Technical Issue Report: Domestic Round Trip Booking Failure

1. ISSUE DESCRIPTION

Problem: When booking domestic round-trip flights, only the **return flight gets booked** while the **onward flight booking fails**.

Affected Scenario: Domestic flights with round trip (tripType = 1)

Symptoms:

- Search works correctly (shows both onward and return flights)
- Price verification works for both flights
- Payment completes successfully
- But only return flight appears in booking confirmation
- Onward flight booking data is lost/overwritten

2. ROOT CAUSE ANALYSIS

2.1 Frontend Data Handling Issue

Current Flow:

1. First API call → ref.current = 1 → stores onward data
2. Second API call → ref.current = 2 → stores return data
3. But both calls use **same API endpoint with identical payload**
4. Backend cannot distinguish between onward and return searches

Problematic Code (Frontend - searchFlights function):

```
javascript

if (searchData?.serType === 1) {

  if (ref.current === 1) {

    setOnwardData(data);

    setOnwardRefID(data.data?.Status?.refID || "");

    ref.current = 2;

  } else if (ref.current === 2) {

    setReturnData(data);

    setReturnRefId(data.data?.Status?.refID || "");
```

```
 }

}
```

Issue: ref.current is a client-side counter but backend receives **same search parameters** for both onward and return searches.

2.2 Backend Identification Gap

Current Backend Route:

```
javascript

routes.post("/postSearchFlight", checkFlightToken, async (req, res) => {

 const { ...searchData } = req.body; // No identifier for search type

 // Both onward and return searches look identical to backend

});
```

Missing: Backend has **no way to know** if the request is for:

- Onward flights (City A → City B)
- Return flights (City B → City A)
- Because cities are swapped but no metadata indicates this

2.3 Booking Data Loss

During booking:

```
javascript

const reqBody = {

 // Only one flight's data sent

 onwardRefID: onwardRefID,

 returnRefID: returnRefId,

 // But both flights' full data not preserved

};
```

Result: Backend receives mixed/confused data about which flight is which.

3. IMPACT

1. **User Experience:** Users pay for round trip but only get return flight
2. **Revenue Loss:** Potential refunds and customer dissatisfaction

3. **Data Integrity:** Booking records incomplete
4. **Operational Issues:** Manual intervention required to fix bookings

4. PROPOSED SOLUTION

4.1 Backend Changes

File: flight.routes.js

Add searchType parameter to differentiate searches:

javascript

```
routes.post("/postSearchFlight", checkFlightToken, async (req, res) => {
 try {
  const {
   searchType, // ✅ NEW: 'onward' or 'return'
   ...searchData
  } = req.body;

  const flightToken = req.flightToken;

  const response = await axios.post(
   `${API_URL}/postSearchFlight`,
   searchData,
   {
    headers: {
     "x-api-key": flightToken,
     mode: modeNumber,
    },
   }
  );
```

```
    res.status(200).json({
      success: true,
      message: "Flight Search Successfully",
      data: response.data,
      searchType: searchType || 'onward', // ✅ Return search type in response
    });
  } catch (error) {
    // ... error handling
  }
});
```

4.2 Frontend Changes

File: Homepage2.jsx

A. Modify searchFlights function:

```
javascript
const searchFlights = async (searchData, searchType = 'onward') => {
  // ✅ Accept searchType parameter
  try {
    setDebugInfo(`Starting ${searchType} flight search...`);

    const response = await fetch(`${API_BASE}/api/postSearchFlight`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      credentials: "include",
      body: JSON.stringify({
        ...searchData,
        searchType, // ✅ Tell backend what type of search
      }),
```

```
    });

    const data = await response.json();

    // ✅ Store data based on searchType
    if (searchType === 'onward') {
      setOnwardData(data);
      setOnwardRefID(data.data?.Status?.refID || "");
    } else if (searchType === 'return') {
      setReturnData(data);
      setReturnRefId(data.data?.Status?.refID || "");
    }

    return data.data;
  } catch (error) {
    throw error;
  }
};
```

B. Update handleSearch function:

```
javascript

const handleSearch = async (e) => {
  // ... validation code ...

  // ✅ Search onward flights
  const onwardResponse = await searchFlights(onwardSearchData, 'onward');

  // ✅ If round trip, search return flights
```

```
 if (tripType === 1) {
   const returnResponse = await searchFlights(returnSearchData, 'return');
 }

 // ... combine and display results
};
```

C. Enhance booking payload:

```
javascript
const bookSelectedFlight = async (flight, passenger, searchPayload) => {
 const reqBody = {
  // ... existing fields ...

  // ✅ Clear identification of flights
  isRoundTrip: tripType === 1,
  searchType: 'roundtrip',
  onwardData: {
   refID: onwardRefID,
   flightID: flight.onwardFlight?.Flights?.Onward[0]?.flightID,
   flightDetails: flight.onwardFlight,
  },
  returnData: {
   refID: returnRefId,
   flightID: flight.returnFlight?.Flights?.Onward[0]?.flightID,
   flightDetails: flight.returnFlight,
  },
 };
 // ... send to backend
```

};

4.3 Database Schema Enhancement (Optional but Recommended)

Add to FlightBooking model:

javascript

```
const flightBookingSchema = new mongoose.Schema({
  // ... existing fields ...

  tripType: {
    type: Number, // 0 for one-way, 1 for round-trip
    required: true,
  },

  onwardFlightDetails: {
    refID: String,
    flightID: String,
    airline: String,
    departure: Date,
    arrival: Date,
    // ... other relevant fields
  },

  returnFlightDetails: {
    refID: String,
    flightID: String,
    airline: String,
    departure: Date,
    arrival: Date,
```

```
  // ... other relevant fields
 },


 // ... rest of schema
});
```

5. TESTING PLAN

Test Cases:

1. **Domestic One-Way Search** (tripType = 0)
2. **Domestic Round Trip Search** (tripType = 1)
3. **International One-Way Search** (serType = 2)
4. **International Round Trip Search** (tripType = 1, serType = 2)

Verification Points:

- ✅ Both onward and return data stored separately
- ✅ Correct refIDs preserved for both flights
- ✅ Booking payload contains both flights' data
- ✅ Database records show complete round trip details

6. RISK MITIGATION

1. **Backward Compatibility:** Maintain old behavior for one-way trips
2. **Data Migration:** No existing data affected
3. **Fallback Mechanism:** If searchType missing, default to 'onward'
4. **Error Logging:** Enhanced logging to track search types

7. IMPLEMENTATION TIMELINE

Phase 1 (Backend - 1 day):

- Add searchType parameter to API
- Update response structure

Phase 2 (Frontend - 1 day):

- Modify searchFlights function
- Update handleSearch logic
- Enhance booking payload

Phase 3 (Testing - 1 day):

- Test all search combinations
- Verify booking flow

- Deploy to staging

Total Estimated Time: 3 working days

8. SUCCESS METRICS

1. **100% round trip booking success rate**
2. **Zero mixed-up onward/return flights**
3. **Complete booking records in database**
4. **No regression in one-way bookings**

VERDICT:

Yes, this solution will work. The core issue is **lack of metadata to distinguish between onward and return searches**. By adding a searchType parameter and properly segregating the data flow, we ensure each flight's data is preserved and correctly associated throughout the booking process.

Confidence Level: 95% - This addresses the root cause directly and maintains backward compatibility.

Date:- 7th Dec 2025

Approved By:- Ayan Sir, Aman Raj

This is the implementation.