**Comprehensive Algorithm for Airline Management System with Integrated Payment**

This algorithm covers the complete functionality of the system, with detailed steps for initialization, user management, navigation, flight search, booking, and payment processing.

**1. System Setup and Initialization**

**1.1 Environment Preparation:**

**Install Required Packages:**

Import necessary libraries: `customtkinter` for GUI, `Pillow` for images, `pymysql` for database operations, and `tkcalendar` for date selection.

For each package, try importing it; if not installed, use `subprocess` to install it automatically via `pip`.

**1.2 Main Database Connection:**

**Establish MySQL Connection:**

Connect to MySQL using `pymysql.connect()` with credentials (host, user, password, database).

Create a cursor (`cur`) for executing SQL queries.

**1.3 GUI Window Configuration:**

**Initialize Main Window:**

Set up the main window using `CTk()` with parameters for title, geometry, and appearance mode.

Center the window using `Global\_Config` functions for a consistent display across devices.

**1.4 Database Initialization:**

**Create Tables if Needed:**

Use `DB\_INIT\_()` function to ensure required tables are present:

`user\_details`: Stores user information.

`flights`: Stores available flight data.

`booking`: Tracks bookings with references to users and flights.

`payment`: Manages payment records for each booking.

***Outcome: The main GUI window and database are set up, and essential tables are ready for interaction.***

**2. User Authentication Management**

**2.1 Registration Process (`PG\_Sign\_Up`):**

**Display SignUp Form:**

Show input fields for name, username, password, email, phone, gender, and DOB (using `tkcalendar` for date input).

**Validate Inputs:**

Check that no fields are left empty, passwords match, and the username is unique by querying `user\_details`.

**Store User Data:**

If validation succeeds, insert user data into `user\_details`.

Provide feedback on successful registration or display error messages for issues.

**2.2 Login Process (`PG\_Sign\_in`):**

**Display Login Form:**

Show fields for username/email and password.

**Verify Credentials:**

Query `user\_details` to authenticate credentials.

If valid, update `\_isSignedIn` to `True` and assign `User` the username.

**Session Management:**

Redirect authenticated users to the flight search or payment page based on session state.

***Outcome: Users can register and log in, with sessions managed for authenticated access to specific features.***

**3. Navigation and Page Transition Management**

**3.1 Navigation Buttons:**

**Define Navigation Elements:**

Use `CTkButton()` to create buttons for key pages (Sign Up, Sign In, Flight Search).

Implement `go\_back()` to manage backward navigation, clearing the current frame and displaying the previous one.

**3.2 SessionBased Content Display:**

**Conditional Rendering:**

Display buttons and page content based on `\_isSignedIn` state, ensuring users only access relevant features based on their session status.

***Outcome: Seamless navigation across pages, with sessionaware controls for userspecific access.***

**4. Flight Search and Selection**

**4.1 Flight Search Page (`PG\_Get\_Flight\_Details`):**

**Display Search Form:**

Show dropdowns for selecting departure and destination, radio buttons for oneway or return, and date pickers for travel dates.

**Validate Inputs:**

Ensure all fields are complete and dates are valid (e.g., departure before return).

**Query Flights:**

Build and execute SQL queries to fetch available flights from `flights` based on input criteria.

**4.2 Display Search Results (`PG\_search\_flight\_`):**

**Scrollable Results Display:**

Use `CTkScrollableFrame` to display flight options with details like departure, arrival, airline, and price.

**Interactive Selection:**

Add buttons for each flight entry. Authenticated users are redirected to payment; nonauthenticated users are prompted to sign in.

***Outcome: Users can search for flights and view matching options, with booking access based on session state.***

**5. Booking and Ticket Management**

**5.1 Booking Confirmation (`booking()`):**

**Generate Ticket Code:**

Use `Ticket\_Code\_Gen.Gen\_Code()` to create a unique code for each booking.

**Store Booking Data:**

Insert booking details into the `booking` table, associating it with the user and flight.

**Allow Ticket Save:**

Provide users with the option to save ticket details to a local file using `filedialog.asksaveasfilename`.

***Outcome: Bookings are stored with unique ticket codes, and users have access to download their ticket.***

**6. Integrated Payment System**

**6.1 Payment Interface (`PG\_Payment()`):**

**Display Payment Options:**

Create options for UPI and net banking payments. Ensure each option clears previous components in the frame to display relevant input fields.

**6.2 UPI Payment Workflow (`on\_UPI\_btn\_click()`):**

**Display UPI Input:**

Show input fields for UPI number and display the total flight price.

**Validation and Database Entry:**

Check that fields are not empty, validate input, and insert payment record in `payment`.

**PostPayment Handling:**

Display confirmation message, initiate booking, and redirect back to the main page.

**6.3 Net Banking Workflow (`on\_NET\_btn\_click()`):**

**Display Banking Fields:**

Show fields for entering account number and display the total price.

**Validation and Database Entry:**

Validate inputs and insert the payment record in `payment`.

**PostPayment Handling:**

Confirm payment, initiate ticket booking, and navigate to the main page.

***Outcome: Payment is processed with a choice of methods, updating the database and confirming booking completion.***

**7. User Interface Feedback and Error Handling**

**7.1 Dynamic Feedback (`errorLabeling()`):**

**RealTime Feedback:**

Use `errorLabeling()` to display temporary messages for successful or failed actions, ensuring users receive immediate feedback.

**7.2 Validation and Exception Handling:**

**Input Validation:**

Check all input fields before database operations, and handle exceptions with tryexcept blocks to manage SQL errors.

***Outcome: The system provides clear feedback and error handling, enhancing user experience and stability.***

**8. Additional Account Features**

**8.1 Account Management:**

**Settings and History:**

Implement options for flight cancellation, booking history, account edits, and logout.

**Flight Cancellation:**

Prompt the user for a ticket ID, verify it, and update the booking to inactive if validated.

**View Booking History:**

Retrieve booking records for the loggedin user from `booking` and display them.

**Edit Account:**

Provide options to update user details, ensuring validation of changes.

***Outcome: Users can manage bookings, view history, and update account information securely.***

**9. System Exit and Resource Management**

**9.1 Resource Cleanup:**

**Database Connection Closure:**

Ensure `cur.close()` and `con.close()` are called on application exit to avoid resource leaks.

**Application Termination:**

End the main loop with `root.mainloop()`, ensuring a smooth and errorfree shutdown.

***Outcome: The application exits gracefully, with all resources properly released.***

**Potential Enhancements and Security Considerations**

**Security Enhancements:**

Hash passwords, encrypt sensitive data, and use prepared statements for SQL queries to protect against SQL injection.

**Feature Extensions:**

Add more payment options and integrate thirdparty APIs for additional functionality.

**Notifications:**

Implement email or inapp notifications for payment confirmation and booking reminders.