Technological Institute of the Philippines - Quezon City

Department of Computer Engineering

Lost and Found Management System: Python Implementation Using GUI

Prepared By:

Mamaril, Justin Kenneth

Reyes, Alexzander

San Jose, Alexander

San Juan, Edson Ray

Titong, Lee Ivan

A Final Project Submitted By:

Ma'am Maria Rizette Sayo

In Partial Fulfillment of the Requirements for

Object Oriented Programming 2

December 2024

FINAL PROJECT IN CPE009B

Lost and Found Management System: Python Implementation Using GUI	
Course Code: CPE009B	**Program:** Computer Engineering
Course Title: Object-Oriented Programming 2	**Date Performed:** 10/28/2024
Section: CPE21S4	**Date Submitted:** 12/02/2024
Name(s): - Reyes, Alexzander, - Mamaril, Justin Kenneth, - San Jose, Alexander, - San Juan, Edson Ray, - Titong, Lee Ivan	**Instructor:** Engr. Maria Rizette Sayo

1. Objective(s)

General Objectives:

- The program should have a Graphical User Interface (GUI)
- The program should utilize OOP Concepts (Class, Object, Polymorphism, Inheritance, Encapsulation)
- The program must be related to Computer Engineering
- The program will be done using the Python programming language

2. Intended Learning Outcomes (ILOs)

After this project, the student should be able to:

- Develop and design a graphical user interface in Python for a Lost and Found Management System.
- Develop a comprehensive and user-friendly Lost and Found Management System.
- Implement a system that categorizes items based on their status and allows efficient tracking, management, and retrieval of lost items.
- Integrate features that allow users to accurately match and retrieve lost items based on various criteria such as keywords, categories, and dates, improving the system's overall efficiency.

3. Discussion

Background of the study

In a busy environment of a school, where students are occupied with their daily classes, extracurricular activities, and social interactions, the likelihood of misplacing personal belongings is high. Items like textbooks, money, sports equipment, lunch boxes and other more are often lost in the campus.

To address this issue, here in the Technological Institute of the Philippines Quezon City, the implementation and benefits of Lost and Found application within the school environment is tackled. By digitizing the traditional lost and found process, the application enhances the efficiency of item retrieval. With features such as item tracking, user-friendly buttons, and log-in system and the user can also browse

Significance of the Study

The study's goal is to transform traditional log books into a digital lost and found system that is efficient for the Technological Institute of the Philippines' campus, which will be built in Python programming language.

Students - The study's findings may assist students reduce the hassle of reporting lost and discovered things on campus. Students frequently misplace their belongings on campus, so a system like this would be handy.

Scope and Limitation

The study will focus on the Lost and Found System implemented for the students and instructors at the Technological Institute of the Philippines (TIP) campus in Quezon City. It will explore how the Lost and Found system can cater to their specific needs in managing lost items. The program includes features such as User registration and Login System, User-friendly Interface and Search Functionalities. The Data such as user information, reported lost items, found items and user's account will be saved in textfiles.

to TIP's Quezon City campus and will not extend to other locations. Our primary focus is on the technical development and testing of the system, rather than extensively exploring its financial aspects. Technical Limitation such as limit of accessibility compared to web or mobile applications as the study will not cover the development of mobile version. The study will not address complex user permission levels or roles beyond basic user functionalities. Advanced database management features may be considered for future iterations of the application.

4. Materials and Equipment

- Personal Computer/Laptop
- Anaconda Python
- PyQt5
- Python IDEs (Spyder, Jupyter and/or Pycharm)

5. Procedure

Methodology:

Flowchart

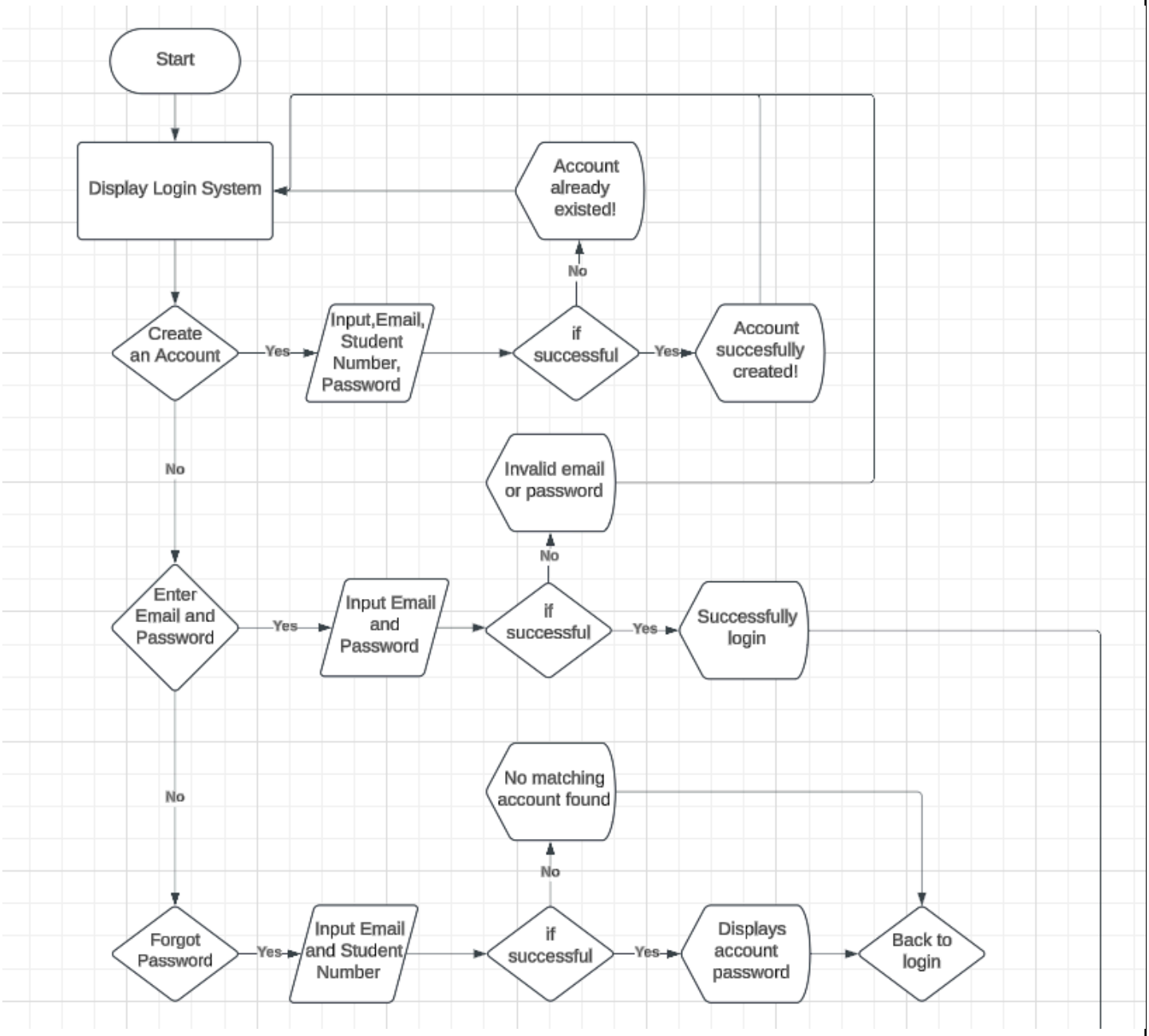

Figure 5.1: *Flowchart of the Program: Login Window*

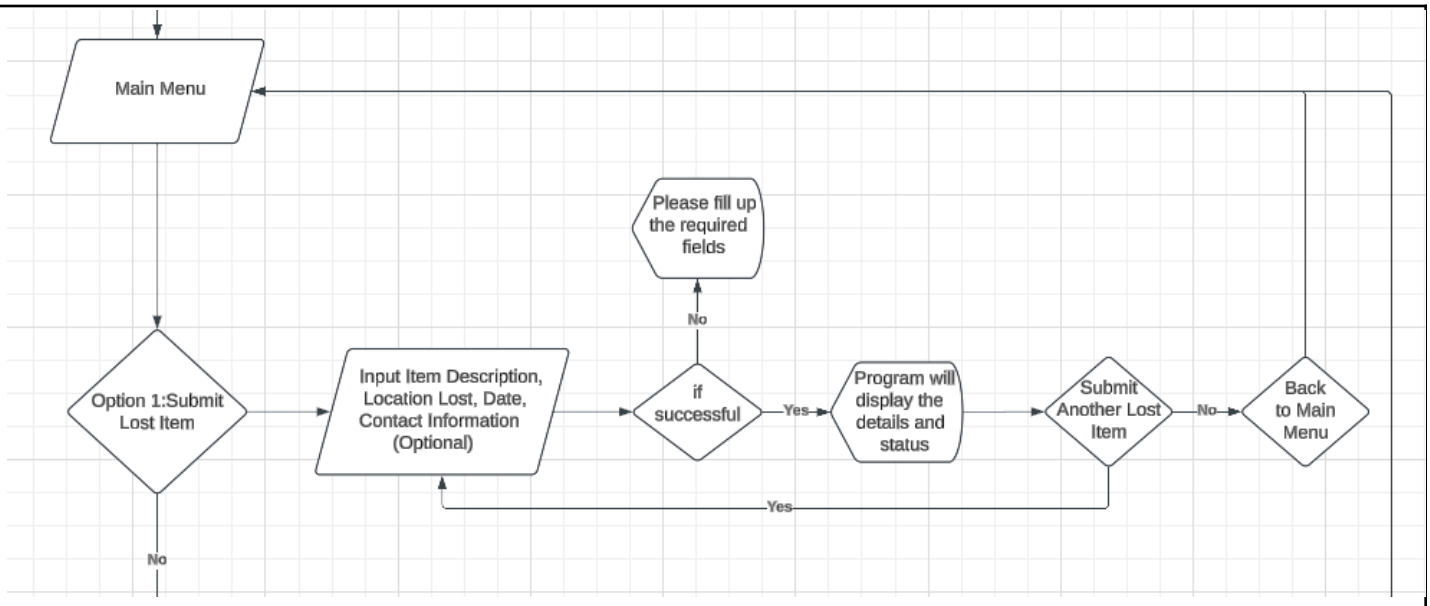

Figure 5.2: *Flowchart of the Program: Main Menu and Option 1 (Submit Lost Item)*

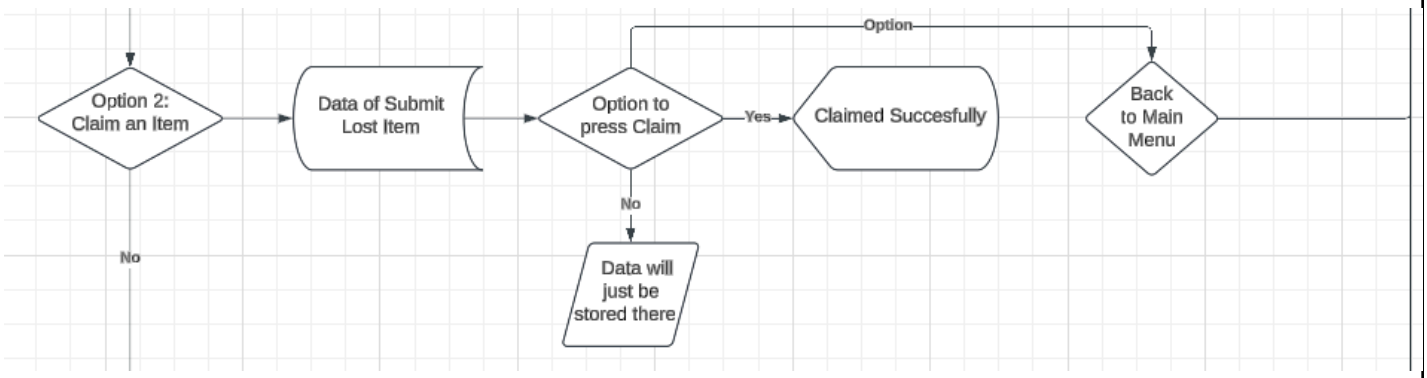

Figure 5.3: *Flowchart of the Program: Option 2 (Claim an Item)*

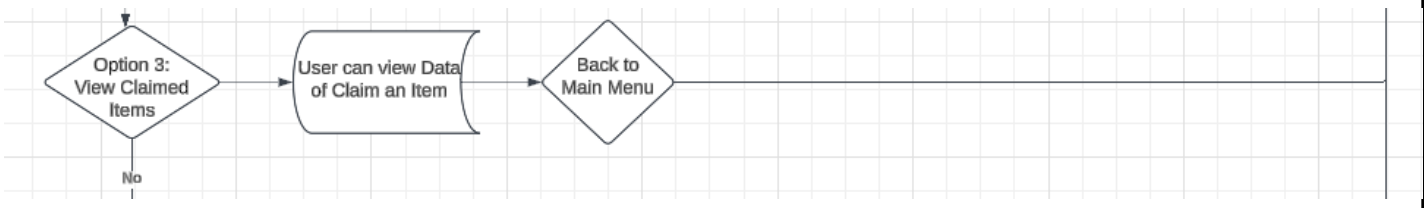

Figure 5.4: *Flowchart of the Program: Option 3 (View Claimed Item)*

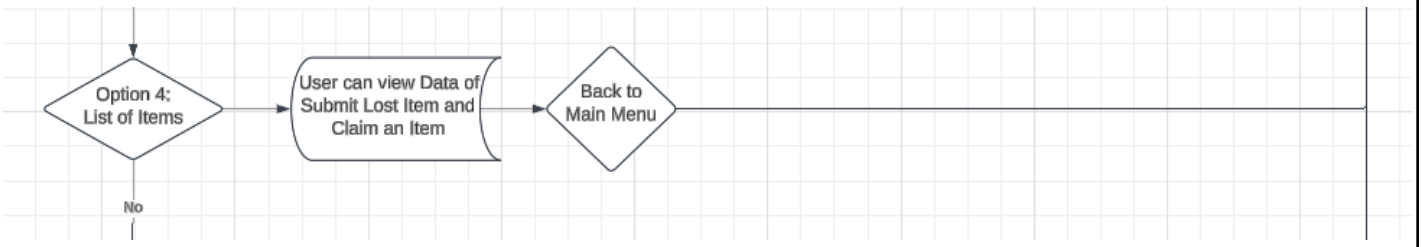

Figure 5.5: *Flowchart of the Program: Option 4 (View Claimed Item)*

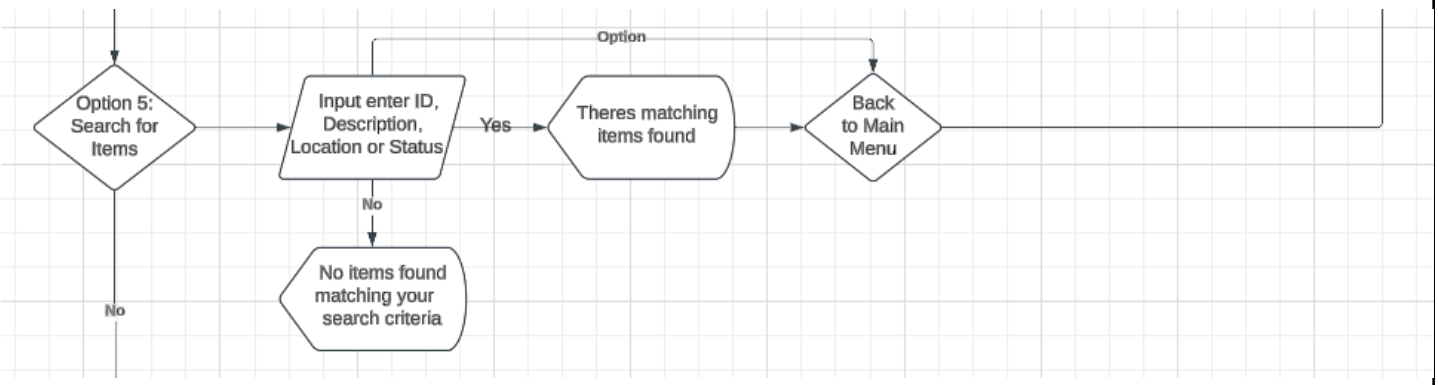

Figure 5.6: *Flowchart of the Program: Option 5 (Search for Item)*

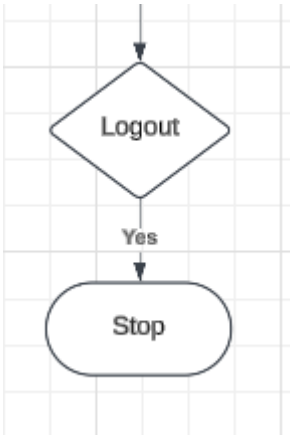

Figure 5.7: *Flowchart of the Program: Option 6 (Logout) and end of program*

Object Oriented Programming Concepts used: (Class, Object, Polymorphism, Inheritance, Encapsulation)

1. Class

Concept:

A class is a blueprint for creating objects. It groups related data and methods to define functionality.

How It Was Used:

- Classes were used to represent various components of the application, such as LoginWindow, DashboardWindow, ClaimItem, and ListOfItems.
- Each class encapsulates specific functionality for different parts of the application.

Example:

```
class LoginWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    def init_ui(self):
        # Define UI elements and logic for the Login window
```

Explanation:

- LoginWindow groups all methods (e.g., handle_login, open_register_window) and attributes (e.g., email and password inputs) related to the login functionality.
- Similarly, classes like ClaimItem and ListOfItems are defined for specific features.

---

2. Object

Concept:

An object is an instance of a class that holds both state (data) and behavior (methods).

How It Was Used:

- Instances (objects) were created for each window or feature in the application. These objects allow interaction with the corresponding UI and functionality.

- For example:
    - login_window = LoginWindow() creates an object of the LoginWindow class.
    - self.claim_item_window = ClaimItem(self, self.user_email) creates an object for the "Claim Item" window.

Example:

```
login_window = LoginWindow()  # Creates a LoginWindow object
login_window.show()  # Displays the login window
```

Explanation:

- Each object represents a specific part of the application, like the login screen or the dashboard.
- Objects allow the program to separate and organize different functionalities.

---

3. Encapsulation

Concept:
Encapsulation is the bundling of attributes and methods inside a class to restrict direct access and enforce controlled interaction.

How It Was Used:

- Attributes like user_email in DashboardWindow and ClaimItem are stored as private or protected attributes. These are accessed and modified only through class methods.
- The methods inside the class act as an interface to interact with the encapsulated data.

Example:

```
class DashboardWindow(QWidget):
    def __init__(self, user_email):
        super().__init__()
        self.user_email = user_email  # Encapsulation of user's email
```

- The user_email is private to the DashboardWindow class and is passed to other windows when needed (e.g., ClaimItem).

Explanation:

- Direct access to user_email is restricted to enforce encapsulation. Other classes interact with it through controlled methods.

---

4. Inheritance

Concept:
Inheritance is a mechanism where one class derives from another, reusing and extending its functionality.

How It Was Used:

- All custom classes like LoginWindow, DashboardWindow, and ClaimItem inherit from QWidget to gain access to PyQt5 GUI features.
- These classes inherit the QWidget attributes and methods while extending them with their own logic.

Example:

```
class LoginWindow(QWidget):  # Inherits QWidget
   def __init__(self):
      super().__init__()
      self.init_ui()
```

Explanation:

- LoginWindow inherits GUI-related methods like show() from QWidget, making it easier to design and display windows.
- Without inheritance, you would need to manually implement these functionalities.

5. Polymorphism

Concept:

Polymorphism allows methods in different classes to have the same name but behave differently based on the context.

How It Was Used:

- Overridden methods like showEvent are customized in child classes to define specific behaviors (e.g., loading items when a window is displayed).
- Similar methods like load_items in ListOfItems and ClaimItem work differently but share the same name, depending on the class context.

Example:

```
def showEvent(self, event):
    self.load_claimed_items()  # Specific behavior for ViewClaimedItems
    super().showEvent(event)
```

- The showEvent method is overridden in ViewClaimedItems to load claimed items when the window is displayed.

Explanation:

- The same method (showEvent) behaves differently in different classes, demonstrating polymorphism.

6. Abstraction

Concept:

Abstraction simplifies complex systems by hiding unnecessary details and exposing only the essential features.

How It Was Used:

- The program hides backend complexities (file reading, data validation) from the user, exposing only a user-friendly interface through buttons and forms.

- For instance, users can submit or claim items using buttons without worrying about how the data is saved or retrieved.

Example:

```
submit_button = QPushButton("Submit Item")
submit_button.clicked.connect(self.submit_item)
```

- The submit_item method handles all complexities (e.g., generating IDs, writing to files), while the user only interacts with the "Submit Item" button.

Explanation:

- Users interact with an abstracted, simplified interface (the GUI) while the program handles the underlying operations.

Summary Table

OOP Concept	How It Was Used
Class	Grouped related data and methods for functionality (e.g., LoginWindow, ClaimItem).
Object	Created instances (e.g., LoginWindow, ClaimItem) to represent specific parts of the application.
Encapsulation	Attributes (e.g., user_email) are bundled and accessed via class methods for controlled interaction.
Inheritance	All classes derived from QWidget to reuse and extend PyQt5 GUI features.
Polymorphism	Overridden methods like showEvent and similar methods (load_items) behave differently across classes.
Abstraction	Simplified user interaction (e.g., buttons and forms) while hiding backend file handling and logic.

Practical Implications

The use of OOP concepts makes the program:

1. **Modular:** Easy to maintain and extend.
2. **Reusable:** Methods and classes can be reused in different contexts.
3. **Scalable:** Additional features can be integrated without disrupting existing code.
4. **User-Friendly:** Abstraction ensures users interact with a simple, intuitive interface.

6. Output

I. Source Code:

```python
import sys
import os
from PyQt5.QtWidgets import (QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel,
QLineEdit, QPushButton, QMessageBox, QComboBox, QListWidget, QTableWidget, QTableWidgetItem)
from PyQt5.QtGui import QFont
from PyQt5.QtCore import Qt, pyqtSignal


def ensure_file_exists(filename):
    if not os.path.exists(filename):
        with open(filename, 'w') as file:
            if filename == "accounts.txt":
                file.write("Email,StudentNumber,Password\n")  # accounts.txt


ensure_file_exists("accounts.txt")
ensure_file_exists("lost_items.txt")
ensure_file_exists("claimed_items.txt")

# Login Window
class LoginWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    def init_ui(self):
        self.setWindowTitle("Lost and Found")
        self.setGeometry(100, 100, 400, 300)
        self.setStyleSheet("background-color: #f7f7f7;")  # Light gray background

        layout = QVBoxLayout()

        self.title = QLabel("LOST AND FOUND")
        self.title.setFont(QFont("Arial", 16, QFont.Bold))
```

```
        self.title.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.title)

        self.subtitle = QLabel("Login System")
        self.subtitle.setFont(QFont("Arial", 9))
        self.subtitle.setAlignment(Qt.AlignCenter)
        self.subtitle.setStyleSheet("color: #4f4f4f;")  # Light gray text
        layout.addWidget(self.subtitle)

        # Email
        self.email_input = QLineEdit(self)
        self.email_input.setPlaceholderText("Email")
        self.email_input.setStyleSheet("""
            background-color: white;
            border: 1px solid #cccccc;
            border-radius: 5px;
            padding: 5px;
    """)
        layout.addWidget(self.email_input)

        # Password
        self.password_input = QLineEdit(self)
        self.password_input.setPlaceholderText("Password")
        self.password_input.setEchoMode(QLineEdit.Password)
        self.password_input.setMaxLength(25)  # Updated to allow up to 25 characters
        self.password_input.setStyleSheet("""
            background-color: white;
            border: 1px solid #cccccc;
            border-radius: 5px;
            padding: 5px;
    """)
        layout.addWidget(self.password_input)

        # Login Button
        self.login_button = QPushButton("Login", self)
        self.login_button.setStyleSheet("""
            background-color: #4caf50;
            color: white;
            border: none;
            border-radius: 5px;
            padding: 10px;
    """)
        self.login_button.clicked.connect(self.handle_login)
        layout.addWidget(self.login_button)

        # Navigation Buttons
        nav_layout = QHBoxLayout()
        self.register_button = QPushButton("Create an Account", self)
        self.register_button.setStyleSheet("""
            background-color: transparent;
            color: #4f4f4f;
            border: none;
```

```
            text-decoration: underline;
        """)
        self.register_button.clicked.connect(self.open_register_window)
        nav_layout.addWidget(self.register_button)

        self.forgot_password_button = QPushButton("Forgot Password?", self)
        self.forgot_password_button.setStyleSheet("""
            background-color: transparent;
            color: #4f4f4f;
            border: none;
            text-decoration: underline;
        """)
        self.forgot_password_button.clicked.connect(self.open_recover_window)
        nav_layout.addWidget(self.forgot_password_button)

        layout.addLayout(nav_layout)

        self.setLayout(layout)

    def handle_login(self):
        email = self.email_input.text()
        password = self.password_input.text()
        try:
            with open("accounts.txt", "r") as file:
                accounts = file.readlines()[1:]
                for account in accounts:
                    parts = account.strip().split(",")
                    if len(parts) == 3:
                        saved_email, _, saved_password = parts
                        if email == saved_email and password == saved_password:
                            QMessageBox.information(self, "Login Successful", "Welcome to
Lost and Found!")
                            self.close()
                            self.dashboard = DashboardWindow(email)  # Redirect to dashboard
after login
                            self.dashboard.show()
                            return
                QMessageBox.warning(self, "Login Failed", "Invalid email or password.")
        except FileNotFoundError:
            QMessageBox.warning(self, "Error", "No accounts found. Please create an account
first.")

    def open_register_window(self):
        self.close()
        self.register_window = RegisterWindow()
        self.register_window.show()

    def open_recover_window(self):
        self.close()
        self.recover_window = RecoverWindow()
        self.recover_window.show()
```

```
# Register Window
class RegisterWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    def init_ui(self):
        self.setWindowTitle("Lost and Found - Create an Account")
        self.setGeometry(100, 100, 400, 300)
        self.setStyleSheet("background-color: #f7f7f7;")

        layout = QVBoxLayout()

        self.title = QLabel("Create an Account")
        self.title.setFont(QFont("Arial", 16, QFont.Bold))
        self.title.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.title)

        self.email_input = QLineEdit(self)
        self.email_input.setPlaceholderText("Email")
        self.email_input.setStyleSheet("""
          background-color: white;
          border: 1px solid #cccccc;
          border-radius: 5px;
          padding: 5px;
      """)
        layout.addWidget(self.email_input)

        self.password_input = QLineEdit(self)
        self.password_input.setPlaceholderText("New Password")
        self.password_input.setStyleSheet("""
          background-color: white;
          border: 1px solid #cccccc;
          border-radius: 5px;
          padding: 5px;
      """)
        layout.addWidget(self.password_input)

        self.student_number_input = QLineEdit(self)
        self.student_number_input.setPlaceholderText("Student Number")
        self.student_number_input.setStyleSheet("""
          background-color: white;
          border: 1px solid #cccccc;
          border-radius: 5px;
          padding: 5px;
      """)
        layout.addWidget(self.student_number_input)

        # Buttons
        self.save_button = QPushButton("Save")
        self.save_button.setStyleSheet("""
          background-color: #4caf50;
```

```
          color: white;
          border-radius: 5px;
          padding: 10px;
      """)
        self.save_button.clicked.connect(self.save_account)
        layout.addWidget(self.save_button)

        self.back_button = QPushButton("Back to Login")
        self.back_button.setStyleSheet("""
          background-color: transparent;
          color: #4f4f4f;
          text-decoration: underline;
      """)
        self.back_button.clicked.connect(self.back_to_login)
        layout.addWidget(self.back_button)

        self.setLayout(layout)

    def save_account(self):
        email = self.email_input.text()
        password = self.password_input.text()
        student_number = self.student_number_input.text()

        if not email or not password or not student_number:
            QMessageBox.warning(self, "Error", "All fields must be filled!")
            return

        try:
            with open("accounts.txt", "r") as file:
                accounts = file.readlines()[1:]
                for account in accounts:
                    saved_email, _, _ = account.strip().split(",")
                    if email.strip().lower() == saved_email.strip().lower():  # Check if
there is an existing email already
                        QMessageBox.warning(self, "Error", "An account with this email
already exists!")
                        return

            # Creating account
            with open("accounts.txt", "a") as file:
                file.write(f"{email},{student_number},{password}\n")  # Add entry below the
header
            QMessageBox.information(self, "Success", "Account created successfully!")
            self.back_to_login()
        except Exception as e:
            QMessageBox.critical(self, "Error", f"Failed to save account: {e}")

    def back_to_login(self):
        self.close()
        self.login_window = LoginWindow()
        self.login_window.show()
```

```
# Forget Password
class RecoverWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    def init_ui(self):
        self.setWindowTitle("Lost and Found - Recover Password")
        self.setGeometry(400, 400, 400, 400)
        self.setStyleSheet("background-color: #f7f7f7;")

        layout = QVBoxLayout()

        self.title = QLabel("Recover Password")
        self.title.setFont(QFont("Arial", 16, QFont.Bold))
        self.title.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.title)

        self.email_input = QLineEdit(self)
        self.email_input.setPlaceholderText("Email")
        self.email_input.setStyleSheet("""
          background-color: white;
          border: 1px solid #cccccc;
          border-radius: 5px;
          padding: 5px;
      """)
        layout.addWidget(self.email_input)

        self.student_number_input = QLineEdit(self)
        self.student_number_input.setPlaceholderText("Student Number")
        self.student_number_input.setStyleSheet("""
          background-color: white;
          border: 1px solid #cccccc;
          border-radius: 5px;
          padding: 5px;
      """)
        layout.addWidget(self.student_number_input)

        # Buttons
        self.recover_button = QPushButton("Recover")
        self.recover_button.setStyleSheet("""
          background-color: #4caf50;
          color: white;
          border-radius: 5px;
          padding: 10px;
      """)
        self.recover_button.clicked.connect(self.recover_account)
        layout.addWidget(self.recover_button)

        self.back_button = QPushButton("Back to Login")
        self.back_button.setStyleSheet("""
          background -color: transparent;
```

```
            color: #4f4f4f;
            text-decoration: underline;
        """)
        self.back_button.clicked.connect(self.back_to_login)
        layout.addWidget(self.back_button)

        self.setLayout(layout)

    def recover_account(self):
        email = self.email_input.text()
        student_number = self.student_number_input.text()
        if not email or not student_number:
            QMessageBox.warning(self, "Error", "All fields must be filled!")
            return

        try:
            with open("accounts.txt", "r") as file:
                accounts = file.readlines()
                for account in accounts:
                    saved_email, saved_student_number, saved_password =
account.strip().split(",")
                    if email == saved_email and student_number == saved_student_number:
                        QMessageBox.information(self, "Password Recovery", f"Your password
is: {saved_password}")
                        return
                QMessageBox.warning(self, "Error", "No matching account found. Please check
your details.")
        except FileNotFoundError:
            QMessageBox.warning(self, "Error", "No accounts found. Please create an account
first.")

    def back_to_login(self):
        self.close()
        self.login_window = LoginWindow()
        self.login_window.show()

# Main Menu
class DashboardWindow(QWidget):
    def __init__(self, user_email):
        super().__init__()
        self.user_email = user_email  # Store user's email
        self.init_ui()

    def init_ui(self):
        # Window settings
        self.setWindowTitle("Lost and Found - Dashboard")
        self.setGeometry(100, 100, 1000, 600)
        self.setStyleSheet("background-color: #f7f7f7;")  # Light gray background color

        # Main layout
        main_layout = QHBoxLayout()
```

```
        # Sidebar layout
        sidebar_layout = QVBoxLayout()
        sidebar_layout.setContentsMargins(10, 10, 10, 10)

        # Sidebar buttons
        sidebar_buttons = [
            ("Submit Lost Item", self.submit_lost_item),
            ("Claim an Item", self.claim_item),
            ("View Claimed Items", self.view_claimed_items),
            ("List of Items", self.list_of_items),
            ("Search for Items", self.search_items),
            ("Logout", self.logout)  # Logout remains last
        ]

        for text, handler in sidebar_buttons:
            button = QPushButton(text)
            button.setStyleSheet("""
               background-color: #4caf50;
               color: white;
               border-radius: 5px;
               padding: 10px;
               margin-bottom: 10px;
           """)
            button.clicked.connect(handler)
            sidebar_layout.addWidget(button)

        # Sidebar widget
        sidebar_widget = QWidget()
        sidebar_widget.setLayout(sidebar_layout)
        sidebar_widget.setStyleSheet("background-color: #eeeeee;")  # Light gray color
sidebar
        sidebar_widget.setFixedWidth(200)

        # Dashboard layout
        self.content_layout = QVBoxLayout()

        self.content_title = QLabel("Welcome to the Dashboard")
        self.content_title.setFont(QFont("Arial", 16, QFont.Bold))
        self.content_title.setAlignment(Qt.AlignCenter)
        self.content_layout.addWidget(self.content_title)

        self.content_message = QLabel("Select an option from the sidebar.")
        self.content_message.setFont(QFont("Arial", 12))
        self.content_message.setAlignment(Qt.AlignCenter)
        self.content_layout.addWidget(self.content_message)

        # Dashboard widget
        content_widget = QWidget()
        content_widget.setLayout(self.content_layout)
        content_widget.setStyleSheet("background-color: #f7f7f7;")

        # Adding sidebar to dashboard layout
```

```
        main_layout.addWidget(sidebar_widget)  # Sidebar place on left
        main_layout.addWidget(content_widget, 1)  # Option features

        self.setLayout(main_layout)

    # Command for sidebar buttons
    def submit_lost_item(self):
        self.clear_content()
        self.submit_window = SubmitLostItem(self)
        self.content_layout.addWidget(self.submit_window)

    def search_items(self):
        self.clear_content()  # Clear the current content
        search_items_widget = SearchItems(self)  # Pass self as the parent
        search_items_widget.back_to_dashboard.connect(self.handle_back_to_dashboard)  #
Connect back to main menu
        self.content_layout.addWidget(search_items_widget)  # SearchItems widget layout

    def view_claimed_items(self):
        self.clear_content()
        self.claimed_window = ViewClaimedItems(self, self.user_email)
        self.claimed_window.back_to_dashboard.connect(self.handle_back_to_dashboard)
        self.content_layout.addWidget(self.claimed_window) # ViewClaimIterm widget layout

    def claim_item(self):
        self.clear_content()
        self.claim_item_window = ClaimItem(self, self.user_email)
        self.claim_item_window.back_to_dashboard.connect(self.handle_back_to_dashboard)
        self.content_layout.addWidget(self.claim_item_window) # ClaimItem widget layout

    def list_of_items(self):  # Handle listing items
        self.clear_content()
        self.list_of_items_window = ListOfItems(self)
        self.list_of_items_window.back_to_dashboard.connect(self.handle_back_to_dashboard)
        self.content_layout.addWidget(self.list_of_items_window) # ListOfItem widget layout

    def logout(self):
        QMessageBox.information(self, "Logout", "Logging out...")
        self.close()
        self.login_window = LoginWindow()
        self.login_window.show()

    def clear_content(self):
        for i in reversed(range(self.content_layout.count())):
            widget = self.content_layout.itemAt(i).widget()
            if widget is not None:
                widget.deleteLater()

    def handle_back_to_dashboard(self):
        self.clear_content()
        self.content_title = QLabel("Welcome to the Dashboard")
        self.content_title.setFont(QFont("Arial", 16, QFont.Bold))
```

```
        self.content_title.setAlignment(Qt.AlignCenter)
        self.content_layout.addWidget(self.content_title)

        self.content_message = QLabel("Select an option from the sidebar.")
        self.content_message.setFont(QFont("Arial", 12))
        self.content_message.setAlignment(Qt.AlignCenter)
        self.content_layout.addWidget(self.content_message)


class SubmitLostItem(QWidget):
    submitted_items = []  # Store submitted items

    def __init__(self, parent=None):
        super().__init__(parent)
        self.parent_window = parent
        self.user_email = parent.user_email  # Logged-in user's email
        self.student_number = self.get_student_number()  # Student number
        self.setWindowTitle("Submit a Lost Item")
        self.init_ui()

    def init_ui(self):
        layout = QVBoxLayout()
        layout.setContentsMargins(20, 20, 20, 20)
        layout.setSpacing(15)

        title_label = QLabel("Submit Item Lost", self)
        title_label.setFont(QFont("Arial", 20, QFont.Bold))
        layout.addWidget(title_label)

        # Item description input
        self.description_input = QLineEdit(self)
        self.description_input.setPlaceholderText("Item Description")
        self.description_input.setFont(QFont("Arial", 12))
        layout.addWidget(self.description_input)

        # Location lost input
        self.location_input = QLineEdit(self)
        self.location_input.setPlaceholderText("Location Lost")
        self.location_input.setFont(QFont("Arial", 12))
        layout.addWidget(self.location_input)

        # Date input
        self.date_input = QLineEdit(self)
        self.date_input.setPlaceholderText("Date (YYYY-MM-DD)")
        self.date_input.setFont(QFont("Arial", 12))
        layout.addWidget(self.date_input)

        # Contact information input
        self.contact_input = QLineEdit(self)
        self.contact_input.setPlaceholderText("Contact Information (optional)")
        self.contact_input.setFont(QFont("Arial", 12))
        layout.addWidget(self.contact_input)
```

```
        # Submit button
        submit_button = QPushButton("Submit Item")
        submit_button.setFont(QFont("Arial", 12))
        submit_button.clicked.connect(self.submit_item)
        layout.addWidget(submit_button)

        # Back to main menu button
        back_button = QPushButton("Back to Main Menu")
        back_button.setFont(QFont("Arial", 12))
        back_button.clicked.connect(self.back_to_menu)
        layout.addWidget(back_button)

        self.setLayout(layout)

    def get_student_number(self):
        """Retrieve student number of the logged-in user."""
        try:
            with open("accounts.txt", "r") as file:
                for line in file:
                    email, _, student_number = line.strip().split(",")
                    if email == self.user_email:
                        return student_number
        except Exception as e:
            print(f"Error retrieving student number: {e}")
        return "Unknown"  # Else student number is not found

    def submit_item(self):
        description = self.description_input.text()
        location = self.location_input.text()
        date = self.date_input.text()
        contact = self.contact_input.text()
        status = "Lost"  # Default status

        if not description or not location or not date:
            QMessageBox.warning(self, "Error", "Please fill in all required fields.")
            return

        # Item ID update
        item_id = self.get_next_item_id()

        # Store item information
        item_info = {
            "id": item_id,
            "email": self.user_email,
            "student_number": self.student_number,
            "description": description,
            "location": location,
            "date": date,
            "status": status,
            "contact": contact
        }
```

```
        SubmitLostItem.submitted_items.append(item_info)

        self.save_to_file(item_info)

        # Display a confirmation message
        item_display_info = (f"Item successfully submitted! Here are the details:\n\n"
                             f"Item ID: {item_id}\n"
                             f"Email Used: {self.user_email}\n"
                             f"Student Number Used: {self.student_number}\n"
                             f"Status: {status}")
        QMessageBox.information(self, "Item Submitted", item_display_info)

        self.clear_fields()

    def save_to_file(self, item_info):
        """Save the submitted item to the lost_items.txt file."""
        try:
            # Check if  file already exists
            file_exists = False
            try:
                with open("lost_items.txt", "r") as file:
                    if file.readline().strip() ==
"ItemID,Email,StudentNumber,Description,Location,Date,Status,Contact":
                        file_exists = True
            except FileNotFoundError:
                pass

            # Append the item to the file
            with open("lost_items.txt", "a") as file:
                if not file_exists:

file.write("ItemID,Email,StudentNumber,Description,Location,Date,Status,Contact\n")

file.write(f"{item_info['id']},{item_info['email']},{item_info['student_number']},"
                           f"{item_info['description']},{item_info['location']},"

f"{item_info['date']},{item_info['status']},{item_info['contact']}\n")
        except Exception as e:
            print(f"Error saving item to file: {e}")

    def get_next_item_id(self):
        """Determining the next available Item ID."""
        try:
            with open("lost_items.txt", "r") as file:
                lines = file.readlines()[1:]
                if lines:
                    last_line = lines[-1]
                    last_id = int(last_line.split(",")[0])
                    return last_id + 1
        except FileNotFoundError:
            return 1  # Start with ID 1 if the file does not exist
        except Exception as e:
```

```
            print(f"Error determining next item ID: {e}")
        return 1

    def clear_fields(self):
        self.description_input.clear()
        self.location_input.clear()
        self.date_input.clear()
        self.contact_input.clear()

    def back_to_menu(self):
        if not hasattr(self, 'parent_window') or not self.parent_window:
            QMessageBox.critical(self, "Error", "Parent window is not defined. Cannot
navigate back to the menu.")
            return

        self.parent_window.clear_content()

        # Back to main menu dashboard message
        self.parent_window.content_title = QLabel("Welcome to the Dashboard")
        self.parent_window.content_title.setFont(QFont("Arial", 16, QFont.Bold))
        self.parent_window.content_title.setAlignment(Qt.AlignCenter)
        self.parent_window.content_layout.addWidget(self.parent_window.content_title)

        self.parent_window.content_message = QLabel("Select an option from the sidebar.")
        self.parent_window.content_message.setFont(QFont("Arial", 12))
        self.parent_window.content_message.setAlignment(Qt.AlignCenter)
        self.parent_window.content_layout.addWidget(self.parent_window.content_message)

# Option 2
class SearchItems(QWidget):
    back_to_dashboard = pyqtSignal()

    def __init__(self, parent):
        super().__init__(parent)
        self.setWindowTitle("Search for Items")
        self.init_ui()

    def init_ui(self):
        layout = QVBoxLayout()

        title_label = QLabel("Search an Item", self)
        title_label.setFont(QFont("Arial", 20, QFont.Bold))
        layout.addWidget(title_label)

        self.keywords_input = QLineEdit(self)
        self.keywords_input.setPlaceholderText("Enter ID, Description, Location, Date, or
Status")
        layout.addWidget(self.keywords_input)

        search_button = QPushButton("Search")
        search_button.clicked.connect(self.search_items)
        layout.addWidget(search_button)
```

```
        # Create a table to display search results
        self.results_table = QTableWidget(self)
        self.results_table.setColumnCount(5)
        self.results_table.setHorizontalHeaderLabels(["ID", "Description", "Location",
"Date", "Status"])
        self.results_table.setRowCount(0)  # Default at 0 rows
        layout.addWidget(self.results_table)

        back_button = QPushButton("Back to Main Menu")
        back_button.clicked.connect(self.back_to_menu)
        layout.addWidget(back_button)

        self.setLayout(layout)

    def search_items(self):
        self.results_table.setRowCount(0)
        keywords = self.keywords_input.text().strip()

        if not keywords:
            QMessageBox.warning(self, "Error", "Please enter a keyword to search.")
            return

        try:
            with open("lost_items.txt", "r") as lost_file:
                lost_items = lost_file.readlines()[1:]
                for lost_item in lost_items:
                    parts = lost_item.strip().split(",")
                    if len(parts) == 8:
                        item_id, email, student_number, description, location, date, status,
contact = parts

                        # Check if the keyword matches any field
                        if (
                                keywords.lower() in item_id.lower()
                                or keywords.lower() in description.lower()
                                or keywords.lower() in location.lower()
                                or keywords.lower() in date.lower()
                                or keywords.lower() in status.lower()
                        ):
                            row_position = self.results_table.rowCount()
                            self.results_table.insertRow(row_position)
                            self.results_table.setItem(row_position, 0,
QTableWidgetItem(item_id))
                            self.results_table.setItem(row_position, 1,
QTableWidgetItem(description))
                            self.results_table.setItem(row_position, 2,
QTableWidgetItem(location))
                            self.results_table.setItem(row_position, 3,
QTableWidgetItem(date))
                            self.results_table.setItem(row_position, 4,
QTableWidgetItem(status))
```

```
        except FileNotFoundError:
            QMessageBox.warning(self, "Error", "The lost items file is missing.")

        try:
            with open("claimed_items.txt", "r") as claimed_file:
                claimed_items = claimed_file.readlines()
                for claimed_item in claimed_items:
                    parts = claimed_item.strip().split(",")
                    if len(parts) == 6:
                        email, item_id, description, location, date, status = parts

                        if (
                                keywords.lower() in item_id.lower()
                                or keywords.lower() in description.lower()
                                or keywords.lower() in location.lower()
                                or keywords.lower() in date.lower()
                                or keywords.lower() in status.lower()
                        ):
                            row_position = self.results_table.rowCount()
                            self.results_table.insertRow(row_position)
                            self.results_table.setItem(row_position, 0,
QTableWidgetItem(item_id))
                            self.results_table.setItem(row_position, 1,
QTableWidgetItem(description))
                            self.results_table.setItem(row_position, 2,
QTableWidgetItem(location))
                            self.results_table.setItem (row_position, 3,
QTableWidgetItem(date))
                            self.results_table.setItem(row_position, 4,
QTableWidgetItem(status))
        except FileNotFoundError:
            QMessageBox.warning(self, "Error", "The claimed items file is missing.")


        if self.results_table.rowCount() == 0:
            QMessageBox.information(self, "No Results", "No items found matching your search
criteria.")

    def back_to_menu(self):
        self.back_to_dashboard.emit()

        if hasattr(self.parent(), 'clear_content'):
            self.parent().clear_content()

        if hasattr(self.parent(), 'init_ui'):
            self.parent().init_ui()
# Option 3
class ViewClaimedItems(QWidget):
    back_to_dashboard = pyqtSignal()

    def __init__(self, parent, user_email):
        super().__init__(parent)
```

```
        self.user_email = user_email
        self.setWindowTitle("View Claimed Items")
        self.init_ui()

    def init_ui(self):
        layout = QVBoxLayout()

        title_label = QLabel("Claimed Items", self)
        title_label.setFont(QFont("Arial", 20, QFont.Bold))
        layout.addWidget(title_label)

        self.claimed_items_table = QTableWidget(self)
        self.claimed_items_table.setColumnCount(6)
        self.claimed_items_table.setHorizontalHeaderLabels(["Email", "ID", "Description",
"Location", "Date", "Status"])

        layout.addWidget(self.claimed_items_table)

        back_button = QPushButton("Back to Main Menu")
        back_button.clicked.connect(self.back_to_menu)
        layout.addWidget(back_button)

        self.setLayout(layout)

    def load_claimed_items(self):
        self.claimed_items_table.setRowCount(0)
        claimed_items_found = False
        try:
            with open("claimed_items.txt", "r") as file:
                claimed_items = file.readlines()
                for item in claimed_items:
                    parts = item.strip().split(",")
                    if len(parts) == 6:
                        email, item_id, description, location, date, status = parts
                        if email == self.user_email:
                            row_position = self.claimed_items_table.rowCount()
                            self.claimed_items_table.insertRow(row_position)
                            self.claimed_items_table.setItem(row_position, 0,
QTableWidgetItem(email))  # Email
                            self.claimed_items_table.setItem(row_position, 1,
QTableWidgetItem(item_id))  # ID
                            self.claimed_items_table.setItem(row_position, 2,
QTableWidgetItem(description))  # Description
                            self.claimed_items_table.setItem(row_position, 3,
QTableWidgetItem(location))  # Location
                            self.claimed_items_table.setItem(row_position, 4,
QTableWidgetItem(date))  # Date
                            self.claimed_items_table.setItem(row_position, 5,
QTableWidgetItem(status))  # Status
                            claimed_items_found = True
                    else:
                        print(f"Skipping line with unexpected format: {item.strip()}")
```

```
            if not claimed_items_found:
                self.show_empty_warning()

        except FileNotFoundError:
            QMessageBox.warning(self, "Error", "No claimed items file found.")

    def show_empty_warning(self):
        msg = QMessageBox()
        msg.setIcon(QMessageBox.Warning)
        msg.setText("No claimed items found!")
        msg.setInformativeText("Please make sure you have claimed items to view.")
        msg.setWindowTitle("Warning")

        parent_geometry = self.parent().geometry()
        msg_size = msg.sizeHint()
        x = parent_geometry.x() + (parent_geometry.width() - msg_size.width()) // 2
        y = parent_geometry.y() + (parent_geometry.height() - msg_size.height()) // 2
        msg.move(x, y)

        msg.exec_()

    def showEvent(self, event):
        self.load_claimed_items()
        super().showEvent(event)

    def back_to_menu(self):
        self.back_to_dashboard.emit()

        if hasattr(self.parent(), 'clear_content'):
            self.parent().clear_content()

        if hasattr(self.parent(), 'init_ui'):
            self.parent().init_ui()

# Option 4
class ClaimItem(QWidget):
    back_to_dashboard = pyqtSignal()

    def __init__(self, parent, user_email):
        super().__init__(parent)
        self.user_email = user_email
        self.claimed_items = []  # List of stored claimed items
        self.setWindowTitle("Claim an Item")
        self.init_ui()
        self.load_claimed_items()  # Load previously claimed items

    def init_ui(self):
        layout = QVBoxLayout()

        title_label = QLabel("Claim an Item", self)
        title_label.setFont(QFont("Arial", 20, QFont.Bold))
```

```
        layout.addWidget(title_label)

        self.items_table = QTableWidget(self)
        self.items_table.setColumnCount(6)
        self.items_table.setHorizontalHeaderLabels(["ID", "Description", "Location", "Date",
"Status", "Action"])
        self.load_items()

        layout.addWidget(self.items_table)

        back_button = QPushButton("Back to Main Menu")
        back_button.clicked.connect(self.back_to_menu)
        layout.addWidget(back_button)

        self.setLayout(layout)

    def load_items(self):
        self.items_table.setRowCount(len(SubmitLostItem.submitted_items))
        for row, item in enumerate(SubmitLostItem.submitted_items):
            self.items_table.setItem(row, 0, QTableWidgetItem(str(item['id'])))
            self.items_table.setItem(row, 1, QTableWidgetItem(item['description']))
            self.items_table.setItem(row, 2, QTableWidgetItem(item['location']))
            self.items_table.setItem(row, 3, QTableWidgetItem(item['date']))
            self.items_table.setItem(row, 4, QTableWidgetItem(item['status']))

            # Claim button for each item
            claim_button = QPushButton("Claim")
            claim_button.clicked.connect(lambda checked, row=row: self.show_warning(row))
            self.items_table.setCellWidget(row, 5, claim_button)

    def show_warning(self, row):
        # Message box
        msg = QMessageBox()
        msg.setIcon(QMessageBox.Warning)
        msg.setText("Are you sure you want to claim this item?")
        msg.setWindowTitle("Claim Item Warning")
        msg.setStandardButtons(QMessageBox.Yes | QMessageBox.No)

        parent_geometry = self.parent().geometry()
        msg_size = msg.sizeHint()
        x = parent_geometry.x() + (parent_geometry.width() - msg_size.width()) // 2
        y = parent_geometry.y() + (parent_geometry.height() - msg_size.height()) // 2
        msg.move(x, y)
        response = msg.exec_()

        if response == QMessageBox.Yes:
            self.claim_item(row)

    # Item details
    def claim_item(self, row):
        item_id = self.items_table.item(row, 0).text()
        description = self.items_table.item(row, 1).text()
```

```
        location = self.items_table.item(row, 2).text()
        date = self.items_table.item(row, 3).text()

        status = "Claimed"

        self.claimed_items.append({
            'user_email': self.user_email,
            'item_id': item_id,
            'description': description,
            'location': location,
            'date': date,
            'status': status  # Updated status
 })

        self.save_claimed_items()

        self.items_table.removeRow(row)  # Remove from the table

        SubmitLostItem.submitted_items = [
            item for item in SubmitLostItem.submitted_items if item['id'] != int(item_id)
        ]

        QMessageBox.information(self, "Claim Successful", f"Item {item_id} has been claimed
successfully!")

        self.load_items()

    def save_claimed_items(self):
        try:
            existing_items = set()
            if os.path.exists("claimed_items.txt"):
                with open("claimed_items.txt", "r") as file:
                    existing_items = {line.strip() for line in file}

            new_items = set()
            for item in self.claimed_items:
                line =
f"{item['user_email']},{item['item_id']},{item['description']},{item['location']},{item['date
']},{item['status']}"
                new_items.add(line)

            with open("claimed_items.txt", "w") as file:
                file.write("\n".join(existing_items | new_items) + "\n")

            self.claimed_items.clear()

        except Exception as e:
            QMessageBox.critical(self, "Error", f"An error occurred while saving claimed
items: {e}")

    def load_claimed_items(self):
        try:
```

```
            with open("claimed_items.txt", "r") as file:
                for line in file:
                    user_email, item_id, description, location, date, status =
line.strip().split(',')
                    self.claimed_items.append({
                        'user_email': user_email,
                        'item_id': item_id,
                        'description': description,
                        'location': location,
                        'date': date,
                        'status': status
                    })
        except FileNotFoundError:
            self.claimed_items = []
        except Exception as e:
            print(f"Error loading claimed items: {e}")

    def back_to_menu(self):
        self.back_to_dashboard.emit()

        if hasattr(self.parent(), 'clear_content'):
            self.parent().clear_content()

        if hasattr(self.parent(), 'init_ui'):
            self.parent().init_ui()

# Option 5
class ListOfItems(QWidget):
    back_to_dashboard = pyqtSignal()

    def __init__(self, parent):
        super().__init__(parent)
        self.setWindowTitle("List of Items")
        self.init_ui()

    def init_ui(self):
        layout = QVBoxLayout()

        title_label = QLabel("List of Submitted Items", self)
        title_label.setFont(QFont("Arial", 20, QFont.Bold))
        layout.addWidget(title_label)

        self.items_table = QTableWidget(self)
        self.items_table.setColumnCount(5)
        self.items_table.setHorizontalHeaderLabels(["ID", "Description", "Location", "Date",
"Status"])
        self.items_table.setRowCount(0)

        self.load_items()

        layout.addWidget(self.items_table)
```

```
        back_button = QPushButton("Back to Main Menu")
        back_button.clicked.connect(self.back_to_menu)
        layout.addWidget(back_button)

        self.setLayout(layout)

    def load_items(self):
        self.items_table.setRowCount(0)
        items_dict = {}  # Avoid duplicates input by ItemID

        try:
            with open("lost_items.txt", "r") as lost_file:
                lost_items = lost_file.readlines()[1:]
                for lost_item in lost_items:
                    parts = lost_item.strip().split(",")
                    if len(parts) == 8:
                        item_id, email, student_number, description, location, date, status,
contact = parts
                        items_dict[item_id] = (description, location, date, status)
        except FileNotFoundError:
            QMessageBox.warning(self, "Error", "The lost items file is missing.")
        except Exception as e:
            QMessageBox.critical(self, "Error", f"An error occurred while loading lost items:
{e}")

        try:
            with open("claimed_items.txt", "r") as claimed_file:
                claimed_items = claimed_file.readlines()
                for claimed_item in claimed_items:
                    parts = claimed_item.strip().split(",")
                    if len(parts) == 6:
                        email, item_id, description, location, date, status = parts
                        items_dict[item_id] = (description, location, date, status)
        except FileNotFoundError:
            QMessageBox.warning(self, "Error", "The claimed items file is missing.")
        except Exception as e:
            QMessageBox.critical(self, "Error", f"An error occurred while loading claimed
items: {e}")

        for item_id, (description, location, date, status) in items_dict.items():
            row_position = self.items_table.rowCount()
            self.items_table.insertRow(row_position)
            self.items_table.setItem(row_position, 0, QTableWidgetItem(item_id))
            self.items_table.setItem(row_position, 1, QTableWidgetItem(description))
            self.items_table.setItem(row_position, 2, QTableWidgetItem(location))
            self.items_table.setItem(row_position, 3, QTableWidgetItem(date))
            self.items_table.setItem(row_position, 4, QTableWidgetItem(status))

    def back_to_menu(self):
        self.back_to_dashboard.emit()

        if hasattr(self.parent(), 'clear_content'):
```

```
            self.parent().clear_content()

        if hasattr(self.parent(), 'init_ui'):
            self.parent().init_ui()

def main():
    app = QApplication(sys.argv)
    login_window = LoginWindow()
    login_window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

Gui Output:

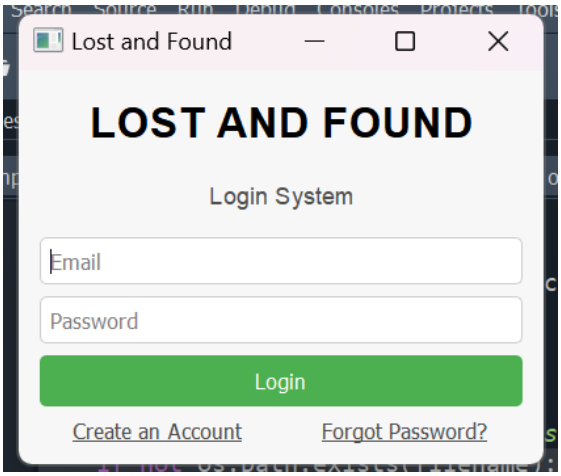

Figure 6.1: *Users Window*

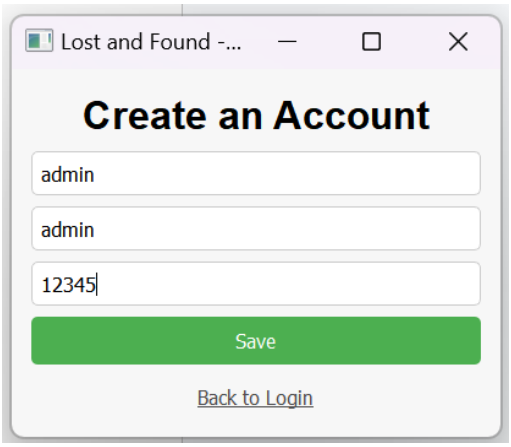

Figure 6.2: *Creating Account*

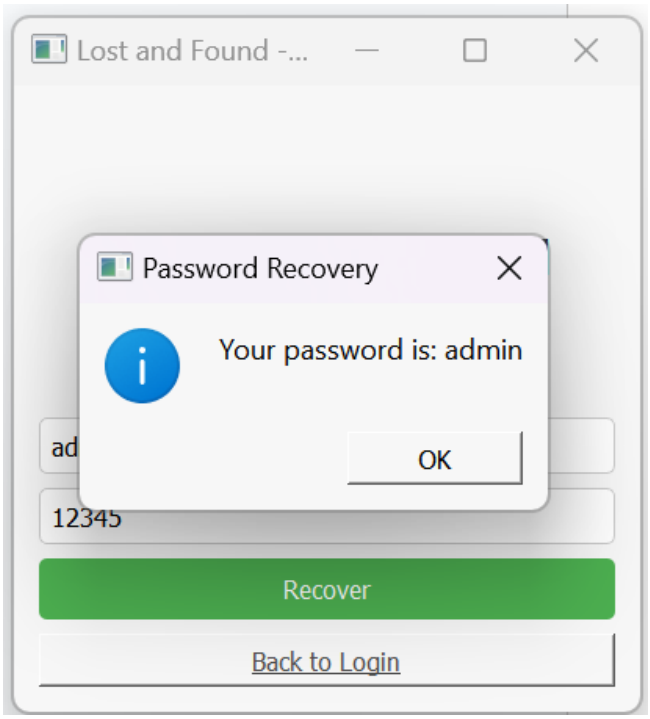

Figure 6.3: *If User Forgets Password*

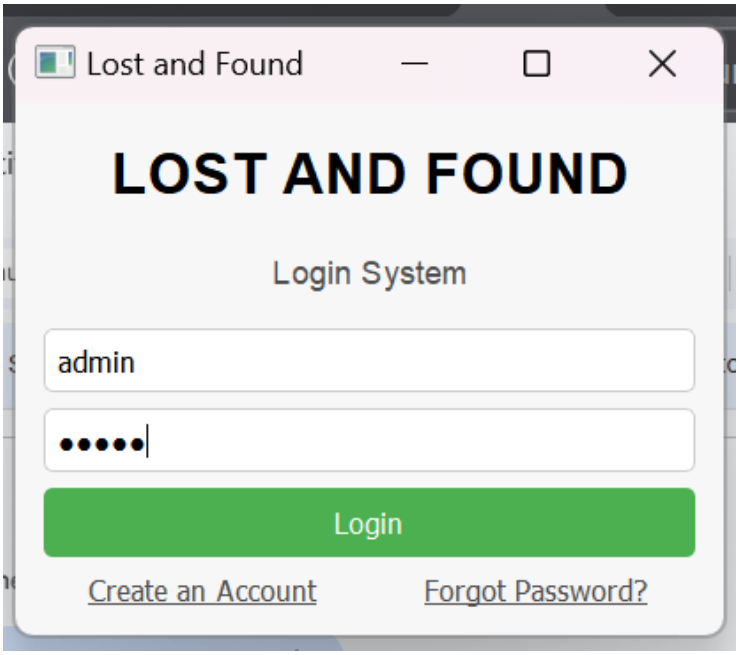

Figure 6.4: *User Login*

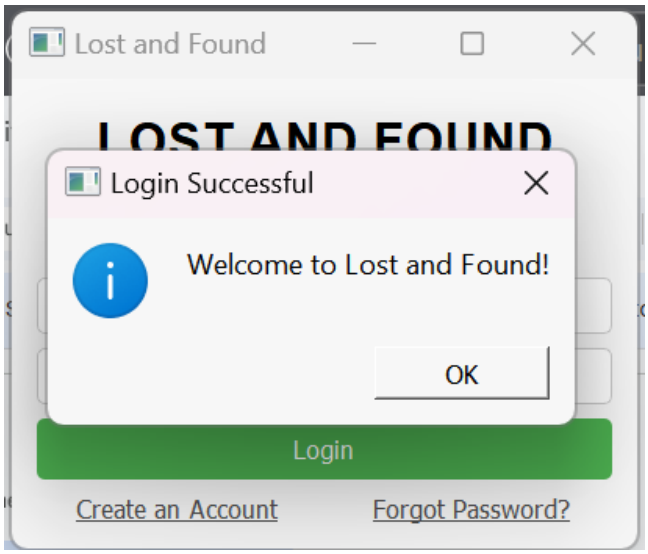

Figure 6.5: *Successful Login*

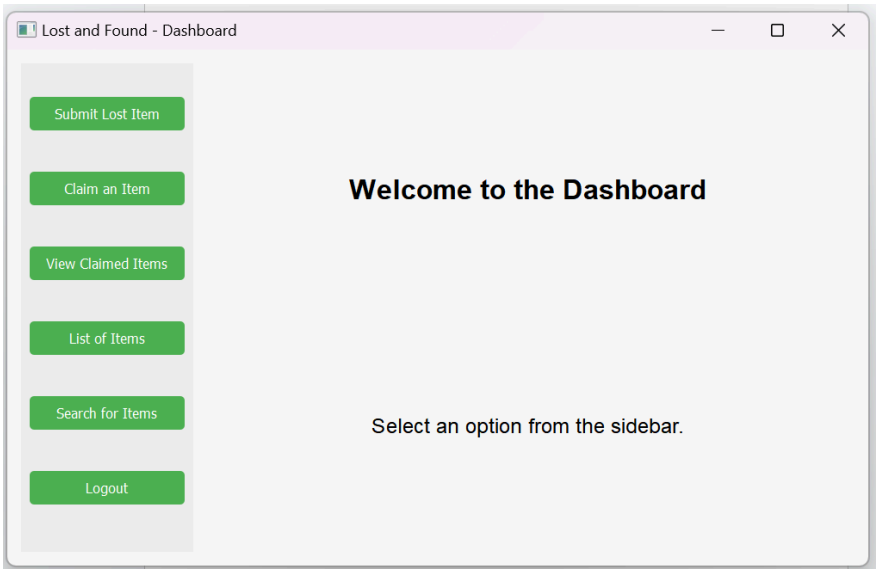

Figure 6.6: *The program redirect the*

User to the Dashboard Main Menu

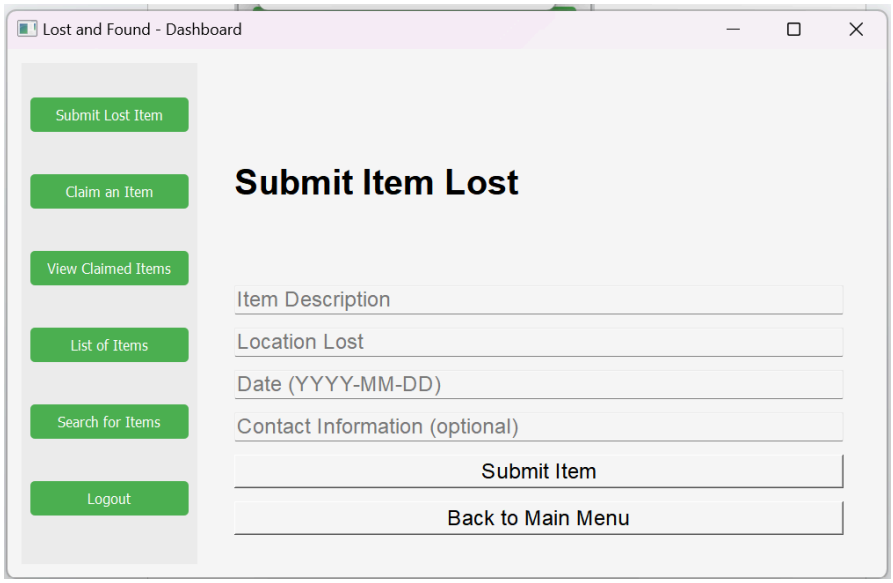

Figure 6.7: *User pressed Option 1(Submit Lost Item), and filled up the fields*

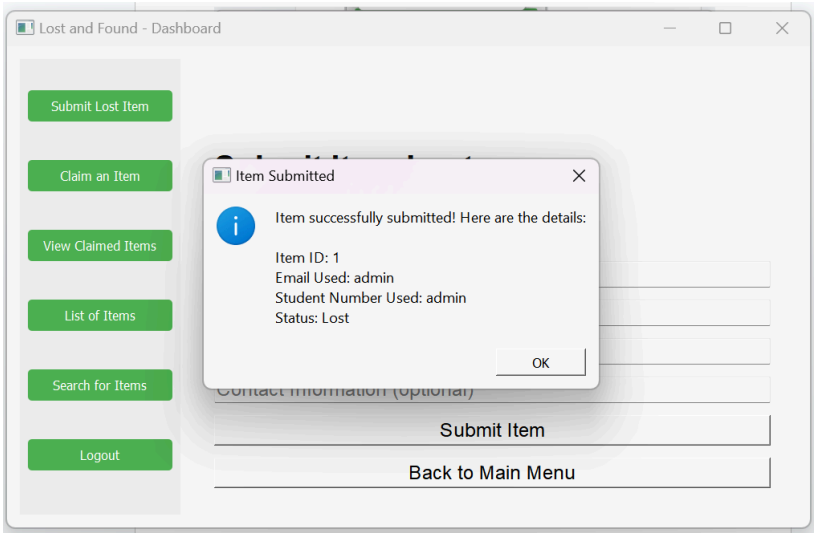

Figure 6.8: *User submits the item and a message pop up displays the items description*

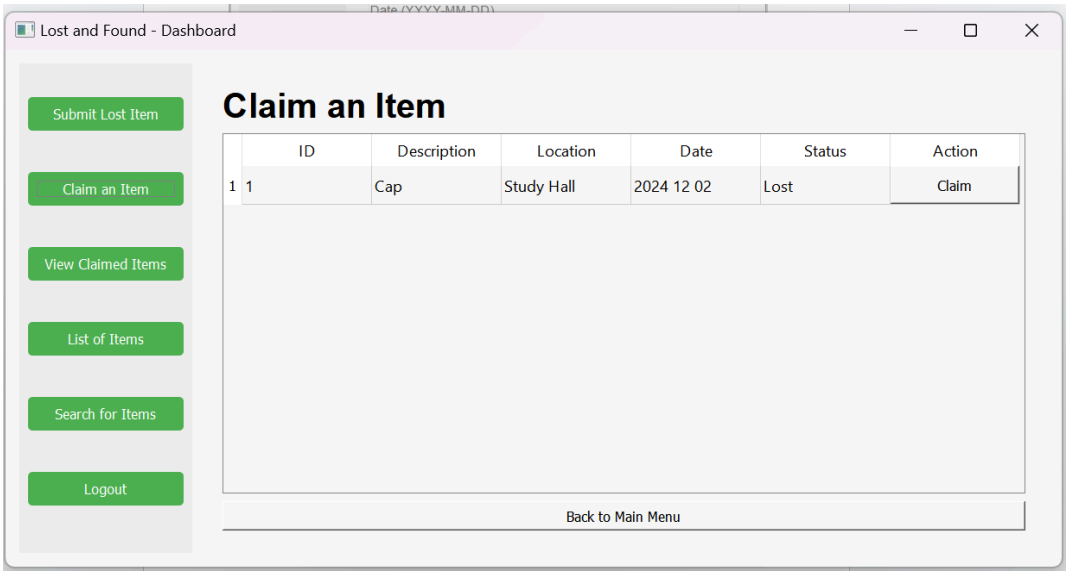

Figure 6.9: *User pressed Option 2(Claim an Item)*

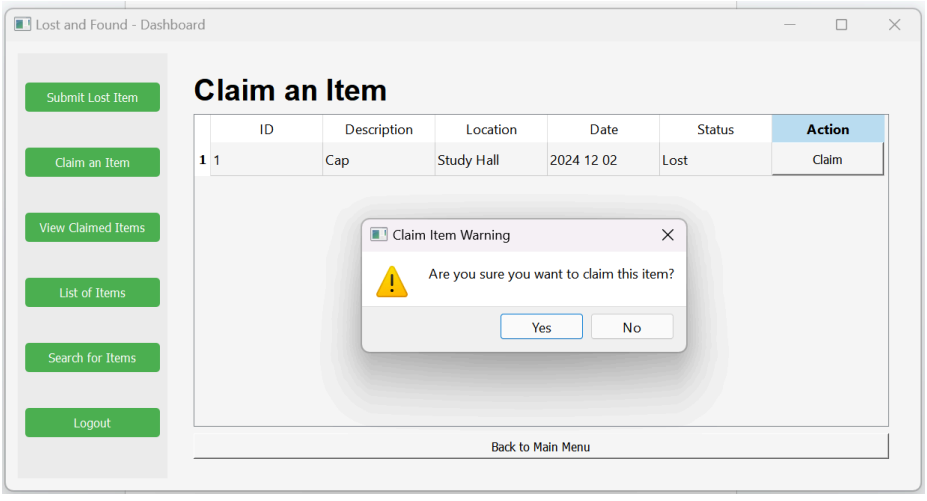

Figure 6.10: *User pressed Option 2(Claim an Item), and has the option to claim the item*

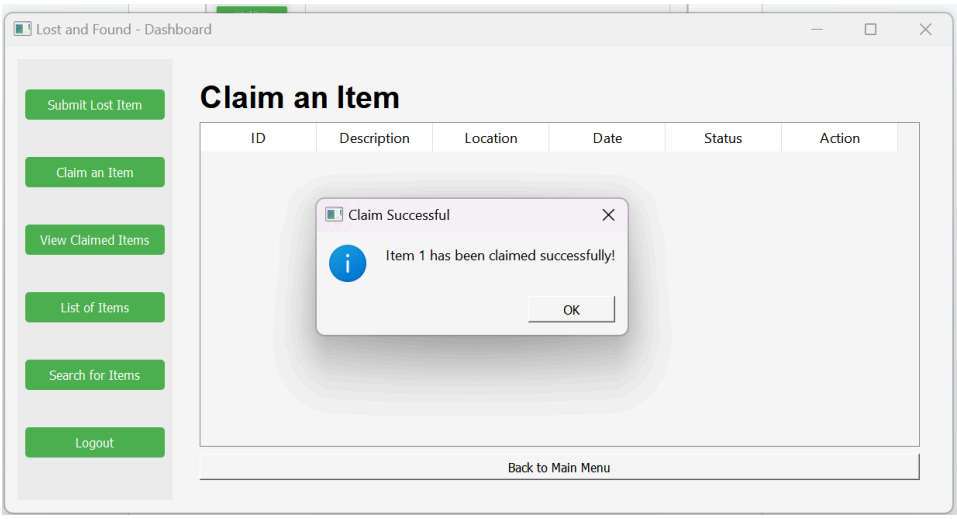

Figure 6.11: *If user Claimed the item a pop up message will appear*

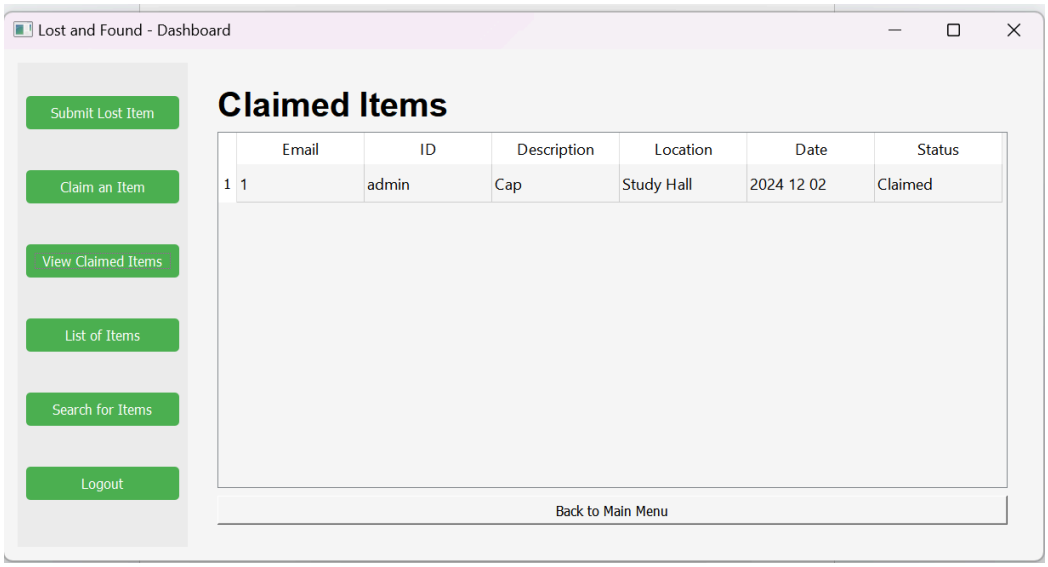

Figure 6.12: *User pressed Option 3(View Claimed items), and will see claimed items*

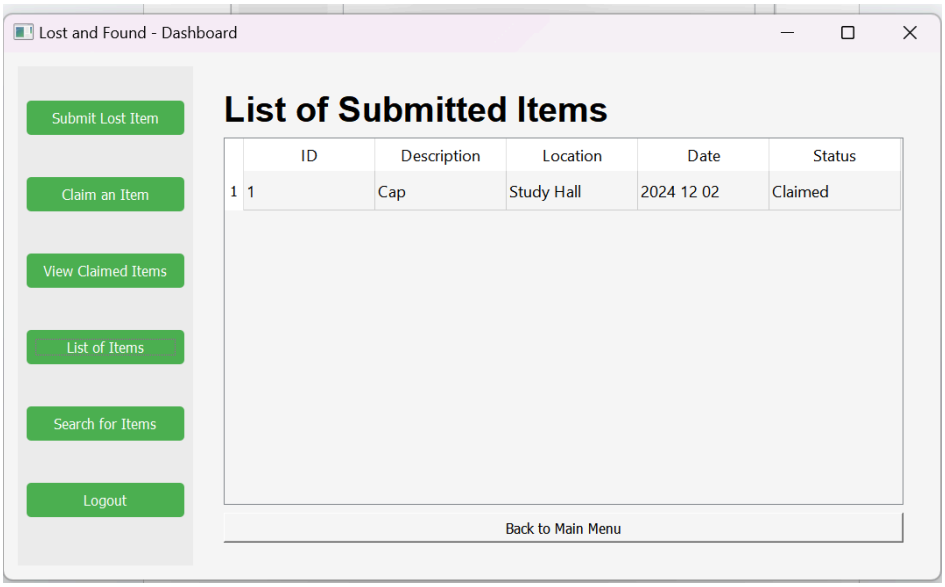

Figure 6.13: *User pressed Option 4 (List of Items), and will see claimed items*

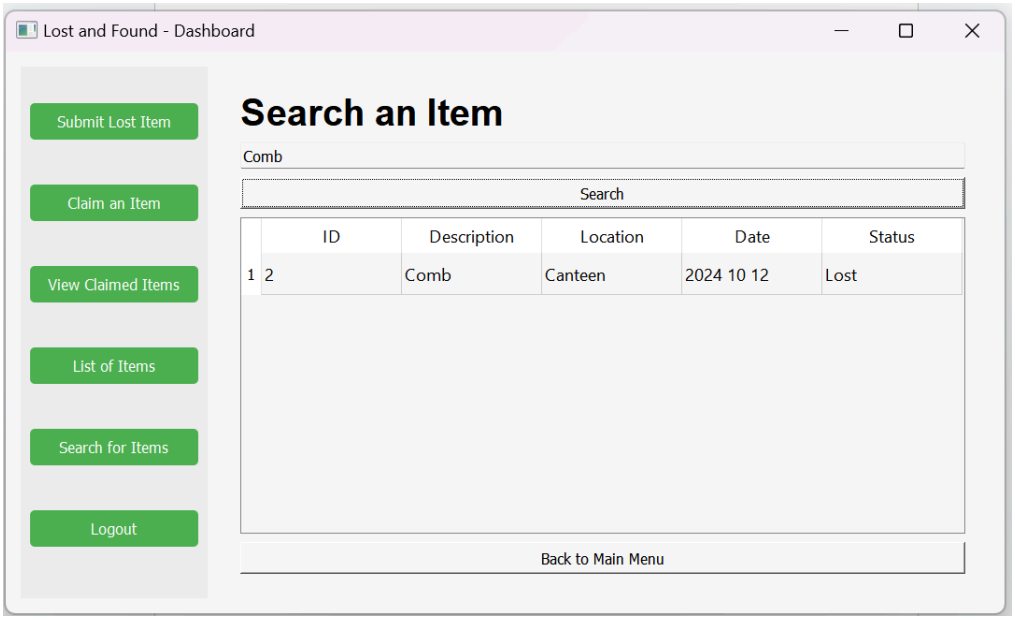

Figure 6.14: *User pressed Option 5 (Search an Item), and the user can search any details relating to the lost item*

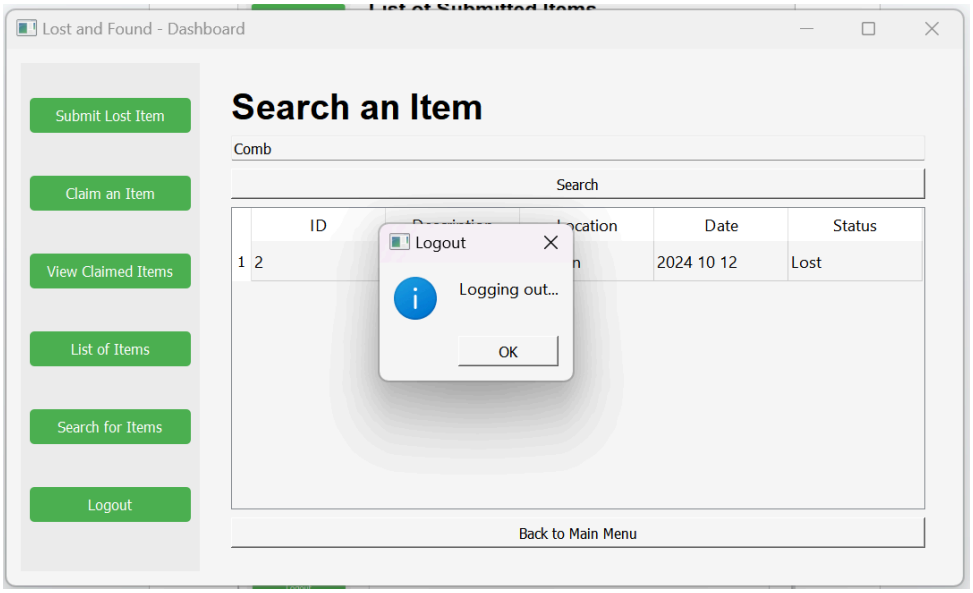

Figure 6.15: *User pressed Option 6 (Logout), and the user will go back to user window screen*

7. Conclusion

I. Conclusion

In conclusion, the objectives of Lost and Found Management System realize effective strategies of the intended learning outcomes. This project presents requirement to provide a cohesive visually oriented interface design for the end users in python and its user-friendly nature. The system improves the aid of item classification, statuses, and the introduction of item tracking processes to enable quick return of lost objects. The efficiency could also be increased by the features like a keyword search, layered search of text, date and type of an item, as well as better correlations. In spite of possible difficulties with the input of user information as well as the use of different devices, the system is created to resolve those issues and in fact create a viable option for the loss and find items.

II. Recommendations

- **Audit Logs**
  - Include an audit log feature to track all user activities, such as item submissions, edits, and

claims, for better system accountability and transparency.

- **Search Optimization**
  - Implement advanced search filters, such as fuzzy matching or auto-suggestions, to improve the accuracy and speed of item searches.
- **User Role Expansion**
  - Introduce additional user roles, such as administrators, who can manage reported items, verify claims, and oversee the database.
- **Affidavit Submission Requirement**
  - Introduce a feature requiring users to submit an affidavit of loss as proof before claiming an item. This enhances the system's reliability by ensuring the legitimacy of claims and providing additional accountability for item retrieval.

8. References

- Suchana, K., Alam, S. Md. E., Meem, A. T., Turjo, M. D., & Khan, M. M. (2021). Development of User-Friendly Web-Based Lost and Found System. Journal of Software Engineering and Applications, 14(10), 575–590. https://doi.org/10.4236/jsea.2021.1410034
- Python Linked List. (2023, July 12). GeeksforGeeks. https://www.geeksforgeeks.org/python-linked-list/
- PyQt5, M. F. L. updated G. started with. (2019, May 21). Create your first Python GUI with PyQt5 — A simple Hello world app. Python GUIs. https://www.pythonguis.com/tutorials/creating-your-first-pyqt-window/
- tutorialspoint.com. (2019). Python GUI Programming (Tkinter). Www.tutorialspoint.com. https://www.tutorialspoint.com/python/python_gui_programming.htm
- Mawa, J. (2021, December 17). How to build a GUI with PyQt. LogRocket Blog. https://blog.logrocket.com/how-to-build-gui-pyqt/