Summer Industry Enrichment Program 2025

L3DOTNET02 Data Structure and Algorithm SIEP Project

Group: DotNet 2

Project Title: Music Playlist Manager

Student Name: Simanka Paudel

London Met ID: 23050271

College ID:NP01CP4A230098

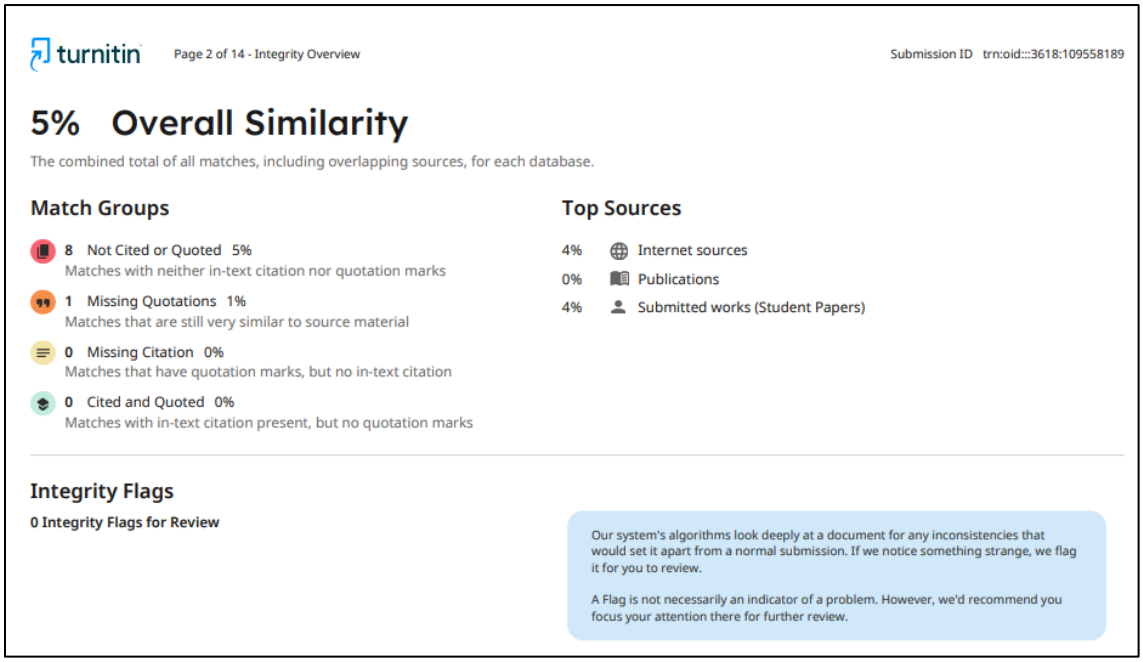
Assignment Due Date: August 28th, 2025

Assignment Submission Date: August 27th, 2025

Submitted to: Abhishek Anand

I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Similarity check



turnitin Page 2 of 14 - Integrity Overview Submission ID trn:oid:::3618:109558189

5% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **8** Not Cited or Quoted 5%
  Matches with neither in-text citation nor quotation marks
- **1** Missing Quotations 1%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

- 4% Internet sources
- 0% Publications
- 4% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Table of Contents

Table of figures

Table of tables

1 Introduction

1.1 Background

The medium used to consume music in the present digital era has changed significantly with most people using digital avenues. It is common to find that the user may be more inclined towards using software that enables the user to organize, play and manages the music experience in a quick and efficient way. Complex interfaces among music player applications exist meaning that the casual user might become overwhelmed when using the application. Simply organized music playlist manager A platform that is command-line based to interact with a music library, make playlists, shuffle songs, and a record of listening history in an easily accessible light-weight manner.

1.2 Problem statement

There are numerous music player applications that are highly graphical, have to be connected to the internet or cloud-based user accounts, which is not preferred by everyone. In addition, almost all the functions such as playlists, queuing and listening histories often involve multiple applications. A light-weight, local and friendly user tool that is capable of managing extensive play lists, and at the same time exemplifying such fundamental data structures as lists, recursion, linked lists, stacks, queues is desired.

1.3 Scope of work

This assignment involves the formulation of a command-line music playlists manager that will emulate the fundamental functionalities of a music player. The system enables end users to:

- Get and have a local song library
- Refine by artist
- Playlists and queues Add songs
- Recursively shuffle playlists
- Keeping track with stacks: listening history

- Control a queue of priority playback of popular songs (party mode)
- Play songs by importing pygame

I did the project in Python with OOP concepts and link lists to describe the playlists, recursion to shuffle and stack/queues to model the history and its queued playback. The system will lack internet streaming, DRM functionality and a graphical user interface features.

2 Aims and objectives

2.1 Aims

To create a command-line music playlist manager that is lightweight, simple and that illustrates fundamental programming concepts and data structures and subsequently give the user a convenient means of managing and listening to their music.

2.2 Objectives

- Add a song library and simple song management capability.
- Using linked lists to support a playlist with flexible insertion and deletion.
- Use recursive shuffle playlist.
- A history of the listening could be kept through stacks.
- Find queues and priority queues to have enhanced playback management.
- Play song accordingly

3 Expected outcomes and deliverables

3.1 Expected Outcomes

- Users are able to browse, filter and sort their song library.
- Playlists may be edited and created.
- Users are able to make songs play queued and give the priority.
- The history of listening is traced and shown.
- Adequate use of data structures and OOP concepts are exhibited in the system.

3.2 Deliverables

- A real python command line music playlist manager.
- Main.py, source code file(song.py, pltalist.py, queuestack.py, utils.py).
- Documentation such as project brief, data structure usage and user guidelines.
- Uses cases tests for core features (add song, shuffle, play next e cetera).

4 Project risks and contingency plans

4.1 Potential risks and threats

- Technical Risks:
  Linked list or recursion implementation bugs might influence the work of playlists.
- Data Loss:
  Upon program exit data loss will occur whether in the list of playlists or playback history (no persistent storage).
- User Input Errors:
  Sometimes invalid input can result in crashes or unexpected behavior.
- Time constraints:
  Complex nature of the implementation of all features within the time frame.

4.2 Contingency Plans

- Technical Hazards:
  Quality testing and design that keeps different parts modular so as to be able to identify and correct faults with the shortest time possible.
- Data Loss:
  Also allow optional file based storing of playlists and history.
- Input User Input Errors:
  Input validation, no crash on errors.
- Limited Time:
  Focus on the fundamental features (a playlist management tool, the play queue, the history) and only later add advanced features (a party mode, a recursive shuffle).

5 Methodology

5.1 Approach and design

5.1.1 Paradigm:

Object-Oriented Programming (OOP) with python. Object-oriented programming OOP is a programming paradigm that creates a method of structuring programs so that properties and behaviors are organized into individual objects.Such objects may include a file representing a person having name, age and address and behaviors like walking, talking, breathing and running. Or it might be an email containing things like a list of recipients, a subject and a body and capabilities such as adding attachments and sending.In other words, object-oriented programming is a method of modeling real-life tangible objects, such as cars, and of course, the relationships between things, e.g. companies and staff or students and teachers. The OOP models entities of a real world as software objects with some data contained on them and the ability to perform a set of operations. (Real Python, 2025)

5.1.2 Architecture:

Package based, modular, CLI is separated by concerns:

- song.py: Domain entity (Song object + upvote state).
- playlist.py: Double linked list to order playback and easily locate previous/next.
- queue_stack.py: PlayQueue (FIFO + priority using heapq), ListeningHistory.
- utils.py (Pure functions, filter, and recursive shuffle).
- main.py -> CLI Controller / Orchestrator (loop of menu, I/O).
- songs(folder): To store downloaded songs

5.1.3 Important Data Structures:

- List (Python list): song library, ordinary queue, history stack container.
  List is an in-built Python structure that is used to store multiple objects. Lots of algorithms use lists (w3 schools, 2025)
- Doubly Linked List: nodes of playlist (prior, next).

Doubly linked list is a one-way traversability in either direction. The reason is that in the list, each of the nodes has two pointers, one to the preceding node and the other one to pointer to the next node. (GeeksforGeeks, 2025)

- Stack: history of what was listened to (push/pop of Song).
  A Stack is a linear data structure that allow only two operations — push and pop , according to the FIFO (first-in-first-out) principle. The order of evalution can be LIFO(Last In First Out) or FILO(First In Last Out). LIFO is a first come – first serve basis that is element inserted last is extracted first and FILO is last in first out that is element inserted first is extracted last. (GeeksforGeeks, 2025)
- Queue: ordinary play queue (FIFO by list front pop). A Queue Data Structure is a building block in computer science that is employed to save and administer information that bears a distinct order. It has a principle of the first in, first out (FIFO) where the first item that was put in the queue will be the first to go out. (GeeksforGeeks, 2025)
- Priority Queue: party mode with heapq on (tuples (-upvotes, song).
  A priority queue is a special queue modified with a value that gives a priority to each element presented in the queue. And, elements are only presented based on their precedence. What this means is that, the elements with higher priority are served first.
- Recursion: Fisher–Yates–style recursive shuffle on a transient array view of the playlist. The cycle when a function calls itself either directly or indirectly is termed recursion and the corresponding function is a recursive function. (GeeksforGeeks, 2025)

5.1.4 Design Principles:

- Single Responsibility: a module has single concern.
- Care at boundaries: the playlist input to utils functions are never mutated except by making an explicit playlist reconstruction.
- Fail-safe I/O: user input will always be checked (scheduled; see 6.2) that a crash can be avoided.
- Extensible: simple to add in persistence, search or ratings or a GUI later.

Main.py lets the user operate the CLI menu. Then, operations are passed to relevant module (playlist/queue/history/utils). Play back takes off of the priority queue, then falls back to playlist order. After that, the song is relegated to the past after any play operation.

5.2 Implementation steps

5.2.1 Model Layer

- Apply Song with title, artist, time, upvotes + __str__.

5.2.2 Linked List (Playlist)

- Song, prev, next-node class.
- Add_song, play_next, play_previous, show, to_list, from_list, clear.

5.2.3 Queues & Stack

- PlayQueue add, add_priority (heap push), pop, show.
- ListeningHistory push, pop, show.

5.2.4 Utilities

- filter_songs(library, artist).
- shuffle_playlist(playlist) -> list, recursive shuffle, rebuild.

5.2.5 Controller / CLI

- Menu with loop (see library, filter, add to playlist, play next/earlier, view, shuffle, queue, party mode, history).
- Playback rule: play_queue.pop() or playlist.play_next().

5.2.6 Error handling (and recommended additions)

- Indexes of guard lists, non-enabled input, wasted collections.
- Non-destructive feedbacks of invalid selections.

5.2.7 Testing & Verification

- Unitlike add/filter/shuffle/queues/history

6 Resources and requirements

6.1 Hardware Requirements

- Any decent PC or laptop that is capable of running Python 3.9+.
- Minimal: 1 CPU core and 2 GB RAM and < 10 MB of storage space on the source code and text data.

6.2 Requirements on the Software

- Python (tested with Python interpreter). Python is very well known programming language. Guido van Rossum developed it and it was launched in the year 1991. Python is executed in the interpreter environment, it can be run immediately after execution of the code. This allows prototyping to be done very fast. (W3Schools, 2025)
- Modules in Standard Library used (heapq, random).
- Installed pygame
- Runs in a common terminal/console (Windows Command Prompt/PowerShell, macOS Terminal, Linux shell).

6.3 Other Resources

- As the primary software development environment, VS Code (VS Code) was adopted. It is a platform where we can work in the place where we can be most productive, whether we are plugged into a cloud service, a remote repository, or surfing in the browser via VS Code in the Web (vscode.dev). (Visual Studio Code, 2025)
- Version control and backup (with Git, optional). Git is a free and open source distributed version control system to manage small to very large projects with blistering speed. (Git, 2025)

7 Work breakdown structure and milestones

7.1 Milestones

Milestones	Deliverables	Learning
Foundations	Song model, base library, CLI menu skeleton	Week 1
Linked List	Playlist with add/next/previous/show	Week 2
Queues/Stack	Play Queue, Listening History	Week 3
Recursion	Recusrive shuffle utilities incorporated	Week 4
Priority mode	Upvotes+heap-based priority queue	Week 5
Robustness	The input validation check, error handling	

Table 1: Milestones

7.2 Work breakdown structure

7.2.1 Requirements Gathering

- Determine that there is need of an offline, command-line music playlist manager, which has to be lightweight.
- Features should be decided: song library, playlist, shuffle, queue, party mode, history.

7.2.2 System Design

- Modular system (song.py, playlist.py, queue_stack.py, utils.py, main.py, song folder).
- Data structures that Map requires: list, linked list, stack, queue, priority queue, recursion.

7.2.3 Model Implementation

- Create the class of a song with the attributes (title, artist, duration, upvotes).

7.2.4 Development of Data Structure

- Design Playlist with doubly linked list.
- Normal + priority queue (heapq) implementation of Play Queue.
- Implementing Listening History.
- Make recursive shuffle utility.

7.2.5 CLI Controller

- Develop main.py where we have user menu and choices (view, filter, add, play next/previous, shuffle, queue, party mode, history).
- Manage communications among modules.

7.2.6 Testing & Validation

- Trying out features on sample library (3 songs).
- Checking the order of playlist, randomness of shuffle, priority of queue and tracking of history.
- Validating error check input (e.g. song number is not available).

8 Tools and technologies used

- Language: Python 3.x,
- Standard built-in library modules: heapq, random, typing (optional hints).

  A heap is a binary tree such that the value of any node is less than or equal to the value of each of its children. We call this the heap invariant. (Python Documentation, 2025)
- Development kits: Git, VS Code, OS terminal.

9 System Dashboard and screenshots

9.1 Main Menu

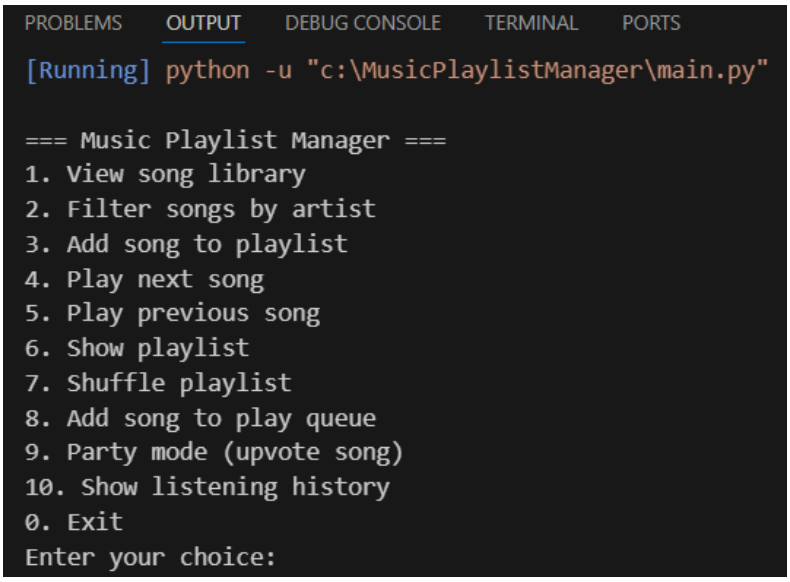


Figure 1: Main menu

The main dashboard provides 9 options to manage the music player system:

- View song library
- Filter songs by artist
- Add song to playlist
- Play next song
- Play previous song
- Show playlist
- Shuffle playlist
- Add song to play queue
- Show listening history

9.2 View Song Library

The system has a song library comprising Songs objects that contain title, artist and duration and upvotes. Such songs are maintained in a list so that they can be traversed, searched, and displayed in entirety.

Structure Data Type: Python List

When I entered my choice 1, which is to “view song library”, the songs in the library are displayed.

```
Enter your choice: 1

--- Song Library ---
1. Shape of You by Ed Sheeran (4.2 min,Upvotes: 0)
2. Blinding Lights by The Weeknd (3.5 min,Upvotes: 0)
3. Levitating by Dua Lipa (3.9 min,Upvotes: 0)
```

Figure 2: Viewing song library

9.3 Filter Songs by Artist

The process of filtering is performed as the iterations are performed on the song library and only the songs whose artists name is equal to the clicked one are chosen. This is done efficiently with list comprehensions to filter fast without modification on the original library.

Data Structures: Python List

The choice ‘2’ is entered in order to filter songs by artist from menu, After, entering choice, it asks for artist name and artist’s name is provided. Then, it displays the songs filtered by artist.

```
Enter your choice: 2
Enter artist name to filter: Ed Sheeran

--- Songs by Ed Sheeran ---
Shape of You by Ed Sheeran (4.2 min,Upvotes: 0)
```

Figure 3: Filtering songs by artist

9.4 Add Song to Playlist

Once the user inserts a song into the playlist, the user places the song in a doubly linked list.The songs get converted to Nodes that point forward (next) and backward (prev).This facilitates normal movement to and fro when playing songs.

The Kind of Data Structure used: Doubly Linked List (Playlist class with nodes).

In order to add playlist, the choice ‘3’ is entered. Then the song number is asked to be passed and after the song number is selected, the song is added to playlist. Two songs are to be add in the playlist.

```
Enter your choice: 3

--- Add Song to Playlist ---
1. Shape of You by Ed Sheeran (4.2 min,Upvotes: 0)
2. Blinding Lights by The Weeknd (3.5 min,Upvotes: 0)
3. Levitating by Dua Lipa (3.9 min,Upvotes: 0)
Select song number: 1
Added 'Shape of You' to playlist.
```

Figure 4: Adding song to playlist

```
--- Add Song to Playlist ---
1. Shape of You by Ed Sheeran (4.2 min,Upvotes: 0)
2. Blinding Lights by The Weeknd (3.5 min,Upvotes: 0)
3. Levitating by Dua Lipa (3.9 min,Upvotes: 0)
Select song number: 2
Added 'Blinding Lights' to playlist.
```

Figure 5: Song added to playlist

9.5 Play Next Song

The second song feature works by moving the pointer on the song on the linked list to go to the next song in the linked list. In case there are no other nodes left, the end of the playlist is reached.

Data Structure: Linked List Traversal (in order of an ECG device) After the successful addition of songs in the playlist, in order to play next song, choice’4’ is entered and it plays the next song in the playlist.

```
Enter your choice: 4
Now playing: Shape of You by Ed Sheeran (4.2 min,Upvotes: 0)
```

Figure 6: Playing next song

9.6 Play Previous Song

The above song feature act of moving the pointer to the preceding node of the linked list. This can be done since each node is kept in possession of a reference to its previous element.

The data structure is the linked list with traversal (tail - previous).

```
Enter your choice: 5
Playing previous: Blinding Lights by The Weeknd (3.5 min,Upvotes: 0)
```

Figure 7: Playing previous song

9.7 Show Playlist

Playlist is presented by traversing the linked list, step by step ranging the list head and printing songs of the elements. This, displays the songs in the order of addition or random order.

The data structure adopted is Doubly linked list.

After the choice, ‘6’ is passed in order to see the playlist. It showed the playlist we added in our system.

```
Enter your choice: 6

--- Playlist ---
1. Shape of You by Ed Sheeran (4.2 min,Upvotes: 0)
2. Blinding Lights by The Weeknd (3.5 min,Upvotes: 0)
```

Figure 8: Playlist shown

9.8 Shuffle Playlist (Recursive Shuffle)

This is what the shuffle feature does in three steps:

- Implement an interface with a linked list converted to a list of songs.
- Use a recursive function of the shuffle that randomly swaps two elements until the list is completely shuffled.

- Delete the previous playlist and recreate as linked list in the new random order.

  Data Structures Used

  - Recursion as shuffling
  - List for temporary storage
  - Linked List for end playlist

```
Enter your choice: 7
Playlist shuffled!
```

Figure 9: Shuffling playlist

In order to shuffle a playlist, choice ‘7’ is entered.

9.9 Add Song to Play Queue

The shuffle feature has two steps:

- Convert the linked list to the list of the songs.
- The play queue enables songs to be queue.

In normal mode, they store songs in a queue and play them in first-in-first out (FIFO) way.

When the priority mode is enabled, songs are inserted using upvotes and stored in a heap based priority queue, so that the most upvoted songs are played first.

Data Structures

- Normal FIFO play Queue
- Upvote-based play using Priority Queue (Heap)

```
Enter your choice: 8

--- Add Song to Play Queue ---
1. Shape of You by Ed Sheeran (4.2 min,Upvotes: 0)
2. Blinding Lights by The Weeknd (3.5 min,Upvotes: 1)
3. Levitating by Dua Lipa (3.9 min,Upvotes: 0)
Select song number: 1
Added 'Shape of You' to queue.
```

Figure 10: song added to queue

After the song is added to queue, it can be played next.

```
Enter your choice: 4
Now playing: Shape of You by Ed Sheeran (4.2 min,Upvotes: 0)
```

9.10 Party Mode (Upvote Priority Queue)

Party Mode enhances the play queue by letting users upvote songs.Songs with more upvotes are pushed into a priority queue (heap), where the highest-priority songs are always played first. This creates a dynamic and fun “party playlist” where popular songs rise to the top automatically.

Data Structure Used: Priority Queue (Heap with upvotes as priority key)

```
Enter your choice: 9

--- Party Mode: Upvote a Song ---
1. Shape of You by Ed Sheeran (4.2 min,Upvotes: 0)
2. Blinding Lights by The Weeknd (3.5 min,Upvotes: 0)
3. Levitating by Dua Lipa (3.9 min,Upvotes: 0)
Select song number to upvote: 2
Upvoted 'Blinding Lights' and added to party queue!
```

Figure 11: Upvoting

To upvote a song, choice ‘9’ is entered and the chosen song is upvoted. Now when we view the playlist, it shows the upvotes each song holds.

```
Enter your choice: 1

--- Song Library ---
1. Shape of You by Ed Sheeran (4.2 min,Upvotes: 0)
2. Blinding Lights by The Weeknd (3.5 min,Upvotes: 1)
3. Levitating by Dua Lipa (3.9 min,Upvotes: 0)
```

Figure 12: Song library

9.11 Show Listening History

When they play certain songs, they are placed on a pile where a history of listening is created. A currently played song is always on the top, and it is easy to go through previously played songs in the opposite (chronologically earlier) order. Data Structure used: Stack (LIFO).

After songs are played in a playlist, it showcases listening history.

```
--- Listening History ---
Blinding Lights by The Weeknd (3.5 min,Upvotes: 0)
Shape of You by Ed Sheeran (4.2 min,Upvotes: 1)
```

Figure 13: Listening history

10 Conclusion and future scopes

The Music Playlist Manager effectively demonstrates how classic data structures such as lists, linked lists, stacks, queues, and priority queues, together with recursion, can be brought together to form a cohesive and lightweight command-line music management tool. Each component of the system highlights the practical application of these data structures in solving real-world problems: lists are used for storing the song library and simple collections, the doubly linked list supports dynamic playlist management with efficient navigation between songs, stacks are employed to maintain a listening history in reverse chronological order, and queues—including a priority queue—enable both regular playback and a party mode where upvoted songs are prioritized. The use of recursion in implementing a shuffle feature further emphasizes how fundamental programming concepts can be applied in creative and efficient ways. The modular design of the project, where the domain model, playlist handling, queue and stack logic, and the main controller are kept separate, ensures clarity, maintainability, and testability. This separation of concerns not only makes the program easier to extend and debug but also highlights good programming practice, making the project a strong example of how theoretical concepts can be translated into a working application that is simple, intuitive, and educational.

The Music Playlist Manager can be improved by a series of further attributes in future. Persistence may be enabled so that libraries, playlists and listening history can be stored and restored using file formats such as JSON. More sophisticated search and sorting, like partial-match search and sorting by duration, title, or artist, would make it very usable. It would also be possible to have the system maintain more extensive metadata; per-song ratings and play counts, last-played track time-stamps. To ensure voting, party mode could be complemented with the decay functions or time priority rules. Timed delays and progress bars during playback simulation would be more realistic experiences and user interface could be thrustful by using colorized output libraries such as curses or rich. Owing to necessity in some ways (as we now have the necessary tools and libraries to do it!), the project may also become a graphical or web application on top of Flask, or Fast API. Lastly, to enhance the reliability and long-time maintenance, one can add automated testing using unit test or py-test.

11 References

GeeksforGeeks, 2025. *GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/dsa/doubly-linked-list/
[Accessed 22 August 2025].

GeeksforGeeks, 2025. *GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/dsa/stack-data-structure/
[Accessed 22 August 2025].

GeeksforGeeks, 2025. *GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/dsa/queue-data-structure/
[Accessed 22 August 2025].

GeeksforGeeks, 2025. *GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/introduction-to-recursion-2/
[Accessed 24 August 2025].

Git, 2025. *Git.* [Online]
Available at: https://git-scm.com/
[Accessed 24 August 2025].

Programiz, 2025. *Programiz.* [Online]
Available at: https://www.programiz.com/dsa/priority-queue
[Accessed 24 August 2025].

Python Documentation, 2025. *Python Documentation.* [Online]
Available at: https://docs.python.org/3/library/heapq.html
[Accessed 24 August 2025].

Real Python, 2025. *Real Python.* [Online]
Available at: https://realpython.com/python3-object-oriented-programming/
[Accessed 22 August 2025].

Visual Studio Code, 2025. *Visual Studio Code.* [Online]
Available at: https://code.visualstudio.com/
[Accessed 24 August 2025].

w3 schools, 2025. *w3 schools.* [Online]
Available at: https://www.w3schools.com/python/python_dsa_lists.asp
[Accessed 22 August 2025].

W3Schools, 2025. *W3Schools.* [Online]
Available at: https://www.w3schools.com/python/python_intro.asp
[Accessed 24 August 2025].