Université de CY Cergy Paris Université

Projet CodYngame

Groupe 4 :
Hachem Fayçal
Thoyer Roman
Cottrant Axel
Langlais Thomas
Limousin Brice

Professeur référent : ROMUALD GRIGNON

Date de rendue : 26/05/2024

Année Universitaire 2023/2024

Sommaire :

1. Introduction :

1.1 Motivation :

Le projet de fin d'année constitue une étape incontournable dans le cursus du cycle ingénieur de la classe ING1-GI. Cette étape est obligatoire car elle permet d'acquérir une expérience précieuse en gestion de projet, d'exercer son autonomie, de relever des défis, de trouver des solutions à divers problèmes, d'envisager plusieurs options pour chaque cas d'usage, et de gérer la cohésion de groupe. Toutes ces compétences sont essentielles dans la formation d'un futur ingénieur.

Nous avons eu le choix entre quatre projets différents : Chromatik, CodIngame, CY-Books et Généalogie. Parmi ces options, le projet CodIngame a immédiatement retenu notre attention. L'idée de réaliser une plateforme permettant l'apprentissage de la programmation à travers un jeu nous a semblé particulièrement intéressante. En effet, ce type d'approche éducative s'est avéré très efficace et innovant par rapport aux méthodes traditionnelles. De plus, ce projet permet de créer un premier lien avec une des nombreuses facettes de l'informatique, en l'occurrence, le développement de jeux vidéo.

La réalisation de CodIngame présente non seulement un intérêt pédagogique, mais elle permet aussi de développer des compétences techniques variées, allant de la programmation à la conception d'interfaces utilisateur, en passant par la gestion de bases de données et l'intégration de divers langages de programmation. Ce projet est ainsi une excellente opportunité pour mettre en pratique nos connaissances et pour les enrichir dans un contexte concret et stimulant.

1.2 Description du projet CodIngame

L'objectif principal de CodIngame est de développer une plateforme graphique interactive permettant aux utilisateurs de s'exercer sur différents langages de programmation, tels que C, Java, Python, PHP, et JavaScript, à l'aide d'exercices pratiques. Ces exercices sont stockés dans une base de données, ce qui permet de les filtrer en fonction du langage de programmation et du mode souhaité (**STDIN/STDOUT** ou **INCLUDE**).

Fonctionnalités des exercices

Chaque exercice comprend :

- Énoncé détaillé : Un énoncé clair et précis permettant à l'utilisateur de comprendre le problème à résoudre.
- Code minimal : Un exemple de code minimal montrant comment lire une ligne depuis l'entrée standard et écrire sur la sortie standard (par exemple, scanf()/printf() en C, input()/print() en Python, fgets(STDIN)/echo() en PHP).
- Choix du langage de programmation : Les utilisateurs peuvent choisir le langage de programmation qu'ils souhaitent utiliser pour résoudre l'exercice.
- Zone de code avec coloration syntaxique et gestion des indentations : Un éditeur de code intégré avec des fonctionnalités avancées comme la coloration syntaxique et la gestion des indentations, facilitant l'écriture et la lecture du code.

Processus d'utilisation

Lors de la sélection d'un exercice, celui-ci s'affiche dans la fenêtre principale de l'application. L'utilisateur peut alors proposer une solution en écrivant son code dans l'éditeur intégré. Une fois le code prêt, l'utilisateur peut soumettre sa solution en cliquant sur un bouton dédié. L'application compile, exécute et interprète ensuite le code en fonction du langage de l'exercice en utilisant des appels système vers les différents compilateurs et interprètes appropriés.

Conclusion

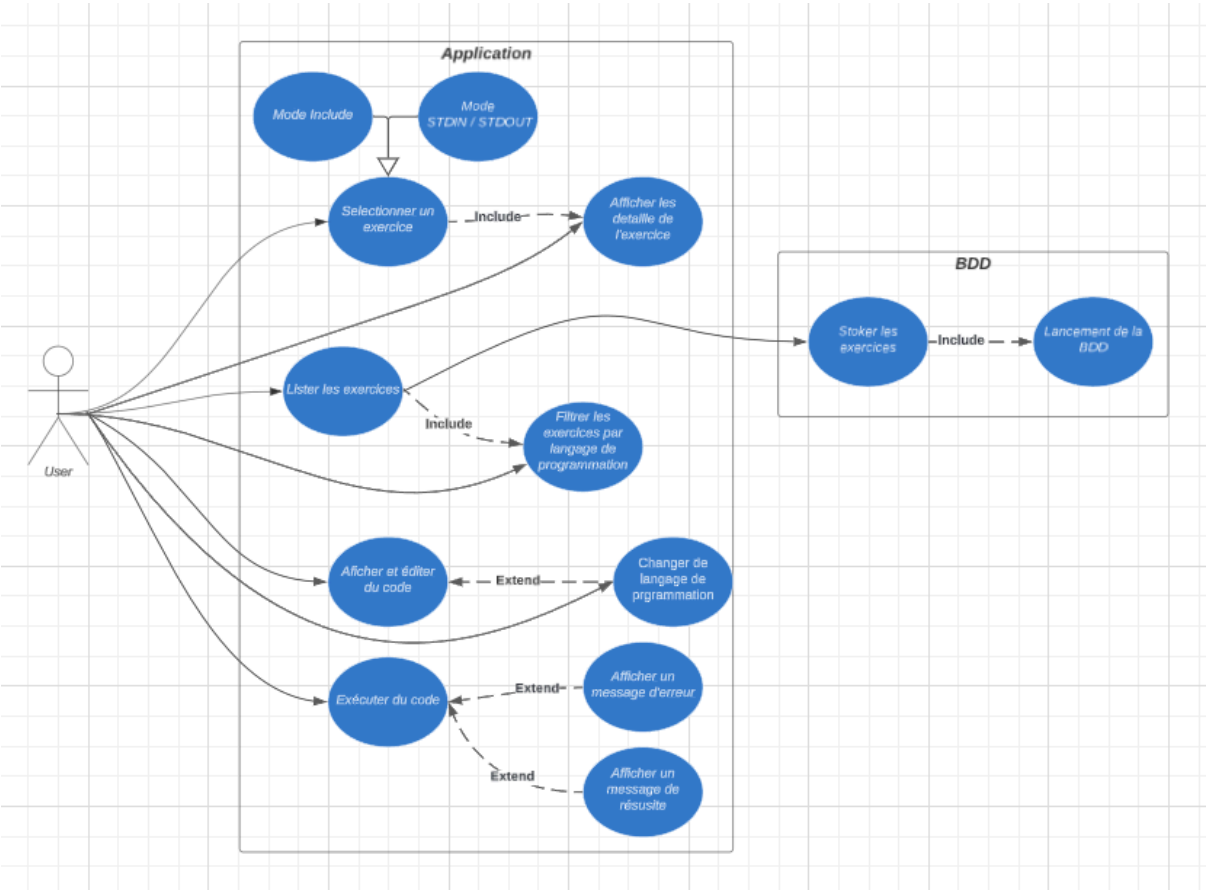
CodIngame offre ainsi une expérience d'apprentissage pratique et immersive, facilitant l'amélioration des compétences en programmation des utilisateurs à travers une interface graphique intuitive et des exercices structurés et variés. Cette approche permet aux utilisateurs de se familiariser avec les concepts de base et avancés de différents langages de programmation, tout en offrant un retour immédiat sur leurs progrès et performances. Le projet met également un accent particulier sur la gestion efficace des appels système vers les compilateurs et interprètes, ainsi que sur la gestion et l'organisation de la base de données des exercices disponibles.

2. Méthodologie de développement :

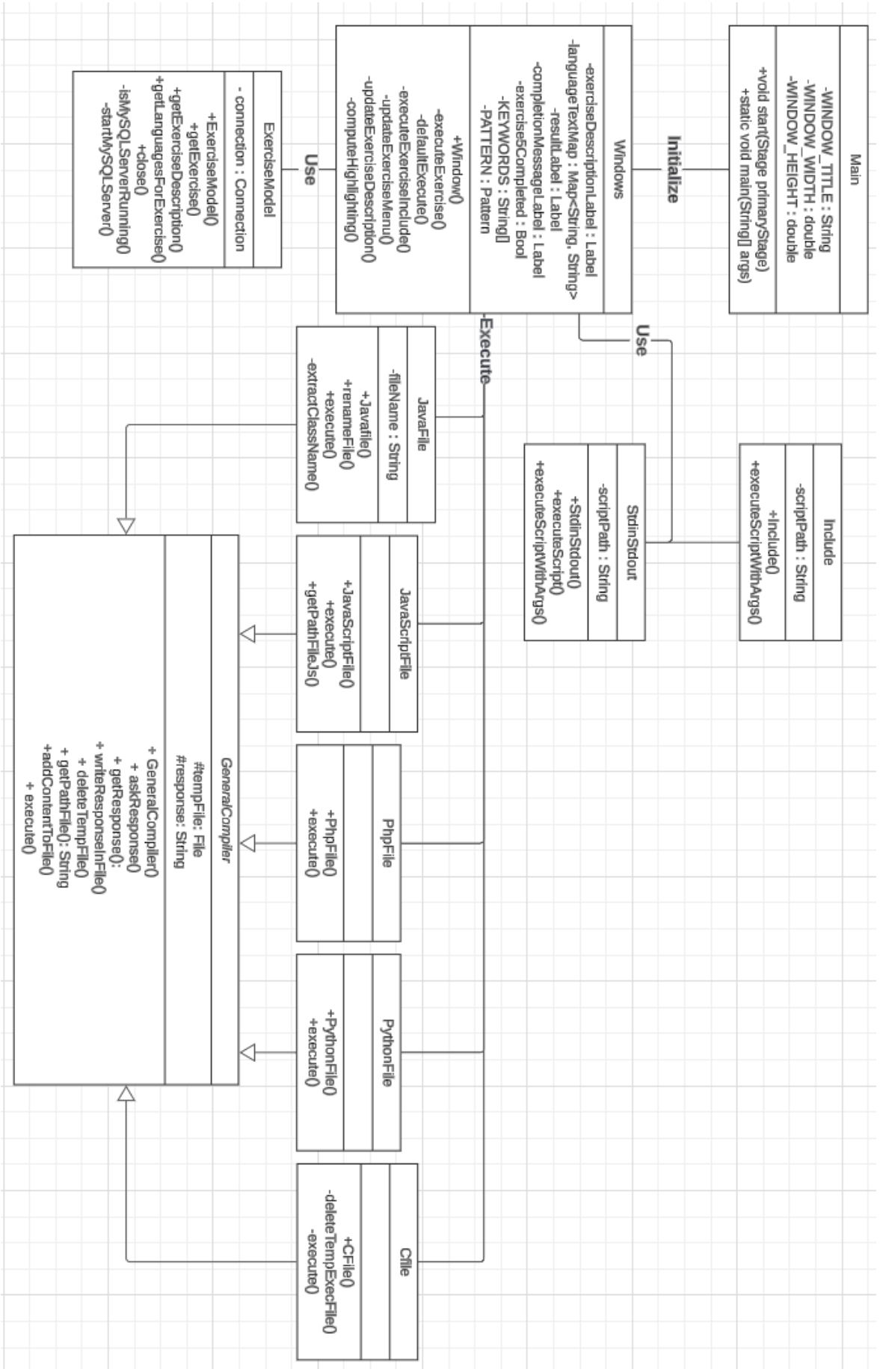
2.1 Organisation :

Durant la première partie du projet, l'organisation a été principalement effectuée de manière orale, basée sur une réunion quotidienne au cours de laquelle un point est fait sur le travail accompli et celui à venir. Par la suite, nous nous assignons chacun une tâche à réaliser, seul ou en groupe, en fonction de sa complexité et de sa durée. Nous restons ensuite tous connectés sur le même salon Discord, ce qui permet d'interagir directement avec tous les membres du groupe en cas de besoin. De plus, la possibilité de partager nos écrans et d'utiliser un dépôt de code communautaire sur IntelliJ facilite la mise en commun de nos travaux et l'assistance mutuelle en cas de nécessité. À la suite d'une réunion, nous avons amélioré notre système d'organisation en utilisant Google Agenda, ce qui nous a permis de mieux visualiser toutes les tâches en cours, celles déjà réalisées, ainsi que de savoir qui travaille sur quoi.

2.2 Diagramme Use Case:



2.3 Diagramme de Classe:



3. Difficultées rencontrées :

3.1 Mode STDIN/STDOUT

Lors de la mise en place du mode **STDIN/STDOUT** dans notre projet, nous avons rencontré plusieurs difficultés courantes et spécifiques que nous allons expliquer en détail, en les liant à nos notes et aux défis généraux observés dans ce contexte.

1. Compréhension et Orientation Initiale

Difficulté : Compréhension du Sujet et Orientation

Explication : Au début, comprendre les concepts fondamentaux du mode **STDIN/STDOUT** était difficile. Cette incompréhension initiale a conduit à une mauvaise orientation dans la mise en place du projet. Les termes et les mécanismes sous-jacents n'étaient pas clairs, ce qui a retardé le développement et causé de la frustration. En conséquence, nous avons passé une semaine à essayer de comprendre comment les données étaient lues et écrites via **STDIN/STDOUT**, ce qui a considérablement ralenti le progrès global du projet.

Solution : Pour surmonter ce défi, nous avons pris du recul pour étudier en profondeur les concepts du mode **STDIN/STDOUT**. Nous avons consulté des ressources supplémentaires, telles que des tutoriels et des documentations, pour clarifier les termes et les mécanismes sous-jacents. En outre, nous avons réalisé des exercices pratiques pour renforcer notre compréhension. Cette approche nous a permis de lever les obstacles et de progresser dans la mise en place du projet. En conséquence, nous avons pu reprendre un rythme de développement plus rapide et plus efficace.

2. Gestion des Flux de Données

Difficulté : Mettre la Sortie Standard d'un Programme en Entrée Standard d'un Autre

Explication : La redirection des flux de sortie standard d'un programme vers l'entrée standard d'un autre programme est une tâche courante mais complexe qui manque particulièrement de documentation sous Windows. Nous avons essayé de passer sous Linux sans plus de résultat. Par conséquent, la mise en place de cette redirection a été délicate.

Solution : Pour résoudre ce problème complexe de redirection des flux de sortie standard, nous avons finalement choisi de partir sur une méthode nécessitant trois

programmes : un programme qui génère des nombres aléatoires selon les besoins de l'exercice, un programme que l'utilisateur écrit dans la fenêtre de l'application et un programme qui résout l'exercice et retourne les données de sortie attendues. Pour garantir que chaque programme reçoive les données correctes, nous avons effectué des vérifications approfondies et des tests de validation à chaque étape du processus.

3.2 Appels système vers les différents compilateurs/interpréteurs

3.2.1 Langage C

Explication : Nous avons rencontré plusieurs problèmes avec le compilateur en C. Initialement, nous avons fait face à des erreurs de logique concernant l'ordre des opérations : il était nécessaire de compiler le code avant d'exécuter le fichier exécutable résultant. Cette erreur conceptuelle nous a obligé à revoir presque entièrement la méthode "execute" pour l'adapter correctement à ce flux de travail.La gestion des fichiers temporaires a également présenté des défis. Nous avons réalisé qu'il était nécessaire de supprimer deux fichiers temporaires (le fichier source et le fichier exécutable) au lieu d'un seul. Cette étape était cruciale pour éviter l'accumulation de fichiers inutiles et pour maintenir l'environnement de travail propre et ordonné.En ce qui concerne la gestion des erreurs, nous avons d'abord suivi une approche similaire à celle utilisée avec Python, en récupérant les erreurs d'exécution. Cependant, nous avons rapidement compris que, dans le contexte de la compilation en C, il était crucial de récupérer les erreurs de compilation plutôt que les erreurs d'exécution. À un certain moment, nous avions même mis en place un système pour récupérer à la fois les erreurs de compilation et d'exécution, ce qui n'avait pas de sens et ajoutait une complexité inutile à notre processus de gestion des erreurs.

Solution : Pour résoudre ces problèmes, nous avons dû procéder à plusieurs ajustements significatifs. Tout d'abord, nous avons revu notre méthode "execute" pour garantir que le code soit compilé avant d'être exécuté, ce qui a nécessité une révision approfondie de notre logique et de notre flux de travail.En ce qui concerne la gestion des fichiers temporaires, nous avons modifié notre approche pour inclure la suppression des deux fichiers temporaires nécessaires, assurant ainsi une gestion propre et efficace de ces fichiers dans notre système.Enfin, en ce qui concerne la gestion des erreurs, nous avons réajusté notre système pour ne récupérer que les erreurs de compilation, en abandonnant l'idée de récupérer simultanément les erreurs de compilation et d'exécution, ce qui simplifiait notre processus et le rendait plus efficace.Ces ajustements nous ont permis de résoudre efficacement les problèmes rencontrés avec le compilateur en C et d'améliorer considérablement la fiabilité et l'efficacité de notre processus de compilation et d'exécution.

3.2.2 Langage Java

Explication : :Au tout début du processus, lors de la compilation en ligne du code Java, nous avons rencontré un problème majeur : renommer le fichier était nécessaire, car si le nom de la classe ne correspondait pas exactement au nom du fichier, cela générait une erreur. Cela a engendré des complications, notamment lorsque l'utilisateur insérait un espace entre le nom de la classe et l'accolade ouvrante. Dans ces cas, notre méthode de récupération du nom de fichier basée sur la première accolade rencontrée échouait.

Solution : Pour remédier à ce problème, nous avons développé une solution innovante. Initialement, nous récupérions le nom de la classe en cherchant l'indice de la première accolade ouvrante, puis en retrouvant l'indice de l'espace précédent cette accolade pour extraire le nom. Cependant, cela ne fonctionnait pas lorsque l'utilisateur ajoutait un espace entre le nom de la classe et l'accolade ouvrante. Après une analyse approfondie, nous avons découvert que renommer le fichier sans spécifier de nom le renommait automatiquement en "null.java". Cette solution astucieuse a permis de contourner le problème initial, assurant ainsi le bon fonctionnement de notre processus de compilation en ligne de code Java.

3.3 Coloration Syntaxique

Explication : Le principal problème que nous avons rencontré résidait dans notre ignorance initiale quant à la nécessité d'utiliser des bibliothèques spécifiques pour notre projet. En effet, nous ne savions pas qu'il nous fallait intégrer certaines bibliothèques pour réaliser les fonctionnalités que nous avions envisagées. Cette méconnaissance nous a causé un contretemps considérable, car nous avons passé une partie non négligeable de notre temps à chercher des solutions sans avoir une direction claire. Cela a entraîné des retards importants dans notre planning et a affecté notre efficacité globale.En outre, notre manque de connaissance concernant la boucle de correspondance matcher.find() a constitué un obstacle majeur. La méthode matcher.find() est essentielle pour parcourir et identifier les motifs dans une chaîne de texte, une fonctionnalité cruciale pour la mise en œuvre de la coloration syntaxique dans notre éditeur de code. Ne pas connaître cette méthode nous a empêché de développer une solution efficace pour détecter et mettre en surbrillance les éléments syntaxiques du code, ce qui a entravé notre progression pendant un certain temps. Ce manque d'informations sur les bibliothèques nécessaires et sur

l'utilisation de matcher.find() a non seulement retardé notre développement, mais a également affecté notre moral. En effet, nous avons dû investir beaucoup de temps et d'efforts dans des recherches et des expérimentations infructueuses avant de trouver les bonnes ressources et méthodes. Cette situation a généré une frustration palpable au sein de l'équipe et a diminué notre motivation à certains moments critiques du projet.

Solution : La solution à nos problèmes a consisté en plusieurs étapes cruciales. Tout d'abord, nous avons intégré plusieurs bibliothèques essentielles, telles que "Flowless", "ReactFX", "RichTextFX", "UndoFX" et "WellBehavedFX". L'ajout de ces bibliothèques a considérablement amélioré notre capacité à gérer les fonctionnalités requises pour notre projet. Par exemple, Flawless et RéactiFS ont facilité la manipulation d'éléments de l'interface utilisateur et des flux réactifs, tandis que RichTextFX a été déterminant pour la gestion du texte enrichi, ce qui est crucial pour notre éditeur de code. "UndoFX" a permis d'implémenter des fonctionnalités de gestion des actions annulables, et WellBehaved FX a amélioré la gestion des comportements utilisateur. En outre, la compréhension et l'intégration de la méthode matcher.find() ont été déterminantes pour résoudre nos difficultés avec la coloration syntaxique. Auparavant, nous ne parvenions pas à implémenter cette fonctionnalité de manière efficace en raison de notre méconnaissance de cette méthode. Grâce à une meilleure compréhension de matcher.find(), nous avons pu développer une boucle de correspondance qui identifie et colore correctement les différentes parties du code, selon les règles syntaxiques définies. Cette avancée nous a permis d'obtenir une coloration syntaxique fonctionnant exactement comme nous le souhaitions, améliorant ainsi l'expérience utilisateur de notre éditeur de code.

3.5 Motivation

À l'annonce du projet, nous avons été extrêmement motivés car il correspondait directement à nos aspirations, nous permettant ainsi de nous investir dès les premiers instants. Cette motivation initiale nous a permis de maintenir un rythme soutenu et régulier jusqu'à parvenir au mode **STDIN/STDOUT**. Cependant, face aux difficultés rencontrées par la suite, nous avons subi une importante baisse de motivation. En effet, la plupart des autres problèmes ont pu être résolus assez rapidement, que ce soit par un changement de direction ou grâce à la découverte de nouvelles documentations. En revanche, celui-ci, malgré tous nos efforts et tentatives, a nécessité plus d'une semaine pour être résolu, ce qui nous a retardés par rapport à notre planning initial et a grandement entamé notre motivation, car nous pensions ne pas pouvoir respecter les délais. Heureusement, la réunion du 21/05 a permis de lever les blocages, ravivant ainsi notre élan de motivation et nous permettant de reprendre le projet avec encore plus de détermination.

4.Conclusion

En conclusion, notre application est sur le point de répondre pleinement aux exigences du cahier des charges, mais quelques détails restent à finaliser. Cette réalisation représente un investissement significatif en termes de temps et d'efforts, et nous sommes fiers du résultat obtenu. Cette expérience de développement nous a permis de faire des progrès considérables, tant sur le plan technique que sur celui de la gestion de projet. Nous avons appris de nos erreurs et des défis rencontrés, que ce soit au niveau du développement du code ou de la collaboration au sein de l'équipe.