Nom : KHADOUM AMABOUA

Prénom : Khamis Ahmat

Année : 2024-2025

Etablissement : Supdeco/ESITEC

PROJET : Système de recommandations de produits basé sur des graphes

Explication du code et exécution/test

Introduction

Le système de recommandations de produits basé sur des graphes est un programme développé en C qui permet aux utilisateurs de s'inscrire, de se connecter, d'acheter des produits et de visualiser leurs achats. Ce rapport présente les différentes sections du code, leur fonctionnalité, ainsi que des captures d'écran des exécutions et des tests effectués.

Structure du Projet

Le projet est organisé en plusieurs fichiers, chacun ayant un rôle spécifique :

recommandation.h : Ce fichier d'en-tête contient les définitions des structures de données et les déclarations des fonctions utilisées dans le programme.

main.c : Ce fichier contient l'implémentation principale de l'application, y compris la gestion des utilisateurs, des produits et des achats.

data.txt : Ce fichier stocke les informations des utilisateurs.

purchases.txt : Ce fichier enregistre les achats effectués par les utilisateurs.

projet : Fichier binaire généré après la compilation du code source.

Sections du Code

1. Gestion des Utilisateurs

La gestion des utilisateurs est implémentée dans le fichier main.c. Les principales fonctions incluent :

initialize_users : Charge les utilisateurs à partir du fichier data.txt.

add_user : Permet d'ajouter un nouvel utilisateur.

authenticate_user : Vérifie les informations d'identification d'un utilisateur lors de la connexion.

save_users : Sauvegarde les informations des utilisateurs dans le fichier data.txt.

2. Gestion des Produits

Les produits sont définis dans le tableau products dans main.c. Chaque produit a un nom et un prix. Les utilisateurs peuvent parcourir ces produits et effectuer des achats.

3. Gestion des Achats

Les achats sont gérés à l'aide d'un graphe, où les utilisateurs sont liés aux produits qu'ils ont achetés. Les fonctions clés incluent :

initialize_graph : Initialise le graphe des achats.

add_edge : Ajoute une relation entre un utilisateur et un produit avec un poids représentant la quantité achetée.

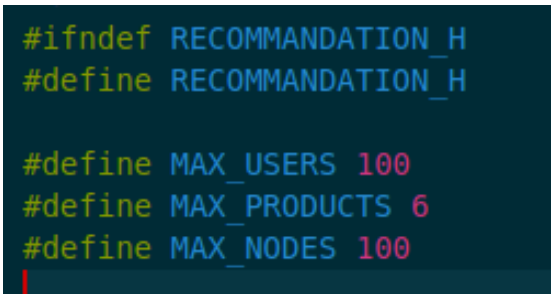
load_purchases : Charge les achats à partir du fichier purchases.txt.

4. Interface Utilisateur

L'interface utilisateur est textuelle et se compose d'un menu principal qui permet aux utilisateurs de naviguer dans les différentes fonctionnalités de l'application. Les utilisateurs peuvent s'inscrire, se connecter, faire des achats, visualiser leur panier et afficher le graphe des achats.

Explication du code

Screenshots du code : recommandation.h



L'extrait de code est un fichier d'en-tête de mon code qui utilise des directives de préprocesseur pour éviter les inclusions multiples.

#ifndef RECOMMANDATION_H et #define RECOMMANDATION_H garantissent que le contenu du fichier n'est inclus qu'une seule fois lors de la compilation.

Les constantes MAX_USERS, MAX_PRODUCTS, et MAX_NODES définissent respectivement les limites maximales pour le nombre d'utilisateurs, de produits et de nœuds dans le système, facilitant ainsi la gestion des ressources et la maintenance du code.

```c
typedef struct {
    char username[50];
    char password[50];
} User;

typedef struct {
    char name[50];
    double price;
} Product;

typedef struct {
    float weights[MAX_NODES][MAX_NODES];
    int node_count;
} Graph;
```

Ici, le code définit trois structures de données, chacune représentant un concept clé dans le système de recommandation.

User :

- Description: Cette structure représente un utilisateur du système.
- Attributs:

username[50]: Un tableau de caractères pour stocker le nom d'utilisateur, limité à 50 caractères.

password[50]: Un tableau de caractères pour stocker le mot de passe, également limité à 50 caractères.

Product:

- Description: Cette structure représente un produit dans le système.
- Attributs:

name[50]: Un tableau de caractères pour stocker le nom du produit, limité à 50 caractères.

price: Un double pour stocker le prix du produit.

Graph:

- Description: Cette structure représente un graphe qui modélise les relations entre utilisateurs et produits.
- Attributs:

weights[MAX_NODES][MAX_NODES]: Une matrice pour stocker les poids des arêtes entre les nœuds, où chaque nœud peut représenter un utilisateur ou un produit.

node_count: Un entier pour suivre le nombre total de nœuds dans le graphe.

Ces structures facilitent l'organisation et la gestion des données nécessaires pour le fonctionnement du système de recommandation.

```
void initialize_users(const char *filename);
int add_user(const char *username, const char *password);
int authenticate_user(const char *username, const char *password)
void save_users(const char *filename);

void initialize_store();
void store_menu(const char *username);

void initialize_graph();
void add_edge(int user_id, int product_id, float weight);
void recommend_products(int user_id);
void display_graph();
void free_graph();

void clear_screen();

#endif
```

Ce code présente des déclarations de fonctions pour gérer les utilisateurs, les produits et le graphe dans un système de recommandation :

- Gestion des Utilisateurs :

initialize_users charge les utilisateurs depuis un fichier.

add_user ajoute un nouvel utilisateur.

authenticate_user vérifie les informations d'identification d'un utilisateur.

save_users enregistre les utilisateurs dans un fichier.

- Gestion du Magasin :

initialize_store initialise les données du magasin.

store_menu affiche le menu du magasin pour un utilisateur spécifique.

- Gestion du Graphe :

initialize_graph initialise le graphe.

add_edge ajoute une relation entre un utilisateur et un produit.

recommend_products génère des recommandations pour un utilisateur.

display_graph affiche le contenu du graphe.

free_graph libère la mémoire allouée pour le graphe.

- Autres Fonctions :

clear_screen efface l'écran de la console.

Ces fonctions sont essentielles pour le fonctionnement et l'interaction du système.

- #endif est une directive de préprocesseur en C qui marque la fin d'une condition d'inclusion, généralement utilisée après #ifndef, #ifdef, ou #if. Il indique que le bloc de code entre #ifndef et #endif ne doit être inclus qu'une seule fois lors de la compilation, empêchant ainsi les inclusions multiples d'un même fichier d'en-tête. Cela aide à éviter les erreurs de compilation dues à des définitions redondantes.

Screenshots du code : main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h> // Pour sleep() sur Linux/Mac
#endif
#include "../include/recommandation.h"

#define DATA_FILE "data.txt"
```

- Bibliothèques standard :

stdio.h, stdlib.h, et string.h sont incluses pour les entrées/sorties, la gestion de la mémoire et la

manipulation de chaînes.

- Compatibilité multiplateforme :

Si le système est Windows (_WIN32), on inclut <windows.h> pour utiliser Sleep(). Sinon, on inclut <unistd.h> pour sleep() sur Linux/Mac.

- Fichier d'en-tête personnalisé :

#include "../include/recommandationt.h" inclut un fichier spécifique au projet.

- Macro :

#define DATA_FILE "data.txt" définit une constante pour le nom du fichier de données.

```
User users[MAX_USERS];
int user_count = 0;
int current_user_id = -1; // Identifiant de l'utilisateur connecté
```

User users[MAX_USERS];

- Description: Déclare un tableau nommé users qui peut contenir jusqu'à MAX_USERS utilisateurs. Chaque élément du tableau est de type User , représentant un utilisateur avec un nom d'utilisateur et un mot de passe.

int user_count = 0;

- Description: Initialise une variable entière user_count à 0, qui sert à suivre le nombre d'utilisateurs actuellement enregistrés dans le système.

int current_user_id = -1;

- Description: Initialise une variable entière current_user_id à -1, qui représente l'identifiant de l'utilisateur actuellement connecté. Une valeur de -1 indique qu'aucun utilisateur n'est connecté.

```c
int get_user_id(const char *username) {
    for (int i = 0; i < user_count; i++) {
        if (strcmp(users[i].username, username) == 0) {
            return i;
        }
    }
    return -1; // Utilisateur non trouvé
}

Product products[MAX_PRODUCTS] = {
    {"Laptop", 1000.0},
    {"Smartphone", 800.0},
    {"Headphones", 150.0},
    {"Backpack", 50.0},
    {"Smartwatch", 200.0},
    {"Tablet", 400.0}
};

Graph graph;
```

int get_user_id(const char *username):

- Description: Cette fonction recherche l'identifiant d'un utilisateur dans le tableau users en comparant le nom d'utilisateur fourni avec ceux des utilisateurs enregistrés.
- Fonctionnement: Elle parcourt le tableau jusqu'à trouver un utilisateur dont le nom correspond à username. Si trouvé, elle retourne l'indice de l'utilisateur dans le tableau. Si aucun utilisateur n'est trouvé, elle retourne -1.

Product products[MAX_PRODUCTS] = {...}; :

- Description: Déclare et initialise un tableau products contenant des objets de type Product, chacun représentant un produit avec un nom et un prix.
- Contenu: Le tableau est prérempli avec six produits, incluant un ordinateur portable, un smartphone, des écouteurs, un sac à dos, une montre connectée et une tablette, chacun avec son prix respectif.

Graph graph;:

- Description: Déclare une variable graph de type Graph, qui sera utilisée pour modéliser les relations entre utilisateurs et produits dans le système de recommandation.
- Fonctionnement: Cette variable servira à stocker les nœuds (utilisateurs et produits) et les arêtes (relations) du graphe, permettant ainsi de gérer les recommandations.

```c
// Gestion des utilisateurs
void initialize_users(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (file) {
        while (fscanf(file, "%49s %49s", users[user_count].username, users[user_count].password) == 2) {
            user_count++;
        }
        fclose(file);
    }
}
```

void initialize_users(const char *filename):

- Description: Cette fonction initialise la liste des utilisateurs en lisant leurs informations à partir d'un fichier spécifié par filename.
- Fonctionnement:Elle ouvre le fichier en mode lecture ("r").Si le fichier est ouvert avec succès, elle utilise fscanf pour lire les noms d'utilisateur et les mots de passe, en les stockant dans le tableau users.La fonction lit jusqu'à 49 caractères pour chaque champ (username et password) pour éviter les débordements de mémoire.Pour chaque paire de nom d'utilisateur et mot de passe lue, elle incrémente le compteur user_count.Une fois la lecture terminée, elle ferme le fichier.
- Utilité: Cela permet de charger les utilisateurs enregistrés dans le système à partir d'un fichier, facilitant ainsi la gestion des utilisateurs.

```c
#define PURCHASES_FILE "purchases.txt"

// Sauvegarder les achats dans un fichier
void save_purchases() {
    FILE *file = fopen(PURCHASES_FILE, "w");
    if (file) {
        for (int i = 0; i < user_count; i++) {
            if (strlen(users[i].username) > 0) { // Vérifier si l'utilisateur existe
                for (int j = 0; j < MAX_PRODUCTS; j++) {
                    if (graph.weights[i][j] > 0) { // Vérifier si l'utilisateur a acheté ce produit
                        fprintf(file, "%s %s %.1f\n", users[i].username, products[j].name, graph.weights[i][j]);
                    }
                }
            }
        }
        fclose(file);
    } else {
        printf("Error: Could not save purchases to file.\n");
    }
}
```

#define PURCHASES_FILE "purchases.txt":

- Description: Cette directive de préprocesseur définit une constante PURCHASES_FILE qui contient le nom du fichier où les achats seront sauvegardés. Dans ce cas, le fichier s'appelle "purchases.txt".

void save_purchases():

- Description: Cette fonction sauvegarde les achats des utilisateurs dans le fichier spécifié par PURCHASES_FILE.
- Fonctionnement:Elle ouvre le fichier en mode écriture ("w"). Si le fichier est ouvert avec succès, elle procède à l'écriture.Pour chaque utilisateur dans le tableau users, elle vérifie si le nom

d'utilisateur n'est pas vide (indiquant que l'utilisateur existe).Pour chaque produit, elle vérifie si l'utilisateur a un poids positif dans la matrice graph.weights, ce qui indique qu'il a acheté ce produit.Si ces conditions sont remplies, elle écrit dans le fichier le nom d'utilisateur, le nom du produit et le poids (qui représente probablement la quantité ou le montant dépensé).Une fois toutes les données écrites, elle ferme le fichier.Si le fichier ne peut pas être ouvert, elle affiche un message d'erreur.

- Utilité: Cette fonction permet de conserver un enregistrement des achats des utilisateurs, facilitant ainsi la gestion et l'analyse des données d'achat.

```c
// Charger les achats depuis un fichier
void load_purchases() {
    FILE *file = fopen(PURCHASES_FILE, "r");
    if (file) {
        char username[50], product_name[50];
        float weight;

        // Initialiser le graphe avant de charger les achats
        initialize_graph();

        while (fscanf(file, "%49s %49s %f", username, product_name, &weight) == 3) {
            int user_id = -1, product_id = -1;

            // Trouver l'ID de l'utilisateur
            for (int i = 0; i < user_count; i++) {
                if (strcmp(users[i].username, username) == 0) {
                    user_id = i;
                    break;
                }
            }

            // Trouver l'ID du produit
            for (int i = 0; i < MAX_PRODUCTS; i++) {
                if (strcmp(products[i].name, product_name) == 0) {
                    product_id = i;
                    break;
                }
            }

            // Ajouter l'achat au graphe
            if (user_id != -1 && product_id != -1) {
                graph.weights[user_id][product_id] = weight; // Remplacer le poids au lieu de l'ajouter
            }
        }
        fclose(file);
    } else {
        printf("No purchases file found. Starting with an empty graph.\n");
    }
}
```

void load_purchases():

- Description: Cette fonction charge les achats des utilisateurs à partir du fichier spécifié par PURCHASES_FILE et les intègre dans le graphe.
- Fonctionnement: Elle ouvre le fichier en mode lecture ("r"). Si le fichier est ouvert avec succès, elle continue le traitement. Elle déclare des variables pour stocker le nom d'utilisateur, le nom du produit et le poids (quantité ou montant dépensé). Avant de charger les achats, elle appelle initialize_graph() pour s'assurer que la structure du graphe est prête à recevoir les données. Elle utilise fscanf pour lire les données du fichier, en s'assurant de lire trois valeurs à la fois (nom d'utilisateur, nom du produit, poids). Pour chaque achat, elle cherche l'identifiant de l'utilisateur dans le tableau users et l'identifiant du produit dans le tableau products. Si les identifiants de l'utilisateur et du produit sont trouvés (c'est-à-dire qu'ils ne sont pas -1), elle met à jour le poids correspondant dans la matrice graph.weights, remplaçant la valeur existante. Une fois toutes les

données traitées, elle ferme le fichier.Si le fichier ne peut pas être ouvert, elle affiche un message indiquant qu'aucun fichier d'achats n'a été trouvé et que le graphe commence vide.

- Utilité: Cette fonction permet de restaurer les achats précédemment enregistrés, intégrant ainsi les données dans le système pour une utilisation ultérieure dans les recommandations.

```c
int add_user(const char *username, const char *password) {
    // Vérifier que les champs ne sont pas vides
    if (strlen(username) == 0 || strlen(password) == 0) {
        printf("Error: Username and password cannot be empty.\n");
        return 0; // Échec de l'inscription
    }

    // Vérifier que l'utilisateur n'existe pas déjà
    for (int i = 0; i < user_count; i++) {
        if (strcmp(users[i].username, username) == 0) {
            printf("Error: Username already exists.\n");
            return 0; // Échec de l'inscription
        }
    }

    // Ajouter l'utilisateur
    strcpy(users[user_count].username, username);
    strcpy(users[user_count].password, password);
    user_count++;
    save_users(DATA_FILE); // Sauvegarder immédiatement après ajout
    return 1; // Inscription réussie
}
```

int add_user(const char *username, const char *password):

- Description: Cette fonction ajoute un nouvel utilisateur au système avec un nom d'utilisateur et un mot de passe fournis.
- Fonctionnement: Elle commence par vérifier si les champs username ou password sont vides. Si l'un d'eux l'est, elle affiche un message d'erreur et retourne 0, indiquant un échec de l'inscription. Ensuite, elle parcourt le tableau users pour vérifier si le nom d'utilisateur existe déjà. Si un utilisateur avec le même nom est trouvé, elle affiche un message d'erreur et retourne 0. Si les vérifications passent, elle copie le nom d'utilisateur et le mot de passe dans le tableau users à l'indice user_count, puis incrémente user_count. Elle appelle ensuite save_users(DATA_FILE) pour sauvegarder immédiatement les utilisateurs dans le fichier de données. Enfin, elle retourne 1 pour indiquer que l'inscription a réussi.
- Utilité: Cette fonction gère l'ajout d'utilisateurs tout en s'assurant que les données sont valides et que les doublons sont évités, contribuant ainsi à la sécurité et à l'intégrité des données du système.

```c
int authenticate_user(const char *username, const char *password) {
    for (int i = 0; i < user_count; i++) {
        if (strcmp(users[i].username, username) == 0 && strcmp(users[i].password, password) == 0) {
            return 1; // Connexion réussie
        }
    }
    return 0;
}
```

int authenticate_user(const char *username, const char *password):

- Description: Cette fonction vérifie les informations d'identification d'un utilisateur pour l'authentification.

- Fonctionnement: Elle parcourt le tableau users pour comparer le nom d'utilisateur et le mot de passe fournis avec ceux des utilisateurs enregistrés. Pour chaque utilisateur, elle utilise strcmp pour vérifier si le username et le password correspondent à ceux stockés. Si une correspondance est trouvée, elle retourne 1, indiquant que la connexion a réussi. Si aucune correspondance n'est trouvée après avoir vérifié tous les utilisateurs, elle retourne 0, indiquant un échec de l'authentification.
- Utilité: Cette fonction est essentielle pour sécuriser l'accès au système, en garantissant que seuls les utilisateurs avec des informations d'identification valides peuvent se connecter.

```c
void save_users(const char *filename) {
    FILE *file = fopen(filename, "w");
    if (file) {
        for (int i = 0; i < user_count; i++) {
            fprintf(file, "%s %s\n", users[i].username, users[i].password);
        }
        fclose(file);
    }
}
```

void save_users(const char *filename):

- Description: Cette fonction sauvegarde les informations des utilisateurs dans un fichier spécifié par filename.
- Fonctionnement: Elle ouvre le fichier en mode écriture ("w"). Si le fichier est ouvert avec succès, elle continue le traitement. Elle parcourt le tableau users et utilise fprintf pour écrire le nom d'utilisateur et le mot de passe de chaque utilisateur dans le fichier, séparés par un espace. Une fois toutes les informations écrites, elle ferme le fichier.
- Utilité: Cette fonction permet de conserver un enregistrement persistant des utilisateurs, garantissant que leurs informations sont sauvegardées et peuvent être rechargées lors de la prochaine exécution du programme.

```c
// Gestion du magasin
void initialize_store() {
    printf("Store initialized with %d products.\n", MAX_PRODUCTS);
}

// Gestion des graphes
void initialize_graph() {
    memset(graph.weights, 0, sizeof(graph.weights)); // Initialiser toutes les valeurs à 0
    graph.node_count = 0;
    printf("Graph initialized.\n"); // Message de débogage
}

void add_edge(int user_id, int product_id, float weight) {
    if (user_id >= 0 && user_id < MAX_USERS && product_id >= 0 && product_id < MAX_PRODUCTS) {
        graph.weights[user_id][product_id] += weight; // Ajouter le poids de l'achat
        printf("Added purchase: User %d -> Product %d (Weight: %.1f)\n", user_id, product_id, graph.weights[user_id][product_id]);
        save_purchases(); // Sauvegarder immédiatement après modification
    } else {
        printf("Invalid user_id or product_id.\n");
    }
}
```

void initialize_store():

- Description: Cette fonction initialise le magasin et affiche un message indiquant le nombre de produits disponibles.
- Fonctionnement: Elle imprime simplement un message de débogage avec la constante MAX_PRODUCTS, indiquant que le magasin a été initialisé avec ce nombre de produits.

void initialize_graph():

- Description: Cette fonction initialise la structure du graphe en mettant toutes les valeurs de la matrice graph.weights à zéro.
- Fonctionnement: Elle utilise memset pour réinitialiser la matrice, puis met graph.node_count à 0 pour indiquer qu'aucun nœud n'a encore été ajouté. Un message de débogage est affiché pour confirmer l'initialisation.

void add_edge(int user_id, int product_id, float weight):

- Description: Cette fonction ajoute une relation (arête) entre un utilisateur et un produit dans le graphe, en spécifiant un poids qui représente l'achat.
- Fonctionnement: Elle vérifie d'abord que les identifiants user_id et product_id sont valides (dans les limites définies). Si les identifiants sont valides, elle ajoute le poids de l'achat à la matrice graph.weights à la position correspondante. Un message est affiché pour confirmer l'ajout de l'achat, incluant les identifiants de l'utilisateur et du produit ainsi que le poids. Elle appelle ensuite save_purchases() pour sauvegarder immédiatement les modifications dans le fichier. Si les identifiants ne sont pas valides, un message d'erreur est affiché.

- Utilité: Ces fonctions gèrent l'initialisation du magasin et du graphe, ainsi que l'ajout de relations entre utilisateurs et produits, ce qui est essentiel pour le fonctionnement du système de recommandation.

```c
void recommend_products(int user_id) {
    printf("Recommended products based on your purchases:\n");
    for (int i = 0; i < MAX_PRODUCTS; i++) {
        if (graph.weights[user_id][i] > 0) {
            printf("- %s (Interest: %.1f)\n", products[i].name, graph.weights[user_id][i]);
        }
    }
    printf("\nPress any key to continue...");
    getchar(); getchar(); // Pause avant de continuer
}
```

void recommend_products(int user_id):

- Description: Cette fonction génère et affiche des recommandations de produits pour un utilisateur donné, basées sur ses achats précédents.
- Fonctionnement: Elle commence par afficher un message indiquant que des produits recommandés seront présentés en fonction des achats de l'utilisateur. Elle parcourt la matrice graph.weights pour l'utilisateur spécifié par user_id, vérifiant chaque produit (de 0 à MAX_PRODUCTS). Si le poids (intérêt) pour un produit est supérieur à 0, cela signifie que l'utilisateur a manifesté un intérêt pour ce produit, et le nom du produit ainsi que le poids (intérêt) sont affichés. Après avoir affiché toutes

les recommandations, un message invite l'utilisateur à appuyer sur une touche pour continuer, et la fonction utilise getchar() pour créer une pause avant de continuer.

- Utilité: Cette fonction permet de fournir des recommandations personnalisées aux utilisateurs, améliorant ainsi leur expérience en leur suggérant des produits en fonction de leurs intérêts et achats antérieurs.

```c
void display_graph() {
    printf("\n=== Relations Graph (User -> Products) ===\n");

    for (int i = 0; i < user_count; i++) {
        if (strlen(users[i].username) > 0) {
            printf("User %s -> ", users[i].username);

            int has_purchases = 0;
            for (int j = 0; j < MAX_PRODUCTS; j++) {
                if (graph.weights[i][j] > 0) {
                    printf("%s (Quantity: %.1f) ", products[j].name, graph.weights[i][j]);
                    has_purchases = 1;
                }
            }

            if (!has_purchases) {
                printf("No purchases yet.");
            }
            printf("\n");
        }
    }

    printf("\nPress any key to continue...");
    getchar();
}
```

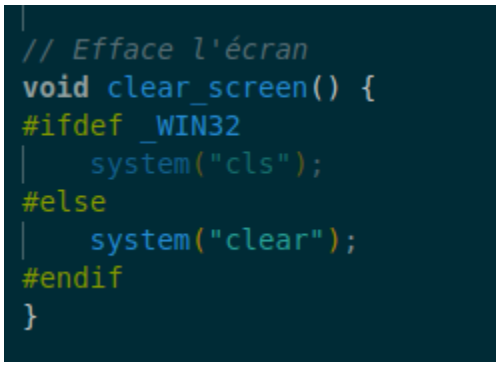
void display_graph():

- Description: Cette fonction affiche les relations entre les utilisateurs et les produits dans le graphe, montrant quels utilisateurs ont acheté quels produits.
- Fonctionnement: Elle commence par imprimer un en-tête pour indiquer qu'elle va afficher le graphe des relations. Pour chaque utilisateur dans le tableau users, elle vérifie si le nom d'utilisateur n'est pas vide. Si l'utilisateur a des achats, elle imprime le nom de l'utilisateur suivi d'une flèche (->). Elle parcourt ensuite la matrice graph.weights pour cet utilisateur, vérifiant chaque produit. Si le poids (quantité) est supérieur à 0, elle affiche le nom du produit et la quantité achetée. Si l'utilisateur n'a effectué aucun achat, elle affiche "No purchases yet." Après avoir affiché toutes les relations, elle invite l'utilisateur à appuyer sur une touche pour continuer, en utilisant getchar() pour créer une pause.
- Utilité: Cette fonction permet de visualiser les interactions entre utilisateurs et produits, facilitant ainsi la compréhension des comportements d'achat et des relations dans le système de recommandation.

```c
void clear_all_data() {
    // Vider les utilisateurs
    memset(users, 0, sizeof(users)); // Réinitialiser le tableau des utilisateurs
    user_count = 0; // Réinitialiser le compteur d'utilisateurs
    current_user_id = -1; // Réinitialiser l'ID de l'utilisateur connecté

    // Vider les achats (graphe)
    memset(graph.weights, 0, sizeof(graph.weights)); // Réinitialiser le graphe
    graph.node_count = 0;

    // Vider les fichiers de données
    FILE *file = fopen(DATA_FILE, "w");
    if (file) {
        fclose(file); // Crée un fichier vide
    } else {
        printf("Error: Could not clear user data file.\n");
    }

    file = fopen(PURCHASES_FILE, "w");
    if (file) {
        fclose(file); // Crée un fichier vide
    } else {
        printf("Error: Could not clear purchases data file.\n");
    }

    printf("All data has been cleared successfully!\n");
    #ifdef _WIN32
    Sleep(2000); // Pause de 2 secondes sur Windows
    #else
    sleep(2); // Pause de 2 secondes sur Linux/Mac
    #endif
    }
```

void clear_all_data():

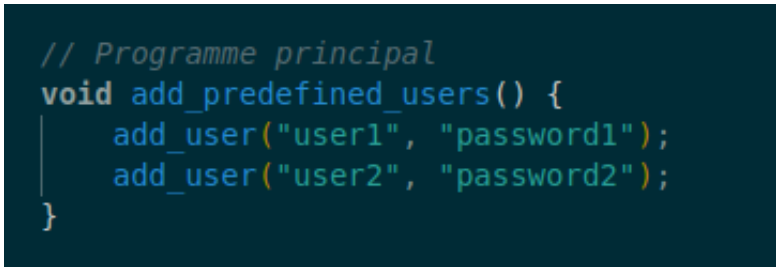
- Description: Cette fonction réinitialise toutes les données du système, y compris les utilisateurs, les achats et les fichiers de données associés.
- Fonctionnement: Elle commence par vider le tableau users en utilisant memset, ce qui réinitialise toutes les entrées à zéro. Elle remet également user_count à 0 et current_user_id à -1 pour indiquer qu'aucun utilisateur n'est connecté. Ensuite, elle réinitialise la matrice graph.weights à zéro pour effacer toutes les relations d'achat, et remet graph.node_count à 0. Pour vider les fichiers de données, elle ouvre DATA_FILE en mode écriture ("w"), ce qui crée un fichier vide. Si l'ouverture échoue, elle affiche un message d'erreur. Elle répète le même processus pour PURCHASES_FILE, créant également un fichier vide. Une fois toutes les données effacées, elle affiche un message de confirmation. Enfin, elle inclut une pause de 2 secondes, avec une gestion conditionnelle pour Windows (Sleep(2000)) et Linux/Mac (sleep(2)).
- Utilité: Cette fonction est essentielle pour réinitialiser complètement le système, permettant de commencer avec des données fraîches, ce qui peut être utile lors de tests ou de réinitialisations de l'application.



```
// Efface l'écran
void clear_screen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}
```

void clear_screen():

- Description: Cette fonction efface l'affichage de la console, offrant une interface plus propre pour l'utilisateur.
- Fonctionnement: Elle utilise une directive conditionnelle pour déterminer le système d'exploitation. Si le programme s'exécute sur Windows (_WIN32), elle appelle system("cls"), qui est la commande pour effacer l'écran dans l'invite de commandes Windows. Pour les systèmes Unix/Linux ou Mac, elle appelle system("clear"), qui est la commande équivalente pour effacer l'écran dans ces environnements.
- Utilité: Cette fonction améliore l'expérience utilisateur en permettant de nettoyer l'affichage de la console, ce qui est particulièrement utile lors de l'exécution de programmes interactifs ou de menus.



```
// Programme principal
void add_predefined_users() {
    add_user("user1", "password1");
    add_user("user2", "password2");
}
```

void add_predefined_users():

- Description: Cette fonction ajoute des utilisateurs prédéfinis au système avec des noms d'utilisateur et des mots de passe spécifiés.
- Fonctionnement:

Elle appelle la fonction add_user deux fois, une fois pour chaque utilisateur prédéfini :

    - add_user("user1", "password1") pour ajouter un utilisateur nommé "user1" avec le mot de passe "password1".
    - add_user("user2", "password2") pour ajouter un utilisateur nommé "user2" avec le mot de passe "password2".
- Utilité: Cette fonction est utile pour initialiser rapidement le système avec des utilisateurs de test, facilitant ainsi le développement et les tests sans nécessiter une saisie manuelle des informations d'utilisateur.

```c
// Fonction pour afficher le panier
void view_cart(int user_id) {
    printf("\n=== Your Cart ===\n");
    int has_purchases = 0;
    for (int i = 0; i < MAX_PRODUCTS; i++) {
        if (graph.weights[user_id][i] > 0) {
            printf("- %s (Quantity: %.1f)\n", products[i].name, graph.weights[user_id][i]);
            has_purchases = 1;
        }
    }
    if (!has_purchases) {
        printf("Your cart is empty.\n");
    }
    printf("\nPress any key to continue...");
    getchar(); getchar(); // Pause avant de continuer
}
```

void view_cart(int user_id):

- Description: Cette fonction affiche le contenu du panier d'un utilisateur spécifié par user_id, montrant les produits qu'il a achetés.
- Fonctionnement: Elle commence par imprimer un en-tête pour indiquer que le contenu du panier sera affiché. Elle initialise une variable has_purchases à 0 pour suivre si l'utilisateur a des achats dans son panier. Elle parcourt la matrice graph.weights pour l'utilisateur donné, vérifiant chaque produit (de 0 à MAX_PRODUCTS).Si le poids (quantité) d'un produit est supérieur à 0, cela signifie que l 'utilisateur a acheté ce produit. Dans ce cas, elle affiche le nom du produit et la quantité, et met has_purchases à 1. Si aucun produit n'a été trouvé dans le panier (c'est-à-dire que has_purchases reste 0), elle affiche un message indiquant que le panier est vide. Après avoir affiché le contenu du panier, elle invite l'utilisateur à appuyer sur une touche pour continuer, en utilisant getchar() pour créer une pause.
- Utilité: Cette fonction permet aux utilisateurs de visualiser les produits qu'ils ont ajoutés à leur panier, améliorant ainsi l'expérience d'achat en leur fournissant un aperçu de leurs sélections.

```c
int get_personal_recommendation() {
    if (current_user_id == -1) {
        printf("No user is currently logged in.\n");
        return -1;
    }

    float max_score = 0.0;
    int recommended_product = -1;

    for (int product_id = 0; product_id < MAX_PRODUCTS; product_id++) {
        float score = graph.weights[current_user_id][product_id];

        // Trouver le produit avec le score le plus élevé
        if (score > max_score) {
            max_score = score;
            recommended_product = product_id;
        }
    }

    return recommended_product;
}
```

int get_personal_recommendation():

- Description: Cette fonction génère une recommandation de produit personnalisée pour l'utilisateur actuellement connecté, en se basant sur ses achats précédents.
- Fonctionnement: Elle commence par vérifier si un utilisateur est connecté en vérifiant si current_user_id est égal. Si aucun utilisateur n'est connecté, elle affiche un message d'erreur et retourne -1.Elle initialise deux variables : max_score à 0.0 pour suivre le score le plus élevé trouvé et recommended_product à -1 pour stocker l'identifiant du produit recommandé. Elle parcourt tous les produits (de 0 à MAX_PRODUCTS) et récupère le score (poids) correspondant à chaque produit pour l'utilisateur actuel à partir de graph.weights. Si le score d'un produit est supérieur à max_score, elle met à jour max_score et recommended_product avec l'identifiant de ce produit. À la fin de la boucle, elle retourne l'identifiant du produit recommandé.
- Utilité: Cette fonction permet de fournir une recommandation personnalisée à l'utilisateur en identifiant le produit qu'il a le plus acheté, améliorant ainsi l'expérience d'achat en suggérant des articles qui l'intéressent probablement.

```c
int get_most_popular_product() {
    int max_count = 0;
    int most_popular_product = -1;

    for (int product_id = 0; product_id < MAX_PRODUCTS; product_id++) {
        int count = 0;

        // Parcourir toutes les lignes pour compter les utilisateurs ayant ce produit
        for (int user_id = 0; user_id < user_count; user_id++) {
            if (graph.weights[user_id][product_id] > 0) {
                count++;
            }
        }

        // Mise à jour si le produit est plus populaire
        if (count > max_count) {
            max_count = count;
            most_popular_product = product_id;
        }
    }

    return most_popular_product;
}
```

int get_most_popular_product():

- Description: Cette fonction détermine le produit le plus populaire parmi tous les utilisateurs en comptant combien d'utilisateurs ont acheté chaque produit.
- Fonctionnement: Elle initialise deux variables : max_count à 0 pour suivre le nombre maximum d'acheteurs d'un produit et most_popular_product à -1 pour stocker l'identifiant du produit le plus populaire. Elle parcourt tous les produits (de 0 à MAX_PRODUCTS) et, pour chaque produit, initialise un compteur count à 0. Pour chaque produit, elle parcourt toutes les lignes de la matrice graph.weights pour compter combien d'utilisateurs ont un poids (indiquant un achat) supérieur à 0 pour ce produit. Si le compteur count pour un produit est supérieur à max_count, elle met à jour max_count et most_popular_product avec l'identifiant de ce produit. À la fin de la boucle, elle retourne l'identifiant du produit le plus populaire.
- Utilité: Cette fonction permet d'identifier le produit le plus populaire dans le système, ce qui peut être utile pour des recommandations générales ou des promotions, en mettant en avant les articles qui intéressent le plus d'utilisateurs.

```c
void login(const char *username) {
    current_user_id = get_user_id(username); // Récupérer l'identifiant de l'utilisateur
    printf("User  '%s' logged in.\n", username);
}
```

void login(const char *username):

- Description: Cette fonction gère la connexion d'un utilisateur en utilisant son nom d'utilisateur.
- Fonctionnement: Elle commence par appeler la fonction get_user_id(username) pour récupérer l'identifiant de l'utilisateur correspondant au nom d'utilisateur fourni. Cette fonction doit retourner l'ID de l'utilisateur si celui-ci existe. Une fois l'identifiant récupéré, elle assigne cette valeur à

current_user_id, ce qui indique que l'utilisateur est maintenant connecté. Elle affiche un message confirmant que l'utilisateur a été connecté avec succès, en incluant le nom d'utilisateur dans le message.

- Utilité: Cette fonction est essentielle pour gérer le processus de connexion des utilisateurs, permettant au système de suivre quel utilisateur est actuellement connecté et d'adapter les fonctionnalités et les recommandations en conséquence.

```c
void store_menu(const char *username) {
    login(username); // Identifier l'utilisateur connecté
    int choice;

    printf("\nWelcome to the store, %s!\n", username);

    // Recommandation générale
    int general_product = get_most_popular_product();
    if (general_product != -1) {
        printf("\nPopular product (for all users):\n");
        printf("- %s ($%.2f)\n", products[general_product].name, products[general_product].price);
    }

    // Recommandation personnalisée
    int personal_product = get_personal_recommendation();
    if (personal_product != -1) {
        printf("\nRecommended for you:\n");
        printf("- %s ($%.2f)\n", products[personal_product].name, products[personal_product].price);
    }

    // Afficher les produits disponibles
    printf("\n=== Available Products ===\n");
    for (int i = 0; i < MAX_PRODUCTS; i++) {
        printf("%d. %s ($%.2f)\n", i + 1, products[i].name, products[i].price);
    }

    printf("\nChoose products to buy (enter product numbers separated by spaces, or 0 to exit): ");
    while (1) {
        scanf("%d", &choice);
        if (choice == 0) break;
        if (choice > 0 && choice <= MAX_PRODUCTS) {
            printf("You bought %s for $%.2f!\n", products[choice - 1].name, products[choice - 1].price);
            if (current_user_id != -1) {
                add_edge(current_user_id, choice - 1, 1.0); // Ajouter au panier
            } else {
                printf("Error: User is not logged in.\n");
                break;
            }
        } else {
            printf("Invalid choice. Please try again.\n");
        }
    }

    printf("\nPress any key to return to the menu...");
    getchar(); getchar(); // Pause avant de continuer
}
```

void store_menu(const char *username):

- Description: Cette fonction gère le menu principal du magasin, permettant à un utilisateur de se connecter, de voir des recommandations de produits, d'afficher les produits disponibles et d'effectuer des achats.
- Fonctionnement:

Elle commence par appeler la fonction login(username) pour identifier l'utilisateur connecté.Un message de bienvenue est affiché, incluant le nom de l'utilisateur.

- Recommandation générale: Elle appelle get_most_popular_product() pour obtenir le produit le plus populaire parmi tous les utilisateurs. Si un produit est trouvé, il est affiché avec son prix.
- Recommandation personnalisée: Elle appelle get_personal_recommendation() pour obtenir une recommandation personnalisée pour l'utilisateur. Si un produit est trouvé, il est affiché avec son prix.
- Affichage des produits disponibles: Elle imprime une liste de tous les produits disponibles, en affichant leur numéro, nom et prix.
- Processus d'achat: Elle invite l'utilisateur à entrer les numéros des produits qu'il souhaite acheter, séparés par des espaces, ou à entrer 0 pour quitter.Dans une boucle, elle lit les choix de l'utilisateur. Si l'utilisateur entre 0, la boucle se termine.Si l'utilisateur entre un numéro valide, elle affiche un message confirmant l'achat et appelle add_edge(current_user_id, choice - 1, 1.0) pour ajouter le produit au panier de l'utilisateur.Si l'utilisateur n'est pas connecté (current_user_id est -1), un message d'erreur est affiché.Si l'entrée est invalide, un message d'erreur est affiché et l'utilisateur est invité à réessayer.À la fin, elle invite l'utilisateur à appuyer sur une touche pour retourner au menu.
- Utilité: Cette fonction constitue le cœur de l'interaction utilisateur dans le magasin, permettant aux utilisateurs de naviguer, de recevoir des recommandations et d'effectuer des achats, tout en gérant leur état de connexion.

```c
int main() {
    int choice;
    char username[50], password[50];
    int is_logged_in = 0; // 0 = déconnecté, 1 = connecté
    int current_user_id = -1; // ID de l'utilisateur connecté

    initialize_users(DATA_FILE);
    initialize_store();
    initialize_graph();
    load_purchases(); // Charger les achats depuis le fichier
    add_predefined_users();

    while (1) {
        clear_screen();
        printf("\n");
        printf("╔═══════════════════════════════════════════════════════════════╗\n");
        printf("║                        🛒 MINI STORE 🛒                       ║\n");
        printf("╠═══════════════════════════════════════════════════════════════╣\n");

        if (!is_logged_in) {
            printf("║ 1. Register          | Create a new account               ║\n");
            printf("║ 2. Login             | Access your account                ║\n");
            printf("║ 3. Exit              | Quit the program                   ║\n");
        } else {
            printf("║ 1. View Cart         | See your purchased items           ║\n");
            printf("║ 2. Shop              | Browse and buy products            ║\n");
            printf("║ 3. Display Graph     | View purchase relationships        ║\n");
            printf("║ 4. Logout            | Log out from your account          ║\n");
            printf("║ 5. Clear All Data    | Reset all data (users & products)  ║\n");
            printf("║ 6. Exit              | Quit the program                   ║\n");
        }

        printf("╚═══════════════════════════════════════════════════════════════╝\n");
        printf("\nChoose an option: ");
        scanf("%d", &choice);
        getchar();
```

```c
switch (choice) {
case 1:
    if (!is_logged_in) {
        // Enregistrement
        clear_screen();
        char username[50];
        int registration_success = 0;

        while (!registration_success) {
            printf("\nEnter username: ");
            fgets(username, sizeof(username), stdin);
            username[strcspn(username, "\n")] = '\0'; // Supprimer le saut de ligne

            printf("Enter password: ");
            fgets(password, sizeof(password), stdin);
            password[strcspn(password, "\n")] = '\0'; // Supprimer le saut de ligne

            if (add_user(username, password)) {
                printf("User registered successfully!\n");
                registration_success = 1; // Sortir de la boucle
            } else {
                printf("Please try again.\n");
            }
        }
        #ifdef _WIN32
        Sleep(2000); // Pause de 2 secondes sur Windows
        #else
        sleep(2); // Pause de 2 secondes sur Linux/Mac
        #endif
    } else {
        // Afficher le panier
        clear_screen();
        view_cart(current_user_id);
    }
    break;
```

```c
case 2:
    if (!is_logged_in) {
        // Connexion
        clear_screen();
        int login_success = 0;

        while (!login_success) {
            printf("\nEnter username: ");
            fgets(username, sizeof(username), stdin);
            username[strcspn(username, "\n")] = '\0'; // Supprimer le saut de ligne

            printf("Enter password: ");
            fgets(password, sizeof(password), stdin);
            password[strcspn(password, "\n")] = '\0'; // Supprimer le saut de ligne

            if (authenticate_user(username, password)) {
                printf("Login successful!\n");
                is_logged_in = 1;
                current_user_id = get_user_id(username); // Mettre à jour l'ID de l'utilisateur
                login_success = 1; // Sortir de la boucle
            } else {
                printf("Invalid username or password. Please try again.\n");
            }
        }
        #ifdef _WIN32
        Sleep(2000); // Pause de 2 secondes sur Windows
        #else
        sleep(2); // Pause de 2 secondes sur Linux/Mac
        #endif
    } else {
        // Faire des courses
        clear_screen();
        store_menu(users[current_user_id].username);
    }
    break;
```

```c
case 3:
    if (!is_logged_in) {
        // Quitter
        printf("Exiting the program. Goodbye!\n");
        exit(0);
    } else {
        // Afficher le graphe
        clear_screen();
        display_graph();
    }
    break;

case 4:
    if (is_logged_in) {
        // Déconnexion
        is_logged_in = 0;
        current_user_id = -1;
        printf("You have been logged out.\n");
        #ifdef _WIN32
        Sleep(2000); // Pause de 2 secondes sur Windows
        #else
        sleep(2); // Pause de 2 secondes sur Linux/Mac
        #endif
    } else {
        printf("Invalid option. Please try again.\n");
    }
    break;

case 5:
    if (is_logged_in) {
        clear_all_data();
    } else {
        printf("Invalid option. Please try again.\n");
    }
    break;
```

int main():

- Description: La fonction principale du programme, qui gère l'exécution de l'application de magasin, y compris l'initialisation, la connexion des utilisateurs, l'affichage des menus et la gestion des choix de l'utilisateur.
- Fonctionnement:
  - Initialisation: Elle commence par déclarer des variables pour stocker le choix de l'utilisateur, le nom d'utilisateur, le mot de passe, l'état de connexion (is_logged_in), et l'identifiant de l'utilisateur connecté (current_user_id). Elle appelle plusieurs fonctions pour initialiser les utilisateurs, le magasin, le graphe, charger les achats depuis un fichier, et ajouter des utilisateurs prédéfinis.
  - Boucle Principale: Une boucle infinie est utilisée pour afficher le menu principal et gérer les choix de l'utilisateur. La fonction clear_screen() est appelée pour nettoyer l'affichage à chaque itération. Un menu est affiché, qui varie selon que l'utilisateur est connecté ou non. L'utilisateur est invité à entrer un choix, qui est ensuite traité par un switch.
  - Gestion des Choix:
    - Choix 1: Si l'utilisateur n'est pas connecté, il peut s'enregistrer. Il est invité à entrer un nom d'utilisateur et un mot de passe, et la fonction add_user est appelée pour enregistrer

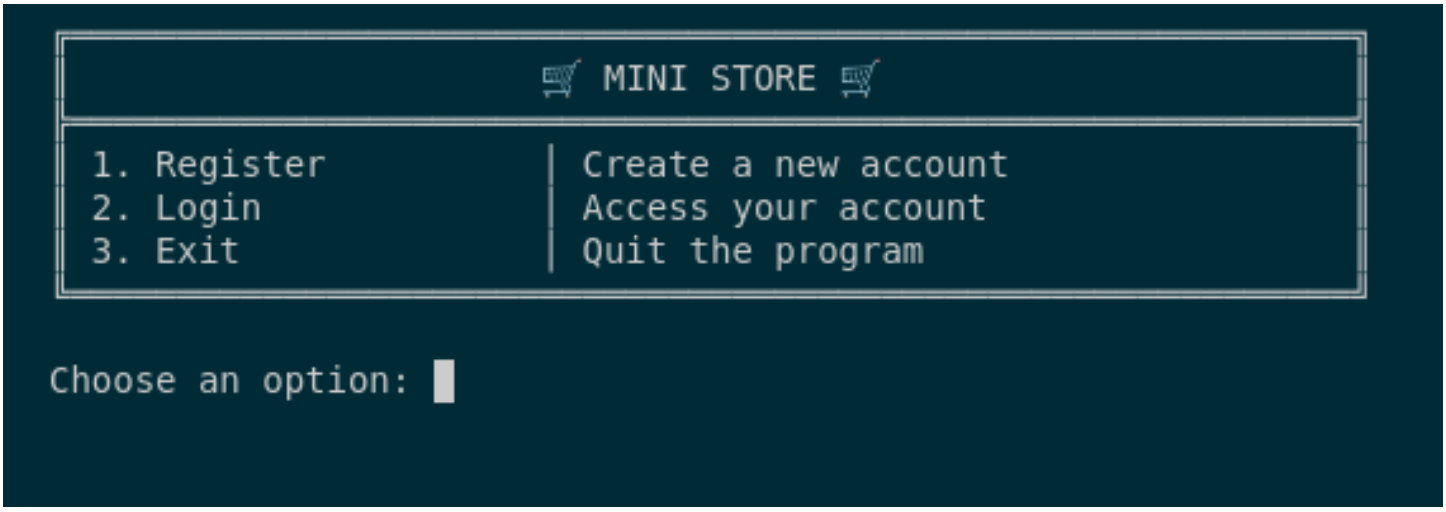
l'utilisateur. Si l'utilisateur est connecté, il peut voir son panier en appelant view_cart(current_user_id).

- Choix 2: Si l'utilisateur n'est pas connecté, il peut se connecter. Il entre son nom d'utilisateur et son mot de passe, et la fonction authenticate_user est appelée pour vérifier ses informations d'identification. Si l'utilisateur est connecté, il peut faire des courses en appelant store_menu(users[current_user_id].username).
- Choix 3: Si l'utilisateur n'est pas connecté, il quitte le programme. S'il est connecté, il affiche le graphe des relations d'achat en appelant display_graph().
- Choix 4: Si l'utilisateur est connecté, il se déconnecte, et les variables d'état sont réinitialisées.
- Choix 5: Si l'utilisateur est connecté, il peut effacer toutes les données en appelant clear_all_data().
- Choix 6: Quitte le programme.

- Gestion des Erreurs: Pour chaque choix, des messages d'erreur sont affichés si l'entrée est invalide ou si l'utilisateur essaie d'accéder à des fonctionnalités non disponibles.
- Utilité: Cette fonction constitue le point d'entrée de l'application, orchestrant l'ensemble des fonctionnalités du magasin, de la gestion des utilisateurs à l'affichage des produits et à la gestion des achats, tout en assurant une expérience utilisateur fluide et interactive.

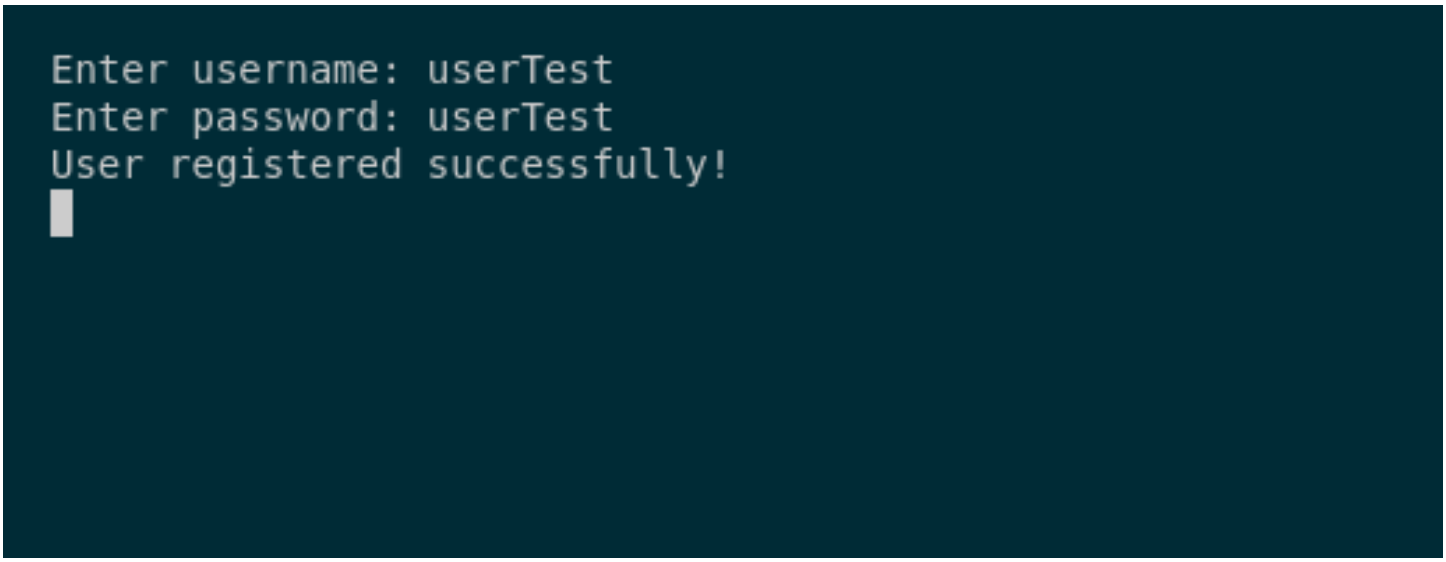
Exécution et Tests

Capture d'Écran 1 : Menu Principal



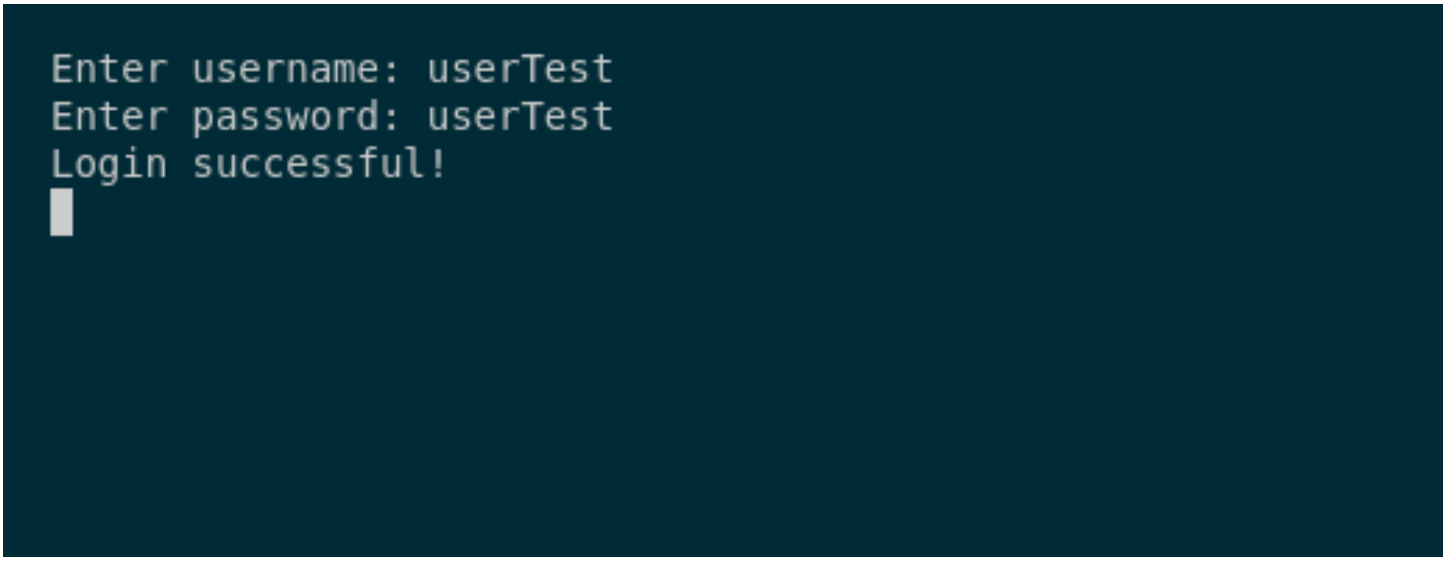
Description : Capture d'écran du menu principal de l'application, affichant les options disponibles pour les utilisateurs non connectés.

Capture d'Écran 2 : Inscription d'un Utilisateur

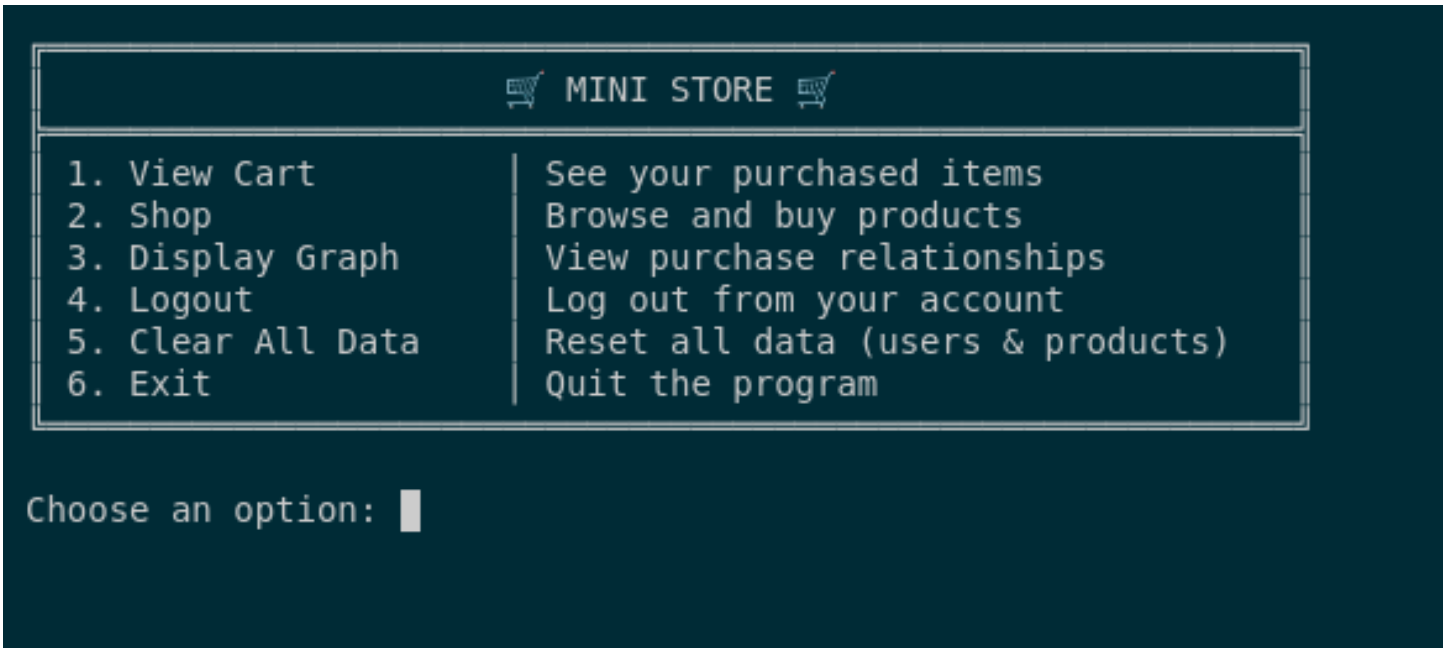


Description : Capture d'écran du processus d'inscription d'un nouvel utilisateur, où l'utilisateur entre son nom d'utilisateur et son mot de passe.

Capture d'Écran 3 : Connexion d'un Utilisateur et affichage du nouveau menu

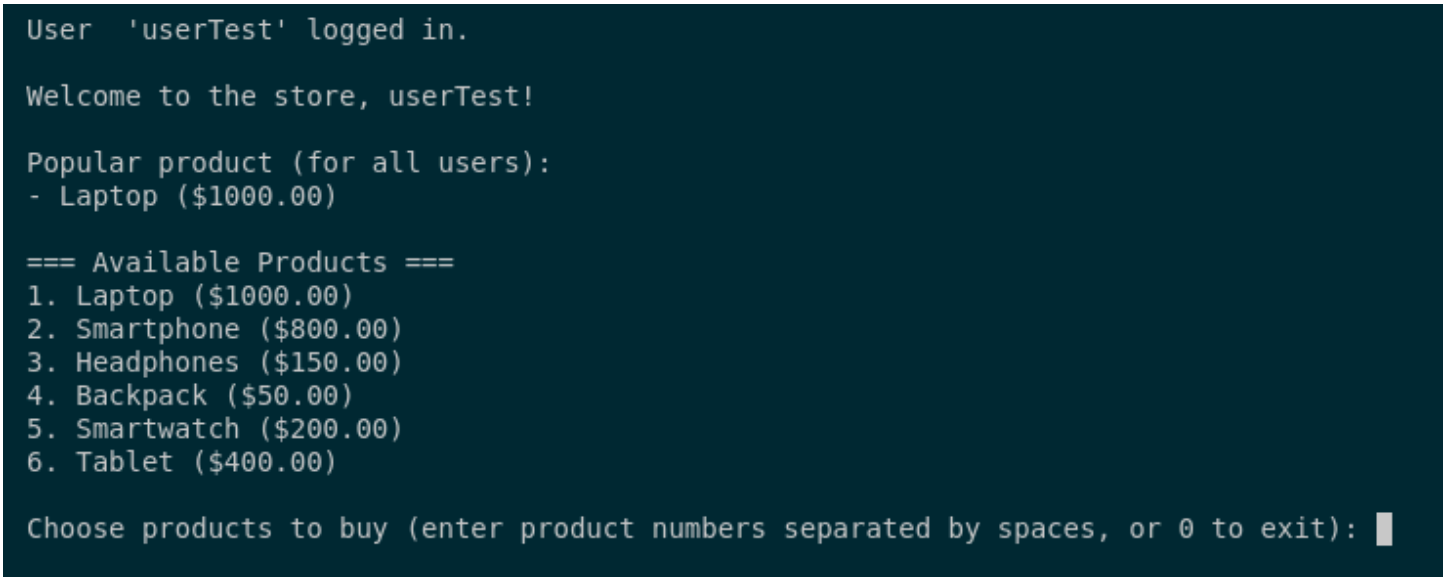


Description : Capture d'écran du processus de connexion, où l'utilisateur entre ses informations d'identification.

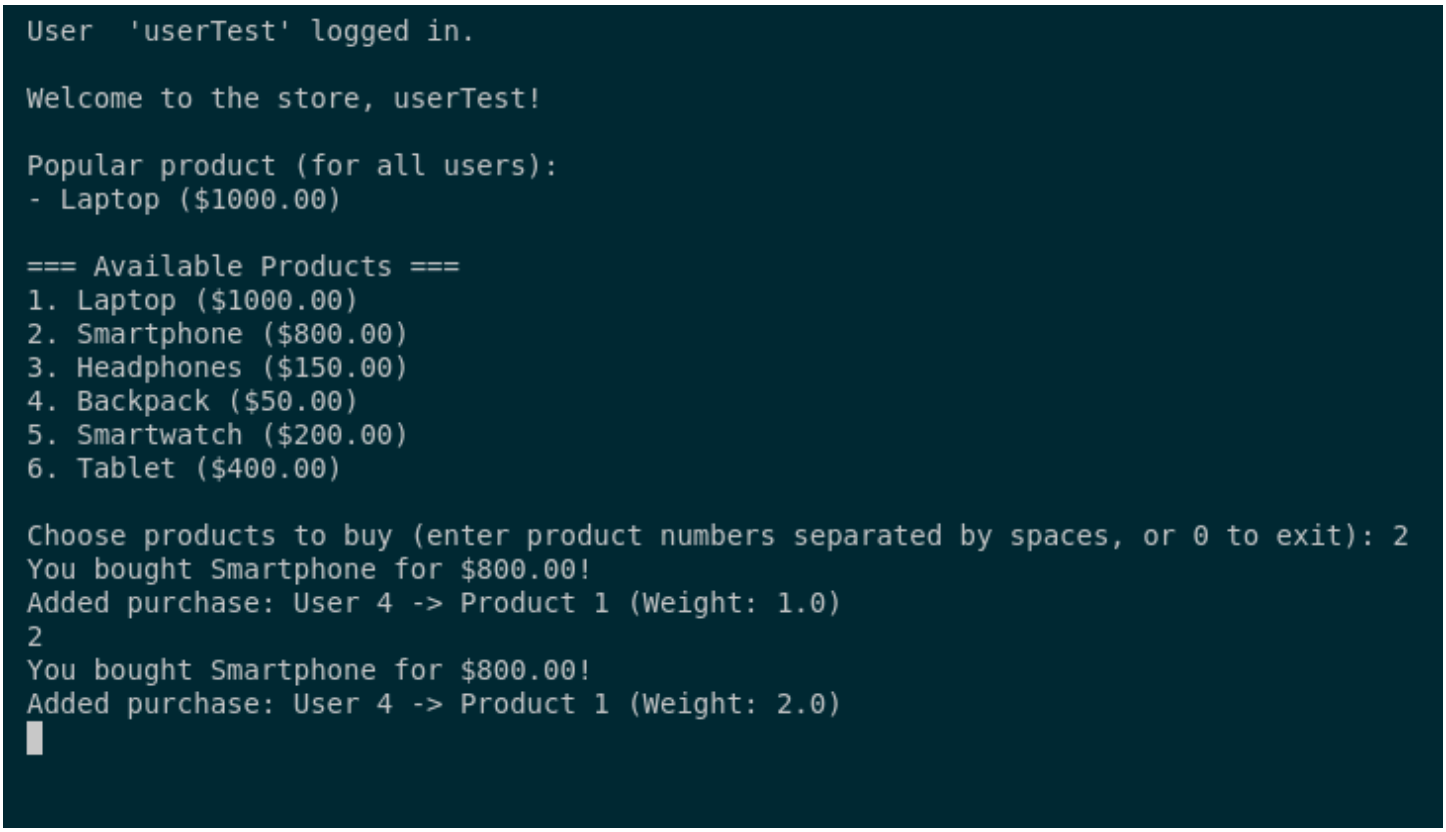


Description : Capture d'écran du menu d'après connexion , affichant les options disponibles pour les utilisateurs connectés.

Capture d'Écran 4 : Achat de Produits

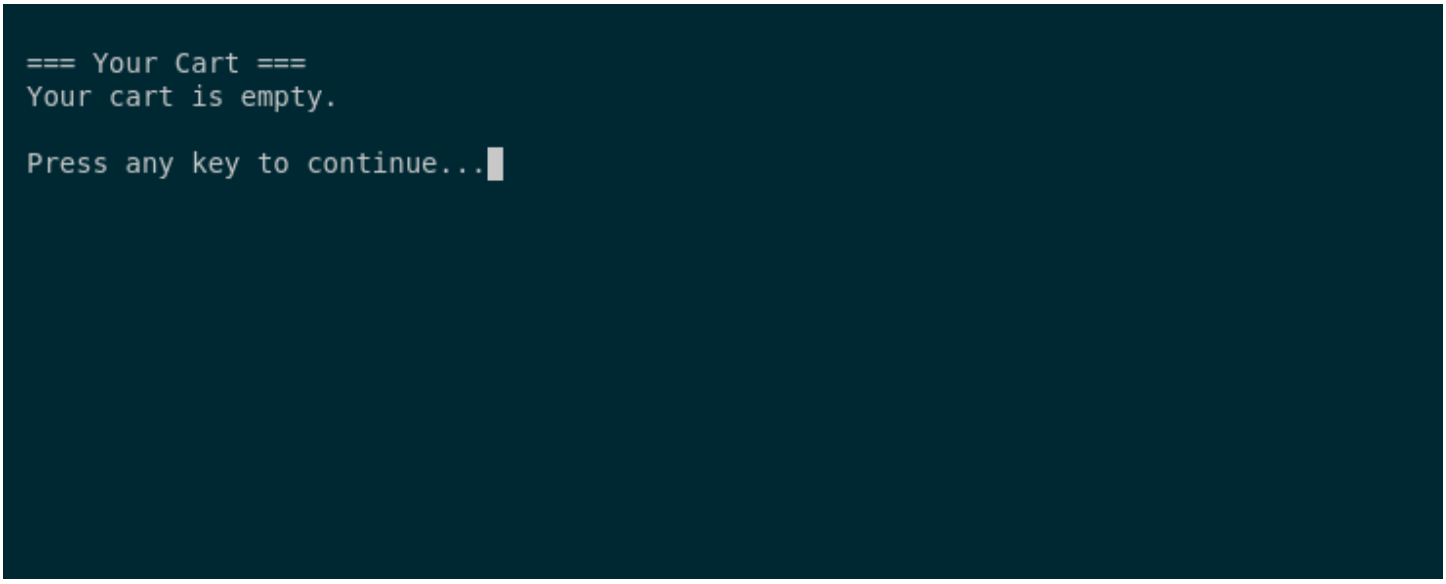


Description : Capture d'écran montrant l'interface d'achat, où l'utilisateur peut sélectionner des produits à acheter, il lui est proposer dès son entrée le produit phare du magasin et si il fait des achats il lui sera proposer le produit phare vis-à-vis de tous les utilisateurs et le produit qu'il a le plus acheté.



Description : Capture d'écran montrant l'interface d'achat, où l'utilisateur peut sélectionner des produits à acheter, plus l'utilisateur achete un produit plus sa relation avec ce dernier augmente cela servira pour les recommendations.

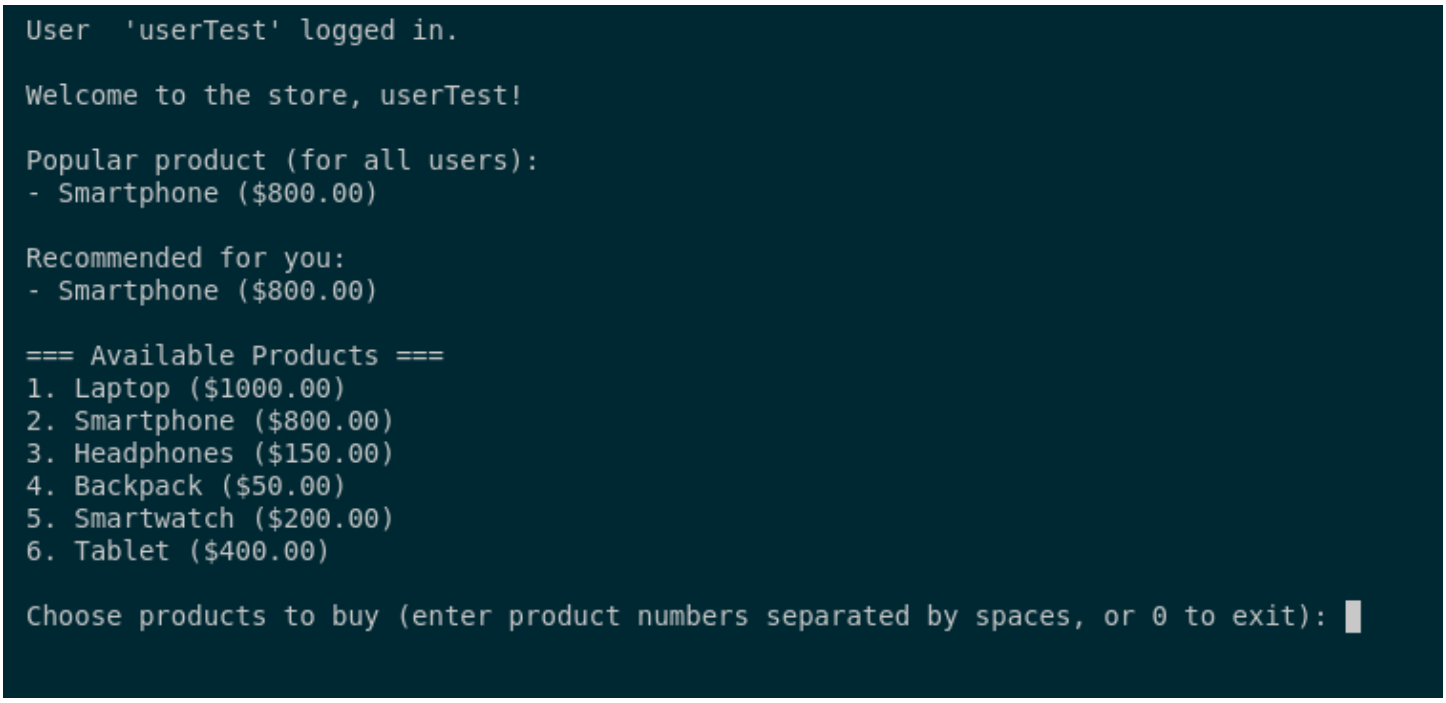
Capture d'Écran 5 : Visualisation du Panier



Description : Capture d'écran affichant le panier de l'utilisateur vide.

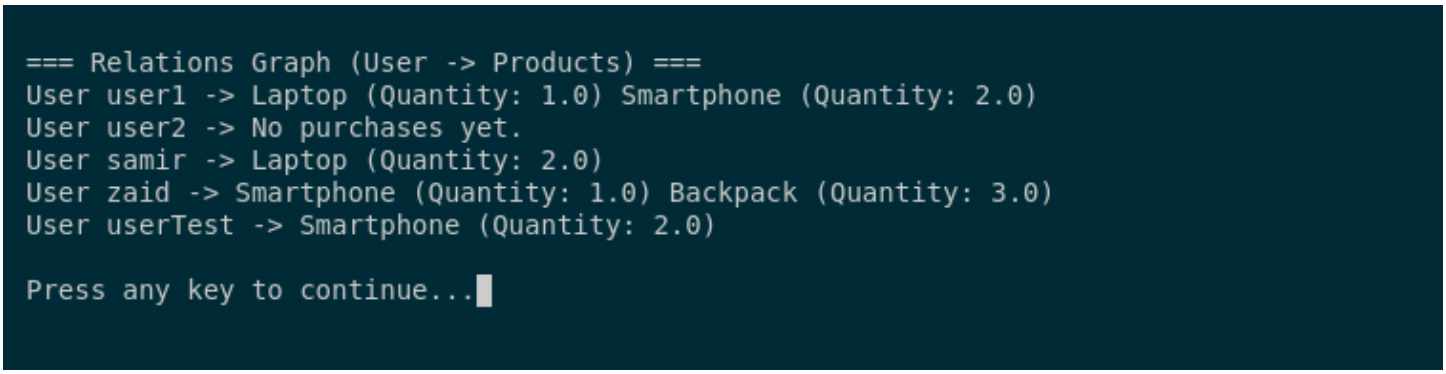
Description : Capture d'écran affichant le panier de l'utilisateur avec les produits achetés, dans notre cas il s'agit de deux smartphones.

Capture d'Écran 6 : Visualisation des recommendations des achats du magasin après que l'utilisateur ait fait des courses.



Description : Capture d'écran affichant le magasin de l'utilisateur avec les produits les plus appréciés par tous et par l'utilisateur, au depart pour des raisons de test le produit le plus apprécié par tous est le Laptop, mais cela change au fur et à mesure selon les achats de tout un chacun.

Capture d'Écran 7 : Graphe des relations



Description : Capture d'écran affichant le graphe de relations de chaque utilisateur avec les produits achetés par lui, si le panier d'un utilisateur est vide cela s'affichera, cette option sert également à la recommendation de produits dans le magasin.

Conclusion

Le Système de recommandations de produits basé sur des graphes est un projet complet qui démontre la gestion des utilisateurs, des produits et des achats à travers une interface utilisateur simple. Les tests effectués ont montré que toutes les fonctionnalités fonctionnent comme prévu, offrant une expérience utilisateur fluide. Ce projet peut être étendu avec des fonctionnalités supplémentaires, telle que une interface graphique pour améliorer l'expérience utilisateur.