TaskManager

Generated by Doxygen 1.10.0

Chapter 1

Task Manager

Task Manager to aplikacja konsolowa stworzona przez Adama Szczotka, mająca na celu zarządzanie ticketami (zadaniami). Projekt został zrealizowany na zaliczenie pierwszego semestru przedmiotu programowanie na Wyższej Szkole Technologii Informatycznych w Katowicach.

1.1 Wprowadzenie

Task Manager to prosta, ale funkcjonalna aplikacja konsolowa umożliwiająca zarządzanie ticketami. Użytkownik może dodawać, aktualizować, usuwać oraz wyszukiwać tickety według określonych kryteriów. Aplikacja również umożliwia eksportowanie ticketów do plików tekstowych.

1.2 Funkcje

Aplikacja oferuje następujące funkcje:

- Dodawanie, aktualizowanie, usuwanie i wyświetlanie ticketów.
- Eksportowanie ticketów do plików tekstowych.
- Wyszukiwanie ticketów według priorytetu lub statusu.

1.3 Technologie

Projekt został wykonany w języku C++ z wykorzystaniem standardowej biblioteki C++.

1.4 Uruchomienie

Aby uruchomić aplikację, wykonaj następujące kroki:

1. Sklonuj repozytorium na swoje urządzenie.
2. Otwórz wiersz poleceń lub terminal w folderze projektu.
3. Skompiluj projekt (makefile) za pomocą kompilatora C++ (np. `g++`).
4. Uruchom skompilowany program.

1.5 Struktura Projektu

Projekt składa się z następujących głównych folderów:

- `src/`: Folder zawierający pliki źródłowe projektu.
- `include/`: Folder z plikami nagłówkowymi.
- `data/`: Folder na pliki danych, np. tickety.
- `bin/`: Folder na skompilowaną juz aplikację.

1.6 Autor

Adam Szczotka - student Wyższej Szkoły Technologii Informatycznych w Katowicach.

1.7 Podziękowania

Podziękowania dla wykładowców i kolegów z Wyższej Szkoły Technologii Informatycznych w Katowicach za wsparcie i pomoc w realizacji projektu.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 FileHandler Class Reference

Klasa do obsługi plików.

`#include <FileHandler.h>`

Static Public Member Functions

- static void createFile (const std::string &filename)

  Tworzy nowy plik, jeśli jeszcze nie istnieje.
- static std::vector< std::string > readFile (const std::string &filename)

  Czyta zawartość pliku i zwraca ją jako wektor ciągów znaków.
- static void writeFile (const std::string &filename, const std::vector< std::string > &data)

  Zapisuje dane do pliku.
- static void exportTicketsToTxt (const std::string &sourceFilename, const std::string &targetFilename)

  Eksportuje dane ticketów do pliku tekstowego.

4.1.1 Detailed Description

Klasa do obsługi plików.

FileHandler to klasa pomocnicza służąca do zarządzania operacjami na plikach. Oferuje funkcje do tworzenia, czytania, zapisywania i eksportowania danych do plików.

Definition at line 11 of file FileHandler.h.

4.1.2 Member Function Documentation

4.1.2.1 createFile()

```
void FileHandler::createFile (
            const std::string & filename )  [static]
```

Tworzy nowy plik, jeśli jeszcze nie istnieje.

Parameters

filename	Nazwa pliku do utworzenia.

Definition at line 9 of file FileHandler.cpp.

4.1.2.2 exportTicketsToTxt()

```
void FileHandler::exportTicketsToTxt (
            const std::string & sourceFilename,
            const std::string & targetFilename )  [static]
```

Eksportuje dane ticketów do pliku tekstowego.

Parameters

sourceFilename	Nazwa pliku źródłowego z danymi ticketów.
targetFilename	Nazwa pliku docelowego, do którego dane mają być eksportowane.

Definition at line 50 of file FileHandler.cpp.

4.1.2.3 readFile()

```
std::vector< std::string > FileHandler::readFile (
            const std::string & filename )  [static]
```

Czyta zawartość pliku i zwraca ją jako wektor ciągów znaków.

Parameters

filename	Nazwa pliku do odczytu.

Returns

std::vector<std::string> Wektor zawierający linie tekstu z pliku.

Definition at line 17 of file FileHandler.cpp.

4.1.2.4 writeFile()

```
void FileHandler::writeFile (
            const std::string & filename,
            const std::vector< std::string > & data )  [static]
```

Zapisuje dane do pliku.

Parameters

filename	Nazwa pliku, do którego dane mają być zapisane.
data	Wektor ciągów znaków do zapisania w pliku.

Definition at line 35 of file FileHandler.cpp.

The documentation for this class was generated from the following files:

- include/FileHandler.h
- src/FileHandler.cpp

4.2 Ticket Class Reference

Klasa reprezentująca pojedynczy ticket.

`#include <Ticket.h>`

Public Member Functions

- Ticket ()=default

  Konstruktor domyślny.
- Ticket (int id, std::string title, std::string description, std::string status, int priority)

  Konstruktor inicjalizujący ticket z podanymi wartościami.
- int getID () const

  Pobiera ID ticketu.
- std::string getTitle () const

  Pobiera tytuł ticketu.
- std::string getDescription () const

  Pobiera opis ticketu.
- time_t getCreationDate () const

  Pobiera datę utworzenia ticketu.
- std::string getStatus () const

  Pobiera status ticketu.
- int getPriority () const

  Pobiera priorytet ticketu.
- void setID (int newID)

  Ustawia ID ticketu.
- void setTitle (const std::string &newTitle)

  Ustawia tytuł ticketu.
- void setDescription (const std::string &newDescription)

  Ustawia opis ticketu.
- void setCreationDate (time_t newCreationDate)

  Ustawia datę utworzenia ticketu.
- void setStatus (const std::string &newStatus)

  Ustawia status ticketu.
- void setPriority (int newPriority)

  Ustawia priorytet ticketu.

4.2.1 Detailed Description

Klasa reprezentująca pojedynczy ticket.

Klasa Ticket przechowuje informacje o pojedynczym tickecie, takie jak ID, tytuł, opis, data utworzenia, status i priorytet.

Definition at line 11 of file Ticket.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Ticket() [1/2]

```
Ticket::Ticket ( )  [default]
```

Konstruktor domyślny.

4.2.2.2 Ticket() [2/2]

```
Ticket::Ticket (
            int id,
            std::string title,
            std::string description,
            std::string status,
            int priority )
```

Konstruktor inicjalizujący ticket z podanymi wartościami.

Parameters

id	Numer identyfikacyjny ticketu.
title	Tytuł ticketu.
description	Opis ticketu.
status	Status ticketu.
priority	Priorytet ticketu.

Definition at line 4 of file Ticket.cpp.

4.2.3 Member Function Documentation

4.2.3.1 getCreationDate()

```
time_t Ticket::getCreationDate ( ) const
```

Pobiera datę utworzenia ticketu.

Returns

time_t Data utworzenia ticketu.

Definition at line 11 of file Ticket.cpp.

4.2.3.2 getDescription()

```
std::string Ticket::getDescription ( ) const
```

Pobiera opis ticketu.

Returns

std::string Opis ticketu.

Definition at line 10 of file Ticket.cpp.

4.2.3.3 getID()

```
int Ticket::getID ( ) const
```

Pobiera ID ticketu.

Returns

int ID ticketu.

Definition at line 8 of file Ticket.cpp.

4.2.3.4 getPriority()

```
int Ticket::getPriority ( ) const
```

Pobiera priorytet ticketu.

Returns

int Priorytet ticketu.

Definition at line 13 of file Ticket.cpp.

4.2.3.5 getStatus()

```
std::string Ticket::getStatus ( ) const
```

Pobiera status ticketu.

Returns

std::string Status ticketu.

Definition at line 12 of file Ticket.cpp.

4.2.3.6 getTitle()

```
std::string Ticket::getTitle ( ) const
```

Pobiera tytuł ticketu.

Returns

std::string Tytuł ticketu.

Definition at line 9 of file Ticket.cpp.

4.2.3.7 setCreationDate()

```
void Ticket::setCreationDate (
            time_t newCreationDate )
```

Ustawia datę utworzenia ticketu.

Parameters

newCreationDate	Nowa data utworzenia ticketu.

Definition at line 19 of file Ticket.cpp.

4.2.3.8 setDescription()

```
void Ticket::setDescription (
            const std::string & newDescription )
```

Ustawia opis ticketu.

Parameters

newDescription	Nowy opis ticketu.

Definition at line 18 of file Ticket.cpp.

4.2.3.9 setID()

```
void Ticket::setID (
            int newID )
```

Ustawia ID ticketu.

Parameters

newID	Nowe ID ticketu.

Definition at line 16 of file Ticket.cpp.

4.2.3.10 setPriority()

```
void Ticket::setPriority (
            int newPriority )
```

Ustawia priorytet ticketu.

Parameters

newPriority	Nowy priorytet ticketu.

Definition at line 21 of file Ticket.cpp.

4.2.3.11 setStatus()

```
void Ticket::setStatus (
            const std::string & newStatus )
```

Ustawia status ticketu.

Parameters

newStatus	Nowy status ticketu.

Definition at line 20 of file Ticket.cpp.

4.2.3.12 setTitle()

```
void Ticket::setTitle (
            const std::string & newTitle )
```

Ustawia tytuł ticketu.

Parameters

newTitle	Nowy tytuł ticketu.

Definition at line 17 of file Ticket.cpp.

The documentation for this class was generated from the following files:

- include/Ticket.h
- src/Ticket.cpp

4.3 TicketManager Class Reference

Klasa zarządzająca ticketami.

```
#include <TicketManager.h>
```

Public Member Functions

- void addTicket (Ticket &ticket)

  Dodaje nowy ticket do kolekcji.
- void loadTicketsFromFile (const std::string &filename)

  Wczytuje tickety z pliku.
- void saveTicketsToFile (const std::string &filename)

  Zapisuje wszystkie tickety do pliku.
- void viewTickets () const

  Wyświetla informacje o wszystkich ticketach.

- void readStringFromFile (std::ifstream &file, std::string &str)

  Czyta string z pliku.
- int getLastTicketId ()

  Pobiera ID ostatniego ticketu w kolekcji.
- void updateTicket (int id, const std::string &newTitle, const std::string &newDescription, const std::string &newStatus, int newPriority)

  Aktualizuje ticket o podanym ID.
- void removeTicket (int id)

  Usuwa ticket o podanym ID.
- void searchTicketsByPriority (int priority) const

  Wyszukuje tickety według priorytetu.
- void searchTicketsByStatus (const std::string &status) const

  Wyszukuje tickety według statusu.

4.3.1 Detailed Description

Klasa zarządzająca ticketami.

TicketManager odpowiada za zarządzanie kolekcją ticketów, w tym dodawanie, aktualizowanie, usuwanie, wyszukiwanie oraz zapisywanie i wczytywanie ticketów do/z pliku.

Definition at line 12 of file TicketManager.h.

4.3.2 Member Function Documentation

4.3.2.1 addTicket()

```
void TicketManager::addTicket (
            Ticket & ticket )
```

Dodaje nowy ticket do kolekcji.

Parameters

ticket	Referencja do obiektu Ticket, który ma zostać dodany.

Definition at line 9 of file TicketManager.cpp.

4.3.2.2 getLastTicketId()

```
int TicketManager::getLastTicketId ( )
```

Pobiera ID ostatniego ticketu w kolekcji.

Returns

int ID ostatniego ticketu.

Definition at line 129 of file TicketManager.cpp.

4.3.2.3 loadTicketsFromFile()

```
void TicketManager::loadTicketsFromFile (
            const std::string & filename )
```

Wczytuje tickety z pliku.

Parameters

filename	Nazwa pliku, z którego mają zostać wczytane tickety.

Definition at line 30 of file TicketManager.cpp.

4.3.2.4 readStringFromFile()

```
void TicketManager::readStringFromFile (
            std::ifstream & file,
            std::string & str )
```

Czyta string z pliku.

Parameters

file	Referencja do strumienia pliku.
str	Referencja do stringa, do którego zostanie wczytana zawartość.

Definition at line 23 of file TicketManager.cpp.

4.3.2.5 removeTicket()

```
void TicketManager::removeTicket (
            int id )
```

Usuwa ticket o podanym ID.

Parameters

id	ID ticketu do usunięcia.

Definition at line 153 of file TicketManager.cpp.

4.3.2.6 saveTicketsToFile()

```
void TicketManager::saveTicketsToFile (
            const std::string & filename )
```

Zapisuje wszystkie tickety do pliku.

Parameters

filename	Nazwa pliku, do którego mają zostać zapisane tickety.

Definition at line 81 of file TicketManager.cpp.

4.3.2.7 searchTicketsByPriority()

```
void TicketManager::searchTicketsByPriority (
            int priority ) const
```

Wyszukuje tickety według priorytetu.

Parameters

priority	Priorytet ticketów do wyszukania.

Definition at line 166 of file TicketManager.cpp.

4.3.2.8 searchTicketsByStatus()

```
void TicketManager::searchTicketsByStatus (
            const std::string & status ) const
```

Wyszukuje tickety według statusu.

Parameters

status	Status ticketów do wyszukania.

Definition at line 192 of file TicketManager.cpp.

4.3.2.9 updateTicket()

```
void TicketManager::updateTicket (
            int id,
            const std::string & newTitle,
            const std::string & newDescription,
            const std::string & newStatus,
            int newPriority )
```

Aktualizuje ticket o podanym ID.

Parameters

id	ID ticketu do aktualizacji.
newTitle	Nowy tytuł ticketu.
newDescription	Nowy opis ticketu.
newStatus	Nowy status ticketu.
newPriority	Nowy priorytet ticketu.

Definition at line 139 of file TicketManager.cpp.

4.3.2.10 viewTickets()

```
void TicketManager::viewTickets ( ) const
```

Wyświetla informacje o wszystkich ticketach.

Definition at line 107 of file TicketManager.cpp.

The documentation for this class was generated from the following files:

- include/TicketManager.h
- src/TicketManager.cpp

4.4 UserManager Class Reference

Klasa zarządzająca użytkownikami i ich uprawnieniami.

```
#include <UserManager.h>
```

Public Member Functions

- void addUser (const std::string &username, const std::string &permission)

  Dodaje nowego użytkownika z określonymi uprawnieniami.
- void removeUser (const std::string &username)

  Usuwa użytkownika z systemu.
- bool checkPermission (const std::string &username, const std::string &permission) const

  Sprawdza, czy użytkownik posiada określone uprawnienia.

4.4.1 Detailed Description

Klasa zarządzająca użytkownikami i ich uprawnieniami.

UserManager odpowiada za zarządzanie użytkownikami i ich uprawnieniami w kontekście aplikacji. Pozwala na dodawanie, usuwanie użytkowników oraz sprawdzanie ich uprawnień.

Definition at line 12 of file UserManager.h.

4.4.2 Member Function Documentation

4.4.2.1 addUser()

```
void UserManager::addUser (
            const std::string & username,
            const std::string & permission )
```

Dodaje nowego użytkownika z określonymi uprawnieniami.

Parameters

username	Nazwa użytkownika.
permission	Uprawnienia przypisane użytkownikowi.

Definition at line 3 of file UserManager.cpp.

4.4.2.2 checkPermission()

```
bool UserManager::checkPermission (
            const std::string & username,
            const std::string & permission ) const
```

Sprawdza, czy użytkownik posiada określone uprawnienia.

Parameters

username	Nazwa użytkownika.
permission	Uprawnienia do sprawdzenia.

Returns

true Jeśli użytkownik posiada uprawnienia.

false Jeśli użytkownik nie posiada uprawnienia.

Definition at line 11 of file UserManager.cpp.

4.4.2.3 removeUser()

```
void UserManager::removeUser (
            const std::string & username )
```

Usuwa użytkownika z systemu.

Parameters

username	Nazwa użytkownika do usunięcia.

Definition at line 7 of file UserManager.cpp.

The documentation for this class was generated from the following files:

- include/UserManager.h
- src/UserManager.cpp

Chapter 5

File Documentation

5.1 include/FileHandler.h File Reference

```
#include <string>
#include <vector>
```

Classes

- class FileHandler

  Klasa do obsługi plików.

5.2 FileHandler.h

Go to the documentation of this file.

```
00001
00007 #pragma once
00008 #include <string>
00009 #include <vector>
00010
00011 class FileHandler {
00012 public:
00018     static void createFile(const std::string& filename);
00019
00026     static std::vector<std::string> readFile(const std::string& filename);
00027
00034     static void writeFile(const std::string& filename, const std::vector<std::string>& data);
00035
00042     static void exportTicketsToTxt(const std::string& sourceFilename, const std::string&
      targetFilename);
00043 };
00044
```

5.3 include/Ticket.h File Reference

```
#include <string>
#include <ctime>
```

Classes

- class Ticket

  Klasa reprezentująca pojedynczy ticket.

5.4 Ticket.h

Go to the documentation of this file.

```
00001
00007 #pragma once
00008 #include <string>
00009 #include <ctime>
00010
00011 class Ticket {
00012 private:
00013     int ID;
00014     std::string title;
00015     std::string description;
00016     time_t creationDate;
00017     std::string status;
00018     int priority;
00019
00020 public:
00024     Ticket() = default;
00025
00035     Ticket(int id, std::string title, std::string description, std::string status, int priority);
00036
00037     // Gettery
00038
00044     int getID() const;
00045
00051     std::string getTitle() const;
00052
00058     std::string getDescription() const;
00059
00065     time_t getCreationDate() const;
00066
00072     std::string getStatus() const;
00073
00079     int getPriority() const;
00080
00081     // Settery
00082
00088     void setID(int newID);
00089
00095     void setTitle(const std::string& newTitle);
00096
00102     void setDescription(const std::string& newDescription);
00103
00109     void setCreationDate(time_t newCreationDate);
00110
00116     void setStatus(const std::string& newStatus);
00117
00123     void setPriority(int newPriority);
00124 };
00125
```

5.5 include/TicketManager.h File Reference

```
#include <vector>
#include <fstream>
#include "Ticket.h"
```

Classes

- class TicketManager

  Klasa zarządzająca ticketami.

5.6 TicketManager.h

Go to the documentation of this file.

```
00001
00007 #pragma once
00008 #include <vector>
00009 #include <fstream>
00010 #include "Ticket.h"
00011
00012 class TicketManager {
00013 private:
00014     std::vector<Ticket> tickets;
00015
00016 public:
00022     void addTicket(Ticket& ticket);
00023
00029     void loadTicketsFromFile(const std::string& filename);
00030
00036     void saveTicketsToFile(const std::string& filename);
00037
00041     void viewTickets() const;
00042
00049     void readStringFromFile(std::ifstream& file, std::string& str);
00050
00056     int getLastTicketId();
00057
00067     void updateTicket(int id, const std::string& newTitle, const std::string& newDescription, const
      std::string& newStatus, int newPriority);
00068
00074     void removeTicket(int id);
00075
00081     void searchTicketsByPriority(int priority) const;
00082
00088     void searchTicketsByStatus(const std::string& status) const;
00089
00090     // Pozostałe metody...
00091 };
00092
```

5.7 include/UserManager.h File Reference

```
#include <string>
#include <unordered_map>
```

Classes

- class UserManager

  Klasa zarządzająca użytkownikami i ich uprawnieniami.

5.8 UserManager.h

Go to the documentation of this file.

```
00001
00008 #pragma once
00009 #include <string>
00010 #include <unordered_map>
00011
00012 class UserManager {
00013 private:
00019     std::unordered_map<std::string, std::string> userPermissions;
00020
00021 public:
00028     void addUser(const std::string& username, const std::string& permission);
00029
00035     void removeUser(const std::string& username);
00036
00045     bool checkPermission(const std::string& username, const std::string& permission) const;
00046
00047     // Dodatkowe metody zarządzania użytkownikami...
00048     // Klasa posłuży do rozbudowy programu na 2 semestrze
00049 };
```

5.9 README.md File Reference

5.10 src/FileHandler.cpp File Reference

```
#include "../include/FileHandler.h"
#include "../include/Ticket.h"
#include "../include/TicketManager.h"
#include <fstream>
#include <iostream>
#include <string>
#include <ctime>
```

5.11 FileHandler.cpp

Go to the documentation of this file.

```
00001 #include "../include/FileHandler.h"
00002 #include "../include/Ticket.h"
00003 #include "../include/TicketManager.h"
00004 #include <fstream>
00005 #include <iostream>
00006 #include <string>
00007 #include <ctime>
00008
00009 void FileHandler::createFile(const std::string& filename) {
00010     std::ofstream file(filename, std::ios::app); // Używamy flagi 'app' aby nie nadpisywać
      istniejącego pliku
00011     if (!file) {
00012         std::cerr « "Nie można utworzyć pliku: " « filename « std::endl;
00013     }
00014     file.close();
00015 }
00016
00017 std::vector<std::string> FileHandler::readFile(const std::string& filename) {
00018     std::vector<std::string> data;
00019     std::string line;
00020     std::ifstream file(filename);
00021
00022     if (!file) {
00023         std::cerr « "Nie można otworzyć pliku: " « filename « std::endl;
00024         return data;
00025     }
00026
00027     while (getline(file, line)) {
00028         data.push_back(line);
00029     }
00030
00031     file.close();
00032     return data;
00033 }
00034
00035 void FileHandler::writeFile(const std::string& filename, const std::vector<std::string>& data) {
00036     std::ofstream file(filename);
00037
00038     if (!file) {
00039         std::cerr « "Nie można otworzyć pliku: " « filename « std::endl;
00040         return;
00041     }
00042
00043     for (const auto& line : data) {
00044         file « line « std::endl;
00045     }
00046
00047     file.close();
00048 }
00049
00050 void FileHandler::exportTicketsToTxt(const std::string& sourceFilename, const std::string&
      targetFilename) {
00051     std::ifstream sourceFile(sourceFilename, std::ios::binary | std::ios::in);
00052     std::ofstream targetFile(targetFilename+".txt");
00053
00054     TicketManager manager;
```

```
00055
00056     if (!sourceFile) {
00057         std::cerr « "Nie można otworzyć pliku źródłowego: " « sourceFilename « std::endl;
00058         return;
00059     }
00060
00061     if (!targetFile) {
00062         std::cerr « "Nie można utworzyć pliku docelowego: " « targetFilename « std::endl;
00063         return;
00064     }
00065
00066     size_t size = 0;
00067     sourceFile.read(reinterpret_cast<char*>(&size), sizeof(size));
00068
00069     for (size_t i = 0; i < size; ++i) {
00070         Ticket ticket;
00071
00072         // Deserializacja ticketu
00073         int id;
00074         std::string title, description, status;
00075         time_t creationDate;
00076         int priority;
00077
00078         sourceFile.read(reinterpret_cast<char*>(&id), sizeof(id));
00079         manager.readStringFromFile(sourceFile, title);
00080         manager.readStringFromFile(sourceFile, description);
00081         sourceFile.read(reinterpret_cast<char*>(&creationDate), sizeof(creationDate));
00082         manager.readStringFromFile(sourceFile, status);
00083         sourceFile.read(reinterpret_cast<char*>(&priority), sizeof(priority));
00084
00085         ticket.setID(id);
00086         ticket.setTitle(title);
00087         ticket.setDescription(description);
00088         ticket.setCreationDate(creationDate);
00089         ticket.setStatus(status);
00090         ticket.setPriority(priority);
00091
00092         // Formatowanie ticketu do postaci tekstowej
00093         char buffer[80];
00094         std::strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", std::localtime(&creationDate));
00095         targetFile « "Ticket ID: " « ticket.getID() « "\n"
00096                    « "Title: " « ticket.getTitle() « "\n"
00097                    « "Description: " « ticket.getDescription() « "\n"
00098                    « "Creation Date: " « buffer « "\n"
00099                    « "Status: " « ticket.getStatus() « "\n"
00100                    « "Priority: " « ticket.getPriority() « "\n\n";
00101     }
00102
00103     sourceFile.close();
00104     targetFile.close();
00105     std::cout « "Tickety zostały pomyślnie wyeksportowane do " « targetFilename « std::endl;
00106 }
```

5.12 src/main.cpp File Reference

```
#include <iostream>
#include "../include/TicketManager.h"
#include "../include/FileHandler.h"
#include "../include/UserManager.h"
#include <windows.h>
```

Functions

- int main ()

5.12.1 Function Documentation

5.12.1.1 main()

```
int main ( )
```

Definition at line 7 of file main.cpp.

5.13 main.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include "../include/TicketManager.h"
00003 #include "../include/FileHandler.h"
00004 #include "../include/UserManager.h"
00005 #include <windows.h>
00006
00007 int main() {
00008     SetConsoleOutputCP(CP_UTF8); // Ustawienie strony kodowej konsoli na UTF-8
00009     TicketManager ticketManager;
00010     UserManager userManager;
00011     FileHandler fileHandler;
00012     std::string filename = "../data/tickets.bin";
00013
00014     // Załaduj tickety z pliku, jeśli istnieje
00015     ticketManager.loadTicketsFromFile(filename);
00016
00017     int choice;
00018     bool running = true;
00019     bool validPriority;
00020
00021     while (running) {
00022         std::cout « "Menu:\n";
00023         std::cout « "1. Dodaj Ticket\n";
00024         std::cout « "2. Pokaż Tickety\n";
00025         std::cout « "3. Aktualizuj Ticket\n";
00026         std::cout « "4. Usuń Ticket\n";
00027         std::cout « "5. Eksportuj Tickety do TXT\n";
00028         std::cout « "6. Wyszukaj Tickety\n";
00029         std::cout « "7. Zapisz i wyjdź\n";
00030         std::cout « "Wybierz opcję: ";
00031         std::cin » choice;
00032
00033         switch (choice) {
00034             case 1: {
00035                 std::string title, description, status;
00036
00037                 int id = ticketManager.getLastTicketId() + 1;
00038
00039                 std::cin.ignore();
00040
00041                 std::cout « "Wprowadź tytuł ticketu: ";
00042                 std::getline(std::cin, title);
00043
00044                 std::cout « "Wprowadź opis ticketu: ";
00045                 std::getline(std::cin, description);
00046
00047                 std::cout « "Wprowadź status ticketu: Zrobione lub Do Zrobienia ";
00048                 std::getline(std::cin, status);
00049
00050                 int priority;
00051                 bool validPriority;
00052
00053                 do {
00054                     std::cout « "Wprowadź priorytet ticketu (1 - najważniejszy, 2 - średnio, 3 -
      najmniej ważny): ";
00055                     std::cin » priority;
00056
00057                     validPriority = (priority == 1 || priority == 2 || priority == 3);
00058
00059                     if (!validPriority) {
00060                         std::cout « "Niepoprawny priorytet. Wybierz 1, 2 lub 3." « std::endl;
00061                     }
00062                 } while (!validPriority);
00063
00064                 Ticket newTicket(id, title, description, status, priority);
00065                 ticketManager.addTicket(newTicket);
00066
00067                 break;
00068             }
00069             case 2: {
00070                 ticketManager.viewTickets();
00071                 break;
00072             }
00073             case 3: {
00074                 int id;
00075                 std::string title, description, status;
00076
00077                 std::cout « "Podaj ID ticketu do aktualizacji: ";
00078                 std::cin » id;
00079                 std::cin.ignore();
00080
00081                 std::cout « "Nowy tytuł ticketu: ";
```

```
00082                 std::getline(std::cin, title);
00083
00084                 std::cout « "Nowy opis ticketu: ";
00085                 std::getline(std::cin, description);
00086
00087                 std::cout « "Nowy status ticketu: Zrobione lub Do Zrobienia ";
00088                 std::getline(std::cin, status);
00089
00090                 int priority;
00091
00092                 do {
00093                     std::cout « "Wprowadź priorytet ticketu (1 - najważniejszy, 2 - średnio, 3 -
      najmniej ważny): ";
00094                     std::cin » priority;
00095
00096                     validPriority = (priority == 1 || priority == 2 || priority == 3);
00097
00098                     if (!validPriority) {
00099                         std::cout « "Niepoprawny priorytet. Wybierz 1, 2 lub 3." « std::endl;
00100                     }
00101                 } while (!validPriority);
00102
00103
00104                 ticketManager.updateTicket(id, title, description, status, priority);
00105                 break;
00106             }
00107             case 4: {
00108                 int id;
00109                 std::cout « "Podaj ID ticketu do usunięcia: ";
00110                 std::cin » id;
00111                 ticketManager.removeTicket(id);
00112                 break;
00113             }
00114             case 5: {
00115                 std::string sourceFilename = filename; // Nazwa pliku binarnego
00116                 std::string targetFilename;
00117
00118                 std::cout « "Podaj nazwę pliku docelowego do eksportu: ";
00119                 std::cin » targetFilename;
00120
00121                 fileHandler.exportTicketsToTxt(sourceFilename, "../"+targetFilename);
00122                 break;
00123             }
00124             case 6: {
00125                 int searchType;
00126                 std::cout « "Wybierz typ wyszukiwania: 1 - Priorytet, 2 - Status: ";
00127                 std::cin » searchType;
00128
00129                 if (searchType == 1) {
00130                     int priority;
00131                     std::cout « "Wpisz priorytet (1, 2, 3): ";
00132                     std::cin » priority;
00133                     ticketManager.searchTicketsByPriority(priority);
00134                 } else if (searchType == 2) {
00135                     std::string status;
00136                     std::cout « "Wpisz status (np. Zrobione, Do Zrobienia): ";
00137                     std::cin.ignore();
00138                     std::getline(std::cin, status);
00139                     ticketManager.searchTicketsByStatus(status);
00140                 } else {
00141                     std::cout « "Nieprawidłowy wybór." « std::endl;
00142                 }
00143                 break;
00144             }
00145             case 7: {
00146                 ticketManager.saveTicketsToFile(filename);
00147                 running = false;
00148                 break;
00149             }
00150             default:
00151                 std::cout « "Nieznana opcja!\n";
00152         }
00153     }
00154
00155     return 0;
00156 }
```

5.14 src/Ticket.cpp File Reference

```
#include "../include/Ticket.h"
```

5.15 Ticket.cpp

Go to the documentation of this file.

```
00001 #include "../include/Ticket.h"
00002
00003 // Konstruktor
00004 Ticket::Ticket(int id, std::string title, std::string description, std::string status, int priority)
00005     : ID(id), title(title), description(description), creationDate(time(nullptr)), status(status),
      priority(priority) {}
00006
00007 // Gettery
00008 int Ticket::getID() const { return ID; }
00009 std::string Ticket::getTitle() const { return title; }
00010 std::string Ticket::getDescription() const { return description; }
00011 time_t Ticket::getCreationDate() const { return creationDate; }
00012 std::string Ticket::getStatus() const { return status; }
00013 int Ticket::getPriority() const { return priority; }
00014
00015 // Settery
00016 void Ticket::setID(int newID) { ID = newID; }
00017 void Ticket::setTitle(const std::string& newTitle) { title = newTitle; }
00018 void Ticket::setDescription(const std::string& newDescription) { description = newDescription; }
00019 void Ticket::setCreationDate(time_t newCreationDate) { creationDate = newCreationDate; }
00020 void Ticket::setStatus(const std::string& newStatus) { status = newStatus; }
00021 void Ticket::setPriority(int newPriority) { priority = newPriority; }
```

5.16 src/TicketManager.cpp File Reference

```
#include "../include/TicketManager.h"
#include "../include/Ticket.h"
#include <fstream>
#include <iostream>
#include <iomanip>
#include <ctime>
#include <algorithm>
```

Functions

- void writeStringToFile (std::ofstream &file, const std::string &str)

5.16.1 Function Documentation

5.16.1.1 writeStringToFile()

```
void writeStringToFile (
            std::ofstream & file,
            const std::string & str )
```

Definition at line 75 of file TicketManager.cpp.

5.17 TicketManager.cpp

Go to the documentation of this file.

```
00001 #include "../include/TicketManager.h"
00002 #include "../include/Ticket.h"
00003 #include <fstream>
00004 #include <iostream>
00005 #include <iomanip>
00006 #include <ctime>
00007 #include <algorithm>
00008
00009 void TicketManager::addTicket(Ticket& ticket) {
00010   // Pobierz następny dostępny identyfikator
00011   int nextId = getLastTicketId() + 1;
00012
00013   // Ustaw identyfikator nowego biletu
00014   ticket.setID(nextId);
00015
00016   // Dodaj nowy bilet do wektora
00017   tickets.push_back(ticket);
00018
00019   // Wyświetl komunikat o sukcesie
00020   std::cout « "Ticket dodany pomyślnie." « std::endl;
00021 }
00022
00023 void TicketManager::readStringFromFile(std::ifstream& file, std::string& str) {
00024     size_t length;
00025     file.read(reinterpret_cast<char*>(&length), sizeof(length));
00026     str.resize(length);
00027     file.read(&str[0], length);
00028 }
00029
00030 void TicketManager::loadTicketsFromFile(const std::string& filename) {
00031     std::ifstream file(filename, std::ios::binary | std::ios::in);
00032
00033     if (!file) {
00034         std::cerr « "Nie można otworzyć pliku do odczytu: " « filename « std::endl;
00035         return;
00036     }
00037
00038     size_t size = 0;
00039     file.read(reinterpret_cast<char*>(&size), sizeof(size));
00040
00041     tickets.clear();
00042     for (size_t i = 0; i < size; ++i) {
00043         Ticket ticket;
00044
00045         int id;
00046         file.read(reinterpret_cast<char*>(&id), sizeof(id));
00047         ticket.setID(id);
00048
00049         std::string title;
00050         readStringFromFile(file, title);
00051         ticket.setTitle(title);
00052
00053         std::string description;
00054         readStringFromFile(file, description);
00055         ticket.setDescription(description);
00056
00057         time_t creationDate;
00058         file.read(reinterpret_cast<char*>(&creationDate), sizeof(creationDate));
00059         ticket.setCreationDate(creationDate);
00060
00061         std::string status;
00062         readStringFromFile(file, status);
00063         ticket.setStatus(status);
00064
00065         int priority;
00066         file.read(reinterpret_cast<char*>(&priority), sizeof(priority));
00067         ticket.setPriority(priority);
00068
00069         tickets.push_back(ticket);
00070     }
00071
00072     file.close();
00073 }
00074
00075 void writeStringToFile(std::ofstream& file, const std::string& str) {
00076     size_t length = str.size();
00077     file.write(reinterpret_cast<const char*>(&length), sizeof(length));
00078     file.write(str.c_str(), length);
00079 }
00080
00081 void TicketManager::saveTicketsToFile(const std::string& filename) {
00082     std::ofstream file(filename, std::ios::binary | std::ios::out);
```

```
00083
00084     if (!file) {
00085         std::cerr « "Nie można otworzyć pliku do zapisu: " « filename « std::endl;
00086         return;
00087     }
00088
00089     size_t size = tickets.size();
00090     file.write(reinterpret_cast<char*>(&size), sizeof(size));
00091
00092     for (const auto& ticket : tickets) {
00093         int id = ticket.getID();
00094         file.write(reinterpret_cast<const char*>(&id), sizeof(id));
00095         writeStringToFile(file, ticket.getTitle());
00096         writeStringToFile(file, ticket.getDescription());
00097         time_t creationDate = ticket.getCreationDate();
00098         file.write(reinterpret_cast<const char*>(&creationDate), sizeof(creationDate));
00099         writeStringToFile(file, ticket.getStatus());
00100         int priority = ticket.getPriority();
00101         file.write(reinterpret_cast<const char*>(&priority), sizeof(priority));
00102     }
00103
00104     file.close();
00105 }
00106
00107 void TicketManager::viewTickets() const {
00108     if (tickets.empty()) {
00109         std::cout « "Brak ticketów do wyświetlenia." « std::endl;
00110         return;
00111     }
00112
00113     for (const auto& ticket : tickets) {
00114         std::cout « "Ticket ID: " « ticket.getID() « "\n"
00115                   « "Tytuł: " « ticket.getTitle() « "\n"
00116                   « "Opis: " « ticket.getDescription() « "\n"
00117                   « "Status: " « ticket.getStatus() « "\n"
00118                   « "Priorytet: " « ticket.getPriority() « "\n";
00119
00120         // Formatowanie daty
00121         char buffer[80];
00122         std::time_t creationDate = ticket.getCreationDate();
00123         std::strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", std::localtime(&creationDate));
00124         std::cout « "Data utworzenia: " « buffer « "\n"
00125                   « "--------------------------------\n";
00126     }
00127 }
00128
00129 int TicketManager::getLastTicketId() {
00130   // Jeśli wektor biletów jest pusty, zwracaj 0
00131   if (tickets.empty()) {
00132     return 0;
00133   }
00134
00135   // W przeciwnym razie zwróć identyfikator ostatniego biletu
00136   return tickets.back().getID();
00137 }
00138
00139 void TicketManager::updateTicket(int id, const std::string& newTitle, const std::string&
      newDescription, const std::string& newStatus, int newPriority) {
00140     for (auto& ticket : tickets) {
00141         if (ticket.getID() == id) {
00142             ticket.setTitle(newTitle);
00143             ticket.setDescription(newDescription);
00144             ticket.setStatus(newStatus);
00145             ticket.setPriority(newPriority);
00146             std::cout « "Ticket o ID " « id « " został zaktualizowany." « std::endl;
00147             return;
00148         }
00149     }
00150     std::cout « "Nie znaleziono ticketu o ID " « id « "." « std::endl;
00151 }
00152
00153 void TicketManager::removeTicket(int id) {
00154     auto it = std::find_if(tickets.begin(), tickets.end(), [id](const Ticket& ticket) {
00155         return ticket.getID() == id;
00156     });
00157
00158     if (it != tickets.end()) {
00159         tickets.erase(it);
00160         std::cout « "Ticket o ID " « id « " został usunięty." « std::endl;
00161     } else {
00162         std::cout « "Nie znaleziono ticketu o ID " « id « "." « std::endl;
00163     }
00164 }
00165
00166 void TicketManager::searchTicketsByPriority(int priority) const {
00167     bool found = false;
00168     for (const auto& ticket : tickets) {
```

```
00169         if (ticket.getPriority() == priority)
00170         {
00171             std::cout « "Ticket ID: " « ticket.getID() « "\n"
00172             « "Tytuł: " « ticket.getTitle() « "\n"
00173             « "Opis: " « ticket.getDescription() « "\n"
00174             « "Status: " « ticket.getStatus() « "\n"
00175             « "Priorytet: " « ticket.getPriority() « "\n";
00176 
00177             // Formatowanie daty
00178             char buffer[80];
00179             std::time_t creationDate = ticket.getCreationDate();
00180             std::strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", std::localtime(&creationDate));
00181             std::cout « "Data utworzenia: " « buffer « "\n"
00182             « "--------------------------------\n";
00183             found = true;
00184         }
00185 
00186     }
00187     if (!found) {
00188         std::cout « "Nie znaleziono ticketów o podanym priorytecie." « std::endl;
00189     }
00190 }
00191 
00192 void TicketManager::searchTicketsByStatus(const std::string& status) const {
00193     bool found = false;
00194     for (const auto& ticket : tickets) {
00195         if (ticket.getStatus() == status)
00196         {
00197             std::cout « "Ticket ID: " « ticket.getID() « "\n"
00198             « "Tytuł: " « ticket.getTitle() « "\n"
00199             « "Opis: " « ticket.getDescription() « "\n"
00200             « "Status: " « ticket.getStatus() « "\n"
00201             « "Priorytet: " « ticket.getPriority() « "\n";
00202 
00203             // Formatowanie daty
00204             char buffer[80];
00205             std::time_t creationDate = ticket.getCreationDate();
00206             std::strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", std::localtime(&creationDate));
00207             std::cout « "Data utworzenia: " « buffer « "\n"
00208             « "--------------------------------\n";
00209             found = true;
00210         }
00211     }
00212     if (!found) {
00213         std::cout « "Nie znaleziono ticketów o podanym statusie." « std::endl;
00214     }
00215 }
00216 
00217 
00218 // Implementacja pozostałych metod
```

5.18 src/UserManager.cpp File Reference

```
#include "../include/UserManager.h"
```

5.19 UserManager.cpp

Go to the documentation of this file.

```
00001 #include "../include/UserManager.h"
00002 
00003 void UserManager::addUser(const std::string& username, const std::string& permission) {
00004     userPermissions[username] = permission;
00005 }
00006 
00007 void UserManager::removeUser(const std::string& username) {
00008     userPermissions.erase(username);
00009 }
00010 
00011 bool UserManager::checkPermission(const std::string& username, const std::string& permission) const {
00012     auto it = userPermissions.find(username);
00013     return it != userPermissions.end() && it->second == permission;
00014 }
00015 
00016 // Implementacja dodatkowych metod
```

Index