

Bersin Mathieu

Bonneau Corentin

Goyalongo Francois

Puccetti Valentin

RT2-FA

Compte Rendu

SAE :

Objectif de la SAE: Développer des applications
communicantes.

Table des matières

1/ Introduction	5
Explication du projet	5
Récupération du projet sur github	5
Prérequis	5
Fichier Make	7
Structure et communication	8
1.1/ Chiffrement	8
1.2/ Requêtes	9
1.3/ Réponses	10
Liste des programmes	11
1.4/ Backend (serveur)	11
1.5/ Clients	11
2/ Diagramme de cas d'utilisation	12
2.2/ Diagramme de classes	13
3/ Conception du backend	14
Structure	14
Gestion des données	14
Démarrage des éléments	15
3.1/ Initialisation du chiffrement	16
3.2/ Initialisation de la base de données	16
3.4/ Initialisation du socket	18
Déchiffrement et établissement des sessions	19
Gestion des requêtes	20
Trie des status	20
Exécution des fonctions liés aux requêtes	20
Fonctions de créations	21
Manipulation des données	22
Création d'un élément	22
Liste de plusieurs éléments	22
Suppression d'un élément	24
4/ Le client en C	25
4.2/ Les fichiers handlers :	25
4.3/ Les fichiers network :	32
4.4/ Le fichier main :	37
4.5/ Le Makefile :	40

Démonstration du client C en CLI	42
5/ Le client en Java (CLI + GUI)	45
5.1/ La classe Client :	46
5.2/ La classe Etudiant.....	50
5.3/ La classe Seance :.....	51
5.4/ La classe Absence :.....	54
5.5/ Le fichier Main :	56
Explication du mode CLI	56
Démonstration du client Java en CLI	66
Explication du mode GUI	69
5.6/ La classe Accueil.....	70
5.7/ La classe Connected.....	74
5.8/ La classe Attendance.....	81
5.9/ La classe ManageSeances	85
5.9/ La classe ManageStudents.....	92
Démonstration du client Java en GUI.....	93
6/ Le client Android	97
6.1/ Le fichier Manifest:	97
6.2/ Le fichier MyThread:.....	98
6.3/ Le fichier MainActivity:	100
6.4/ Le fichier MyThreadCommand :	102
6.5/ Les fichiers ManageSeancesActivity et ManageStudentsActivity:	105
6.6/ Les fichiers CreaSeancesActivity et CreateStudentsActivity:.....	108
6.7/ Le fichier MenuActivity:.....	111
6.8/ Les fichiers ViewAttendanceActivity et SaveAttendanceActivity:.....	112
Démonstration du client Android.....	116
7/ Le client Python	116
7.2/ Le programme en CLI (cli.py):	117
7.3/ Le fichier Main.py:	120
Démonstration du client python en CLI	125
Démonstration du client python en GUI	126
Conclusion	131

1/ Introduction

Explication du projet

Le projet consiste à [développer une application de gestion des absences](#) basée sur une architecture client-serveur utilisant les sockets pour la communication réseau.

Cette application permettra aux utilisateurs (étudiants, enseignants, administrateurs) de gérer et de suivre les absences de manière efficace et centralisée.

Récupération du projet sur github

Afin de faciliter l'installation de notre serveur ainsi que nos différents clients, le projet est sur un github. Ainsi voici la commande à effectuer afin de récupérer la totalité de notre projet, il faut avoir installer le package « git » pour le réaliser en CLI (yum install git).

Commande = « git clone <https://github.com/itsmrval/iut-sae-302> ».

Par la suite, vous pouvez compiler la totalité de nos programmes grâce à un Makefile qui va compiler tout le programme. Dans le répertoire « iut-sae-302 » faites la commande « make » et un répertoire sera créé se nommant « dist » avec les exécutables du serveur et des clients.

Pour l'émulateur android, le démarrer avec “make run_android”. Il lancera un téléphone avec l'apk “app.apk”

Prérequis

Installer gradle si vous n'avez pas la version du projet avec le dossier gradle. Pour cela allez sur le site « <https://docs.gradle.org/current/userguide/installation.html> » et mettez le répertoire gradle au-sein du projet.

Pour information, dans ces exemples, nous allons montrer comment effectuer les installations des packages nécessaires sur un ordinateur CentOS9 comme cela a été demandé.

Avoir installé GCC sur l'ordinateur. :

- « dnf install gcc »

Avoir installé la librairie OpenSSL :

- « dnf install -y openssl-devel »

Avoir installé Make permettant la compilation :

- « dnf install make »

Avoir installé Java :

- « dnf install java-devel »

Avoir installé Python :

- « dnf install python3-tkinter »

Voici les commandes à effectuer pour installer les librairies sur MAC.

Si GCC n'est pas déjà installé veuillez le faire :

- « brew install gcc »

Avoir installé la librairie OpenSSL :

- « brew install openssl »

Avoir installé la librairie make :

- « brew install make »

Avoir installé Java :

- « brew install openjdk »

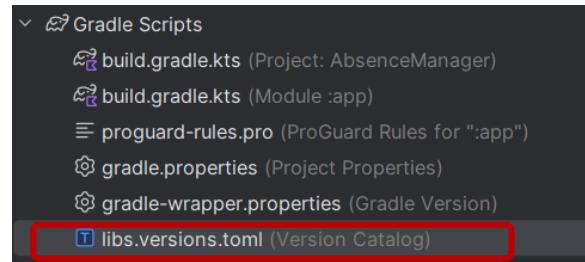
Avoir installé Python :

- « brew install python-tk »

Installer android studio:

Note: Si le SDK android studio n'est pas présent dans ~/Library/Android/sdk/ sur votre installation, merci de le spécifier dans le Makefile à la ligne "ANDROID_SDK"

Enfin, des problèmes peuvent apparaître si la version de l'AGP (plugin utilisé par Gradle pour gérer la construction des projets Android) n'est pas compatible. Ainsi, lors de la construction du projet, l'application indique la version de l'AGP nécessaire en fonction de la version installée sur votre machine. Par exemple, une erreur pourrait être : « The project is using an incompatible version (AGP 8.2.0) of the Android Gradle plugin. The latest supported version is AGP 8.2.0-beta05.



Pour cela, indiqué dans l'élément AGP, la version indiquée dans l'erreur.

```
[versions]
agp = "8.8.0"
```

Fichier Make

Enfin, vous pouvez exécutée la commande « make » qui va installer tous les éléments nécessaires. Vous trouverez dans le repertoire dist/ les clients et le server. Ainsi, vous pouvez simplement lancer le client android avec par exemple la commande « make run_android ».

Structure et communication

Afin de permettre la [communication entre les clients et le serveur](#), nous avons dû choisir une logique cohérente, fiable et sécurisée

1.1/ Chiffrement

Celle-ci est appuyée par un chiffrement OpenSSL en utilisant un combo [clé privée et publique](#) utilisant RSA 4096 entre les clients et le serveur.

Pour créer ces clés, nous utilisons la commande suivante :

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -sha256 -days 3650 -nodes -config config.cnf
```

Celle-ci crée un combo valide pendant 10 ans, utilisant la configuration suivante (définissant le DN, et les hostname autorisés).

```
× ..../r302/openssl (-zsh)
▶ cat config.cnf
[ req ]
default_bits      = 4096
distinguished_name = req_distinguished_name
req_extensions    = v3_req
prompt            = no

[ req_distinguished_name ]
C      = FR
ST     = France
L      = Paris
O      = IUT
OU    = Vélizy
CN    = localhost

[ v3_req ]
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = localhost
▶ ~/Documents/Repositories/r302/openssl on main *2 !7 ?3
```

1.2/ Requêtes

Se rapprochant des headers du protocole HTTP, nous avons choisi de définir le format des messages suivant :

<TYPE>endpoint/param1/param2/..

Type :

Il s'agit d'un string, permettant de savoir l'**action à effectuer** sur l'endpoint par le serveur

- "<g>" : GET
 - o Permet de récupérer une information depuis le serveur, sans en modifier le contenu
- "<p>" : POST
 - o Crée un élément sur le serveur
- "<d>" : DELETE
 - o Supprime un élément sur le serveur
- "<a>" : ACTION
 - o Effectue une action tierce relative à la liaison

Endpoint :

Il s'agit d'un string spécifiant sur **quel type d'élément** l'action doit être effectuée

- "student" : Gestion des étudiants
- "seance" : Gestion des séances d'enseignement
- "attendance" : Gestion de la présence des étudiants pour chaque séance

Param :

Il s'agit des **paramètres**, qui peuvent être un entier ou un string en fonction de l'action et de l'élément, dont voici la liste exhaustive :

Type	Endpoint	Paramètres	Description
GET	student	(int) id	Récupère la liste complète des étudiants ou celui spécifié par l'id
GET	seance	(int) id	Récupère la liste complète des séances ou celle spécifiée par l'id
GET	attendance	(int) id_seance (int) id_student	Vérifie la présence d'un étudiant sur une séance donnée.
POST	student	(string) name	Crée un étudiant avec un nom donné
POST	seance	(string) name (int) unix_time	Crée une séance avec un nom, et une date sous format UNIX
POST	attendance	(int) id_student (int) id_seance (int) status	Définis la présence d'un étudiant sur une séance donnée, avec comme status 1 si présent, et 0 si absent
DELETE	student	(int) id_student	Supprime un étudiant à partir de son identifiant unique

<i>DELETE</i>	seance (int) id_seance	Supprime une séance à partir de son identifiant unique
<i>ACTION</i>	disconnect	Déconnecte le socket et le log
<i>ACTION</i>	close	Ferme le serveur - à des fins de développement

1.3/ Réponses

Les [réponses](#) aux requêtes précédentes s'organisent de la manière suivante :

[status](#)/[resp](#):[data](#),[resp2](#):[data2](#),...

Status:

Le status se rapproche des codes HTTP, dont voici la liste utilisée :

- 202 : OK
 - o Lors-ce qu'une requête est [correctement exécutée](#)
- 400 : BAD
 - o Lors-ce que la requête est [mal formulée](#)
- 499 : DISCONNECTED / CLOSED
 - o Lors-ce que la requête mène à la [déconnexion](#) du client
- 500 : ERROR
 - o Lors-ce que la requête [mène à une erreur](#)

Resp et data

Il s'agit du nom de la colonne réponse, et de son contenu.

Par exemple, lors-ce que l'on récupère la liste des étudiants manuellement à travers un terminal de développement réalisé à cet effet :

```
Connecting to server at localhost:8081...
[INFO] Connected to the server. Type commands or 'exit' to quit.
>> <g>student
[SERVER RESPONSE]
:202:id:3,name:Valentin PUCCETTI:id:0,name:Corentin BONNEAU:id:1,name:François GOYA:id:2,name:Mathieu BERSIN;
>> |
```

Contenus multiples

Si la réponse contient plusieurs éléments, une liste est réalisée et séparés à l'aide des caractères ";"

Liste des programmes

Voici la liste exhaustive des différents programmes réalisés pour le bon fonctionnement de cette SAE

1.4/ Backend (serveur)

Le backend est le [cœur du projet](#), il s'occupe du traitement des données envoyées par les différents clients. Celui-ci a été développé en langage C utilisant la librairie [OpenSSL](#) pour le chiffrement des données.

1.5/ Clients

Il existe différents clients pouvant [interagir avec le backend](#), ceux-ci sont listés ci-dessous :

Android

L'application Android a été développée sur [Android Studio](#) utilisant le langage de programmation Java.

Celle-ci propose une interface graphique permettant d'utiliser toutes les fonctionnalités du projet

Java GUI / CLI

Une [application java](#) fonctionnant sur PC et Mac permettant une interaction en ligne de commande, et en interface graphique.

C CLI

Un programme C fonctionnant sur PC et Mac permettant une interaction en ligne de commande.

Python GUI

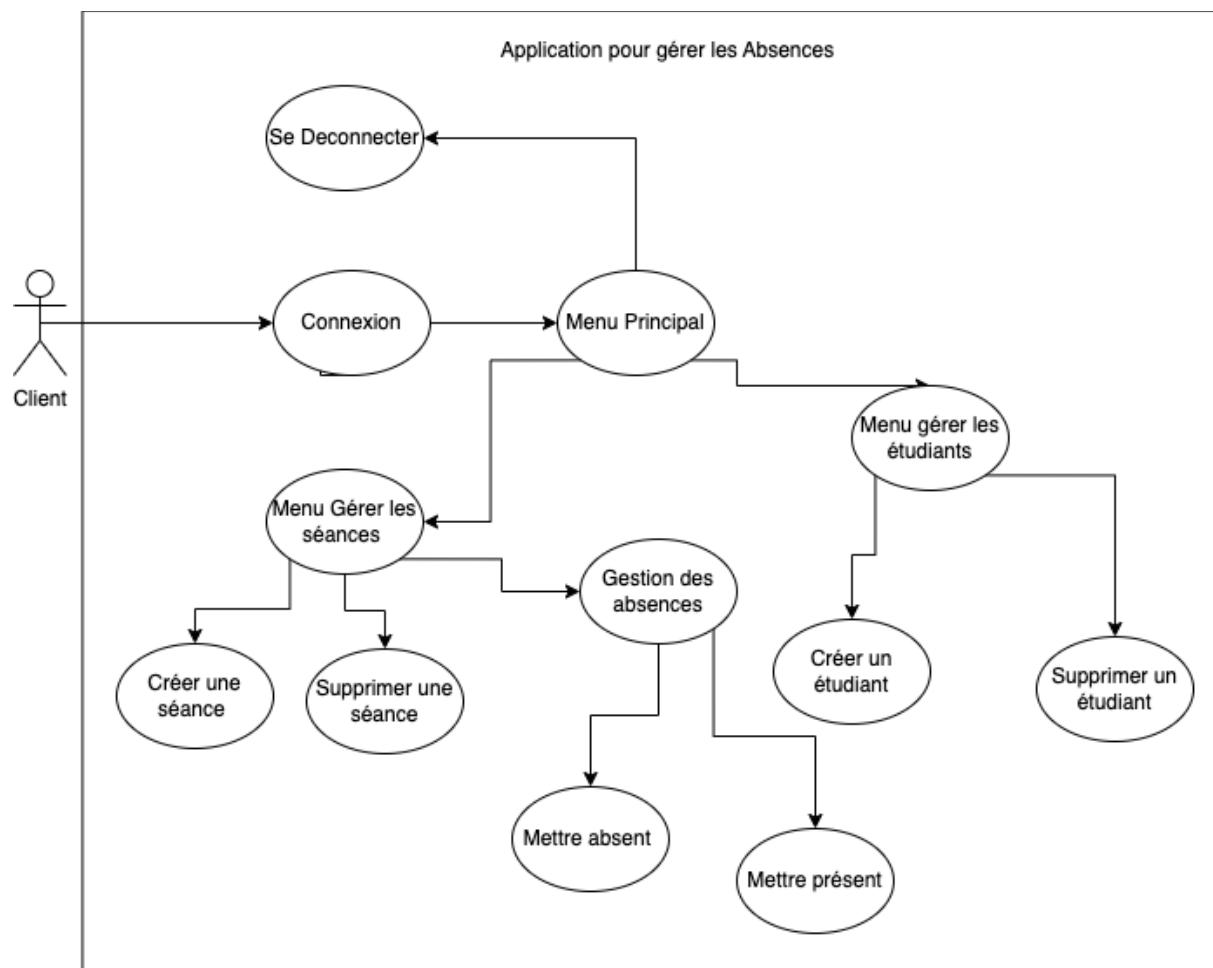
Un programme Python fonctionnant sur PC et Mac permettant une interaction en interface graphique

Python CLI - DEV

Un terminal créé à des fins de développement permettant d'interagir [directement avec le protocole](#).

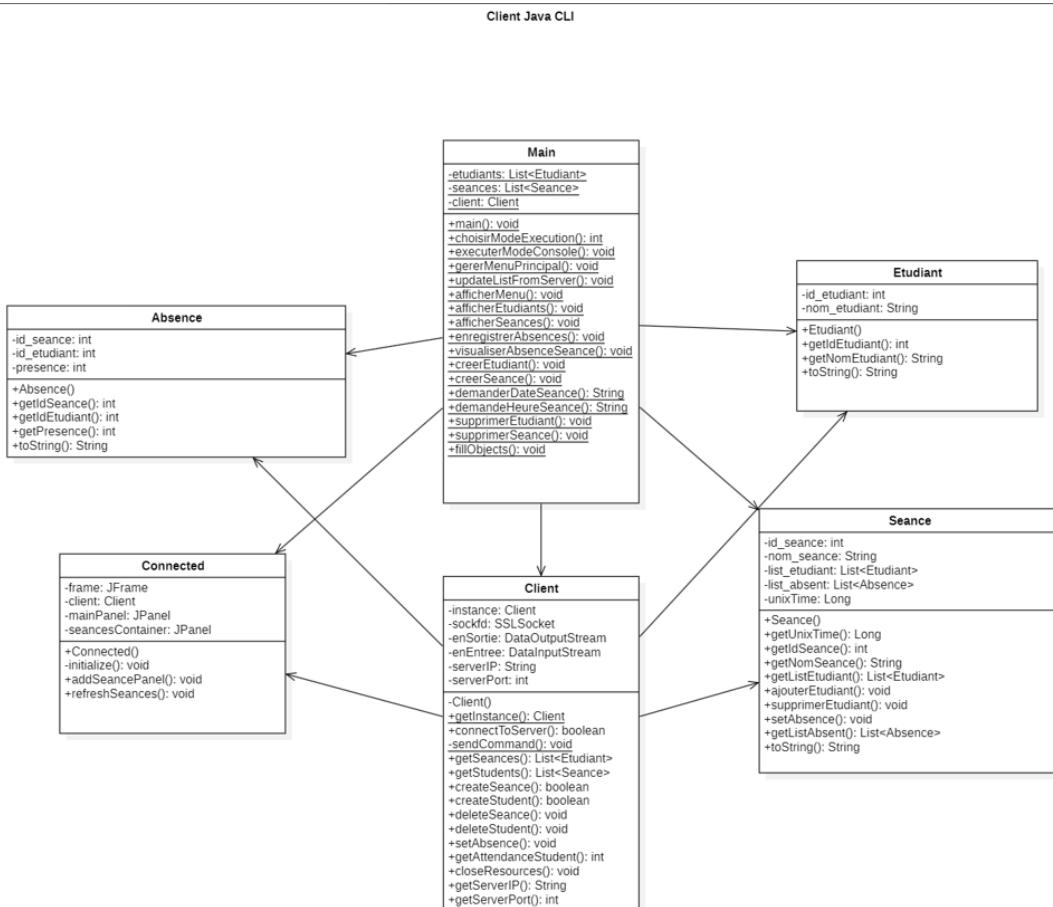
2/ Diagramme de cas d'utilisation

Voici le diagramme de cas d'utilisation pour gérer les absences. Ce diagramme a pour but de représenter le comportement de l'application lorsqu'un utilisateur l'utilise.



2.2/ Diagramme de classes

Le diagramme de classe représente les différentes classes implémentées en Java. Ainsi, vous pouvez vous rendre compte des classes avec leurs attributs qui sont implémentés dans l'application ainsi que leur méthode.



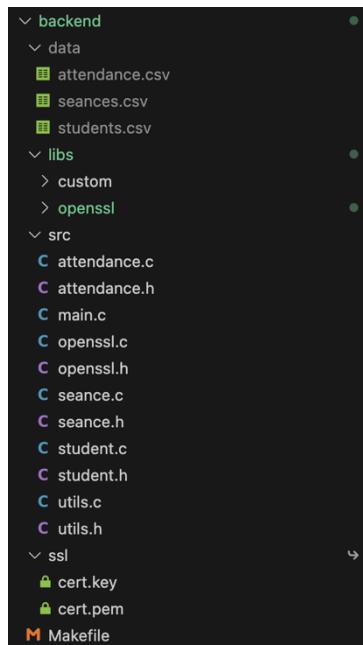
3/ Conception du backend

Le backend a été réalisé en C, il s'occupe de [traiter les requêtes](#) en utilisant le [protocole](#) présenté précédemment.

Structure

Le projet se structure de la manière suivante:

Le dossier data stocke les fichiers nécessaires à la [gestion des données](#), le dossier src répertorie les fichiers .c et .h [du code](#), le dossier ssl stocke les clés du certificat, et enfin le dossier libs contient les [librairies nécessaires](#).



Gestion des données

Les données sont stockées dans [trois CSV](#) nommés “attendance.csv”, “seances.csv”, “students.csv”.

students.csv	seances.csv	attendance.csv
<pre>× ./backend/data (-zsh) > cat students.csv 3,Valentin PUCETTI 0,Corentin BONNEAU 1,François GOYA 2,Mathieu BERSIN</pre>	<pre>× ./backend/data (-zsh) > cat seances.csv id,name,unix_time 0,CM - Mathématiques,1738251000 1,TP - Fibre optique,1738155600</pre>	<pre>× ./backend/data (-zsh) > cat attendance.csv seance_id,student_id,status 1,1,1 0,0,1 0,1,1 0,2,1</pre>
Stocke les étudiants :	Stocke les séances :	Stocke les présences :
<ul style="list-style-type: none">- ID- Nom complet	<ul style="list-style-type: none">- ID- Nom- Unix time	<ul style="list-style-type: none">- ID de la séance- ID de l'étudiant- Status de présence

Démarrage des éléments

En lançant l'exécutable du serveur, de nombreuses actions sont effectuées. Ci-dessous est représentée la fonction main, celle lancée en tout premier dans le programme. Elle attend en paramètre le port, et vérifie s'il est valide

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "[USAGE] %s <port>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int port = atoi(argv[1]);
    if (port <= 0 || port > 65535) {
        fprintf(stderr, "[ERROR] Invalid port number. Must be between 1 and 65535.\n");
        exit(EXIT_FAILURE);
    }

    init_openssl();

    if (!init_db()) {
        fprintf(stderr, "[ERROR] Failed to initialize the database.\n");
        cleanup_openssl();
        exit(EXIT_FAILURE);
    }

    signal(SIGINT, cleanup_and_exit);

    initialize_server(port);

    while (1) {
        struct sockaddr_in address;
        socklen_t addrlen = sizeof(address);
        int* client_socket = malloc(sizeof(int));
        *client_socket = accept(server_socket, (struct sockaddr*)&address, &addrlen);
        if (*client_socket < 0) {
            perror("[ERROR] Accept failed");
            free(client_socket);
            continue;
        }

        pthread_t thread_id;
        if (pthread_create(&thread_id, NULL, client_handler, client_socket) != 0) {
            perror("[ERROR] Thread creation failed");
            close(*client_socket);
            free(client_socket);
        } else {
            pthread_detach(thread_id);
        }
    }

    return 0;
}
```

3.1/ Initialisation du chiffrement

Dans un premier temps, le programme appelle la fonction `init_openssl()` pour charger la [librairie de chiffrement](#).

```
// Function to initialize OpenSSL
Pieces: Comment | Pieces: Explain
void init_openssl() {
    SSL_library_init();
    SSL_load_error_strings();
    OpenSSL_add_all_algorithms();

    ctx = SSL_CTX_new(TLS_server_method());
    if (!ctx) {
        perror("[ERROR] Unable to create SSL context");
        exit(EXIT_FAILURE);
    }

    // Load your certificate and private key
    if (SSL_CTX_use_certificate_file(ctx, "./ssl/cert.pem", SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    if (SSL_CTX_use_PrivateKey_file(ctx, "./ssl/cert.key", SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }
}
```

Cette fonction initialise la librairie avec les premières fonctions appelées, puis le [contexte SSL](#) pour la suite des échanges.

Si celui-ci n'est pas initié, le programme [retourne une erreur](#).

Ensuite, on fournit les fichiers du certificats ([privkey & pubkey](#)) nécessaires au bon fonctionnement de ce contexte, qui [rejette une erreur](#) si non trouvé, ou non valide.

3.2/ Initialisation de la base de données

Ensuite, le programme vérifie si la base de données est initialisée par la fonction `init_db()` présente dans le fichier `utils.c`

```
// Initialize CSV files with headers if they don't exist
Pieces: Comment | Pieces: Explain
bool init_db() {
    FILE *file;

    // Initialize students.csv
    file = fopen(STUDENTS_FILE, "r");
    if (!file) {
        file = fopen(STUDENTS_FILE, "w");
        if (!file) {
            perror("[ERROR] Unable to create students.csv");
            return false;
        }
        fprintf(file, "id,name\n");
    }
    fclose(file);

    // Initialize seances.csv
    file = fopen(SEANCES_FILE, "r");
    if (!file) {
        file = fopen(SEANCES_FILE, "w");
        if (!file) {
            perror("[ERROR] Unable to create seances.csv");
            return false;
        }
        fprintf(file, "id,name,unix_time\n");
    }
    fclose(file);

    // Initialize attendance.csv
    file = fopen(ATTENDANCE_FILE, "r");
    if (!file) {
        file = fopen(ATTENDANCE_FILE, "w");
        if (!file) {
            perror("[ERROR] Unable to create attendance.csv");
            return false;
        }
        fprintf(file, "seance_id,student_id,status\n");
    }
    fclose(file);

    return true;
}
```

Dans un premier temps on définit la [structure des dossiers](#) et des fichiers

```
#define STUDENTS_FILE "data/students.csv"
#define SEANCES_FILE "data/seances.csv"
#define ATTENDANCE_FILE "data/attendance.csv"
```

Ensuite, la fonction `init_db()` vérifie pour chaque fichier [s'ils existent](#).

Si ceux-ci n'existent pas, ils sont créés avec une entête CSV mentionnant les colonnes

Enfin, la fonction retourne true pour indiquer au `main()` que les données sont prêtes à être insérer

1) Signal handler

Pour permettre un arrêt propre du socket et du programme lors ce qu'un sigint est envoyé (par exemple en faisant ctrl+c), on lui définit la fonction à exécuter à ce moment.

```
// Cleanup function for graceful shutdown
void cleanup_and_exit() {
    cleanup_openssl();
    close(server_socket);
    printf("[INFO] Server is shutting down...\n");
    exit(0);
}
```

Celle-ci nous permet de libérer le port du socket à l'extinction, et d'effacer le contexte openssl.

3.4/ Initialisation du socket

Nous pouvons maintenant initier le socket, pour accepter les connexions entrantes à travers la fonction `initialize_server()`

```
void initialize_server(int port) {
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket < 0) {
        perror("[ERROR] Socket creation failed");
        exit(EXIT_FAILURE);
    }

    int opt = 1;
    if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) < 0) {
        perror("[ERROR] setsockopt failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in address;
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(port);

    if (bind(server_socket, (struct sockaddr*)&address, sizeof(address)) < 0) {
        perror("[ERROR] Bind failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    if (listen(server_socket, 3) < 0) {
        perror("[ERROR] Listen failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    printf("[INFO] Server is listening on port %d...\n", port);
}
```

Dans un premier on crée le socket (en IPv4, et TCP grâce aux paramètres `AF_INET` et `SOCK_STREAM`) puis on catch si une erreur a eu lieu lors de sa création.

On définit l'option `SO_REUSEADDR` pour réutiliser l'adresse locale

Ensuite, on définit sur quel type d'adresse, l'adresse et le port sur lesquels on écoute.

Enfin, on attache le socket sur les adresses et port en question

Pour terminer, on met le socket en mode écoute avec un back log de 3 connections

2) Acceptation des connexions et “sessions”

Enfin, le serveur attend et accepte les connexions des clients, la fonction `accept` attend une connexion entrante et retourne un descripteur de socket pour la connexion client.

Si `accept` échoue, un message d'erreur est affiché et la boucle continue pour attendre la prochaine connexion. Si il réussit, la fonction `client_handler()` est appelée dans un nouveau Thread pour gérer la connexion client et ses requêtes.

```
// Function to handle each client in a separate thread
void* client_handler(void* arg) {
    int client_socket = *(int*)arg;
    free(arg);

    struct sockaddr_in address;
    socklen_t addrlen = sizeof(address);
    getpeername(client_socket, (struct sockaddr*)&address, &addrlen);

    handle_ssl_client(client_socket, address);
    return NULL;
}
```

La fonction `client_handler` est exécutée par chaque thread créé pour gérer les connexions des clients. Elle extrait le descripteur de socket du client, récupère les informations d'adresse du client et délègue la gestion de la connexion SSL à la fonction `handle_ssl_client`. Cela permet au serveur de gérer plusieurs connexions clients simultanément dans des threads séparés

Déchiffrement et établissement des sessions

La fonction `handle_ssl_client()` du fichier `openssl.c` gère la [communication sécurisée](#) avec un client en utilisant SSL.

```
// Function to handle client connections with SSL
Pieces: Comment | Pieces: Explain
void handle_ssl_client(int client_socket, struct sockaddr_in client_addr) {
    SSL *ssl = SSL_new(ctx);
    SSL_set_fd(ssl, client_socket);

    if (SSL_accept(ssl) <= 0) {
        ERR_print_errors_fp(stderr);
    } else {
        char buffer[BUFFER_SIZE] = {0};
        char client_ip[INET_ADDRSTRLEN];

        inet_ntop(AF_INET, &client_addr.sin_addr, client_ip, INET_ADDRSTRLEN);
        printf("[%s] Connected\n", client_ip);

        while (1) {
            memset(buffer, 0, BUFFER_SIZE);
            int bytes_read = SSL_read(ssl, buffer, BUFFER_SIZE - 1);
            if (bytes_read <= 0) {
                printf("[%s] Disconnected\n", client_ip);
                break;
            }

            // We need to pass the SSL object to handle_request
            handle_request(ssl, buffer, client_ip);
        }
    }

    SSL_shutdown(ssl);
    SSL_free(ssl);
    close(client_socket);
}
```

Tout d'abord, un nouvel objet SSL est créé avec `SSL_new()` et associé au descripteur de socket du client.

Ensuite, une [connexion SSL avec le client est établie](#) en appelant `SSL_accept()`. Si la connexion est réussie, l'adresse IP du client est convertie avec `inet_ntop` et un message de connexion est affiché.

La fonction entre ensuite dans une boucle où elle récupère et [lit les données envoyées par le client](#) dans un buffer en utilisant `SSL_read()`. Si la lecture échoue, un message de déconnexion est affiché et la boucle se termine.

Les données lues sont [ensuite traitées par la fonction `handle_request\(\)`](#). Enfin, la connexion SSL est fermée proprement avec `SSL_shutdown()`, les ressources associées sont libérées avec `SSL_free()`, et le socket du client est fermé avec `close()`.

Gestion des requêtes

Trie des status

Une fois la requête déchiffrée et lisible, nous allons regarder s'il s'agit d'un [POST](#), [GET](#), [DELETE](#) etc..

```
// Function to handle client requests and delegate to appropriate handlers
Pieces: Comment | Pieces: Explain
void handle_request(SSL *ssl, const char* request, const char* client_ip) {
    if (strncmp(request, "<g>", 3) == 0) {
        printf("[%s] GET Request: %s\n", client_ip, request + 3);
        handle_get(ssl, request + 3);
    } else if (strncmp(request, "<p>", 3) == 0) {
        printf("[%s] POST Request: %s\n", client_ip, request + 3);
        handle_post(ssl, request + 3);
    } else if (strncmp(request, "<a>", 3) == 0) {
        printf("[%s] ACTION Request: %s\n", client_ip, request + 3);
        handle_action(ssl, request + 3);
    } else if (strncmp(request, "<d>", 3) == 0) {
        printf("[%s] DELETE Request: %s\n", client_ip, request + 3);
        handle_delete(ssl, request + 3);
    } else {
        printf("[%s] Invalid request format: %s\n", client_ip, request);
        SSL_write(ssl, RESPONSE_BAD_REQUEST, strlen(RESPONSE_BAD_REQUEST));
    }
}
```

Le rôle de cette fonction qui est appelée après l'établissement de la session est de répartir les requêtes reçues vers les différents handlers.

Elle compare les premiers caractères de la requête avec les entêtes définis, et ensuite la renvoie vers le handler définis. S'il ne sait pas où la rediriger, il retourne une bad request.

Exécution des fonctions liés aux requêtes

Pour chaque type de requête, nous disposons d'une [fonction similaire à celle ci-dessous](#) (exemple avec les requêtes GET)

```
// Function to handle GET requests
Pieces: Comment | Pieces: Explain
void handle_get(SSL *ssl, const char* content) {
    char response[BUFFER_SIZE];
    if (strncmp(content, "student", 7) == 0) {
        int id;
        if (sscanf(content + 8, "%d", &id) == 1) {
            char name[100];
            if (get_student(id, name)) {
                sprintf(response, sizeof(response), "202/id:%d,name:%s", id, name);
                SSL_write(ssl, response, strlen(response));
            } else {
                SSL_write(ssl, RESPONSE_ERROR, strlen(RESPONSE_ERROR));
            }
        } else {
            char list[BUFFER_SIZE] = "";
            if (get_student_list(list)) {
                sprintf(response, sizeof(response), "202/%s", list);
                SSL_write(ssl, response, strlen(response));
            } else {
                SSL_write(ssl, RESPONSE_ERROR, strlen(RESPONSE_ERROR));
            }
        }
    }
}
```

On vérifie si les premiers caractères [correspondent à un endpoint](#) (définis à l'introduction), s'il n'est pas reconnu on retourne une bad request.

Sinon on extrait les [paramètres](#) de la requête pour les passer en paramètre à la fonction assignée à l'action.

S'ils ne sont pas définis, on retourne une bad request. Si la fonction répond par une erreur, on retourne l'erreur.

Fonctions de créations

Pour la création des données, la fonction handle_post() est utilisée.

```
// Function to handle POST requests
Pieces: Comment | Pieces: Explain
void handle_post(SSL *ssl, const char* content) {
    if (strcmp(content, "student/", 8) == 0) {
        char name[100];
        if (sscanf(content + 8, "%99[^/\n]", name) == 1) {
            Student student;
            student.id = 0;
            char existing_name[100];
            while (get_student(student.id, existing_name)) {
                student.id++;
            }
            strncpy(student.name, name, sizeof(student.name) - 1);
            student.name[sizeof(student.name) - 1] = '\0';

            if (add_student(&student)) {
                char response[BUFFER_SIZE];
                snprintf(response, sizeof(response), "202/id:%d,name:%s", student.id, student.name);
                SSL_write(ssl, response, strlen(response));
            } else {
                SSL_write(ssl, RESPONSE_BAD_REQUEST, strlen(RESPONSE_BAD_REQUEST));
            }
        } else {
            SSL_write(ssl, RESPONSE_BAD_REQUEST, strlen(RESPONSE_BAD_REQUEST));
        }
    } else if (strcmp(content, "attendance/", 11) == 0) {
```

Elle utilise la même logique que la fonction handle_get() pour les endpoint, cependant elle crée les objets (struct) des données et utilise un identifiant unique.

Celui-ci est trouvé à l'aide d'une simple boucle while qui s'incrémente, et la fonction get_student().

Voici les différents structs utilisés pour le stockage des données :

```
typedef struct {
    int id;
    char name[100];
} Student;
```

student.h

```
typedef struct {
    int id;
    char name[100];
    int unix_time;
} Seance;
```

seance.h

```
typedef struct {
    int seance_id;
    int student_id;
    int status;
} Attendance;
```

attendance.h

Manipulation des données

Nous pouvons donc exécuter des fonctions à partir des requêtes. Maintenant, nous devons nous occuper de [créer les données, les récupérer, les supprimer](#) etc...

Création d'un élément

Pour créer un élément (dans notre exemple nous allons utiliser les étudiants), nous utilisons la fonction suivante.

```
// Function to add a student to the database
Pieces: Comment | Pieces: Explain
bool add_student(Student* student) {
    FILE *file = fopen(STUDENTS_FILE, "a+");
    if (!file) return false;

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        int existing_id;
        if (sscanf(line, "%d", &existing_id) == 1 && existing_id == student->id) {
            fclose(file);
            return false;
        }
    }

    fprintf(file, "%d,%s\n", student->id, student->name);
    fclose(file);
    return true;
}
```

Celle-ci commence par ouvrir le fichier CSV en mode [ajout et lecture \(a+\)](#). Si le fichier ne peut pas être ouvert, la fonction retourne false.

Ensuite, pour chaque ligne du fichier elle utilise sscanf pour [extraire l'identifiant](#) existant et le compare avec celui de l'étudiant à ajouter pour vérifier l'id existe déjà. Si oui, le fichier est fermé et la fonction retourne false, sinon [l'étudiant est ajouté](#) en écrivant son id et son nom dans le fichier au format CSV (%d,%s\n).

Le fichier est ensuite fermé et la fonction retourne true, indiquant que l'étudiant a été ajouté.

Liste de plusieurs éléments

Pour lister plusieurs éléments tels que la liste des étudiants par exemple.

```

// Function to get a list of students from the database
Pieces: Comment | Pieces: Explain
bool get_student_list(char* result) {
    FILE *file = fopen(STUDENTS_FILE, "r");
    if (!file) return false;

    char line[256];
    *result = '\0';

    while (fgets(line, sizeof(line), file)) {
        int id;
        char name[100];
        if (sscanf(line, "%d,%99[^\\n]", &id, name) == 2) {
            sprintf(result + strlen(result), "id:%d,name:%s;", id, name);
        }
    }

    fclose(file);
    return true;
}

```

La fonction `get_student_list()` récupère la [liste de tous les étudiants](#) stockés dans un fichier et les formate dans une chaîne de caractères.

Elle commence par ouvrir le CSV en mode lecture (r) et initialiser [la chaîne de résultat result](#).

La fonction lit ensuite chaque ligne du fichier en utilisant `fgets()`. Pour chaque ligne, elle utilise `sscanf()` pour extraire l'identifiant (id) et le nom de l'étudiant. Si l'extraction réussit, elle ajoute ces informations à la chaîne `result` en utilisant `sprintf()`, [en formatant sous la forme id:,name::](#).

Après avoir lu toutes les lignes, le fichier est fermé et la fonction retourne `true`, indiquant que la liste des étudiants a été récupérée avec [succès](#).

Suppression d'un élément

Maintenant, nous allons expliquer comment un élément est supprimé.

```
// Function to delete a student from the database
Pieces: Comment | Pieces: Explain
bool delete_student(int id) {
    FILE *file = fopen(STUDENTS_FILE, "r");
    if (!file) return false;

    FILE *temp_file = fopen("data/temp_students.csv", "w");
    if (!temp_file) {
        fclose(file);
        return false;
    }

    char line[256];
    bool found = false;

    while (fgets(line, sizeof(line), file)) {
        int current_id;
        sscanf(line, "%d", &current_id);
        if (current_id == id) {
            found = true;
            continue;
        }
        fprintf(temp_file, "%s", line);
    }

    fclose(file);
    fclose(temp_file);

    remove(STUDENTS_FILE);
    rename("data/temp_students.csv", STUDENTS_FILE);
    delete_student_attendance(id);

    return found;
}
```

Elle commence par ouvrir le CSV en mode lecture (r). Ensuite, elle ouvre un [fichier temporaire](#) en mode écriture (w) pour stocker les données des [séances restantes](#).

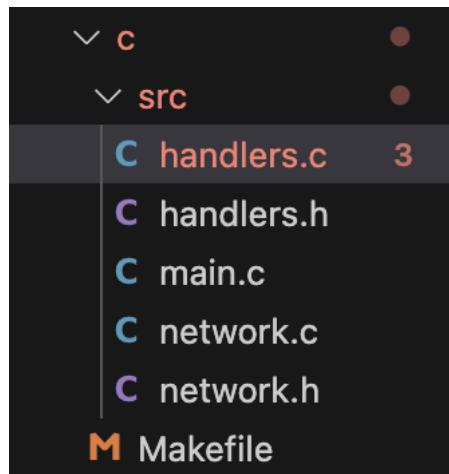
Elle lit ensuite [chaque ligne du fichier](#) original en utilisant fgets. Pour chaque ligne, elle utilise sscanf() pour extraire l'identifiant de la séance (current_id). Si l'identifiant extrait correspond à l'[identifiant de la séance à supprimer](#) (id), la ligne est ignorée et la variable found est mise à true. Sinon, la ligne est écrite dans le fichier temporaire. Pour finir, les fichiers original et temporaire sont fermés, et le [temporaire remplace l'original](#).

La fonction delete_student_attendance() est ensuite appelée pour [supprimer les présences existantes](#) de l'étudiant supprimé.

4/ Le client en C

Dans le cadre de ce projet de SAE, nous avons conçu un client en C en nous appuyant sur les compétences acquises lors de nos cours à l'IUT. Ce client utilise une interface en ligne de commande, ce qui signifie qu'il ne dispose pas d'une interface graphique et que l'utilisateur interagit avec l'application exclusivement via un terminal comme PowerShell, Bash ou d'autres.

Notre partie client est répartie sur plusieurs fichiers C :



4.2/ Les fichiers handlers :

Le fichier handlers est contient l'implémentation des fonctions d'affichage. On y trouve également une structure de données locale pour représenter un élément (Item) utilisé dans l'affichage des listes. Il implémente aussi les gestionnaires d'actions. Il envoie une requête au serveur pour récupérer les données (liste des étudiants, séances ou absences), il affiche des données à l'aide des fonctions du module display, il lit des choix de l'utilisateur et envoi des commandes correspondantes au serveur. Comme pour display nous allons importer les bibliothèques.

Tout d'abord on commence par inclure les bibliothèques :

- `#include <stdio.h>` : Cette directive inclut la bibliothèque standard d'entrées/sorties du langage C. Utilisation : Elle fournit des fonctions telles que `printf`, `scanf`, `fgets`, etc., qui permettent d'afficher des informations à l'écran ou de lire des entrées depuis le clavier.
- `#include <stdlib.h>` : Cette directive inclut la bibliothèque standard du C qui contient des fonctions utilitaires générales. On y trouve des fonctions pour la gestion de la mémoire (comme `malloc`, `free`), le contrôle de l'exécution (comme `exit`) et des conversions de types (comme `atoi`).

- #include <string.h> : Cette directive permet d'inclure les fonctions de manipulation de chaînes de caractères. Des fonctions comme strlen, strcpy, strcat, strcmp, et strstr sont disponibles pour travailler avec des chaînes de caractères.
- #include <time.h> : Cette directive inclut les fonctions de gestion du temps en C. Elle fournit des fonctions pour obtenir l'heure actuelle, manipuler des dates et heures (comme time, localtime, mktime, et strftime), ce qui est utile lorsqu'il faut afficher ou convertir des timestamps.
- #define MAX_ITEMS 50 : Cette directive définit une constante symbolique nommée MAX_ITEMS avec la valeur 50. Elle est utilisée pour spécifier le nombre maximal d'éléments (par exemple, des étudiants ou des séances) que le programme peut gérer ou stocker dans un tableau. Ainsi, dans le code, lorsqu'on déclare un tableau d'items, on peut l'écrire comme Item items[MAX_ITEMS], ce qui rend le code plus lisible et facile à modifier ultérieurement si la limite devait changer.

```
typedef struct {
    int id;
    char name[100];
    time_t unix_time;
} Item;
```

Nous avons une fonction qui affiche un titre entouré d'un cadre graphique avec des caractères spéciaux. La longueur du cadre s'adapte dynamiquement à la longueur du titre.

```
void display_title(const char* title) {
    int len = strlen(title);
    printf("\n==");
    for(int i = 0; i < len + 2; i++) printf("=");
    printf("==\n");
    printf("|| %s ||\n", title);
    printf("==");
    for(int i = 0; i < len + 2; i++) printf("=");
    printf("==\n\n");
}
```

Exemple de résultat de cette fonction :



La fonction parse_server_response permet d'extraire les informations envoyées par le serveur à partir d'une chaîne de caractères. La réponse attendue commence par un statut "202". Après le statut, les données sont séparées par des ;, chaque segment représentant un item. Pour chaque item, la fonction recherche des sous-chaînes telles que "id:", "name:" et éventuellement "unix_time:". sscanf est utilisé pour extraire les valeurs dans la structure Item. La fonction s'arrête après MAX_ITEMS (50) éléments.

```
static void parse_server_response(const char* response, Item* items, int* count) {
    *count = 0;
    char* resp_copy = strdup(response);
    char* status = strtok(resp_copy, "/");
    if (strcmp(status, "202") == 0) {
        char* data = strtok(NULL, "");
        char* item = strtok(data, ";");
        while (item != NULL && *count < MAX_ITEMS) {
            char* id_str = strstr(item, "id:");
            char* name_str = strstr(item, "name:");
            char* time_str = strstr(item, "unix_time:");
            if (id_str && name_str) {
                sscanf(id_str, "id:%d", &items[*count].id);
                char name[100] = {0};
                if (time_str) {
                    sscanf(name_str, "name:[^,]", name);
                    sscanf(time_str, "unix_time:%ld", &items[*count].unix_time);
                } else {
                    sscanf(name_str, "name:[^;]", name);
                    items[*count].unix_time = 0;
                }
                strcpy(items[*count].name, name);
                (*count)++;
            }
            item = strtok(NULL, ";");
        }
        free(resp_copy);
    }
}
```

Nous avons créé 3 Fonctions d'affichage des listes :

```

void display_seance_list(const char* response) {
    Item seances[MAX_ITEMS];
    int count;
    parse_server_response(response, seances, &count);

    display_title("Seance Management");

    for (int i = 0; i < count; i++) {
        struct tm *tm_info = localtime(&seances[i].unix_time);
        char time_str[20];
        strftime(time_str, 20, "%d/%m/%Y %H:%M", tm_info);
        printf("%d. %s (%s)\n",
               seances[i].id,
               seances[i].name,
               time_str);
    }

    printf("\nOptions:\n");
    printf("1. Go back\n");
    printf("2. Create seance\n");
    printf("3. Delete seance\n");
}

void display_attendance_list(const char* response) {
    Item students[MAX_ITEMS];
    int count;
    parse_server_response(response, students, &count);

    display_title("Attendance Management");

    for (int i = 0; i < count; i++) {
        printf("(%d) %s\n", students[i].id, students[i].name);
    }
    printf("\nOptions:\n");
    printf("1. Go back\n");
    printf("2. Take attendance\n");
    printf("Choice: ");
}

void display_student_list(const char* response) {
    Item students[MAX_ITEMS];
    int count;
    parse_server_response(response, students, &count);

    display_title("Student Management");

    for (int i = 0; i < count; i++) {
        printf("(%d) %s\n", students[i].id, students[i].name);
    }

    printf("\nOptions:\n");
    printf("1. Go back\n");
    printf("2. Create new student\n");
    printf("3. Delete student\n");
}

```

Ces 3 fonctions permettent d'afficher un titre "" en utilisant la fonction `display_title`. Utilise `parse_server_response` pour remplir un tableau (étudiants séances attendance) , Affiche chaque étudiant sous la forme (id) nom, Affiche chaque séance avec son id, son nom et la date/heure correspondante, pour la partie séance on convertit le timestamp (unix_time) en date/heure lisible grâce à `localtime` et `strftime`. Affiche ensuite les options disponibles pour la gestion des étudiants ou des séances ou des attendances .

#define BUFFER_SIZE 1024 : Cette directive définit une constante symbolique nommée BUFFER_SIZE avec une valeur de 1024. Elle est utilisée pour allouer des buffers de taille fixe pour le stockage temporaire des données, par exemple lors de la réception des réponses du serveur via la connexion réseau. Cela permet de s'assurer que les buffers utilisés ont une taille cohérente et facilite la maintenance du code.

Nous avons créé 3 fonctions avec un processus similaires :

- Pour la fonction void handle_student_management(SSL* ssl), elle boucle jusqu'à ce que l'utilisateur choisisse l'option "Go back" (option 1). La commande "<g>student" demande la liste des étudiants. En fonction du choix de l'utilisateur : Option 2 : Créer un nouvel étudiant. Le nom est récupéré via fgets, puis une commande de type "<p>student/<nom>" est envoyée. Option 3 : Supprimer un étudiant. L'identifiant est saisi et une commande de suppression "<d>student/<id>" est envoyée. Après chaque action, la liste est rafraîchie en renvoyant la commande de récupération.

```
void handle_student_management(SSL* ssl) {
    char buffer[BUFFER_SIZE];
    char response[BUFFER_SIZE];
    int choice = 0;

    while (choice != 1) {
        ssl_send_receive(ssl, "<g>student", response, sizeof(response));
        display_student_list(response);

        scanf("%d", &choice);
        getchar();

        if (choice == 2) {
            printf("Enter student name: ");
            char name[100];
            fgets(name, sizeof(name), stdin);
            name[strcspn(name, "\n")] = 0;

            sprintf(buffer, sizeof(buffer), "<p>student/%s", name);
            ssl_send_receive(ssl, buffer, response, sizeof(response));
            display_student_list(response);
        }
        if (choice == 3) {
            printf("Enter student id: ");
            int id;
            scanf("%d", &id);
            getchar();

            sprintf(buffer, sizeof(buffer), "<d>student/%d", id);
            ssl_send_receive(ssl, buffer, response, sizeof(response));
            display_student_list(response);
        }
    }
}
```

- Pour la fonction void handle_seance_management(SSL* ssl), on crée une boucle d'affichage et de gestion jusqu'à l'option "Go back" (1). Récupère la liste des séances avec "seance//". est envoyée. Option 3 : Suppression d'une séance. L'utilisateur saisit l'identifiant, puis la commande "seance//". est envoyée. La liste est rafraîchie après chaque modification.

```

void handle_seance_management(SSL* ssl) {
    char buffer[BUFFER_SIZE];
    char response[BUFFER_SIZE];
    int choice = 0;

    while (choice != 1) {
        ssl_send_receive(ssl, "<g>seance", response, sizeof(response));
        display_seance_list(response);

        scanf("%d", &choice);
        getchar();
        if (choice == 2) {
            char name[100], date[20], time[10];

            printf("Enter seance name: ");
            fgets(name, sizeof(name), stdin);
            name[strcspn(name, "\n")] = 0;

            printf("Enter date (DD/MM/YYYY): ");
            fgets(date, sizeof(date), stdin);
            date[strcspn(date, "\n")] = 0;

            printf("Enter time (HH:MM): ");
            fgets(time, sizeof(time), stdin);
            time[strcspn(time, "\n")] = 0;

            struct tm tm = {0};
            char datetime[50];
            snprintf(datetime, sizeof(datetime), "%s %s", date, time);

            if (strptime(datetime, "%d/%m/%Y %H:%M", &tm) != NULL) {
                time_t unix_time = mktime(&tm);
                snprintf(buffer, sizeof(buffer), "<p>seance/%s/%ld", name, unix_time);
                ssl_send_receive(ssl, buffer, response, sizeof(response));

                ssl_send_receive(ssl, "<g>seance", response, sizeof(response));
                display_seance_list(response);
            }
        }
    }
}

```

- Pour la fonction void void handle_attendance_management(SSL* ssl), tout comme les deux autres, On crée deux tableaux de caractères (buffer et response) qui sont utilisés pour préparer et recevoir les messages envoyés/recus via SSL. L'utilisateur est invité à entrer l'identifiant d'une séance. Cet identifiant sera utilisé pour récupérer et mettre à jour l'assiduité des étudiants pour cette séance. La fonction prépare une commande (ici la chaîne "<g>seance") qui est envoyée au serveur via ssl_send_receive. Le préfixe <g> semble indiquer une requête de type "get" (récupération). La réponse reçue est ensuite analysée par la fonction parse_server_response qui extrait dans un tableau seances les informations sur les séances (par exemple, l'ID, le nom, et éventuellement la date). Une boucle parcourt les séances récupérées pour trouver celle dont l'ID correspond à celui entré par l'utilisateur. Si elle est trouvée, la variable seance_name est mise à jour avec le nom réel de la séance ; sinon, elle reste "Unknown Seance". La commande "<g>student" est envoyée pour obtenir la liste de tous les étudiants. La réponse est analysée par parse_server_response pour remplir un tableau students et déterminer le nombre total d'étudiants (count). Pour chaque étudiant, on construit une commande qui ressemble à "<g>attendance/<seance_id>/<student_id>". Cette commande demande au serveur l'état d'assiduité (par exemple, présent ou absent) de l'étudiant pour la séance spécifiée. La réponse pour chaque étudiant est analysée par parse_attendance_response, qui extrait le statut (stocké dans status). Ce statut est ensuite sauvégarde dans le tableau student_statuses au même indice que l'étudiant correspondant. La fonction display_seance_attendance_status est appelée pour afficher le nom de la séance, la liste des étudiants et leur statut d'assiduité correspondant. On peut imaginer que cette fonction affiche, par exemple, pour chaque étudiant son nom, son ID et s'il est marqué « present » ou « absent ». Le programme demande à l'utilisateur de saisir, pour chaque étudiant, le nouveau statut d'assiduité (1 pour présent, 0 pour absent).
- Pour chaque étudiant, le programme affiche son nom, son ID et son état actuel (en affichant « Present » ou « Absent »).
- Pour chaque étudiant, après avoir saisi le nouveau statut, une commande est construite avec le format "<p>attendance/<seance_id>/<student_id>/<status>". Le préfixe <p> indique ici

une opération de type « post » (mise à jour ou modification). La commande est envoyée au serveur pour mettre à jour l'état d'assiduité de l'étudiant pour la séance en cours.

Une fois la boucle terminée (c'est-à-dire que tous les étudiants ont été traités), un message confirme que l'assiduité a été mise à jour pour tous les étudiants.

```
void handle_attendance_management(SSL* ssl) {
    char buffer[BUFFER_SIZE];
    char response[BUFFER_SIZE];
    int choice = 0;
    int seance_id;

    printf("Enter seance id: ");
    scanf("%d", &seance_id);
    getchar();

    snprintf(buffer, sizeof(buffer), "<g>seance", response);
    ssl_send_receive(ssl, buffer, response, sizeof(response));
    Item seances[MAX_ITEMS];
    int seance_count;
    parse_server_response(response, seances, &seance_count);

    char* seance_name = "Unknown Seance";
    for (int i = 0; i < seance_count; i++) {
        if (seances[i].id == seance_id) {
            seance_name = seances[i].name;
            break;
        }
    }

    while (1) {
        ssl_send_receive(ssl, "<g>student", response, sizeof(response));
        Item students[MAX_ITEMS];
        int count;
        parse_server_response(response, students, &count);

        int student_statuses[MAX_ITEMS] = {0};

        for (int i = 0; i < count; i++) {
            snprintf(buffer, sizeof(buffer), "<g>attendance/%d/%d", seance_id, students[i].id);
            ssl_send_receive(ssl, buffer, response, sizeof(response));
        }
    }
}
```

Le fichier handlers.h est un fichier d'en-tête qui déclare trois fonctions qui gèrent les interactions de l'utilisateur pour la gestion des étudiants (handle_student_management), la gestion des séances (handle_seance_management), la gestion des absences (handle_attendance_management).

Chaque fonction reçoit un pointeur SSL* qui représente la connexion sécurisée avec le serveur.

```

#ifndef HANDLERS_H
#define HANDLERS_H

#include <openssl/ssl.h>

void handle_student_management(SSL* ssl);
void handle_seance_management(SSL* ssl);
void handle_attendance_management(SSL* ssl);

#endif

```

4.3/ Les fichiers network :

Ce fichier networks.c est le module qui gère toute la communication réseau sécurisée de l'application. Il s'appuie sur OpenSSL pour établir des connexions chiffrées via SSL/TLS et utilise les appels système Unix pour manipuler les sockets. Nous allons détailler ici les parties essentielles du fichier et expliquer le rôle et le fonctionnement de chaque fonction.

Tout d'abord comme le reste de notre code nous implémentons les bibliothèques :

- #include <stdio.h> : Cette inclusion permet d'utiliser la bibliothèque standard d'entrées/sorties du langage C. Elle fournit des fonctions telles que printf, fprintf, scanf, etc., qui permettent d'afficher des informations à l'écran ou de lire des données à partir de l'entrée standard.
- #include <stdlib.h> : Cette inclusion intègre la bibliothèque standard du C pour les fonctions utilitaires. Elle offre des fonctions pour la gestion de la mémoire (comme malloc et free), le contrôle du flux du programme (exit), et diverses fonctions de conversion et de génération de nombres aléatoires.
- #include <string.h> : Cette inclusion permet d'utiliser les fonctions de manipulation de chaînes de caractères. Elle fournit des fonctions telles que strlen, strcpy, strncpy, strcmp, strstr, etc., qui facilitent le traitement des chaînes de caractères en C.
- #include <unistd.h> : Ce fichier d'en-tête est spécifique aux systèmes de type Unix (Linux, macOS, etc.) et fournit des fonctions pour l'interface système. Il offre des fonctions comme close, read, write, sleep, et d'autres appels système essentiels pour gérer des opérations de bas niveau, notamment la manipulation de fichiers et de descripteurs de fichiers.
- #include <sys/socket.h> : Ce fichier d'en-tête est utilisé pour la programmation réseau sous Unix. Il fournit les définitions et déclarations nécessaires pour créer et manipuler des sockets (points de communication), telles que les fonctions socket(), bind(), listen(), accept(), etc.

- #include <arpa/inet.h> : Cet en-tête est également lié à la programmation réseau et fournit des fonctions pour manipuler les adresses IP. Il contient des fonctions comme inet_addr ou inet_ntoa, qui permettent de convertir des adresses IP entre leurs représentations binaires et textuelles.
- #include <openssl/ssl.h> : Ce fichier d'en-tête fait partie de la bibliothèque OpenSSL et fournit les déclarations nécessaires pour utiliser les fonctionnalités SSL/TLS. Il permet de créer et de gérer des connexions sécurisées via SSL/TLS, en offrant des fonctions pour établir des connexions, chiffrer/déchiffrer des données et gérer le contexte SSL.
- #include <openssl/err.h> : Cet en-tête complète les fonctionnalités d'OpenSSL en permettant la gestion des erreurs. Il offre des fonctions pour obtenir des messages d'erreur détaillés en cas d'échec d'une opération SSL, facilitant ainsi le débogage et la gestion des erreurs dans une application utilisant OpenSSL.
- #include "network.h" : Cette inclusion concerne un fichier d'en-tête spécifique au projet (entre guillemets indique qu'il se trouve dans le répertoire local). Le fichier network.h contient les déclarations des fonctions et types utilisés pour la communication réseau dans le projet. Cela permet de centraliser la logique réseau (par exemple, l'établissement et la gestion de connexions SSL) et de faciliter leur utilisation dans le module source.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include "network.h"
```

Notre première fonction dans ce fichier a pour but de créer un contexte SSL en utilisant le mode client TLS. Il crée et initialise un contexte SSL, qui servira de base à toutes les opérations SSL de l'application.

- TLS_client_method() : Retourne un pointeur vers une méthode TLS pour un client. Cela indique à OpenSSL que nous allons établir une connexion en tant que client.
- SSL_CTX_new(method) : Crée un nouveau contexte SSL en utilisant la méthode spécifiée. Le contexte contient la configuration pour les connexions SSL, notamment les paramètres de chiffrement.

Si la création du contexte échoue, la fonction termine le programme en appelant exit(EXIT_FAILURE). Le contexte SSL (de type SSL_CTX) est indispensable pour gérer plusieurs connexions SSL. Il centralise la configuration et peut être partagé entre plusieurs connexions.

```
static SSL_CTX* create_context() {
    const SSL_METHOD *method = TLS_client_method();
    SSL_CTX *ctx = SSL_CTX_new(method);
    if (!ctx) {
        exit(EXIT_FAILURE);
    }
    return ctx;
}
```

Nous avons créé une fonction qui initialise la bibliothèque OpenSSL avant de l'utiliser.

- `SSL_load_error_strings()` : Charge les chaînes d'erreur OpenSSL. Cela permet d'obtenir des messages d'erreur compréhensibles en cas de problème lors des opérations SSL.
- `OpenSSL_add_ssl_algorithms()` : Charge l'ensemble des algorithmes cryptographiques nécessaires pour effectuer des opérations SSL/TLS (ex : chiffrement, déchiffrement, etc.).

```
static void init_openssl() {
    SSL_load_error_strings();
    OpenSSL_add_ssl_algorithms();
}
```

La fonction `cleanup_ssl` permet de libérer les ressources et algorithmes alloués par OpenSSL grâce à `EVP_cleanup`.

```
static void cleanup_openssl() {
    EVP_cleanup();
}
```

Pour créer le socket TCP et se connecter au serveur spécifié par son adresse IP (`hostname`) et le port nous avons créé la fonction `create_socket`. On crée la socket(`AF_INET`, `SOCK_STREAM`, 0) crée une socket pour IPv4 (`AF_INET`) utilisant le protocole TCP (`SOCK_STREAM`). En cas d'échec (socket négatif), le programme termine avec `exit(EXIT_FAILURE)`. Pour la configuration de l'adresse, une structure `sockaddr_in` est utilisée pour stocker l'adresse. `addr.sin_family` est mis à `AF_INET`. `addr.sin_port` est défini via `htons(port)`, convertissant le port en format réseau (big-endian). `addr.sin_addr.s_addr` est rempli par `inet_addr(hostname)`, qui convertit l'adresse IP de son format textuel (ex: "192.168.0.1") en format numérique. Pour le côté connexion, la fonction `connect()` tente d'établir une connexion avec le serveur en utilisant la structure d'adresse. Si la connexion échoue (retour différent de zéro), le programme s'arrête.

```

static int create_socket(const char* hostname, int port) {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = inet_addr(hostname);

    if (connect(sock, (struct sockaddr*)&addr, sizeof(addr)) != 0) {
        exit(EXIT_FAILURE);
    }

    return sock;
}

```

Avec cette fonction `setup_ssl_connection` nous mettons en place une connexion sécurisée SSL avec le serveur.

- Initialisation d'OpenSSL : Appel à `init_openssl()` pour préparer la bibliothèque. Utilisation de `create_context()` pour obtenir un contexte configuré pour un client TLS.
- `SSL_CTX_load_verify_locations(ctx, cert_file, NULL)` charge le certificat (ou la chaîne de certificats) utilisé pour vérifier l'identité du serveur. Si le chargement échoue (retour ≤ 0), le programme se termine.
- La fonction `create_socket()` établit la connexion TCP. `SSL_new(ctx)` crée un nouvel objet SSL à partir du contexte. `SSL_set_fd(ssl, sock)` associe la socket TCP à l'objet SSL.
- `SSL_connect(ssl)` initie la connexion SSL. Si cela échoue (retour ≤ 0), le programme se termine.
- L'objet SSL* est retourné et servira pour envoyer et recevoir des données de manière sécurisée.

Cette fonction encapsule toute la logique nécessaire pour passer d'une connexion TCP simple à une connexion sécurisée en SSL/TLS. Elle permet de s'assurer que toutes les étapes (initialisation, vérification du certificat, connexion, etc.) sont correctement effectuées avant de commencer la communication sécurisée.

```

SSL* setup_ssl_connection(const char* hostname, int port, const char* cert_file) {
    init_openssl();
    SSL_CTX *ctx = create_context();

    if (SSL_CTX_load_verify_locations(ctx, cert_file, NULL) <= 0) {
        exit(EXIT_FAILURE);
    }

    int sock = create_socket(hostname, port);
    SSL *ssl = SSL_new(ctx);
    SSL_set_fd(ssl, sock);

    if (SSL_connect(ssl) <= 0) {
        exit(EXIT_FAILURE);
    }

    return ssl;
}

```

On a du créer une fonction pour libérer toutes les ressources allouées pour la connexion SSL et fermer proprement la connexion car il est crucial de libérer toutes les ressources afin d'éviter les fuites de mémoire et de garantir que la connexion est proprement terminée. Cela permet également à l'application de fermer correctement la session sécurisée.

`SSL_get_fd(ssl)` récupère le descripteur de la socket associé. `SSL_get_SSL_CTX(ssl)` obtient le contexte SSL utilisé. `SSL_free(ssl)` libère l'objet SSL, ce qui inclut la fermeture de la connexion SSL. `close(sock)` ferme le descripteur de la socket. `SSL_CTX_free(ctx)` libère le contexte SSL. `cleanup_openssl()` libère les ressources OpenSSL utilisées par l'application.

```

void cleanup_ssl_connection(SSL* ssl) {
    int sock = SSL_get_fd(ssl);
    SSL_CTX *ctx = SSL_get_SSL_CTX(ssl);
    SSL_free(ssl);
    close(sock);
    SSL_CTX_free(ctx);
    cleanup_openssl();
}

```

Cette dernière fonction de notre fichier `network.c` va envoyer une requête au serveur via la connexion SSL et lire la réponse dans un buffer. Il envoie de la requête : `SSL_write(ssl, request, strlen(request))` envoie les données contenues dans `request` via la connexion SSL. `SSL_read(ssl, response, response_size - 1)` lit les données reçues depuis la connexion SSL et les stocke dans le buffer `response`. La taille du buffer est passée comme argument pour éviter un dépassement. La valeur `response_size - 1` est utilisée afin de laisser de la place pour le caractère nul `\0` qui termine la chaîne. `response[bytes] = '\0';` garantit que la chaîne est bien terminée et peut être manipulée en toute sécurité. Le nombre d'octets lus est retourné, ce qui peut être utile pour vérifier le succès de l'opération ou pour la gestion d'erreurs.

```

int ssl_send_receive(SSL* ssl, const char* request, char* response, int response_size) {
    SSL_write(ssl, request, strlen(request));
    int bytes = SSL_read(ssl, response, response_size - 1);
    response[bytes] = '\0';
    return bytes;
}

```

Le fichier network.h est un fichier d'en-tête qui déclare les fonctions gérant la connexion sécurisée SSL avec le serveur.

```

#ifndef NETWORK_H
#define NETWORK_H

#include <openssl/ssl.h>

SSL* setup_ssl_connection(const char* hostname, int port, const char* cert_file);
void cleanup_ssl_connection(SSL* ssl);
int ssl_send_receive(SSL* ssl, const char* request, char* response, int response_size);

#endif

```

4.4/ Le fichier main :

Le fichier main est le fichier principal de notre code client C,

Comme le reste de notre code nous allons importer les bibliothèques :

```

#include <stdio.h>
#include <stdlib.h>
#include "network.h"
#include "display.h"
#include "handlers.h"

void display_title(const char* title);

```

- #include <stdio.h>: il inclut la bibliothèque standard d'entrées/sorties du langage C. Cela fournit des fonctions comme printf, scanf, fprintf, etc., qui sont indispensables pour afficher des messages à l'écran ou lire des entrées de l'utilisateur.
- #include <stdlib.h> : Il inclut la bibliothèque standard du C pour les fonctions utilitaires. Cela fournit des fonctions pour la gestion de la mémoire (malloc, free), la gestion des processus (comme exit), et d'autres fonctions de conversion et de contrôle de flux.
- #include "network.h" Il inclut le fichier d'en-tête network.h qui est propre au projet. Cela contient les déclarations de fonctions et types nécessaires pour établir et gérer les connexions réseau sécurisées.

- `#include "handlers.h"` : Il inclut le fichier d'en-tête `handlers.h` spécifique au projet. Cela contient les déclarations des fonctions qui gèrent la logique applicative.

Cette dernière ligne déclare la fonction `display_title`, qui est responsable de l'affichage d'un titre dans l'interface utilisateur.

Pour la fonction `main` :

- Le programme attend trois arguments lors de son appel : L'adresse IP du serveur. Le port sur lequel se connecter. Le chemin vers le fichier de certificat à utiliser pour la connexion SSL. Si le nombre d'arguments (`argc`) n'est pas égal à 4 (le nom du programme plus les trois arguments), le programme affiche un message d'erreur et s'arrête.
- Ensuite, `hostname` reçoit l'adresse IP du serveur. `port` est converti en entier grâce à `atoi` (la chaîne de caractères fournis est convertie en un nombre entier). `cert_file` est le chemin vers le certificat qui sera utilisé pour vérifier le serveur lors de l'établissement de la connexion
- La fonction `setup_ssl_connection` (définie dans `network.c`) s'occupe d'initialiser OpenSSL. Créer un contexte SSL. Charger le certificat. Créer une socket TCP et établir la connexion avec le serveur. Associer la socket à l'objet SSL et établir la connexion sécurisée. Ce qui va créer un pointeur vers un objet SSL qui est retourné et stocké dans la variable `ssl`. Ce pointeur sera utilisé pour toutes les communications ultérieures avec le serveur.
- La fonction `display_title` est utilisée pour afficher un titre formaté ("Main Menu"). Ensuite, le menu affiche quatre options : Gérer les séances. Gérer les étudiants. Gérer les absences. Quitter l'application. Le programme attend que l'utilisateur saisisse un entier correspondant à son choix. La fonction `getchar()` est utilisée ensuite pour consommer le caractère de retour à la ligne restant dans le buffer.
- Traitement du choix avec un switch : Selon le choix de l'utilisateur, le programme appelle : `handle_seance_management(ssl)` pour gérer les séances, `handle_student_management(ssl)` pour gérer les étudiants, `handle_attendance_management(ssl)` pour gérer les absences. L'option 4 (Exit) utilise l'instruction `goto cleanup` pour sortir de la boucle et passer à la phase de nettoyage.
- Si le choix n'est pas reconnu, le programme affiche "Invalid choice" et la boucle recommence.
- Lorsque l'utilisateur choisit de quitter (option 4), le programme se rend à l'étiquette `cleanup`. Fonction appelée : La fonction `cleanup_ssl_connection(ssl)` (définie dans `network.c`) s'occupe de récupérer la socket et le contexte SSL associés. Libérer l'objet SSL. Fermer la socket. Libérer le contexte SSL. Nettoyer les ressources OpenSSL.
- La fonction `main` retourne 0, ce qui indique une terminaison normale de l'application.

```

int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <ip> <port> <cert_file>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    const char* hostname = argv[1];
    int port = atoi(argv[2]);
    const char* cert_file = argv[3];

    SSL* ssl = setup_ssl_connection(hostname, port, cert_file);

    while (1) {
        display_title("Main Menu");
        printf("1. Manage seances\n");
        printf("2. Manage students\n");
        printf("3. Attendance\n");
        printf("4. Exit\n");
        printf("Choice: ");
        |
        int choice;
        scanf("%d", &choice);
        getchar();

        switch (choice) {
            case 1:
                handle_seance_management(ssl);
                break;
            case 2:
                handle_student_management(ssl);
                break;
            case 3:
                handle_attendance_management(ssl);
                break;
            case 4:
                goto cleanup;
            default:
                printf("Invalid choice\n");
        }
    }

cleanup:

```

4.5/ Le Makefile :

Le Makefile va permettre de compiler en utilisant le compilateur gcc et de construire l'application client en intégrant des bibliothèques OpenSSL et d'autres dépendances :

- **CC** définit le compilateur à utiliser, ici gcc (le compilateur GNU C).
- **CFLAGS** contient les options de compilation : -Wall active tous les avertissements du compilateur, afin d'aider à détecter d'éventuelles erreurs. -g inclut les informations de débogage dans les binaires générés, utiles pour le débogage avec un outil comme gdb.
- **DIST_DIR** définit le répertoire de destination où seront placés les fichiers objets (.o) et l'exécutable final. Dans notre projet il est situé dans un dossier `../dist/client_c`.
- **OPENSSL_CFLAGS** indique l'option d'inclusion pour le compilateur. Ici, -I`./libs/openssl/include` ajoute le répertoire d'en-tête d'OpenSSL, permettant ainsi d'inclure les fichiers nécessaires à la compilation.
- **OPENSSL_LIBS** définit les options de l'éditeur de liens (linker) pour intégrer OpenSSL : -L`./libs/openssl/lib` indique au linker où chercher les bibliothèques. -lssl et -lcrypto lient respectivement la bibliothèque SSL et la bibliothèque de cryptographie d'OpenSSL.
- **OPENSSL_DIR** indique un autre chemin, ici utilisé plus tard dans la règle `copy_ssl`, pour copier le fichier de certificat (`cert.pem`) depuis ce répertoire vers le dossier de distribution.
- **SRCS** liste tous les fichiers source C qui composent le projet.
- **OBJS** utilise une substitution de suffixe pour transformer la liste des sources en une liste des fichiers objets correspondants. Pour chaque fichier `src/nom.c`, on obtiendra un fichier objet `$(DIST_DIR)/nom.o`. Cette syntaxe permet de centraliser la gestion des fichiers objets et de les placer dans le dossier de distribution.
- **EXEC** définit le chemin et le nom de l'exécutable final, ici `client` qui sera placé dans le répertoire de distribution.
- La cible `all` est la règle par défaut qui sera exécutée si vous tapez simplement `make` ou `make all`. Elle dépend de plusieurs cibles : `create_dist_dir` : Crée le répertoire de distribution s'il n'existe pas. `$(EXEC)` : Construit l'exécutable final à partir des fichiers objets. `clean_o_files` : Supprime les fichiers objets une fois l'exécutable créé. `copy_ssl` : Copie le fichier `cert.pem` dans le dossier de distribution.
- `$(EXEC)` dépend des fichiers objets `$(OBJS)`. `$(CC)` est utilisée pour compiler et lier. `-o $@` indique que l'exécutable généré doit porter le nom de la cible (`$@` se réfère à `$(EXEC)`). `$^` représente tous les prérequis (les fichiers objets). Puis, on ajoute les options de l'éditeur de liens pour OpenSSL et pour d'autres bibliothèques telles que GTK, GLib, Pango, HB, Cairo, Pixbuf, et ATK.

- La règle générique compile chaque fichier source .c situé dans le dossier src/ en un fichier objet .o qui sera placé dans le dossier \$(DIST_DIR). \$< est la première dépendance (le fichier source) et \$@ est la cible (le fichier objet). Les options utilisées : \$(CFLAGS) pour les options de compilation générales. \$(OPENSSL_CFLAGS) pour inclure les en-têtes OpenSSL. Les variables comme \$(GTK_CFLAGS), etc. (pour d'autres bibliothèques) doivent être définies ailleurs pour ajouter les options d'inclusion spécifiques. -c indique que l'on compile en générant un fichier objet sans effectuer l'édition de liens.
- La cible create_dist_dir exécute la commande mkdir -p \$(DIST_DIR) qui crée le dossier de distribution s'il n'existe pas déjà. L'option -p permet de créer également les dossiers parents si besoin.
- La cible **copy_ssl** copie récursivement le fichier cert.pem (situé dans \$(OPENSSL_DIR)) vers le dossier de distribution \$(DIST_DIR). Cela permet d'inclure le certificat nécessaire au fonctionnement de la connexion SSL dans le dossier de distribution.
- La cible **clean_o_files** supprime tous les fichiers objets (*.o) du répertoire de distribution afin de nettoyer l'environnement de build après la création de l'exécutable.
- La cible **clean** est destinée à supprimer complètement l'exécutable et le répertoire de distribution. rm -rf force la suppression récursive de tous les fichiers/dossiers indiqués. En faisant un make clean vous supprimerez donc tout ce que make a pu créer
- La directive .PHONY indique que les cibles listées (all et clean) ne correspondent pas à des fichiers. Cela signifie que même s'il existe un fichier nommé clean ou all, Make exécutera toujours la commande associée à la cible.

```

CC = gcc
CFLAGS = -Wall -g

DIST_DIR = ../../dist/client_c

OPENSSL_CFLAGS = -I./libs/openssl/include
OPENSSL_LIBS = -L./libs/openssl/lib -lssl -lcrypto

SRCS = src/main.c src/display.c src/handlers.c src/network.c
OBJS = $(SRCS:src/%.c=$(DIST_DIR)/%.o)

EXEC = $(DIST_DIR)/client

OPENSSL_DIR = ../../openssl/ssl

all: create_dist_dir $(EXEC) clean_o_files copy_ssl

$(EXEC): $(OBJS)
    $(CC) -o $@ $(OPENSSL_LIBS) $(GTK_LIBS) $(GLIB_LIBS) $(PANGO_LIBS) $(HB_LIBS) $(CAIRO_LIBS) $(PIXBUF_LIBS)
    $(DIST_DIR)/%.o: src/%.c
        $(CC) $(CFLAGS) $(OPENSSL_CFLAGS) $(GTK_CFLAGS) $(GLIB_CFLAGS) $(PANGO_CFLAGS) $(HB_CFLAGS) $(CAIRO_CFLAGS)

create_dist_dir:
    mkdir -p $(DIST_DIR)

copy_ssl:
    cp -r $(OPENSSL_DIR)/cert.pem $(DIST_DIR)/

clean_o_files:
    rm -f $(DIST_DIR)/*.*o

clean:
    rm -rf $(EXEC) $(DIST_DIR)

.PHONY: all clean

```

Démonstration du client C en CLI

Tout d'abord il faut faire un make ce qui va instancier les différents clients dont le client et le backend. Pour ce client il n'est pas lié au make seulement le backend.

```
[mathieubersin@MacBook-Pro-de-Mathieu iut-sae-302 % make  
mkdir -p ./dist
```

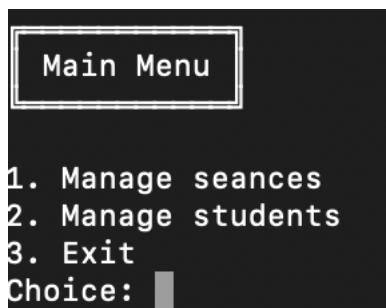
On lance le server en allant dans le répertoire dist est en lancant le server sur le port 8081

```
mathieubersin@MacBook-Pro-de-Mathieu iut-sae-302 % cd dist/server  
mathieubersin@MacBook-Pro-de-Mathieu server % ./server 8081  
[INFO] Server is listening on port 8081...
```

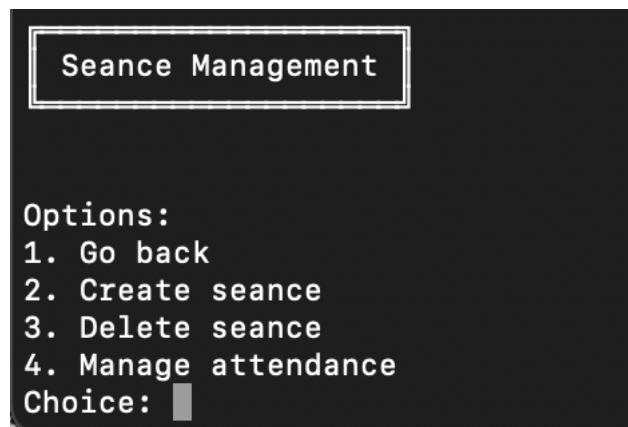
On lance le client C en allant dans le répertoire dist/client_c/ :

```
mathieubersin@MacBook-Pro-de-Mathieu client_c % ./client 127.0.0.1 8081 cert.pem
```

On arrive donc sur ce petit menu en cli :



On a donc trois choix, 1. Manage seances ou 2.Manage students ou 3. Exit, On va taper 1 pour manager les seances :



On a donc 4 choix soit de revenir en arrière, créer une séance, Delete une seance, ou gérer les absences. Nous allons donc deux séances une test qu'on supprimera et une qu'on nommera EPS.

```
Enter seance name: test  
Enter date (DD/MM/YYYY): 10/10/2024  
Enter time (HH:MM): 10:00
```

Seance Management

```
0. test (10/10/2024 11:00)
```

Options:

- 1. Go back
- 2. Create seance
- 3. Delete seance
- 4. Manage attendance

Choice:

On demande le nom, la date et l'heure de la séance.

<---- On voit bien que la séance est créée

```
0. test (10/10/2024 11:00)  
1. EPS (10/10/2024 13:00)
```

Options:

- 1. Go back
- 2. Create seance
- 3. Delete seance
- 4. Manage attendance

Choice: 3

Enter seance id: 0

Seance Management

```
0. test (10/10/2024 11:00)  
1. EPS (10/10/2024 15:00)
```

Options:

- 1. Go back
- 2. Create seance
- 3. Delete seance
- 4. Manage attendance

Choice: ■

```
1. EPS (10/10/2024 13:00)
```

Options:

- 1. Go back
- 2. Create seance
- 3. Delete seance
- 4. Manage attendance

Choice:

On supprime la séance avec son id. Maintenant avec l'id de la seance EPS on arrive sur cette vue pour pouvoir gérer les absences. Si on appuie sur 2 on devra donner l'id de l'élève et deux choix : 1 présent ou 0 absent.

```
Attendance

Seance: EPS

Mathieu (ID: 0): Absent

Options:
1. Go back
2. Take attendance
Choice: █
```

Maintenant voici la vue student où l'on peut supprimer et rajouter des students.

```
Student Management

(0) Mathieu

Options:
1. Go back
2. Create new student
3. Delete student
Choice: █
```

Et enfin on quitte le client ce qui va le déconnecter du serveur avec le choix 3 du main menu.

```
Choice: 3
mathieubersin@MacBook-Pro-de-Mathieu client_c % █
```

5/ Le client en Java (CLI + GUI)

Comme convenu, afin de réaliser ce projet de SAE, nous avons développé un client en Java. Nous avons choisi de réaliser le programme en Java sous deux formes d'utilisation pour l'utilisateur. Le premier cas est en CLI (qui signifie Command Line Interface), c'est-à-dire qu'il n'y a pas d'interface graphique et que l'utilisateur communique et interagit avec notre application uniquement via un terminal comme PowerShell, Bash ou autre. Le second cas d'usage que nous avons décidé de développer pour améliorer l'expérience utilisateur de notre application est une interface graphique. Cela est donc plus agréable pour l'utilisateur, qui peut cliquer sur des boutons pour interagir avec l'application/serveur.

Nous allons donc dans un premier temps s'intéresser aux différentes classes en Java qui vont être utilisées à la fois pour le client Java (CLI + GUI) et pour le client Android (développé avec le langage aussi donc l'application utilise les même classes).

- La classe Seance qui représente les séances dans le code local.
- La classe Etudiant qui représente les étudiants du code local.
- La classe Absence représentant le statut d'un étudiant (présent ou absent) dans une seance.
- La classe Client qui permet via un socket la connexion au serveur.

5.1/ La classe Client :

Le fichier client.java implémente une classe permettant de gérer une connexion sécurisée via SSL/TLS avec un serveur distant. Cette classe utilise des sockets SSL (SSLSocket) pour établir la communication et propose plusieurs méthodes pour interagir avec le serveur. L'objectif principal de ce client est de permettre la gestion des séances et des étudiants pour un système d'absence, en utilisant un protocole sécurisé afin de protéger les données échangées. L'utilisation de SSL/TLS garantit que les informations transmises ne peuvent pas être interceptées ou modifiées par un tiers.

Voici les différentes classes importées :

Titre	Utilité
FileInputStream	Lit les données d'un fichier, octet par octet, pour traiter des fichiers sur disque.
IOException	Gère les erreurs d'entrée/sortie lors de la lecture ou de l'écriture de données.
DataInputStream	Lit des données primitives à partir d'un flux de données de manière plus structurée.
DataOutputStream	Permet d'écrire des données primitives dans un flux de sortie de manière structurée.
File	Représente un fichier ou un répertoire sur le système de fichiers.
KeyStore	Contient des clés cryptographiques et des certificats, par exemple pour SSL.
CertificateFactory	Crée des objets de certificat X.509 à partir de données d'entrée.
X509Certificate	Représente un certificat X.509, utilisé dans les protocoles de sécurité.
SSLContext	Contexte pour les connexions SSL, utilisé pour définir les paramètres de sécurité.
SSLSocket	Implémente une connexion client SSL, assurant la communication sécurisée.
SSLSocketFactory	Fabrique des sockets SSL qui peuvent être utilisés pour la création de connexions SSL sécurisées.
TrustManagerFactory	Crée des gestionnaires de confiance basés sur une source de données de confiance, comme un KeyStore.
ArrayList	Une liste qui peut s'agrandir ou se rétrécir, idéale pour stocker plusieurs objets.
List	Collection ordonnée d'éléments, comme une liste d'objets.

Le socket SSL, représenté par sockfd, est utilisé pour établir une connexion sécurisée avec le serveur. Les flux enSortie et enEntrée permettent respectivement d'envoyer et de recevoir des données sous forme binaire. L'adresse IP et le port du serveur sont stockés dans serverIP et serverPort après la connexion afin de garder une référence aux informations du serveur.

```

private static Client instance; // Instance unique de Client
private SSLSocket sockfd;
private DataOutputStream enSortie;
private DataInputStream enEntree;
private String serverIP;
private int serverPort;

// Constructeur privé pour éviter l'instanciation externe
private Client() {}

```

On implémente le modèle Singleton en Java qui est un modèle de conception qui garantit qu'une classe n'a qu'une seule instance et fournit un point d'accès global à cette instance. C'est un concept très important lorsque l'on veut contrôler l'accès à certaines ressources partagées comme dans notre application, avec une connexion à notre serveur via un socket.

```

// Méthode pour récupérer l'instance unique
public static Client getInstance() {
    if (instance == null) {
        synchronized (Client.class) {
            if (instance == null) {
                instance = new Client();
            }
        }
    }
    return instance;
}

```

La fonction connectToServer établit une connexion sécurisée avec un serveur en utilisant SSL. Elle charge d'abord un certificat de sécurité, configure le contexte SSL, puis crée une connexion sécurisée (SSLSocket) à l'adresse IP et au port spécifiés par l'utilisateur. Une fois la connexion établie, elle prépare les flux de données nécessaires pour la communication. Si une erreur se produit à n'importe quelle étape, elle capture l'erreur et retourne false. Sinon, elle confirme la connexion réussie et retourne true.

- Initialise un KeyStore vide pour stocker les certificats de sécurité.
- Charge un certificat depuis un fichier (cert.pem) dans le KeyStore.
- Configure un TrustManagerFactory avec le KeyStore pour gérer la vérification de certificats.
- Crée un SSLContext avec le protocole TLSv1.2 pour établir des connexions sécurisées.
- Génère une SSLSocketFactory à partir du SSLContext pour créer des connexions sécurisées.
- Établit une connexion sécurisée (SSLSocket) au serveur à l'adresse IP et au port spécifiés.
- Prépare des flux de données pour envoyer (DataOutputStream) et recevoir (DataInputStream) des informations à travers la connexion.
- Affiche un message si la connexion est réussie et retourne true.

- Capture et affiche des erreurs, puis retourne false en cas d'échec de connexion.

```

public boolean connectToServer(String hote, int port) {
    try {
        // Load the certificate
        KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
        keyStore.load(null, null); // Initialize an empty KeyStore

        String certPath = new File("ssl" + File.separator + "cert.pem").getCanonicalPath();
        System.out.println("[INFO] Loading certificate from: " + certPath);

        try (FileInputStream fis = new FileInputStream(certPath)) {
            CertificateFactory cf = CertificateFactory.getInstance("X.509");
            X509Certificate cert = (X509Certificate) cf.generateCertificate(fis);
            keyStore.setCertificateEntry("cert", cert);
        }

        // Initialize the TrustManagerFactory
        TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        tmf.init(keyStore);

        // Initialize the SSLContext
        SSLContext sslContext = SSLContext.getInstance("TLSV1.2");
        sslContext.init(null, tmf.getTrustManagers(), null);

        // Create an SSLSocketFactory
        SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();

        // Create an SSLSocket
        sockfd = (SSLSocket) sslSocketFactory.createSocket(hote, port);
        enSortie = new DataOutputStream(sockfd.getOutputStream());
        enEntree = new DataInputStream(sockfd.getInputStream());
        serverIP = hote;
        serverPort = port;
        System.out.println("[INFO] Connected to the server | " + hote + ":" + port + "\n");
        return true;

    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

```

La méthode sendCommand envoie une commande sous forme de chaîne de caractères convertis en tableau d'octets à travers le flux de sortie DataOutputStream permettant ainsi l'envoi de requêtes au serveur.

```

private void sendCommand(String command) throws IOException {
    byte[] buffer = command.getBytes();
    enSortie.write(buffer, 0, buffer.length);
}

```

La méthode createstudent() est une méthode qui prend en paramètre le nom de l'étudiant à créer et envoie une requête au serveur pour créer cet étudiant. À l'aide de la méthode sendcommand(), la commande suivante est envoyée au serveur "<p>student" + name.

```

public boolean createStudent(String name) {
    try {
        sendCommand("<p>student/" + name);
        byte[] buffer = new byte[1024];
        int n = enEntree.read(buffer, off:0, len:1023);
        String response = new String(buffer, offset:0, n);

        if (response.startsWith(prefix:"202/")) {
            System.out.println("\n[INFO] - Seance '" + name + "' created successfully.");
            return true;
        }
    }
}

```

Ensuite la méthode vérifie la réponse du serveur si la réponse commence par “202” ça veut dire que l’opération a réussi sinon ça veut dire qu’il y’a eu un problème. La méthode CreateSeances fonctionnent sur le même principe sauf que cette dernière prend en paramètres le nom de la séance ainsi que l’unixTime qui correspond aux horaires de la séance à créer. De plus les méthodes deleteSeance() et deleteStudent fonctionnent pareil :

DeleteSeance() prend en paramètre l’id de la seance et à l’aide de la méthode sendCommand() elle envoie au serveur la commande suivante “<d>seance/+idSeance”

DeleteStudent() prend en paramètre l’id de l’étudiant et à l’aide de la méthode sendCommand() elle envoie au serveur la commande suivante “<d>seance/+idStudent”

GetAttendance() prend en paramètre l’id de la séance et l’id de l’étudiant et à l’aide de la même méthode elle envoie au serveur la commande suivant “<g>attendance/ +idStudent / +idSeance”

SetAbsence() prend en paramètre l’id de la séance et l’id de l’étudiant et à l’aide de la même méthode elle envoie au serveur la commande suivant “<p>attendance/ +idStudent / +idSeance / + statut”

CloseRessource() Cette fonction sert à fermer proprement toutes les ressources liées à la connexion au serveur distant

```
usage
public boolean closeResources() {
    try {
        if (enSortie != null) {
            enSortie.close();
            enSortie = null;
        }
        if (enEntree != null) {
            enEntree.close();
            enEntree = null;
        }
        if (sockfd != null) {
            sockfd.close();
            sockfd = null;
        }
        System.out.println("[INFO] - Disconnected from the server");
        return true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}
```

5.2/ La classe Etudiant

La classe Etudiant représente un étudiant dans l'application. Elle contient deux attributs principaux : un identifiant unique pour chaque étudiant (id_etudiant) et un nom pour l'étudiant (nom_etudiant).

Ces deux attributs sont définis comme privés afin d'encapsuler les données et d'assurer que leur valeur ne puisse être modifiée directement en dehors de la classe. Cette approche suit le principe de l'encapsulation de la programmation orientée objet, qui est utilisée pour protéger l'intégrité des données en forçant l'accès à travers des méthodes (getters et setters).

```
public class Etudiant {  
    3 usages  
    private int id_etudiant;  
    3 usages  
    private String nom_etudiant;
```

Le constructeur de la classe Etudiant prend deux arguments : un identifiant d'étudiant (id_etudiant) et un nom d'étudiant (nom_etudiant). Lorsque ce constructeur est appelé, il initialise les variables d'instance de l'objet Etudiant avec les valeurs passées en paramètres. L'utilisation du mot-clé this permet de différencier les attributs de la classe des paramètres du constructeur (qui portent le même nom).

```
    public Etudiant(int id_etudiant, String nom_etudiant) {  
        this.id_etudiant = id_etudiant;  
        this.nom_etudiant = nom_etudiant;  
    }
```

Les méthodes getIdEtudiant() et getNomEtudiant() sont des méthodes d'accès (également appelées getters) qui permettent d'accéder aux valeurs des attributs privés de la classe Etudiant. Ces méthodes ne font que retourner la valeur stockée dans l'attribut correspondant :

- getIdEtudiant() retourne l'identifiant unique de l'étudiant.
- getNomEtudiant() retourne le nom de l'étudiant.

```
    public int getIdEtudiant() {  
        return id_etudiant;  
    }  
  
    public String getNomEtudiant() {  
        return nom_etudiant;  
    }
```

La méthode `toString()` est redéfinie dans cette classe pour fournir une représentation textuelle de l'objet `Etudiant`. En redéfinissant cette méthode, nous permettons à l'objet `Etudiant` d'afficher des informations plus pertinentes sous forme de chaîne de caractères. Ici, la méthode retourne une chaîne qui inclut à la fois le nom de l'étudiant et son identifiant.

```
public String toString() {  
    return "Name = " + nom_etudiant + " | ID [" + id_etudiant + "]";  
}
```

5.3/ La classe `Seance` :

La classe `Seance` est une représentation d'une séance d'enseignement. Elle contient des informations sur la séance, comme son identifiant, son nom, la liste des étudiants inscrits et la liste des absences enregistrées.

Elle permet aussi de gérer l'ajout et la suppression d'étudiants, d'enregistrer les absences et de formater les informations sous forme de texte.

Déclaration des variables

```
4 usages  
private int id_seance;  
3 usages  
private String nom_seance;  
6 usages  
private List<Etudiant> list_etudiant;  
7 usages  
private List<Absence> list_absent;  
4 usages  
private Long unixTime;
```

- `id_seance` : Identifiant unique de la séance.
- `nom_seance` : Nom de la séance.
- `list_etudiant` : Liste des étudiants inscrits.
- `list_absent` : Liste des absences enregistrées.
- `unixTime` : Timestamp Unix représentant la date et l'heure de la séance.

Constructeur de la classe

```
16     public Seance(int id_seance, String nom_seance, Long unixTime) {  
17         this.id_seance = id_seance;  
18         this.nom_seance = nom_seance;  
19         this.list_etudiant = new ArrayList<>();  
20         this.list_absent = new ArrayList<>();  
21         this.unixTime = unixTime;  
22     }  
23 }
```

- Initialise l'identifiant et le nom de la séance.
- Initialise les listes list_etudiant et list_absent comme des listes vides.
- Stocke les horaires de la séance.

Définitions des Getters

```
25 >     public Long getUnixTime() { return unixTime; }  
28  
29 >     3 usages  
29 >     public int getIdSeance() { return id_seance; }  
32  
33 >     no usages  
33 >     public String getNomSeance() { return nom_seance; }  
36  
37 >     no usages  
37 >     public List<Etudiant> getListEtudiant() { return list_etudiant; }  
38 }
```

Ces méthodes permettent d'accéder aux valeurs des attributs privés de la classe à partir des autres classes de l'application.

Gestion des étudiants

Ajout d'un étudiant

```
no usages  
>     public void ajouterEtudiant(Etudiant etudiant) { list_etudiant.add(etudiant); }
```

Ajoute un étudiant à la liste des étudiants inscrits.

Suppression d'un étudiant

```
5     public void supprimerEtudiant(int etudiantId) {  
6         for (Etudiant etudiant : list_etudiant) {  
7             if (etudiant.getIdEtudiant() == etudiantId) {  
8                 list_etudiant.remove(etudiant);  
9                 break;  
10            }  
11        }  
12    }
```

Parcourt la liste des étudiants et supprime celui qui correspond à l'ID donné.

Ajout ou modification d'une absence

```
54     public void setAbsence(int id_etudiant, int presence) {  
55         boolean found = false;  
56         for (int i = 0; i < list_absent.size(); i++) {  
57             Absence absence = list_absent.get(i);  
58             if (absence.getIdEtudiant() == id_etudiant) {  
59                 list_absent.set(i, new Absence(id_seance, id_etudiant, presence));  
60  
61                 found = true;  
62                 break;  
63             }  
64         }  
65         if (!found) {  
66             list_absent.add(new Absence(id_seance, id_etudiant, presence));  
67         }  
}
```

Ce code :

- Vérifie si l'étudiant est déjà marqué comme absent ou présent.
- Met à jour son statut si trouvé.
- Sinon, ajoute un nouvel enregistrement d'absence.

Récupération de la liste des absents :

```
public List<Absence> getListAbsent() {  
    return list_absent;  
}
```

Ce code Retourne la liste des étudiants absents.

```
public String toString() {
```

Ce code Effectue :

La Conversion du timestamp :Convertit unixTime en une date lisible.

Utilise SimpleDateFormat pour le formater en dd/MM/yyyy HH:mm.

La Construction d'une chaîne de texte : Ajoute le nom de la séance et la date.

Parcourt la liste des étudiants. Vérifie leur statut de présence ou d'absence.

Retourne un texte formaté affichant ces informations.

La classe Seance permet donc de gérer une séance de cours avec les fonctionnalités suivantes :

- Stockage des informations de séance.
- Gestion des étudiants (ajout, suppression).
- Gestion des absences.
- Formatage des informations pour affichage.

Elle joue un rôle clé dans la gestion de la présence des étudiants dans un système de suivi des absences.

5.4/ La classe Absence :

```
public class Absence {  
    private int id_seance;  
    private int id_etudiant;  
    private int presence;
```

Comme pour les autres classes précédentes la classe absence est défini en public afin que l'on puisse y accéder n'importe où dans l'application. Créer une absence avec un identifiant de séance, un identifiant d'étudiant et un état de présence. Récupérer ces informations via les méthodes getter. Afficher l'absence sous forme de chaîne de caractères grâce à la méthode `toString()`.

Constructeur de la classe

```
public Absence(int id_seance, int id_etudiant, int presence) {  
    this.id_seance = id_seance;  
    this.id_etudiant = id_etudiant;  
    this.presence = presence;  
}
```

Le constructeur prend trois paramètres :

- `id_seance` : L'identifiant de la séance.
- `id_etudiant` : L'identifiant de l'étudiant.
- `presence` : L'état de présence (1 pour présent, 0 pour absent).

Le constructeur initialise les trois attributs de la classe avec les valeurs fournies lors de la création d'un objet `Absence`.

Les méthodes Getters

```
public int getIdSeance() {  
    return id_seance;  
}  
  
public int getIdEtudiant() {  
    return id_etudiant;  
}  
  
public int getPresence() {  
    return presence;  
}
```

Ces Getters permettent de récupérer l'id des seance, l'id de l'étudiant et le statut de présence des étudiants.

La classe Absence permet donc de Créer une absence avec un identifiant de séance, un identifiant d'étudiant et un état de présence. Récupérer ces informations via les méthodes getter. Afficher l'absence sous forme de chaîne de caractères grâce à la méthode `toString()`.

5.5/ Le fichier Main :

Pour développer notre client java, nous allons tout d'abord importer des librairies, voici leurs explications respectives et donc à quoi celles-ci vont nous servir au sein de notre programme :

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.List;
import java.util.Scanner;
import java.util.Date;
import java.util.ArrayList;
import java.util.TimeZone;
```

Titre	Utilité
ParseException	Gère des erreurs au sein de notre code.
SimpleDateFormat	Outil pour transformer une date en texte et vice versa, selon un modèle précis
List	Collection ordonnée d'éléments, comme une liste d'objets
Scanner	Outil pour lire des données entrées par l'utilisateur, comme du texte ou des nombres
Date	Représente une date et une heure précises.
ArrayList	Une liste qui peut s'agrandir ou rétrécir, idéale pour stocker plusieurs objets
TimeZone	Représente des zones horaires différentes (UTC dans notre cas).

Explication du mode CLI

Dans le fichier main, 3 variables vont être déclarées de façon « private » afin de respecter l'encapsulation (vu en cours durant ce module). De plus, les variables vont être « static » ce qui va nous permettre d'interagir directement avec ces variables partout dans notre code (typiquement dans nos fonctions).

```
// Déclarer les listes comme variables statiques
private static List<Etudiant> etudiants = new ArrayList<>();
private static List<Seance> seances = new ArrayList<>();
private static Client client; //
```

Par la suite, comme nous avons développé une fonctionnalité supplémentaire en GUI, l'utilisateur de notre programme a le choix d'utiliser soit le programmeur en CLI soit en interface graphique. Ainsi, nous allons donc lire le choix de l'utilisateur via une valeur entrée par l'utilisateur et lire par un objet de la

classe Scanner. Pour cela, le programme exécute une fonction appelée « choisirModeExecution() » prenant comme paramètre l'objet scanner.

Cette fonction permet ainsi de vérifier tout d'abord que l'utilisateur entre une valeur qui est (1 ou 2) grâce à la boucle while, et elle retourne le choix de l'utilisateur s'il est donc valide. De plus, elle affiche des informations en CLI pour indiquer par exemple à l'utilisateur si l'entrée choisie (autre que 1 ou 2 donc), n'est pas valide.

```
// Fonction pour choisir le mode d'exécution (console ou graphique)
private static int choisirModeExecution(Scanner scanner) {
    int choice = 0;
    while (choice != 1 && choice != 2) {
        System.out.println("Veuillez choisir votre mode d'exécution:\n [1] - console\n [2] - Interface graphique");
        if (scanner.hasNextInt()) {
            choice = scanner.nextInt();
            if (choice != 1 && choice != 2) {
                System.out.println("Choix incorrect. Veuillez réessayer.");
            }
        } else {
            System.out.println("Entrée invalide. Veuillez entrer un nombre.");
            scanner.next(); // Consomme l'entrée invalide
        }
    }
    return choice;
}
```

Ainsi, grâce à ce choix retourné par la fonction nous allons exécuter la méthode correspondante au choix de l'utilisateur.

```
public static void main(String[] args) {
    // Affiche un message de bienvenue
    System.out.println("Bienvenue dans le gestionnaire d'absences");
    Scanner scanner = new Scanner(System.in);
    int choice = 0;

    // Boucle principale pour demander à l'utilisateur de choisir un mode d'exécution
    while (true) {
        choice = choisirModeExecution(scanner);

        switch (choice) {
            case 1:
                executerModeConsole(scanner);
                choice = 0; // Réinitialise le choix pour permettre de revenir au menu
                break;

            case 2:
                executerModeGraphique();
                return;

            default:
                break;
        }
    }
}
```

La méthode executerModeConsole() permet d'afficher à l'utilisateur une demande pour entrer le port et l'adresse IP du serveur sur lequel il souhaite se connecter. Nous allons lire les données entrées par l'utilisateur via un objet de la classe Scanner. Ensuite, nous allons initialiser un objet de la classe Client

appelé client. Nous faisons cela en utilisant la méthode getInstance() (basée sur le concept du Singleton en Java), comme nous l'avons vu précédemment 5.1/ La classe Client ::

```
// Fonction pour exécuter le mode console
private static void executerModeConsole(Scanner scanner) {
    System.out.println("Bienvenue dans le gestionnaire d'absences [Console]");

    // Demande l'adresse IP et le port du serveur
    System.out.print("Veuillez entrer l'adresse IP du serveur|");
    String ip = scanner.next();
    System.out.print("Veuillez entrer le port du serveur|");
    int port = scanner.nextInt();

    Client client = Client.getInstance();
    try {
        if (client.connectToServer(ip, port)) {
            gererMenuPrincipal(scanner);
        } else {
            System.out.println("[ERROR] - Veuillez vérifier l'adresse IP et le port.");
        }
    } catch (Exception e) {
        System.out.println("[ERROR] - Veuillez vérifier l'adresse IP et le port.");
    }
}
```

Lorsque la connexion au serveur est correctement établie, le programme appelle la méthode gererMenuPrincipal(). La première méthode exécutée est updateListsFromServer(). Cette méthode commence par vider les listes existantes d'étudiants et de séances, représentées par les variables étudiants et séances, respectivement.

Ensuite, ces variables sont réinitialisées grâce à la méthode addAll(), une méthode en Java qui permet d'ajouter tous les éléments d'une collection à une autre. Étant une partie de l'interface Collection, elle peut être utilisée avec toutes les classes implémentant cette interface, comme par exemple ArrayList dans ce cas.

Le programme utilise les méthodes getStudents() et getSeances() du client pour obtenir les listes depuis le serveur, et ces listes sont ensuite passées en paramètre à addAll() pour mettre à jour complètement les listes locales d'étudiants et de séances.

```
// Mettez à jour les listes avec les données du serveur
private static void updateListsFromServer() {
    etudiants.clear();
    seances.clear();
    etudiants.addAll(client.getStudents());
    seances.addAll(client.getSeances());
}
```

Ensuite, c'est dans le menu que l'utilisateur pourra faire des choix pour interagir avec le serveur. Pour cela, le programme utilise un objet de la classe Scanner afin de lire les demandes de l'utilisateur. Les différents choix possibles sont exécutés suivant un switch case. Le switch case permet d'exécuter différentes méthodes selon la valeur d'une variable, dans notre cas la variable choix, représentée par un entier (int).

```
// Fonction pour gérer le menu principal en mode console
private static void gererMenuPrincipal(Scanner scanner) {
    updateListsFromServer();

    while (true) {
        afficherMenu();
        int choix = scanner.nextInt();
        switch (choix) {
            case 1:
                afficherEtudiants();
                break;
            case 2:
                afficherSeances();
                break;
            case 3:
                enregistrerAbsences(scanner);
                break;
            case 4:
                creerEtudiant(scanner);
                break;
            case 5:
                creerSeance(scanner);
                break;
            case 6:
                supprimerEtudiant(scanner);
                break;
            case 7:
                supprimerSeance(scanner);
                break;
            case 8:
                visualiserAbsencesSeance(scanner);
                break;
            case 0:
                System.out.println("Au revoir!");
                client.closeResources();
                return;
            default:
                System.out.println("Veuillez entrer une réponse valide.");
        }
    }
}
```

La méthode « afficherMenu() » permet simplement d'imprimer le menu à l'utilisateur afin qu'il sache quel chiffre entrer en fonction de la fonctionnalité qu'il souhaite effectuer. Comme le montre la capture d'écran ci-dessus, le `switch case` permet donc à l'utilisateur d'interagir avec le serveur dans son ensemble en fonction du chiffre entré. De plus, dans le cas par défaut (c'est-à-dire lorsque le choix de l'utilisateur n'est pas parmi ceux attendus par le programme), un message est affiché à l'utilisateur.

```

// Fonction pour afficher le menu des opérations disponibles
private static void afficherMenu() {
    System.out.println("-----Menu-----");
    System.out.println("[1] Visualiser les étudiants");
    System.out.println("[2] Visualiser les séances");
    System.out.println("[3] Enregistrer les absences d'une seance");
    System.out.println("[4] Creer un etudiant");
    System.out.println("[5] Creer une seance");
    System.out.println("[6] Supprimer un etudiant");
    System.out.println("[7] Supprimer une seance");
    System.out.println("[8] Visualiser les absences");
    System.out.println("[0] Quitter");
    System.out.print("Choix | ");
}

```

Les deux premiers cas du switch case permettent à l'utilisateur d'afficher les étudiants (choix 1) et les séances (choix 2). Ces deux méthodes ont une structure similaire. Elles utilisent une boucle pour parcourir la liste des étudiants ou des séances, qui a été récupérée précédemment depuis le serveur, et affichent chaque objet à l'utilisateur. Pour améliorer la visibilité, des lignes de « = » sont affichées pour séparer les informations demandées par l'utilisateur du menu de choix.

```

// Fonction pour afficher la liste des étudiants
private static void afficherEtudiants() {
    System.out.println("\n=====Students=====\n");
    for (Etudiant etudiant : etudiants) {
        System.out.println(etudiant);
    }
    System.out.print("\n=====");
}

// Fonction pour afficher la liste des séances
private static void afficherSeances() {
    System.out.println("\n=====Seances=====\n");
    for (Seance seance : seances) {
        System.out.println(seance);
    }
    System.out.print("\n=====");
}

```

Le choix 3 appelle la méthode enregistrerAbsences(), qui permet à l'utilisateur d'enregistrer les absences des étudiants pour une séance spécifique. D'abord, elle affiche la liste des séances disponibles. Ensuite, l'utilisateur est invité à entrer l'ID de la séance souhaitée. La méthode vérifie alors si la séance existe en parcourant la liste des séances.

Pour chaque étudiant, elle demande à l'utilisateur de confirmer sa présence en saisissant "O" pour présent ou "N" pour absent. Selon la réponse de l'utilisateur, l'absence est enregistrée sur le serveur. Pour cela, on appelle la méthode setAbsence() implémentée dans la classe Client, qui initialise la présence d'un étudiant en prenant comme paramètres l'ID de la séance, l'ID de l'étudiant et son statut (0 pour absent, 1 pour présent). De plus, dans le programme local de l'utilisateur, l'absence est également initialisée sur l'objet Seance en prenant en paramètres l'ID de l'étudiant et son statut. Cela permet d'enregistrer l'absence sur le serveur, mais aussi dans le programme local.

Le booléen « responseValid » permet, au sein de la méthode, de vérifier que lorsque l'utilisateur veut initialiser la présence d'un étudiant, la valeur entrée est bien "O" ou "N". Si l'utilisateur entre une autre valeur, cela permet d'effectuer des vérifications pour éviter de créer des erreurs dans le programme. De plus, la méthode equalsIgnoreCase(), appliquée sur l'entrée utilisateur, permet d'ignorer la casse, qu'il s'agisse de majuscules ou de minuscules.

Des messages de confirmation sont affichés après chaque saisie valide, indiquant si l'étudiant est marqué présent ou absent. Si l'ID de la séance entré par l'utilisateur ne correspond à aucune séance existante, un message informe l'utilisateur que la séance n'a pas été trouvée.

```
private static void enregistrerAbsences(Scanner scanner) {
    afficherSeances();

    System.out.print("\nEntrez ID de la séance | ");
    int id_seance = scanner.nextInt();

    for (Seance seance : seances) {
        if (seance.getIdSeance() == id_seance) {
            System.out.println("\n[Seance] - " + seance.getNomSeance() + "\n");
            for (Etudiant etudiant : etudiants) {
                boolean responseValid = false;

                while (!responseValid) {
                    System.out.print("L'étudiant " + etudiant.getNomEtudiant() + " est-il présent? (O/N) | ");
                    String presence = scanner.next();

                    if (presence.equalsIgnoreCase("N")) {
                        client.setAbsence(id_seance, etudiant.getIdEtudiant(), statut:0);
                        seance.setAbsence(etudiant.getIdEtudiant(), presence:0);
                        System.out.println("Etudiant " + etudiant.getNomEtudiant() + " marqué absent");
                        responseValid = true;
                    } else if (presence.equalsIgnoreCase("O")) {
                        client.setAbsence(id_seance, etudiant.getIdEtudiant(), statut:1);
                        seance.setAbsence(etudiant.getIdEtudiant(), presence:1);
                        System.out.println("Etudiant " + etudiant.getNomEtudiant() + " marqué présent");
                        responseValid = true;
                    } else {
                        System.out.println("Réponse invalide. Veuillez entrer 'O' pour présent ou 'N' pour absent.");
                    }
                }
            }
            System.out.println("\n[INFO] - Absences marquées pour la séance ");
            System.out.println(seance);
            return;
        }
    }
    System.out.println("[INFO] - Séance non trouvée");
}
```

Le quatrième choix permet de créer un nouvel étudiant. Pour cela, la méthode utilise un objet Scanner pour demander à l'utilisateur d'entrer le nom de l'étudiant. Elle récupère cette valeur grâce à la méthode nextLine() appelée sur l'objet Scanner. Le programme appelle ensuite la méthode createStudent() sur l'objet client avec le nom de l'étudiant comme paramètre, ce qui permet de créer un nouvel étudiant sur le serveur.

Après cela, la méthode updateListsFromServer() est appelée pour garantir que les listes d'étudiants et de séances soient à jour. Ensuite, la méthode parcourt, grâce à des boucles itératives, chaque objet dans la liste des séances et chaque étudiant, puis vérifie si l'ID de l'étudiant n'est pas déjà présent dans la liste des étudiants de chaque séance. Si l'étudiant n'est pas présent (ce qui est normalement impossible car le serveur ajoute automatiquement les IDs aux étudiants après vérification), alors

chaque séance ajoute l'étudiant grâce à la méthode ajouterEtudiant() de la classe Seance, prenant l'étudiant comme paramètre. De plus, par défaut, l'étudiant est ajouté à chaque séance avec une valeur de présence de 0 (c'est-à-dire absent à la séance).

```
// Fonction pour créer un nouvel étudiant
private static void creerEtudiant(Scanner scanner) {
    System.out.print("Veuillez entrer le nom de l'étudiant | ");
    scanner.nextLine();
    String name = scanner.nextLine();
    client.createStudent(name);
    updateListsFromServer();

    for (Seance seance : seances) {
        for (Etudiant etudiant : etudiants) {
            boolean etudiantExists = false;
            for (Etudiant existingEtudiant : seance.getListEtudiant()) {
                if (existingEtudiant.getIdEtudiant() == etudiant.getIdEtudiant()) {
                    etudiantExists = true;
                    break;
                }
            }
            if (!etudiantExists) {
                seance.ajouterEtudiant(etudiant);
                seance.setAbsence(etudiant.getIdEtudiant(), presence:0);
            }
        }
    }
}
```

Le cinquième choix du switch case permet de créer une nouvelle séance. La méthode creerSeance() permet de créer une nouvelle séance. Elle commence par lire l'entrée utilisateur qui est le nom de la séance avec le scanner. Ensuite, elle utilise deux méthodes afin demander la date et l'heure de la séance.

```
// Fonction pour créer une nouvelle séance
private static void creerSeance(Scanner scanner) {
    scanner.nextLine(); // Consomme la nouvelle ligne
    System.out.println("Veuillez entrer le nom de la séance :");
    String nomSeance = scanner.nextLine();

    String dateSeance = demanderDateSeance(scanner);
    String heureSeance = demanderHeureSeance(scanner);

    String dateTime = dateSeance + " " + heureSeance;
    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");
    sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
    long unixTime = 0;

    try {
        Date date = sdf.parse(dateTime);
        unixTime = date.getTime() / 1000; // Convert to seconds
    } catch (ParseException e) {
        e.printStackTrace();
    }

    client.createSeance(nomSeance, unixTime);
    updateListsFromServer();
    fillObjects();
}
```

La méthode demanderDateSeance(), demande à l'utilisateur d'entrer une date dans le format "JJ/MM/AAAA". Elle utilise une boucle pour s'assurer que l'utilisateur entre une date valide en utilisant une expression régulière (REGEX) qui vérifie le format correct. Si l'utilisateur entre une date incorrecte, un message d'erreur est affiché et l'utilisateur est invité à réessayer. Une fois que l'utilisateur entre une date valide, la fonction retourne cette date.

La méthode demanderHeureSeance() est une méthode programmée de façon similaire à celle pour demander la date . Cette méthode demande donc à l'utilisateur d'entrer une heure dans le format "HH:MM". Elle utilise une boucle similaire à celle de demanderDateSeance() pour vérifier que l'entrée de l'utilisateur correspond au format d'heure valide. Si l'entrée n'est pas valide, un message d'erreur est affiché, et l'utilisateur doit réessayer. La fonction retourne l'heure saisie une fois qu'elle est validée.

```
// Fonction pour demander la date de la séance
private static String demanderDateSeance(Scanner scanner) {
    String dateSeance;
    while (true) {
        System.out.println("Veuillez entrer la date de la séance (format JJ/MM/AAAA) :");
        dateSeance = scanner.nextLine();
        if (dateSeance.matches("^([0-9]{2})/([0-9]{2})/([0-9]{4})$")) {
            break;
        } else {
            System.out.println("Format de date invalide. Veuillez réessayer.");
        }
    }
    return dateSeance;
}

// Fonction pour demander l'heure de la séance
private static String demanderHeureSeance(Scanner scanner) {
    String heureSeance;
    while (true) {
        System.out.println("Veuillez entrer l'heure de la séance (format HH:MM) :");
        heureSeance = scanner.nextLine();
        if (heureSeance.matches("^([0-9]{2}):([0-9]{2})$")) {
            break;
        } else {
            System.out.println("Format d'heure invalide. Veuillez réessayer.");
        }
    }
    return heureSeance;
}
```

Par la suite, ma methode pour créer une nouvelle seance va combiner en une chaîne de caractères, les valeurs rentrées par les méthodes precedentes (demanderHeureSeance(), demanderDateSeance()). Ensuite, au-sein de la methode, une variable « sdf » represnete un objet de la classe SimpleDateFormat, qui en Java permet de formater et d'analyser des dates. Le format spécifié ici est "jour/mois/année heure:minute". De plus on définit le fuseau horaire utilisé par l'objet SimpleDateFormat sur UTC (Temps Universel Coordonné), ce qui est utile pour garantir que la conversion en temps Unix n'est pas affectée par le fuseau horaire local de la machine. Efnin, nous allons converitr donc la variable String (date + heure) en temps Unix.

La méthode createSeance() est ensuite appelée sur l'objet client, avec le nom et le temps Unix de la séance en tant que paramètres, pour créer la nouvelle séance sur le serveur. Ensuite, updateListsFromServer() est appelée pour mettre à jour les listes locales de séances et d'étudiants.

Enfin, la méthode appelle fillObjects() qui sert à associer les étudiants à chaque séance et à mettre à jour leur statut de présence. Elle fonctionne de la manière suivante :

Tout d'abord, elle parcourt chaque séance dans la liste seances. Pour chaque séance, elle parcourt également la liste étudiants. Pour chaque combinaison de séance et d'étudiant, elle ajoute l'étudiant à la liste des participants de la séance avec la méthode ajouterEtudiant().

Ensuite, elle récupère le statut de présence de l'étudiant pour cette séance en appelant getAttendanceStudent() sur l'objet client. Cette méthode retourne un statut qui indique si l'étudiant était présent ou absent. Enfin, le statut de présence de l'étudiant est mis à jour dans l'objet Seance avec la méthode setAbsence(). Cela garantit que chaque séance a les informations de présence à jour pour tous les étudiants.

```
// Ajoutez cette méthode pour remplir les objets étudiants et séances
private static void fillobjects() {
    for (Seance seance : seances) {
        for (Etudiant etudiant : etudiants) {
            seance.ajouterEtudiant(etudiant);
            int status = client.getAttendanceStudent(seance.getIdSeance(), etudiant.getIdEtudiant());
            seance.setAbsence(etudiant.getIdEtudiant(), status);
        }
    }
}
```

Le choix 6 est une fonction qui permet de supprimer un étudiant. Elle commence par afficher la liste des étudiants pour que l'utilisateur puisse voir les identifiants disponibles. L'utilisateur est ensuite invité à entrer l'ID de l'étudiant à supprimer. La fonction parcourt ensuite la liste des étudiants pour trouver celui correspondant à l'ID fourni. Si l'étudiant est trouvé (vérification réalisée par la valeur du booléen), la fonction utilise deleteStudent() sur l'objet client pour supprimer l'étudiant du serveur et met à jour les listes locales d'étudiants avec updateListsFromServer(). Si l'étudiant n'est pas trouvé, un message indiquant "[INFO] Étudiant non trouvé" est affiché.

```
// Fonction pour supprimer un étudiant
private static void supprimerEtudiant(Scanner scanner) {
    afficherEtudiants();
    System.out.println("Veuillez entrer l'ID de l'étudiant à supprimer |");
    int idsupp = scanner.nextInt();
    boolean etudiantTrouve = false;
    for (Etudiant etudiant : etudiants) {
        if (etudiant.getIdEtudiant() == idsupp) {
            client.deleteStudent(idsupp);
            etudiantTrouve = true;
            break;
        }
    }
    if(etudiantTrouve){
        updateListsFromServer();
    }else{
        System.out.println("[INFO] Etudiant non trouvé");
    }
}
```

Le choix 7, la méthode supprimerSeance() permet de supprimer une séance. Elle commence par afficher la liste des séances, permettant à l'utilisateur de visualiser les IDs disponibles. L'utilisateur doit ensuite entrer l'ID de la séance qu'il souhaite supprimer. La fonction parcourt la liste des séances pour trouver celle qui correspond à l'ID saisi. Si la séance est trouvée, elle appelle deleteSeance() sur l'objet client pour la supprimer du serveur, puis elle met à jour les listes locales avec updateListsFromServer() et réinitialise les objets en utilisant fillObjects(). Si aucune séance correspondante n'est trouvée, la fonction affiche "[INFO] Séance non trouvée".

Démonstration du client Java en CLI

Exécution du script et connexion au serveur

```
[etudiant@localhost client_java]$ java -jar java-1.0.jar
Bienvenue dans le gestionnaire d'absences
Veuillez choisir votre mode d'exécution:
[1] - Console
[2] - Interface graphique
1
Bienvenue dans le gestionnaire d'absences [Console]
Veuillez entrer l'adresse IP du serveur|127.0.0.1
Veuillez entrer le port du serveur|8081
[INFO] Loading certificate from: /home/etudiant/Bureau/iut-sae-302/dist/client_java/ssl/cert.pem
[INFO] Connected to the server | 127.0.0.1:8081

[INFO] - Connexion réussie au serveur127.0.0.1:8081
No students available
No seances available
-----Menu-----
[1] Visualiser les étudiants
[2] Visualiser les séances
[3] Enregistrer les absences d'une seance
[4] Creer un etudiant
[5] Creer une seance
[6] Supprimer un etudiant
[7] Supprimer une seance
[8] Visualiser les absences
[0] Quitter
Choix | ■
```

Création d'étudiants

```
Choix | 4
Veuillez entrer le nom de l'étudiant | Toto

[INFO] - Student 'Toto' created successfully.
No seances available
-----Menu-----
[1] Visualiser les étudiants
[2] Visualiser les séances
[3] Enregistrer les absences d'une seance
[4] Creer un etudiant
[5] Creer une seance
[6] Supprimer un etudiant
[7] Supprimer une seance
[8] Visualiser les absences
[0] Quitter
Choix | 4
Veuillez entrer le nom de l'étudiant | Titi

[INFO] - Student 'Titi' created successfully.
No seances available
-----Menu-----
[1] Visualiser les étudiants
[2] Visualiser les séances
[3] Enregistrer les absences d'une seance
[4] Creer un etudiant
[5] Creer une seance
[6] Supprimer un etudiant
[7] Supprimer une seance
[8] Visualiser les absences
[0] Quitter
Choix | ■
```

Création de séances

```
[0] Quitter
Choix | 5
Veuillez entrer le nom de la séance :
Mathematiques
Veuillez entrer la date de la séance (format JJ/MM/AAAA) :
05/02/2025
Veuillez entrer l'heure de la séance (format HH:MM) :
8:00
Format d'heure invalide. Veuillez réessayer.
Veuillez entrer l'heure de la séance (format HH:MM) :
08:00

[INFO] - Seance 'Mathematiques' created successfully.
```

```
Choix | 5
Veuillez entrer le nom de la séance :
Programmation
Veuillez entrer la date de la séance (format JJ/MM/AAAA) :
05/14/2025
Format de date invalide. Veuillez réessayer.
Veuillez entrer la date de la séance (format JJ/MM/AAAA) :
05/12/2025
Veuillez entrer l'heure de la séance (format HH:MM) :
10:30

[INFO] - Seance 'Programmation' created successfully.
```

Visualiser les étudiants

```
[0] Quitter
Choix | 1
=====
=====Students=====
Name = Toto | ID [0]
Name = Titi | ID [1]
=====
```

Visualiser les séances

```
Choix | 2
=====
=====Seances=====
Seance: Mathematiques [ID 0] - [05/02/2025 08:00]
Etudiant: Toto - Présence: Absent
Etudiant: Titi - Présence: Absent

Seance: Programmation [ID 1] - [05/12/2025 10:30]
Etudiant: Toto - Présence: Absent
Etudiant: Titi - Présence: Absent
=====
```

Faire l'appel des étudiants au sein d'une séance

```
Choix | 3
=====
=====Seances=====
Seance: Mathematiques [ID 0] - [05/02/2025 08:00]
Etudiant: Toto - Présence: Absent
Etudiant: Titi - Présence: Absent

Seance: Programmation [ID 1] - [05/12/2025 10:30]
Etudiant: Toto - Présence: Absent
Etudiant: Titi - Présence: Absent
=====

Entrez ID de la séance | 1
[Seance] - Programmation
L'étudiant Toto est-il présent? (O/N) | o
[INFO] - Absence recorded successfully.
Etudiant Toto marqué présent
L'étudiant Titi est-il présent? (O/N) | o
[INFO] - Absence recorded successfully.
Etudiant Titi marqué présent
[INFO] - Absences marquées pour la séance
Seance: Programmation [ID 1] - [05/12/2025 10:30]
Etudiant: Toto - Présence: Présent
Etudiant: Titi - Présence: Présent
```

Explication du mode GUI

Dans ce cas, dans le fichier main, le choix 2 a été effectué par l'utilisateur et donc la méthode executerModeGraphique() est exécutée. Cette méthode permet de créer un nouvel objet Accueil, qui est un objet représentant une fenêtre afin de se connecter au serveur mais de façon graphique cette fois-ci.

```
// Fonction pour exécuter le mode graphique
private static void executerModeGraphique() {
    System.out.println("Mode interface graphique");
    new Accueil();
}
```

En effet, nous avons choisi de prendre le temps de développer l'application en Java avec la possibilité d'avoir une interface graphique. Cela permet d'améliorer l'expérience de nos utilisateurs et de simplifier son utilisation. Nous espérons que cette interface graphique vous permettra d'apprécier l'utilisation de notre application au quotidien.

5.6/ La classe Accueil

Cette classe permet de créer une nouvelle fenêtre en Java afin de se connecter au serveur. Elle présente à l'utilisateur une fenêtre graphique où il peut saisir l'adresse IP et le port du serveur auquel ils souhaitent se connecter. Cette interface est cruciale pour établir la connexion entre le client et le serveur, occupant une position centrale dans le processus de démarrage de l'application.

Lors de linstanciation de la classe, une fenêtre principale est créée avec un titre visible "Absence Manager". Cette fenêtre, construite avec lobjet « JFrame », est paramétrée pour avoir une taille confortable de 800x600 pixels et est configurée pour se fermer proprement lorsque lutilisateur lindique. Afin dassurer une expérience utilisateur agréable, le fond de la fenêtre est configuré pour être une teinte sombre.

```
public class Accueil {
    private JFrame frame;
    private CardLayout cardLayout;
    private JPanel mainPanel;
    private JPanel connectPanel;

    private JTextField textFieldIP;
    private JTextField textFieldPort;

    public Accueil() {
        // Créer la fenêtre
        frame = new JFrame("Absence Manager");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800, 600);

        // Définir le fond sombre
        frame.getContentPane().setBackground(Color.DARK_GRAY);

        // Créer un CardLayout pour basculer entre les panneaux
        cardLayout = new CardLayout();
        mainPanel = new JPanel(cardLayout);

        // Initialiser les panneaux
        initializeConnectPanel();

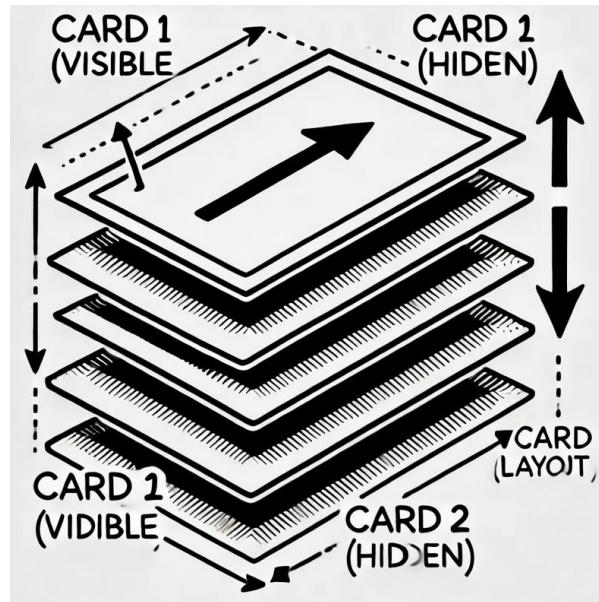
        // Ajouter les panneaux au CardLayout
        mainPanel.add(connectPanel, "connectPanel");

        // Ajouter le panneau principal à la fenêtre
        frame.getContentPane().add(mainPanel, BorderLayout.CENTER);

        // Rendre la fenêtre visible
        frame.setVisible(true);
    }
}
```

Linterface graphique est gérée par un »CardLayout », un gestionnaire de mise en page qui permet de naviguer entre différents panneaux. Cela se gère avec des panneaux et donc nous pouvons afficher des panneaux en changeant les uns après les autres ce qui rend dynamique lapplication. Dans cette configuration, le « mainPanel » est le conteneur principal intégrant actuellement le panneau de connexion (« connectPanel »).

Voici une image qui permet de représenter le fonctionnement du fonctionnement du CardLayout.



La méthode `initializeConnectPanel()`, appelée dans notre programme, permet de créer le « JPanel » qui va ensuite permettre d'insérer les éléments dans la fenêtre (et plus précisément dans ce panneau). Le panneau de connexion est configuré avec un « `GridBagLayout` », ce qui offre une flexibilité pour aligner les composants de manière ordonnée. Dans ce panneau, les utilisateurs trouvent des champs de saisie pour l'adresse IP et le port du serveur, chacun accompagné d'une étiquette descriptive en blanc pour une meilleure visibilité sur le fond sombre.

```
private void initializeConnectPanel() {
    // Panneau de connexion
    connectPanel = new JPanel();
    connectPanel.setBackground(Color.DARK_GRAY);
    connectPanel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);

    // Créer les champs de texte et les étiquettes
    JLabel labelIP = new JLabel("Server Address:");
    labelIP.setForeground(Color.WHITE);
    textFieldIP = new JTextField(10);

    JLabel labelPort = new JLabel("Server Port:");
    labelPort.setForeground(Color.WHITE);
    textFieldPort = new JTextField(5);

    // Créer le bouton de connexion
    JButton buttonConnect = new JButton("Connect");
    buttonConnect.setBackground(Color.GRAY);
    buttonConnect.setForeground(Color.WHITE);
```

Le bouton « Connect » est également stylisé avec des éléments (fond gris / texte blanc) et un objet de la classe ActionListener est associé à notre bouton. Cela permet d'exécuter le code suivant lorsque l'utilisateur clique sur le bouton. Lorsqu'un utilisateur clique sur le bouton « Connect », l'application récupère les informations saisies dans les champs de texte. Le port est validé pour s'assurer qu'il se situe dans les limites appropriées (0 à 65535), minimisant ainsi les erreurs de saisie qui pourraient entraîner des échecs de connexion. Si la connexion est validée et l'accès au serveur est établi, la fenêtre actuelle se ferme et une nouvelle fenêtre « Connected » est lancée, signalant que l'application a réussi à se connecter. Cependant, en cas d'échec, un message d'erreur simple et direct guide l'utilisateur pour vérifier ses informations de connexion.

```
// Créer le bouton de connexion
JButton buttonConnect = new JButton("Connect");
buttonConnect.setBackground(Color.GRAY);
buttonConnect.setForeground(Color.WHITE);

// Ajouter des actions au bouton de connexion

ActionListener actionlisten = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String ip = textFieldIP.getText();
        int port;
        try {
            port = Integer.parseInt(textFieldPort.getText());
            if (port < 0 || port > 65535) {
                throw new NumberFormatException("Port number out of range");
            }
            Client client = Client.getInstance(); // Initialisation de l'objet Client

            if (client.connectToServer(ip, port)) {

                frame.dispose(); // Fermer la fenêtre de connexion
                new Connected(client); // Ouvrir la fenêtre Connected

            } else {
                JOptionPane.showMessageDialog(frame, "Failed to connect to the server", "Error", JOptionPane.ERROR_MESSAGE);
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(frame, "Invalid port number", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
};
buttonConnect.addActionListener(actionlisten);
```

Ensuite, le programme va positionner les composants de saisie pour l'adresse IP et le port du serveur. Il utilise un objet GridBagConstraints (gbc) pour spécifier l'emplacement de chaque composant en termes de lignes (gridy) et de colonnes (gridx) dans une grille.

Pour la première ligne de la grille, l'étiquette labelIP est placée à la colonne 0 et la ligne 0, suivie du champ de texte textFieldIP à la colonne 1 de la même ligne. Cela permet à l'étiquette et au champ de saisie de l'adresse IP d'être alignés côté à côté.

Ensuite, le processus est répété pour l'étiquette et le champ de saisie du port. L'étiquette labelPort est placée à la colonne 0 de la deuxième ligne (ligne 1), et le champ de texte textFieldPort à la colonne 1 de la même ligne. Cela permet d'avoir une présentation claire et organisée des champs de saisie pour l'utilisateur. Le bouton de connexion buttonConnect est ensuite ajouté. Il est positionné sur la troisième ligne (ligne 2), mais cette fois-ci, sa largeur s'étend sur deux colonnes (gridwidth = 2), et il est centré

horizontalement dans le panneau (anchor = GridBagConstraints.CENTER). Ce choix de conception aide à donner une place centrale au bouton, facilitant son utilisation.

Enfin, un label contenant un texte de copyright est ajouté en bas du panneau (ligne 3). Il s'étend également sur deux colonnes et est centré, assurant une uniformité visuelle à l'ensemble du panneau. Le texte est en blanc pour se détacher sur le fond, améliorant ainsi la lisibilité.

```
// Ajouter les composants au panneau de connexion avec GridBagConstraints
gbc.gridx = 0;
gbc.gridy = 0;
connectPanel.add(labelIP, gbc);

gbc.gridx = 1;
gbc.gridy = 0;
connectPanel.add(textFieldIP, gbc);

gbc.gridx = 0;
gbc.gridy = 1;
connectPanel.add(labelPort, gbc);

gbc.gridx = 1;
gbc.gridy = 1;
connectPanel.add(textFieldPort, gbc);

gbc.gridx = 0;
gbc.gridy = 2;
gbc.gridwidth = 2;
gbc.anchor = GridBagConstraints.CENTER;
connectPanel.add(buttonConnect, gbc);

// Ajouter le copyright en bas du connectPanel
gbc.gridx = 0;
gbc.gridy = 3;
gbc.gridwidth = 2;
gbc.anchor = GridBagConstraints.CENTER;
JLabel copyright = new JLabel("© 2024 Absence Manager | All Rights Reserved\nBERSIN & BONNEAU & GOYA & PUCCETI");
copyright.setForeground(Color.WHITE);
connectPanel.add(copyright, gbc);
```

5.7/ La classe Connected

La classe « Connected » est la seconde fenêtre qui est exécutée dans le programme si la connexion au serveur s'est établie correctement. Pour cela, lors de l'appel de cette classe, le programme passe en paramètre l'objet Client qui est utilisé dans le constructeur de cet objet.

```
public class Connected {  
    private JFrame frame;  
    private Client client;  
    private JPanel mainPanel;  
    private JPanel seancesContainer;  
  
    public Connected(Client client) {  
        this.client = client;  
        initialize();  
    }  
}
```

Lorsqu'un objet Connected est initialisé, celui-ci appelle une méthode initialize qui permet de créer une nouvelle fenêtre avec différents paramètres. Tout d'abord, le programme initialise un objet JFrame nommé "Absence Manager", qui représente la fenêtre principale de l'application. Cette fenêtre est positionnée avec un coin supérieur gauche aux coordonnées (100, 100) sur l'écran et a une taille de 800 par 600 pixels. Lorsqu'on ferme cette fenêtre, l'application se termine grâce à setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE). Le fond de la fenêtre est défini en gris foncé par frame.getContentPane().setBackground(Color.DARK_GRAY), et la disposition des composants à l'intérieur du cadre est gérée par un BorderLayout. Ensuite, un JPanel nommé mainPanel est créé avec la même couleur de fond et une disposition BorderLayout pour organiser des composants supplémentaires. Un autre JPanel, topPanel, est configuré avec un GridBagLayout pour intégrer les boutons en haut de l'interface. Les contraintes GridBagConstraints (gbcTop) sont définies pour contrôler l'espacement autour des composants à l'intérieur de ce panneau avec des marges de 10 pixels. Le composant est positionné au coin supérieur gauche en utilisant l'ancre GridBagConstraints.NORTHWEST. L'ensemble de cette configuration permet de structurer l'interface utilisateur de manière organisée et esthétique.

```
private void initialize() {  
    frame = new JFrame("Absence Manager");  
    frame.setBounds(100, 100, 800, 600);  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.getContentPane().setBackground(Color.DARK_GRAY);  
    frame.setLayout(new BorderLayout());  
  
    mainPanel = new JPanel(new BorderLayout());  
    mainPanel.setBackground(Color.DARK_GRAY);  
  
    // Panneau pour les boutons en haut  
    JPanel topPanel = new JPanel(new GridBagLayout());  
    topPanel.setBackground(Color.DARK_GRAY);  
    GridBagConstraints gbcTop = new GridBagConstraints();  
    gbcTop.insets = new Insets(10, 10, 10, 10);  
    gbcTop.gridx = 0;  
    gbcTop.gridy = 0;  
    gbcTop.anchor = GridBagConstraints.NORTHWEST;
```

Tout d'abord, un objet JButton nommé buttonDisconnect est créé avec l'étiquette « Disconnect ». Ce bouton est configuré pour avoir une taille spécifique de 120 par 30 pixels, un fond gris, et un texte blanc pour une meilleure lisibilité. L'effet visuel qui entoure le bouton lorsque celui-ci est sélectionné est désactivé avec setFocusPainted(false). De plus, une bordure blanche d'un pixel d'épaisseur et arrondie est appliquée avec setBorder(new LineBorder(Color.WHITE, 1, true)).

Ensuite, un écouteur d'action (ActionListener) est défini pour le bouton. Cette action est déclenchée lorsque l'utilisateur clique sur « Disconnect ». L'intérieur de la méthode actionPerformed contient un bloc try-catch qui tente de fermer les ressources client en appelant client.closeResources(), ferme la fenêtre courante avec frame.dispose(), et crée un nouvel objet Accueil (classe expliquée précédemment qui va permet à l'utilisateur de se connecter au serveur), probablement pour retourner à l'écran d'accueil de l'application. Si une exception est levée lors de ces opérations, ses détails sont affichés dans la console avec ex.printStackTrace().

Enfin, le bouton "Disconnect" est ajouté au topPanel, un panneau probablement destiné à contenir des contrôles ou des options de navigation situés en haut de l'interface utilisateur. L'ajout de ce bouton utilise les mêmes contraintes préalablement définies par gbcTop.

```
// Créer et configurer le bouton "Disconnect"
JButton buttonDisconnect = new JButton("Disconnect");
buttonDisconnect.setPreferredSize(new Dimension(120, 30)); // Taille carrée et petite
buttonDisconnect.setBackground(Color.GRAY);
buttonDisconnect.setForeground(Color.WHITE);
buttonDisconnect.setFocusPainted(false); // Désactiver l'effet de peinture
buttonDisconnect.setBorder(new LineBorder(Color.WHITE, 1, true));

ActionListener actionlistendisconnect = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            client.closeResources();
            frame.dispose();
            new Accueil();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
};
buttonDisconnect.addActionListener(actionlistendisconnect);

// Ajouter le bouton "Disconnect" au panneau du haut
topPanel.add(buttonDisconnect, gbcTop);
```

Ensuite, on va aussi mettre un label en haut de la fenetre qui va permettre d'afficher le serveur sur lequel l'utilisateur est connecté. On ajoute aussi ce label au « gbcTop ».

```
// Créer et configurer le label "SRV"
JLabel labelSrv = new JLabel("SRV CONNECTED | " + client.getServerIP() + ":" + client.getServerPort());
labelSrv.setForeground(Color.WHITE);
gbcTop.gridx = 1;
gbcTop.insets = new Insets(15, 10, 10, 10); // Ajouter un espace vertical pour centrer le label
topPanel.add(labelSrv, gbcTop);
```

Le bouton buttonStudent est initialisé pour afficher le texte "Manage Students". Sa taille est fixée à 120 par 30 pixels, et il est configuré pour avoir un fond gris avec un texte en blanc, améliorant sa visibilité sur le fond sombre que nous avons configuré précédemment. De plus, le bouton est encadré par une bordure blanche d'un pixel d'épaisseur avec des coins arrondis, grâce à setBorder(new LineBorder(Color.WHITE, 1, true)).

Un ActionListener est également défini pour ce bouton, spécifiant ce qui doit se produire lorsqu'un utilisateur clique dessus. La méthode actionPerformed est implémentée pour instancier un nouvel objet ManageStudents (représentant une Classe qui crée une fenêtre permettant de gérer les étudiants), en passant un objet client en tant que paramètre. Tout cela se passe à l'intérieur d'un bloc try-catch, qui attrape et imprime toute exception qui pourrait se produire lors de l'exécution, en utilisant ex.printStackTrace(). Cette gestion des erreurs assure que tout problème inattendu est signalé sans faisant «bugée» complètement l'application. Enfin, le ActionListener est ajouté au bouton buttonStudent via addActionListener(actionlistenstudent), activant cette logique lorsque le bouton est cliqué.

```
// Créer et configurer le bouton "Manage Students"
JButton buttonStudent = new JButton("Manage Students");
buttonStudent.setPreferredSize(new Dimension(120, 30)); // Taille carrée et petite
buttonStudent.setBackground(Color.GRAY);
buttonStudent.setForeground(Color.WHITE);
buttonStudent.setFocusPainted(false); // Désactiver l'effet de peinture
buttonStudent.setBorder(new LineBorder(Color.WHITE, 1, true));

ActionListener actionlistenstudent = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            new ManageStudents(client);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
};

buttonStudent.addActionListener(actionlistenstudent);

// Ajouter le bouton "Manage Students" au panneau du haut
gbcTop.gridx = 2;
gbcTop.insets = new Insets(10, 10, 10, 10); // Réinitialiser les insets pour le bouton
topPanel.add(buttonStudent, gbcTop);
```

Nous allons effectuer une configuration similaire au bouton ManageSeances. La différence va être que lorsque le bouton est cliqué, un nouveau objet de la classe ManageSeances (avec comme paramètre le client toujours) afin de lancer une nouvelle fenêtre qui va permettre de gérer les séances.

```

// Ajouter le bouton "Manage Students" au panneau du haut
gbcTop.gridx = 2;
gbcTop.insets = new Insets(10, 10, 10, 10); // Réinitialiser les insets pour le bouton
topPanel.add(buttonStudent, gbcTop);

// Créer et configurer le bouton "Manage Séances"
JButton buttonSeance = new JButton("Manage Séances");
buttonSeance.setPreferredSize(new Dimension(120, 30)); // Taille carrée et petite
buttonSeance.setBackground(Color.GRAY);
buttonSeance.setForeground(Color.WHITE);
buttonSeance.setFocusPainted(false); // Désactiver l'effet de peinture
buttonSeance.setBorder(new LineBorder(Color.WHITE, 1, true));

ActionListener actionlistenseance = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            new ManageSeances(client, Connected.this);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
};
buttonSeance.addActionListener(actionlistenseance);

```

Par la suite, nous allons ajouter le bouton `buttonSeance` au panneau du haut. Ensuite, ce panneau sera ajouté au panneau principal, qui lui-même sera ajouté à la fenêtre. Grâce à la fonction `addSeancePanel()`, prenant comme paramètre le panneau principal, nous ajouterons à celui-ci un panneau permettant d'afficher les séances au centre de l'écran pour l'utilisateur.

```

// Ajouter le bouton "Manage Séances" au panneau du haut
gbcTop.gridx = 3;
topPanel.add(buttonSeance, gbcTop);

// Ajouter le panneau du haut au panneau principal
mainPanel.add(topPanel, BorderLayout.NORTH);

// Ajouter un panneau pour les séances au centre
addSeancePanel(mainPanel);

// Ajouter le panneau principal à la fenêtre
frame.getContentPane().add(mainPanel, BorderLayout.CENTER);
frame.setVisible(true);

```

La méthode débute par créer un panneau (`seancesContainer`) pour afficher ces séances, en le configurant avec un `GridBagLayout` pour assurer une disposition flexible. Le panneau a un fond gris foncé et une bordure vide pour ajouter un espace autour du contenu (semblable aux précédents). Un objet `GridBagConstraints` est initialisé pour gérer l'espacement (`insets`) entre les composants, les remplissant horizontalement et positionnant les composants à la première colonne et ligne (index 0,0).

```

public void addSeancePanel(JPanel mainPanel) {
    // Récupérer la liste des séances du client
    List<Seance> seances = client.getSeances();

    seancesContainer = new JPanel();
    seancesContainer.setLayout(new GridBagLayout());
    seancesContainer.setBackground(Color.DARK_GRAY);
    seancesContainer.setBorder(new EmptyBorder(20, 20, 20, 20)); // Ajouter un padding autour du conteneur

    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridx = 0;
    gbc.gridy = 0;

```

Grace à une boucle itérative, pour chaque objet Seance dans la liste, un nouveau JPanel est généré. Ce panneau, appelé seancePanel, est configuré avec des paramètres semblables aux précédents (déjà expliqués dans la documentation).

Chaque séance a un temps Unix, qui représente le moment de la séance que l'on récupère via la méthode getUnixTime() implémentée dans notre classe Seance. Ce temps Unix est converti en une date lisible en utilisant la classe Instant et le DateTimeFormatter. L'instant est d'abord créé à partir des secondes Unix, puis formaté pour une représentation lisible en utilisant le style MEDIUM, la locale par défaut de l'utilisateur, et le fuseau horaire du système local.

Ensuite, une étiquette (JLabel) est créée pour afficher le nom de la séance (récupéré via un getter de la classe Seance avec la méthode getNomSeance()) et la date formatée, en les combinant en une seule chaîne. Le texte de cette étiquette est défini en blanc pour se détacher sur le fond gris foncé. Cette disposition permet d'afficher clairement chaque séance avec son nom et sa date dans l'interface utilisateur.

```

// Créer un panneau pour chaque séance
for (Seance seance : seances) {
    JPanel seancePanel = new JPanel();
    seancePanel.setBackground(Color.DARK_GRAY);
    seancePanel.setLayout(new GridBagLayout());
    seancePanel.setBorder(new EmptyBorder(10, 10, 10, 10));

    // Convertir le temps Unix en date lisible
    long unixSeconds = seance.getUnixTime();
    Instant instant = Instant.ofEpochSecond(unixSeconds);
    String formattedDate = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM)
        .withLocale(Locale.getDefault())
        .withZone(ZoneId.systemDefault())
        .format(instant);

    JLabel labelSeance = new JLabel(seance.getNomSeance() + " - " + formattedDate);
    labelSeance.setForeground(Color.WHITE);

```

Ensuite, nous allons créer un nouveau bouton (l'explication concernant sa configuration a déjà été fournie dans la documentation précédente, nous ne la répéterons donc pas ici). L'action déclenchée par ce bouton consiste à créer un nouvel objet de la classe Attendance, qui permet d'ouvrir une fenêtre pour cocher les absences des étudiants dans le cadre d'une séance.

```

// Créer et configurer le bouton "Take Attendance"
JButton seanceButton = new JButton("Take Attendance");
seanceButton.setBackground(Color.GRAY);
seanceButton.setForeground(Color.WHITE);
seanceButton.setFocusPainted(false);
seanceButton.setBorder(new LineBorder(Color.WHITE, 1, true)); // Rounded border

ActionListener actionlistenseance = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        new Attendance(client, seance);
    }
};
seanceButton.addActionListener(actionlistenseance);

```

Ensute, nous allons créer un nouveau bouton (l'explication concernant sa configuration a déjà été fournie dans la documentation précédente, nous ne la répéterons donc pas ici). L'action déclenchée par ce bouton consiste à créer un nouvel objet de la classe ViewAttendance, qui permet d'ouvrir une fenêtre pour juste visualiser les absences des étudiants dans le cadre d'une séance.

```

// Créer et configurer le bouton "View Attendance"
JButton seanceViewAttendance = new JButton("View Attendance");
seanceViewAttendance.setBackground(Color.GRAY);
seanceViewAttendance.setForeground(Color.WHITE);
seanceViewAttendance.setFocusPainted(false);
seanceViewAttendance.setBorder(new LineBorder(Color.WHITE, 1, true)); // Rounded border
ActionListener actionlistenviewseance = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        new ViewAttendance(client, seance);
    }
};

seanceViewAttendance.addActionListener(actionlistenviewseance);

```

Ensute, nous allons disposer les différents les éléments sur le « seancePanel » avec des GridBagConstraints() qui est un element dont son fonctionnement a été déjà été explique précédemment. Tout d'abord, une étiquette (labelSeance) est ajoutée à seancePanel à la position de la première colonne (0) sur la première ligne (0), alignée à gauche avec l'ancre GridBagConstraints.WEST.

Ensute, un espace horizontal est créé pour séparer les composants. Cela se fait avec Box.createHorizontalStrut(20), qui ajoute un espace fixe de 20 pixels. Cet espace est placé à la deuxième colonne (1) et utilise weightx = 1.0 pour s'assurer qu'il s'étend si nécessaire, offrant une flexibilité en termes de taille.

Le bouton principal de la séance (seanceButton) est placé à la troisième colonne (2), aligné à droite avec GridBagConstraints.EAST.

Un autre espace horizontal est ajouté après le bouton principal pour séparer visuellement les composants suivants. Cet espace est placé à la quatrième colonne (3) avec weightx = 0.1, offrant une proportion d'espace flexible plus petite comparée à l'autre, car son poids est inférieur.

Finalement, un bouton pour voir les présences (seanceViewAttendance) est placé à la cinquième colonne (4), également aligné à droite.

Après avoir configuré et ajouté tous ces composants dans seancePanel, le panneau entier pour la séance est ajouté à seancesContainer avec la ligne suivante disponible (gbc.gridx++ indique l'incrémentation de la ligne pour la prochaine insertion). Cette logique est répétée dans une boucle, assurant que chaque séance obtient son propre ensemble de composants affichés à l'écran.

```
// Ajouter les composants au panneau de la séance avec GridBagConstraints
GridBagConstraints gbcLabel = new GridBagConstraints();
gbcLabel.gridx = 0;
gbcLabel.gridy = 0;
gbcLabel.anchor = GridBagConstraints.WEST;
seancePanel.add(labelSeance, gbcLabel);

GridBagConstraints gbcSpace = new GridBagConstraints();
gbcSpace.gridx = 1;
gbcSpace.gridy = 0;
gbcSpace.weightx = 1.0; // Ajuster la largeur de l'espace
seancePanel.add(Box.createHorizontalStrut(20), gbcSpace);

GridBagConstraints gbcButton = new GridBagConstraints();
gbcButton.gridx = 2;
gbcButton.gridy = 0;
gbcButton.anchor = GridBagConstraints.EAST;
seancePanel.add(seanceButton, gbcButton);

// Ajouter un espace horizontal entre les boutons
GridBagConstraints gbcSpaceBetweenButtons = new GridBagConstraints();
gbcSpaceBetweenButtons.gridx = 3;
gbcSpaceBetweenButtons.gridy = 0;
gbcSpaceBetweenButtons.weightx = 0.1; // Ajuster la largeur de l'espace
seancePanel.add(Box.createHorizontalStrut(20), gbcSpaceBetweenButtons);

GridBagConstraints gbcButtonView = new GridBagConstraints();
gbcButtonView.gridx = 4;
gbcButtonView.gridy = 0;
gbcButtonView.anchor = GridBagConstraints.EAST;
seancePanel.add(seanceViewAttendance, gbcButtonView);

gbc.gridx++;
seancesContainer.add(seancePanel, gbc);
```

La méthode refreshSeances est utilisée pour mettre à jour l'affichage des séances dans l'interface utilisateur. Elle commence par retirer le conteneur actuel des séances, seancesContainer, du panneau principal, mainPanel. Cela est nécessaire afin de supprimer les données actuellement affichées, ce qui est utile lorsque vous souhaitez actualiser les informations. Ensuite, la méthode appelle addSeancePanel(mainPanel) que l'on vient d'expliquer. Cette méthode va permettre de mettre à jour le panneau avec la liste des séances notamment lorsque l'utilisateur crée ou supprime une séance.

```
// Method to refresh the seances list
public void refreshSeances() {
    mainPanel.remove(seancesContainer);
    addSeancePanel(mainPanel);
}
```

5.8/ La classe Attendance

La classe Java Attendance est conçue pour gérer la présence des étudiants à une séance en créant une fenêtre avec le même concept de programmation que pour les autres fenêtres. Elle contient plusieurs attributs privés, dont un JFrame qui est utilisé pour l'interface graphique de l'application, un Client qui est utilisé pour la connexion ou la communication avec un serveur, ainsi que des informations spécifiques à une séance : un identifiant unique seanceId, le nom de la séance seanceName, et un unixTime qui indique le moment de la séance. De plus, elle dispose d'une map (studentCheckBoxMap) qui associe chaque étudiant à une case à cocher, ce qui est utilisé pour marquer leur présence.

Le constructeur de la classe, Attendance(Client client, Seance seance), initialise ces attributs en utilisant un objet Seance qui fournit les informations nécessaires. Cela inclut l'identifiant de la séance, son nom, et le timestamp Unix. La map est initialisée en vue d'associer chaque étudiant à une case à cocher. Enfin, il est mentionné qu'une méthode initialize() est invoquée, bien que sa définition ne soit pas fournie dans le texte; cette méthode est probablement responsable de configurer l'interface utilisateur et de préparer d'autres éléments nécessaires pour le fonctionnement de la classe.

```
public class Attendance {
    private JFrame frame;
    private Client client;
    private int seanceId;
    private String seanceName;
    private Long unixTime;
    private Map<Etudiant, JCheckBox> studentCheckBoxMap;

    public Attendance(Client client, Seance seance) {
        this.client = client;
        this.seanceId = seance.getIdSeance();
        this.seanceName = seance.getNomSeance();
        this.unixTime = seance.getUnixTime();
        this.studentCheckBoxMap = new HashMap<>();
        initialize();
    }
}
```

Au départ, la méthode initialize, qui est exécutée lors de l'appel au constructeur de notre classe, a déjà été expliquée lors de la description de la classe précédente, par exemple. Ce début sert uniquement à afficher les informations du serveur en haut de la fenêtre, comme nous l'avons déjà fait pour les objets de la classe Connected.

Ensuite, dans cette méthode, nous allons ajouter un panneau "validate" au panneau principal. L'explication sur le style graphique du bouton ne sera pas répétée ici, car elle est exactement la même que pour les autres boutons. Si vous souhaitez obtenir plus de détails sur les méthodes utilisées pour styliser notre bouton, vous les trouverez dans la description des méthodes précédentes.

Un ActionListener est ensuite attaché au bouton. L'ActionListener est une interface qui permet de définir ce qui doit se passer lorsque le bouton est pressé. Dans ce cas, l'écouteur parcourt une collection d'étudiants, représentée par des paires (étudiant, case à cocher) dans une structure appelée studentCheckBoxMap. Pour chaque étudiant, il vérifie l'état de la case à cocher associée. Si la case est

cochée, cela signifie que l'étudiant est présent, et la méthode client.setAbsence est appelée pour enregistrer son état de présence. Si la case n'est pas cochée, cela signifie que l'étudiant est absent, et l'appel enregistre son absence. Enfin, après avoir vérifié l'état de toutes les cases à cocher, la fenêtre actuelle de l'application est fermée. Pour avoir des informations complémentaires sur la méthode setAbsence() du client, cela est expliqué dans la partie de la documentation technique expliquant cela.

```
// Ajouter un panneau pour le bouton "Validate"
JPanel topLeftPanel = new JPanel(new BorderLayout());
topLeftPanel.setBackground(Color.DARK_GRAY);

// Créer et configurer le bouton "Validate"
JButton validateButton = new JButton("Validate");
validateButton.setPreferredSize(new Dimension(80, 30));
validateButton.setBackground(Color.GRAY);
validateButton.setForeground(Color.GREEN);
validateButton.setFocusPainted(false);
validateButton.setBorder(new LineBorder(Color.WHITE, 1, true));
ActionListener validateListener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Vérifier l'état des cases à cocher et effectuer des actions en fonction de leur état
        for (Map.Entry<Etudiant, JCheckBox> entry : studentCheckBoxMap.entrySet()) {
            Etudiant student = entry.getKey();
            JCheckBox checkBox = entry.getValue();
            if (checkBox.isSelected()) {
                // Action si la case est cochée
                client.setAbsence(seanceId, student.getIdEtudiant(), statut:1);

            } else {
                // Action si la case n'est pas cochée
                client.setAbsence(seanceId, student.getIdEtudiant(), statut:0);
            }
        }
        // Fermeture de la fenêtre actuelle
        frame.dispose();
    }
}
```

Après avoir configuré le bouton "Validate", nous l'ajoutons à un panneau situé en haut à gauche de l'interface. Ce panneau utilise un agencement de type BorderLayout, ce qui signifie que le bouton est placé spécifiquement dans la partie nord du panneau (c'est-à-dire en haut).

Ensuite, ce panneau contenant le bouton est ajouté au panneau principal (mainPanel), à la position ouest (à gauche) selon le BorderLayout utilisé par le mainPanel. Cela permet d'organiser l'interface utilisateur de manière structurée, avec le bouton "Validate" positionné sur la partie supérieure gauche.

Après cela, nous appelons la méthode addStudentsPanel(mainPanel) qui ajoute les panneaux contenant les informations des étudiants au panneau principal. Cette étape n'est pas explicitement détaillée ici, mais elle implique vraisemblablement l'ajout de composants ou d'éléments liés aux étudiants dans l'interface.

Enfin, le panneau principal, incluant tous ces éléments (le bouton, les informations des étudiants, etc.), est ajouté au contenu principal de la fenêtre (frame) avec l'organisation BorderLayout.CENTER, ce qui signifie qu'il occupe l'espace central de la fenêtre.

Pour conclure, la fenêtre est rendue visible par l'appel de frame.setVisible(true), ce qui actualise l'affichage de l'application pour présenter l'interface utilisateur construite avec ces différents éléments.

```
// Ajouter le bouton "Validate" à son panneau
topLeftPanel.add(validateButton, BorderLayout.NORTH);
// Ajouter le panneau du bouton en haut à gauche
mainPanel.add(topLeftPanel, BorderLayout.WEST);
// Ajouter les panneaux des étudiants
addStudentsPanel(mainPanel);
// Ajouter le panneau principal à la fenêtre
frame.getContentPane().add(mainPanel, BorderLayout.CENTER);
// Rendre la fenêtre visible
frame.setVisible(true);
```

La méthode addStudentsPanel() permet de gérer les absences au sein d'une séance. Pour cela, on récupère la liste des étudiants. Ensuite, on crée, pour chaque étudiant, via l'utilisation d'une boucle itérative for, un nouveau panel associé à cet étudiant. Ce panel affiche le nom de l'étudiant ainsi qu'une case à cocher à côté de celui-ci. On vérifie ensuite la méthode getAttendanceStudent() du client en lui passant comme paramètres l'ID de la séance sélectionnée et l'ID de l'étudiant. Si l'étudiant est présent, la case sera cochée, sinon elle restera décochée. Cela permet de s'assurer que si, sur le serveur, un étudiant X est marqué comme présent, alors la case à cocher sera déjà activée.

```
private void addStudentsPanel(JPanel mainPanel) {
    List<Etudiant> students = client.getStudents(); // Assuming this method exists in Client class

    JPanel studentsContainer = new JPanel();
    studentsContainer.setLayout(new GridBagLayout());
    studentsContainer.setBackground(Color.DARK_GRAY);
    studentsContainer.setBorder(new EmptyBorder(20, 20, 20, 20)); // Add padding around the container

    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridx = 0;
    gbc.gridy = 0;

    for (Etudiant student : students) {
        JPanel studentPanel = new JPanel();
        studentPanel.setBackground(Color.DARK_GRAY);
        studentPanel.setLayout(new GridBagLayout());
        studentPanel.setBorder(new EmptyBorder(10, 10, 10, 10)); // Add padding around each student panel

        JLabel labelStudent = new JLabel("Student: " + student.getNomEtudiant());
        labelStudent.setForeground(Color.WHITE);

        JCheckBox checkBox = new JCheckBox();
        checkBox.setBackground(Color.DARK_GRAY);
        checkBox.setForeground(Color.WHITE);

        int presence = client.getAttendanceStudent(seanceId, student.getIdEtudiant());
        if (presence == 1) {
            checkBox.setSelected(true);
        }
        // Store the checkbox in the map
        studentCheckBoxMap.put(student, checkBox);

        GridBagConstraints gbcLabel = new GridBagConstraints();
        gbcLabel.gridx = 0;
        gbcLabel.gridy = 0;
        gbcLabel.anchor = GridBagConstraints.WEST;
        studentPanel.add(labelStudent, gbcLabel);

        GridBagConstraints gbcCheckBox = new GridBagConstraints();
        gbcCheckBox.gridx = 1;
        gbcCheckBox.gridy = 0;
        gbcCheckBox.anchor = GridBagConstraints.EAST;
        studentPanel.add(checkBox, gbcCheckBox);

        gbc.gridy++;
        studentsContainer.add(studentPanel, gbc);
    }
}
```

Ensuite on ajoute le conteneur studentsContainer au panneau principal (mainPanel) au centre (BorderLayout.CENTER). Ensuite, on met à jour l'affichage avec revalidate() pour recalculer la disposition et repaint() pour redessiner l'interface. Cela permet d'afficher correctement la liste des étudiants après son chargement.

```
mainPanel.add(studentsContainer, BorderLayout.CENTER);
mainPanel.revalidate();
mainPanel.repaint();
```

5.9/ La classe ManageSeances

La classe ManageSeances permet de créer une nouvelle fenêtre afin de pouvoir gérer les séances. Ainsi, nous allons pouvoir supprimer une séance existante ou en créer une nouvelle. Comme expliqué précédemment, nous allons utiliser un CardLayout en Java qui permet d'avoir plusieurs "cartes" que nous affichons de façon dynamique, permettant de passer d'un onglet à un autre tout en restant sur la même fenêtre. Ainsi, l'expérience pour l'utilisateur est agréable, et cela facilite aussi le code.

Semblable aux autres classes, cette classe prend comme paramètre dans son constructeur le client et aussi la fenêtre d'accueil qui est Connected.

```
public class ManageSeances {
    private JFrame frame;
    private Client client;
    private JPanel cards;
    private CardLayout cardLayout;
    private Connected connected;

    public ManageSeances(Client client, Connected connected) {
        this.client = client;
        this.connected = connected;
        initialize();
    }
}
```

Par la suite, nous configurons notre interface pour la gestion des séances. Pour commencer, nous créons une nouvelle fenêtre, que nous appelons "Manage Seances". Nous fixons sa taille et sa position avec les lignes frame.setBounds(100, 100, 800, 600), ce qui signifie qu'elle apparaîtra à l'écran avec une largeur de 800 pixels et une hauteur de 600 pixels, et elle sera placée à 100 pixels du bord gauche et 100 pixels du haut de l'écran.

Nous indiquons aussi à cette fenêtre de se fermer complètement lorsque l'utilisateur clique sur la croix rouge dans le coin (grâce à frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)). Ensuite, nous donnons à l'intérieur de la fenêtre une couleur de fond gris foncé avec frame.getContentPane().setBackgroundColor(Color.DARK_GRAY) pour l'apparence. Cela est identique à la configuration des fenêtres effectuées pour les autres classes déjà expliquées au sein de la documentation technique.

Ensuite, nous configurons la fenêtre pour organiser son contenu avec un format appelé BorderLayout, qui permet de placer des éléments de manière structurée (par exemple, en haut, en bas, au centre). Cette organisation va nous aider à disposer les différents composants de façon ordonnée.

```

private void initialize() {
    frame = new JFrame("Manage Séances");
    frame.setBounds(100, 100, 800, 600);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setBackground(Color.DARK_GRAY);
    frame.setLayout(new BorderLayout());

    cardLayout = new CardLayout();
    cards = new JPanel(cardLayout);

    JPanel mainPanel = new JPanel(new BorderLayout());
    mainPanel.setBackground(Color.DARK_GRAY);

    // Panneau pour les boutons en haut
    JPanel topPanel = new JPanel(new GridBagLayout());
    topPanel.setBackground(Color.DARK_GRAY);

    addTopPanelButtons(topPanel);

    // Ajouter le panneau du haut au panneau principal
    mainPanel.add(topPanel, BorderLayout.NORTH);

    // Ajouter un panneau pour les séances au centre
    addSeancePanel(mainPanel);

    // Ajouter le panneau principal à la carte
    cards.add(mainPanel, "mainPanel");
}

```

Dans cette partie du code, nous construisons un "panel secondaire", c'est-à-dire une autre partie de notre interface utilisateur. Ce panel a son propre ensemble d'éléments que l'utilisateur peut voir et avec lesquels il peut interagir.

Nous commençons par créer ce panneau secondaire (secondPanel) en utilisant une disposition appelée BorderLayout, ce qui nous permet de disposer les éléments de manière structurée. Nous lui donnons aussi un fond gris foncé pour rester cohérent avec le reste de l'interface.

Ensuite, nous créons un autre petit panel, topPanelTwo, qui est destiné à contenir des éléments alignés à gauche, comme des boutons. Pour cela, nous utilisons un FlowLayout qui facilite cet arrangement. Ce panneau a également un fond gris foncé.

Nous ajoutons un bouton nommé "Back" (buttonBackMain). Ce bouton a une taille préférée de 120 par 30 pixels et est stylisé avec une couleur de fond grise et du texte blanc. Nous lui donnons aussi un style simple avec une bordure blanche et bloquons l'effet de mise au point (focus) quand on clique dessus.

Le bouton "Back" est programmé pour revenir à l'écran principal quand on clique dessus. Cela est géré par un "écouteur d'action" (action listener) qui dit, en gros, "quand tu es cliqué, montre la carte avec le nom 'mainPanel'".

Nous ajoutons ce bouton à notre panneau en haut, topPanelTwo, puis nous plaçons ce panneau en haut du secondPanel.

Ensuite, nous ajoutons un autre petit panel central (centerPanel) au secondPanel, qui prendra en charge les champs de saisie de texte. Il utilise GridBagLayout pour arranger les éléments de manière plus flexible, et son fond est aussi en gris foncé.

Pour organiser les champs, nous utilisons un objet GridBagConstraints (gbcCenter) qui définit comment chaque élément sera positionné. Par exemple, il règle l'espacement autour des éléments pour qu'il y ait 10 pixels de marge de chaque côté. Nous ajoutons une étiquette de texte "Name:" avec du texte en blanc (nameLabel) dans le centerPanel, suivie d'un champ de texte (nameField) où l'utilisateur peut écrire jusqu'à une quinzaine de caractères. Ce champ a une taille préférée de 200 par 30 pixels. Après cela, nous ajoutons une autre étiquette de texte "Date:" (dateLabel), avec les mêmes styles que précédemment. Cela se fait en ajustant les contraintes de position pour placer ces éléments dans le bon ordre.

```
// Panel secondaire
JPanel secondPanel = new JPanel(new BorderLayout());
secondPanel.setBackground(Color.DARK_GRAY);

// Utiliser un FlowLayout aligné à gauche pour topPanelTwo
JPanel topPanelTwo = new JPanel(new FlowLayout(FlowLayout.LEFT));
topPanelTwo.setBackground(Color.DARK_GRAY);

JButton buttonBackMain = new JButton("Back");
buttonBackMain.setPreferredSize(new Dimension(120, 30));
buttonBackMain.setBackground(Color.GRAY);
buttonBackMain.setForeground(Color.WHITE);
buttonBackMain.setFocusPainted(false);
buttonBackMain.setBorder(new LineBorder(Color.WHITE, 1, true));
buttonBackMain.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        cardLayout.show(cards, "mainPanel");
    }
});

topPanelTwo.add(buttonBackMain);
secondPanel.add(topPanelTwo, BorderLayout.NORTH);

// Ajouter un panneau au centre du second panel pour les champs
JPanel centerPanel = new JPanel(new GridBagLayout());
centerPanel.setBackground(Color.DARK_GRAY);

GridBagConstraints gbcCenter = new GridBagConstraints();
gbcCenter.insets = new Insets(10, 10, 10, 10);
gbcCenter.gridx = 0;
gbcCenter.gridy = 0;
gbcCenter.anchor = GridBagConstraints.LINE_START;

JLabel nameLabel = new JLabel("Name:");
nameLabel.setForeground(Color.WHITE);
centerPanel.add(nameLabel, gbcCenter);

gbcCenter.gridx = 1;
JTextField nameField = new JTextField(15);
nameField.setPreferredSize(new Dimension(200, 30));
centerPanel.add(nameField, gbcCenter);

gbcCenter.gridx = 0;
gbcCenter.gridy++;
JLabel dateLabel = new JLabel("Date:");
dateLabel.setForeground(Color.WHITE);
centerPanel.add(dateLabel, gbcCenter);
```

Nous avons ajouté un calendrier à l'écran, connu sous le nom de JDateChooser. Cela permet aux utilisateurs de choisir facilement une date. Juste à côté, nous avons mis un petit élément qui ressemble à une boîte de sélection pour l'heure, appelée JSpinner. Il aide l'utilisateur à faire défiler et choisir un moment précis de la journée. Cela va donc permettre grâce à la classe JDateChooser() de créer une sorte de calendrier pour que l'utilisateur puisse choisir la date souhaitée.

```
// JDateChooser for Date Selection
gbcCenter.gridx = 1; // Positionner à la droite du label "Date"
JDateChooser dateChooser = new JDateChooser();
dateChooser.setPreferredSize(new Dimension(120, 30));
centerPanel.add(dateChooser, gbcCenter);

gbcCenter.gridx = 2; // Position pour le time spinner
SpinnerDateModel timeModel = new SpinnerDateModel();
JSpinner timeSpinner = new JSpinner(timeModel);
timeSpinner.setPreferredSize(new Dimension(80, 30));

// Format de l'affichage du temps
JSpinner.DateEditor timeEditor = new JSpinner.DateEditor(timeSpinner, "HH:mm");
SimpleDateFormat sdf = ((SimpleDateFormat) timeEditor.getFormat());
sdf.applyPattern("HH:mm");
timeSpinner.setEditor(timeEditor);

centerPanel.add(timeSpinner, gbcCenter);
```

Par la suite pour pouvoir valider la création d'une séance, on va ajouter une action associée au bouton validateButton. L'action associée va permettre de récupérer les données de la date et de l'heure et vérifie que celles-ci ne sont pas vides. Ensuite, nous allons combiner la date et l'heure afin de convertir ces données en temps unix afin de pouvoir créer la séance. Par la suite, on appelle la méthode createSeance() du client avec le nom et le temps unix que l'on a récupérée des champs entrés par l'utilisateur. Enfin, on va recharger le panneau afin de mettre à jour la liste ses séances.

Pour recharger le panneau principal de l'interface utilisateur, nous utilisons une méthode nommée reloadMainPanel, prenant en paramètre un objet JPanel appelé mainPanel. Ce processus commence par la suppression de tous les composants actuellement présents dans mainPanel à l'aide de la méthode removeAll().

Ensuite, nous créons un nouveau JPanel nommé topPanel, en utilisant un agencement de type GridBagLayout, qui facilite la gestion sophistiquée du positionnement des composants. Nous appliquons ensuite une couleur de fond foncée (DARK_GRAY) à ce panneau, améliorant ainsi l'aspect visuel de l'interface.

Nous ajoutons des boutons au topPanel en utilisant une méthode séparée appelée addTopPanelButtons(), ce qui implique que cette méthode gère l'ajout et la configuration des boutons nécessaires. Après cela, le topPanel est ajouté au nord du mainPanel en utilisant BorderLayout.NORTH, ce qui place le topPanel en haut du mainPanel.

Ensuite, nous appelons la méthode addSeancePanel() qui, est responsable de l'ajout de composants supplémentaires à mainPanel, probablement pour gérer ou afficher des séances.

Enfin, une fois que tous les composants nécessaires ont été ajoutés et les modifications effectuées, nous utilisons mainPanel.revalidate() pour dire au gestionnaire de mise en page de recalculer la disposition du mainPanel, et mainPanel.repaint() pour rafraîchir l'affichage du panneau, s'assurant que toutes les modifications sont bien visibles à l'écran.

```
ActionListener actionvalidatebutton = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String name = nameField.getText().trim(); // Trimmer les espaces pour le nom
        Date selectedDate = dateChooser.getDate();
        Calendar calendar = Calendar.getInstance();

        if (name.isEmpty()) {
            System.out.println("Veuillez entrer un nom.");
            return;
        }

        if (selectedDate == null) {
            System.out.println("Veuillez sélectionner une date.");
            return;
        }

        // Obtenir l'heure depuis le JSpinner
        Date time = (Date) timeSpinner.getValue();
        Calendar timeCalendar = Calendar.getInstance();
        timeCalendar.setTime(time);

        // Combiner la date et l'heure
        calendar.setTime(selectedDate);
        calendar.set(Calendar.HOUR_OF_DAY, timeCalendar.get(Calendar.HOUR_OF_DAY));
        calendar.set(Calendar.MINUTE, timeCalendar.get(Calendar.MINUTE));
        calendar.set(Calendar.SECOND, 0);
        calendar.set(Calendar.MILLISECOND, 0);

        Date finalDateTime = calendar.getTime();

        // Conversion de la date complète en timestamp Unix
        long unixTime = finalDateTime.getTime() / 1000L;

        // Appel de la méthode createSeance
        boolean result = client.createSeance(name, unixTime);
        if (result) {
            reloadMainPanel(mainPanel); // Recharger le panneau principal
            cardLayout.show(cards, "mainPanel");
        }
    }
};

validateButton.addActionListener(actionvalidatebutton);
```

Voici comment est implémentée la méthode reloadMainPanel().

```
private void reloadMainPanel(JPanel mainPanel) {
    mainPanel.removeAll();
    JPanel topPanel = new JPanel(new GridBagLayout());
    topPanel.setBackground(Color.DARK_GRAY);
    addTopPanelButtons(topPanel);
    mainPanel.add(topPanel, BorderLayout.NORTH);
    addSeancePanel(mainPanel);
    mainPanel.revalidate();
    mainPanel.repaint();
}
```

Enfin, nous allons nous intéresser à la méthode addTopPanelButtons(). Cette méthode ajoute deux boutons au panneau supérieur de l'interface utilisateur. Le premier bouton, intitulé "Back", est configuré avec une couleur de fond grise et un texte blanc. Lorsqu'il est cliqué, il ferme la fenêtre actuelle et rafraîchit la liste des séances. Le second bouton, nommé "Create Seance", partage la même couleur de fond mais avec un texte vert, et permet de créer une nouvelle séance. En cliquant sur ce bouton, l'utilisateur est dirigé vers un autre panneau de l'application, grâce à l'utilisation d'un gestionnaire de présentation en cartes. Ces deux boutons sont positionnés dans le panneau supérieur à l'aide d'un gestionnaire de mise en page, ce qui place le bouton "Back" sur la gauche, tandis que le bouton "Create Seance" est aligné à droite.

```
// Ajouter le bouton "Back" au panneau du haut
private void addTopPanelButtons(JPanel topPanel) {
    // Créer et configurer le bouton "Back"
    JButton buttonBack = new JButton("Back");
    buttonBack.setPreferredSize(new Dimension(120, 30));
    buttonBack.setBackground(Color.GRAY);
    buttonBack.setForeground(Color.WHITE);
    buttonBack.setFocusPainted(false);
    buttonBack.setBorder(new LineBorder(Color.WHITE, 1, true));
    ActionListener actionBack = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            frame.dispose();
            connected.refreshSeances(); // Rafraîchir la liste des séances
        }
    };
    buttonBack.addActionListener(actionBack);

    GridBagConstraints gbcBack = new GridBagConstraints();
    gbcBack.insets = new Insets(10, 10, 10, 10);
    gbcBack.gridx = 0;
    gbcBack.gridy = 0;
    gbcBack.anchor = GridBagConstraints.WEST;
    topPanel.add(buttonBack, gbcBack);

    // Créer et configurer le bouton "Create Seance"
    JButton buttonCreateSeance = new JButton("Create Seance");
    buttonCreateSeance.setPreferredSize(new Dimension(120, 30));
    buttonCreateSeance.setBackground(Color.GRAY);
    buttonCreateSeance.setForeground(Color.GREEN);
    buttonCreateSeance.setFocusPainted(false);
    buttonCreateSeance.setBorder(new LineBorder(Color.WHITE, 1, true));
    ActionListener actionCreateSeance = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            cardLayout.show(cards, "secondPanel");
        }
    };
    buttonCreateSeance.addActionListener(actionCreateSeance);

    GridBagConstraints gbcCreate = new GridBagConstraints();
    gbcCreate.insets = new Insets(10, 10, 10, 10);
    gbcCreate.gridx = 1;
    gbcCreate.gridy = 0;
    gbcCreate.anchor = GridBagConstraints.EAST;
    gbcCreate.weightx = 1.0; // Pour pousser le bouton à droite
    topPanel.add(buttonCreateSeance, gbcCreate);
}
```

Pour finir, intéressons-nous à la méthode `addSeancePanel()`. Cette méthode a pour but d'ajouter toutes les séances récupérées du serveur à l'intérieur du panneau principal. Elle commence par obtenir la liste des séances en interrogeant le client grâce à la méthode `getSeances()`. Ensuite, une boucle itérative `for` parcourt cette liste de séances. Chaque séance est représentée par un panneau individuel, avec des marges et des bordures pour assurer un affichage clair et espacé.

Durant cette itération, pour chaque séance, le panneau affiche le nom et l'heure de la séance, cette dernière étant convertie à partir du temps Unix pour la rendre lisible et compréhensible pour l'utilisateur. À côté du nom de la séance, un bouton "Delete Seance" permet à l'utilisateur de supprimer la séance de la liste. Lorsqu'un utilisateur clique sur ce bouton, la séance est retirée du serveur et du conteneur visuel grâce aux actions définies pour le bouton. Cela se fait via la méthode `deleteSeance()` de la classe `Client`, qui prend en paramètre l'ID de la séance à supprimer, récupéré suite au clic de l'utilisateur. Après la suppression, nous retirons tous les éléments du panneau, puis les rajoutons à nouveau, en utilisant les méthodes expliquées précédemment, afin de mettre à jour les informations et de refléter les changements introduits.

Les panneaux des séances sont ajoutés les uns après les autres à l'intérieur d'un autre panneau, `seancesContainer`, utilisant une structure de mise en page qui assure que chaque élément est correctement aligné et espacé horizontalement. Le rôle de cette mise en page est de maintenir une présentation ordonnée et claire des séances. À la fin de ce processus, le conteneur est intégré au panneau principal, qui est ensuite actualisé pour garantir que toutes les modifications sont visuellement appliquées et visibles pour l'utilisateur.

```

public void addSeancePanel(JPanel mainPanel) {
    List<Seance> seances = client.getSeances(); // Assuming this method exists in Client class

    JPanel seancesContainer = new JPanel();
    seancesContainer.setLayout(new GridBagLayout());
    seancesContainer.setBackground(Color.DARK_GRAY);
    seancesContainer.setBorder(new EmptyBorder(20, 20, 20, 20)); // Add padding around the container

    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets(10, 10, 10, 10);
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridx = 0;
    gbc.gridy = 0;

    for (Seance seance : seances) {
        JPanel seancePanel = new JPanel();
        seancePanel.setBackground(Color.DARK_GRAY);
        seancePanel.setLayout(new GridBagLayout());
        seancePanel.setBorder(new EmptyBorder(10, 10, 10, 10)); // Add padding around each seance panel

        // Convert Unix time to human-readable date and time
        long unixSeconds = seance.getUnixTime();
        Instant instant = Instant.ofEpochSecond(unixSeconds);
        String formattedDate = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM)
            .withLocale(Locale.getDefault())
            .withZone(ZoneId.systemDefault())
            .format(instant);

        JLabel labelSeance = new JLabel(seance.getNomSeance() + " - " + formattedDate);
        labelSeance.setForeground(Color.WHITE);

        JButton seanceButton = new JButton("Delete Seance");
        seanceButton.setBackground(Color.GRAY);
        seanceButton.setForeground(Color.RED);
        seanceButton.setFocusPainted(false);
        seanceButton.setBorder(new LineBorder(Color.WHITE, 1, true)); // Rounded border
        ActionListener actiondelete = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                client.deleteSeance(seance.getIdSeance());
                // Supprimer uniquement le panneau correspondant de seancesContainer
                seancesContainer.remove(seancePanel);
                seancesContainer.revalidate();
                seancesContainer.repaint();
            }
        };
        seanceButton.addActionListener(actiondelete);
    }
}

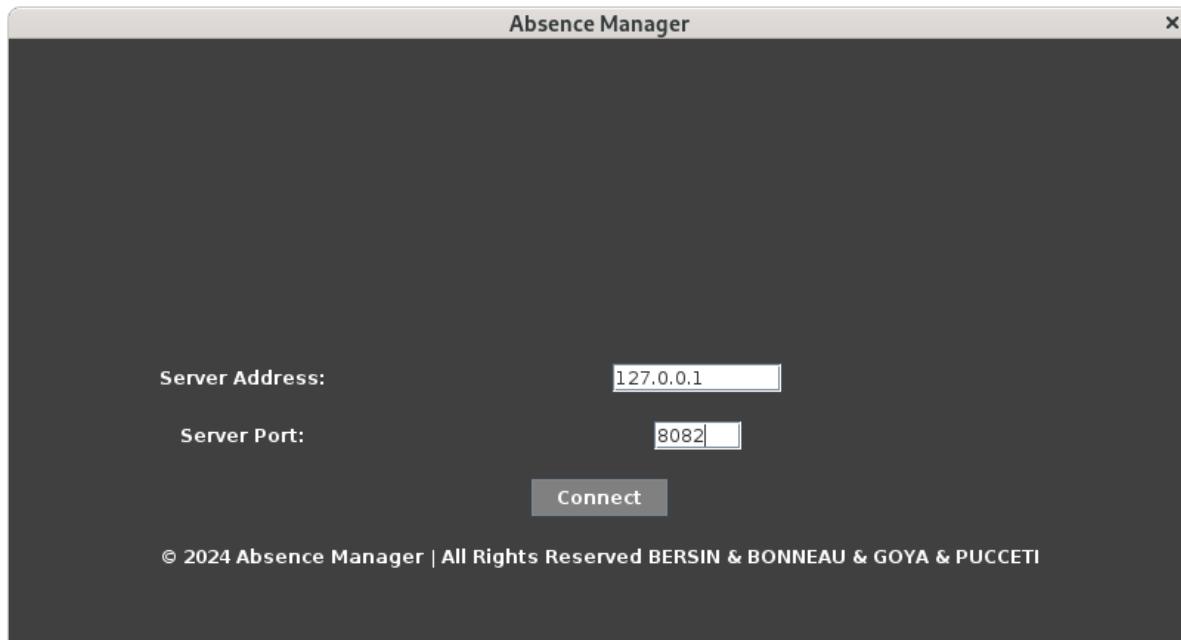
```

5.9/ La classe ManageStudents

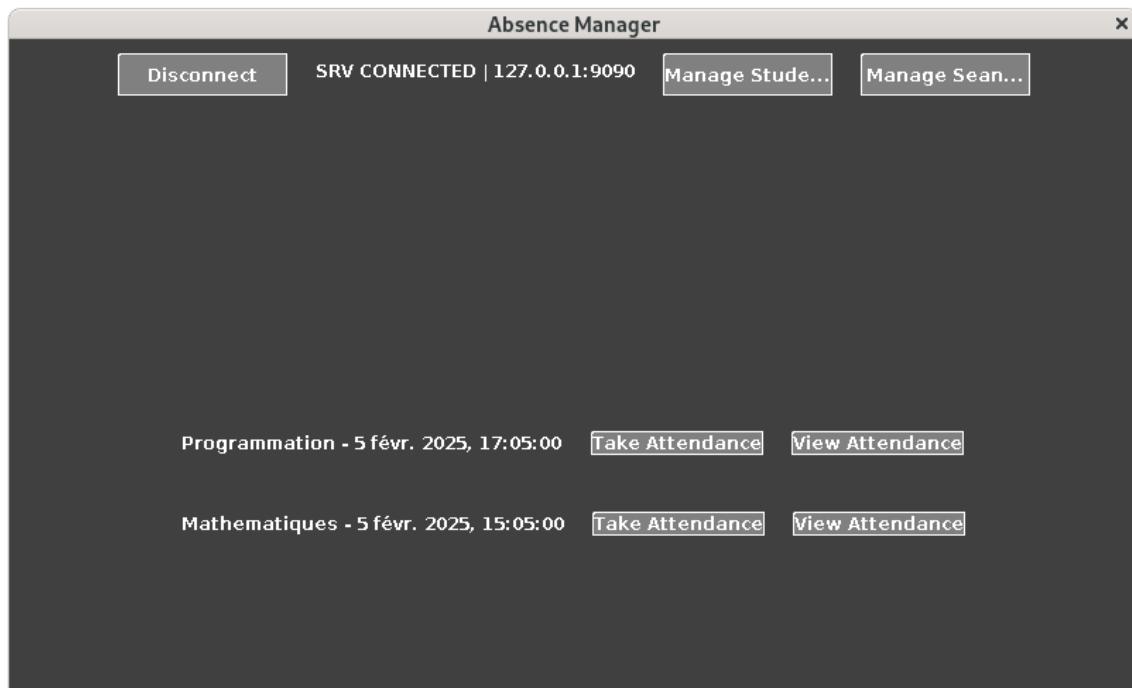
Cette classe est identique à celle précédemment expliquée (ManageSeances) en détail ci-dessus. La différence réside dans le fait que, pour créer un étudiant, seul le champ "nom" sera demandé à l'utilisateur. Ainsi, la logique utilisée reste la même et ne sera donc pas expliquée en détail dans cette section. Veuillez-vous référer aux fonctionnalités de la classe ManageSeances.

Démonstration du client Java en GUI

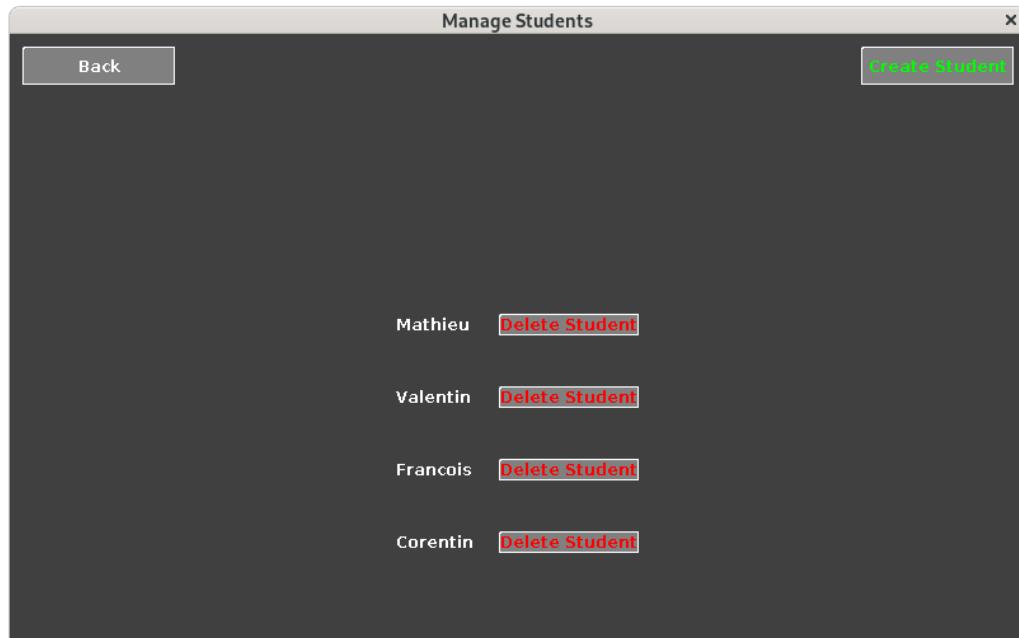
Exécution du script et connexion au serveur



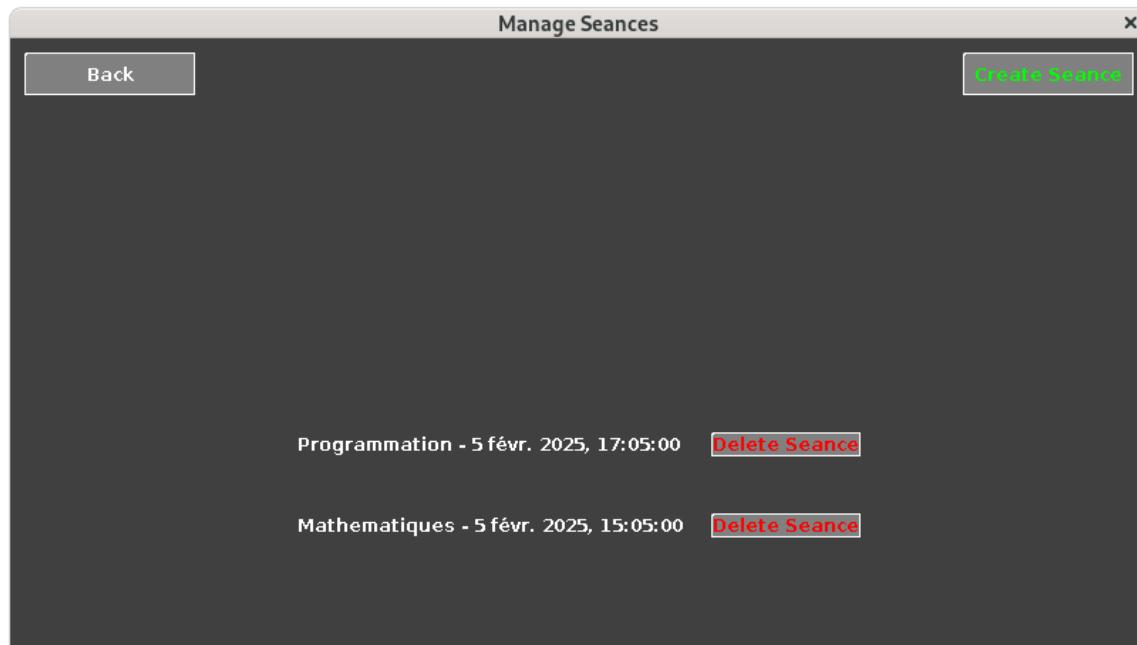
Page d'accueil

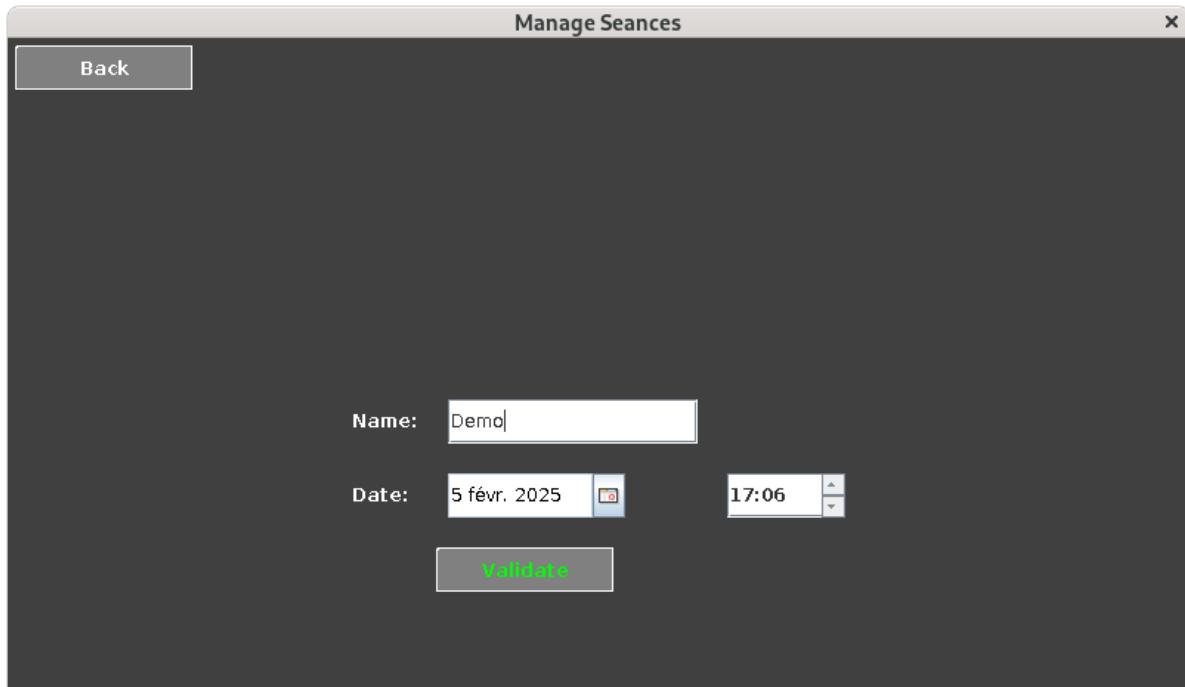


Création/Suppression d'étudiants

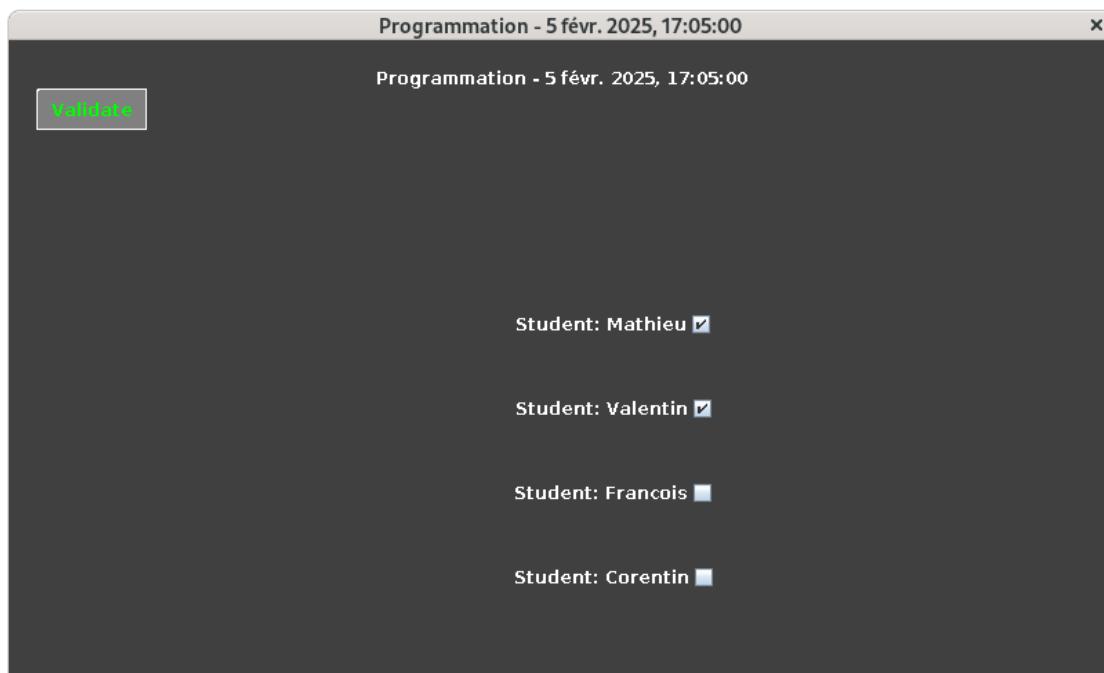


Création/Suppression de séances

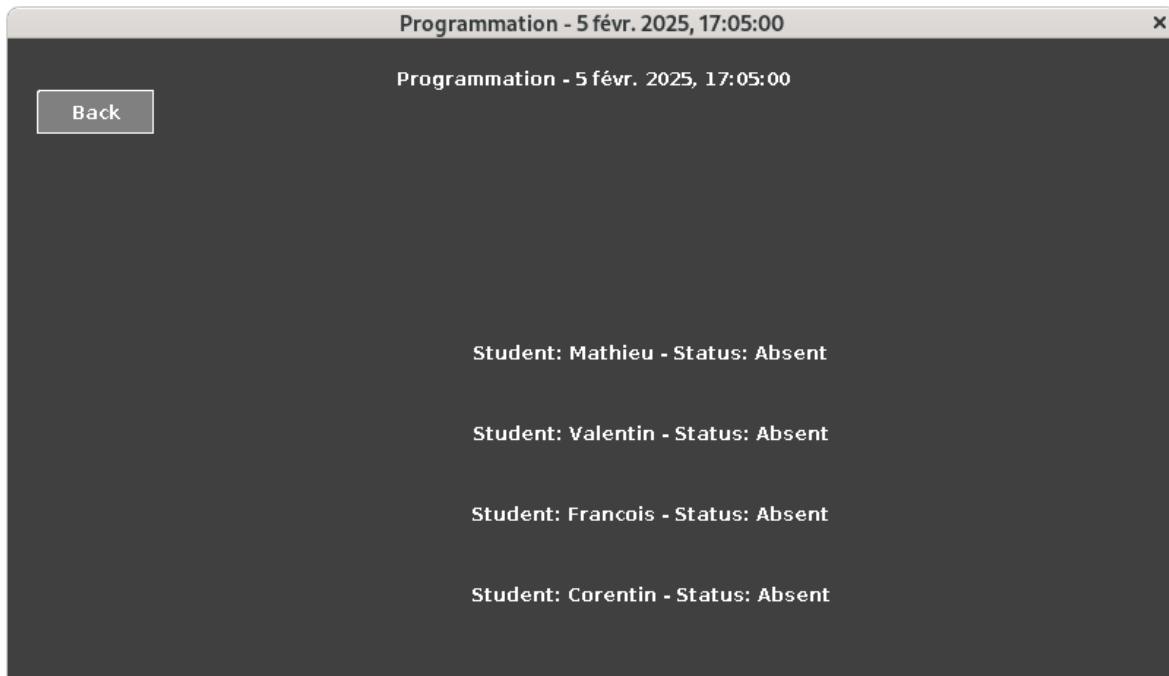




Faire l'appel des étudiants au sein d'une séance



Visualiser l'appel des étudiants au sein d'une séance



6/ Le client Android

Dans le cadre de ce projet, il nous a été demandé de réaliser un client Android. Pour cela, nous avons conçu une application mobile en utilisant Android Studio. Cette partie va donc détailler les différentes étapes de conception et de développement du client Android.

6.1/ Le fichier Manifest:

Le fichier `AndroidManifest.xml` est un fichier essentiel dans une application Android. Il sert à définir les paramètres de base de l'application, notamment ses activités (Activity), ses permissions, son thème et d'autres configurations importantes.

Dans ce fichier, nous déclarons les **activités** (Activity) afin que le système Android puisse les reconnaître et les gérer. Sans déclaration dans le manifest, une activité ne pourra pas être lancée. Ici, nous avons par exemple :

- `MainActivity` qui est l'activité principale avec un intent-filter la définissant comme le point d'entrée de l'application.
- `MenuActivity` et `ManageSeancesActivity` qui sont également déclarées pour être accessibles dans l'application.

Nous avons ajouté **deux permissions** :

`android.permission.INTERNET` : Permet à l'application d'accéder à Internet, nécessaire pour communiquer avec un serveur.

`android.permission.ACCESS_NETWORK_STATE` : Permet de vérifier l'état de la connexion réseau (WiFi, données mobiles) avant d'effectuer une requête.

Ces permissions sont importantes pour assurer la connectivité et éviter les erreurs lorsque l'application tente d'accéder à Internet.

```

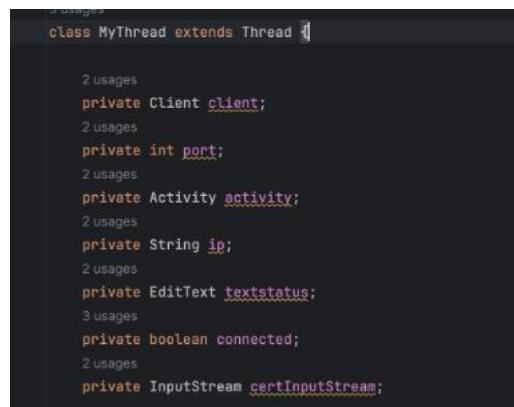
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:tools="http://schemas.android.com/tools">
4
5       
6       <uses-permission android:name="android.permission.INTERNET" />
7       
8       <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
9
10      <application
11          android:allowBackup="true"
12          android:dataExtractionRules="@xml/data_extraction_rules"
13          android:fullBackupContent="@xml/backup_rules"
14          android:icon="@mipmap/ic_launcher"
15          android:label="AbsenceManager"
16          android:roundIcon="@mipmap/ic_launcher_round"
17          android:supportsRtl="true"
18          android:theme="@style/Theme.AbsenceManager"
19          tools:targetApi="31">
20              <activity
21                  android:name=".MainActivity"
22                  android:exported="true">
23                  <intent-filter>
24                      <action android:name="android.intent.action.MAIN" />
25                      <category android:name="android.intent.category.LAUNCHER" />
26                  </intent-filter>
27              </activity>
28              <activity
29                  android:name=".MenuActivity"
30                  android:exported="true">
31              </activity>
32              <activity
33                  android:name=".ManageSeancesActivity"

```

Text Merged Manifest

6.2/ Le fichier MyThread:

La classe MyThread permet de gérer la connexion à un serveur dans un thread séparé. Cela évite de bloquer l'interface utilisateur pendant la tentative de connexion.



La classe contient plusieurs variables privées qui permettent de gérer la connexion et d'interagir avec l'interface utilisateur. Ces variables sont utilisées pour stocker l'instance du client, le port de connexion, l'activité Android associée, l'adresse IP du serveur, un champ d'affichage du statut de connexion, un indicateur de connexion et un flux d'entrée pour le certificat de connexion sécurisée.

```
public MyThread(Activity activity, Client client, String ip, int port, InputStream certInputStream) {
    this.activity = activity;
    this.client = client;
    this.port = port;
    this.ip = ip;
    this.certInputStream = certInputStream;
    // Initialize the EditText
    textStatus = activity.findViewById(R.id.status_connect);
}
```

Le constructeur MyThread est chargé d'initialiser les variables et de récupérer l'élément de l'interface utilisateur permettant d'afficher l'état de la connexion. Ce constructeur prend en paramètres l'activité associée, une instance de Client, l'adresse IP et le port du serveur, ainsi qu'un flux d'entrée pour le certificat.

La méthode run() est exécutée lorsqu'on démarre le thread avec .start(). Elle tente d'établir la connexion au serveur et met à jour l'interface utilisateur en conséquence.

```
@Override
public void run() {
    // Code à exécuter dans le thread
    connected = client.connectToServer(ip, port, certInputStream);
    updateStatusInUI();
}
```

Elle appelle la méthode connectToServer() du Client en lui passant l'IP, le port et le certificat, puis stocke le résultat dans connected avant d'appeler updateStatusInUI() pour mettre à jour l'affichage. Le « Client » est un singleton, ce qui signifie qu'il existe une seule instance de cet objet dans l'application. Lorsqu'il est utilisé ici, il est instancié et partagé à travers l'application, garantissant ainsi une gestion cohérente de la connexion.

```
1 usage
>     private void updateStatusInUI() { activity.runOnUiThread(new UpdateUITask()); }
```

Cette méthode garantit que la mise à jour de l'interface utilisateur est effectuée sur le thread principal d'Android. Android interdit les modifications de l'UI depuis un thread secondaire, c'est pourquoi runOnUiThread() est utilisé pour exécuter UpdateUITask sur le thread principal. La classe interne UpdateUITask implémente Runnable et permet de modifier l'affichage du statut de connexion.

```
private class UpdateUITask implements Runnable {  
    @Override  
>     public void run() { textStatus.setText(connecte ? "Connected" : "Not Connected"); }  
  
    1 usage  
>     public boolean isConnected() { return connecte; }  
}
```

La méthode isConnected() permet d'obtenir l'état actuel de la connexion. Elle retourne la valeur de connected, permettant ainsi à d'autres classes de vérifier si la connexion a réussi ou échoué.

Le thread MyThread est instancié avec les paramètres de connexion nécessaires. Lorsqu'il est démarré avec .start(), il tente de se connecter au serveur en arrière-plan sans bloquer l'interface utilisateur. Une fois la tentative terminée, l'affichage est mis à jour via updateStatusInUI(), qui exécute UpdateUITask pour modifier textStatus. L'utilisateur peut ainsi voir s'il est connecté ou non. Ce mécanisme garantit une expérience fluide et réactive pour l'utilisateur.

6.3/ Le fichier MainActivity:

La classe MainActivity représente l'écran principal de l'application. Son rôle est de permettre à l'utilisateur de saisir une adresse IP et un port pour établir une connexion au serveur. Déclaration des variables La classe contient plusieurs attributs permettant de gérer l'interface utilisateur et la connexion.

```
public class MainActivity extends AppCompatActivity {
```

```
    2 usages  
    private EditText ipInput;  
    2 usages  
    private EditText portInput;  
    2 usages  
    private EditText textStatus;
```

Ces variables stockent les références des champs de saisie de l'IP, du port, ainsi que de la zone d'affichage du statut de connexion.

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Initialize views
    ipInput = findViewById(R.id.ip_input);
    portInput = findViewById(R.id.port_input);
    textStatus = findViewById(R.id.status_connect);
    Button connectButton = findViewById(R.id.connect_button);

    // Set click listener for the Connect button
    connectButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            String ip = ipInput.getText().toString();
            String port = portInput.getText().toString();

            if (validateInput(ip, port)) {

                InputStream certInputStream = getResources().openRawResource(R.raw.cert);
                Client client = Client.getInstance(); // Obtenir l'instance unique du client
                MyThread myThread = new MyThread(activity: MainActivity.this, client, ip, Integer.parseInt(port));
                // Démarrer le thread
                myThread.start();

                try {
                    // Attendre que le thread se termine
                    myThread.join();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    });
}

```

La méthode `onCreate()` est appelée lors de la création de l'activité. Elle initialise l'interface utilisateur et définit le comportement du bouton de connexion.

Elle commence par appeler `setContentView(R.layout.activity_main)` pour définir l'interface utilisateur à partir du fichier XML correspondant. Ensuite, elle initialise les vues en associant les variables aux composants de l'interface

Enfin, elle définit un écouteur d'événements pour le bouton de connexion. Lorsque l'utilisateur clique sur le bouton, les valeurs des champs IP et port sont récupérées et vérifiées.

Vérification des entrées.

```

private boolean validateInput(String ip, String port) {
    return !ip.isEmpty() && !port.isEmpty();
}

```

Avant d'établir la connexion, il est nécessaire de valider les entrées de l'utilisateur. Si les valeurs sont correctes, la tentative de connexion peut commencer.

Création et exécution du thread de connexion

L'application charge un certificat nécessaire à la connexion sécurisée :

```
InputStream certInputStream = getResources().openRawResource(R.raw.cert);
```

Ensuite, elle récupère l'instance unique du Client (singleton) :

```

InputStream certInputStream = getResources().openRawResource(R.raw.cert);
Client client = Client.getInstance(); // Obtenir l'instance unique du client
MyThread myThread = new MyThread( activity: MainActivity.this, client, ip, Integer.parseInt(port), certInputStream);
// Démarrer le thread
myThread.start();

try {
    // Attendre que le thread se termine
    myThread.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}

if (myThread.isConnected()) {
    Log.d( tag: "MainActivity", msg: "Connexion réussie");
    Intent intent = new Intent( packageContext: MainActivity.this, MenuActivity.class);
    intent.putExtra( name: "ip", ip);
    startActivity(intent);
}

```

Puis, elle crée une instance de MyThread pour gérer la connexion dans un thread séparé :

Le thread est ensuite démarré avec : myThread.start(). L'application attend que le thread se termine en appelant join()

Gestion du résultat de la connexion

Si la connexion est établie, l'application affiche un message de succès et lance une nouvelle activité MenuActivity. Si la connexion échoue, un message est affiché dans `textstatus`.

Lors de l'exécution de MainActivity, l'utilisateur peut entrer une adresse IP et un port, puis appuyer sur le bouton de connexion. Après une validation des entrées, un thread MyThread est lancé pour établir la connexion au serveur en arrière-plan. Si la connexion réussit, l'utilisateur est redirigé vers MenuActivity, sinon un message d'erreur est affiché.

6.4/ Le fichier MyThreadCommand :

La classe MyThreadCommand permet d'exécuter différentes commandes liées à la gestion des séances et des étudiants dans un thread séparé. Cela évite de bloquer l'interface utilisateur pendant le traitement des données.

Déclaration des variables

La classe contient plusieurs variables privées et protégées qui permettent de gérer les données et d'interagir avec l'interface utilisateur.

```

28 usages
private final String command;
12 usages
private Client client;
11 usages
protected Activity activity;
4 usages
private EditText textStatus;
3 usages
private ListView listView;
6 usages
private ArrayAdapter<?> adapter;
3 usages
private AttendanceCallback callback;
2 usages
private int position;

4 usages
protected List<Seance> seances;
10 usages
protected List<Etudiant> students;

```

Ces variables stockent la commande à exécuter, l'instance du client (singleton), l'activité associée, un champ d'affichage du statut de connexion, une ListView pour afficher les données, un adaptateur pour la liste et une interface AttendanceCallback utilisée pour récupérer des données spécifiques.

Constructeurs de la classe

Le constructeur permet d'initialiser les éléments de base nécessaires à l'exécution des commandes :

```

public MyThreadCommand(Client client, String command, Activity activity, ListView listView, EditText textStatus) {
    this.client = client;
    this.command = command;
    this.activity = activity;
    this.listView = listView;
    this.textstatus = textStatus;
}

```

Le second constructeur ajoute la gestion d'un callback et d'une position spécifique, utile pour traiter des données précises.

```

public MyThreadCommand(Client client, String command, Activity activity, ListView listView,
                      EditText textStatus, AttendanceCallback callback, int position) {
    this(client, command, activity, listView, textStatus);
    this.callback = callback;
    this.position = position;
}

```

La méthode run()

Cette méthode est exécutée lorsqu'on démarre le thread avec .start(). Elle permet d'exécuter la commande spécifiée lors de l'instanciation de l'objet.

Elle vérifie la valeur de command et effectue l'action correspondante :

- Récupération des étudiants
- Récupération des séances
- Suppression ou création d'une séance
- Suppression ou création d'un étudiant
- Récupération ou mise à jour des présences

Par exemple, pour récupérer les étudiants et mettre à jour la ListView :

```
if ("students".equals(command)) {
    students = client.getStudents();
    updateListView(students, choice: "");
```

Pour récupérer la présence d'un étudiant dans une séance et exécuter un callback :

```
else if (command != null && command.startsWith("getattendancestudent:")) {
    Log.d( tag: "MyThreadGet", msg: "Commande : " + command);
    String[] parts = command.split( regex: ":" );
    if (parts.length == 3) {
        int seanceId = Integer.parseInt(parts[1]);
        int studentId = Integer.parseInt(parts[2]);
        final int attendance = client.getAttendanceStudent(seanceId, studentId);
        Log.d( tag: "MyThreadGet", msg: "Attendance : " + attendance);

        if (callback != null) {
            activity.runOnUiThread(new Runnable() {
                @Override
                public void run() { callback.onAttendanceReceived(attendance, position); }
            });
        }
    }
}
```

Mise à jour de l'interface utilisateur

La méthode `updateListView()` est utilisée pour mettre à jour la liste affichant les étudiants ou les séances en fonction du type de commande reçue. Elle sélectionne un `ArrayAdapter` approprié en fonction du type d'objet récupéré (Etudiant ou Seance) et applique le bon layout.

```

private void updateListView(final List<?> items, String choice) {
    activity.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            if (items != null && !items.isEmpty() && textStatus != null && listView != null) {
                if (items.get(0) instanceof Etudiant && "studentsinseances".equals(choice)) {
                    adapter = new ArrayAdapter<Etudiant>(
                        activity,
                        R.layout.custom_list_take_attendance,
                        R.id.textViewItem,
                        (List<Etudiant>) items
                    );
                } else if (items.get(0) instanceof Etudiant && "viewattendance".equals(choice)) {
                    adapter = new ArrayAdapter<Etudiant>(
                        activity,
                        R.layout.custom_list_item_attendance,
                        R.id.textViewItem,
                        (List<Etudiant>) items
                    );
                } else if (items.get(0) instanceof Etudiant) {
                    adapter = new ArrayAdapter<Etudiant>(
                        activity,
                        R.layout.custom_list_item,
                        R.id.textViewItem,
                        (List<Etudiant>) items
                    );
                }
            }
        }
    });
}

```

Par exemple, si la liste contient des étudiants et que la commande concerne les séances, un layout spécifique est appliqué.

Enfin, l'adaptateur est appliqué à la ListView et l'état de connexion est mis à jour. Si aucune donnée n'est trouvée, l'interface affiche une erreur.

Mise à jour du statut de connexion

La méthode updateStatusInUI() met à jour l'affichage du champ “textstatus” pour indiquer si l'action a réussi ou échoué.

Lors de l'exécution de MyThreadCommand, un thread est lancé pour traiter une commande spécifique. Il interroge le serveur via l'instance Client pour récupérer ou modifier des données concernant les étudiants et les séances. Les résultats sont ensuite affichés dans une ListView et l'utilisateur est informé de l'état de la connexion via un champ texte. Cette gestion asynchrone permet d'éviter les blocages de l'interface utilisateur tout en assurant un affichage fluide et réactif.

6.5/ Les fichiers ManageSeancesActivity et ManageStudentsActivity:

Les classes ManageSeances Activity et ManageStudentsActivity permettent respectivement de gérer les séances et les étudiants de l'application. Elles permettent d'afficher la liste des éléments, d'ajouter de nouveaux items et de supprimer des items existants via une interaction avec le serveur.

Les classes possèdent des attributs permettant de gérer l'interface utilisateur et la connexion avec le serveur.

```
private Client client; private ListView seancesListView; private EditText textStatus;
```

Ces variables stockent l'instance unique de Client (singleton), la ListView permettant d'afficher la liste des éléments et un champ texte pour le statut de connexion.

La classe ManageStudentsActivity possède des attributs similaires mais adaptés à la gestion des étudiants :

```
3 usages
private Client client; // Déclarer client comme attribut
5 usages
private ListView studentsListView; // Déclarer studentsListView comme attribut
3 usages
private EditText textStatus; // Déclarer textStatus comme attribut
```

Méthode onCreate()

Cette méthode est appelée lors de la création de l'activité. Elle initialise les composants graphiques et définit le comportement du bouton permettant d'ajouter un nouvel élément.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_manage_seances);
```

Elle commence par initialiser les variables en associant les composants graphiques aux éléments définis dans le fichier XML de mise en page.

```
client = Client.getInstance(); // Initialiser client
seancesListView = findViewById(R.id.seancesListView); // Initialiser seancesListView
textStatus = new EditText(context: this); // Initialiser textStatus
```

Elle ajoute ensuite un écouteur au bouton permettant de créer une nouvelle séance, elle redirige vers une autre activity qui va se charger de créer les seances. ManageStudent fonctionne sur le même principe

```
Button newSeanceButton = findViewById(R.id.newSeanceButton);

newSeanceButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent_create_seance = new Intent(packageContext: ManageSeancesActivity.this, CreateSeanceActivity.class);
        startActivity(intent_create_seance);
    }
});

// Rafraîchir la liste des séances
refreshSeancesList();
```

De la même manière, ManageStudentsActivity initialise les composants adaptés à la gestion des étudiants et ajoute un écouteur pour le bouton d'ajout d'étudiant.

Rafraîchissement de la liste des éléments

Lorsque l'activité démarre ou reprend, elle doit récupérer et afficher la liste des éléments.

La méthode refreshSeancesList() crée et démarre un thread pour récupérer les séances depuis le serveur.

De la même manière, refreshStudentsList() récupère la liste des étudiants :

```
private void refreshSeancesList() {
    // Créer et démarrer un thread pour récupérer les séances
    MyThreadCommand mythreadcommand = new MyThreadCommand(client, command: "seances", activity: this, seancesListView, tex
    mythreadcommand.start();
}
```

Ces méthodes sont appelées dans onCreate() et onResume() pour toujours afficher des données à jour.

Suppression d'un élément

Lorsqu'un utilisateur appuie sur le bouton de suppression (poubelle), l'application récupère la position de l'élément sélectionné et demande une confirmation.

```
public void trashButtonClick(View view) {
    // Récupérer la ligne parente (l'item de la liste)
    View parentRow = (View) view.getParent();

    // Trouver la position de l'item cliqué dans la ListView
    int position = seancesListView.getPositionForView(parentRow);
```

Si la position est valide, l'élément est récupéré et un AlertDialog s'affiche pour confirmer la suppression.

```
if (position != ListView.INVALID_POSITION) {
    Log.d( tag: "TrashButtonClick", msg: "Position de l'item cliqué : " + position);

    // Récupérer l'adaptateur et l'item à supprimer
    final ArrayAdapter<Seance> adapter = (ArrayAdapter<Seance>) seancesListView.getAdapter();
    final Seance itemToRemove = adapter.getItem(position);

    // Vérifier que l'item à supprimer n'est pas null
    if (itemToRemove != null) {
        // Créer un AlertDialog pour la confirmation
        AlertDialog.Builder alertDialog = new AlertDialog.Builder( context: this);
        alertDialog.setTitle("Confirmation");
        alertDialog.setMessage("Êtes-vous sûr de vouloir supprimer cette séance ?");
    }
}
```

Si l'utilisateur clique sur "Oui", l'élément est supprimé de l'adaptateur et un thread est lancé pour supprimer l'élément côté serveur.

```

if (itemToRemove != null) {
    // Créer un AlertDialog pour la confirmation
    AlertDialog.Builder alertDialog = new AlertDialog.Builder(context);
    alertDialog.setTitle("Confirmation");
    alertDialog.setMessage("Êtes-vous sûr de vouloir supprimer cette séance ?");

    // Créer un OnClickListener pour le bouton "Oui"
    DialogInterface.OnClickListener yesClickListener = new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            // Supprimer l'item de l'adaptateur
            adapter.remove(itemToRemove);
            adapter.notifyDataSetChanged();

            // Récupérer l'ID de la séance à supprimer
            int seanceId = itemToRemove.getIdSeance();
            Log.d(tag, "TrashButtonClick", msg: "ID de la séance à supprimer : " + seanceId);

            // Démarrer un thread pour supprimer la séance côté serveur
            MyThreadCommand myThreadCommand = new MyThreadCommand(client, command: "deleteseance:" + seanceId, activity: myThreadCommand.start());
        }
    };

    // Ajouter un bouton "Oui" pour confirmer la suppression
    alertDialog.setPositiveButton(text: "Oui", yesClickListener);

    // Ajouter un bouton "Non" pour annuler
    alertDialog.setNegativeButton(text: "Non", listener: null);
}

```

La suppression d'un étudiant suit la même logique avec deletetestudent au lieu de deleteseance.

Lors de l'exécution de ManageSeancesActivity et ManageStudentsActivity, l'utilisateur peut voir la liste des séances ou étudiants enregistrés. Il peut ajouter de nouveaux éléments ou supprimer des éléments existants via un bouton de confirmation. Toutes ces opérations sont exécutées en arrière-plan avec MyThreadCommand pour éviter de bloquer l'interface utilisateur. Ainsi, l'application reste fluide et réactive.

6.6/ Les fichiers CreaSeancesActivity et CreateStudentsActivity:

Les classes CreateSeanceActivity et CreateStudentActivity sont des activités Android. Elles sont responsables respectivement de la création de nouvelles séances et de nouveaux étudiants dans l'application. CreateSeanceActivity offre une interface permettant de saisir le nom d'une séance et de sélectionner sa date et son heure, tandis que CreateStudentActivity présente une interface simplifiée pour saisir uniquement le nom d'un nouvel étudiant.

Variables et leurs rôles

```

2 usages
private EditText editTextSeanceName;
2 usages
private Button buttonChooseDateTime;
2 usages
private TextView textViewDateTime;
2 usages
private Button buttonValidate;
13 usages
private Calendar selectedDateTime;

2 usages
private boolean isDateTimeSelected = false; // Indicateur pour vérifier si une date et une heure ont été sélectionnées

```

Pour CreateSeanceActivity, les variables incluent les éléments d'interface utilisateur nécessaires à la création d'une séance. L'activité utilise un EditText pour la saisie du nom, des boutons pour la sélection de la date/heure et la validation, ainsi qu'un Calendar pour stocker la date et l'heure sélectionnées. Un booléen isDateTimeSelected permet de vérifier si l'utilisateur a bien sélectionné une date et une heure. CreateStudentActivity possède une structure plus simple avec seulement un EditText pour la saisie du nom de l'étudiant et un bouton de validation. Cette simplicité reflète la nature plus directe de la création d'un étudiant qui ne nécessite qu'un nom.

```
// Initialisation des vues
editTextSeanceName = findViewById(R.id.editTextSeanceName);
buttonChooseDateTime = findViewById(R.id.buttonChooseDateTime);
textViewDateTime = findViewById(R.id.textViewDateTime);
buttonValidate = findViewById(R.id.buttonValidate);

// Initialisation du calendrier
selectedDateTime = Calendar.getInstance();

// Gestion du clic sur le bouton pour choisir la date et l'heure
buttonChooseDateTime.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { showDatePicker(); }
});
```

Initialisation et configuration de l'interface

La méthode onCreate() de CreateSeanceActivity configure l'interface utilisateur en initialisant les composants graphiques nécessaires à la création d'une séance. Elle met en place les écouteurs d'événements sur les boutons pour gérer la sélection de la date/heure et la validation des données. L'interface est conçue pour guider l'utilisateur étape par étape dans le processus de création d'une séance.

Pour CreateStudentActivity, la méthode onCreate() initialise une interface plus épurée, se concentrant uniquement sur la saisie du nom de l'étudiant et sa validation. Cette approche minimaliste facilite la création rapide de nouveaux étudiants dans le système.

Gestion de la sélection de date et heure

```
private void showDatePicker() {
    // Afficher le DatePickerDialog pour choisir la date
    DatePickerDialog datePickerDialog = new DatePickerDialog(
        context, this,
        new DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker view, int year, int month, int dayOfMonth) {
                selectedDateTime.set(Calendar.YEAR, year);
                selectedDateTime.set(Calendar.MONTH, month);
                selectedDateTime.set(Calendar.DAY_OF_MONTH, dayOfMonth);

                // Après avoir choisi la date, afficher le TimePickerDialog pour choisir l'heure
                showTimePicker();
            }
        },
        selectedDateTime.get(Calendar.YEAR),
        selectedDateTime.get(Calendar.MONTH),
        selectedDateTime.get(Calendar.DAY_OF_MONTH)
    );
    datePickerDialog.show();
}
```

```
private void showTimePicker() {
    // Afficher le TimePickerDialog pour choisir l'heure
    TimePickerDialog timePickerDialog = new TimePickerDialog(
        context: this,
        new TimePickerDialog.OnTimeSetListener() {
            @Override
            public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
                selectedDateTime.set(Calendar.HOUR_OF_DAY, hourOfDay);
                selectedDateTime.set(Calendar.MINUTE, minute);

                // Mettre à jour le TextView avec la date et l'heure sélectionnées
                updateDateTimeTextView();

                // Marquer que l'utilisateur a sélectionné une date et une heure
                isDateTimeSelected = true;
            }
        },
        selectedDateTime.get(Calendar.HOUR_OF_DAY),
        selectedDateTime.get(Calendar.MINUTE),
        is24HourView: true
    );
    timePickerDialog.show();
}
```

CreateSeanceActivity implémente un système de sélection de date et heure en deux étapes via les méthodes `showDateTimePicker()` et `showTimePicker()`. La première étape permet à l'utilisateur de choisir une date dans un calendrier via un `DatePickerDialog`. Une fois la date sélectionnée, un `TimePickerDialog` s'affiche automatiquement pour permettre la sélection de l'heure. Cette approche séquentielle rend le processus de sélection clair et intuitif pour l'utilisateur.

Processus de validation et envoi des données

Pour CreateSeanceActivity, le processus de validation vérifie que le nom de la séance n'est pas vide et qu'une date et une heure ont été sélectionnées. Si ces conditions sont remplies, l'activité convertit la date et l'heure en timestamp Unix et prépare l'envoi des données au serveur avec le format

```
// Créez et démarrez le thread pour envoyer la commande au serveur
MyThreadCommand mythreadcommand = new MyThreadCommand(client, command: "createseance:" + seanceName + ":" + unixTime,
mythreadcommand.start();
```

CreateStudentActivity effectue une validation plus simple en vérifiant uniquement que le champ du nom n'est pas vide. Si le nom est valide, l'activité prépare l'envoi des données au serveur avec le format.

```
// Créez et démarrez le thread pour envoyer la commande au serveur
MyThreadCommand mythreadcommand = new MyThreadCommand(client, command: "createstudent:" + studentName, activity: CreateStud
mythreadcommand.start();
```

Communication avec le serveur et gestion des résultats

Les deux activités utilisent la classe `MyThreadCommand` pour communiquer avec le serveur de manière asynchrone. Cette approche évite de bloquer l'interface utilisateur pendant l'envoi des données. Après l'envoi, les activités attendent la confirmation du serveur. En cas de succès, elles affichent un message de confirmation via un `Toast` et se terminent automatiquement pour retourner à l'écran précédent. En cas d'erreur, elles affichent un message d'erreur approprié pour informer l'utilisateur.

6.7/ Le fichier MenuActivity:

La classe MenuActivity est une activité Android qui sert de menu principal pour l'application de gestion des absences. Elle permet aux utilisateurs de naviguer vers différentes sections de l'application :

- Consultation des présences
- Marquage des présences
- Gestion des étudiants
- Gestion des séances
- Déconnexion

```
public class MenuActivity extends
```

Initialisation dans onCreate()

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_menu);
```

- onCreate() est la méthode principale appelée lors de la création de l'activité.
- setContentView(R.layout.activity_menu); lie l'activité au fichier XML correspondant à l'interface graphique.

Initialisation des éléments de l'interface

```
// Initialisation des boutons  
LinearLayout viewAttendanceButton = findViewById(R.id.view_attendance_button);  
LinearLayout takeAttendanceButton = findViewById(R.id.take_attendance_button);  
LinearLayout manageStudentsButton = findViewById(R.id.manage_students_button);  
LinearLayout manageSeancesButton = findViewById(R.id.manage_seances_button);  
LinearLayout disconnectButton = findViewById(R.id.disconnect_button);
```

- Ces lignes récupèrent les références aux boutons définis dans le fichier XML.
- Chaque bouton représente une action différente dans l'application.

Gestion de la Toolbar

```
28     // Obtenez une référence à la Toolbar depuis le layout  
29     Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  
30     // Définissez la Toolbar comme la barre d'action pour cet Activity  
31     setSupportActionBar(toolbar);
```

On récupère la Toolbar depuis le layout et on la définit comme la barre d'action pour l'activité.

Mise à jour du titre de l'application avec les informations du serveur

```
String appName = "AbsenceManager";
// Ajoutez une information supplémentaire
String newAppName = appName + " | " + client.getServerIP() + ":" + client.getServerPort() + ""
// Définissez le nouveau titre à la Toolbar
getSupportActionBar().setTitle(newAppName);
```

- Client.getInstance() permet de récupérer l'instance du client connectée au serveur.
- Le titre de la barre d'outils est mis à jour pour inclure l'adresse IP et le port du serveur.

Définition des écouteurs d'événements pour les boutons de navigation

Lorsque l'on clique sur un bouton l'application lance un intent de l'activité demandé par exemple :

```
42     viewAttendanceButton.setOnClickListener(new View.OnClickListener() {
43
44     @Override
45     public void onClick(View view) {
46         Intent intent_view = new Intent(getApplicationContext(), ViewAttendanceSeanceActivity.class);
47         startActivity(intent_view);
48     }
});
```

Lorsqu'on clique sur ce bouton, l'application ouvre ViewAttendanceSeanceActivity pour afficher les présences enregistrées.

Tous les autres boutons définis dans ce code fonctionnent de la même façon.

La classe MenuActivity joue un rôle central dans l'application en permettant aux utilisateurs d'accéder aux différentes fonctionnalités via une interface intuitive. Grâce à l'utilisation de Intent, elle facilite la navigation entre les différentes activités tout en maintenant la connexion avec le serveur.

6.8/ Les fichiers ViewAttendanceActivity et SaveAttendanceActivity:

Ces deux classes Android permettent de gérer la présence des étudiants dans une séance.

- SaveAttendanceActivity permet de marquer la présence des étudiants.
- ViewAttendanceActivity permet de consulter l'état des présences.

Les deux classes interagissent avec le serveur via des threads (MyThreadCommand) pour récupérer et envoyer les données.

Classe SaveAttendanceActivity

Déclaration des variables:

```
17     5 usages
18     private Client client;
19     9 usages
20     private ListView studentsListView;
21     5 usages
22     private EditText textStatus;
23     2 usages
24     private Button validateButton;
25     5 usages
26     private int seanceId;
```

- client : Instance du client permettant la communication avec le serveur.
- studentsListView : Liste des étudiants affichée sur l'interface.
- textStatus : Zone de texte pour afficher l'état de la récupération des données.
- validateButton : Bouton pour valider la présence des étudiants.
- seanceId : Identifiant de la séance en cours.

La méthode Oncreate()

```
23     @Override
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.activity_save_attendance);
27
28         client = Client.getInstance();
29         studentsListView = findViewById(R.id.studentsListView);
30         textStatus = new EditText(context: this);
31         validateButton = findViewById(R.id.validateButton);
32
33         seanceId = getIntent().getIntExtra(name: "seance_id", defaultValue: -1);
34         Log.d(tag: "SeanceDetail", msg: "ID de la séance reçue : " + seanceId);
35
36         refreshStudentsList();
37
38         validateButton.setOnClickListener(new ValidateButtonClickListener());
39     }
40 }
```

- Charge l'interface graphique.
- Récupère l'ID de la séance.
- Rafraîchit la liste des étudiants.
- Ajoute un écouteur d'évènements au bouton de validation.

Rafraîchir de la liste des étudiants

```
59     private void refreshStudentsList() {
60         MyThreadCommand myThreadCommand = new StudentListThreadCommand(client, command: "students"
61         myThreadCommand.start();
62     }
```

Lance un thread pour charger la liste des étudiants de la séance.

Gestion de la réponse du serveur

```

    @Override
    public void onAttendanceReceived(int attendance, int position) {
        View itemView = studentsListView.getChildAt(position);
        if (itemView != null) {
            CheckBox checkBox = itemView.findViewById(R.id.checkBoxSelect);
            if (checkBox != null) {
                checkBox.setChecked(attendance == 1);
            }
        }
    }
}

```

Met à jour l'état des checkboxes en fonction de la présence enregistrée.

Validation des présences :

```

95     private class ValidateButtonClickListener implements View.OnClickListener {
96
97     @Override
98     public void onClick(View v) {
99
100         for (int i = 0; i < studentsListView.getCount(); i++) {
101             View itemView = studentsListView.getChildAt(i);
102             if (itemView != null) {
103                 CheckBox checkBox = itemView.findViewById(R.id.checkBoxSelect);
104                 Etudiant etudiant = (Etudiant) studentsListView.getItemAtPosition(i);
105                 int etudiantId = etudiant.getIdEtudiant();
106
107                 if (checkBox.isChecked()) {
108                     String etudiantNom = etudiant.getNomEtudiant();
109                     Log.d( tag: "SelectedStudent", msg: "ID: " + etudiantId + ", Nom: " + etudi
110                     MyThreadCommand mythreadattendance = new MyThreadCommand(client, command:
111                     mythreadattendance.start();
112                 } else {
113                     Log.d( tag: "SelectedStudent", msg: "ID: " + etudiantId);
114                     MyThreadCommand mythreadattendance = new MyThreadCommand(client, command:
115                     mythreadattendance.start();
116                 }
117             }
118             Toast.makeText( context: SaveAttendanceActivity.this, text: "Validation effectuée", Toast
119             finish();

```

- Parcourt la liste des étudiants et enregistre leur présence sur le serveur.
- Affiche un message de confirmation et ferme l'activité.

Classe ViewAttendanceActivity

```

20
21     @Override
22     protected void onCreate(Bundle savedInstanceState) {
23         super.onCreate(savedInstanceState);
24         setContentView(R.layout.activity_view_attendance);
25
26         client = Client.getInstance();
27         studentsListView = findViewById(R.id.studentsListView);
28         textStatus = new EditText( context: this); // Initialiser textStatus
29
30         seanceId = getIntent().getIntExtra( name: "seance_id", defaultValue: -1);
31
32         refreshStudentsList();
}

```

Charge la liste des étudiants avec leur statut de présence.

Rafraîchir La liste des étudiants :

```
52     private void refreshStudentsList() {  
53         StudentListThreadCommand myThreadCommand = new StudentListThreadCommand(client, command: "  
54             myThreadCommand.start();  
55         }  
56     }
```

Lance un thread pour charger les présences.

Mettre à jour l'UI avec les présences

```
40     @Override  
41     public void onAttendanceReceived(int attendance, int position) {  
42         View itemView = studentsListView.getChildAt(position);  
43         if (itemView != null) {  
44             TextView statusText = itemView.findViewById(R.id.textViewStatus);  
45             if (statusText != null) {  
46                 String status = attendance == 1 ? "Présent" : "Absent";  
47                 statusText.setText(status);  
48             }  
49         }  
50     }  
51 }
```

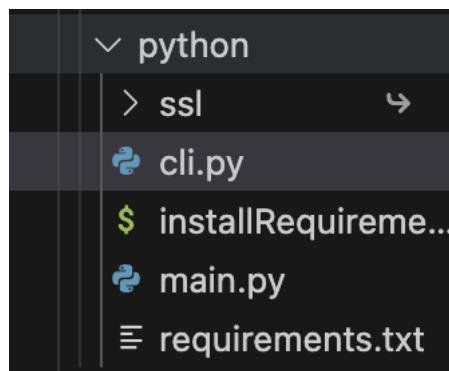
Affiche "Présent" ou "Absent" en fonction des données du serveur.

Ces classes permettent de gérer efficacement la présence des étudiants dans une séance, avec des threads pour assurer la communication avec le serveur sans bloquer l'interface utilisateur.

7/ Le client Python

Par ailleurs, en bonus dans le cadre de ce projet de SAE, nous avons développé un client en Python, en nous appuyant sur les connaissances que nous avons acquises dans ce langage lors des nombreux cours reçus à l'IUT. Le premier cas d'utilisation est basé sur une interface en ligne de commande (CLI - Command Line Interface), ce qui signifie qu'il n'y a pas d'interface graphique et que l'utilisateur interagit avec notre application uniquement via un terminal tel que PowerShell, Bash ou autre. Le second cas d'utilisation que nous avons développé pour améliorer l'expérience utilisateur de notre application est une interface graphique réalisée avec Tkinter. Cette interface graphique permet à l'utilisateur de naviguer à travers des menus, de cliquer sur des boutons pour gérer les sessions et les étudiants, de sélectionner des dates à l'aide d'un calendrier intégré, et d'interagir de manière visuelle et interactive avec l'application et le serveur.

Voici la structure de nos fichiers de codes pour le côté python :



Pour assurer la reproductibilité et la gestion efficace des dépendances de notre projet Python, nous utilisons deux fichiers essentiels : requirements.txt et installRequirements.sh.

Le fichier requirements.txt liste toutes les bibliothèques Python nécessaires au bon fonctionnement de notre application, ainsi que leurs versions spécifiques. Cela permet de garantir que tous les environnements de développement et de production utilisent les mêmes versions des packages, évitant ainsi les conflits et les incompatibilités :

```
altgraph==0.17.4
babel==2.16.0
macholib==1.16.3
...
packaging==24.2
pyinstaller==6.11.1
pyinstaller-hooks-contrib==2025.0
setuptools==75.8.0
tkcalendar==1.6.1
...
```

Les bibliothèques :

- **altgraph** : Utilisé principalement par PyInstaller pour analyser les dépendances des programmes Python.
- **babel** : Fournit des outils de localisation et d'internationalisation, facilitant la gestion des traductions et des formats régionaux.
- **macholib** : Permet l'analyse et la modification des fichiers Mach-O, essentiels pour la création de binaires sur macOS.
- **packaging** : Offre des outils pour la gestion des versions et des dépendances des packages Python.
- **pyinstaller** : Un outil puissant pour convertir des scripts Python en exécutables autonomes.
- **pyinstaller-hooks-contrib** : Fournit des hooks supplémentaires pour PyInstaller, facilitant l'inclusion correcte des dépendances.
- **setuptools** : Une collection de modules facilitant la distribution et l'installation des packages Python.
- **tkcalendar** : Une bibliothèque graphique pour Python offrant des widgets de calendrier, utile pour les interfaces utilisateur.

Le script `install_requirements.sh` automatise le processus de création d'un environnement virtuel (un `venv`) Python et l'installation des dépendances listées dans `requirements.txt` dans cet environnement virtuel.

```
python3 -m venv venv
source venv/bin/activate
python3 -m pip install -r requirements.txt
```

7.2/ Le programme en CLI (cli.py):

Cette partie de notre code implémente une interface en ligne de commande pour interagir avec le serveur de gestion des absences.

Tout d'abord nous importons les dépendances dans notre code ce qui va permettre de créer des connexions réseaux sécurisées :

```
import socket
import ssl
```

- Socket : Pour établir une connexion réseau
- SSL : Utilisation du protocole SSL/TLS pour garantir une communication chiffrée et sécurisée.

Il est essentiel de bien configurer les paramètres de connexion au serveur en définissant précisément l'adresse IP et le port sur lesquels l'application client peut se connecter. Dans notre projet, ces paramètres sont spécifiés dans les fichiers main.py et cli.py, garantissant ainsi que les deux interfaces, graphique et en ligne de commande, communiquent efficacement avec le serveur de gestion des absences.

```
# Define server address and port
SERVER_IP = "localhost"
SERVER_PORT = 8081
CERT_FILE = "ssl/cert.pem" # Path to the server's certificate
```

- **SERVER_IP** : Cette valeur détermine l'emplacement du serveur sur le réseau. Dans notre cas elle sera définie sur localhost car notre application est une application locale (hébergé sur la machine du client).
- **SERVER_PORT**: Le port spécifié détermine le canal de communication utilisé pour établir la connexion entre le client et le serveur. Dans notre projet, le port utilisé est 8081.
- **CERT_FILE** : Pour établir une connexion sécurisée via SSL/TLS, le chemin vers le certificat du serveur doit être correctement spécifié. Dans notre cas notre certificat se trouve dans ce fichier au format PEM ssl/cert.pem.

La Fonction interactive_terminal :

```

def interactive_terminal():
    print(f"Connecting to server at {SERVER_IP}:{SERVER_PORT}...")
    try:
        # Create a socket object
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
            # Wrap the socket with SSL
            context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH)
            context.load_verify_locations(CERT_FILE) # Load the server's certificate
            secure_socket = context.wrap_socket(client_socket, server_hostname=SERVER_IP)

            # Connect to the server
            secure_socket.connect((SERVER_IP, SERVER_PORT))
            print("[INFO] Connected to the server. Type commands or 'exit' to quit.")

        while True:
            # Take input from the user
            user_input = input(">> ").strip()

            # Exit condition
            if user_input.lower() == "exit":
                print("[INFO] Closing connection...")
                break

            # Send the input to the server
            secure_socket.sendall(user_input.encode())

            # Receive and display the server's response
            response = secure_socket.recv(1024).decode()
            print(f"[SERVER RESPONSE]\n{response}\n")

            if "499" in response.lower():
                print("[INFO] Server requested disconnect. Closing connection...")
                break

    except Exception as e:
        print(f"[ERROR] {e}")

```

Cette fonction est responsable de la gestion de la connexion au serveur. Lorsqu'un client tente de se connecter, un message informatif s'affiche, indiquant qu'il se connecte au serveur en précisant l'adresse IP (SERVER_IP) et le port (SERVER_PORT) utilisés. La fonction initialise ensuite un socket TCP/IP en spécifiant l'utilisation du protocole IPv4 (AF_INET). Pour sécuriser cette communication, elle intègre une couche SSL, permettant ainsi d'authentifier le serveur grâce au paramètre SERVER_AUTH. La sécurité des échanges est renforcée par le chargement du certificat SSL depuis le fichier CERT_FILE. Ce processus se déroule en enveloppant le socket standard avec SSL et en spécifiant le nom d'hôte du serveur, ce qui permet de valider le certificat du serveur. Ainsi, une connexion chiffrée et sécurisée est établie de manière fiable entre le client et le serveur, assurant la confidentialité et l'intégrité des données échangées.

Lorsque le client est connecté il reçoit le message [INFO] Connected to the server. Type commands or 'exit' to quit. Sinon en cas d'erreur, il reçoit un message [ERROR] {e}, e étant l'erreur du pourquoi il n'a pas réussi à se connecter. Le client est donc maintenant connecté au terminal interactif, il peut donc saisir du texte :

```

while True:
    # Take input from the user
    user_input = input(">> ").strip()

    # Exit condition
    if user_input.lower() == "exit":
        print("[INFO] Closing connection...")
        break

    # Send the input to the server
    secure_socket.sendall(user_input.encode())

    # Receive and display the server's response
    response = secure_socket.recv(1024).decode()
    print(f"[SERVER RESPONSE]\n{response}\n")

    if "499" in response.lower():
        print("[INFO] Server requested disconnect. Closing connection...")
        break

```

On utilise `input("=>")` pour capturer la saisie de texte dans le terminal, avec `.strip` on enlève les espaces inutiles au début ou fin de la commande saisie. Les commandes sont envoyées au serveur et sont encodées en bytes puis les données sont reçus jusqu'à 1024 bytes de données et retransmis en chaînes de caractère. En cas de réponse 499 le client affiche un message informant que le serveur a demandé la déconnexion et ferme la connexion avec le client.

Dans le cas que l'utilisateur veut arrêter la connexion il a juste à saisir `exit`, ce qui lui affichera un message de fermeture de session et qui va lui faire quitter la boucle entraînant la fermeture de la connexion.

7.3/ Le fichier Main.py:

Main.py implémente une application client en Python qui gère des sessions de présence (par exemple pour des cours ou des réunions) via une interface graphique construite avec Tkinter. L'application communique avec notre backend en utilisant des connexions sécurisées SSL/TLS comme les autres clients :

Pour le fichiers main il a fallu importer différentes bibliothèques :

- **tkinter (alias tk)** : Un module standard de Python permettant de créer des interfaces graphiques (GUI) de manière simple et efficace. En l'important sous l'alias `tk`, on peut aisément accéder à ses widgets et fonctionnalités pour construire l'interface utilisateur.
- **messagebox (depuis tkinter)** : Fournit des boîtes de dialogue standard (alertes, informations, erreurs, etc.) pour interagir avec l'utilisateur, facilitant ainsi la communication d'informations importantes ou d'erreurs dans l'application.
- **ttk (depuis tkinter)** : Propose une collection de widgets thématisés et améliorés par rapport aux widgets classiques de Tkinter. Ces composants offrent une apparence plus moderne et une meilleure intégration visuelle dans les interfaces graphiques.

- **simpdialog (depuis tkinter)** : Offre des dialogues simples pour demander à l'utilisateur des saisies textuelles ou numériques, simplifiant ainsi l'interaction et la collecte de données au sein de l'application.
- **socket** : Un module permettant la communication réseau en Python. Il offre les fonctionnalités nécessaires pour créer des connexions via TCP/IP, facilitant l'échange de données entre applications sur un réseau.
- **ssl** : Fournit des outils pour sécuriser les communications réseau en ajoutant une couche de chiffrement SSL/TLS sur les sockets. Cela garantit que les données échangées restent confidentielles et intègres.
- **datetime (depuis datetime)** : Permet de travailler avec les dates et les heures. Ce module offre des classes pour créer, manipuler et formater des objets représentant des instants précis, ce qui est essentiel pour toute application nécessitant une gestion temporelle.
- **time** : Offre des fonctions pour interagir avec le temps système, telles que la mesure des durées, l'attente (via time.sleep()) et la conversion d'horodatages. Il complète le module datetime pour des opérations liées au temps en général.
- **Calendar (depuis tkcalendar)** : Un widget spécialisé pour afficher et sélectionner des dates de manière conviviale dans une interface graphique. Cette bibliothèque tierce étend les capacités de Tkinter en fournissant des composants dédiés à la gestion du calendrier et des sélections de date.

```
import tkinter as tk
from tkinter import messagebox, ttk, simpledialog
import socket
import ssl
from datetime import datetime
import time
from tkcalendar import Calendar
```

Nous avons créé 3 classes :

- La classe SSL Socket permet d'encapsuler la création et la gestion d'une connexion socket sécurisée (SSL/TLS). On va créer un contexte SSL par défaut avec ssl.create_default_context(). On charge un certificat client et sa clé privée si fournis (certfile et keyfile). On charge des certificats d'autorité (CA) pour vérifier le serveur si nécessaire (cafile). On désactive la vérification du nom d'hôte et la vérification du certificat (paramètres utiles pour le développement ou les tests, mais à ne pas utiliser en production). On crée une socket TCP classique avec socket.socket qui sera ensuite enveloppée en SSL. connect(host, port): Tente de se connecter à l'hôte et au port spécifiés. En cas de succès, enveloppe la socket dans le contexte SSL grâce à wrap_socket(). En cas d'erreur, affiche un message d'erreur via une boîte de dialogue.

`send(data)`: Envoie des données (après les avoir encodées en UTF-8) via le socket SSL.
`recv(buffer_size)`: Réceptionne les données envoyées par le serveur.
`close()`: Ferme la connexion SSL puis le socket sous-jacent.

```
class SSLSocket:  
    def __init__(self, certfile='./ssl/cert.pem', keyfile=None, cafile=None):  
        # Create SSL context  
        self.context = ssl.create_default_context()  
  
        # Optionally load a client certificate and key  
        if certfile and keyfile:  
            self.context.load_cert_chain(certfile=certfile, keyfile=keyfile)  
  
        # Optionally load CA certificates (for server verification)  
        if cafile:  
            self.context.load_verify_locations(cafile=cafile)  
  
        # Don't verify server certificate (for development/testing)  
        self.context.check_hostname = False  
        self.context.verify_mode = ssl.CERT_NONE # Set to CERT_OPTIONAL or CERT_REQUIRED if you want  
  
        # Create socket and wrap with SSL  
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
        self.ssl_sock = None  
  
    def connect(self, host, port):  
        try:  
            self.sock.connect((host, port))  
            self.ssl_sock = self.context.wrap_socket(self.sock, server_hostname=host)  
            return True  
        except Exception as e:  
            messagebox.showerror("Connection Error", f"Failed to establish SSL connection: {str(e)}")  
            return False  
  
    def send(self, data):  
        if self.ssl_sock:  
            return self.ssl_sock.send(data.encode())  
        return 0  
  
    def recv(self, buffer_size):
```

- La class TimeSelector permet à l'utilisateur de sélectionner une date et une heure via une interface graphique. Cette class utilise le widget Calendar (du module tkcalendar) pour la sélection de la date, avec un format de date yyyy-mm-dd. Ajoute deux Spinbox (via ttk.Spinbox) pour sélectionner l'heure (0 à 23) et les minutes (0 à 59). Initialise les valeurs par défaut avec l'heure et la date actuelles.

`get_timestamp()`: Récupère la date sélectionnée dans le calendrier et l'heure/minute des spinbox, construit un objet datetime et retourne le timestamp UNIX correspondant (nombre de secondes depuis le 1er janvier 1970). Si le format est incorrect, renvoie None.

```

class TimeSelector(tk.Frame):
    def __init__(self, master):
        super().__init__(master)

        # Calendar for date selection
        self.calendar = Calendar(self, date_pattern='yyyy-mm-dd')
        self.calendar.pack(fill='x', pady=5)

        # Time selector
        time_frame = tk.Frame(self)
        time_frame.pack(fill='x', pady=5)

        # Hours
        self.hour = ttk.Spinbox(time_frame, from_=0, to=23, width=4)
        self.hour.pack(side=tk.LEFT, padx=2)

        # Minutes
        self.minute = ttk.Spinbox(time_frame, from_=0, to=59, width=4)
        self.minute.pack(side=tk.LEFT, padx=2)

        # Set default time
        now = datetime.now()
        self.hour.set(now.hour)
        self.minute.set(now.minute)
    def get_timestamp(self):
        try:
            selected_date = self.calendar.get_date()
            dt = datetime.strptime(selected_date, '%Y-%m-%d')
            dt = dt.replace(hour=int(self.hour.get()), minute=int(self.minute.get()))
            return int(dt.timestamp())
        except ValueError:
            return None

```

- La class clientApp qui permet de gérer l'interface graphique principale et la logique d'échange avec le serveur.

Initialisation (`__init__`) : Définit le titre et la taille de la fenêtre principale, ce qui configure la grille pour que les widgets s'ajustent correctement. Affiche d'abord une boîte de dialogue de connexion pour saisir l'adresse IP et le port du serveur.

Interface de Connexion Méthode `show_connection_dialog()` : Affiche un formulaire centré demandant à l'utilisateur de saisir l'IP et le port (valeurs par défaut : 127.0.0.1 et 8081). Le bouton « Connect » déclenche la conversion du port en entier et appelle `initialize_connection(ip, port)`.

`initialize_connection(ip, port)` : Instancie la classe `SSLocket` pour établir une connexion sécurisée avec le serveur. Si la connexion échoue, l'application se ferme. Sinon, l'interface principale (la page d'accueil) est créée et le protocole de fermeture de la fenêtre est redéfini via `on_closing()` pour envoyer un message de déconnexion au serveur avant de fermer. Page d'Accueil (Liste des Sessions)

`create_home_page()` : Efface les widgets existants grâce à `clear_window()`. Crée un en-tête avec le titre « Liste des sessions » et des boutons pour créer une nouvelle session et gérer les étudiants. Affiche une table (`ttk.Treeview`) qui liste les sessions existantes. La table se remplit grâce à la méthode `load_table()`, qui envoie une requête au serveur pour récupérer les informations des sessions. Un double-clic sur une ligne de la table appelle `on_table_select()` pour afficher la page d'une session individuelle. Un clic sur l'icône de la corbeille (colonne « actions ») permet de supprimer une session via `delete_seance()`.

`create_seance_window()` : Ouvre une nouvelle fenêtre (dialogue) pour créer une séance. Permet de saisir le nom de la séance et de sélectionner la date et l'heure à l'aide du widget `TimeSelector`. Lorsque l'utilisateur confirme, construit une requête sous la forme "seance/nom_de_seance/timestamp" et l'envoie au serveur via `send_request()`.

`open_manage_students_window()` affiche une interface dédiée pour lister, ajouter et supprimer des étudiants. La liste est affichée dans un `Listbox` et alimentée via `load_students_list()`, qui envoie une requête pour récupérer les étudiants. Un étudiant peut être créé via une boîte de dialogue (`simplidialog.askstring`) et la création se fait en envoyant une requête de type "student/nom_complet". La suppression se fait en récupérant l'index sélectionné dans le `Listbox` et en envoyant une requête "student/student_id".

`show_individual_seance_page(seance_id, seance_name)` affiche une nouvelle vue pour une session donnée. Une table affiche la liste des étudiants et leur présence (présent ou absent). La méthode `load_students(seance_id)` récupère la liste des étudiants et, pour chacun, appelle `get_student_attendance(seance_id, student_id)` afin de déterminer s'il est présent ou non. Un double-clic sur un étudiant dans la table déclenche

`toggle_attendance(seance_id)`, qui inverse l'état de présence en envoyant une requête de mise à jour de type "attendance/seance_id/student_id/nouvel_etat".

`send_request(request)` : Envoie une requête au serveur via l'objet `SSLocket` (méthode `send`). Attend et décode la réponse du serveur. Gère différents codes de réponse : Par exemple, si la réponse indique une déconnexion (499/`DISCONNECTED`) ou la fermeture du serveur (499/`CLOSED`), affiche une information et ferme l'application. Si la réponse commence par 400 ou 500, affiche un message d'erreur. Enlève le préfixe 202/ si présent, et renvoie la réponse nettoyée.

`on_closing()` : Avant de fermer la fenêtre principale, envoie une requête de déconnexion au serveur. Ferme ensuite la connexion SSL et la fenêtre Tkinter.

`clear_window()` : Supprime tous les widgets de la fenêtre principale pour permettre de charger une nouvelle interface (par exemple, lors du changement de vue entre la page d'accueil, la page de session ou la gestion des étudiants).

```

class ClientApp:
    def __init__(self, master):
        self.master = master
        master.title("Attendance Manager")
        master.geometry("500x300")

        master.grid_rowconfigure(1, weight=1)
        master.grid_columnconfigure(0, weight=1)

        self.show_connection_dialog()

    def show_connection_dialog(self):
        # Clear any existing widgets
        for widget in self.master.winfo_children():
            widget.destroy()

        frame = tk.Frame(self.master)
        frame.place(relx=0.5, rely=0.5, anchor="center")

        tk.Label(frame, text="IP Address:").pack(pady=(20,5))
        ip_entry = tk.Entry(frame)
        ip_entry.insert(0, "127.0.0.1")
        ip_entry.pack()

        tk.Label(frame, text="Port:").pack(pady=(10,5))
        port_entry = tk.Entry(frame)
        port_entry.insert(0, "8081")
        port_entry.pack()

    def connect():
        try:
            port = int(port_entry.get())
            if port < 0 or port > 65535:
                raise ValueError
            self.initialize_connection(ip_entry.get(), port)
        except ValueError:
            messagebox.showerror("Error", "Invalid port number")

    tk.Button(frame, text="Connect", command=connect).pack(pady=20)

```

Démonstration du client python en CLI

Tout d'abord, il faut faire un make ce qui va instancier les différents clients et le backend. Pour ce client il n'est pas lié au make seulement le backend.

```
[mathieubersin@MacBook-Pro-de-Mathieu iut-sae-302 % make
mkdir -p ./dist
make[1]: Entering directory '/Users/mathieubersin/Downloads/iut-sae-302'
make[1]: Leaving directory '/Users/mathieubersin/Downloads/iut-sae-302'
make: Nothing to be done for 'dist'.
```

Démonstration du client python en GUI

Tout d'abord comme au début de la CLI il faut faire un make ce qui va instancier les différents clients et le backend. Pour ce client il n'est pas lié au make seulement le backend.

```
[mathieubersin@MacBook-Pro-de-Mathieu iut-sae-302 % make  
mkdir -p ./dist
```

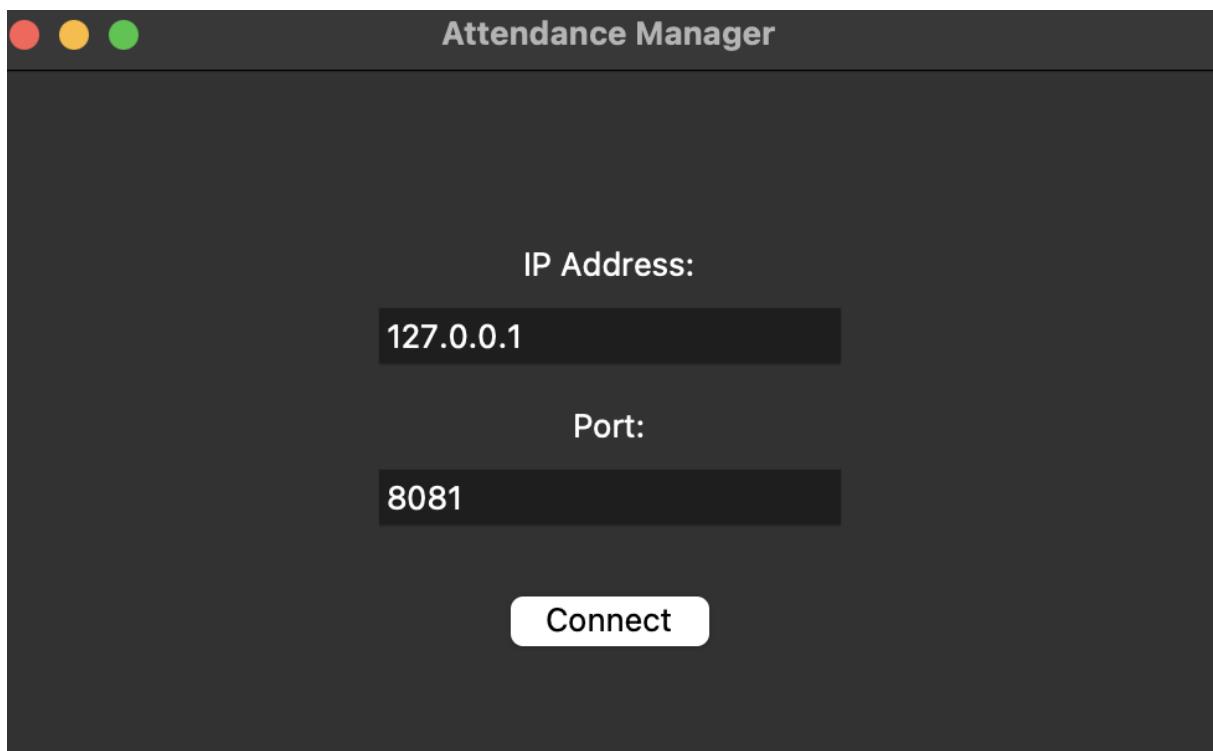
On lance le server en allant dans le répertoire dist est en lancant le server sur le port 8081

```
mathieubersin@MacBook-Pro-de-Mathieu iut-sae-302 % cd dist/server  
mathieubersin@MacBook-Pro-de-Mathieu server % ./server 8081  
[INFO] Server is listening on port 8081...
```

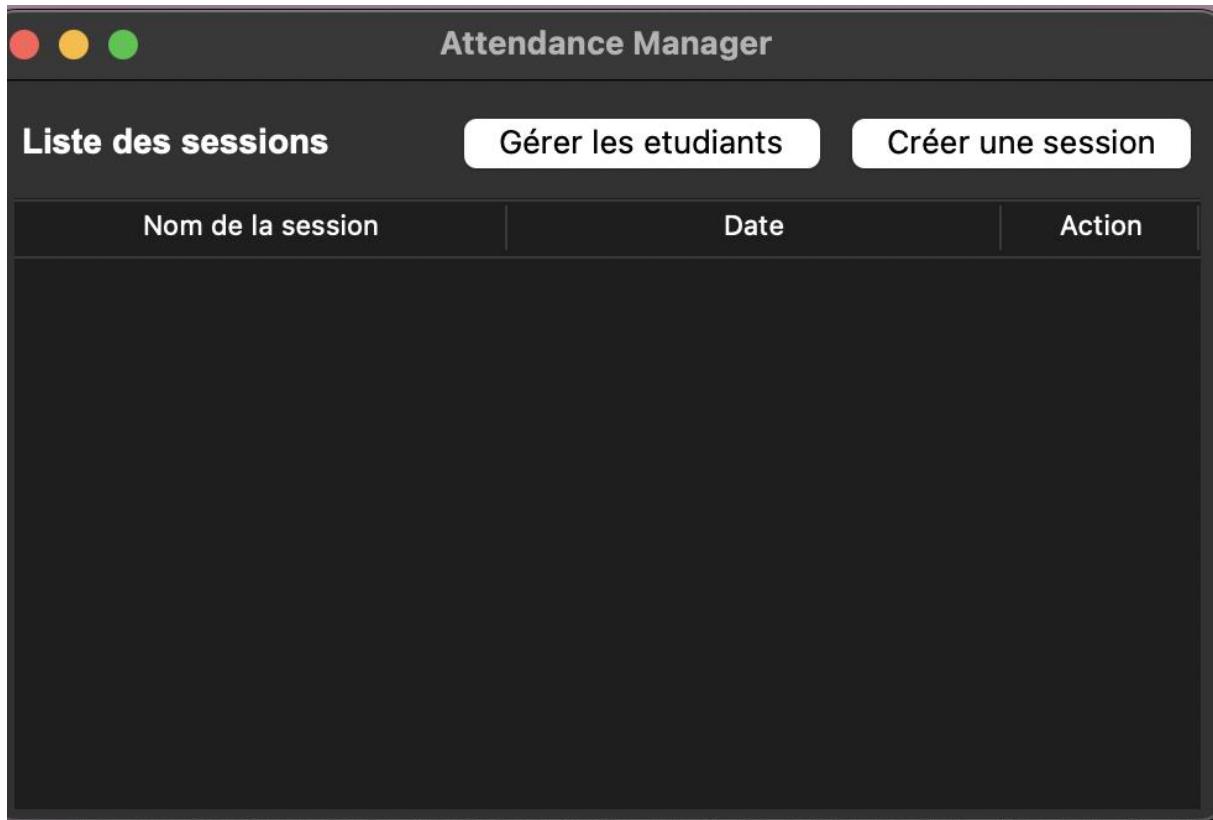
On va dans un le dossier clients / python et on lance main.py

```
mathieubersin@MacBook-Pro-de-Mathieu iut-sae-302 % cd clients/python  
mathieubersin@MacBook-Pro-de-Mathieu python % main.py  
zsh: command not found: main.py  
mathieubersin@MacBook-Pro-de-Mathieu python % python main.py
```

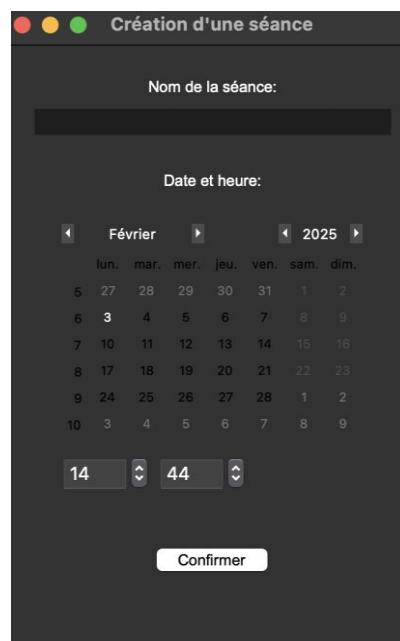
Une fenêtre python s'ouvre et on choisit l'ip et le port sur lequel ce connecter.



Après s'être connecté une nouvelle fenêtre apparaît, depuis cette fenêtre on a deux boutons Gérer les étudiants et créer une session. Et il y'a la listes des sessions affichées.



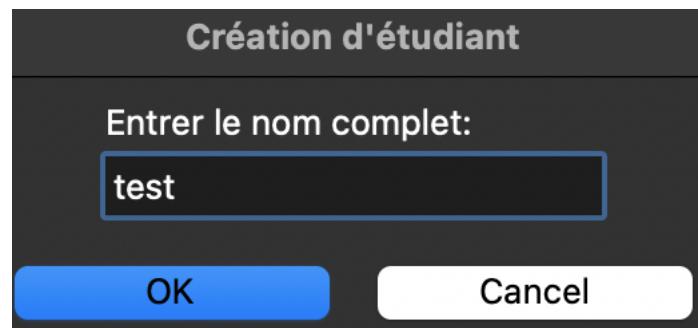
Quand on clic sur le bouton créer une session, on peut donc créer la session en y attribuant un nom la date et l'heure.



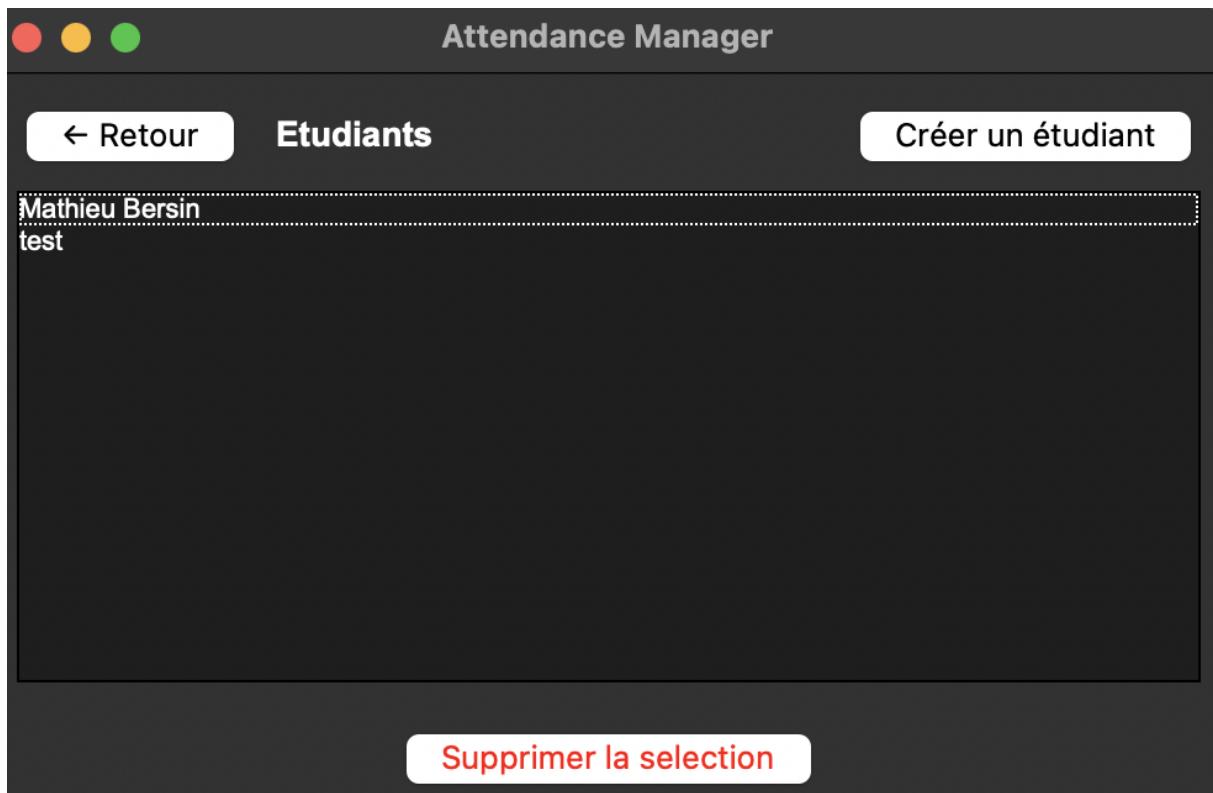
Liste des sessions		Gérer les etudiants	Créer une session
Nom de la session		Date	
Mathématiques	2025-01-28 14:44		

On peut voir ci-dessus que la séance est bien créée. Maintenant on va cliquer sur le bouton gérer les étudiants qui nous amène sur une nouvelle vue où il'y a un bouton retour pour retourner sur l'autre vue, la liste des étudiants (vide pour l'instant), un bouton créer un étudiant et un bouton supprimer la sélection :

← Retour	Etudiants	Créer un étudiant
Supprimer la selection		



Quand on appuie sur le bouton une petite fenêtre se montre et on doit écrire le nom de l'utilisateur,
On crée un utilisateur test.



On supprime l'étudiant test



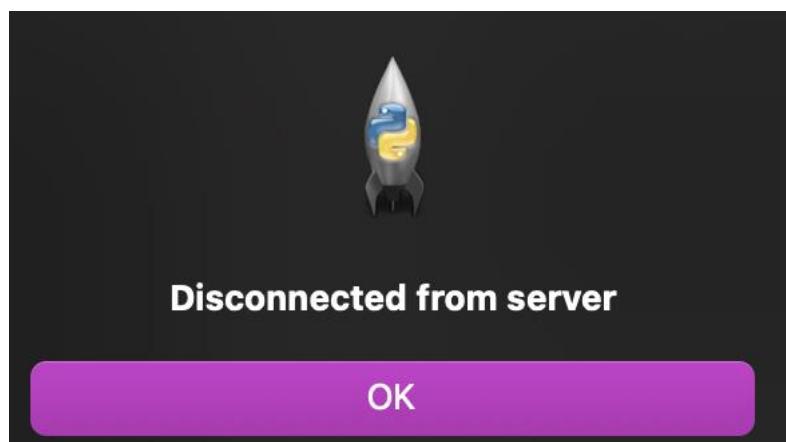
On appuie sur retour puis on double clic sur Mathématiques pour voir les Présence.

Mathématiques	
Nom	Présence
Mathieu Bersin	<input type="radio"/> Absent

On double clic sur absent pour le passer présent.

Mathématiques	
Nom	Présence
Mathieu Bersin	<input checked="" type="radio"/> Présent

On clique sur retour et sur la croix rouge pour bien fermer la communication entre le server et le client python.



Conclusion

Ce projet de développement SAE-302 de l'application de gestion des absences a représenté un défi technique et multidisciplinaire majeur. Il a nécessité la création de plusieurs clients dans des langages et environnements différents — un client en C en CLI, un autre en Python en CLI et GUI, ainsi que deux clients java en CLI et GUI — L'utilisation des sockets pour la communication réseau, a permis d'assurer une transmission fiable et sécurisée des données entre les différents composants de l'application.