IT Helpdesk Platform

Practice Project

By Yannis S.

Table of Contents

Introduction

This project was part of an Object Oriented Software Development course in TU Dublin, Ireland in the year 2022. It was one of the three exercises required to pass the course and it received a mark of 96 out of 100. The project is about the creation of a piece of software that is addressed to IT technicians to manage their IT Helpdesk. The program needs to be able to register and store tickets for various issues reported into the Helpdesk. Additionally, the program needs to be able to sort these issues out by priority and offer a singly linked list for the IT technicians to work on each ticket via a queue. Finally, the program stores the resolved ticket into a database, for data persistence, and logs any errors that might appear in a text log for the technicians to review later on.

The solution provided with this report focuses on all the above with the addition of automated methods to generate a timestamp for each ticket as well as an identification number for each ticket. The program utilises a console menu and everything is completed through it using option menus. The underlying function of the program is fairly simple. It adds, views, and edits tickets. A more specific description for how the program functions follows in the next sections.

Feature 1 - Console Menu User Interface

The user interface assumes the users are familiar with using console menus. For simplification reasons, a decision was made that most of the available operations would be given a limited number of options which would automate and speed up the process of resolving an issue. Therefore, in this interface, users are asked to indicate their choice by choosing the corresponding number from the menu options.
More specifically, the following options are given in the main menu (Figure 1):

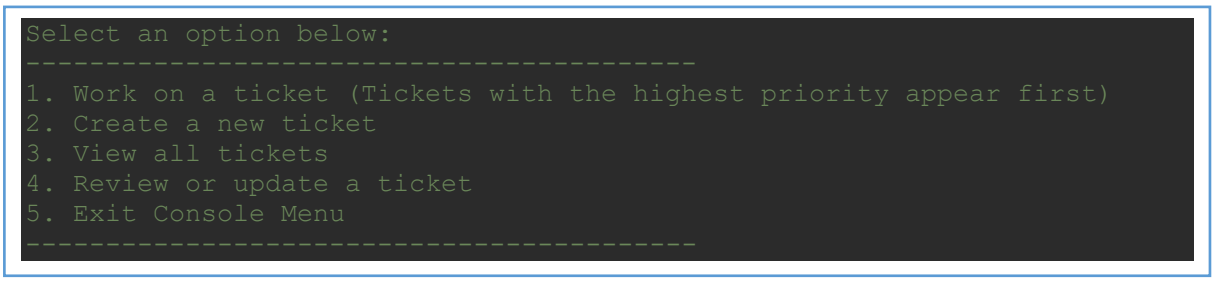

Figure 1

Each menu option above, except option 3 (“View all tickets”), leads to a different submenu of varying complexity that is responsible for a different function of the program. Option 2 asks the user for input for registering a new ticket. Option 3 displays all tickets in the list. Option 4 displays the whole ticket list and offers the options of choosing and editing almost all of the

parameters of a ticket except the timestamp and number ID. Finally, option 5 terminates the program.

Option 1 displays the ticket with the highest priority first and offer the IT technician the ability to view the ticket description, update its status or go back to the main menu. Option 2 allows the user to log in a new ticket. The user is asked to input the following information (Figure 2).

```
ho reports the issue? Please insert the user's name:
 Please fill in the user's email:
 Please fill in the user's department:
 Please fill in in the user's contact number:
 Please fill in a description of the issue:
 Please indicate a level of priority for this issue between 1 and 10:
```

Figure 2

Option 3 displays the whole list of tickets and then returns back to the main menu. When choosing to edit or review a ticket (option 4) then the users are asked which parameter they wish to update and have to input in string format the updated information (Figure 3).

```
What would you like to do with this ticket?
Select an option below:
-------------------------------------------
1. Update the NAME.
2. Update the EMAIL.
3. Update the DEPARTMENT.
4. Update the CONTACT NO.
5. Update the DESCRIPTION.
6. Update the PRIORITY.
7. Update the STATUS.
8. Go back
-------------------------------------------
Please enter your choice here: >
```

Figure 3

Feature 2 – Functionality

As stated above the system is fairly straightforward. It uses a set of console menus that offer displaying, viewing and editing tickets. The users are shown simple menus with specific options to work through on the front end. The tickets are all sorted in a singly linked list depending on their priority value; the higher the priority the closer they are to the front of the list. The system consists of eight classes (Table 1) and several methods per class. The classes were built with the 'single priority' principle in mind and each one focuses on just one set of relevant tasks that all revolve around one task e.g. tasks around the singly linked list, tasks around the construction of nodes etc.

Except for the menus the program works in the following way. The user initialises the system. The system initialises the database and registers the existing ticket objects that were

previously set (set by the software designer for testing purposes in this instance). The tickets are automatically given a timestamp and an identifier number (ID). The ticket ID is created using the FileService class that implements a text file that stores the last ticket ID used and references that to generate new ticket IDs. The tickets are then processed in order to be included in the singly linked list (SLL). This means that they are turned from ticket objects into node objects and then they are added into the SLL where they are finally sorted by priority.

Then the program displays the main menu and waits the user to make a choice. Depending on that choice a different function is run in the background. For example, if the user chooses to view the SLL then the program the relevant method prints out the sorted list. The user can go through all the aforementioned menus of the program and make changes into the list. If for example the status of a ticket is changed to "Resolved" then the ticket is removed from the list. The resolved ticket gets added into a database file that ensures data persistence. This happens on the back end of the program. Finally, if any user errors appear during the runtime of the program, an error log text file is created and they are added unto it for future reference.

Class Name	Main Purpose
Ticket	Create ticket objects for IT issues
Node	Stores ticket objects as linked objects in the SLL
SLL (Singly Linked List)	Stores a chain of linked nodes each corresponding to a ticket
HelpDesk	Runs the main functions of the program and connect all different functions together
Menu	Runs all the external functions that interact with the users and connects it with the HelpDesk and all the backend methods
Fileservice	It is responsible for creating a unique number identifier for each ticket
ErrorLog	It creates a .txt file that stores all error messages appearing due to user errors
DBService	Runs SQL queries and creates a database that stores all resolved tickets that have passed through the system.

Table 1

Feature 3 – Data Persistence

To implement data persistence this system utilises the DBService class that produces a database where all resolved tickets are stored with their full information such as timestamp, unique ID, username, issue description etc. This data is available after the termination of the program. Of course, since, for testing purposes, the tickets are reinitialised every time the program runs it is possible to store essentially some duplicate references in the database. However, the timestamp and/or the unique ID will be different.

Another way that data persistence is implemented is through the use of the Fileservice class which produces of a text file that stores the last unique ID number generated. Although, this does not store a full set of ticket data it does however provide persistence for the generation of new ticket identifiers even after multiple runs of the program.

Feature 4 – Data Structure and Algorithm (DSA).

There were mainly three different data structures chosen for this project. First are the ticket objects. Then are the nodes and finally, the most complex one of all, is the SLL. A detailed description for each is provided in Table 2 below:

Data Structure	Main Purpose
Tickets	It stores all the required information for the the issue and it is provided by the user. Parameters include: username, timestamp, contact info, issue description, ticket ID, priority, and status
Nodes	Each node stores one ticket in its data section and the reference to another node which is the next link in the SLL (points to the next link in the priority queue)
SLL	Stores all the nodes, and therefore all the tickets, in a specific priority order and stores all the links from node to node. It also stores information on which one is the current head node and how many nodes are included in the list.

Table 2

In a nutshell, the algorithm works like this. The system takes user input, stores it into tickets which in order to be stored in a linked and prioritised manner they are turned into nodes, they are linked together where one node is pointing to the next and this series of nodes is stored as a singly linked list.

This structure was adopted because it is a fairly simple and straightforward way of storing complex information in an ordered fashion which is not easily mutable or accessible by outside forces. This way the order of information remains internally consistent and reliable. Additionally, in the front end it provides users with a minimalistic tool that only expects specific actions without making things too complicated. Implementing a SLL, in principle is easy, however establishing the right priority order that needs to run multiple iterations per each session can be challenging. Ultimately, when achieved it offers better system stability.

The priority values are intended to be provided by the IT technician when registering each ticket. This can be later reviewed and changed via the console menu. When a ticket is registered into the system and added into the SLL its priority is the most defining parameter because it is the one that determines what will be its position in the list. The ticket priority is also stored with the node and the SLL class uses that to arrange where the node will be placed. That is achieved by the '*push*' method within the SLL. This method takes as arguments the ticket data (ticket object) and its priority.

Feature 5 – Program Structure

The main program is in the main.py file is very short. Everything is done mainly by the Menu and Helpdesk classes in the front end and most of the system's back end is done by the aforementioned data structures (tickets, nodes, SLL) and the DBService and Fileservice. The structure follows a class within a class structure where not all classes are called from every

other class but there are dependencies. For example, the Node class is only called by the SLL class but the functionality of many of the other classes depends on the correct execution of the Node class.

Each class is provided with its own essential methods for running reliably. There are however a couple of methods that although, not necessary for this implementation of the system if fully developed could marginally improve the functionality of the system in the future (see comments in the .py files).

Conclusion

In conclusion, implementing a priority singly linked list has been a programming challenge. It has taken me two to three weeks just to implement the list in this program. Initially I was not sure why it was useful but as I was working on it and failing to make it run properly it started to make more sense. The first implementations started working but only if the tickets were provided in a specific order (!). That quickly became very frustrating. After a lot (a lot!) of trial and error and using every resource I had available I reached an implementation that worked well regardless of the order the tickets were given.

Another challenge has been the use of classes within classes. The intention is to keep the code as simple, clear and minimalistic as possible. The code works but there might still be some redundancy in it that could be improved in the future, after some careful consideration.

The rest of the system requirements were easy. Implementing the menus, I borrowed heavily from previous assignments. Data persistence has been an interesting exercise in connecting python scripts with SQL queries, but it was straightforward and easy to implement.

Given more time there are certainly a few improvements that could be made aesthetically (making the code more readable) and in terms of memory and speed. All in all, a good and challenging project that has certainly taught me a lot about software development!