle associé à Vincent) assure le bon déroulement du processus Scrum en aidant l'équipe à comprendre et à adopter les principes et les pratiques, en éliminant les obstacles et en facilitant les événements Scrum tels que les réunions de planification, les Daily Scrums, les revues de sprint et les rétrospectives. Ensemble, ces rôles garantissent une collaboration efficace et une livraison régulière de valeur tout au long du projet.

Pour ce faire, nous avons segmenté le projet en plusieurs tâches à réaliser, ce qui nous a permis d'avoir un avancement plutôt rapide. En effet, nous avons priorisé le fait d'avoir une application fonctionnelle en développant simultanément le back-end et le front-end. Nous avons également priorisé les tâches demandées par nos tuteurs. Par la suite, nous nous sommes organisés en sprints d'une semaine, contenant plusieurs tâches à effectuer. Chacun devait choisir sa ou ses tâches à réaliser pour le sprint. Nous nous organisions aussi en moyenne deux réunions pour suivre l'avancement de chacun durant le sprint. Ces réunions pouvaient s'effectuer en présentiel ou bien sur Discord. Grâce à la flexibilité de cette méthode, si nous étions bloqués sur une tâche à effectuer, nous pouvions soit demander de l'aide à une personne qui avait terminé ses tâches, soit changer de tâche le temps de trouver une solution, sans poser de problème d'organisation.

Voici un exemple de sprint :

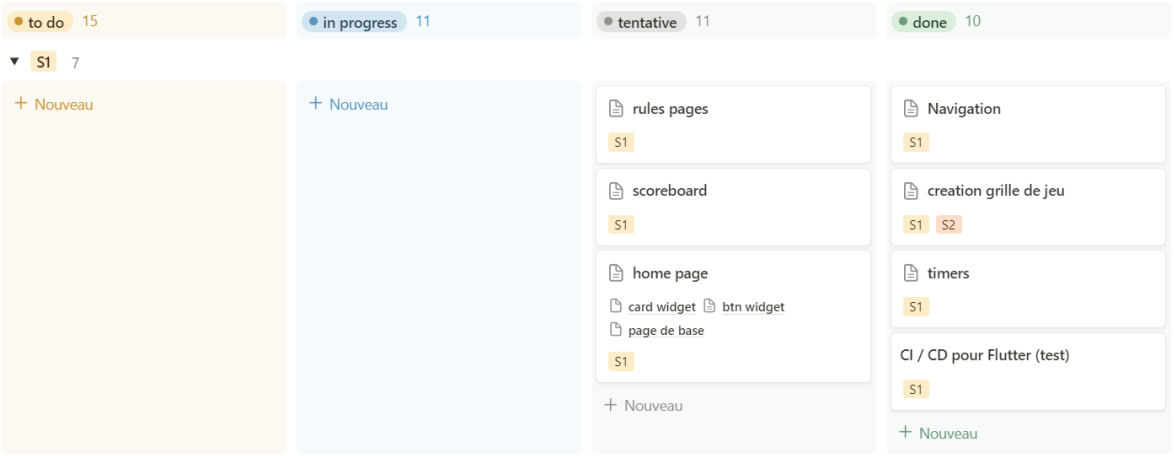


FIGURE 15 – Exemple d'un sprint

Tout d'abord, toutes les tâches se trouvent dans « To Do ». Une fois qu'une tâche est affectée à une personne, elle passe dans la colonne « In Progress ». Lorsque la personne pense avoir terminé la tâche, elle passe en « Tentative ». Par la suite, toutes les tâches se trouvant en « Tentative » finiront dans la colonne « Done » une fois que le Product Owner aura constaté que la tâche répond bien aux spécifications demandées, et qu'aucun changement n'est à prévoir pour cette tâche.

Voici à quoi ressemble une tâche :

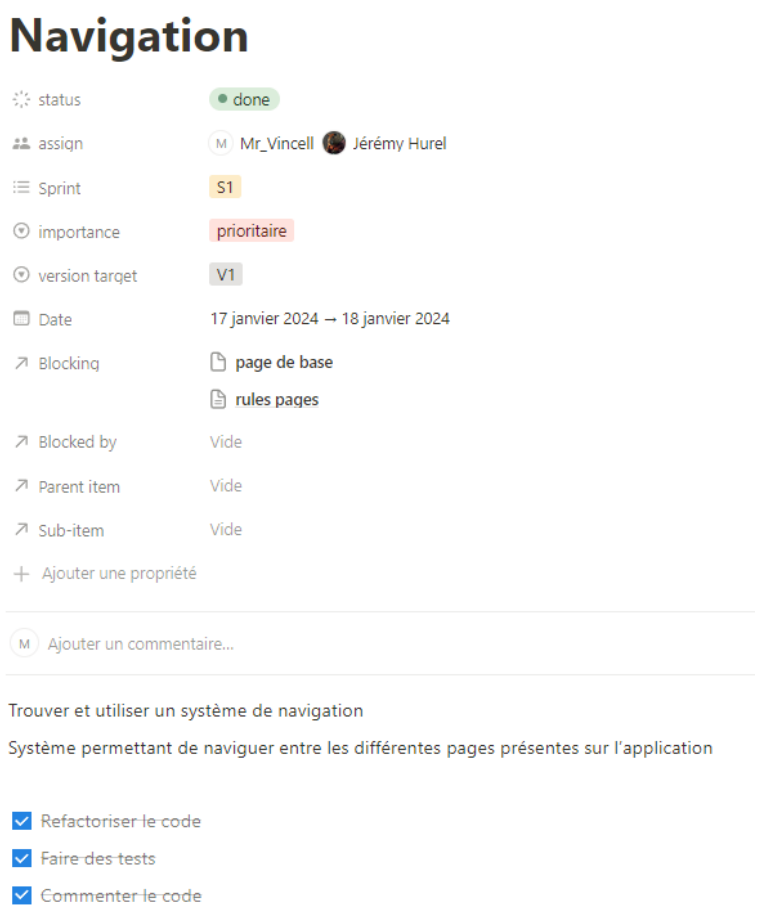


FIGURE 16 – Capture d'écran d'une tâche

Chaque tâche contient plusieurs informations :

— nous pouvons voir le statut de la tâche en question.
— les personnes assignées à cette tâche.
— le sprint associé à la tâche.
— le degré d'importance de la tâche.
— la date de début et de fin prévues de la tâche.
— un descriptif afin de détailler le travail qui doit être réalisé.
— les points majeurs de la tâche.

5 Bilan et Conclusions

5.1 Problèmes rencontrés

Dans cette section, nous aborderons les défis techniques rencontrés durant le projet. Premièrement, la difficulté d'établir une connexion avec les comptes Google. Deuxièmement, nous discuterons du problème de typage en Dart, mais aussi du ramasse-miettes de Goland. Enfin, le débogage des fonctions backend sur Google Cloud s'est avéré complexe.

Lors de la tentative de connexion, nous avons rencontré un problème où l'identification ne se réalisait pas avec des comptes Google. Bien que ce problème n'ait pas été considéré comme une priorité en raison de la présence d'une authentification par e-mail, nous avons néanmoins identifié plusieurs pistes à explorer pour déterminer l'origine du problème. Lors de la mise en œuvre de cette tâche, nous avons constaté que le code était conforme à la documentation de Firebase pour la création de comptes à partir d'un fournisseur d'authentification, et qu'il fonctionnait dans l'application. Cependant, une fois le compte Google sélectionné, l'application ne se connectait à aucun compte et affichait une erreur dans les logs, indiquant que Firebase n'était pas correctement connecté à notre application, malgré sa configuration correcte. Pour comprendre le problème, deux pistes ont été envisagées : la première hypothèse serait que, peut-être, l'application utilisée lors des tests n'était pas considérée comme terminée et était exécutée en mode de test, ce qui aurait pu entraîner une restriction empêchant la connexion avec un compte Google. La seconde possibilité serait de mettre en place une connexion de compte via un autre fournisseur d'authentification, tel que Microsoft, Facebook ou Github, afin de vérifier si le problème est lié aux paramètres de Firebase ou à un problème spécifique à l'authentification Google.

Passons donc maintenant au second problème. Lors de l'intégration de l'algorithme de recherche de tous les mots d'une grille, nous nous sommes heurtés à certaines limites du design du langage Dart. Si celui-ci garantit de bonnes performances natives, il n'est pas conçu pour manipuler des données encodées en binaire. En effet, il n'est pas possible de changer le type des variables comme en C, et il n'existe pas réellement de type char. Cependant, il est possible d'utiliser un autre langage afin de faire ces opérations, et nous avons donc décidé d'utiliser Golang. Une fois l'étape de mise en place de la fonction de recherche de tous les mots, c'est aux contraintes des langages avec un ramasse-miettes que nous nous sommes heurtés. Lors de l'appel à un fonction dite native elle s'exécute et envoie soit une valeur soit un pointeur quand il s'agit d'un tableau ou d'une liste. Les pointeurs de Golang étant attachés au ramasse-miettes de celui-ci, une fois que le code Golang a fini de s'exécuter, la donnée est supprimée. Quelques tentatives pour résoudre le problème ont été faites en passant en mode unsafe de Golang, mais trop de valeur étaient liées à des pointeurs ce qui rendait la tâche difficile et le code peu compréhensible. Au final, une collection de fonctions de calcul sur les bits a été gardée en Golang, mais pas l'algorithme de recherche.

Pour conclure cette section, abordons maintenant le problème de débogage du mode multijoueur de l'application, dû à l'utilisation de Google Cloud. Ce problème est principalement lié aux logs générés par Google Cloud, qui ne sont pas clairs pour nous. La mise en place de tests unitaires a été la clé pour résoudre tous les problèmes liés au mode multijoueur au niveau du serveur backend.

5.2 Bilan du travail réalisé

Dans le cadre du développement d'une application mobile, notre choix s'est porté sur la création du jeu "Boggle". Ce projet nous a plongé entièrement dans le processus de développement d'une application mobile, mettant en évidence à la fois les difficultés techniques et les compétences organisationnelles requises pour réussir un tel projet. Au fil de cette expérience, plusieurs aspects ont émergé, illustrant à la fois nos succès et les pistes d'amélioration à explorer.

Nous avons commencé par élaborer un algorithme performant pour trouver les mots dans la grille, en fusionnant nos connaissances sur les structures de données et l'optimisation pour assurer un fonctionnement fluide. Ensuite, l'adoption de Flutter et Firebase nous a permis de nous familiariser avec un nouveau langage et de gérer les données en ligne de manière efficace.

La collaboration étroite au sein de l'équipe a été essentielle pour surmonter les obstacles rencontrés. Une répartition claire des tâches et une communication régulière ont soutenu notre progression. Parallèlement, le projet nous a exposé à différents aspects de la création d'une application mobile, enrichissant nos compétences techniques et notre compréhension des processus impliqués. En résumé, le projet "Boggle" a été une expérience formatrice, nous permettant d'acquérir de nouvelles compétences tout en nous confrontant aux défis du développement d'une application mobile.

Voici le lien vers la dernière version générée par la pipeline, que vous pouvez utiliser pour tester l'application si vous le souhaitez : `https://github.com/Akkuun/TER_L3_2023_Boggle_Mobile/releases/tag/v1.0.85`

5.3 Ouvertures possibles

Dans l'état actuel du projet, celui-ci n'est pas encore prêt pour être mis en production. Afin de finaliser celui-ci, il serait déjà nécessaire d'ajouter toujours une partie de mentions légales, car nous, et les fournisseurs d'authentification, récoltons un certain nombre de données sur les utilisateurs. Aussi, nous avions planifié au début du projet un certain nombre de fonctionnalités dans l'optique de faire un projet viable si celui-ci était amené à être mis sur le Play Store. Parmi celles-ci, on retrouve un système d'amis et de matchmaking pour améliorer l'expérience utilisateur en multijoueurs, mais aussi un système de grille du jour afin d'augmenter la rétention des utilisateurs et bien sûr l'intégration de pubs pour être à l'équilibre au niveau des coûts d'hébergement.

Bibliographie

Références

[1] Wikipedia (2022) *Boogle*, Wikipedia `https://fr.wikipedia.org/wiki/Boggle`, 29-06-2022.

[2] Flutter documentation | Flutter. Consulté le : 10 décembre 2023. [En ligne]. Disponible sur : `https://docs.flutter.dev/`

[3] Flutter Casual Games Toolkit. Consulté le : 16 décembre 2023. [En ligne]. Disponible sur : `https://flutter.dev/games/`

[4] Introduction to Dart. Consulté le : 16 décembre 2023. [En ligne]. Disponible sur : `https://dart.dev/language/`

[5] Getting Started — Flame. Consulté le : 23 décembre 2023. [En ligne]. Disponible sur : `https://docs.flame-engine.org/latest/`

[6] Ajouter Firebase à votre application Flutter. Consulté le : 18 janvier 2024. [En ligne]. Disponible sur : `https://firebase.google.com/docs/flutter/setup?hl=fr`

[7] flutter_golang_ffi_example/src/Makefile at main · leehack/flutter_golang_ffi_example, GitHub. Consulté le : 27 avril 2024. [En ligne]. Disponible sur : `https://github.com/leehack/flutter_golang_ffi_example/blob/main/src/Makefile`

[8] go/src/hash/maphash/maphash_runtime.go at master · golang/go, GitHub. Consulté le : 27 avril 2024. [En ligne]. Disponible sur : `https://github.com/golang/go/blob/master/src/hash/maphash/maphash_runtime.go`

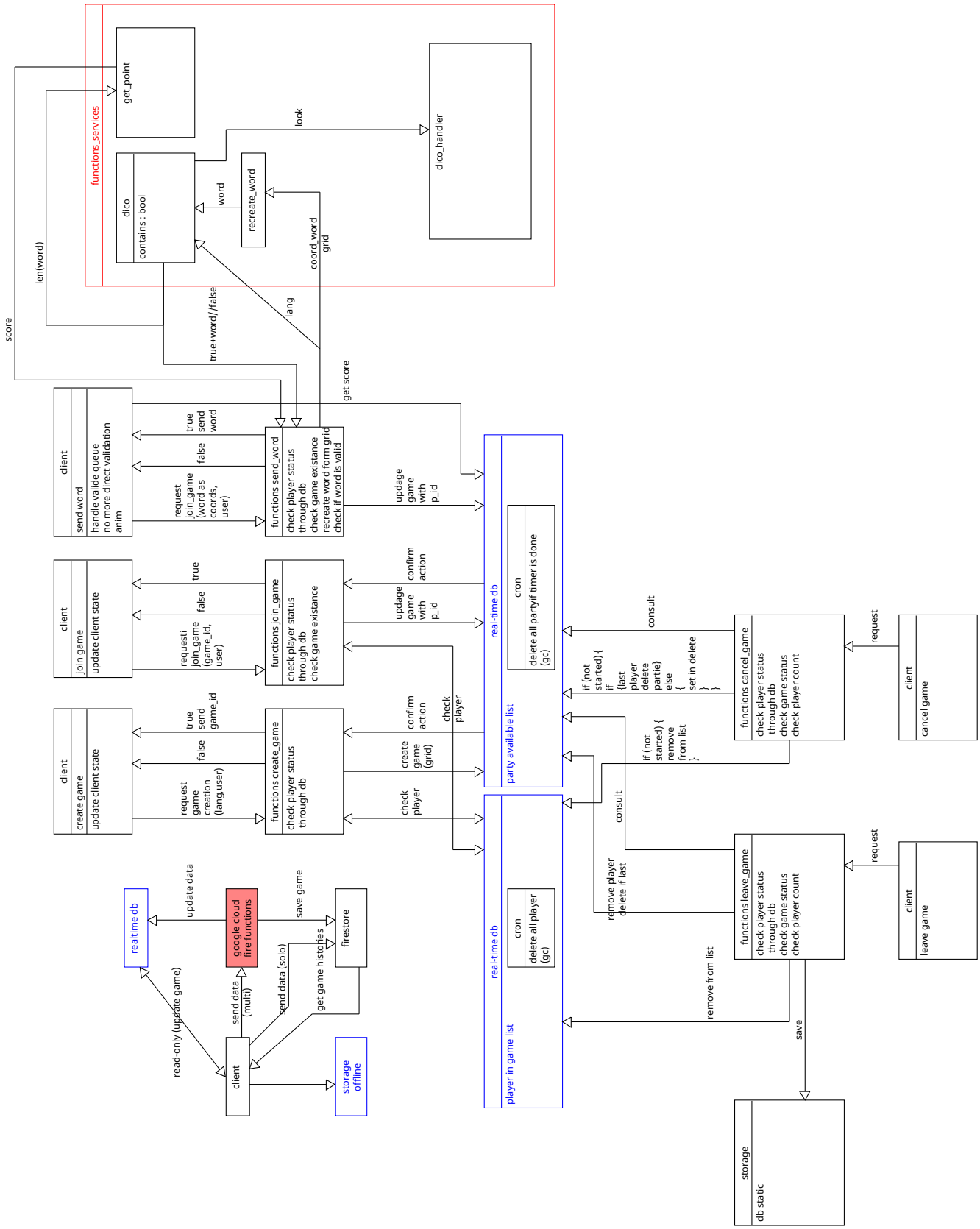
[9] K. Koller, [Flutter] Proper shake detection in Shaky Rock-Paper-Scissors, Medium. Consulté le : 2 février 2024. [En ligne]. Disponible sur : `https://onetdev.medium.com/flutter-shaky-rock-paper-scissors-a81508c9c375`

[10] Firebase Documentation. Consulté le : 27 avril 2024. [En ligne]. Disponible sur : `https://firebase.google.com/docs?hl=fr`

[11] Boggle.fr - Entraînez-vous au jeu de lettres ! Consulté le : 27 avril 2024. [En ligne]. Disponible sur : `https://boggle.fr/`

[12] S. Adjatan, Thecoolsim/French-Scrabble-ODS8. 19 mars 2024. Consulté le : 27 avril 2024. [En ligne]. Disponible sur : `https://github.com/Thecoolsim/French-Scrabble-ODS8`

Annexes


functions_services
get_point
dico
contains : bool
recreate_word
dico_handler
look
word
coord_word
grid
len(word)
score
true+word//false
lang
get score
client
send word
handle valide queue
no more direct validation
anim
request
join_game
(word as
coords,
user)
false
true
send
word
functions send_word
check player status
through db
check game existance
recreate word form grid
check if word is valid
updage
game
with
p_id
client
join game
update client state
requesti
join_game
(game_id,
user)
functions join_game
check player status
through db
check game existance
confirm
action
real-time db
party available list
cron
delete all partyif timer is done
(gc)
client
create game
update client state
request
game
creation
(lang,user)
functions create_game
check player status
through db
true
send
game_id
create
game
(grid)
check
player
player in game list
cron
delete all player
(gc)
if (not
started) {
if
{last
player
delete
partie}
else
{
set in delete
}
}
consult
functions cancel_game
check player status
through db
check game status
check player count
request
client
cancel game
if (not
started) {
remove
from list
}
remove player
delete if last
functions leave_game
check player status
through db
check game status
check player count
remove from list
save
storage
db static
client
leave game
realtime db
update data
google cloud
fire functions
save game
firestore
read-only (update game)
send data
(multi)
send data (solo)
get game histories
client
storage
offline