Airport Management System

By:
Brenda Luis
Chase Marquardt
Oscar Liu
Juan Loza
Kelby Maldonado

Table of Contents

Introduction

Every day there are thousands of flights going in and out of Houston. Airports are vital centers for domestic and global transportation that nearly everyone encounters once in their lifetime. Most of the time, people plan for going on vacation, business trips, relocating, or simply exploring. Needless to say, managing an airport is a very difficult task for software to maintain such as passenger services, workforce obligations, and security. This project is designed to integrate an embracive airport management software resolution to ease employee operations, enhance customer experience, and highlight safety demands.

As we reviewed some first time and frequent fliers, the goal was to understand their opinion on functionality of the airport and its technology. A couple of the complaints we noted down were the self check kiosk, the employees, luggage issues, and wait times for everything. Wait times are pretty much inevitable with the amount of people that fly daily,
however, with better trained staffing and updated software, we will assist in constructing a well organized airport apparatus.

Whether it's your first or hundredth time flying, everyone is in a rush to get through check-in, TSA, and then get to whichever gate they need. Usually, people purchase their tickets online and print them out beforehand or save it to their phone. This in turn costs the passenger/customer’s trust by making them feel as if they have to do all of the extra work themselves.

With the implementation of our software, we will be able to regain the confidence of customers and help them along the way tremendously. The amount of data running throughout the airport is mind boggling to put it lightly. Having a secure software that offers a certain amount of options to select from will help filter all that data into one place with little to no error.

On the performance side of our software, customers will be able to select from three options. The first option will allow for a passenger to check in with their flight and locate their gate in no time. The second option is for when customers land and just get off of the plane. We all know that everyone’s personal items are incredibly valuable to them, and want to help passengers be able to find their luggage at an instant once they deboard the plane. This effectively communicates to the passenger that the airport will do its best to help save time and stress from any possible mistakes being made. The final option will allow the passenger to ask for personal assistance from a skilled employee if needed.

On the employee-end of this software, their time clock will be a simple click of a button once their login is activated. It will also have a list for employees to select the job position that they are filling in for the day along with a list of requirements. Another important implementation of the software will allow for employee's to request for a manager on duty if need be.

Having a fully-functional team of hard working associates is a necessity when it comes to the airport. For day-to-day operations to flow smoothly, every associate must be up to date on the layout of the airport, how to use the software if someone is unable to do so, and have an open ear to any and all questions. With hard working staff geared towards maintaining a positive attitude, the airport will be able to create a welcoming environment magnifying passenger satisfaction.

On the security-end of this software, we will create a dedicated section for accessible personnel which facilitates notifications for help of any potential problems within the airport. It will designate specific sectors in which staff must monitor for the entirety of their shift. Additionally, personnel will be able to report an issue directly through the system, effectively contacting emergency services for response.

Threat detection is of the highest priority for airport security operations. Identifying and managing potential threats is a feature in our software that alerts our personnel to handle the situation. A universal security system integrates the ability to contact emergency services in the event of an injury or medical issue. Communication between sectors is a forethought in developing the security section of our software so additional help may be requested or supplied by demand.

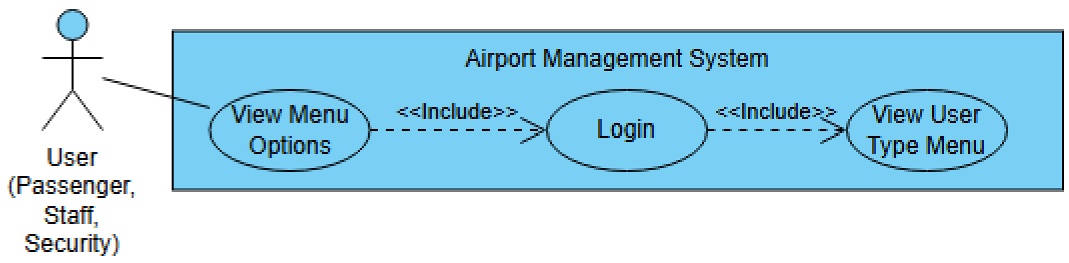
Overall, the airport functionality software we intend to create will optimize every situation in airport operations for passengers, in-house procedures, and security practices. We aim to focus on the efficiency and reliability of our software to transform the experience, workforce and provide a congenial environment for each party involved. With the execution of this software, we will be able to transform the airport from a once repugnant location to an ever evolving paradise.

Functional Requirements

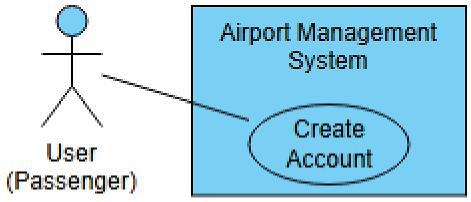
1. Software should allow users to log in using their username and password. The login system will authenticate users (passengers, staff, security personnel) through their credentials.
2. Software should give users (of passenger type) the option to create an account in the main menu.
3. Software should allow passengers to retrieve information regarding their flight ticket.
4. Software should allow passengers to retrieve information regarding their luggage.
5. Software should allow passengers to submit a support ticket.
6. Software should allow passengers to view information on their submitted support tickets.
7. Software should allow staff to view open support tickets.
8. Software should allow staff to view closed support tickets.
9. Software should allow staff to modify open support tickets.
10. Software should allow staff to view information about all flights.
11. Software should allow security personnel to log security threats.
12. Software should allow security personnel to view a list of security threats.
13. Software should allow security personnel to view information about all flights.

Use Cases

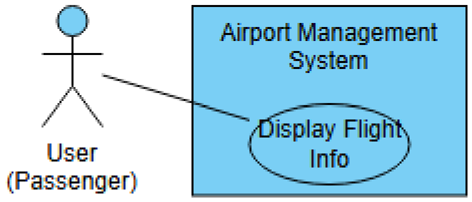
Use Case Name: Scenario: Summary:	User Login A user (passenger, staff, or security personnel) attempts to log into the system using their credentials (username and password). This use case describes the process of authenticating the user through their credentials to access the system based on their role (passenger, staff, or security).
Basic Flow:	1. The user navigates to the login screen. 2. The user enters their username and password. 3. The system authenticates the credentials. 4. If the credentials are valid, the user is granted access to the system and directed to their respective role-based dashboard. 5. If the credentials are invalid, an error message is displayed, and the user receives a new prompt.



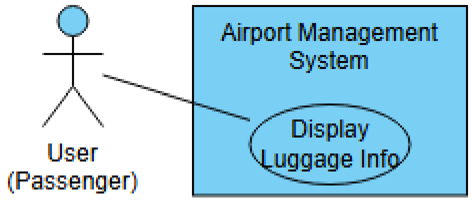
Use Case Name: Scenario: Summary:	User Create Account A passenger creates an account in the system This use case allows passengers to create an account so they can access their flight and luggage details.
Basic Flow:	1. The user selects the "Create Account" option on the main screen. 2. The user enters their required details. 3. The system validates the information. 4. If the data is valid, the system creates the account and confirms the creation. 5. The user is then redirected to the login screen to access the system.



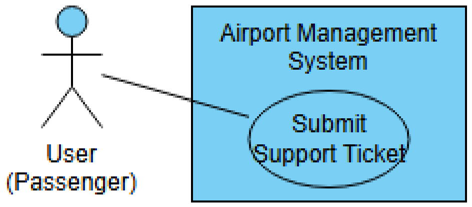
Use Case Name Scenario Summary	Retrieve Flight Information A passenger retrieves information about their flight. This use case enables passengers to access their flight-related information after logging into the system.
Basic Flow:	1. The passenger logs into their account. 2. The passenger navigates to the "Flight Information" section. 3. The system retrieves the relevant flight details for the logged-in passenger. 4. The system displays the flight information to the passenger.



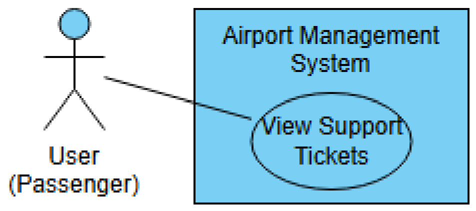
Use Case Name: Scenario: Summary:	Retrieve Luggage Information A passenger retrieves information about their luggage This use case allows passengers to access their luggage information
Basic Flow:	1. The passenger logs into their account. 2. The passenger navigates to the "Luggage Information" section. 3. The system retrieves the relevant luggage details for the logged-in passenger. 4. The system displays the luggage information to the passenger.



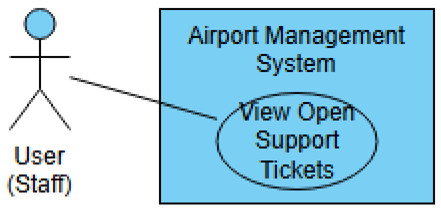
Use Case Name: Scenario: Summary:	Submit Support Ticket A passenger submits a support ticket to request help This use case allows passengers to submit support tickets for assistance from staff
Basic Flow:	1. The passenger logs into their account. 2. The passenger navigates to the "Support" section. 3. The passenger creates a new support ticket by describing the issue or request. 4. The system stores the support ticket in the database and assigns it a unique ID. 5. The system display successful ticket submission to the user.



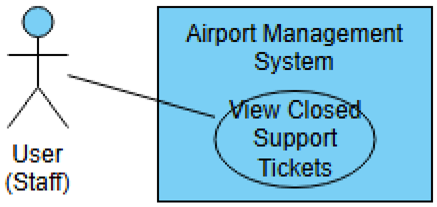
Use Case Name: Scenario: Summary:	View Submitted Support Tickets A passenger views the status of their previously submitted support tickets. This use case allows passengers to view the status (open/closed) of their previously submitted support tickets.
Basic Flow:	1. The passenger logs into their account. 2. The passenger navigates to the "View Support Tickets" section. 3. The system displays a list of the passenger's previously submitted support tickets.



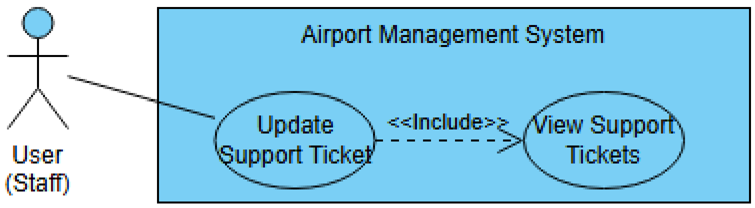
Use Case Name: Scenario: Summary:	View Open Support Tickets A staff member reviews open support tickets submitted by passengers. This use case allows staff to view open support tickets submitted by passengers.
Basic Flow:	1. The staff logs into their account. 2. The staff navigates to the "View Support Tickets" section. 3. The system displays a list of open support tickets.



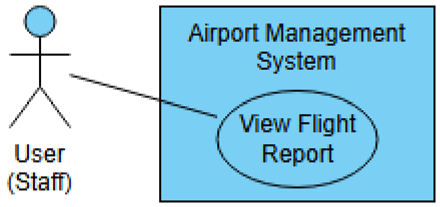
Use Case Name: Scenario: Summary	View Closed Support Tickets A staff member reviews closed support tickets that have been resolved or completed. This use case allows staff to view support tickets that have been marked as closed.
Basic Flow:	1. The staff logs into their account. 2. The staff navigates to the "Closed Support Tickets" section. 3. The system displays a list of closed support tickets.



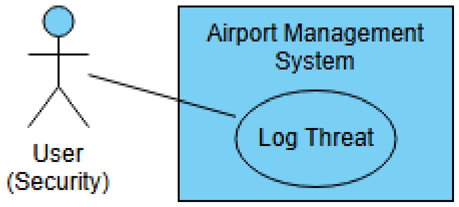
Use Case Name: Scenario: Summary:	Modify Open Support Ticket A staff member modifies an open support ticket This use case allows staff to modify open support tickets
Basic Flow:	1. The staff logs into their account. 2. The staff navigates to the "Update Support Tickets" section. 3. The system calls method to display all Support Tickets 4. The staff selects a ticket to modify. 5. The staff updates the ticket’s status 6. The system saves the updated ticket



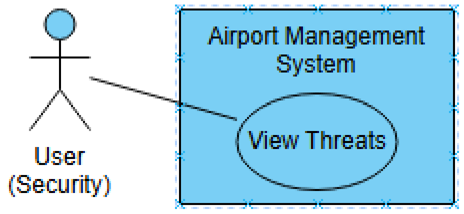
Use Case Name: Scenario: Summary:	Staff View Report On Flights A staff member views information about all available flights in the system. This use case allows staff to access all flight-related information
Basic Flow:	1. The staff logs into their account. 2. The staff navigates to the "View Flight Report" section. 3. The system retrieves all available flight data. 4. The system displays the flight information to the staff.



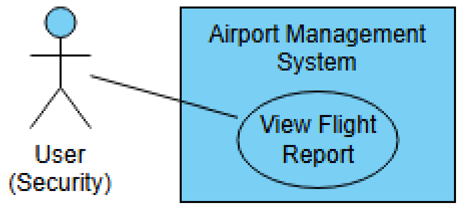
Use Case Name: Scenario: Summary:	Log Security Threat A security personnel logs a potential or active security threat in the system. This use case allows security personnel to log a security threat into the system
Basic Flow:	1. The security personnel logs into their account. 2. The security personnel navigates to the "Log Threats" section. 3. The security personnel enters details of the threat. 4. The system stores the threat in the database and assigns a unique ID to the entry.



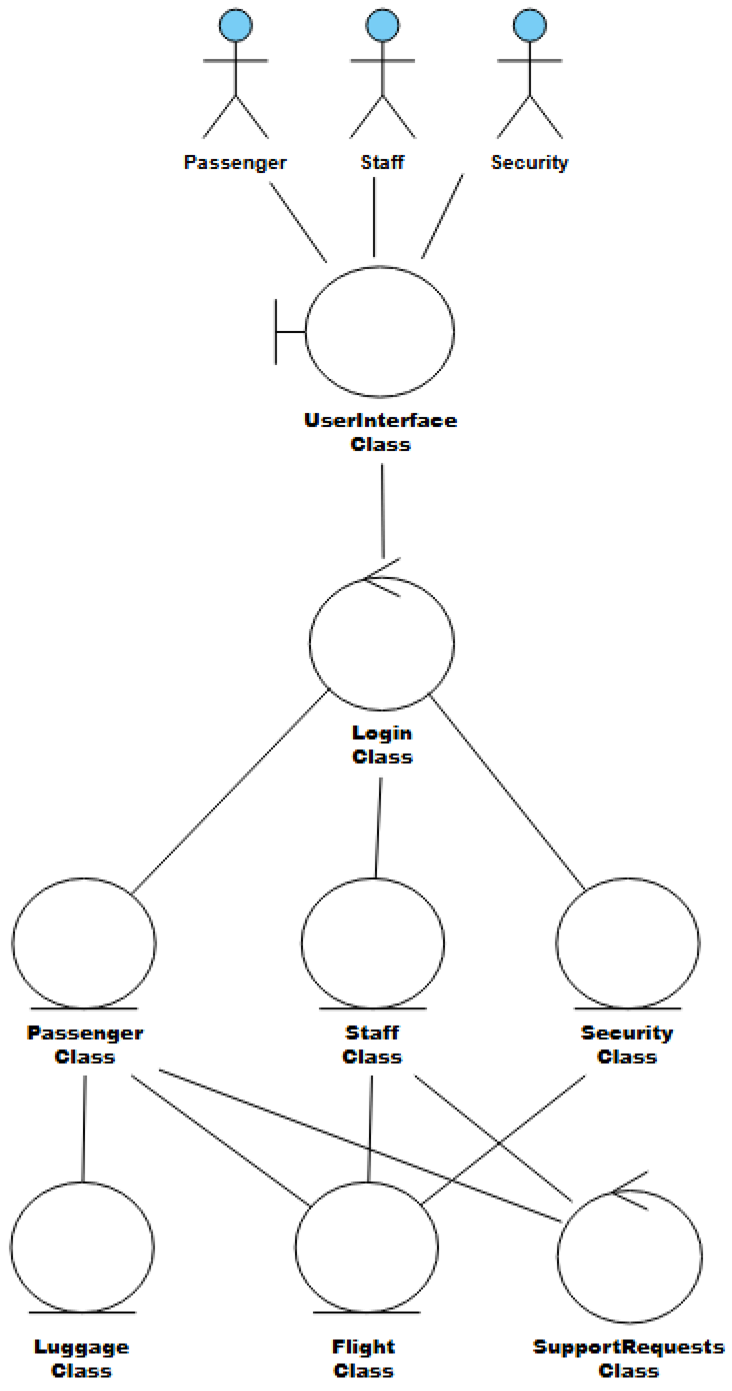
Use Case Name: Scenario: Summary:	View Security Threats List A security personnel views a list of all logged security threats. This use case allows security personnel to access and review all security threats that have been logged in the system.
Basic Flow:	1. The security personnel logs into their account. 2. The security personnel navigates to the "View Security Threats" section. 3. The system displays all recorded security threats.



Use Case Name: Scenario: Summary:	Security View Report On Flights Staff or Security Personnel views information about all available flights in the system. This use case allows staff or security personnel to view information about all flights.
Basic Flow:	1. The security personnel logs into their account. 2. The security personnel navigates to the "View Flight Report" section. 3. The system retrieves all available flight data. 4. The system displays the flight information to the staff member/security personnel.



Class Diagram



<<Boundary Class>>
UserInterface Class

+ displayLoginScreen() : void
+ displaySuccessfulLogin() : void
+ displayUnsuccessfulLogin() : void
+ getPasswordInput() : string
+ getUsernameInput() : string

<<Control Class>>
Login Class

- con : Connection*
- first_name : string
- password : string
- user_id : string
- user_type : string
- username :string

+ ~Login
+ connectToDatabase() : void
+ createAccount(a : string, b : string, c : string, d : string, e : string, f : string, g : string) : bool
+ disconnectDatabase() : void
+ getConnection() : Connection*
+ getUserFirstName() : string
+ getUserID () : int
+ getUserType() : string
+ handleUserType() : void
+ login(username : string, password : string) :bool
+ Login()
+ setUserFirstName(n : string) : void
+ setUserID(id : int) : void
+ setUserType(type : string) : void

<<Control Class>> **Passenger Class**
- choice : int - con : Connection* - description : string - first_name : string - pass_id : int - password : string - user_id : int - user_type : string - username : string
+ ~Passenger() + checkFlightInformation() : void + displayOptions() : void + getChoice() : int + getConnection : Connection* + getIntInput() : int + handleChoice(c : int) : void + luggageInformation() : void + Passenger(eCon : Connection*, u : string) + requestSupport : void + setChoice(c : int) : void + viewSupportRequests() : void

<<Control Class>> **Staff Class**
- choice : int - con : Connection* - username : string
+ ~Staff() + displayOptions() : void + getChoice() : int + getConnection() : Connection* + getIntInput() : int + handleChoice(c : int) : void + setChoice(c : int) : void + Staff(eCon : Connection*, u : string) + updateOpenTickets() : void + viewClosedTickets() : void + viewFlightInformationReport() : void + viewOpenTickets() : void

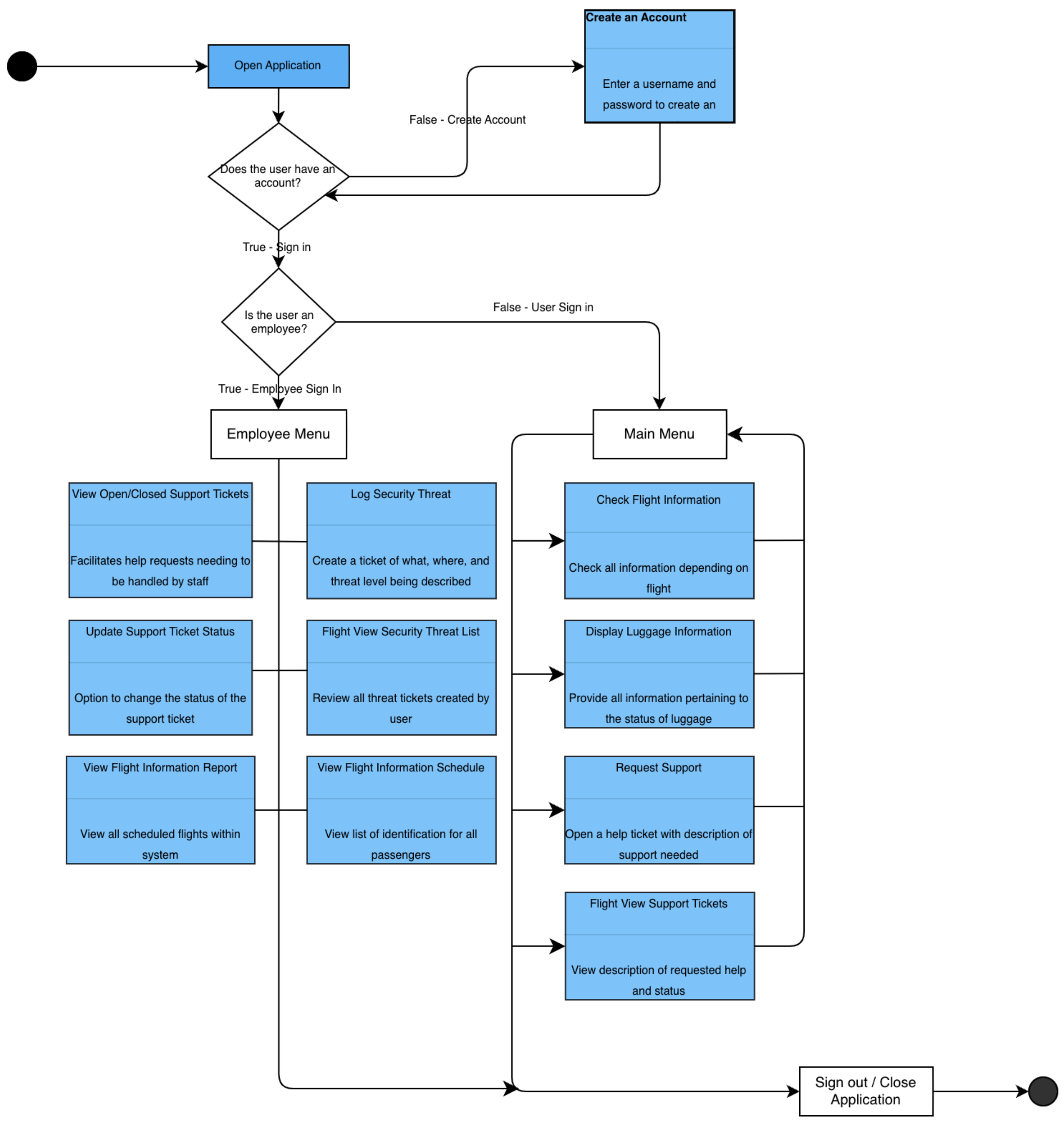
<<Control Class>> **Security Class**
- choice : int - con : Connection* - username : string
+ displayOptions() : void + getChoice() : int + getConnection() : Connection* + getIntInput() : int + handleChoice(c : int) : void + logSecurityThreat() : void + Security(eCon : Connection*, u : string) + setChoice(c : int) : void + viewFlightInformation() : void + viewThreatList() : void

<<Entity Class>> **Luggage Class**
- con : Connection* - flight_id : int - location : string - luggage_id : int - passenger_id : int - user_id : int - username : string
+ displayInfo() : void + Luggage(eCon : Connection*, u : string)

<<Entity Class>> **Flight Class**
- arrival_time : string - capacity : int - con : Connection* - departure_time : string - destination_airport_id : int - destination_airport_name : string - flight_id : int - flight_number : string - origin_airport_id : int - origin_airport_name : string - passenger_id : int - user_id : int - username : string
+ displayFlightReport() : void + displayInfo() : void + Flight(eCon : Connection*, u : string)

<<Entity Class>> **SupportRequests Class**
- con : Connection* - pass_id : int - req_id : int - supportRequestDescription : string - supportRequestExis : bool - username : string
+ getConnection() : Connection* + getIntInput() : void + insertRequest() : void + setPassengerId() : void + SupportRequests(eCon : Connection*, u : string) + updateRequestStatus() : void + viewAllClosedTickets : void + viewAllOpenTickets : void + viewUserRequests() : void

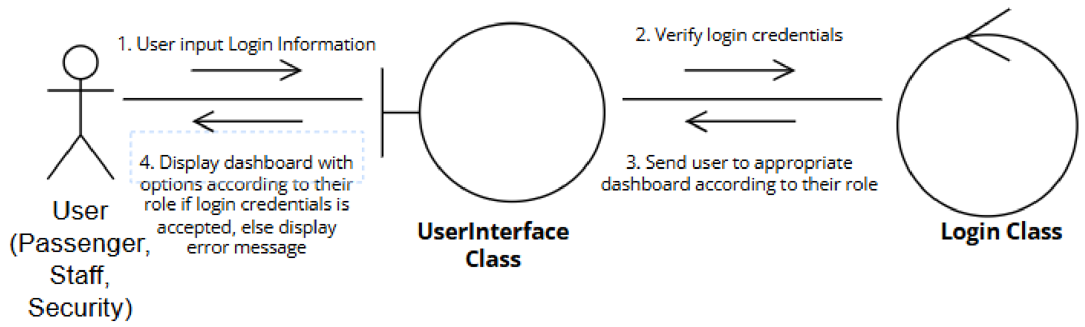
Statechart


Open Application
Create an Account
Enter a username and password to create an
False - Create Account
Does the user have an account?
True - Sign in
Is the user an employee?
False - User Sign in
True - Employee Sign In
Employee Menu
Main Menu
View Open/Closed Support Tickets
Facilitates help requests needing to be handled by staff
Log Security Threat
Create a ticket of what, where, and threat level being described
Update Support Ticket Status
Option to change the status of the support ticket
Flight View Security Threat List
Review all threat tickets created by user
View Flight Information Report
View all scheduled flights within system
View Flight Information Schedule
View list of identification for all passengers
Check Flight Information
Check all information depending on flight
Display Luggage Information
Provide all information pertaining to the status of luggage
Request Support
Open a help ticket with description of support needed
Flight View Support Tickets
View description of requested help and status
Sign out / Close Application

Use Case Realization

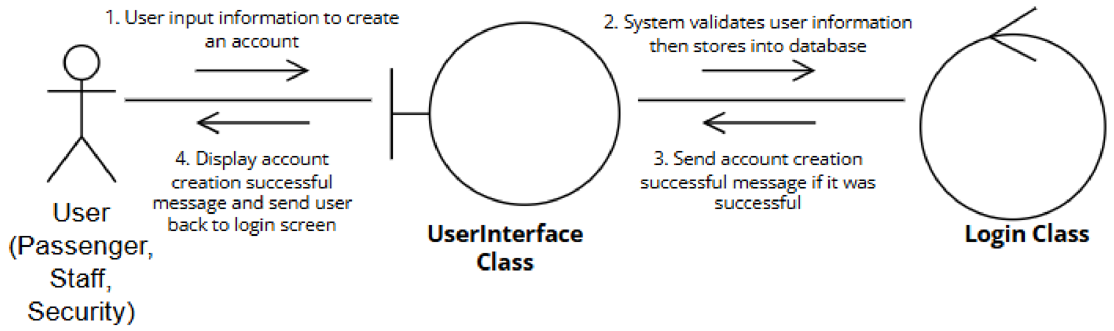
A user (passenger, staff, or security personnel) attempts to log into the system using their credentials (username and password).

1. The user navigates to the login screen.
2. The user enters their username and password.
3. The system authenticates the credentials.
4. If the credentials are valid, the user is granted access to the system and directed to their respective role-based dashboard.
5. If the credentials are invalid, an error message is displayed, and the user receives a new prompt.


1. User input Login Information
4. Display dashboard with options according to their role if login credentials is accepted, else display error message
User (Passenger, Staff, Security)
2. Verify login credentials
3. Send user to appropriate dashboard according to their role
UserInterface Class
Login Class

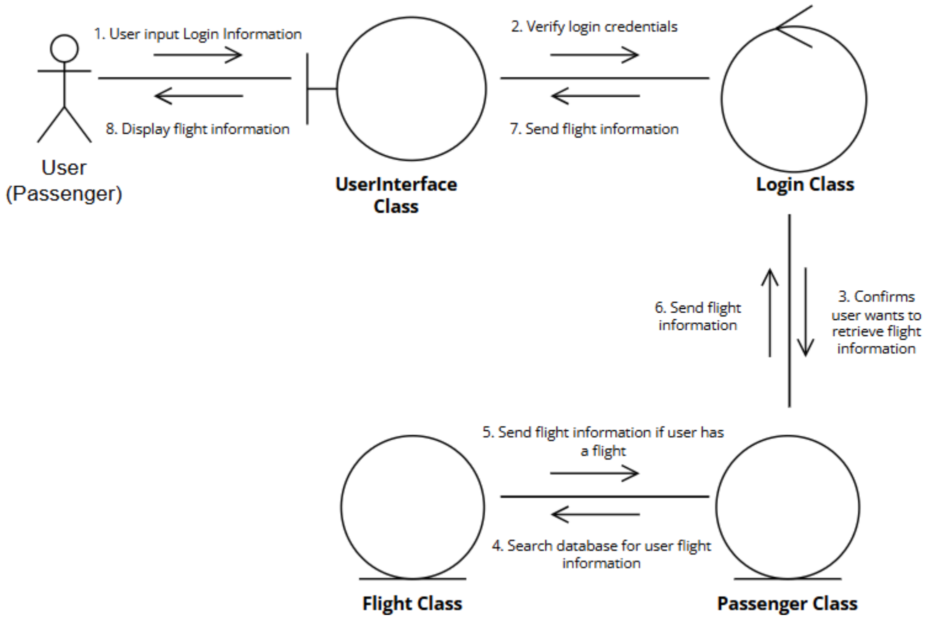
A passenger creates an account in the system

1. The user selects the "Create Account" option on the main screen.
2. The user enters their required details.
3. The system validates the information.
4. If the data is valid, the system creates the account and confirms the creation.
5. The user is then redirected to the login screen to access the system.


1. User input information to create an account
4. Display account creation successful message and send user back to login screen
User (Passenger, Staff, Security)
2. System validates user information then stores into database
3. Send account creation successful message if it was successful
UserInterface Class
Login Class

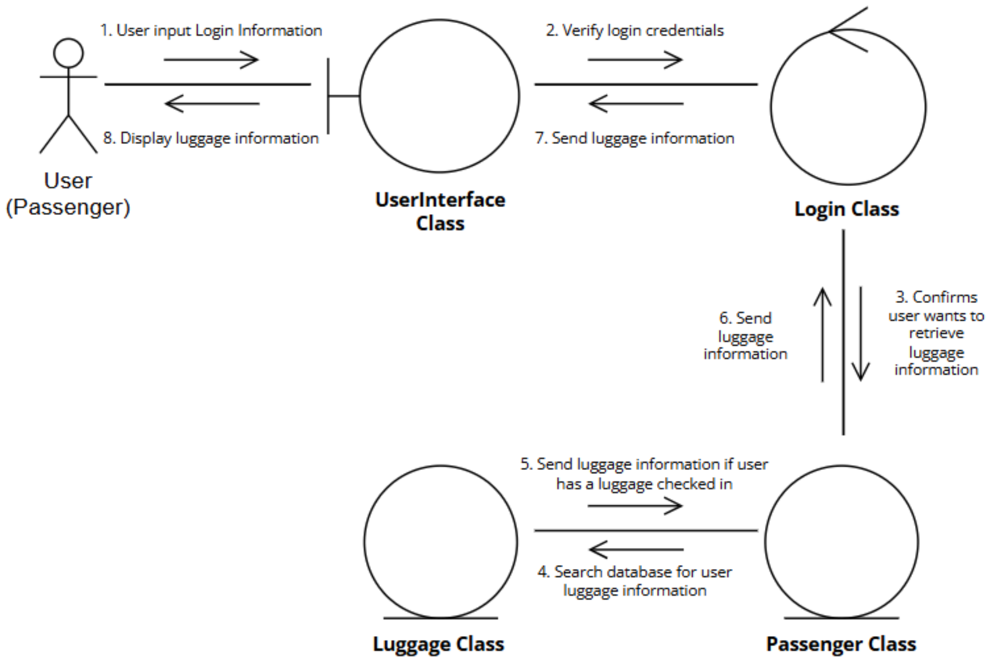
A passenger retrieves information about their flight.

1. The passenger logs into their account.
2. The passenger navigates to the "Flight Information" section.
3. The system retrieves the relevant flight details for the logged-in passenger.
4. The system displays the flight information to the passenger.


1. User input Login Information
8. Display flight information
User (Passenger)
UserInterface Class
2. Verify login credentials
7. Send flight information
Login Class
6. Send flight information
3. Confirms user wants to retrieve flight information
5. Send flight information if user has a flight
4. Search database for user flight information
Flight Class
Passenger Class

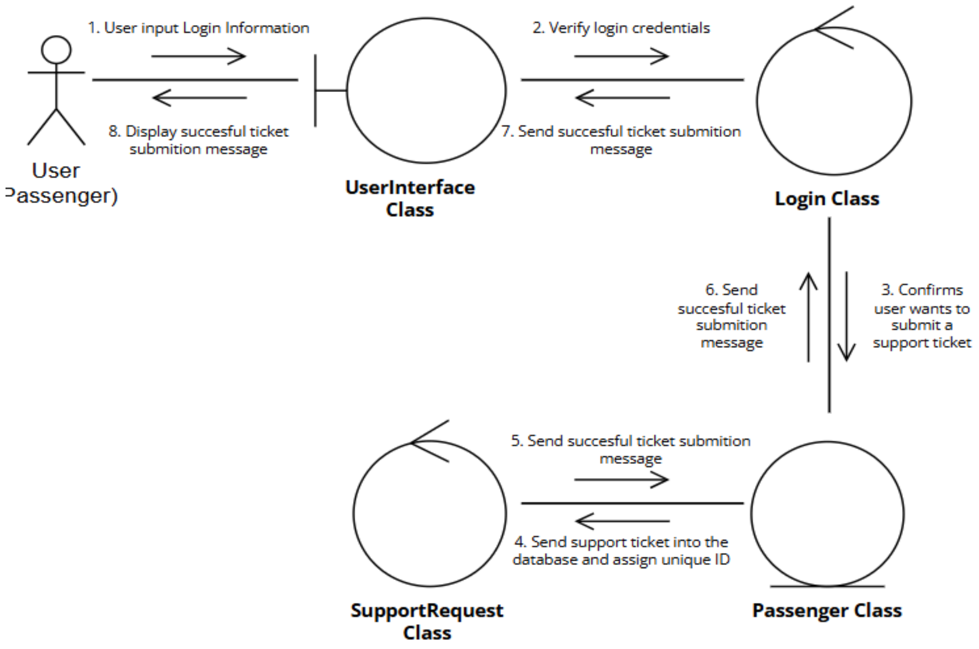
A passenger retrieves information about their luggage

1. The passenger logs into their account.
2. The passenger navigates to the "Luggage Information" section.
3. The system retrieves the relevant luggage details for the logged-in passenger.
4. The system displays the luggage information to the passenger.


1. User input Login Information
8. Display luggage information
User (Passenger)
UserInterface Class
2. Verify login credentials
7. Send luggage information
Login Class
6. Send luggage information
3. Confirms user wants to retrieve luggage information
5. Send luggage information if user has a luggage checked in
4. Search database for user luggage information
Luggage Class
Passenger Class

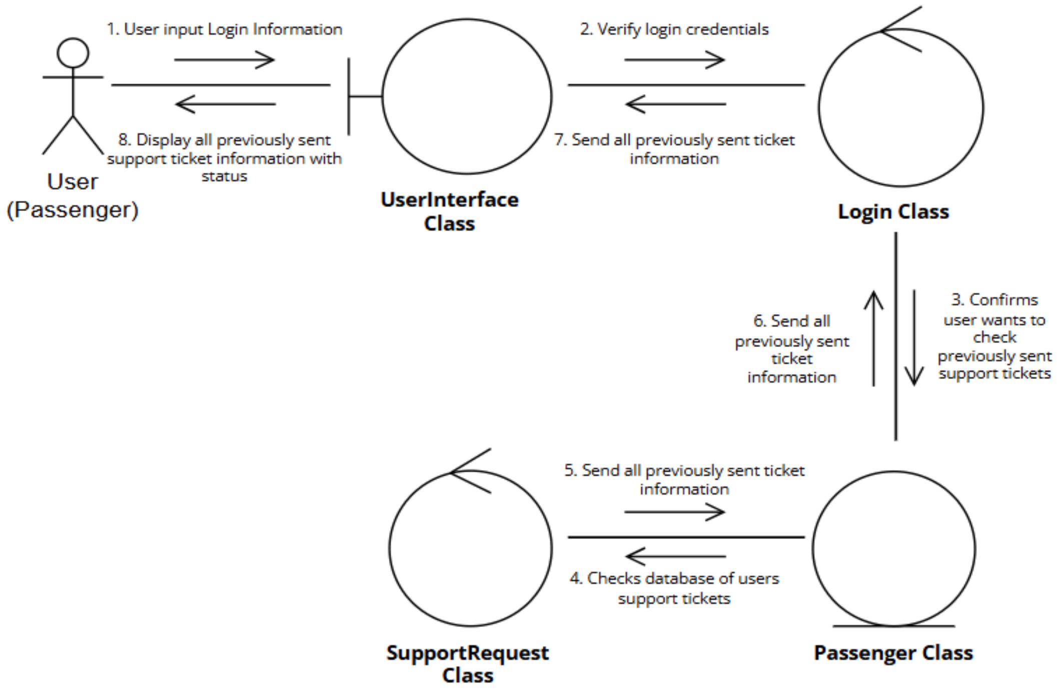
A passenger submits a support ticket to request help

1. The passenger logs into their account.
2. The passenger navigates to the "Support" section.
3. The passenger creates a new support ticket by describing the issue or request.
4. The system stores the support ticket in the database and assigns it a unique ID.
5. The system displays successful ticket submission to the user.


1. User input Login Information
2. Verify login credentials
8. Display succesful ticket submition message
7. Send succesful ticket submition message
User
Passenger)
UserInterface Class
Login Class
6. Send succesful ticket submition message
3. Confirms user wants to submit a support ticket
5. Send succesful ticket submition message
4. Send support ticket into the database and assign unique ID
SupportRequest Class
Passenger Class

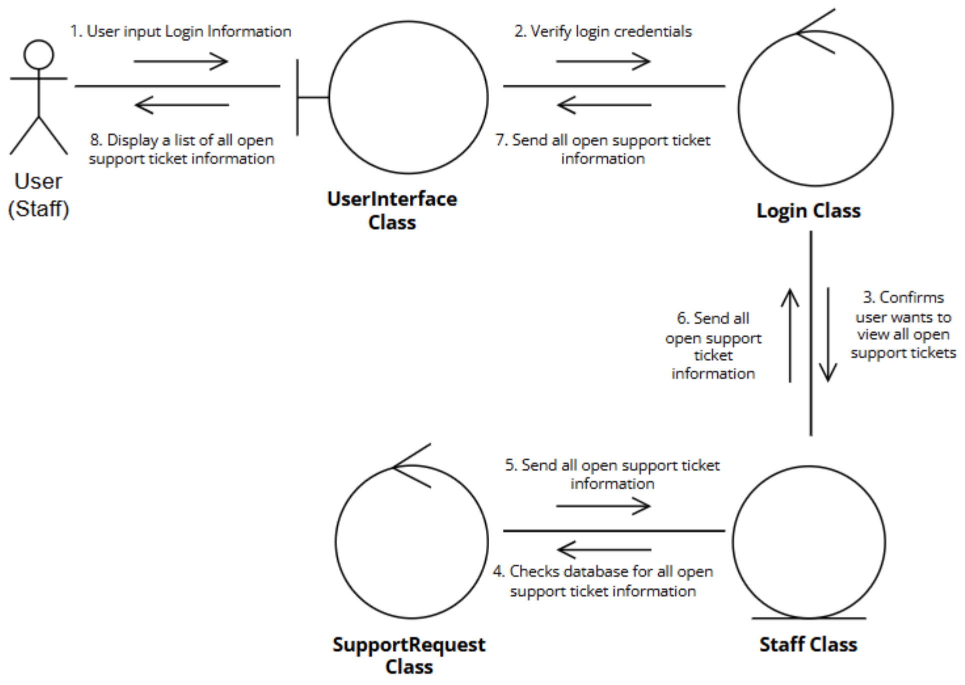
A passenger views the status of their previously submitted support tickets.

1. The passenger logs into their account.
2. The passenger navigates to the "View Support Tickets" section.
3. The system displays a list of the passenger's previously submitted support tickets.


1. User input Login Information
2. Verify login credentials
8. Display all previously sent support ticket information with status
7. Send all previously sent ticket information
User
(Passenger)
UserInterface Class
Login Class
6. Send all previously sent ticket information
3. Confirms user wants to check previously sent support tickets
5. Send all previously sent ticket information
4. Checks database of users support tickets
SupportRequest Class
Passenger Class

A staff member reviews open support tickets submitted by passengers.

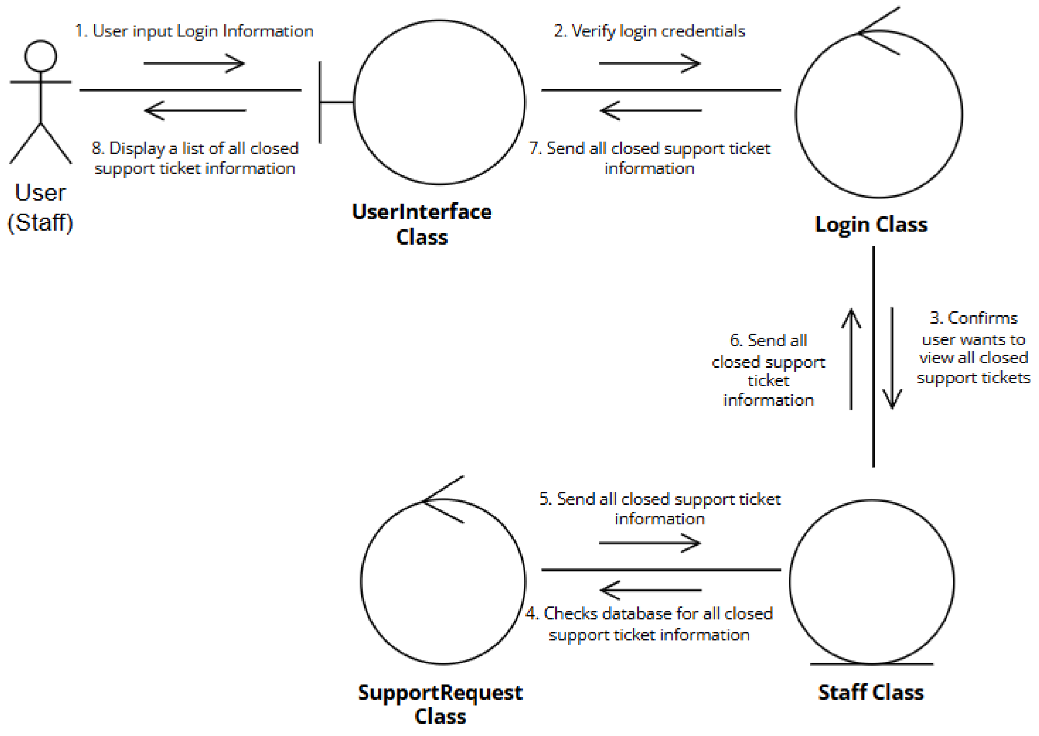
1. The staff logs into their account.
2. The staff navigates to the "View Support Tickets" section.
3. The system displays a list of open support tickets.



1. User input Login Information
8. Display a list of all open support ticket information
User (Staff)
UserInterface Class
2. Verify login credentials
7. Send all open support ticket information
Login Class
6. Send all open support ticket information
3. Confirms user wants to view all open support tickets
5. Send all open support ticket information
4. Checks database for all open support ticket information
SupportRequest Class
Staff Class

A staff member reviews closed support tickets that have been resolved or completed.

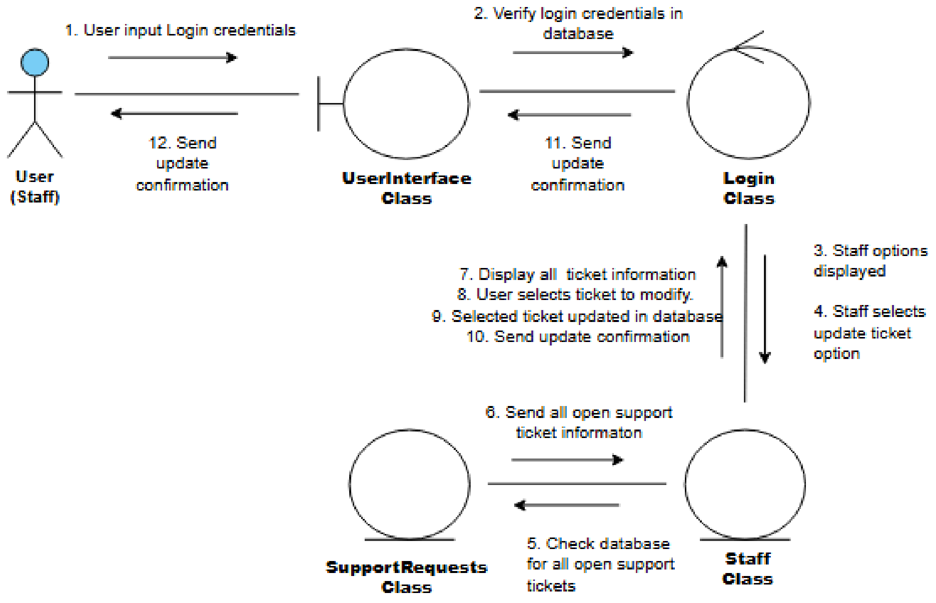
1. The staff logs into their account.
2. The staff navigates to the "Closed Support Tickets" section.
3. The system displays a list of closed support tickets.



1. User input Login Information
8. Display a list of all closed support ticket information
User (Staff)
UserInterface Class
2. Verify login credentials
7. Send all closed support ticket information
Login Class
6. Send all closed support ticket information
3. Confirms user wants to view all closed support tickets
5. Send all closed support ticket information
4. Checks database for all closed support ticket information
SupportRequest Class
Staff Class

A staff member modifies an open support ticket

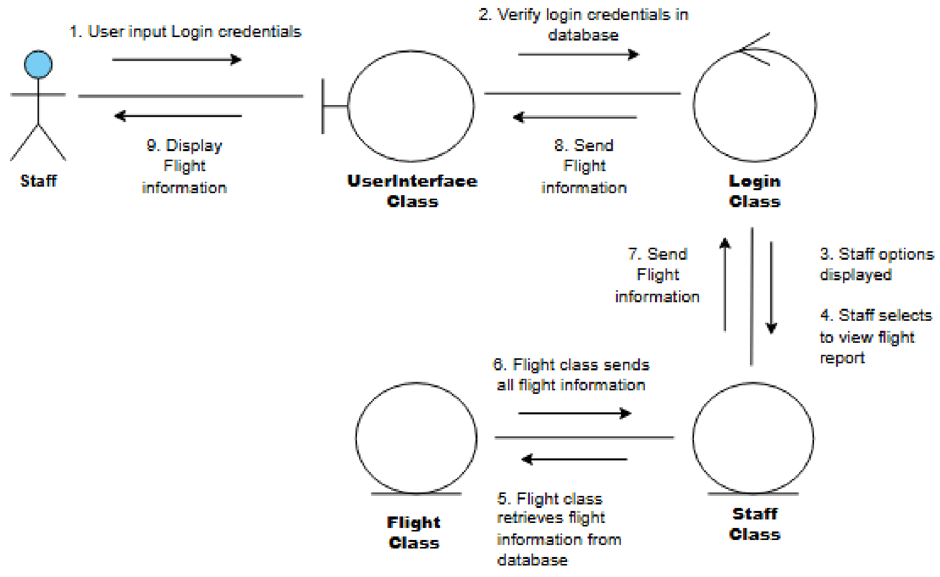
1. The staff logs into their account.
2. The staff navigates to the "Update Support Tickets" section.
3. The system calls method to display all Support Tickets
4. The staff selects a ticket to modify.
5. The staff updates the ticket's status
6. The system saves the updated ticket



1. User input Login credentials
12. Send update confirmation
User (Staff)
UserInterface Class
2. Verify login credentials in database
11. Send update confirmation
Login Class
7. Display all ticket information
8. User selects ticket to modify.
9. Selected ticket updated in database
10. Send update confirmation
3. Staff options displayed
4. Staff selects update ticket option
6. Send all open support ticket informaton
5. Check database for all open support tickets
SupportRequests Class
Staff Class

A staff member views information about all available flights in the system.

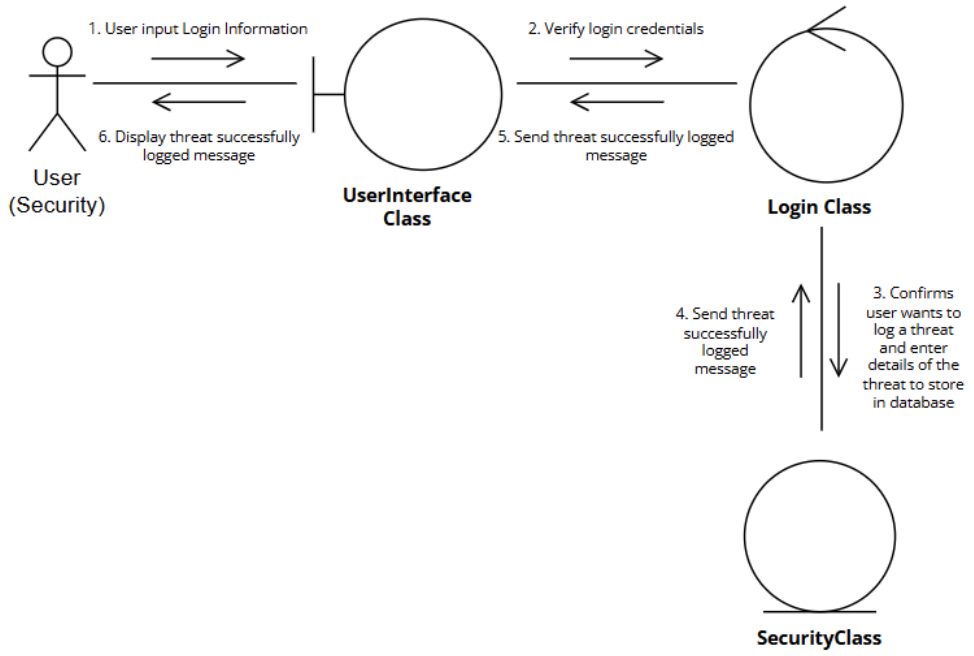
1. The staff logs into their account.
2. The staff navigates to the "View Flight Report" section.
3. The system retrieves all available flight data.
4. The system displays the flight information to the staff.



1. User input Login credentials
9. Display Flight information
Staff
UserInterface Class
2. Verify login credentials in database
8. Send Flight information
Login Class
7. Send Flight information
3. Staff options displayed
4. Staff selects to view flight report
6. Flight class sends all flight information
5. Flight class retrieves flight information from database
Flight Class
Staff Class

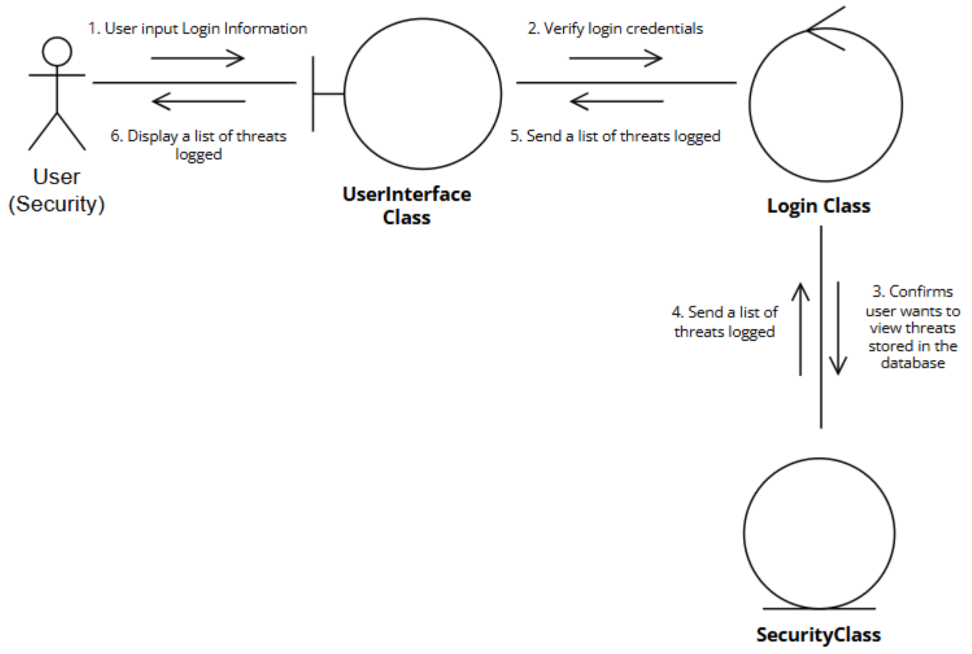
A security personnel logs a potential or active security threat in the system.

1. The security personnel logs into their account.
2. The security personnel navigates to the "Log Threats" section.
3. The security personnel enters details of the threat.
4. The system stores the threat in the database and assigns a unique ID to the entry.


1. User input Login Information
2. Verify login credentials
User (Security)
6. Display threat successfully logged message
UserInterface Class
5. Send threat successfully logged message
Login Class
4. Send threat successfully logged message
3. Confirms user wants to log a threat and enter details of the threat to store in database
SecurityClass

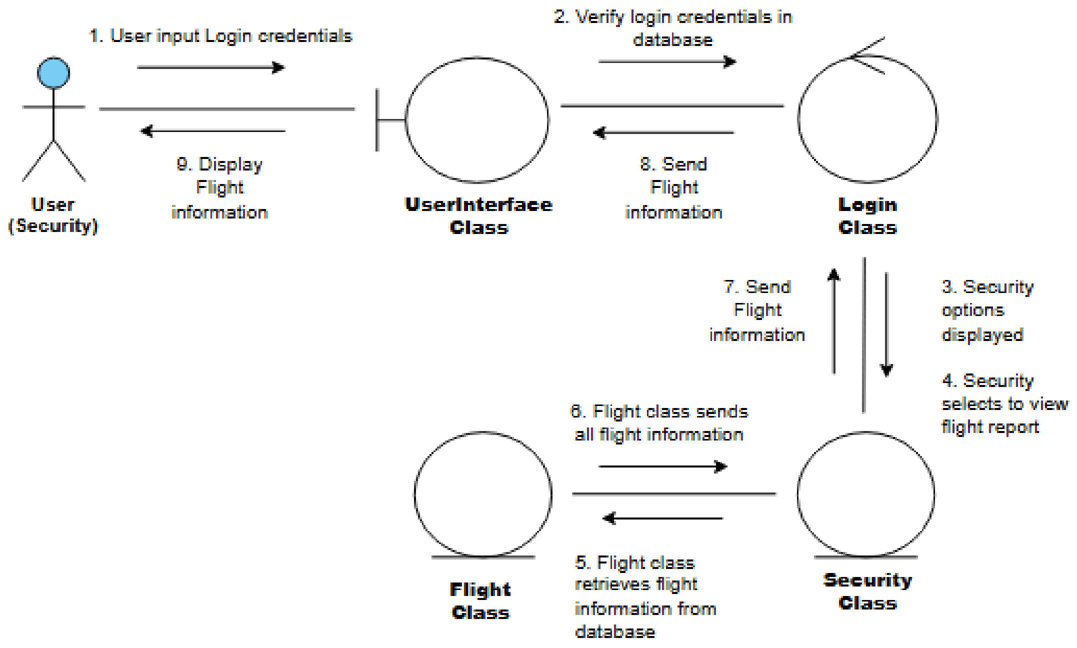
A security personnel views a list of all logged security threats.

1. The security personnel logs into their account.
2. The security personnel navigates to the "View Security Threats" section.
3. The system displays all recorded security threats.


1. User input Login Information
2. Verify login credentials
User (Security)
6. Display a list of threats logged
UserInterface Class
5. Send a list of threats logged
Login Class
4. Send a list of threats logged
3. Confirms user wants to view threats stored in the database
SecurityClass

Security Personnel views information about all available flights in the system.

1. The security personnel logs into their account.
2. The security personnel navigates to the "View Flight Report" section.
3. The system retrieves all available flight data.
4. The system displays the flight information to the staff member/security personnel.


1. User input Login credentials
2. Verify login credentials in database
User (Security)
9. Display Flight information
UserInterface Class
8. Send Flight information
Login Class
7. Send Flight information
3. Security options displayed
4. Security selects to view flight report
6. Flight class sends all flight information
Flight Class
5. Flight class retrieves flight information from database
Security Class

Detailed Design

```
class UserInterface {
public:
   // Method to display the login screen
   static void displayLoginScreen();
         {
                  Welcomes user and prompts them to enter login credentials
         }

   // Method to display successful login message
   static void displaySuccessfulLogin();
         {
                  Notifies user of successful login
         }
   // Method to display unsuccessful login message
   static void displayUnsuccessfulLogin();
         {
                  Notifies user of unsuccessful login
                  Prompts user with 3 options:
                           1)  Try to login again
                           2)  Create new account
                           3) Exit program
         }
   // Method to get username input from the user
   static std::string getUsernameInput();
         {
                  String username;
                  Prompts user to enter username
                  cin>>username;
                  return username;
         }
   // Method to get password input from the user (masked input) - they only see **** when
typing password
   static std::string getPasswordInput();
         {
                  String password =” “;
                  char ch;
                  While (true)
                           {
                           Ch = _getch();
```

```
                        Two if statements for the case of using enter or backspace
                        password.push_back();
                        }
                return password;
        }
};
```

```
class Login {
private:
   // Connect and disconnect from the database
   void connectToDatabase();
        {
        Try to connect to SQL database:
                Retrieve SQL password and if retrieval fails  display error to user
                Create connection to SQL server using driver "root" as the username
                Set the schema for the connection to airport_db
        Catch SQL Exception
                Print error message and set connection to nullptr
        }
   void disconnectDatabase();
        {
        If the connection object (con) is not null
                Set the connection object (con) to nullptr to show no active connection
        }

   // database connection object
   sql::Connection* con;  // stores the database connection

   std::string username;
   std::string password;
   std::string user_type;
   std::string first_name;
   int user_id;

   // setter functions
   // private because they should not be changed after object creation
```

```
    void setUserType(const std::string& type); // setter for user_type
            {
            User_type = type;
            }
    void setUserFirstName(const std::string& name); // setter for user_type
            {
            First_name = name;
            }
    void setUserID(const int& id); // setter for user_type
            {
            User_id = id;
            }

public:
    // Constructor
    Login();

    // Method to handle login logic
    bool login(const std::string& username, const std::string& password);
            {
            Try:
                    Set username = inputUsername
                    Set password = inputPassword

                    Prepare SQL Query to get user information based on inputUsername
                    Set inputUsername as parameter and execute query and store in res

                    If res has a result
                            Retrieve password hash and user type from the result

                    Compare entered password with password stored in database
                    If password is equal to dbPassword
                            Fetch first name from the appropriate table based on user type
                            Prepare query to get first name from appropriate table
                            Execute query and store result in res(user type)
                            Set first name = res(user type).first_name

                     Display  login success message
                    Call  handleUserType() based on userType (Passenger, Staff, Security)
```

```
        Catch SQL Exception as e:
                Display error message
                Return false;
        }

    bool createAccount(const std::string& username, const std::string& password, const
std::string& phone_num, const std::string& email,
                const std::string& first_name, const std::string& last_name, const std::string&
passport_num);
        {
        Try:
                Prepare query  to insert the user into the users table
                Set query parameters : inputUsername, inputPassword, inputEmail,
inputPhonenum
                Execute query

                Retrieve the user_id of new user and prepare query
                Execute query and retrieve user_id from database

                Prepare query to insert passenger details to passengers table
                Set parameters: user_id, inputFirstname, inputLastname, inputPassportnum
                Execute query to insert new passengers details to passengers table

                Display message showing successful account creation
                Return true;

        Catch SQL Exception as e:
                Display error message for account creation
                Return false;

        }
    void handleUserType();
        {
        If user type = " Passengers"
                Create an instance of Passenger with connection (con) and username
                Call displayOptions function on Passenger instance
        Else If user type = " Staff"
                Create an instance of Staff  with connection (con) and username
                Call displayOptions function on Staff  instance
        Else If user type = " Security"
```

```
                Create an instance of Security with connection (con) and username
                Call displayOptions function on Security instance
        Else
                Display a message of unknown user type
        }

    // getter functions
    std::string getUserType() const;
        {
        Return user_type;
        }
    std::string getUserFirstName() const;
        {
        Return first_name;
        }
    int getUserID() const;
        {
        Return user_id;
        }
    sql::Connection* getConnection(); // to pass connection object to other functions
        {
        Return con;
        }

    // Destructor
    ~Login();
};
```

```
class Passenger {
private:
    // database connection object
    sql::Connection* con;  // stores the database connection

    std::string username;
    std::string password;
    std::string user_type;
    std::string first_name;
    std::string description;
    int pass_id;
    int choice;
    int user_id;


public:
    // Constructor
    Passenger(sql::Connection* existingCon, std::string username);

    void displayOptions();
        {
        Display the five options the staff members have
            Check Flight Information
            Display Luggage Information
            Request Support
            View Your Support Request
            Log out

        If invalid choice, prompt user to select a valid choice
        If user hasn't logged out, ask if they want to see the menu again
        If user decides to log out:
            Set exitProgram to true;
            Display user is logging out
        }
    void handleChoice(int c);
        {
        Choice == 1:
            Call checkFlightInformation function
        Choice == 2:
            Call luggageInformation function
        Choice ==3:
            Call requestSupport function
        Choice ==4:
            Call view functionSupportRequest function
        Choice ==5
            Display user is logging out
        }
    void checkFlightInformation();
```

```
        {
        Create a new Flight object (flight) with connection (con) and username
        Call flight.displayInfo() to display the flight information
        }
    void luggageInformation();
        {
        Create a new Luggage object (luggage) with connection (con) and username
        Call luggage.displayInfo() to display the luggage information
        }
    void requestSupport();
        {
        Create a new SupportRequests object (sup) with the connection (con) and username
        Call sup.insertRequest() to insert a new support request
        }
    void viewSupportRequests();

    int getIntInput();

    // getters
    void setChoice(const int& choice);
        {
        If 1 <= c <=5:
                Set choice to c
        Else:
                Set choice to -1
        }
int getChoice() const;
        {
        Return choice;
        }
    sql::Connection* getConnection();
        {
        Return con;
        }

    // Destructor
    ~Passenger();
};
```

```
class Staff {
private:
    sql::Connection* con = nullptr;  // stores the database connection
    std::string username = "";
    int choice = -1;

public:
    Staff(sql::Connection* existingCon, std::string username);
    ~Staff();

    void displayOptions();
            {
            Display the five options the staff members have
                        View Open Support Tickets
                        View CLosed Support Tickets
                        Update Support Ticket Status
                        View Flight Information REport
                        Log out

            If invalid choice, prompt user to select a valid choice
            If user hasn't logged out, ask if they want to see the menu again
            If user decides to log out:
                        Set exitProgram to true;
                        Display user is logging out
            }
    void handleChoice(int c);
            {
            Choice == 1:
                        Call viewOpenTickets function
            Choice == 2:
                        Call viewClosedTickets function
            Choice ==3:
                        Call updateOpenTickets function
            Choice ==4:
                        Call viewFlightInformationReport function
            Choice == 5
                        Exit program
            }
    void viewOpenTickets();
            {
            Call the viewAllOpenTickets method on the sup instance to display open support tickets
            }
    void viewClosedTickets();
            {
            Call the viewAllClosedTickets method on the sup instance to display closed support
tickets
            }
    void updateOpenTickets();
            {
            Call the updateRequestStatus method on the sup instance to update the status of a support
ticket
            }
    void viewFlightInformationReport();
            {
            Call the displayFlightReport method on the f instance to show the flight information
report
            }
    int getIntInput();

    //setters
    void setChoice(const int& choice);
            {
            If 1 <= c <=5:
                        Set choice to c
            Else:
                        Set choice to -1
            }
    // getters
    int getChoice() const;
            {
            Return choice;
            }
    sql::Connection* getConnection();
            {
            Return con;
            }
};
```

```
class Security {
private:
    sql::Connection* con = nullptr;
    std::string username = "";
    int choice = -1;

    void setChoice(const int& choice);
    {
    If 1 <= c <= 4:
        Set choice to c
    Else:
        Set choice to -1

    }
    void handleChoice(int c);
    {
    Choice == 1:
        Call vlogSecurityThreat function
    Choice == 2:
        Call viewThreatList  function
    Choice ==3:
        Call viewFlightInformation function
    Choice ==4:
        Exit program
    }
    void logSecurityThreat();
    {
    String description, location;
    Int threatLevel;

    Get input from user for threat location, description and threat level
    Prepare an SQL query to insert new threat into Threats table with those variables

    Set description as first parameter, threat level as second, and location as third in the query
    Execute query
    Show that threat has been logged successfully
    }
    void viewThreatList();
```

```
        {
        Prepare an SQL query to select one of the following from the Threats table
                - Threat id
                - description
                - threat level
                - location
                - timestamp

        Execute query and store result in res

        While there are more records in the result set
                Retrieve, print and store the information from the set for every threat
        }
        void viewFlightInformation();
        {
        Create an instance of the Flight class with connection (con) and username
        Call the displayFlightReport() method on the Flight instance
        }

public:
        Security(sql::Connection* con, std::string user);

        void displayOptions();
        {
        Display the five options the staff members have
                Log Security Threat
                View Security Threat List
                View Flight Information
                Log out

        If invalid choice, prompt user to select a valid choice
        If user hasn't logged out, ask if they want to see the menu again
        If user decides to log out: (choice == 4)
                Set exitProgram to true;
                        Display user is logging out

        If exitProgram is false
                Ask user if they want to select another option
        }
        int getIntInput();
        int getChoice() const;
        {
        Return choice;
        }
        sql::Connection* getConnection();
        {
        Return con;
        }
};
```

```
class Luggage {
private:
   std::string username;
   int user_id;
   int luggage_id;
   int passenger_id;
   int flight_id;
   std::string location;

   sql::Connection* con;

public:
   Luggage(sql::Connection* existingCon, std::string username);
   void displayInfo();
};
   Prepare SQL query to get luggage details for the user based on their username
Set username in query
Execute Query and store result in res

If the result set is not empty:
        Set luggage_id from result set
        Set passenger_id from result set
        Set flight_id from result set
        Set location from result set
        Display luggage information
```

```
Else:
        Print that the user does not have a flight ticket


Class Flight{
        Private:

        sql::Connection* con;
        std::string username;

        int user_id;
        int passenger_id;
        int flight_id;
        std::string origin_airport_name;
        std::string destination_airport_name;
        std::string flight_number;
        std::string departure_time;
        std::string arrival_time;
        int origin_airport_id;
        int destination_airport_id;
        int capacity;


public:
        Flight(sql::Connection* existingCon, std::string username);
        void displayInfo();
        {
         Prepare SQL query to fetch flight information
        Set username as the parameter for the query
         Execute query and store result in resFlight

   If resFlight has a result:
      Fetch flight details from resFlight:
        flight_number
        departure_time
        arrival_time
        origin_airport_name
        destination_airport_name
        capacity
      Print flight information to the console:
        "Flight number: " + flight_number
        "Departure time: " + departure_time
        "Arrival time: " + arrival_time
        "Origin Airport: " + origin_airport_name
        "Destination Airport: " + destination_airport_name
        "Flight Capacity: " + capacity
   Else:
      Print "You don't have a flight ticket. Please visit the Ticket Booking desk to purchase a
flight ticket."

   Clean up by deleting the resFlight and pstmt objects

        }
        void displayFlightReport();
        {
        Prepare SQL query to fetch all flight numbers from the Flights table
        Execute the query and store results in flightNum_results

        Loop through all flight numbers
        Print flight number

        Prepare SQL query to select first name, last name, and passport number
        Set the flight number as a parameter in the query
        Execute query and store result in passengerRes

        Check if there are any passengers in the flight
        If hasPassengers is false
                    Display that there are no passengers found
        If hasPassengers is true
                    Display first name, last name, and passport number from passengerRes

        Clean up memory by deleting:
                    passenger_Res & passengers_pstmt
                    Flightnum_results & flightnum_pstmt
        }
};
```

```
Class SupportRequest{
```

```
private:
        sql::Connection* con = nullptr;
        std::string username = "";
        std::string supportRequestDescription = "";
        int pass_id = -1;
        int req_id = -1;
        bool supportRequestsExist = false;

public:
        SupportRequests(sql::Connection* con, std::string user);

        void setPassengerId();
        {
        Try:
        Prepare SQL query to select passenger_id from Passengers table
        Set username in the query parameter
        Execute the query and store result in res

        If res has a result
                Get passenger_id from result
                Store passengr_id in private variable pass_id

        Catch SQL Exception:
                Show error obtaining passenger id
        }

        void insertRequest();
        {
        Try:
        Get input from user and store in supportRequestDescription
        Call setPassengerId function

        Prepare SQL query to insert passenger_id and description to SupportRequest table
        Set passenger_id in first parameter and supportRequestDescription in the second

        Execute query:
                Show request has been sent
        Catch SQL Exception
        }
```

```
void viewUserRequests();
{
Try:
Call setPassengerId function
Prepare SQL query to select request_id, description, and status

Set passenger_id in the query parameter
Execute query and store in res

Loop through all the results (user's support requests)
If hasRequests is true:
        Print the request id, description, and status
If hasRequests is false:
        Show no support request were found

Catch SQL Exception
}

void viewAllOpenTickets();
{
 Prepare SQL query to select request_id, passenger_id, description, and status
Execute the query and store the result in res

// Loop through all the results (open tickets)
If hasOpenTickets true:
        Print request_id,passenger_id, description, and status
If hasOpenTickets is false:
        Show no tickets were found with pending or escalated status

Catch SQL Exception
}

void viewAllClosedTickets();
{
 Prepare SQL query to select request_id, passenger_id, description, and status
Execute the query and store the result in res

// Loop through all the results (closed tickets)
If hasOpenTickets true:
Print request_id,passenger_id, description, and status
```

```
    If hasOpenTickets is false:
    Show no tickets were found with resolved status

    Catch SQL Exception
    }
    void updateRequestStatus();
    {
    Call viewAllOpenTickets function
    }

    void getIntInput();
    {
    Prompt input from user
    Try:
        Convert strInput to an integer and store it in intInput
        Assign intInput to the variable req_id
    Catch exception:
        Print "Value entered must be an integer."
    }


    sql::Connection* getConnection();


};
```

Implementation

For this project, we selected C++ as the programming language due to its efficiency and strong support for object-oriented programming. Managing an airport system is complex and utilizing C++ provides the control and performance we need to handle tasks efficiently. For database management we used MySQL Workbench. This tool allowed us to design, manage, and interact without our MySQL database. We also selected Visual Studio 2022 as our IDE due to our familiarity built through coursework. The IDE's debugging tools and integration with Git streamlined the development process.

Throughout the development of our project we followed several programming practices that helped make our code more readable and maintainable. One of the first things we focused on was using meaningful and consistent variable names. For example, in our code variables like 'password', 'username', and 'flightDetails' were chosen because they clearly describe what they represent. For readability we were careful about our code formatting and maintained consistent indentation and spacing. When the code began to look complex we also left comments to describe the purpose of that block to ensure readers can understand what's going on. By sticking to these practices we were able to make our code more readable and maintainable.

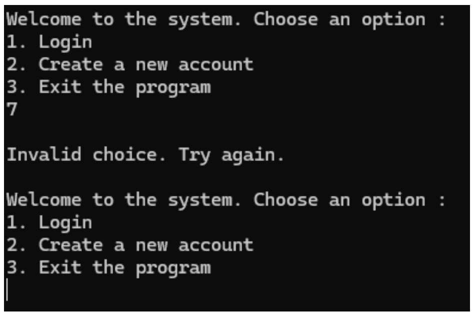
In our project we reused existing code for the password masking functionality in the UserInterface class. This was a common feature we used in previous projects and decided that it would be a good feature to implement in our project. Some code that can be reused from our project include the methods to connect and disconnect from the database. The database connection and disconnection processes were created in their own methods making it easy to extract them for future use.

For portability, our project is designed to be fairly portable. The core functionality relies on C++ and standard libraries. This means that it can be compiled on various operating systems without requiring many changes. The database itself may need some configuration or setup on different systems, but as long as they specify the directories in the project settings to compatible database drivers the project can be executed.

For version control we used Git in combination with GitHub. GitHub provided an easy way for us to manage our project files by allowing us to track changes. The ability to view and track changes over the course of development made it easier to monitor our progress and identify where changes were made. The commit feature in Git was extremely helpful throughout development. It allowed us to add descriptive messages with each change which made it easier to keep track of what was done and why. The commit history helped us keep a log of our progress. Overall, Git was an essential tool that helped us manage our code, track changes, and collaborate efficiently.

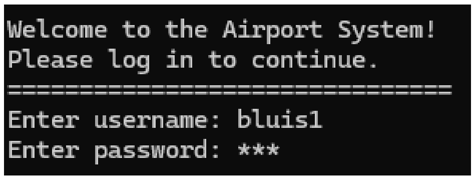
Test Cases and Results

Test Case: User selects invalid option in main menu
Expected Output: The system displays an error message and displays the main menu again.
Observed Output: Functional - the system displays 'invalid choice. Try again' and displays the main menu again for users to select from.



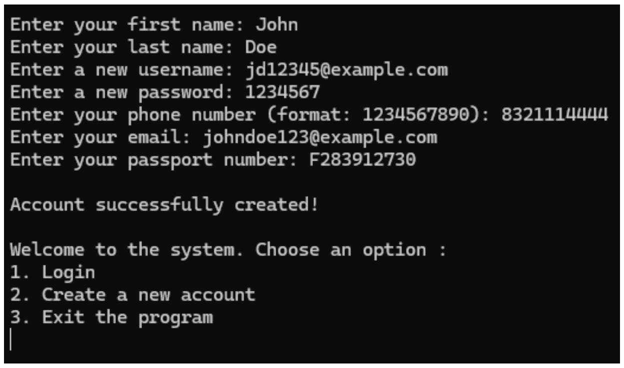
```
Welcome to the system. Choose an option :
1. Login
2. Create a new account
3. Exit the program
7

Invalid choice. Try again.

Welcome to the system. Choose an option :
1. Login
2. Create a new account
3. Exit the program
|
```

Test Case: User selects option 1 (Login) from main menu
Expected Output: System displays welcome message and prompts users to enter their username and password.
Observed Output: Functional - system displays appropriate prompts and takes in users username and password.



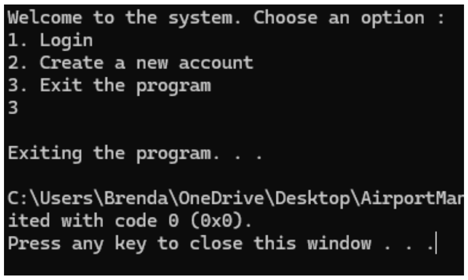
```
Welcome to the Airport System!
Please log in to continue.
==============================
Enter username: bluis1
Enter password: ***
```

Test Case: User selects option 2 (Create an account) from main menu
Expected Output: User is prompted to enter data relevant to account creation. If successful the system will notify the user and will then redisplay the main menu.
Observed Output: Functional - user is prompted to enter their information and system displays message if account creation is successful. The system then redisplays the main menu.



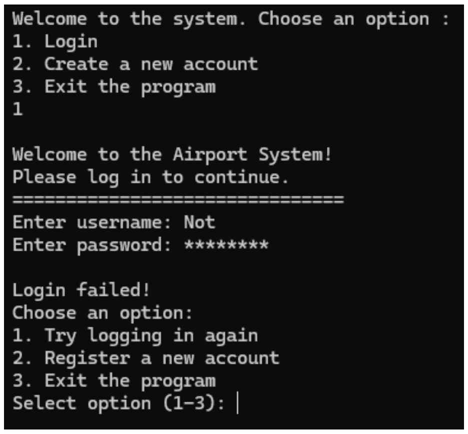
```
Enter your first name: John
Enter your last name: Doe
Enter a new username: jd12345@example.com
Enter a new password: 1234567
Enter your phone number (format: 1234567890): 8321114444
Enter your email: johndoe123@example.com
Enter your passport number: F283912730

Account successfully created!

Welcome to the system. Choose an option :
1. Login
2. Create a new account
3. Exit the program
|
```

Test Case: User selects option 3 (exit program) from main menu
Expected Output: System will display message informing user that program is exiting and then program will terminate.
Observed Output: Functional - program displays message and then continues to termination



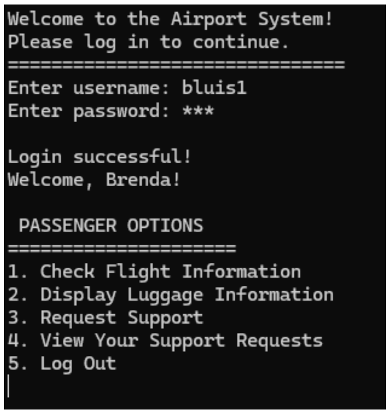
```
Welcome to the system. Choose an option :
1. Login
2. Create a new account
3. Exit the program
3

Exiting the program. . .

C:\Users\Brenda\OneDrive\Desktop\AirportMar
ited with code 0 (0x0).
Press any key to close this window . . .
```

Test Case: User attempts log in with invalid credentials in the database
Expected Output: System will notify user that login failed and prompt them with options to retry login, create an account, or exit the program.
Observed Output: Functional - system notifies user of failed login and prompts them with appropriate options.



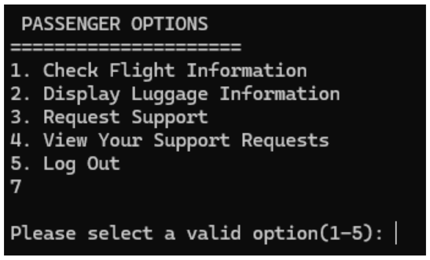
```
Welcome to the system. Choose an option :
1. Login
2. Create a new account
3. Exit the program
1

Welcome to the Airport System!
Please log in to continue.
==============================
Enter username: Not
Enter password: ********

Login failed!
Choose an option:
1. Try logging in again
2. Register a new account
3. Exit the program
Select option (1-3):
```

Test Case: Passenger logs in with existing credentials in the database
Expected Output: The system authenticates the user and displays the corresponding menu based on the user type (Passenger).
Observed Output: Functional - The system provides a successful login message along with a welcome message and first name of passenger. Passenger menu correctly displayed.



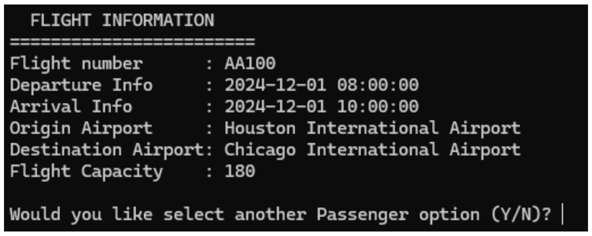
```
Welcome to the Airport System!
Please log in to continue.
==============================
Enter username: bluis1
Enter password: ***

Login successful!
Welcome, Brenda!

 PASSENGER OPTIONS
====================
1. Check Flight Information
2. Display Luggage Information
3. Request Support
4. View Your Support Requests
5. Log Out
```

Test Case: Passenger selects invalid option from Passenger menu
Expected Output: System will prompt user to enter a valid option.
Observed Output: Functional - System prompts user to enter valid option between 1-5.



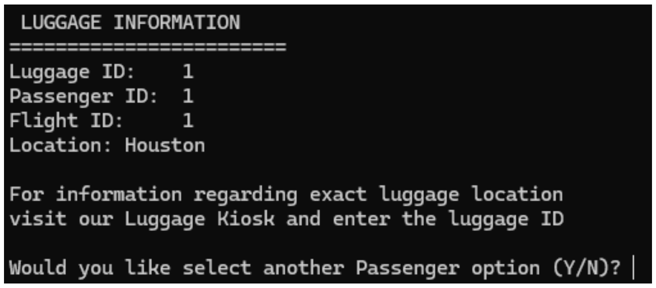
```
 PASSENGER OPTIONS
=====================
1. Check Flight Information
2. Display Luggage Information
3. Request Support
4. View Your Support Requests
5. Log Out
7

Please select a valid option(1-5):
```

Test Case: Passenger selects option 1 (check flight information) from passenger menu
Expected Output: System will output flight information if the user has a ticket in the database. Once the information is displayed the system will prompt the user if they wish to view passenger options again.
Observed Output: Functional - System displays flight information back to the user including details such as flight number, departure information, and arrival information. User is then prompted if they would like to view the passenger options again



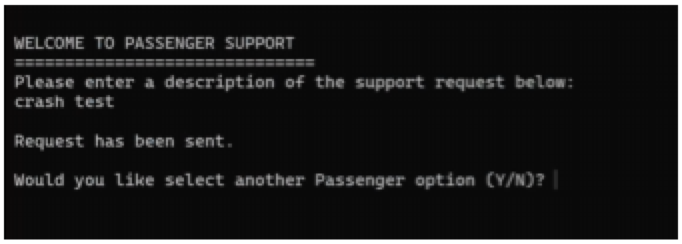
```
  FLIGHT INFORMATION
=======================
Flight number      : AA100
Departure Info     : 2024-12-01 08:00:00
Arrival Info       : 2024-12-01 10:00:00
Origin Airport     : Houston International Airport
Destination Airport: Chicago International Airport
Flight Capacity    : 180

Would you like select another Passenger option (Y/N)?
```

Test Case: Passenger selects option 2 (display luggage information) from passenger menu
Expected Output: System will display luggage information if available for the user in the database. After information is displayed system will prompt user to visit a Luggage Kiosk if they would like more information about their luggage. User is then prompted if they would like to view the passenger options again
Observed Output: Functional - System displays luggage information and appropriate prompts are displayed.



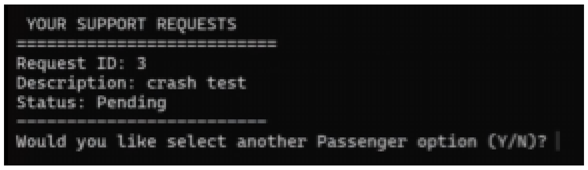
```
 LUGGAGE INFORMATION
=======================
Luggage ID:    1
Passenger ID:  1
Flight ID:     1
Location: Houston

For information regarding exact luggage location
visit our Luggage Kiosk and enter the luggage ID

Would you like select another Passenger option (Y/N)?
```

Test Case: Passenger selects option 3 (request support) from passenger menu
Expected Output: System will prompt users to enter a description of the support request. Once connection is established with the database and insert statement is executed, the system will display confirmation of the request being sent. User is then prompted if they would like to view the passenger options again
Observed Output: Functional - The system displays the appropriate prompts.



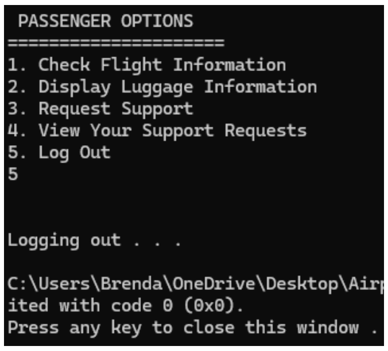
```
WELCOME TO PASSENGER SUPPORT
==============================
Please enter a description of the support request below:
crash test

Request has been sent.

Would you like select another Passenger option (Y/N)?
```

Test Case: Passenger selects option 4 (view your support requests) from passenger menu
Expected Output: The system displays available information on all support requests submitted by the user. Users are then prompted if they would like to view the passenger menu again.
Observed Output: Functional - System displays all support request and provides the user with appropriate prompt.



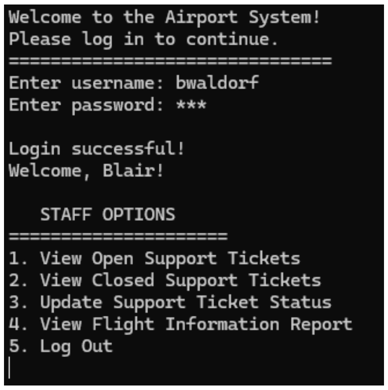
```
 YOUR SUPPORT REQUESTS
==========================
Request ID: 3
Description: crash test
Status: Pending
=========================
Would you like select another Passenger option (Y/N)?
```

Test Case: Passenger selects option 5 (log out) from passenger menu
Expected Output: System will display message informing user that they are logging out. System will then terminate the program.
Observed Output: Functional - System displays appropriate output before terminating program.



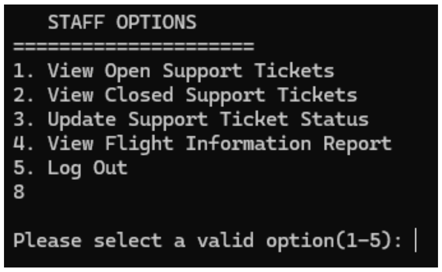
```
 PASSENGER OPTIONS
=====================
1. Check Flight Information
2. Display Luggage Information
3. Request Support
4. View Your Support Requests
5. Log Out
5


Logging out . . .

C:\Users\Brenda\OneDrive\Desktop\Air
ited with code 0 (0x0).
Press any key to close this window .
```

Test Case: Staff logs in with existing credentials in the database
Expected Output: The system authenticates the user and displays the corresponding menu based on the user type (Staff).
Observed Output: Functional - The system provides a successful login message along with a welcome message and first name of staff member. Staff menu correctly displayed.



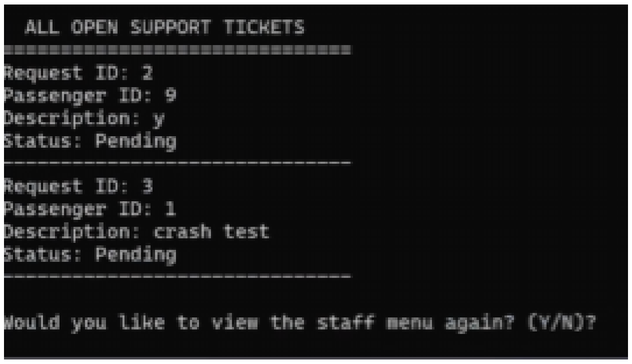
```
Welcome to the Airport System!
Please log in to continue.
===============================
Enter username: bwaldorf
Enter password: ***

Login successful!
Welcome, Blair!

   STAFF OPTIONS
====================
1. View Open Support Tickets
2. View Closed Support Tickets
3. Update Support Ticket Status
4. View Flight Information Report
5. Log Out
```

Test Case: Staff selects invalid option from staff menu
Expected Output: System will prompt user to enter a valid option.
Observed Output: Functional - System prompts user to enter valid option between 1-5.



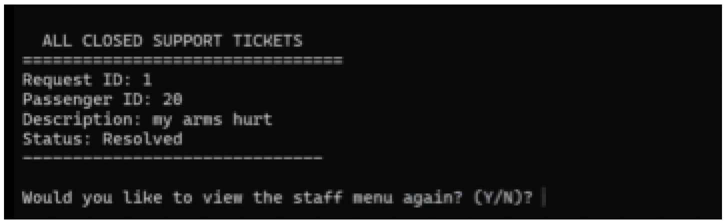
```
   STAFF OPTIONS
====================
1. View Open Support Tickets
2. View Closed Support Tickets
3. Update Support Ticket Status
4. View Flight Information Report
5. Log Out
8

Please select a valid option(1-5):
```

Test Case: Staff selects option 1 (view open support tickets) from staff menu
Expected Output: System will display all support requests from the database where the status is pending or escalated. Users are then prompted if they would like to view the staff menu again.
Observed Output: Functional - System displays appropriate data and prompts.



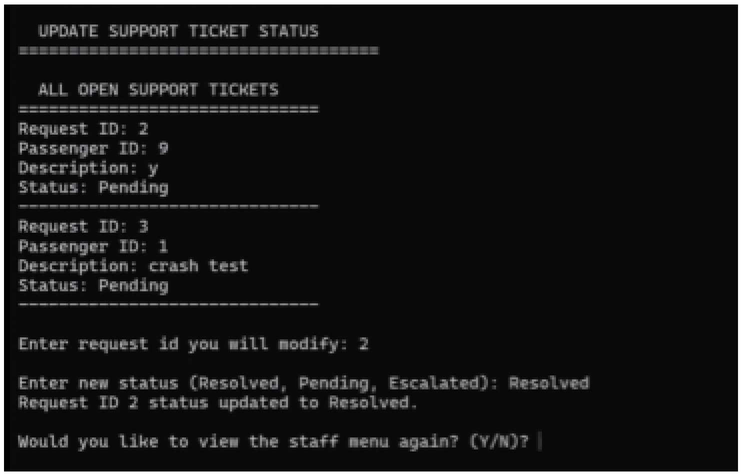
```
  ALL OPEN SUPPORT TICKETS
==============================
Request ID: 2
Passenger ID: 9
Description: y
Status: Pending
------------------------------
Request ID: 3
Passenger ID: 1
Description: crash test
Status: Pending
------------------------------

Would you like to view the staff menu again? (Y/N)?
```

Test Case: Staff selects option 2 (view closed support tickets) from staff menu
Expected Output: System will display all support requests from the database where the status is resolved. Users are then prompted if they would like to view the staff menu again.
Observed Output: Functional - System displays all closed support tickets and their associated information. Appropriate prompt is displayed after information is shown.



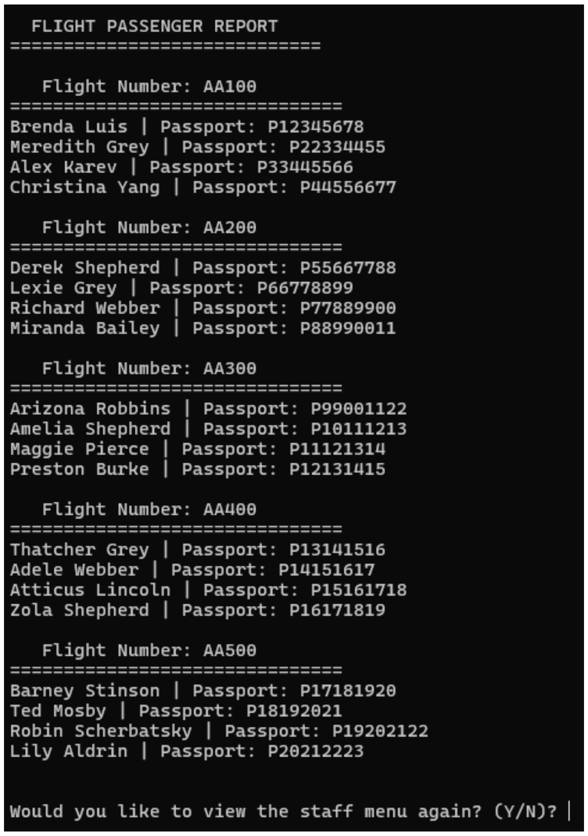
```
  ALL CLOSED SUPPORT TICKETS
================================
Request ID: 1
Passenger ID: 20
Description: my arms hurt
Status: Resolved
------------------------------

Would you like to view the staff menu again? (Y/N)?
```

Test Case: Staff selects option 3 (update support tickets) from staff menu
Expected Output: System will display all open support tickets. The system will then prompt the user to select the id of the request they will modify. If a valid id is entered, the system will prompt the user to enter the status they will update to. If a valid status is entered, confirmation will be displayed. Users are then prompted if they would like to view the staff menu again.
Observed Output: Functional - System displays appropriate ticket options, prompts and confirmation.



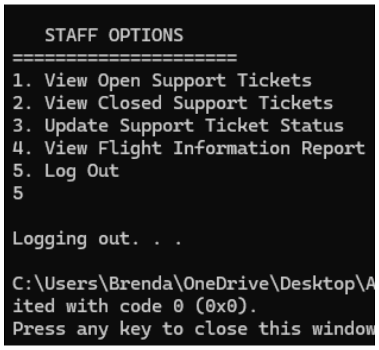
```
  UPDATE SUPPORT TICKET STATUS
====================================

  ALL OPEN SUPPORT TICKETS
==============================
Request ID: 2
Passenger ID: 9
Description: y
Status: Pending
------------------------------
Request ID: 3
Passenger ID: 1
Description: crash test
Status: Pending
------------------------------

Enter request id you will modify: 2

Enter new status (Resolved, Pending, Escalated): Resolved
Request ID 2 status updated to Resolved.

Would you like to view the staff menu again? (Y/N)?
```

Test Case: Staff selects option 4 (view flight information) from staff menu
Expected Output: System will display information about flight and passengers aboard the flight from the database. Users are then prompted if they would like to view the staff menu again.
Observed Output: Functional - System displays the appropriate data from the database. Appropriate prompt is displayed after data is shown.



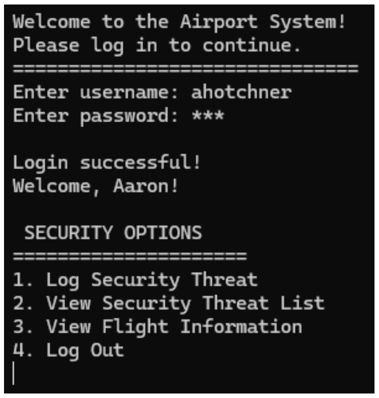
```
  FLIGHT PASSENGER REPORT
=============================

   Flight Number: AA100
===============================
Brenda Luis | Passport: P12345678
Meredith Grey | Passport: P22334455
Alex Karev | Passport: P33445566
Christina Yang | Passport: P44556677

   Flight Number: AA200
===============================
Derek Shepherd | Passport: P55667788
Lexie Grey | Passport: P66778899
Richard Webber | Passport: P77889900
Miranda Bailey | Passport: P88990011

   Flight Number: AA300
===============================
Arizona Robbins | Passport: P99001122
Amelia Shepherd | Passport: P10111213
Maggie Pierce | Passport: P11121314
Preston Burke | Passport: P12131415

   Flight Number: AA400
===============================
Thatcher Grey | Passport: P13141516
Adele Webber | Passport: P14151617
Atticus Lincoln | Passport: P15161718
Zola Shepherd | Passport: P16171819

   Flight Number: AA500
===============================
Barney Stinson | Passport: P17181920
Ted Mosby | Passport: P18192021
Robin Scherbatsky | Passport: P19202122
Lily Aldrin | Passport: P20212223


Would you like to view the staff menu again? (Y/N)?
```

Test Case: Staff selects option 5 (log out) from staff menu
Expected Output: System will display message informing user that they are logging out. System will then terminate the program.
Observed Output: Functional - System displays appropriate output before terminating program.



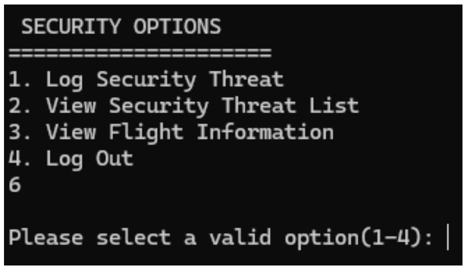
```
   STAFF OPTIONS
=====================
1. View Open Support Tickets
2. View Closed Support Tickets
3. Update Support Ticket Status
4. View Flight Information Report
5. Log Out
5

Logging out. . .

C:\Users\Brenda\OneDrive\Desktop\A
ited with code 0 (0x0).
Press any key to close this window
```

Test Case: Security logs in with existing credentials in the database
Expected Output: The system authenticates the user and displays the corresponding menu based on the user type (Security).
Observed Output: Functional - The system provides a successful login message along with a welcome message and first name of staff member. Security menu correctly displayed.



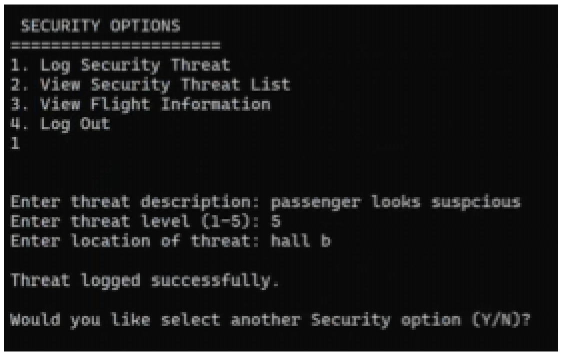
```
Welcome to the Airport System!
Please log in to continue.
==============================
Enter username: ahotchner
Enter password: ***

Login successful!
Welcome, Aaron!

 SECURITY OPTIONS
====================
1. Log Security Threat
2. View Security Threat List
3. View Flight Information
4. Log Out
```

Test Case: Security selects invalid option from security menu
Expected Output: System will prompt user to enter a valid option.
Observed Output: Functional - System prompts user to enter valid option between 1-4.



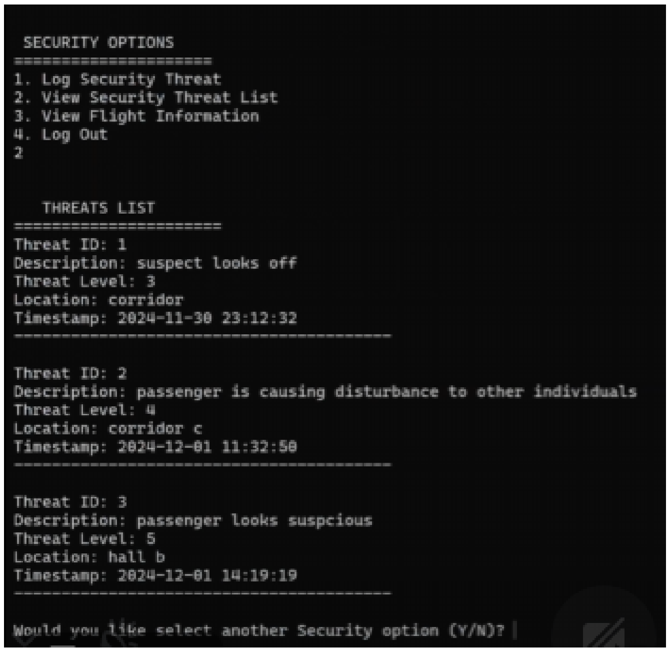
```
 SECURITY OPTIONS
====================
1. Log Security Threat
2. View Security Threat List
3. View Flight Information
4. Log Out
6

Please select a valid option(1-4):
```

Test Case: Security selects option 1 (log security threats) from security menu
Expected Output: System will prompt user to enter required information about the security threat. Upon successful update in the database, the system will display a message informing the user that the log was uploaded successfully. Users are then prompted if they would like to view the security menu again.
Observed Output: Functional - System displays appropriate prompts and console outputs.



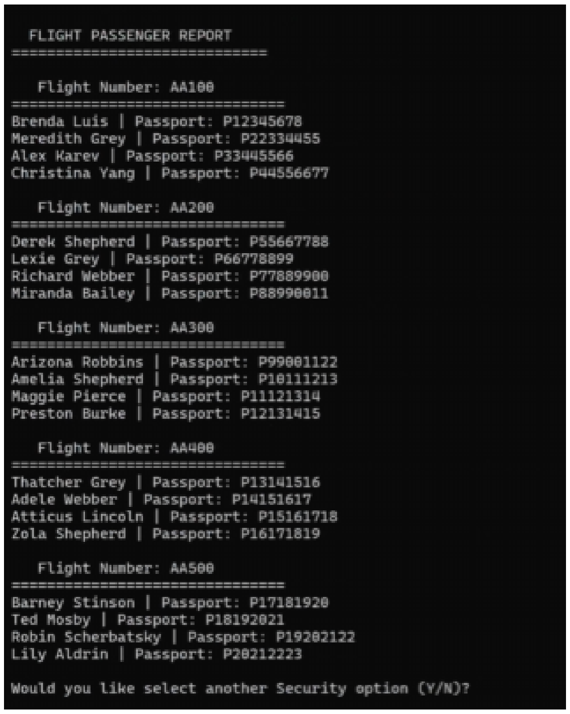
```
 SECURITY OPTIONS
====================
1. Log Security Threat
2. View Security Threat List
3. View Flight Information
4. Log Out
1


Enter threat description: passenger looks suspcious
Enter threat level (1-5): 5
Enter location of threat: hall b

Threat logged successfully.

Would you like select another Security option (Y/N)?
```

Test Case: Security selects option 2 (view a list security threat) from security menu
Expected Output: System will display a list of all security threats in the database. Users are then prompted if they would like to view the security menu again.
Observed Output: Functional - The system displays the appropriate data and prompts.



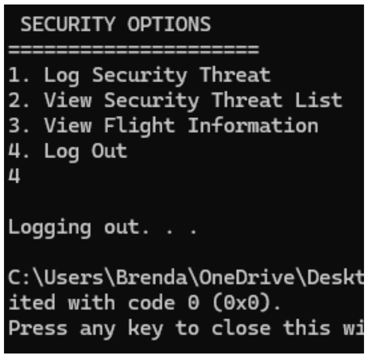
```
 SECURITY OPTIONS
=====================
1. Log Security Threat
2. View Security Threat List
3. View Flight Information
4. Log Out
2


   THREATS LIST
======================
Threat ID: 1
Description: suspect looks off
Threat Level: 3
Location: corridor
Timestamp: 2024-11-30 23:12:32
----------------------------------------

Threat ID: 2
Description: passenger is causing disturbance to other individuals
Threat Level: 4
Location: corridor c
Timestamp: 2024-12-01 11:32:50
----------------------------------------

Threat ID: 3
Description: passenger looks suspcious
Threat Level: 5
Location: hall b
Timestamp: 2024-12-01 14:19:19
----------------------------------------

Would you like select another Security option (Y/N)?
```

Test Case: Security selects option 3 (view flight information) from security menu
Expected Output: System will display information about flight and passengers aboard the flight from the database. Users are then prompted if they would like to view the staff menu again.
Observed Output: Functional - System displays the appropriate data from the database. Appropriate prompt is displayed after data is shown.



```
  FLIGHT PASSENGER REPORT
=============================

   Flight Number: AA100
===============================
Brenda Luis | Passport: P12345678
Meredith Grey | Passport: P22334455
Alex Karev | Passport: P33445566
Christina Yang | Passport: P44556677

   Flight Number: AA200
===============================
Derek Shepherd | Passport: P55667788
Lexie Grey | Passport: P66778899
Richard Webber | Passport: P77889900
Miranda Bailey | Passport: P88990011

   Flight Number: AA300
===============================
Arizona Robbins | Passport: P99001122
Amelia Shepherd | Passport: P10111213
Maggie Pierce | Passport: P11121314
Preston Burke | Passport: P12131415

   Flight Number: AA400
===============================
Thatcher Grey | Passport: P13141516
Adele Webber | Passport: P14151617
Atticus Lincoln | Passport: P15161718
Zola Shepherd | Passport: P16171819

   Flight Number: AA500
===============================
Barney Stinson | Passport: P17181920
Ted Mosby | Passport: P18192021
Robin Scherbatsky | Passport: P19202122
Lily Aldrin | Passport: P20212223

Would you like select another Security option (Y/N)?
```

Test Case: Security selects option 4 (log out) from security menu
Expected Output: System will display message informing user that they are logging out. System will then terminate the program.
Observed Output: Functional - System displays appropriate output before terminating program.



```
 SECURITY OPTIONS
=====================
1. Log Security Threat
2. View Security Threat List
3. View Flight Information
4. Log Out
4

Logging out. . .

C:\Users\Brenda\OneDrive\Deskt
ited with code 0 (0x0).
Press any key to close this wi
```

User Documentation

Installation Guide:

Software Download and Install:

You will receive a download link to access the airport management software. This link will direct you to a secure server where the software package and its necessary files are hosted. Download the zip software package to your system

After downloading, unzip the package and place the contents in a directory of your choice. This folder will contain the application file and required libraries for the software to function.

MySQL Installation:

The airport management software requires a MySQL database to store information for airports, passengers, staff, and security personnel. To setup MySQL:

1. Download MySQL Workbench (https://dev.mysql.com/downloads/workbench/)
2. Follow the installation instructions based on your operating system.
3. Download MySQL Community Server (https://dev.mysql.com/downloads/mysql/8.0.html)
4. Follow the installation instructions based on your operating system.

MySQL Setup:

Once MySQL is installed, you will need to create a database for the software to store the information. To do this:

1. Open MySQL Workbench and connect to your MySQL instance on localhost using the credentials you configured during installation.
2. Create a new schema/database with the name 'airport_db'.
3. Run the provided SQL script from the downloaded files to create the necessary tables for storing airport, passenger, staff, and security data.

Starting the Software:

Once the database setup is complete, navigate to the application file and open the program. The software automatically creates a connection to your MySQL database, and the system will be ready for use by passengers, staff, and security.

Note: The software does not have a predefined admin role in the system. Staff with administration credentials must have access to the MySQL database to manage security and staff records. Steps for administration staff to insert initial staff and security records:

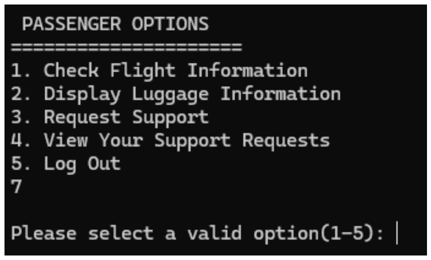
1. Use MySQL Workbench to connect to the airport_db schema.
2. Insert Staff and Security personnel data using SQL INSERT statements as shown below:

```
– inserting a staff member
INSERT INTO Users (username, password_hash, email, phone_number, user_type) VALUES
('bwaldorf', '123', 'blair@email.com', '713-111-1111', 'Staff');

INSERT INTO Staff (user_id, first_name, last_name, position) VALUES
((SELECT user_id FROM Users WHERE username = 'bwaldorf'), 'Blair', 'Waldorf', 'Manager');


– inserting a security member
INSERT INTO Users (username, password_hash, email, phone_number, user_type) VALUES
('ahotchner', '456', 'ahotchner@email.com', '713-444-4444', 'Security');

INSERT INTO Security (user_id, first_name, last_name, security_clearance)
VALUES
((SELECT user_id FROM Users WHERE username = 'ahotchner'), 'Aaron', 'Hotchner', TRUE),;
```

User Guide:
After installation, the software will allow users to access various features based on their roles. The key roles in the system are Passenger, Staff, and Security. Below are the descriptions of how each role interacts with the system.

Passenger Role:
Passengers use the system to find flight information, luggage information, and request assistance.

- Login: Passengers log in using their credentials to access the system.

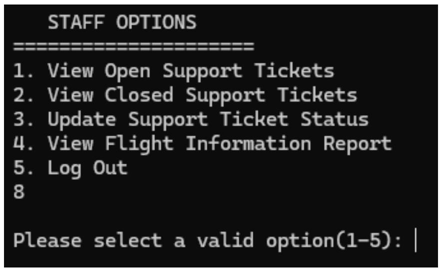


```
 PASSENGER OPTIONS
=====================
1. Check Flight Information
2. Display Luggage Information
3. Request Support
4. View Your Support Requests
5. Log Out
7

Please select a valid option(1-5):
```

- Check Flight Information: Passengers can view their flight details, including departure time, gate number, and status.
- Display Luggage Information: Passengers can view information about their luggage.
- Request Support: Passengers can request assistance for any issues or concerns they have, such as lost luggage, flight information, or general inquiries.
- View Your Support Requests: Passengers can view a list of their past and current support requests.
- Logout: Passengers logout and program closes.

Staff Role:
Staff members interact with the system to assist passengers with their support requests and generate reports on airport information.

- Login: Staff log in using their credentials to access the system.

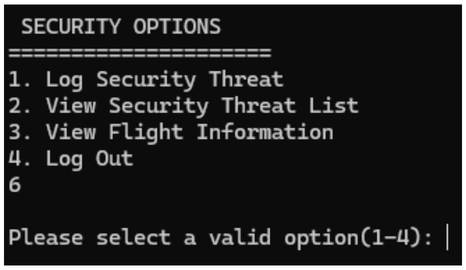


```
   STAFF OPTIONS
=====================
1. View Open Support Tickets
2. View Closed Support Tickets
3. Update Support Ticket Status
4. View Flight Information Report
5. Log Out
8

Please select a valid option(1-5): |
```

- View Open Support Tickets: Staff can view a list of support tickets that are open and awaiting resolution.
- View Closed Support Tickets: Staff can view a list of closed tickets
- Update Support Ticket Status: Staff can update the status of support tickets.
- View Flight Information Report: Staff can access flight information reports.
- Logout: Staff logout and program closes.

Security Role:
Security members interact with the system by logging and viewing threat reports.

- Login: Security log in using their credentials to access the system.



```
 SECURITY OPTIONS
=====================
1. Log Security Threat
2. View Security Threat List
3. View Flight Information
4. Log Out
6

Please select a valid option(1-4): |
```

- Log Security Threat: Security personnel can log security threats that occur in the airport
- View Security Threat List: Security can view a list of all security threats logged
- View Flight Information: Security can access flight information reports.
- Logout: Security logout and program closes.