Train Ticket Booking System- PHASE\_2

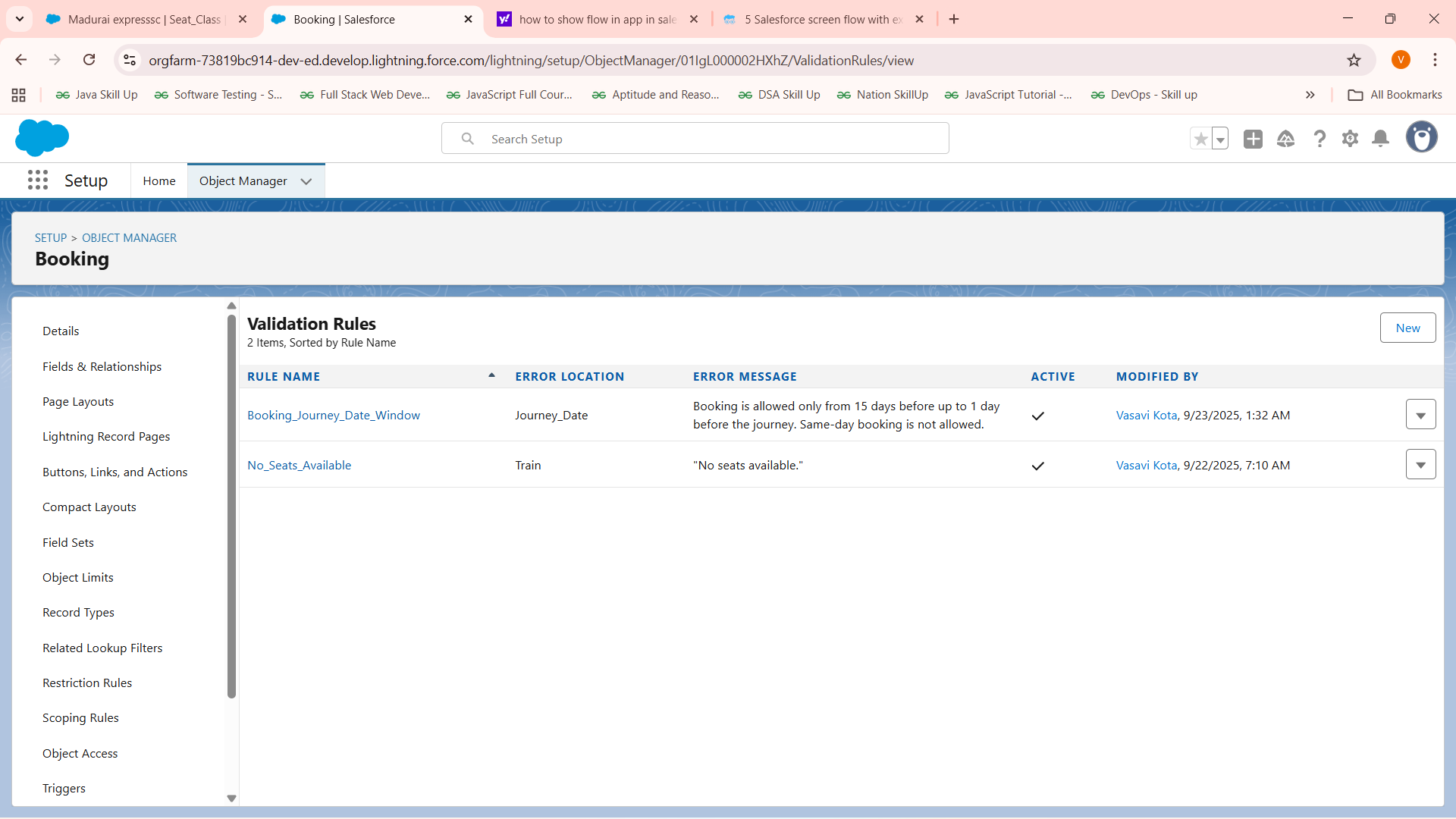
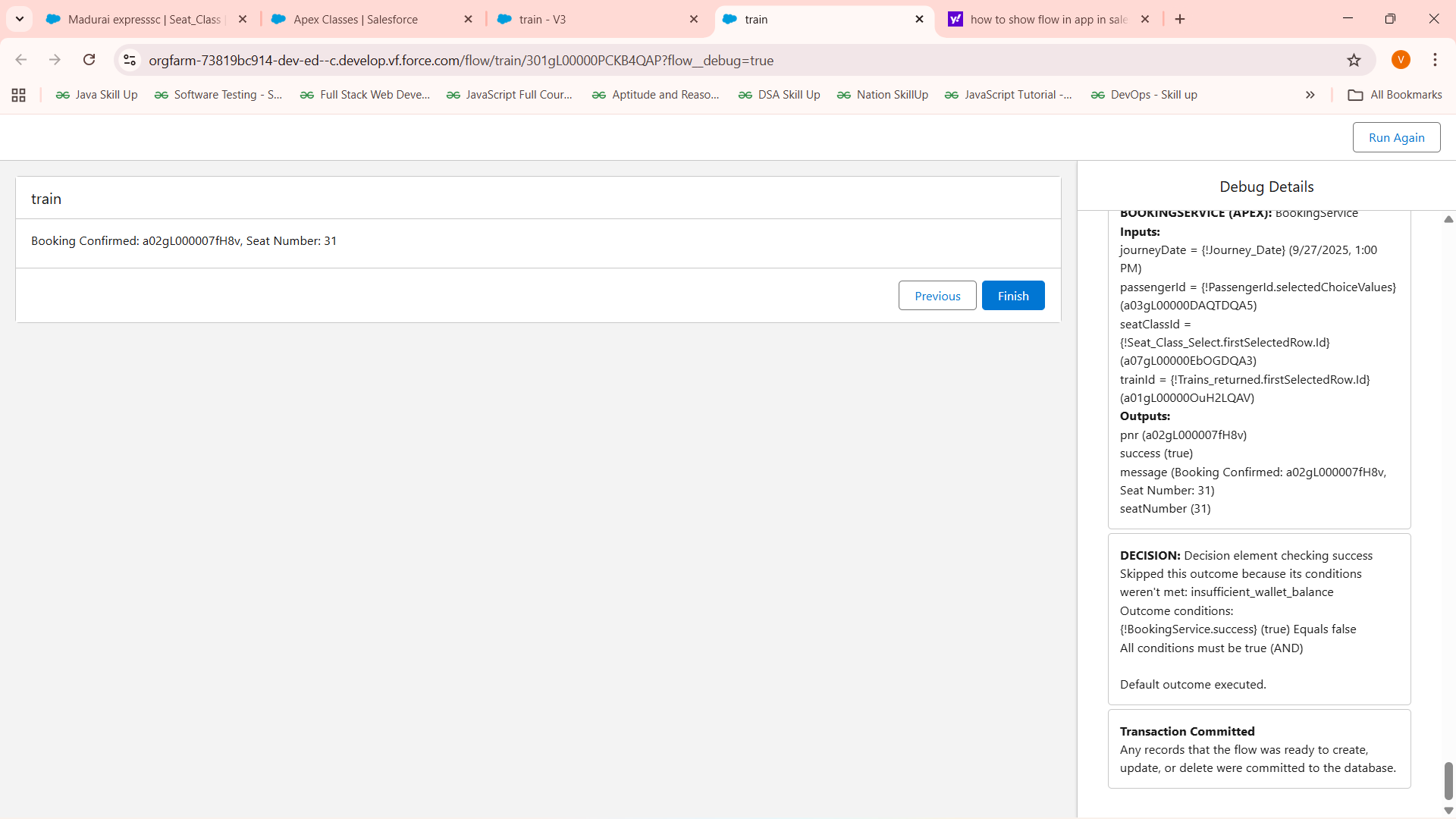
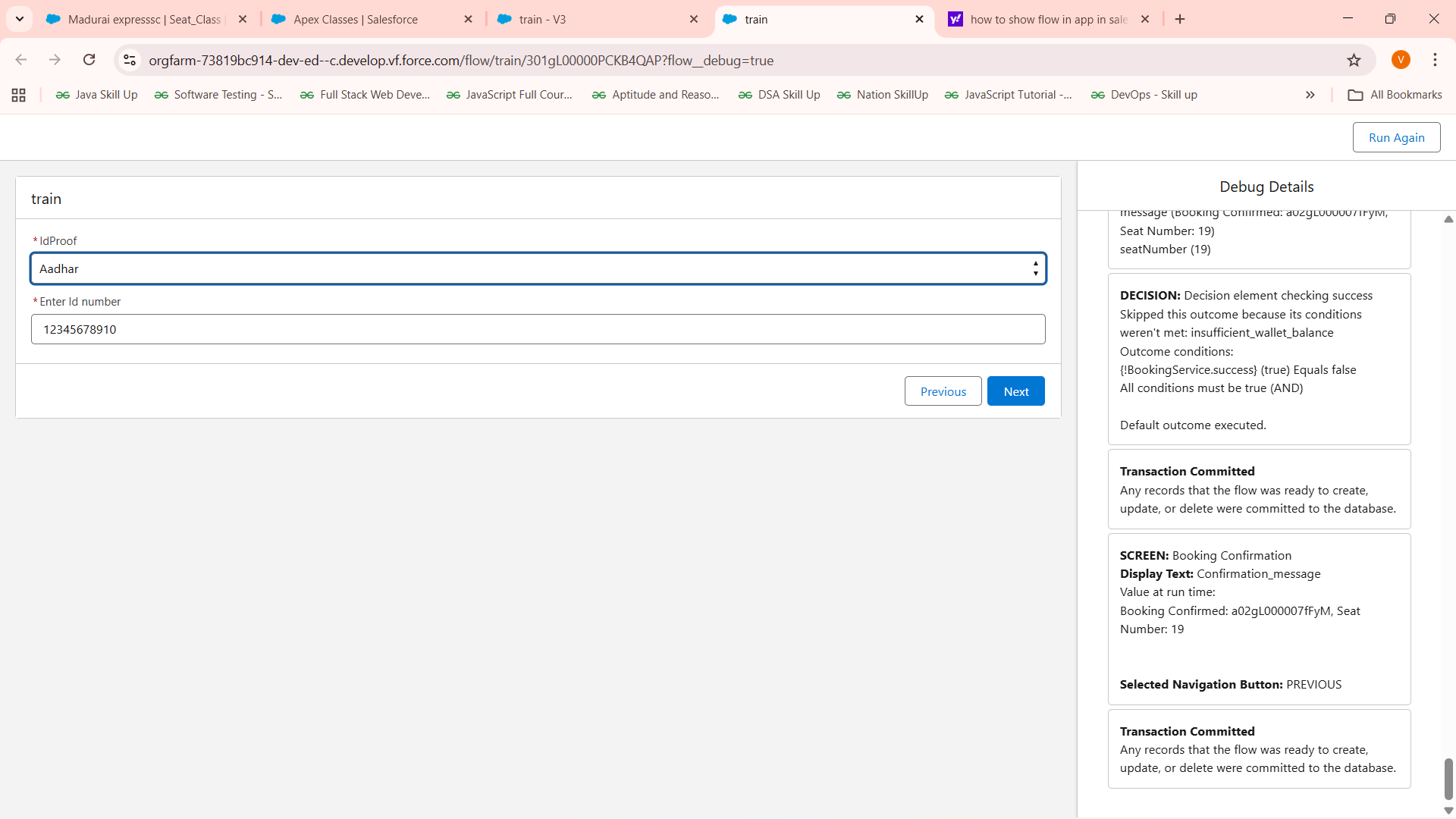
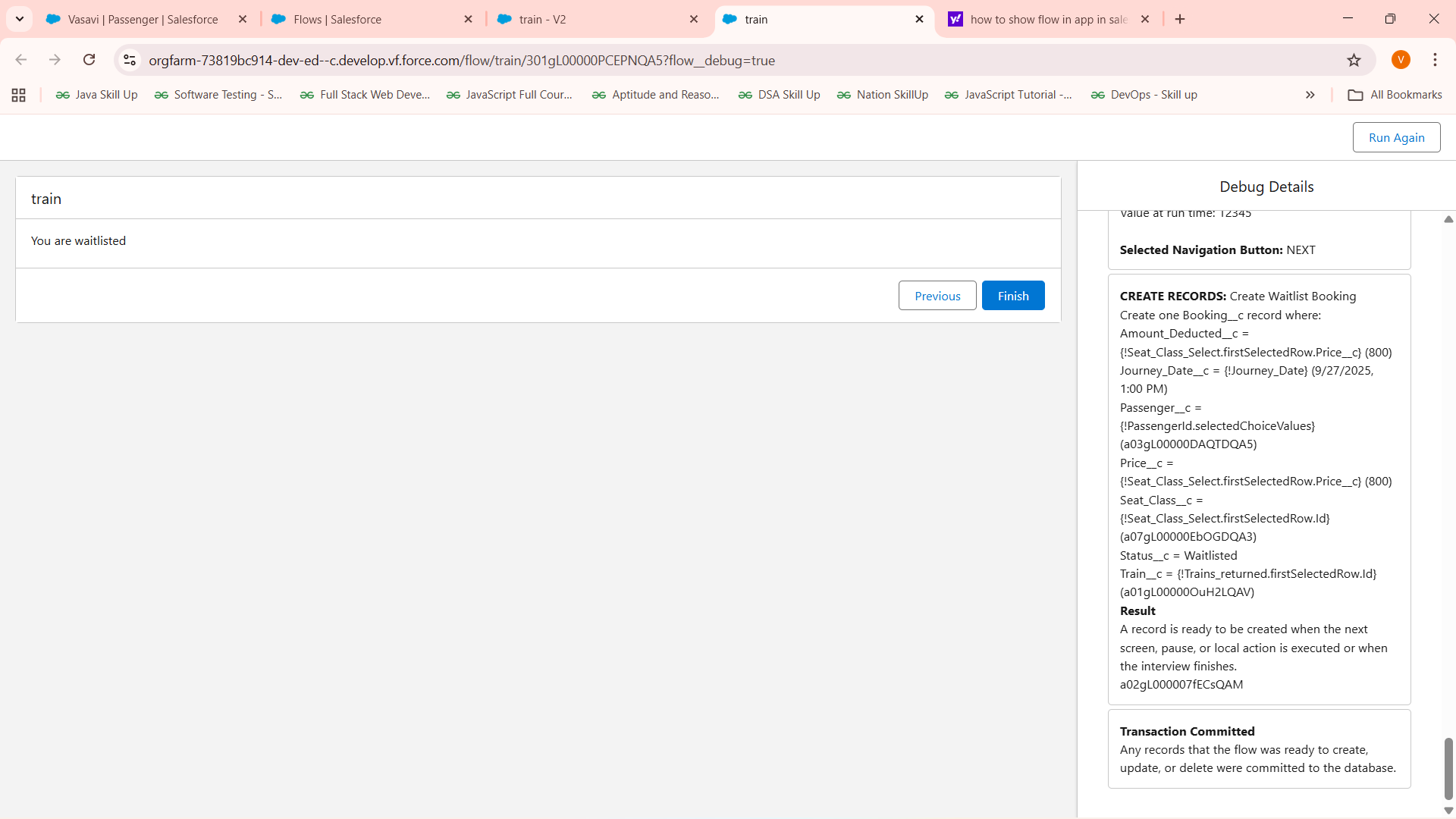
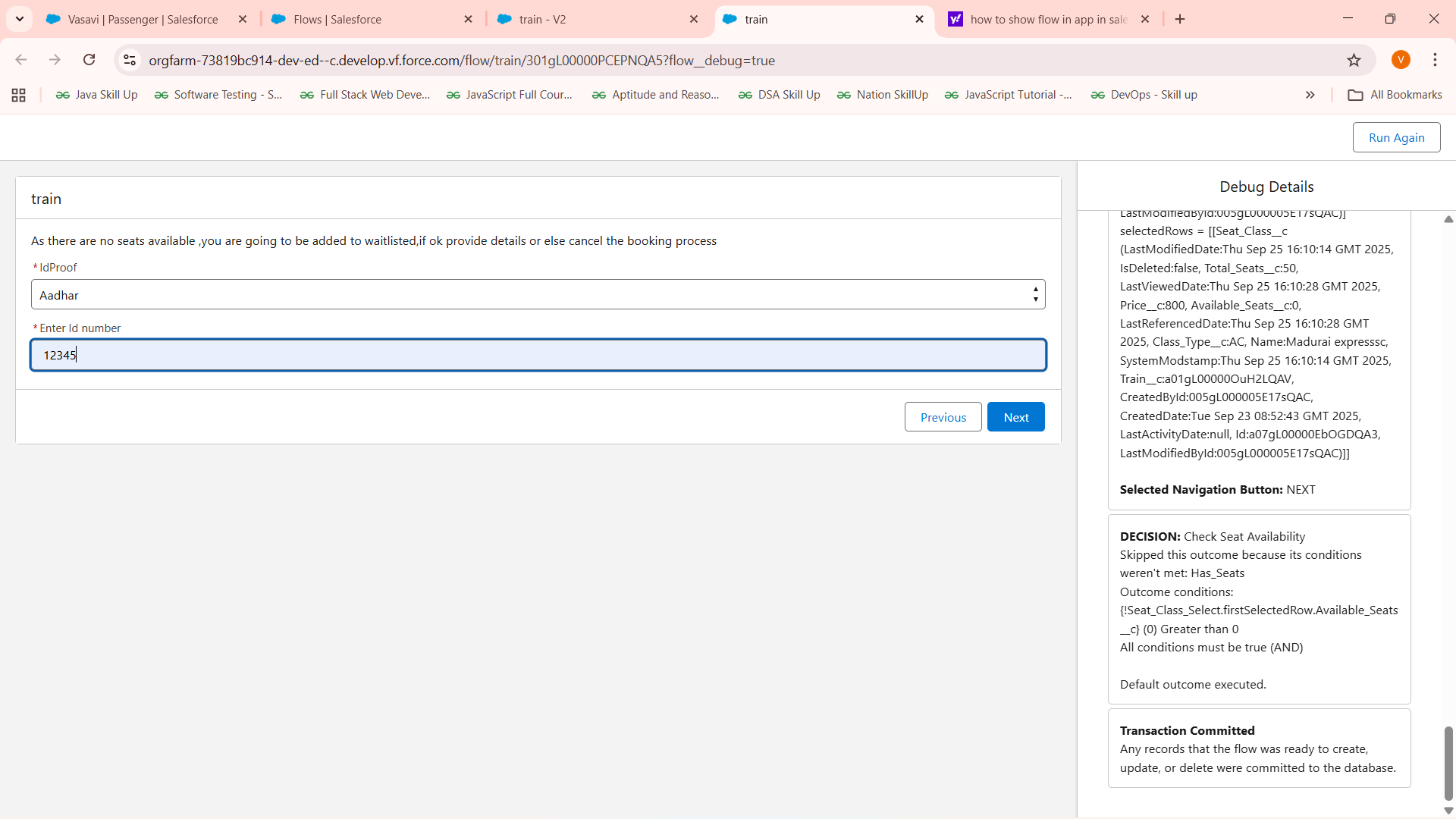
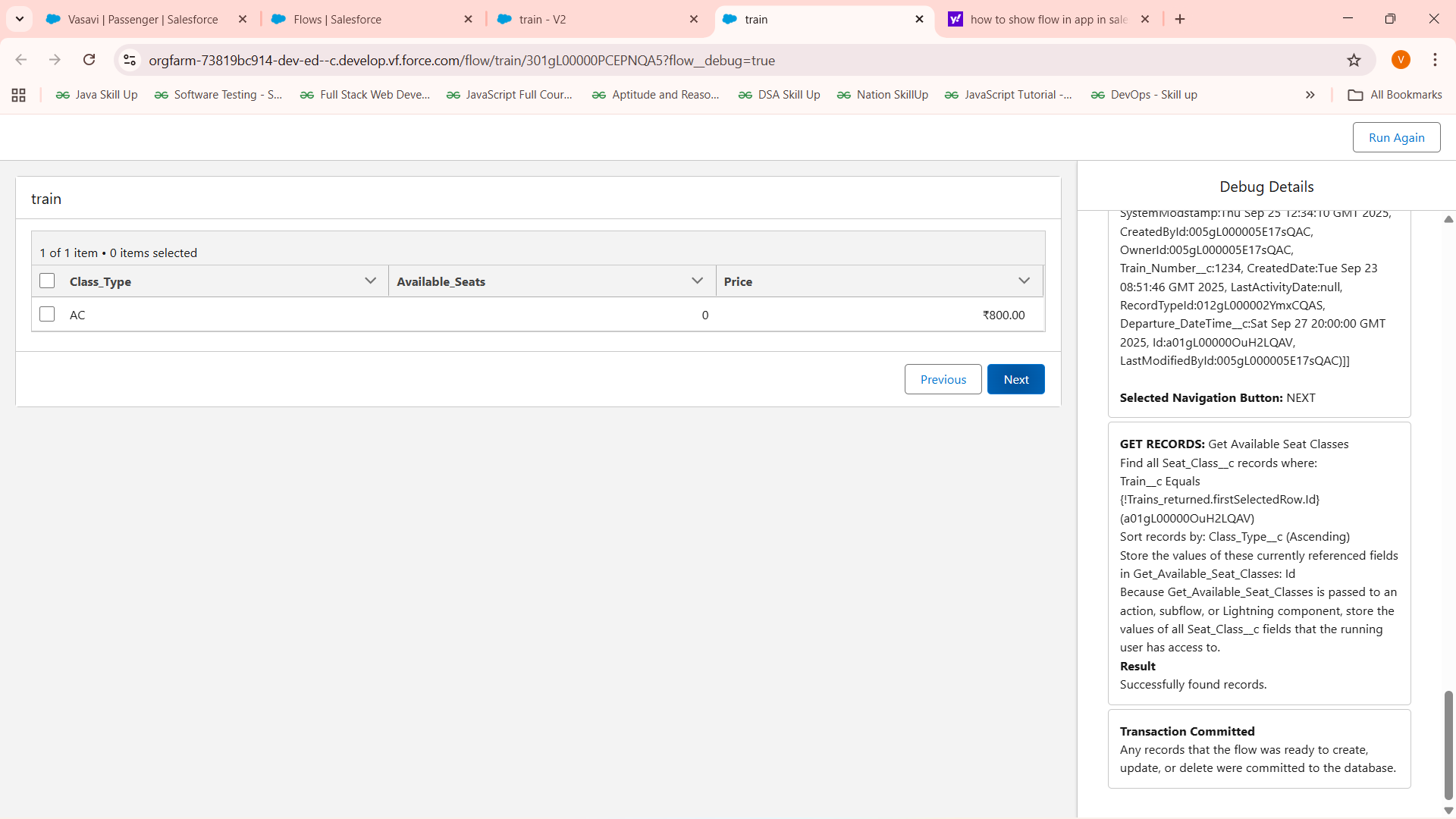
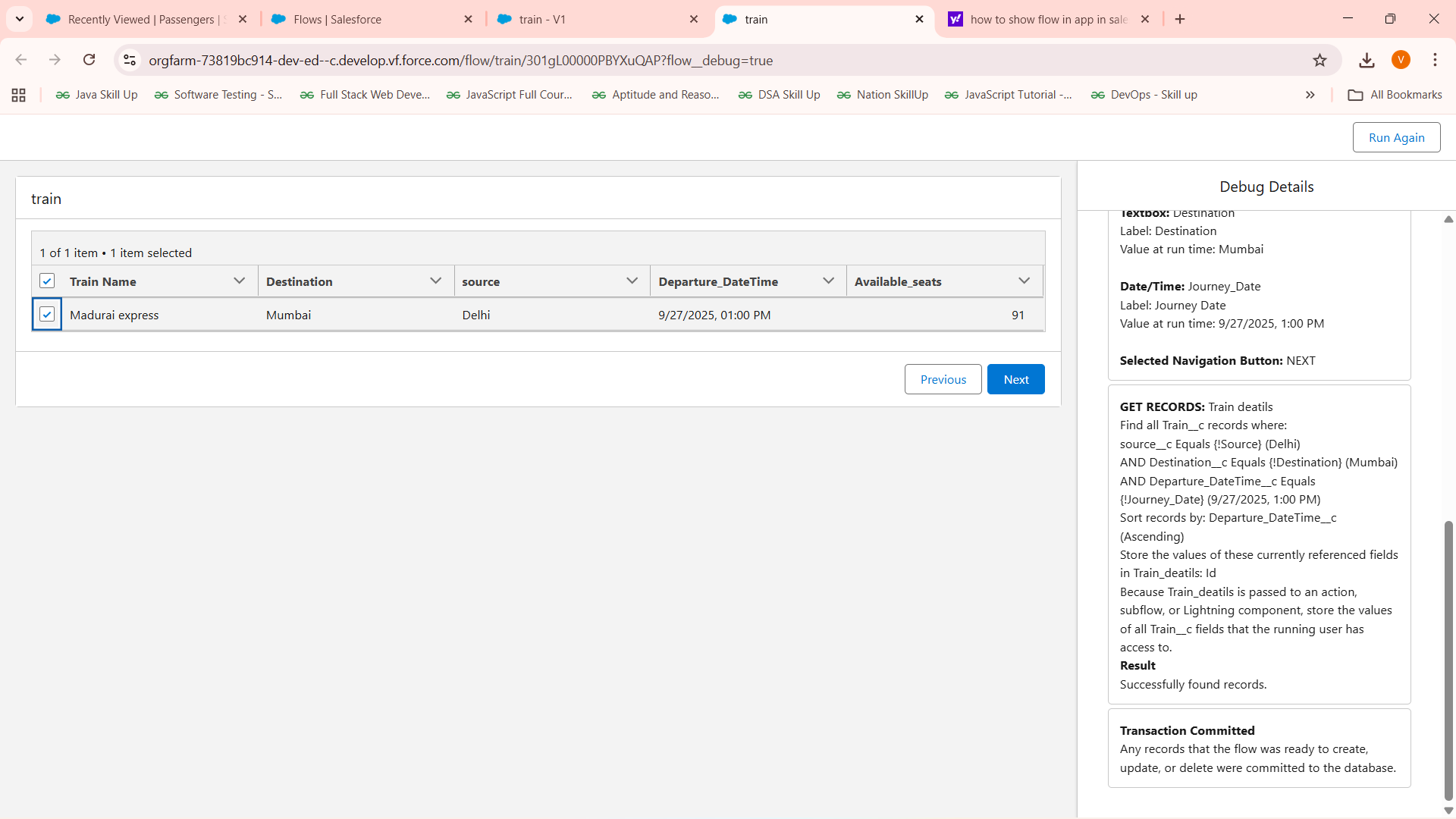
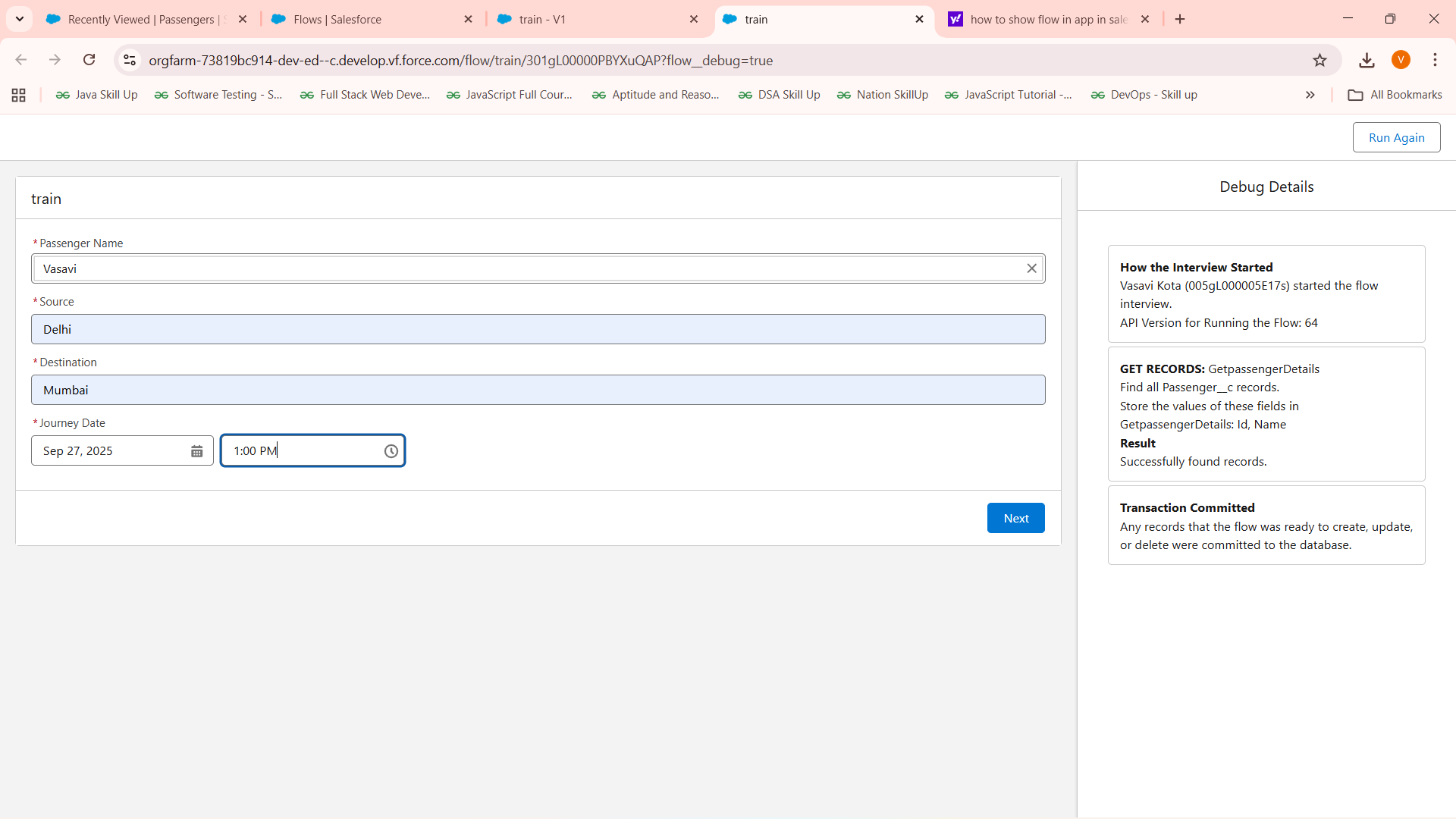
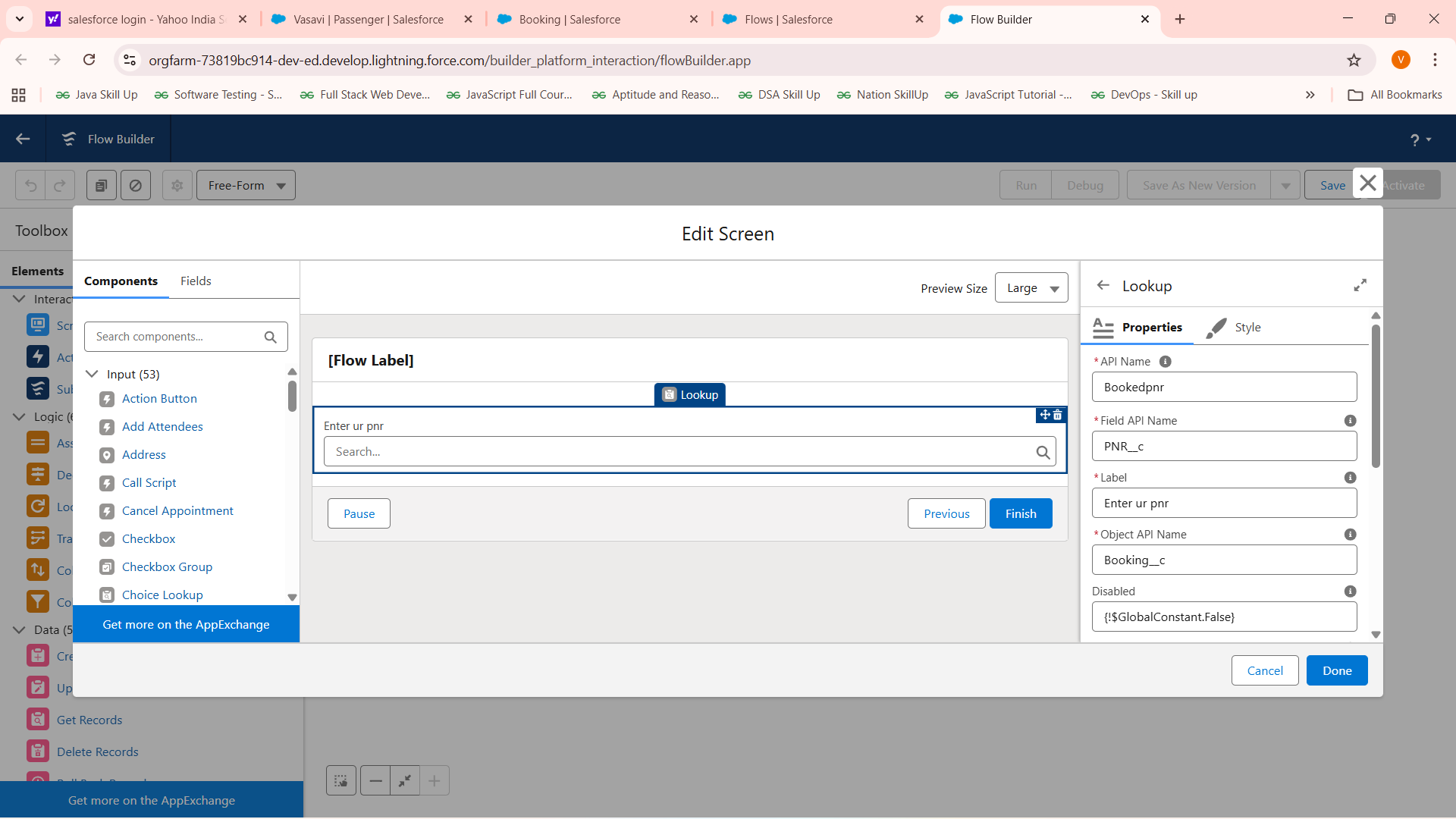
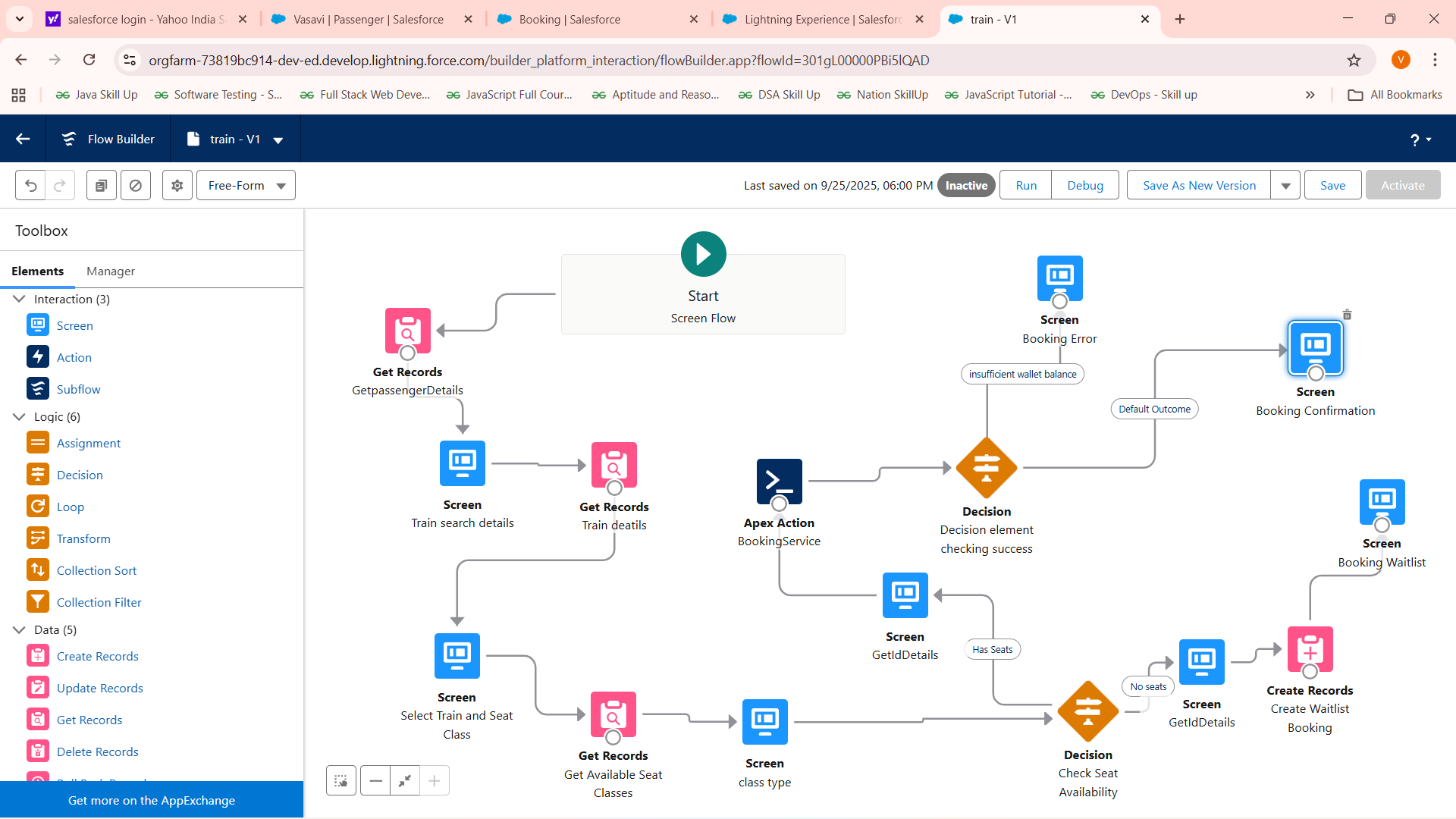
Presented By:

Vasavi Kota

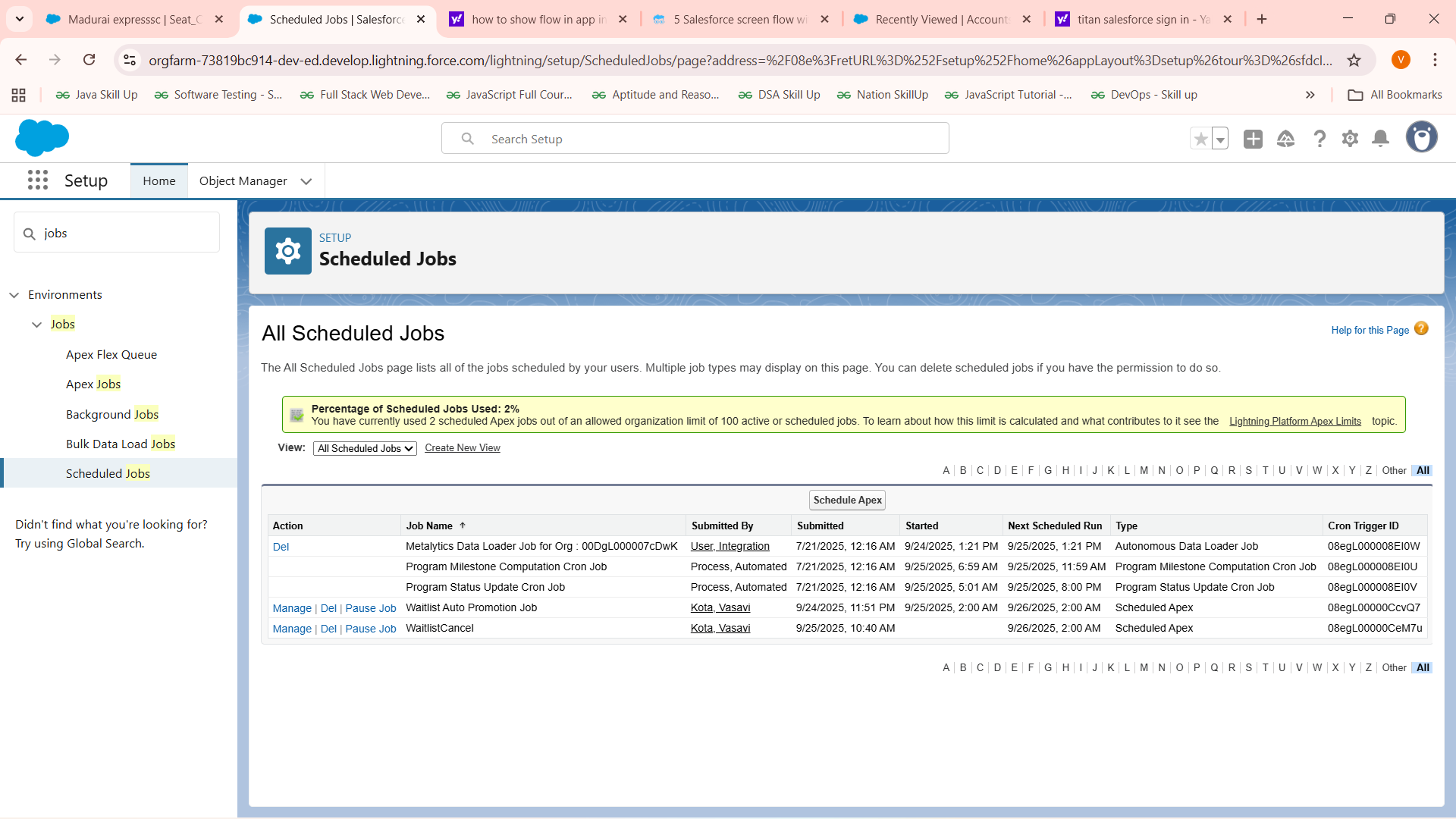
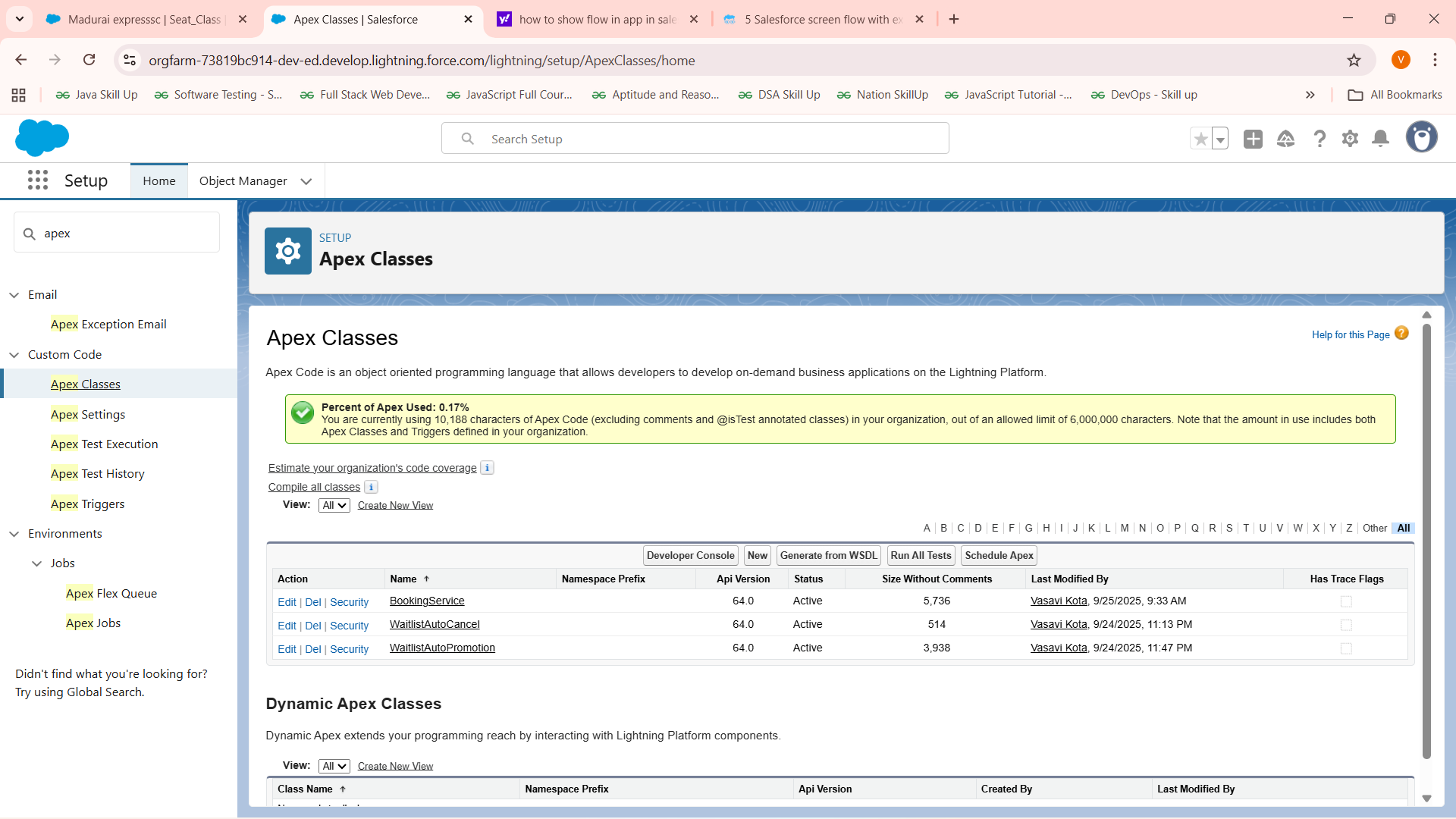
CMRCET,Hyderabad

# Phase 4: Process Automation (Admin)

* Validation Rules
* Workflow Rules
* Process Builder
* Approval Process
* Flow Builder (Screen, Record-Triggered, Scheduled, Auto-launched)
* Email Alerts
* Field Updates
* Tasks
* Custom Notifications



# Phase 5: Apex Programming (Developer)

* Classes & Objects
* Apex Triggers (before/after insert/update/delete)
* Trigger Design Pattern
* SOQL & SOSL
* Collections: List, Set, Map
* Control Statements
* Batch Apex
* Queueable Apex
* Scheduled Apex
* Future Methods
* Exception Handling
* Test Classes
* Asynchronous Processing
* 

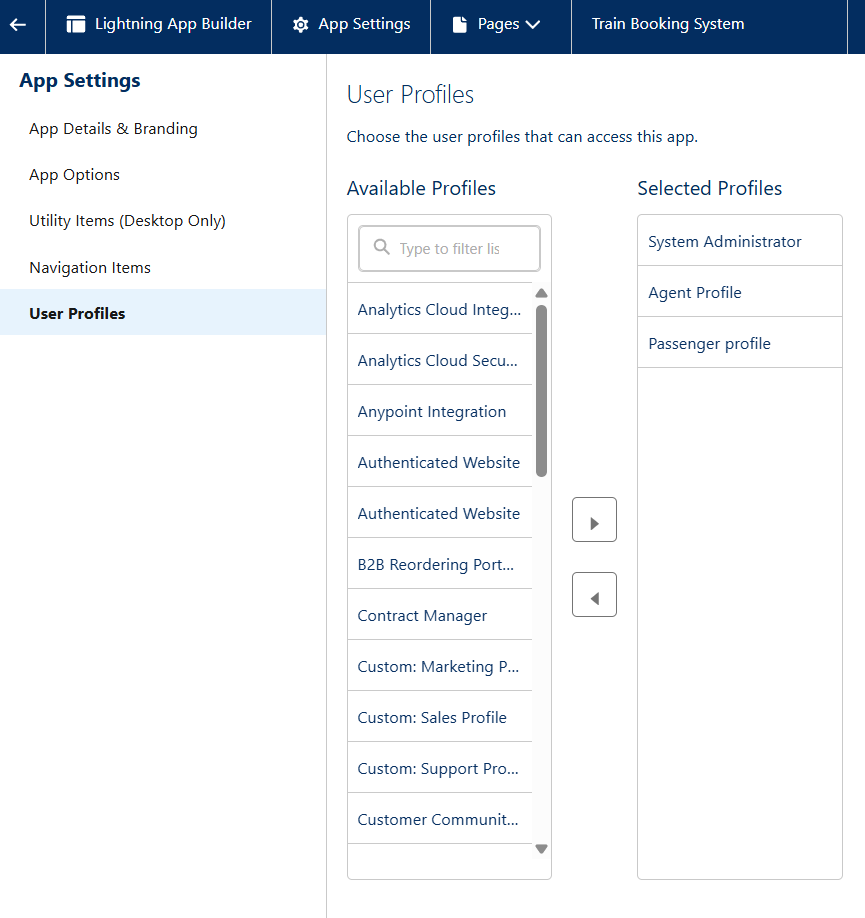
public class BookingService {  
    @AuraEnabled  
    public static String cancelBooking(Id bookingId) {  
        Booking\_\_c b = [SELECT Id, Passenger\_\_c, Amount\_Deducted\_\_c, Status\_\_c, Journey\_Date\_\_c FROM Booking\_\_c WHERE Id = :bookingId FOR UPDATE];  
        if (b.Status\_\_c == 'Cancelled') {  
            return 'Booking already cancelled.';  
        }  
  
        Date today = Date.today();  
        Integer daysToJourney = b.Journey\_Date\_\_c.daysBetween(today);  
  
        if (daysToJourney < 2) {  
            return 'Cannot cancel booking less than 2 days before journey.';  
        }  
  
        b.Status\_\_c = 'Cancelled';  
  
        Passenger\_\_c p = [SELECT Id, Wallet\_Balance\_\_c FROM Passenger\_\_c WHERE Id = :b.Passenger\_\_c FOR UPDATE];  
  
        if (b.Amount\_Deducted\_\_c != null && b.Amount\_Deducted\_\_c > 0) {  
            p.Wallet\_Balance\_\_c += b.Amount\_Deducted\_\_c;  
  
            Wallet\_Transaction\_\_c wt = new Wallet\_Transaction\_\_c(  
                Passenger\_\_c = p.Id,  
                Amount\_\_c = b.Amount\_Deducted\_\_c,  
                Transaction\_Type\_\_c = 'Refund',  
                Transaction\_Date\_\_c = Datetime.now(),  
                Balance\_After\_\_c = p.Wallet\_Balance\_\_c,  
                Reference\_\_c = b.Name  
            );  
            update p;  
            insert wt;  
        }  
  
        update b;  
        return 'Booking cancelled successfully.';  
    }  
  
    public class BookingResult {  
        @InvocableVariable public Boolean success;  
        @InvocableVariable public String message;  
        @InvocableVariable public String pnr;  
        @InvocableVariable public String seatNumber;  
    }  
  
    @InvocableMethod(label='Create Booking')  
    public static List<BookingResult> createBooking(List<Request> requests){  
        List<BookingResult> results = new List<BookingResult>();  
        for(Request req : requests){  
            BookingResult br = new BookingResult();  
            br.success = false;  
            Passenger\_\_c p = [SELECT Id, Wallet\_Balance\_\_c FROM Passenger\_\_c WHERE Id = :req.passengerId FOR UPDATE];  
            Seat\_Class\_\_c sc = [SELECT Id, Available\_Seats\_\_c, Price\_\_c, Train\_\_c, Total\_Seats\_\_c FROM Seat\_Class\_\_c WHERE Id = :req.seatClassId FOR UPDATE];  
            Train\_\_c t = [SELECT Id, Available\_Seats\_\_c, Price\_\_c FROM Train\_\_c WHERE Id = :sc.Train\_\_c FOR UPDATE];  
            Decimal fare = sc.Price\_\_c != null ? sc.Price\_\_c : t.Price\_\_c;  
            if(sc.Available\_Seats\_\_c > 0 && t.Available\_Seats\_\_c > 0){  
                if(p.Wallet\_Balance\_\_c >= fare){  
                    // Assign seat number randomly  
                    Set<Integer> bookedSeats = new Set<Integer>();  
                    for(Booking\_\_c prev : [SELECT Seat\_Number\_\_c FROM Booking\_\_c WHERE Seat\_Class\_\_c = :sc.Id AND Journey\_Date\_\_c = :req.journeyDate]) {  
                        if(prev.Seat\_Number\_\_c != null) bookedSeats.add(Integer.valueOf(prev.Seat\_Number\_\_c));  
                    }  
                    List<Integer> availableSeats = new List<Integer>();  
                    for(Integer i = 1; i <= sc.Total\_Seats\_\_c; i++) {  
                        if(!bookedSeats.contains(i)) availableSeats.add(i);  
                    }  
                    Integer seatNumber = availableSeats.size() > 0 ? availableSeats[Math.mod(Math.abs(Crypto.getRandomInteger()), availableSeats.size())] : null;  
  
                    p.Wallet\_Balance\_\_c -= fare;  
                    Wallet\_Transaction\_\_c wt = new Wallet\_Transaction\_\_c(  
                        Passenger\_\_c = p.Id,  
                        Amount\_\_c = -fare,  
                        Transaction\_Type\_\_c = 'Deduction',  
                        Transaction\_Date\_\_c = Datetime.now(),  
                        Balance\_After\_\_c = p.Wallet\_Balance\_\_c,  
                        Reference\_\_c = 'Booking'  
                    );  
                    Booking\_\_c b = new Booking\_\_c(  
                        Passenger\_\_c = p.Id,  
                        Train\_\_c = t.Id,  
                        Seat\_Class\_\_c = sc.Id,  
                        Status\_\_c = 'Confirmed',  
                        Journey\_Date\_\_c = req.journeyDate,  
                        Price\_\_c = fare,  
                        Amount\_Deducted\_\_c = fare,  
                        Seat\_Number\_\_c = String.valueOf(seatNumber)  
                    );  
                    sc.Available\_Seats\_\_c -= 1;  
                    t.Available\_Seats\_\_c -= 1;  
                    update p;  
                    insert wt;  
                    insert b;  
                    // Query inserted booking to get generated PNR (Name)  
                    Booking\_\_c booked = [SELECT Name FROM Booking\_\_c WHERE Id = :b.Id];  
  
                    update sc;  
                    update t;  
  
                    br.pnr = booked.Name;  
                    br.seatNumber = String.valueOf(seatNumber);  
                    br.success = true;  
                    br.message = 'Booking Confirmed: ' + booked.Name + ', Seat Number: ' + seatNumber;  
                } else {  
                    br.message = 'Insufficient wallet balance.';  
                }  
            } else {  
                Booking\_\_c wb = new Booking\_\_c(  
                    Passenger\_\_c = p.Id,  
                    Train\_\_c = t.Id,  
                    Seat\_Class\_\_c = sc.Id,  
                    Status\_\_c = 'Waitlist',  
                    Journey\_Date\_\_c = req.journeyDate,  
                    Price\_\_c = fare,  
                    Amount\_Deducted\_\_c = 0  
                );  
                insert wb;  
                br.success = true;  
                br.message = 'Added to waitlist. PNR: ' + wb.Name;  
                br.pnr = wb.Name;  
            }  
            results.add(br);  
        }  
        return results;  
    }  
  
    public class Request {  
        @InvocableVariable public Id passengerId;  
        @InvocableVariable public Id trainId;  
        @InvocableVariable public Id seatClassId;  
        @InvocableVariable public Date journeyDate;  
    }  
}

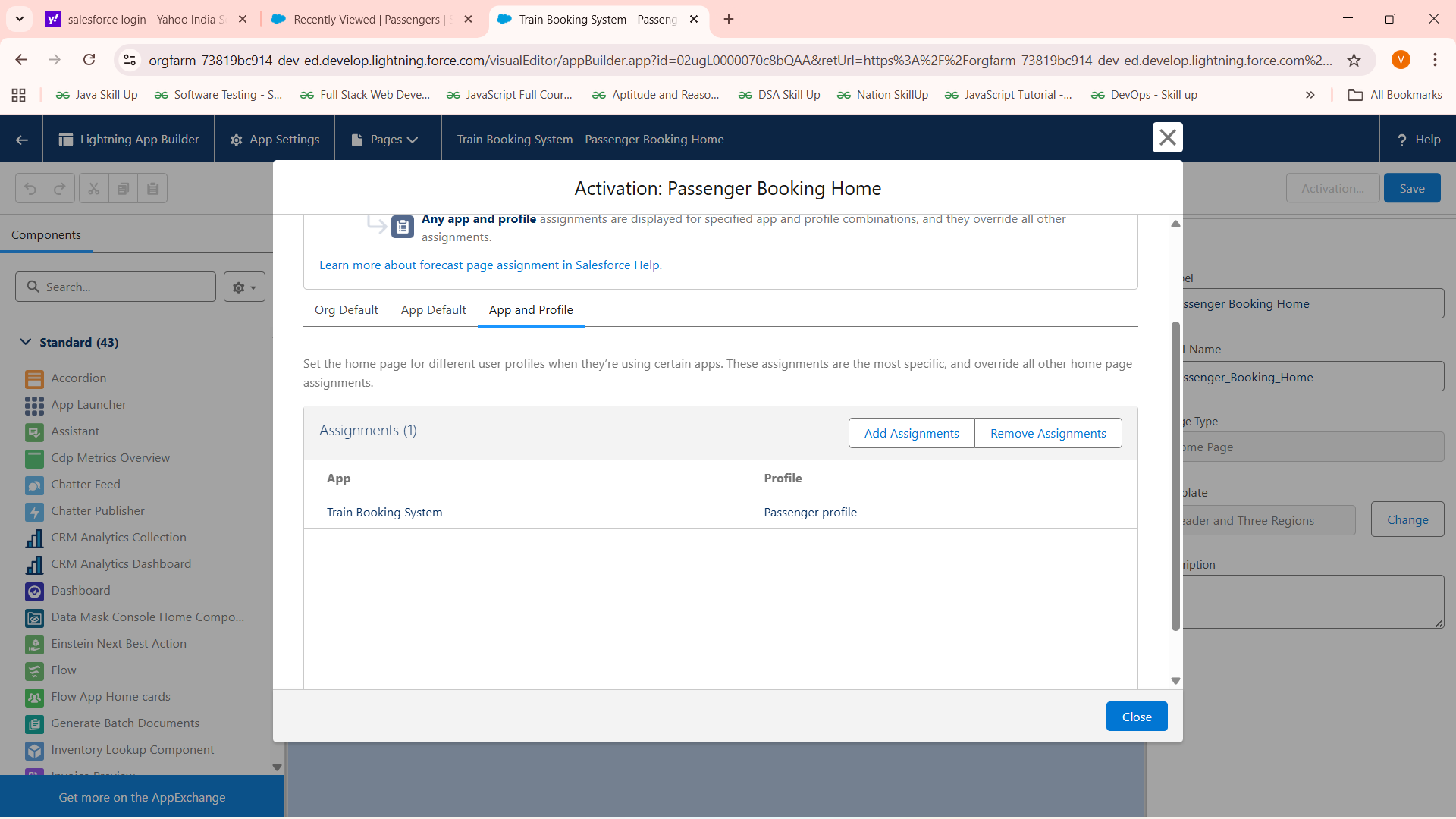
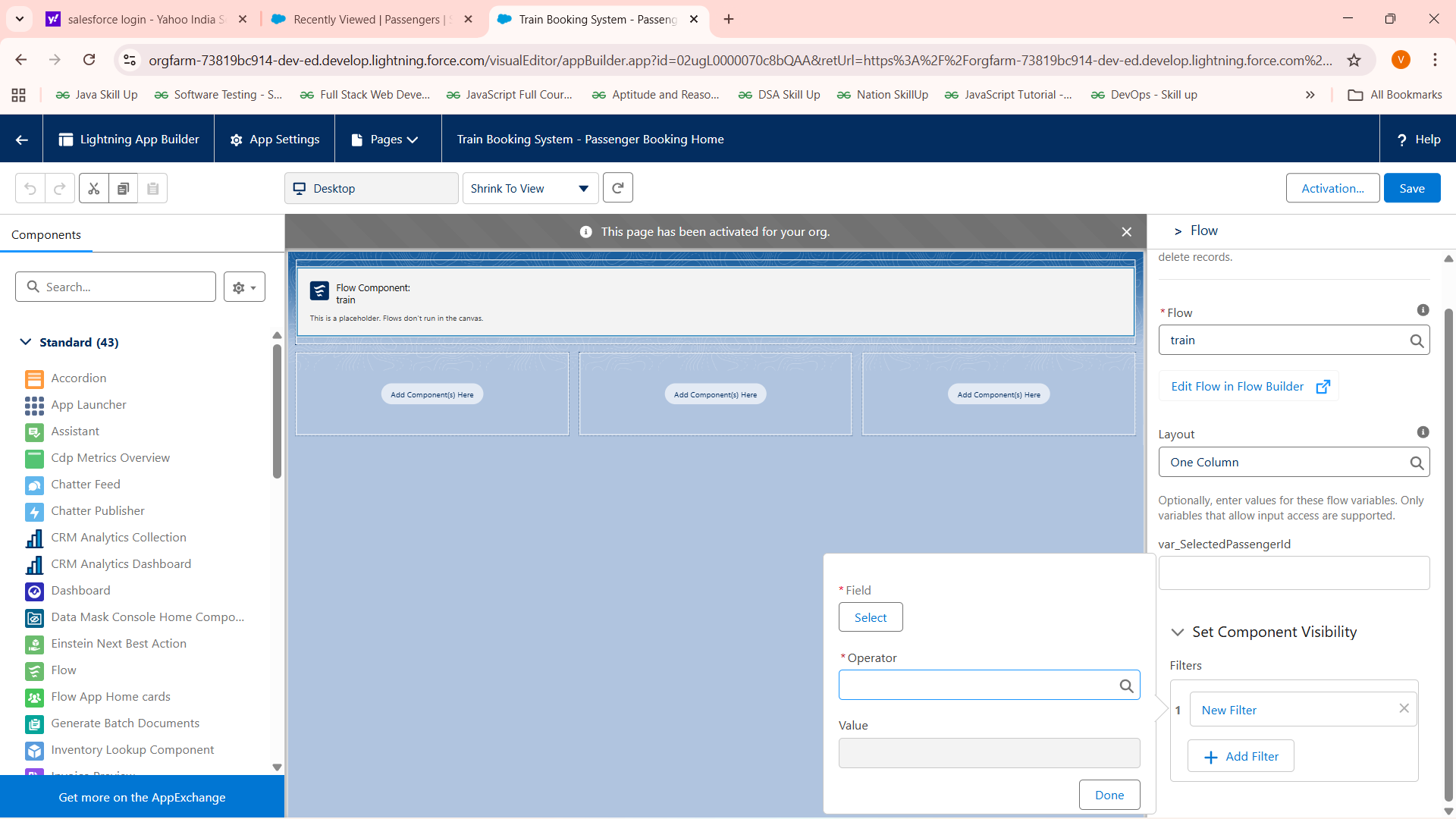
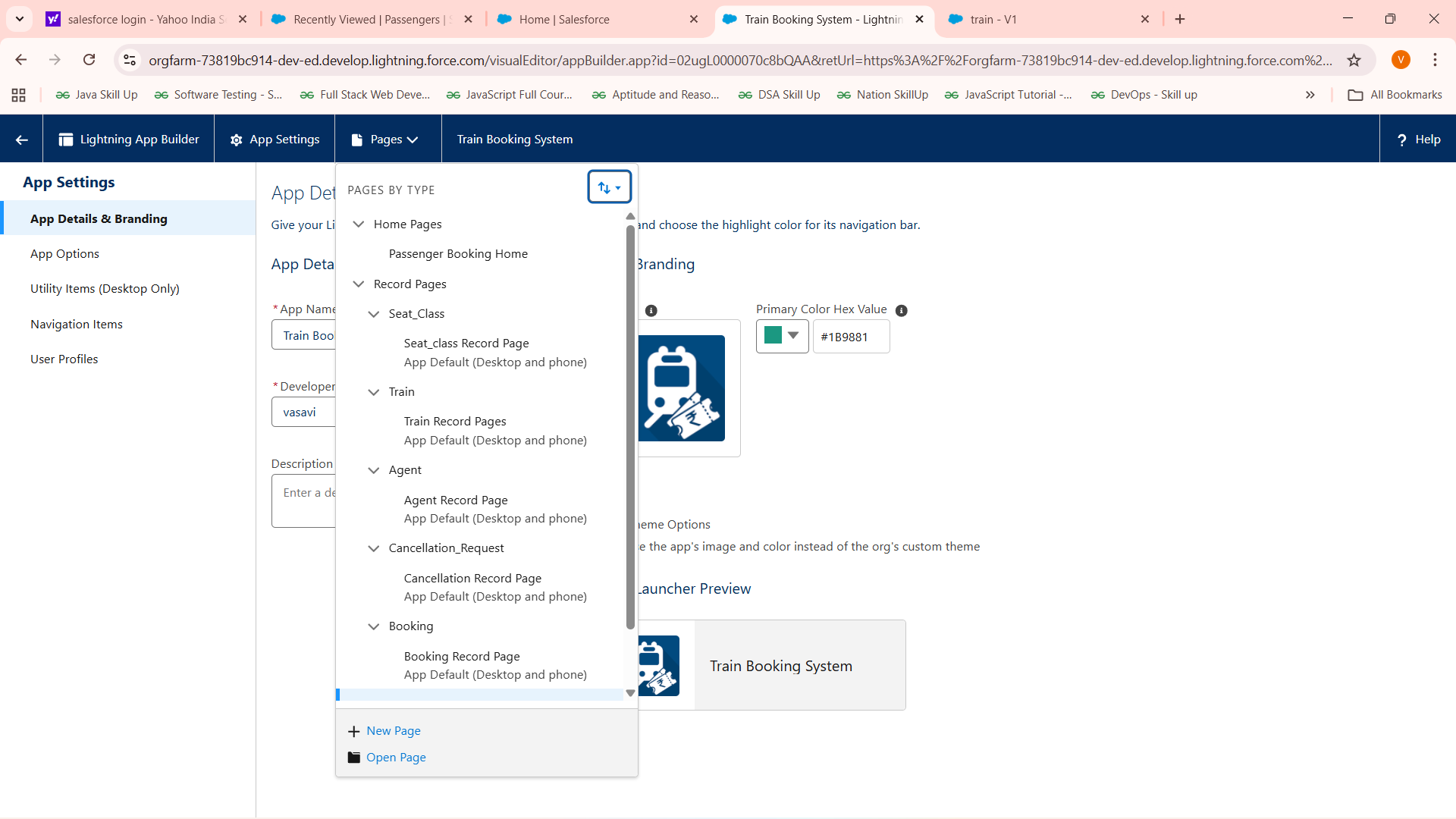
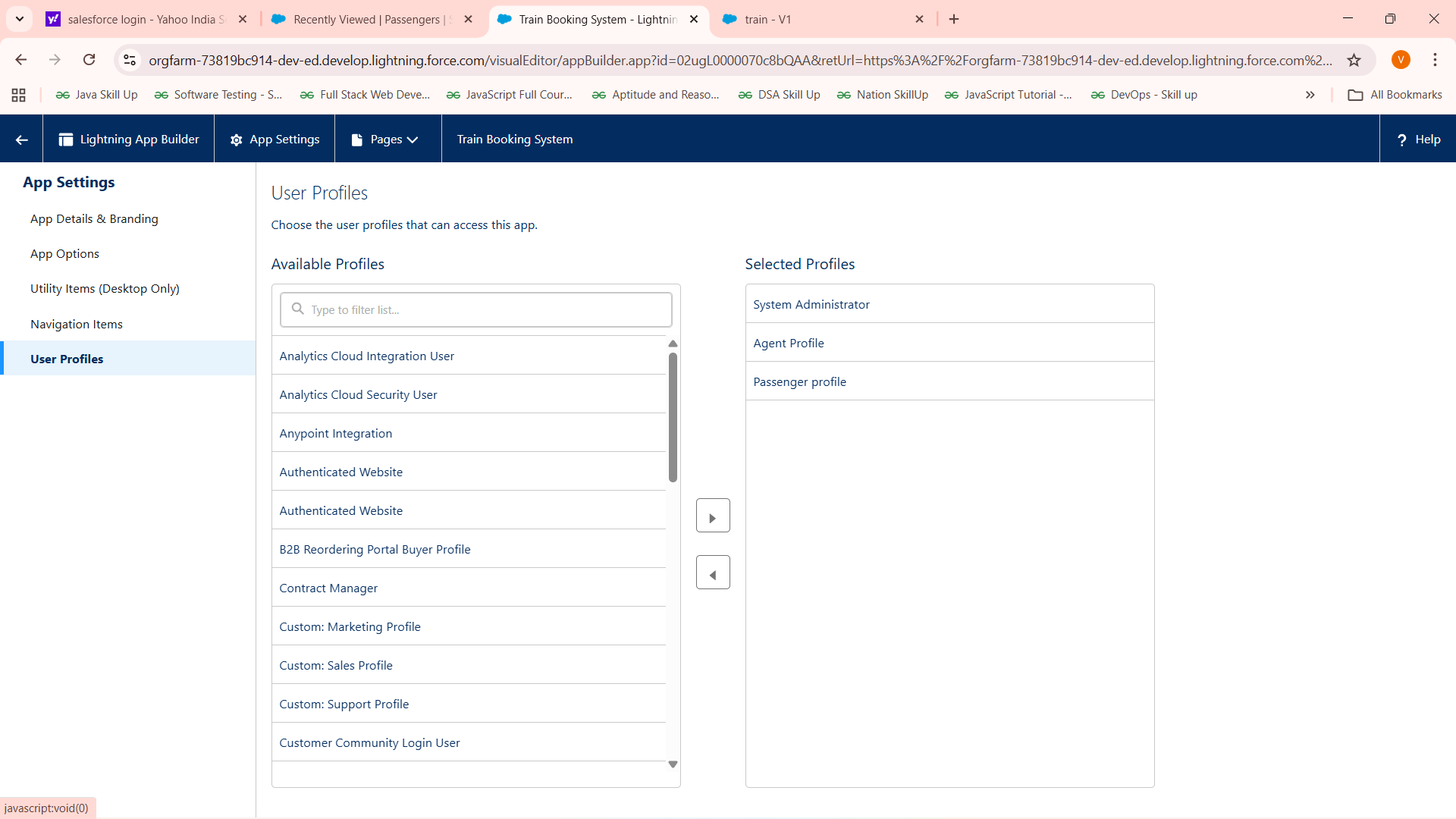
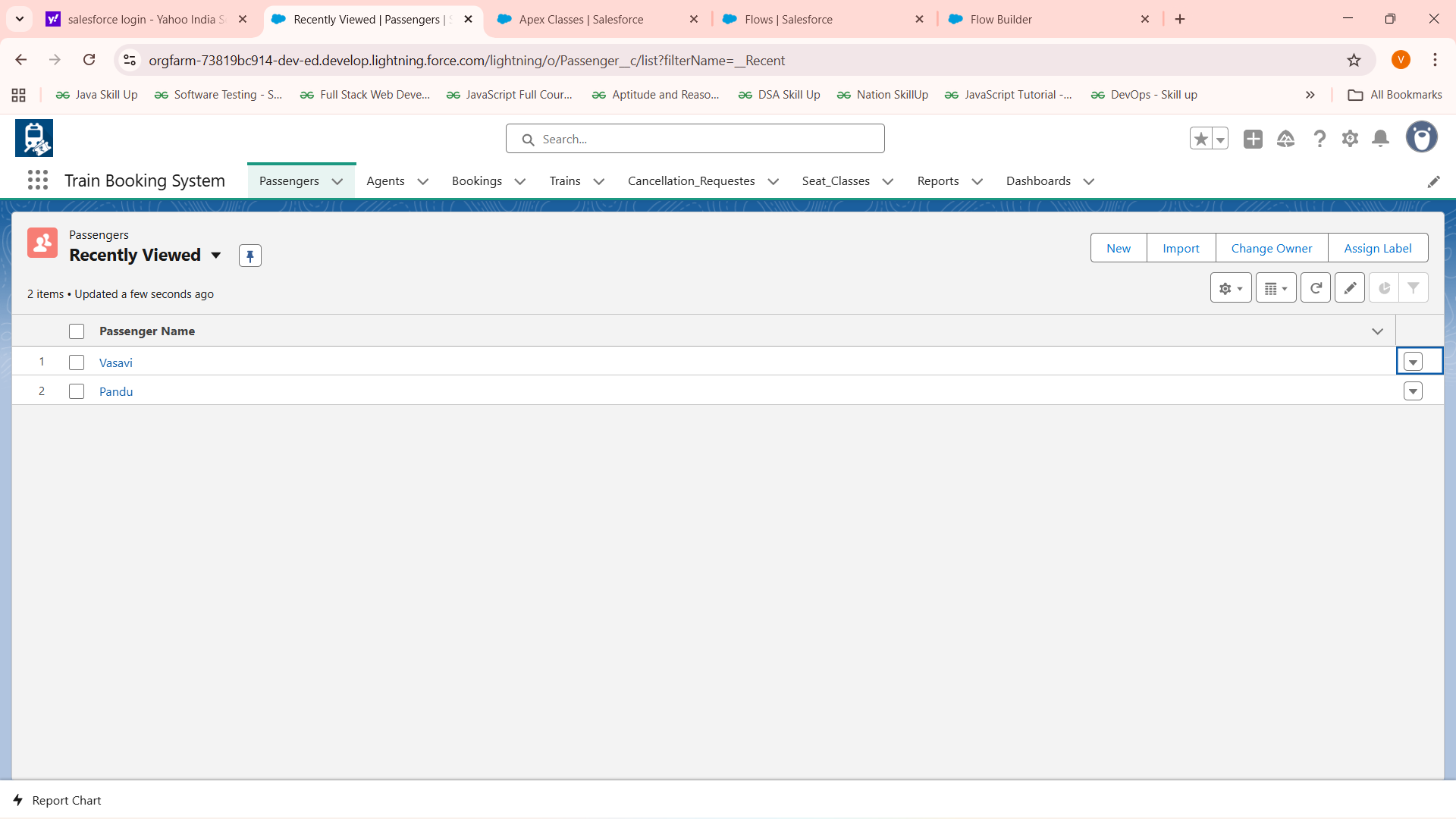
global class WaitlistAutoCancel implements Schedulable {  
  global void execute(SchedulableContext sc) {  
    Date checkDate = Date.today().addDays(1);  
    List<Booking\_\_c> waitlist = [SELECT Id, Passenger\_\_c, Price\_\_c FROM Booking\_\_c WHERE Status\_\_c = 'Waitlist' AND Journey\_Date\_\_c = :checkDate];  
    List<Booking\_\_c> toCancel = new List<Booking\_\_c>();  
    for(Booking\_\_c b : waitlist){  
      b.Status\_\_c = 'Cancelled';  
      toCancel.add(b);  
      // Optionally, call an email utility here to notify the passenger and create refund Wallet\_Transaction\_\_c if needed.  
    }  
    if(!toCancel.isEmpty()) {  
      update toCancel;  
      // Optionally, send notification emails or process further logic.  
    }  
  }  
}

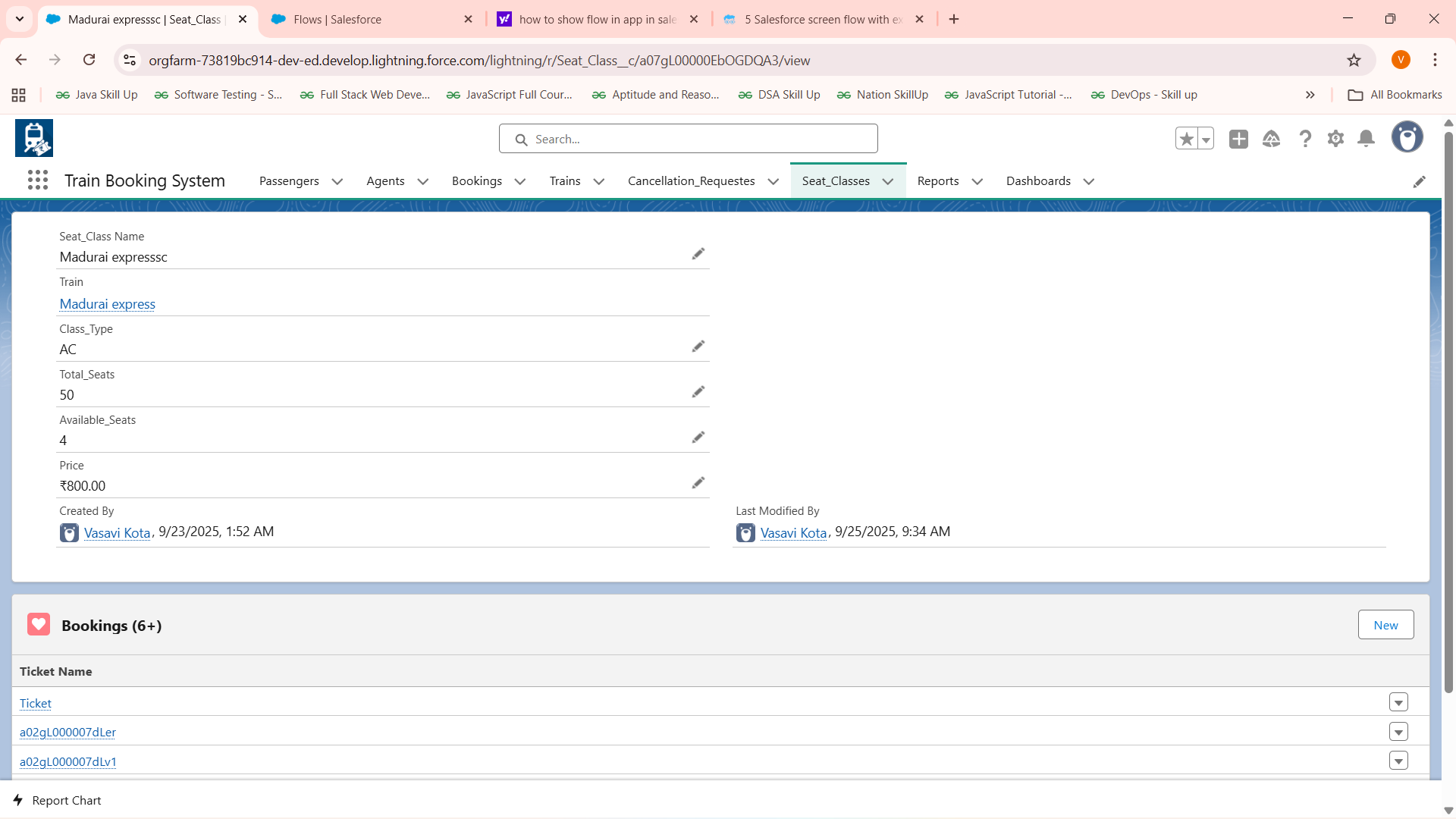
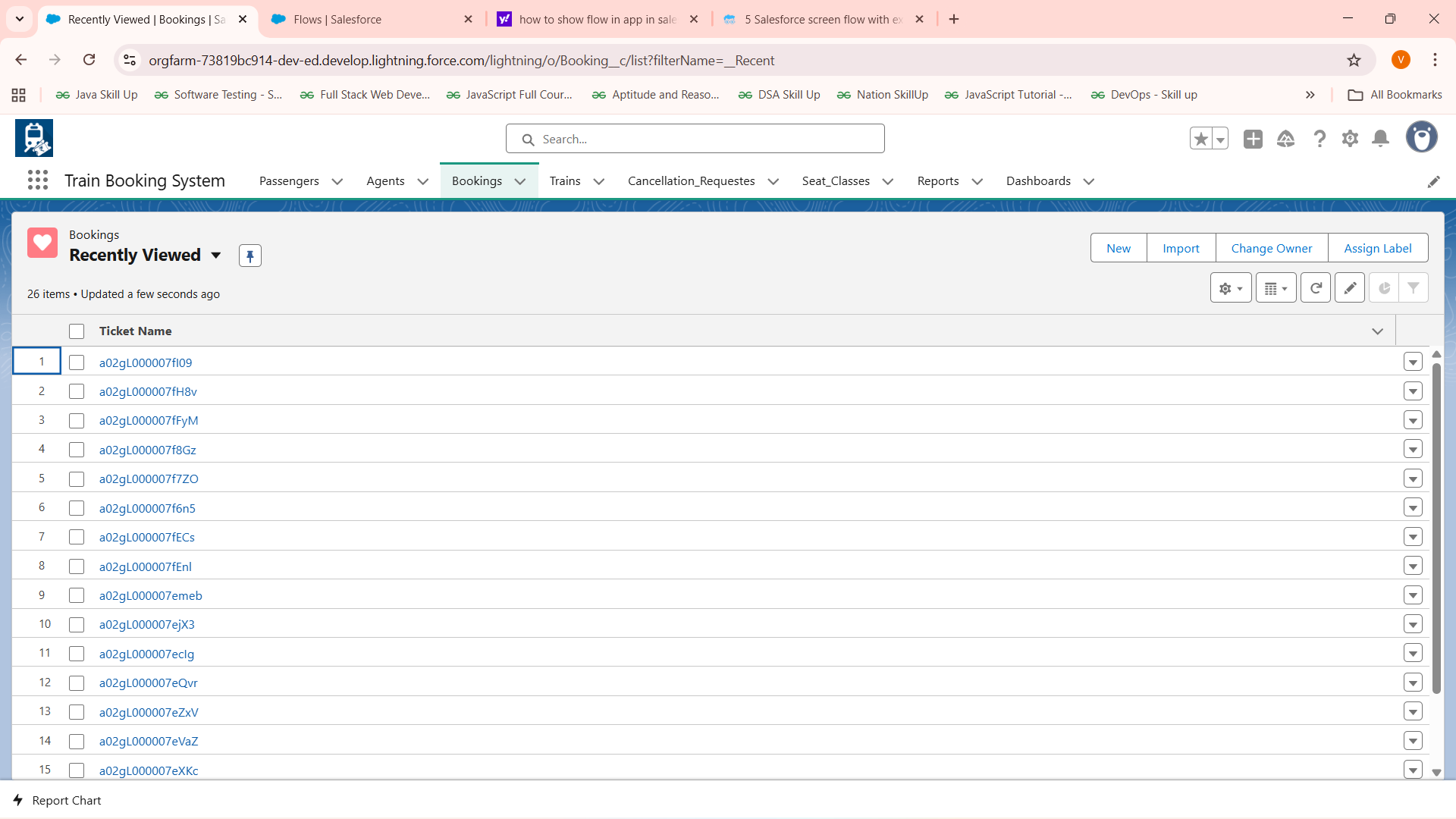
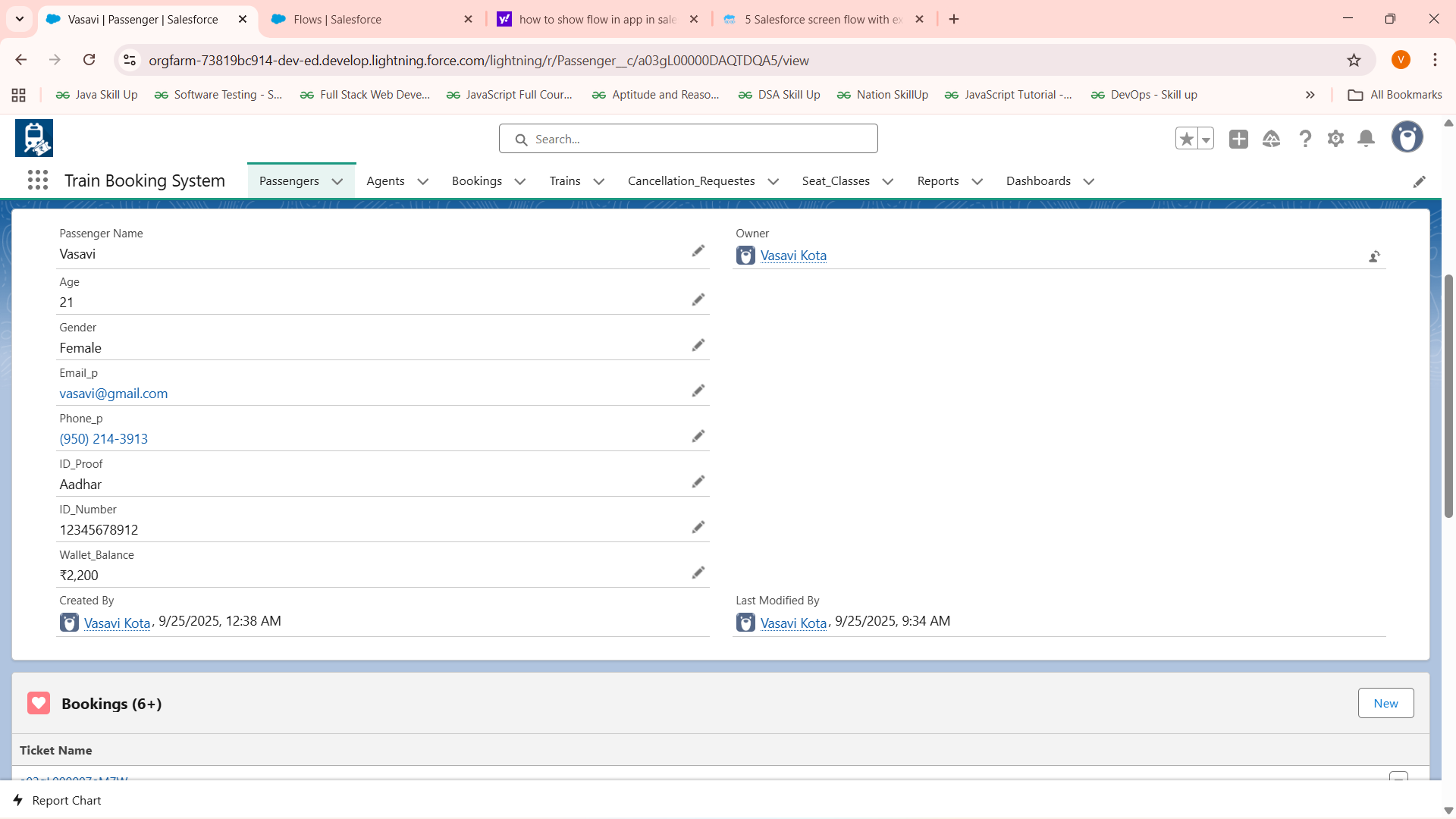
global class WaitlistAutoPromotion implements Schedulable {  
  
  global void execute(SchedulableContext sc) {  
    Date checkDate = Date.today().addDays(1); // day before journey  
  
    // Query all cancelled bookings freeing seats  
    List<Booking\_\_c> cancelledBookings = [  
      SELECT Train\_\_c, Seat\_Class\_\_c, Journey\_Date\_\_c   
      FROM Booking\_\_c   
      WHERE Status\_\_c = 'Cancelled' AND Journey\_Date\_\_c = :checkDate  
    ];  
  
    // Map to count freed seats by train + seat class  
    Map<String, Integer> freedSeatsCount = new Map<String, Integer>();  
    for (Booking\_\_c b : cancelledBookings) {  
      String key = b.Train\_\_c + ':' + b.Seat\_Class\_\_c;  
        Integer currentCount = freedSeatsCount.containsKey(key) ? freedSeatsCount.get(key) : 0;  
freedSeatsCount.put(key, currentCount + 1);  
  
    }  
  
    // Process each train + seat class combination   
    for (String key : freedSeatsCount.keySet()) {  
      String[] parts = key.split(':');  
      Id trainId = parts[0];  
      Id seatClassId = parts[1];  
      Integer seatsToPromote = freedSeatsCount.get(key);  
  
      // Query waitlist bookings FIFO order  
      List<Booking\_\_c> waitlistBookings = [SELECT Id, Passenger\_\_c, Journey\_Date\_\_c FROM Booking\_\_c   
                                          WHERE Status\_\_c = 'Waitlist' AND Train\_\_c = :trainId AND Seat\_Class\_\_c = :seatClassId   
                                          AND Journey\_Date\_\_c = :checkDate ORDER BY CreatedDate ASC LIMIT :seatsToPromote];  
  
      // Bulk query passengers and seat class/train details for efficiency  
      Set<Id> passengerIds = new Set<Id>();  
      for(Booking\_\_c wb : waitlistBookings) passengerIds.add(wb.Passenger\_\_c);  
      Map<Id, Passenger\_\_c> passengers = new Map<Id, Passenger\_\_c>([  
        SELECT Id, Wallet\_Balance\_\_c FROM Passenger\_\_c WHERE Id IN :passengerIds FOR UPDATE  
      ]);  
      Seat\_Class\_\_c scs = [SELECT Id, Available\_Seats\_\_c, Train\_\_c, Total\_Seats\_\_c, Price\_\_c FROM Seat\_Class\_\_c WHERE Id = :seatClassId FOR UPDATE];  
      Train\_\_c train = [SELECT Id, Available\_Seats\_\_c, Price\_\_c FROM Train\_\_c WHERE Id = :trainId FOR UPDATE];  
  
      Decimal fare = scs.Price\_\_c != null ? scs.Price\_\_c : train.Price\_\_c;  
      List<Booking\_\_c> toUpdateBookings = new List<Booking\_\_c>();  
      List<Wallet\_Transaction\_\_c> walletTransactions = new List<Wallet\_Transaction\_\_c>();  
  
      Integer seatsPromoted = 0;  
      for (Booking\_\_c waitlist : waitlistBookings) {  
        Passenger\_\_c p = passengers.get(waitlist.Passenger\_\_c);  
        if (p != null && p.Wallet\_Balance\_\_c >= fare && seatsPromoted < seatsToPromote) {  
  
          // Find and assign next available seat number  
          Set<Integer> bookedSeats = new Set<Integer>();  
          for(Booking\_\_c b : [SELECT Seat\_Number\_\_c FROM Booking\_\_c WHERE Seat\_Class\_\_c = :scs.Id AND Journey\_Date\_\_c = :waitlist.Journey\_Date\_\_c AND Status\_\_c='Confirmed']) {  
            if (b.Seat\_Number\_\_c != null) bookedSeats.add(Integer.valueOf(b.Seat\_Number\_\_c));  
          }  
          List<Integer> availableSeats = new List<Integer>();  
          for (Integer i = 1; i <= scs.Total\_Seats\_\_c; i++) {  
            if (!bookedSeats.contains(i)) availableSeats.add(i);  
          }  
          if (availableSeats.isEmpty()) break; // no seats to assign, break  
  
          Integer seatNumber = availableSeats[Math.mod(Math.abs(Crypto.getRandomInteger()), availableSeats.size())];  
  
          // Deduct fare from wallet  
          p.Wallet\_Balance\_\_c -= fare;  
  
          // Update booking to confirmed  
          waitlist.Status\_\_c = 'Confirmed';  
          waitlist.Amount\_Deducted\_\_c = fare;  
          waitlist.Price\_\_c = fare;  
          waitlist.Seat\_Number\_\_c = String.valueOf(seatNumber);  
  
          // Track records for update/insert  
          toUpdateBookings.add(waitlist);  
          walletTransactions.add(new Wallet\_Transaction\_\_c(  
            Passenger\_\_c = p.Id,  
            Amount\_\_c = -fare,  
            Transaction\_Type\_\_c = 'Deduction',  
            Transaction\_Date\_\_c = Datetime.now(),  
            Balance\_After\_\_c = p.Wallet\_Balance\_\_c,  
            Reference\_\_c = 'Booking Promotion'  
          ));  
  
          seatsPromoted++;  
          scs.Available\_Seats\_\_c -= 1;  
          train.Available\_Seats\_\_c -= 1;  
        } else {  
          // Optionally notify passenger for insufficient balance or handle per policy  
        }  
      }  
  
      if (!toUpdateBookings.isEmpty()) {  
        update toUpdateBookings;  
        update new List<Passenger\_\_c>(passengers.values());  
        insert walletTransactions;  
        update scs;  
        update train;  
      }  
    }  
  }  
}

# Phase 6: User Interface Development

* Lightning App Builder
* Record Pages
* Tabs
* Home Page Layouts
* Utility Bar
* LWC (Lightning Web Components)
* Apex with LWC
* Events in LWC
* Wire Adapters
* Imperative Apex Calls
* Navigation Service







Overall summary:

**Phase 4: Process Automation (Admin)**

* Validation Rules: Implemented to enforce business rules like cancellation timing policies.
* Workflow Rules & Process Builder: Limited use; moved towards Flow Builder for UI and automation.
* Approval Process: Not implemented in current scope; can be added if needed.
* Flow Builder:
  + Created Screen Flows for Booking and Cancellation processes.
  + Used Record-Triggered and Scheduled Flows via Apex scheduled jobs for waitlist management.
* Email Alerts: Setup for booking confirmation, cancellation notifications (integration with Apex or Flow).
* Field Updates & Tasks: Integrated as part of Apex logic and Flow outcomes.
* Custom Notifications: Configured or considered for alerting users on booking statuses.

**Phase 5: Apex Programming (Developer)**

* Classes & Objects: Built core BookingService and Waitlist management Apex classes.
* Apex Triggers: Not extensively implemented; Apex classes handle logic.
* Trigger Design Patterns: Not covered but can be adopted for scale.
* SOQL & SOSL: Used for querying bookings, passengers, trains, seat classes efficiently.
* Collections: Utilized Lists, Sets, and Maps for data processing.
* Control Statements: Implemented complex booking logic and validations.
* Scheduled Apex: Developed WaitlistAutoCancel to cancel waitlisted tickets before 1 day from journeydate and WaitlistAutoPromotion for freed seats from cancellation to confirmed booking schedulable classes
* Exception Handling & Test Classes: Recommendations to enhance for production readiness.
* Asynchronous Processing: Scheduled Apex for batch/scheduled jobs.

**Phase 6: User Interface Development**

* Lightning App Builder: Used for building flow containers and app pages.
* Record Pages & Tabs: Setup for quick access to Booking and related objects.
* Home Page Layouts & Utility Bar: Customized for better UX.
* Lightning Web Components (LWC): Not implemented; but working on it will complete soon
* Apex with LWC Integration: Considered for complex UI functionality.
* Events, Wire Adapters, Imperative Apex Calls: Planned for advanced interaction.
* Navigation Service: Used for smooth UI transitions in Lightning apps.