TABLE OF CONTENTS

Salesforce Documentation: Tours and Travels CRM

Project Overview
The Tours & Travels CRM is a cloud-based Salesforce application built for travel agencies that provide group tours, solo trips, and honeymoon packages. It allows users, such as Travel Agents, Travel Agent Managers, and Administrators, to conveniently handle bookings, client information, payment transactions, feedback, and personnel assignments. The system centralizes operations and automates them using validation rules, Apex triggers, Lightning components, and approval processes, resulting in a comprehensive travel management solution from beginning to end.

Objectives
The primary objective of this CRM is to streamline travel booking operations while ensuring data accuracy, customer satisfaction, and operational efficiency. It achieves this by:

- Automating repetitive tasks like guest and payment creation
- Enforcing data validation rules (e.g., phone format, email syntax)
- Empowering managers to handle approvals through workflows
- Dynamically filtering UI options to enhance user experience
  This reduces manual workload, prevents human errors, and boosts overall productivity within the travel agency.

Phase 1: Requirement Analysis & Planning
Understanding Business Requirements

- Travel agencies need a unified platform to handle bookings, payments, and customer feedback.
- Customers should experience fast, error-free, and personalized service.
- Agents must track booking status, approvals, and traveler details.

Defining Project Scope and Objectives

- Integrate core entities: Customers, Bookings, Packages, Feedback, Employees
- Enable real-time automation using Apex triggers and Flow
- Build approval logic for cancellation handling

Design: Data Model and Security

- Custom Objects: Customer_Info__c, Employee__c, Booking__c, BookingGuest__c, Booking_Payment__c, Feedback__c, TravelPackage__c
- Security via Profiles: Travel Agent Manager, Travel Agent (Salesforce Platform licenses)
- Validation rules and field-level filters to maintain data integrity

Project Roadmap & Milestones

- Phase 1: Object design & validation rules
- Phase 2: Apex Triggers and Flow automation
- Phase 3: LWC & dynamic page updates
- Phase 4: Testing, deployment, and approval setup

Phase 2: Salesforce Development - Backend & Configurations
Validation Rules

- **Customer Info**
  - Phone must have 10 digits: NOT(REGEX(Phone__c, ".*([0-9].*){10,}"))
  - Email format: NOT(REGEX(Email__c, "^[a-zA-Z0-9._]+@[a-zA-Z0-9.]+\\.[a-zA-Z]{2,}$"))
  - Date of Birth: Date_Of_Birth__c > TODAY()
- **BookingGuest**
  - Age must be positive: Age__c <= 0
- **Employee**
  - Email required for Finance/Admin roles: AND(OR(ISPICKVAL(Role__c, "Finance Executive"), ISPICKVAL(Role__c, "Admin")), ISBLANK(Email__c))
  - Languages required for Guides: AND(ISPICKVAL(Role__c, "Guide"), ISBLANK(Languages_Spoken__c))
- **Booking**
  - Status on creation must be Pending: AND(ISNEW(), NOT(ISPICKVAL(Status__c, "Pending")))

BookingTrigger: Automatically handles post-booking automation. When a new Booking__c record is inserted, this trigger fires to

```
trigger BookingTrigger on Booking__c (after insert) {
   if (Trigger.isAfter && Trigger.isInsert) {
      BookingTriggerHandler.createPaymentRecords(Trigger.new);
      BookingTriggerHandler.createBookingGuests(Trigger.new);
   }
}
```

Apex Handler: Encapsulates logic from the BookingTrigger for better separation of concerns and easier maintenance.

```
public class BookingTriggerHandler {
   public static void createPaymentRecords(List<Booking__c> bookings) {
      List<Booking_Payment__c> payments = new List<Booking_Payment__c>();
      for (Booking__c b : bookings) {
         payments.add(new Booking_Payment__c(
            Booking__c = b.Id,
            Payment_Status__c = 'Pending'
         ));
      }
      if (!payments.isEmpty()) insert payments;
   }

   public static void createBookingGuests(List<Booking__c> bookings) {
      List<BookingGuest__c> guests = new List<BookingGuest__c>();
      for (Booking__c b : bookings) {
         for (Integer i = 1; i <= b.Number_of_Travelers__c; i++) {
            guests.add(new BookingGuest__c(
               Booking__c = b.Id,
               Name = 'Guest ' + i
            ));
         }
      }
      if (!guests.isEmpty()) insert guests;
   }
}
```

BookingConfirmationEmailer: Sends a confirmation email to the customer after a booking is successfully created. This is done using a **@future** method to handle email sending asynchronously, improving performance and avoiding limits.

```
public class BookingConfirmationEmailer {

   @future(callout=false)

   public static void sendBookingConfirmation(Set<Id> bookingIds) {

      List<Messaging.SingleEmailMessage> emails = new
List<Messaging.SingleEmailMessage>();
```

```
    List<Booking__c> bookings = [SELECT Id, Name, Customer_Email__c,
Total_Billing_Amount__c

                                 FROM Booking__c

                                 WHERE Id IN :bookingIds];

    for (Booking__c booking : bookings) {

        if (String.isNotBlank(booking.Customer_Email__c)) {

            Messaging.SingleEmailMessage mail = new
Messaging.SingleEmailMessage();

            mail.setToAddresses(new String[] { booking.Customer_Email__c });

            mail.setSubject('Booking Confirmed: ' + booking.Name);

            mail.setPlainTextBody(

                'Dear Customer,' + '\n\n' +

                'Your booking has been confirmed. Please find the details below:\n' +

                'Booking ID: ' + booking.Name + '\n' +

                'Total Bill Amount Paid: $' + booking.Total_Billing_Amount__c +
'\n\n' +

                'Thank you for booking with us!'

            );

            emails.add(mail);

        }

    }

    if (!emails.isEmpty()) {

        Messaging.sendEmail(emails);
```

```
        }

    }

}
```

BookingReminderQueueable: Sends tour start reminder emails to customers whose travel dates are approaching. The logic is executed asynchronously using the Queueable interface to support large volumes and delayed execution.

```
public class BookingReminderQueueable implements Queueable {

    List<Booking__c> bookingsToRemind;

    public BookingReminderQueueable(List<Booking__c> bookings) {

        this.bookingsToRemind = bookings;

    }

    public void execute(QueueableContext context) {

        List<Messaging.SingleEmailMessage> emails = new
List<Messaging.SingleEmailMessage>();

        for (Booking__c booking : bookingsToRemind) {

            if (String.isNotBlank(booking.Customer_Email__c)) {

                Messaging.SingleEmailMessage mail = new
Messaging.SingleEmailMessage();

                mail.setToAddresses(new String[] { booking.Customer_Email__c });

                mail.setSubject('Reminder: Your Tour Starts Soon!');

                mail.setPlainTextBody(

                    'Hello,\n\nThis is a friendly reminder that your tour is starting on ' +

                    booking.Travelling_Start_Date__c.format() +
```

```
                '. Please make necessary arrangements.\n\nThank you for choosing
us!'
            );
            emails.add(mail);
        }
    }
    if (!emails.isEmpty()) {
        Messaging.sendEmail(emails);
    }
  }
}
```

BookingReminderScheduler: Automates reminders by scheduling the BookingReminderQueueable to run daily. It finds bookings with a travel date 3 days from today and queues them for email reminders if the status is Confirmed.

```
public class BookingReminderScheduler implements Schedulable {
  public void execute(SchedulableContext context) {
    Date reminderDate = Date.today().addDays(3);
    List<Booking__c> upcomingBookings = [
        SELECT Id, Travelling_Start_Date__c, Customer_Email__c
        FROM Booking__c
        WHERE Travelling_Start_Date__c = :reminderDate
        AND Booking_Status__c = 'Confirmed'
    ];
```

```
        if (!upcomingBookings.isEmpty()) {

            System.enqueueJob(new
BookingReminderQueueable(upcomingBookings));

        }

    }

}
```

PaymentReminderBatch: Sends payment reminder emails to customers who booked yesterday and haven't completed payment. It's built using the Database.Batchable interface to handle large volumes and efficiently send reminders in bulk.

```
global class PaymentReminderBatch implements Database.Batchable<SObject> {

    global Database.QueryLocator start(Database.BatchableContext bc) {

        Date targetDate = System.today().addDays(-1); // Bookings made yesterday




        String query = 'SELECT Id, Name, Customer_Email__c, Booking_Date__c '
+

                    'FROM Booking__c ' +

                    'WHERE Booking_Status__c = \'Pending\' AND Booking_Date__c
= :targetDate';




        return Database.getQueryLocator(query);

    }
```

```
    global void execute(Database.BatchableContext bc, List<Booking__c> scope) {

        List<Messaging.SingleEmailMessage> emails = new
List<Messaging.SingleEmailMessage>();



        for (Booking__c booking : scope) {

            if (String.isNotBlank(booking.Customer_Email__c)) {

                Messaging.SingleEmailMessage mail = new
Messaging.SingleEmailMessage();

                mail.setToAddresses(new String[] { booking.Customer_Email__c });

                mail.setSubject('Payment Reminder for Your Booking');

                mail.setPlainTextBody('Hi,\n\nThis is a gentle reminder to complete
your payment for booking: ' + booking.Name +

                                        '.\n\nPlease make the payment to confirm your
trip.\n\nThanks,\nTours & Travels CRM');

                emails.add(mail);

            }

        }



        if (!emails.isEmpty()) {

            Messaging.sendEmail(emails);

        }

    }
```

```
    global void finish(Database.BatchableContext bc) {

        // Optional: notify admin

        Messaging.SingleEmailMessage adminMail = new
Messaging.SingleEmailMessage();

        adminMail.setToAddresses(new String[] { 'annapurna@thesmartbridge.com'
});

        adminMail.setSubject('Daily Payment Reminder Batch Completed');

        adminMail.setPlainTextBody('Payment reminders for pending bookings have
been processed.');

        Messaging.sendEmail(new Messaging.SingleEmailMessage[] { adminMail
});

    }

}
```

SchedulePaymentReminderBatch: Schedules and runs the PaymentReminderBatch class to execute daily. This ensures pending payment reminders are sent out automatically each day without manual intervention.

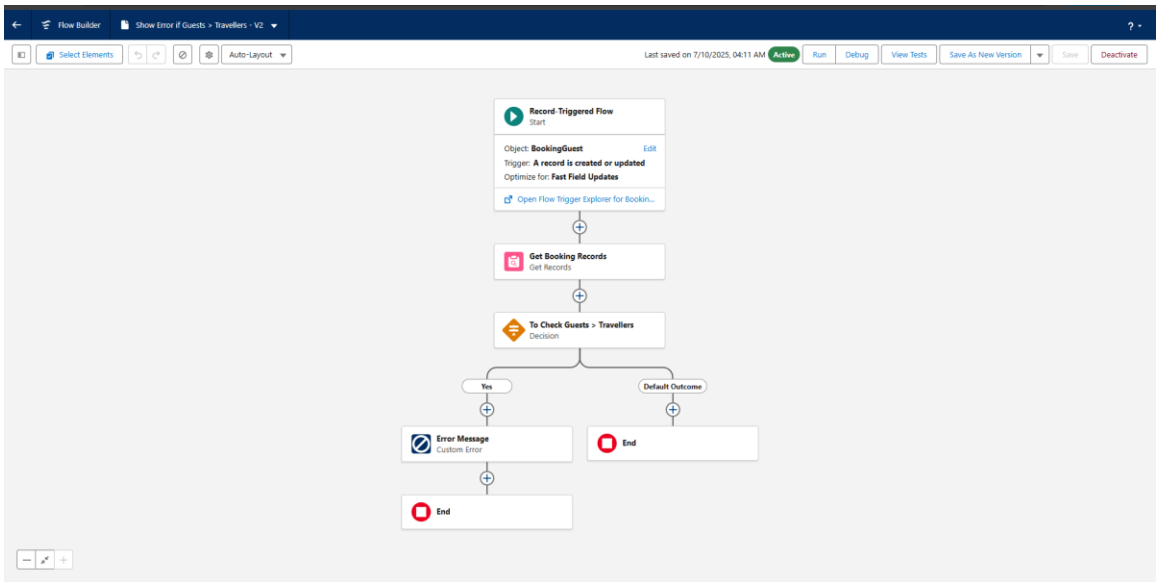
```
public class SchedulePaymentReminderBatch implements Schedulable {

    public void execute(SchedulableContext sc) {

        PaymentReminderBatch batch = new PaymentReminderBatch();

        Database.executeBatch(batch, 200);

    }

}
```

TravelPackageController: An Aura-enabled controller that supports Lightning components. Provides a @AuraEnabled method getPackagesByCountry() that returns travel packages filtered by a selected country—ideal for dynamic UI filtering in Lightning apps.

```
public with sharing class TravelPackageController {

 @AuraEnabled(cacheable=true)

   public static List<TravelPackage__c> getPackagesByCountry(String country) {

  return [SELECT Id, Name, Package_Type__c,Duration_in_Days__c ,
Guide_Included__c,
Membership__c,Region__c,Transportation_Modes__c,Availability_Status__c,
Average_Rating__c,Places_Covered__c FROM TravelPackage__c
WHERE Country__c = :country];

 }

}
```

Flows

Flow Name: Show Error if Guests Exceed Travelers



This flow ensures that users don't accidentally create more BookingGuest records

than allowed based on the number of travelers in the related booking. It's a safeguard to maintain data integrity and prevent mismatch between guest entries and travel slots.

Trigger Type:

- Record-Triggered Flow on the BookingGuest object
- Trigger Conditions: Runs when a record is created or updated
- Optimization: Set for Fast Field Updates

Phase 3: UI/UX Development & Customization
Lightning App Setup via App Manager

The Tours and Travels CRM App was created and configured using Salesforce App Manager. The app brings together key objects such as:

- Booking
- Customer Info
- TravelPackage
- Employee
- Feedback
- BookingPayments

The custom app includes:

- A branded app name and icon
- Navigation tabs for key objects
- Access restricted by profile (e.g., only Travel Agents can access certain tabs)

Page Layouts and Dynamic Forms

Dynamic Forms were configured to enhance the user experience on the Booking record page. The layout was upgraded to allow field-level control and conditional visibility without writing code.

Dynamic Field Visibility Rules:

- Cancellation Date, Cancel Confirmation, and Approval Status fields are only shown when:
    - Booking Status = Cancelled

Setup Steps:

1. Open a Booking record and click the ⚙ **Gear icon > Edit Page**.
2. Click on the **Details** section and upgrade to Dynamic Forms.
3. For each of the fields, set **Field Visibility Filters** using:
    - **Field**: Booking Status
    - **Operator**: Equals
    - **Value**: Cancelled
4. Save and Activate the Lightning Page for both Desktop and Mobile.

This makes the form more intelligent and avoids overwhelming the user with unnecessary fields unless they're relevant.

Lightning Pages

Lightning Pages were customized to:

- Include Dynamic Forms for Booking records
- Use Record Detail and Highlights Panel for important fields
- Improve readability and streamline workflows for Travel Agents

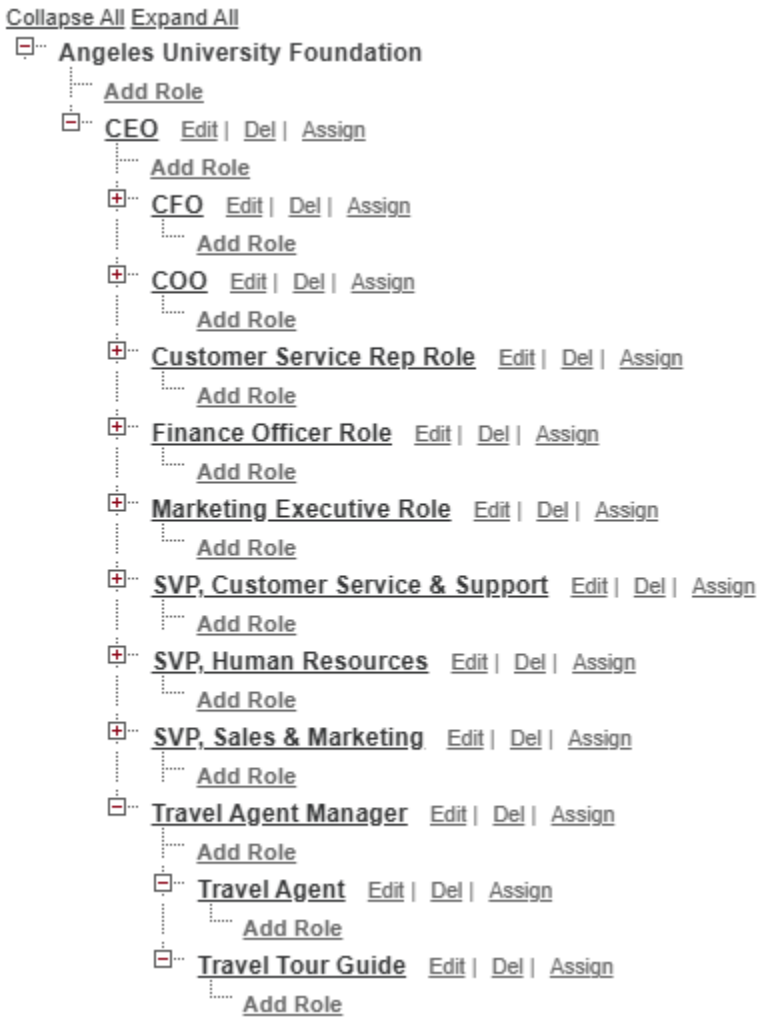
Each page was activated as the default for Desktop and Phone, ensuring a responsive and consistent experience across devices.

User Management

User setup involved assigning:

- **Travel Agent Profile** (Salesforce Platform License)
- **Tour Agent Manager** (Salesforce Platform License)

Users were assigned to **roles** within the role hierarchy to control record visibility:



This allowed managers to view subordinate records and enabled approval routing via the Role Hierarchy.

Reports and Dashboards

While no advanced analytics were built in this phase, the groundwork was laid for:

- List Views for each object (e.g., All Bookings, Bookings by Country)
- Salesforce standard report types were used to explore:
    - Booking Summary by Status
    - Customers by Region

    - Employee Availability Reports

Dashboards are planned for a later phase and will include metrics like:

- Total Bookings by Month
- Most Popular Travel Packages
- Pending vs. Confirmed Bookings

Phase 4: Data Migration, Testing & Security

Data Migration

- Used Data Import Wizard to import:
  - 20 records for Customer Info
  - 20 records for Employees
  - 20 records for TravelPackage
- CSV validation ensured country–city dependency using normalized data

Security

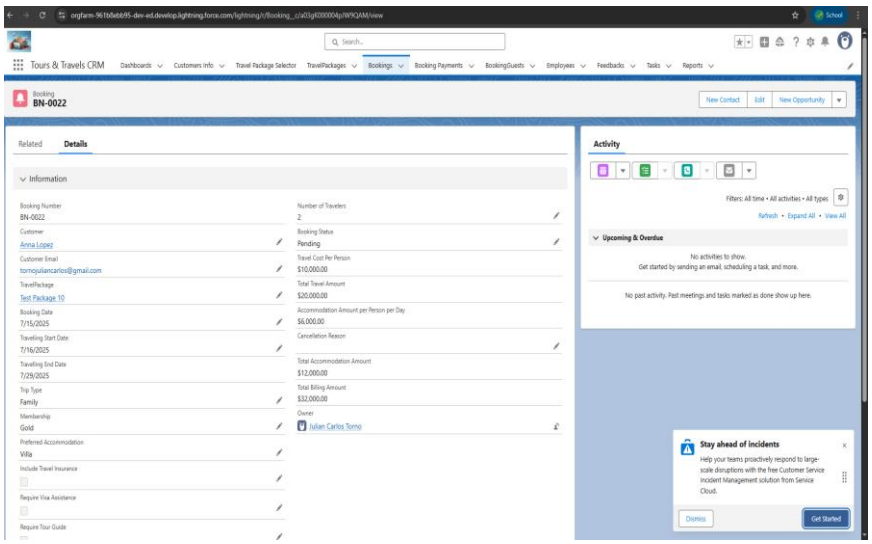
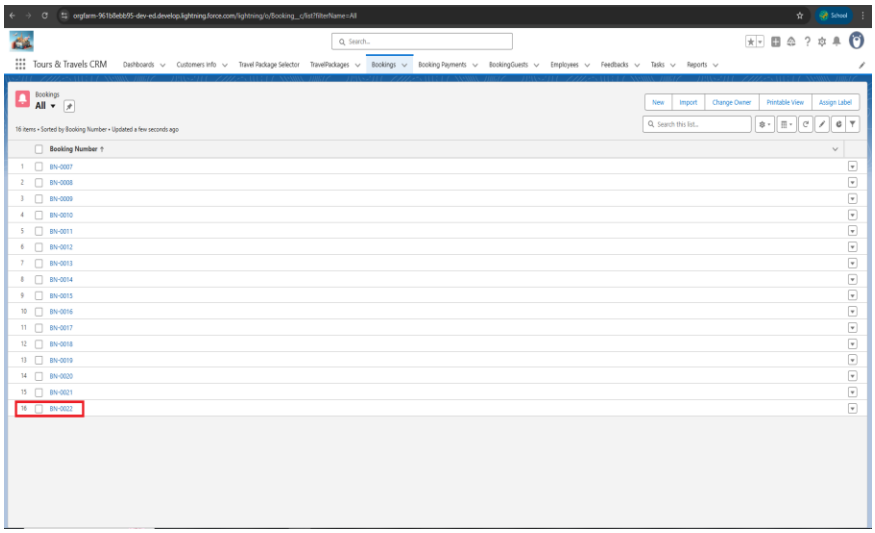
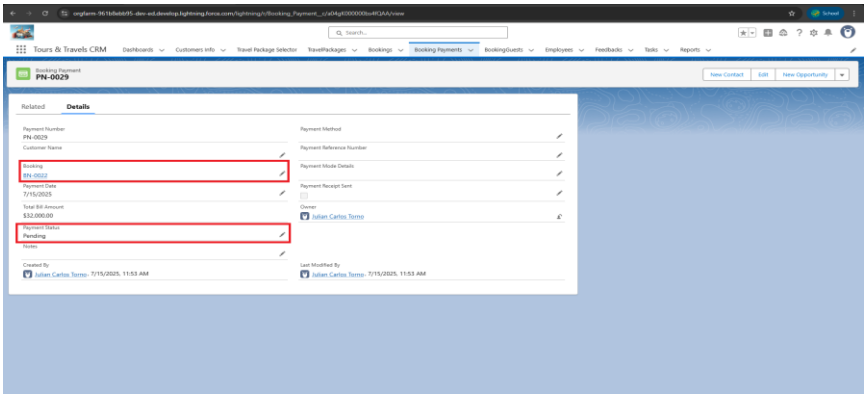
- Profiles:
  - Travel Agent, Tour Guide (both using Salesforce Platform license)
- Role Hierarchy: Manager → Agent
- Basic sharing rules allow record visibility by role

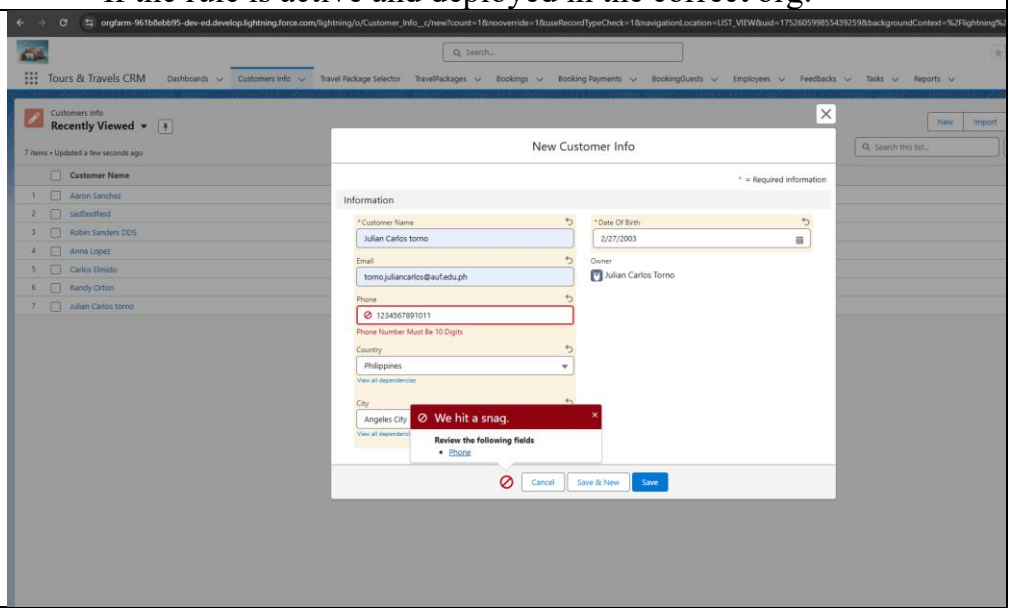
Approval Process: Booking Cancellation

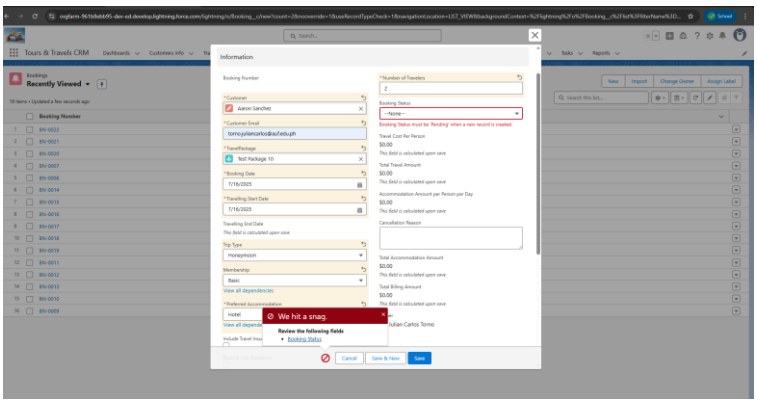
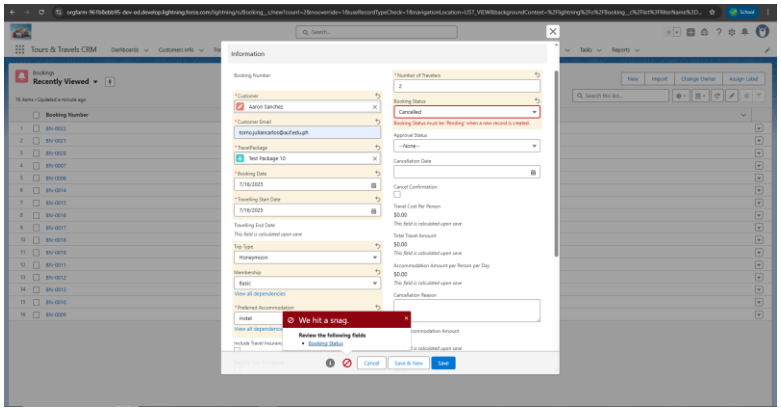
- Trigger: Booking.Status = Cancelled AND Cancel_Confirmation = True
- Approver: Travel Agent Manager
- Actions:
  - **Initial Submit**: Status → Pending
  - **Approval**:
    - Update: Approval Status = Approved
    - Send email: Template Cancellation Approved
  - **Rejection**:
    - Update: Approval Status = Rejected, Booking Status = Confirmed
    - Send email: Template Cancellation Rejected

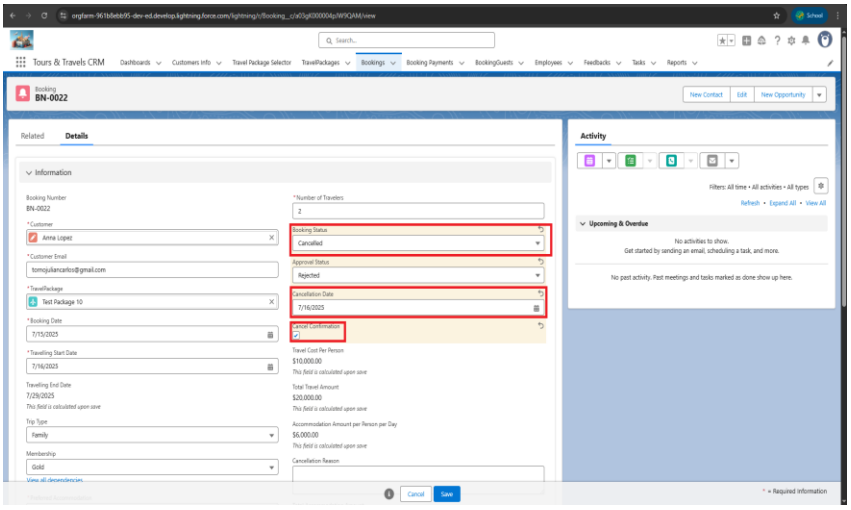
Test Cases:

Test Case	Verify that a new booking can be created successfully with all mandatory fields
Test Steps:	1. Log in as a Travel Agent with appropriate permissions. 2. Navigate to the Booking tab from the App Launcher or main navigation. 3. Click the New button to create a new Booking record. 4. Fill in all required fields: • Customer • Customer Email • Travelling Start Date • Travel Package • Number of Travelers • Booking Date • Booking Status • Preferred Accommodation 5. Click Save.
Expected Result:	• The Booking record should be saved successfully. • The record should appear in the Booking List View. • A related record must be created in Booking Payments with Payment Status = Pending. • BookingGuest records must be auto-generated to match the number of travelers.
Defects (if any):	If the Booking is not saved, check: • Missing required fields • Trigger logic in BookingTriggerHandler

Screenshot:	  

Test Case	Verify that phone number must be exactly 10 digits
Test Steps:	1. Log in as a Travel Agent or Admin. 2. Navigate to the Customer Info tab. 3. Click New to create a new customer record. 4. Enter a phone number with fewer or more than 10 digits (e.g., 123456789 or 12345678901). 5. Fill out the other required fields and click Save.
Expected Result:	• A validation error appears with the message: "Phone Number Must Be 10 Digits". • The record is not saved to Salesforce.
Defects (if any):	If no error appears, check: • The validation rule • If the rule is active and deployed in the correct org.
Screenshot:	

Test Case	Verify that Booking status is set to 'Pending' upon record creation
Test Steps:	1. Log in as a Travel Agent. 2. Go to the Booking tab and click New. 3. Attempt to set the Booking Status field to a value other than Pending. 4. Fill out the rest of the required fields and click Save.
Expected Result:	• A validation error appears blocking the save. • The message prevents users from setting a status other than Pending upon creation.
Defects (if any):	If the record is saved with a non-pending status, check: • Validation rule using ISNEW() is correctly written • Rule is active • Field name in the rule matches the API name exactly (Status__c)
Screenshot:	 

Test Case	Verify that cancellation triggers approval process
Test Steps:	1. Log in as a Travel Agent. 2. Open an existing Booking record. 3. Change the Booking Status to Cancelled. 4. Set Cancel_Confirmation to True. 5. Click Save.
Expected Result:	• An approval process is triggered. • The Booking record's status is set to Pending. • The assigned approver receives an approval email notification.
Defects (if any):	If the approval does not trigger: • Ensure both Booking Status = Cancelled and Cancel_Confirmation = checked • Check that the approval process is active • Confirm approver assignment and valid email template
Screenshot:	

Phase 5: Deployment, Documentation & Maintenance

Deployment Strategy

The deployment of the Tours and Travels CRM was conducted using a combination of browser-based tools and manual configuration methods:

- Lightning Web Components (LWC) were deployed using the Lightning Studio Chrome Extension, which allowed rapid development and real-time testing directly in the browser environment.
- Apex classes, triggers, and validation rules were created and tested directly in the Developer Console of the Salesforce environment.
- For broader deployments (e.g., from Sandbox to Production), the recommended approach would be using Change Sets:
    - Outbound Change Set from the Sandbox includes components such as:
        - Custom Objects (Booking__c, BookingGuest__c, etc.)
        - Apex classes (BookingTriggerHandler)
        - Validation Rules, Flows, Email Templates
    - Inbound Change Set received in Production and deployed after validation.
- In future iterations or team environments, deployment can also be managed via:
    - Salesforce CLI (SFDX) for version-controlled deployment
    - VS Code with Salesforce Extension Pack for advanced metadata handling

System Maintenance and Monitoring

To ensure the system remains functional, accurate, and scalable, the following maintenance plan will be followed:

- **Regular Audit of Validation Rules**
  Validation logic (e.g., email formats, age, status checks) will be reviewed quarterly to ensure they align with changing business rules.
- **Flow Monitoring**
  All record-triggered flows and decision elements (such as guest-to-traveler checks) will be monitored via Setup > Flow > View Flow Usage to detect any errors or failed executions.
- **User Feedback Collection**
  Feedback from Travel Agents and Admins will be gathered to improve page layouts, visibility rules, and form behaviors.
- **Data Cleanup Schedules**
  Duplicate detection and field history tracking will be reviewed monthly to maintain clean and reliable datasets.
- **Profile and Permission Reviews**
  Access rights (especially for Travel Agent Manageer and Travel Agent roles) will be periodically reviewed to ensure proper access control and security.

Troubleshooting Documentation

A structured approach is in place to troubleshoot and resolve any issues within the CRM system:

1. **Validation Error Logs**
   - If a user cannot save a record, check field-level validation messages or enable debug logs in Setup > Debug Logs for that user.
   - Example: Phone or Email validation failure will display specific error messages configured in the rule.
2. **Apex Trigger Failures**
   - Trigger logic is separated into the BookingTriggerHandler class.
   - Any failed DML or logic will be traced via logs and reviewed in Developer Console or Setup > Apex Jobs.

3. **Flow Execution Failures**
    - Navigate to Setup > Flows > View Details and Versions, then check Paused and Failed Interviews.
    - Specific flow elements (e.g., custom error screens) help detect logic breakdowns (e.g., mismatched guest count).
4. **Approval Process Issues**
    - Ensure entry criteria are still valid (e.g., Status = Cancelled, Cancel_Confirmation = True).
    - Check for missing user assignments or inactive email templates in Setup > Approval Processes.
5. **Component Rendering or LWC Errors**
    - Lightning Web Components are monitored via Chrome DevTools and Salesforce Lightning App Builder.
    - Any API-related errors in LWC (e.g., Apex method not returning results) will show in browser console logs.

Conclusion

The Tours and Travels CRM was developed as a functional and simplified system for managing bookings, client data, and company operations. The system addresses critical operational concerns, including manual coordination and record accuracy, by integrating Salesforce features like as Flows, validation rules, approval processes, and Apex custom automation.

With the platform in place, responsibilities such as booking approvals, guest tracking, and payment monitoring are now more accurate and visible. The setup not only saves time for the personnel, but it also provides a better organized experience for clients.

While the system can be enhanced further, it now serves the key business goals of a travel service provider by simplifying day-to-day operations and allowing for long-term scalability.