**Full Algorithm for the Airline Management System**

**1. System Setup and Initialization**

**Objective**: Prepare the environment and initiate the main application window with necessary configurations and database connectivity.

* **Step 1.1**: Check and install dependencies.
  + **Input**: List of required packages (customtkinter, pillow, pymysql, etc.).
  + **Process**:
    1. Iterate over each package in the list.
    2. **Try** to import the package:
       - If successful, print a confirmation message.
       - If ImportError occurs, use subprocess.check\_call() to install the package with pip.
  + **Output**: All packages are installed and imported.
* **Step 1.2**: Import necessary libraries and modules.
  + **Input**: Core Python modules (os, sys), GUI libraries, database libraries.
  + **Process**: Import each library.
  + **Output**: All libraries are ready for use.
* **Step 1.3**: Initialize the main application window.
  + **Input**: Window configuration properties.
  + **Process**:
    1. Instantiate the CTk() object as root.
    2. Set window properties (title, geometry, state, appearance mode).
    3. Center the window using a custom function from Global\_Config.
  + **Output**: The main application window is initialized and ready.
* **Step 1.4**: Establish a database connection.
  + **Input**: Database credentials (host, user, password, database).
  + **Process**:
    1. Connect to the MySQL database using pymysql.connect().
    2. Create a cursor object (cur).
  + **Output**: Database connection established for queries and transactions.

**2. User Session Management**

**Objective**: Define global session control variables and manage user authentication.

* **Step 2.1**: Define global session variables.
  + **Input**: Initial state for session variables (\_isSignedIn = False, User = "").
  + **Output**: Variables set for tracking user status.
* **Step 2.2**: Create functions for login and logout.
  + **PG\_Sign\_in() Function**:
    - **Input**: User credentials (username, password).
    - **Process**:
      1. Display a form with input fields for the username/email and password.
      2. Verify the credentials against the user\_details table in the database.
      3. **If valid**:
         * Update \_isSignedIn to True.
         * Assign the User variable with the authenticated username.
         * Redirect to the main dashboard.
      4. **If invalid**:
         * Display an error message.
    - **Output**: User session is updated; feedback provided on success or failure.
* **Step 2.3**: Implement PG\_Sign\_Up() for user registration.
  + **Input**: User details (name, username, password, email, phone, gender, DOB).
  + **Process**:
    - Display a form with the required fields and a date picker for DOB.
    - Validate the input:
      1. Ensure no field is empty.
      2. Check password match and security strength.
      3. Query the database to ensure username uniqueness.
    - **If valid**:
      1. Insert the user data into user\_details.
      2. Display a success message and redirect to the login page.
    - **If invalid**:
      1. Show appropriate error messages.
  + **Output**: User data stored, or error shown.

**3. Navigation and Page Transitions**

**Objective**: Create a fluid user experience by enabling smooth transitions between pages.

* **Step 3.1**: Implement navigation buttons.
  + **Input**: Button actions (e.g., Sign Up, Sign In, Back).
  + **Process**:
    1. Create buttons using CTkButton() with appropriate commands to switch between pages.
    2. Use lambda functions for parameterized navigation.
  + **Output**: Buttons that navigate between registration, login, and flight search pages.
* **Step 3.2**: Develop go\_back() functions.
  + **Input**: Current page context.
  + **Process**:
    1. Clear the current frame.
    2. Redirect the user to the previous page or main interface.
  + **Output**: User returned to the desired page.

**4. Flight Search and Booking Workflow**

**Objective**: Allow users to search for available flights based on criteria and enable booking actions.

* **Step 4.1**: Build PG\_Get\_Flight\_Details() for initial search input.
  + **Input**: Flight criteria (origin, destination, dates).
  + **Process**:
    1. Display dropdowns for origin and destination using CTkComboBox().
    2. Create radio buttons for selecting flight type (one-way or return).
    3. Add date pickers for departure and return dates.
  + **Output**: Search form ready for user interaction.
* **Step 4.2**: Implement validation in PG\_Get\_Flight\_Details().
  + **Input**: User-selected data (origin, destination, dates).
  + **Process**:
    1. Check if the user has filled out all required fields.
    2. Validate the date selections (e.g., return date cannot be before departure date).
  + **Output**: Validated input or an error message displayed.
* **Step 4.3**: Create PG\_search\_flight\_() to display results.
  + **Input**: Validated search criteria.
  + **Process**:
    1. Query the flights table for matching results.
    2. Display flights in a scrollable frame with details like departure, arrival, airline, and price.
    3. Add buttons for each result, enabling users to proceed with booking or viewing more details.
  + **Output**: Search results displayed; booking enabled.

**5. User Feedback Mechanism**

**Objective**: Enhance user experience with real-time feedback.

* **Step 5.1**: Implement errorLabeling() for dynamic messages.
  + **Input**: Message text, appearance properties (e.g., color, font).
  + **Process**:
    1. Create a temporary label using CTkLabel().
    2. Place it within the current frame and set a timer to auto-destroy using after().
  + **Output**: User feedback message shown temporarily.
* **Step 5.2**: Use errorLabeling() in key functions:
  + Registration and login for validation errors.
  + Flight search for no-results scenarios or incorrect input.

**Outcome**: Users receive immediate feedback on actions and input status.

**6. Session Control and Data Management**

**Objective**: Ensure consistent session tracking and secure data handling.

* **Step 6.1**: Maintain session state.
  + **Input**: \_isSignedIn, User.
  + **Process**:
    1. Update session variables during sign-in or sign-out actions.
    2. Restrict access to sensitive features when \_isSignedIn is False.
  + **Output**: Session-aware interface with conditional functionality.
* **Step 6.2**: Implement secure data handling practices.
  + **Input**: User and flight data.
  + **Process**:
    1. Use parameterized queries to prevent SQL injection.
    2. Encrypt passwords using hashing (e.g., bcrypt or hashlib).
  + **Output**: Secure user data stored and managed.

**7. Resource Management and Application Termination**

**Objective**: Clean up resources and ensure a safe application shutdown.

* **Step 7.1**: Handle database connection cleanup.
  + **Input**: Active database connection and cursor.
  + **Process**:
    1. Close the cursor (cur.close()).
    2. Commit changes and close the connection (con.commit() and con.close()).
  + **Output**: Resources released; database connection closed.
* **Step 7.2**: Gracefully end the application.
  + **Input**: root.mainloop().
  + **Process**:
    1. Ensure that the GUI stops running smoothly.
    2. Handle exceptions that may arise during closure.
  + **Output**: Application exits without errors or resource leaks.

**Summary and Potential Expansions**

**Future Enhancements**:

* Add payment integration and support for international flight data.
* Enhance user experience with a dashboard displaying past and upcoming bookings.
* Implement user notification systems for flight status updates.

**Scalability Considerations**:

* Transition the database to a cloud-based solution for scalability.
* Use asynchronous data fetching for faster UI updates.

**Security Recommendations**:

* Regularly update libraries to patch vulnerabilities.
* Implement session timeouts and multi-factor authentication for enhanced security.