IT314 – Software Engineering
G27 Flight Booking System

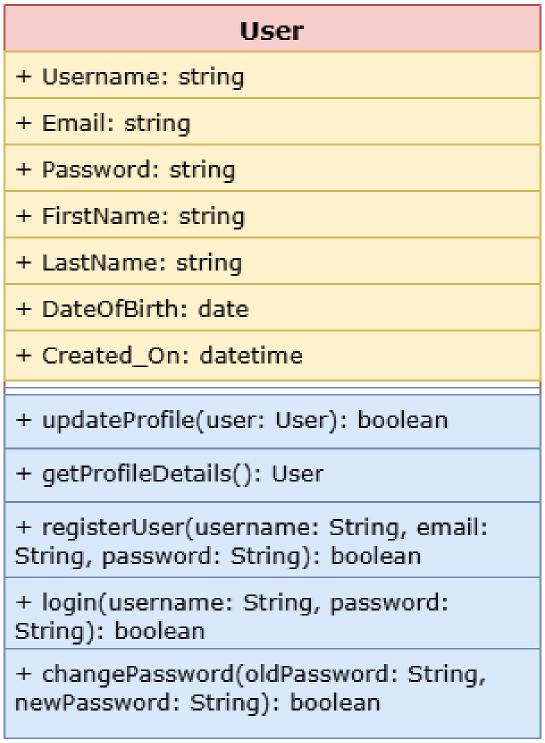
Sprint Details

Document Purpose: This document provides an in-depth overview of the sprint details for the SkyLynx Flight Booking System. Each sprint outlines specific functionality with focus areas, front-end and back-end implementation, testing strategies, and associated function point estimation.

Sprint 1: User Registration & Authentication

Section	Details
Focus Points	- Implement user registration (Sign-up, Login). - Develop user authentication and authorization (Session management). - Create basic profile management (View and update profile). - Database setup for user management.
Front-end	- Design registration and login forms. - Develop profile page UI.
Back-end	- Implement user authentication using JWT/OAuth. - Develop API endpoints for user management. - Setup database schema for user details.
Testing	- Unit testing for authentication and profile management. - User acceptance testing for registration process.

1.a)Class Diagram:



User

+ Username: string

+ Email: string

+ Password: string

+ FirstName: string

+ LastName: string

+ DateOfBirth: date

+ Created_On: datetime

+ updateProfile(user: User): boolean

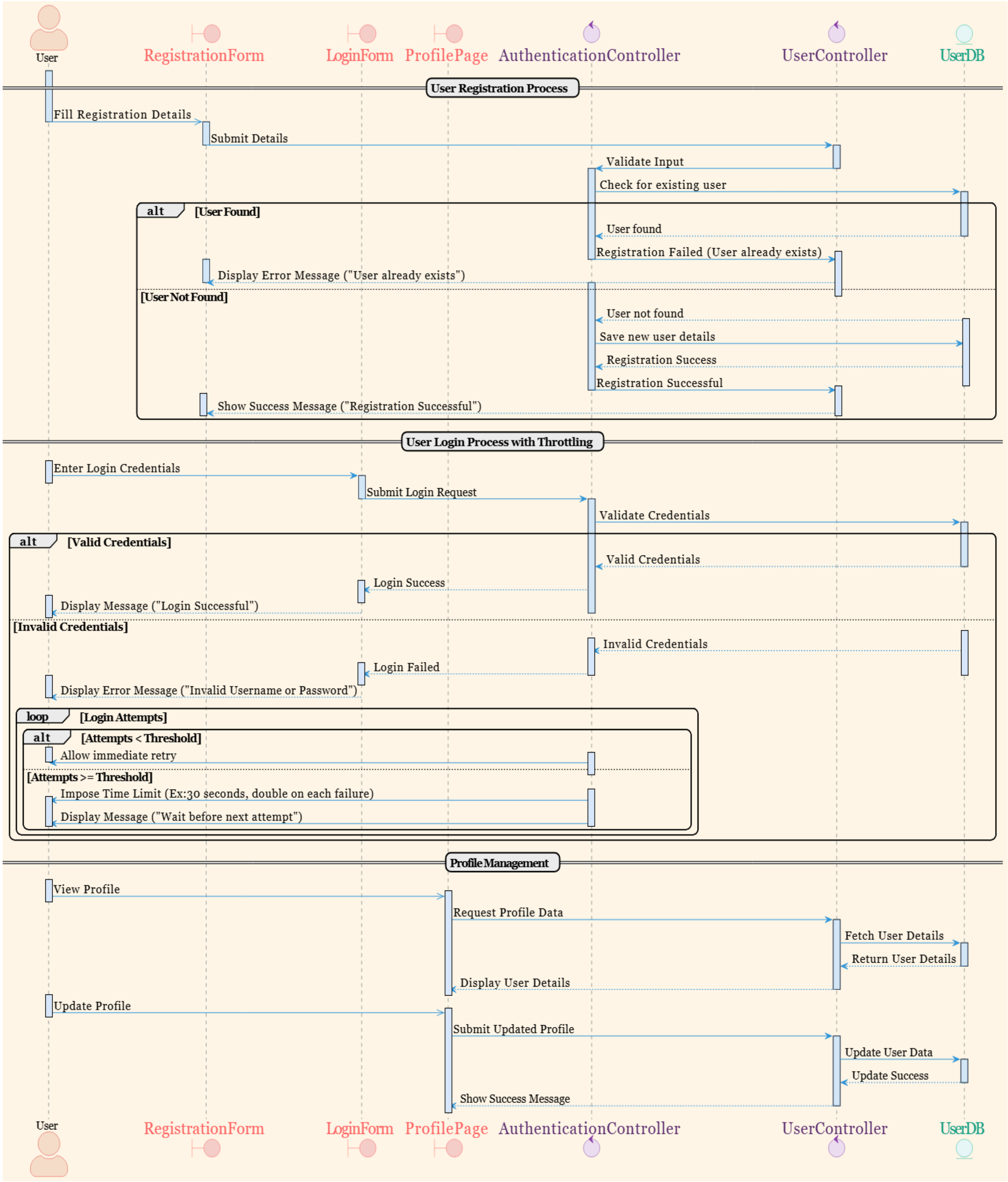
+ getProfileDetails(): User

+ registerUser(username: String, email: String, password: String): boolean

+ login(username: String, password: String): boolean

+ changePassword(oldPassword: String, newPassword: String): boolean

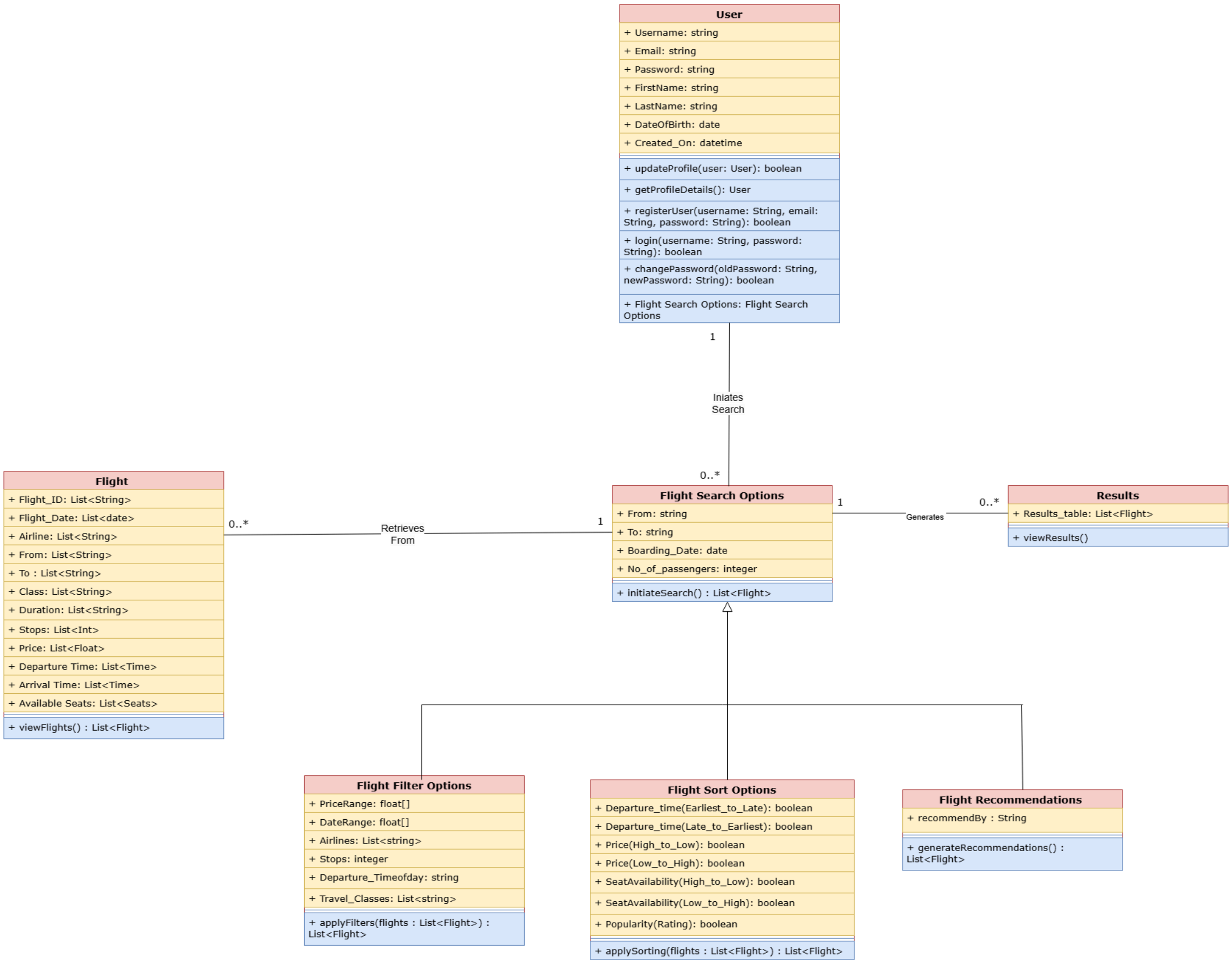
1.b) Sequence Diagram



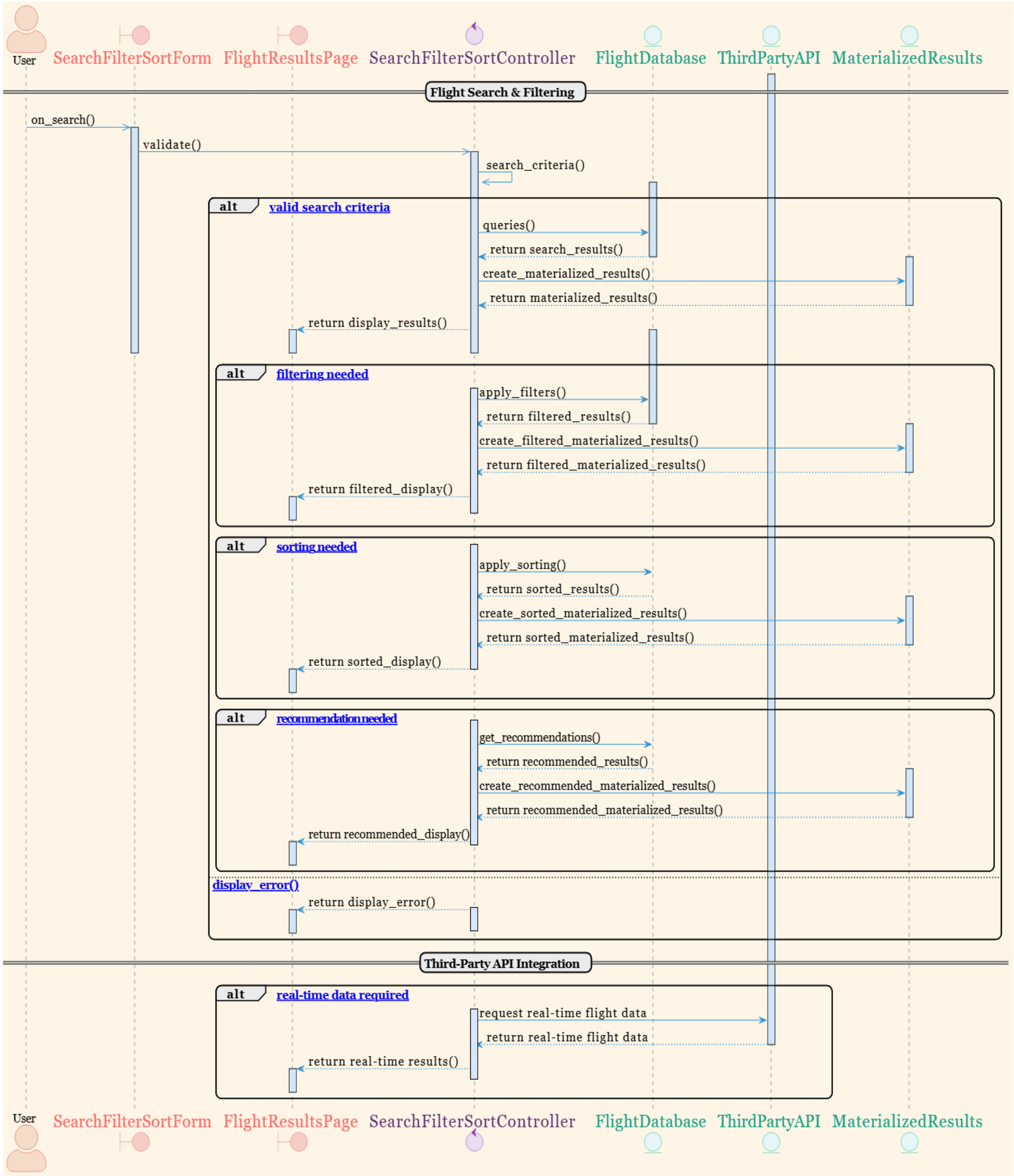
Sprint 2: Flight Search & Filtering

Section	Details
Focus Points	- Implement flight search functionality. - Add filters (Date, Destination, Price, etc.). - Integrate third-party APIs (if required) for real-time flight data. - Develop sorting and recommendation logic.
Front-end	- Design search and filter forms. - Implement flight results UI with sorting options.
Back-end	- Develop API endpoints for flight search. - Implement filtering and sorting logic. - Connect to third-party flight data API.
Testing	- Functional testing for search and filter. - Performance testing for API calls and data fetching.

2.a)Class Diagram:


User
+ Username: string
+ Email: string
+ Password: string
+ FirstName: string
+ LastName: string
+ DateOfBirth: date
+ Created_On: datetime
+ updateProfile(user: User): boolean
+ getProfileDetails(): User
+ registerUser(username: String, email: String, password: String): boolean
+ login(username: String, password: String): boolean
+ changePassword(oldPassword: String, newPassword: String): boolean
+ Flight Search Options: Flight Search Options
1
Iniates Search
0..*
Flight
+ Flight_ID: List<String>
+ Flight_Date: List<date>
+ Airline: List<String>
+ From: List<String>
+ To : List<String>
+ Class: List<String>
+ Duration: List<String>
+ Stops: List<Int>
+ Price: List<Float>
+ Departure Time: List<Time>
+ Arrival Time: List<Time>
+ Available Seats: List<Seats>
+ viewFlights() : List<Flight>
0..*
Retrieves From
1
Flight Search Options
+ From: string
+ To: string
+ Boarding_Date: date
+ No_of_passengers: integer
+ initiateSearch() : List<Flight>
1
Generates
0..*
Results
+ Results_table: List<Flight>
+ viewResults()
Flight Filter Options
+ PriceRange: float[]
+ DateRange: float[]
+ Airlines: List<string>
+ Stops: integer
+ Departure_Timeofday: string
+ Travel_Classes: List<string>
+ applyFilters(flights : List<Flight>) : List<Flight>
Flight Sort Options
+ Departure_time(Earliest_to_Late): boolean
+ Departure_time(Late_to_Earliest): boolean
+ Price(High_to_Low): boolean
+ Price(Low_to_High): boolean
+ SeatAvailability(High_to_Low): boolean
+ SeatAvailability(Low_to_High): boolean
+ Popularity(Rating): boolean
+ applySorting(flights : List<Flight>) : List<Flight>
Flight Recommendations
+ recommendBy : String
+ generateRecommendations() : List<Flight>

2.b) Sequence Diagram

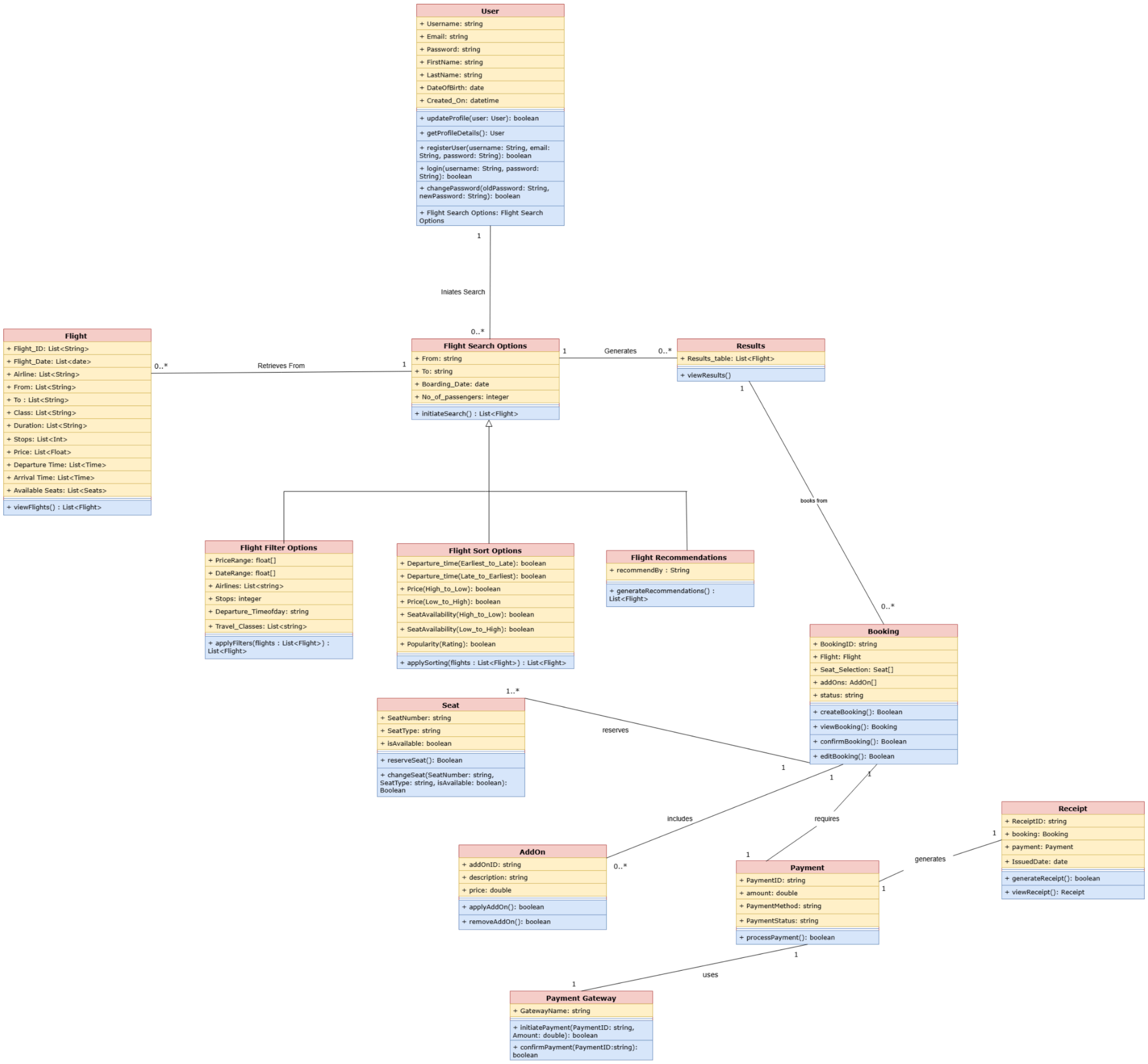

User
SearchFilterSortForm
FlightResultsPage
SearchFilterSortController
FlightDatabase
ThirdPartyAPI
MaterializedResults
Flight Search & Filtering
on_search()
validate()
search_criteria()
alt
valid search criteria
queries()
return search_results()
create_materialized_results()
return materialized_results()
return display_results()
alt
filtering needed
apply_filters()
return filtered_results()
create_filtered_materialized_results()
return filtered_materialized_results()
return filtered_display()
alt
sorting needed
apply_sorting()
return sorted_results()
create_sorted_materialized_results()
return sorted_materialized_results()
return sorted_display()
alt
recommendation needed
get_recommendations()
return recommended_results()
create_recommended_materialized_results()
return recommended_materialized_results()
return recommended_display()
display_error()
return display_error()
Third-Party API Integration
alt
real-time data required
request real-time flight data
return real-time flight data
return real-time results()
User
SearchFilterSortForm
FlightResultsPage
SearchFilterSortController
FlightDatabase
ThirdPartyAPI
MaterializedResults

Sprint 3: Flight Booking & Payment Integration

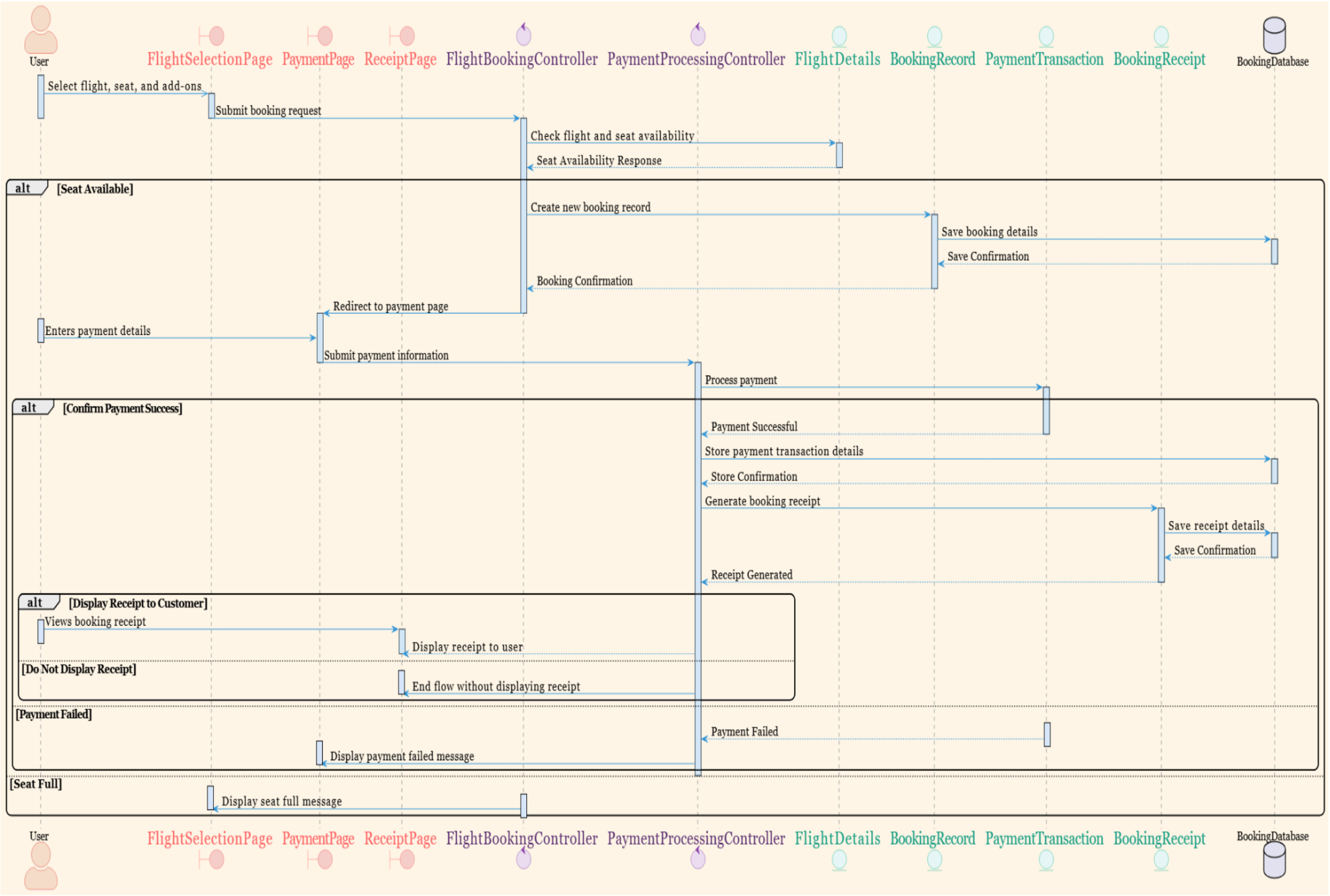
Section	Details
Focus Points	- Implement booking flow (Select flight, add-ons, seat selection). - Integrate payment gateway (Stripe, PayPal,

	etc.). - Generate and display booking receipts.
Front-end	- Develop booking UI (Select flight, seat, add-ons). - Design payment page UI. - Implement a receipt page.
Back-end	- Develop booking management API. - Integrate payment gateway and handle transactions. - Store booking details and generate receipts.
Testing	- End-to-end testing for booking flow. - Integration testing for payment gateway.

3.a) Class Diagram


User
+ Username: string
+ Email: string
+ Password: string
+ FirstName: string
+ LastName: string
+ DateOfBirth: date
+ Created_On: datetime
+ updateProfile(user: User): boolean
+ getProfileDetails(): User
+ registerUser(username: String, email: String, password: String): boolean
+ login(username: String, password: String): boolean
+ changePassword(oldPassword: String, newPassword: String): boolean
+ Flight Search Options: Flight Search Options
1
Iniates Search
0..*
Flight
+ Flight_ID: List<String>
+ Flight_Date: List<date>
+ Airline: List<String>
+ From: List<String>
+ To : List<String>
+ Class: List<String>
+ Duration: List<String>
+ Stops: List<Int>
+ Price: List<Float>
+ Departure Time: List<Time>
+ Arrival Time: List<Time>
+ Available Seats: List<Seats>
+ viewFlights() : List<Flight>
0..*
Retrieves From
1
Flight Search Options
+ From: string
+ To: string
+ Boarding_Date: date
+ No_of_passengers: integer
+ initiateSearch() : List<Flight>
1
Generates
0..*
Results
+ Results_table: List<Flight>
+ viewResults()
1
books from
0..*
Flight Filter Options
+ PriceRange: float[]
+ DateRange: float[]
+ Airlines: List<string>
+ Stops: integer
+ Departure_Timeofday: string
+ Travel_Classes: List<string>
+ applyFilters(flights : List<Flight>) : List<Flight>
Flight Sort Options
+ Departure_time(Earliest_to_Late): boolean
+ Departure_time(Late_to_Earliest): boolean
+ Price(High_to_Low): boolean
+ Price(Low_to_High): boolean
+ SeatAvailability(High_to_Low): boolean
+ SeatAvailability(Low_to_High): boolean
+ Popularity(Rating): boolean
+ applySorting(flights : List<Flight>) : List<Flight>
Flight Recommendations
+ recommendBy : String
+ generateRecommendations() : List<Flight>
Booking
+ BookingID: string
+ Flight: Flight
+ Seat_Selection: Seat[]
+ addOns: AddOn[]
+ status: string
+ createBooking(): Boolean
+ viewBooking(): Booking
+ confirmBooking(): Boolean
+ editBooking(): Boolean
1..*
Seat
+ SeatNumber: string
+ SeatType: string
+ isAvailable: boolean
+ reserveSeat(): Boolean
+ changeSeat(SeatNumber: string, SeatType: string, isAvailable: boolean): Boolean
reserves
1
1
1
includes
requires
0..*
AddOn
+ addOnID: string
+ description: string
+ price: double
+ applyAddOn(): boolean
+ removeAddOn(): boolean
1
Payment
+ PaymentID: string
+ amount: double
+ PaymentMethod: string
+ PaymentStatus: string
+ processPayment(): boolean
1
generates
1
Receipt
+ ReceiptID: string
+ booking: Booking
+ payment: Payment
+ IssuedDate: date
+ generateReceipt(): boolean
+ viewReceipt(): Receipt
1
uses
1
Payment Gateway
+ GatewayName: string
+ initiatePayment(PaymentID: string, Amount: double): boolean
+ confirmPayment(PaymentID:string): boolean

3.b) Sequence Diagram

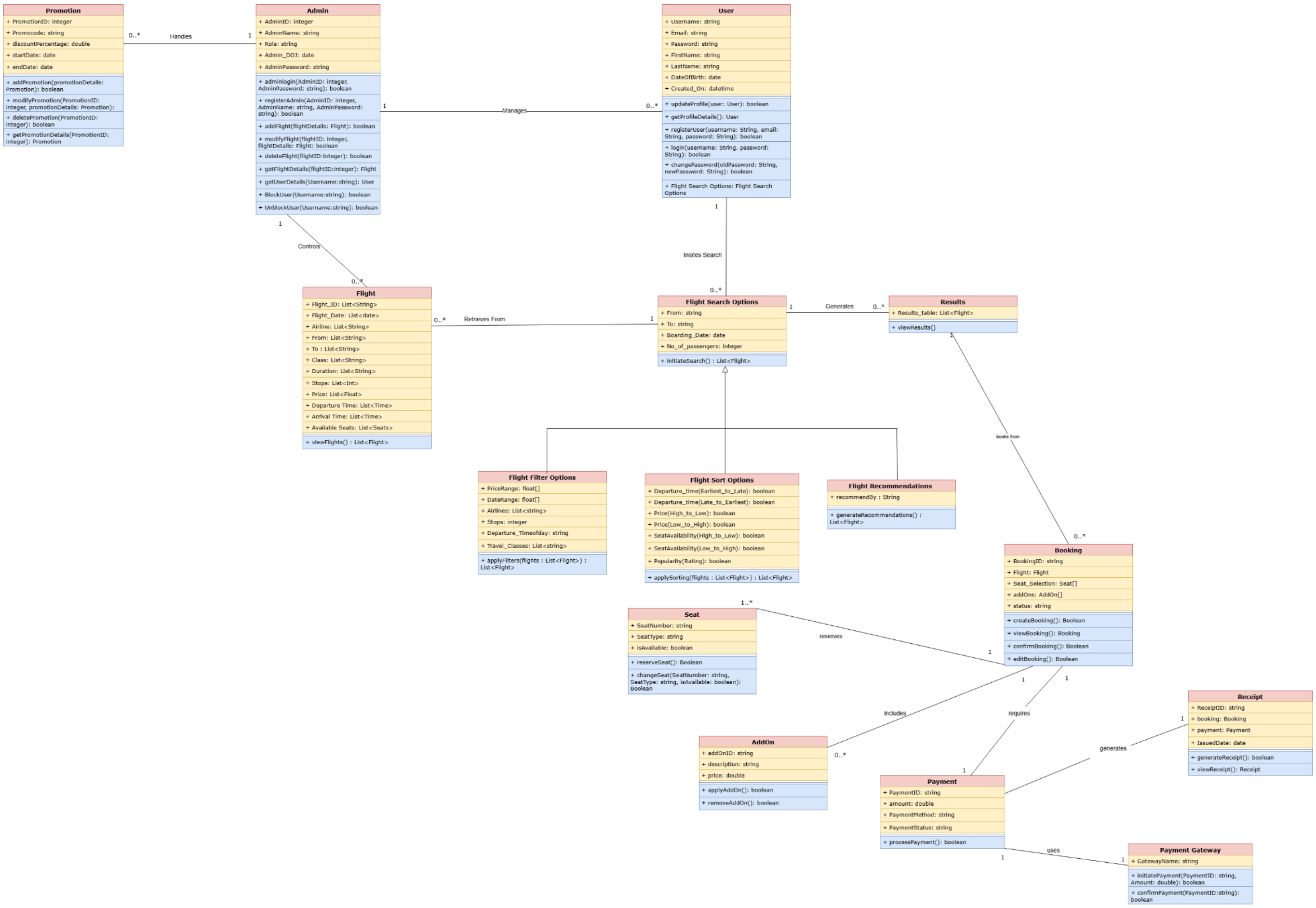


Sprint 4: Admin Panel & User Management

Section	Details
Focus Points	- Develop an admin panel for managing flights, promotions, and users. - Implement flight management features (Add/Modify/Delete flights). - Integrate user management (Block/Unblock users, manage promotions). - Secure admin access with role-based authentication.
Front-end	- Design admin dashboard UI. - Implement forms for flight and promotion management.
Back-end	- Develop CRUD operations for flights and users.

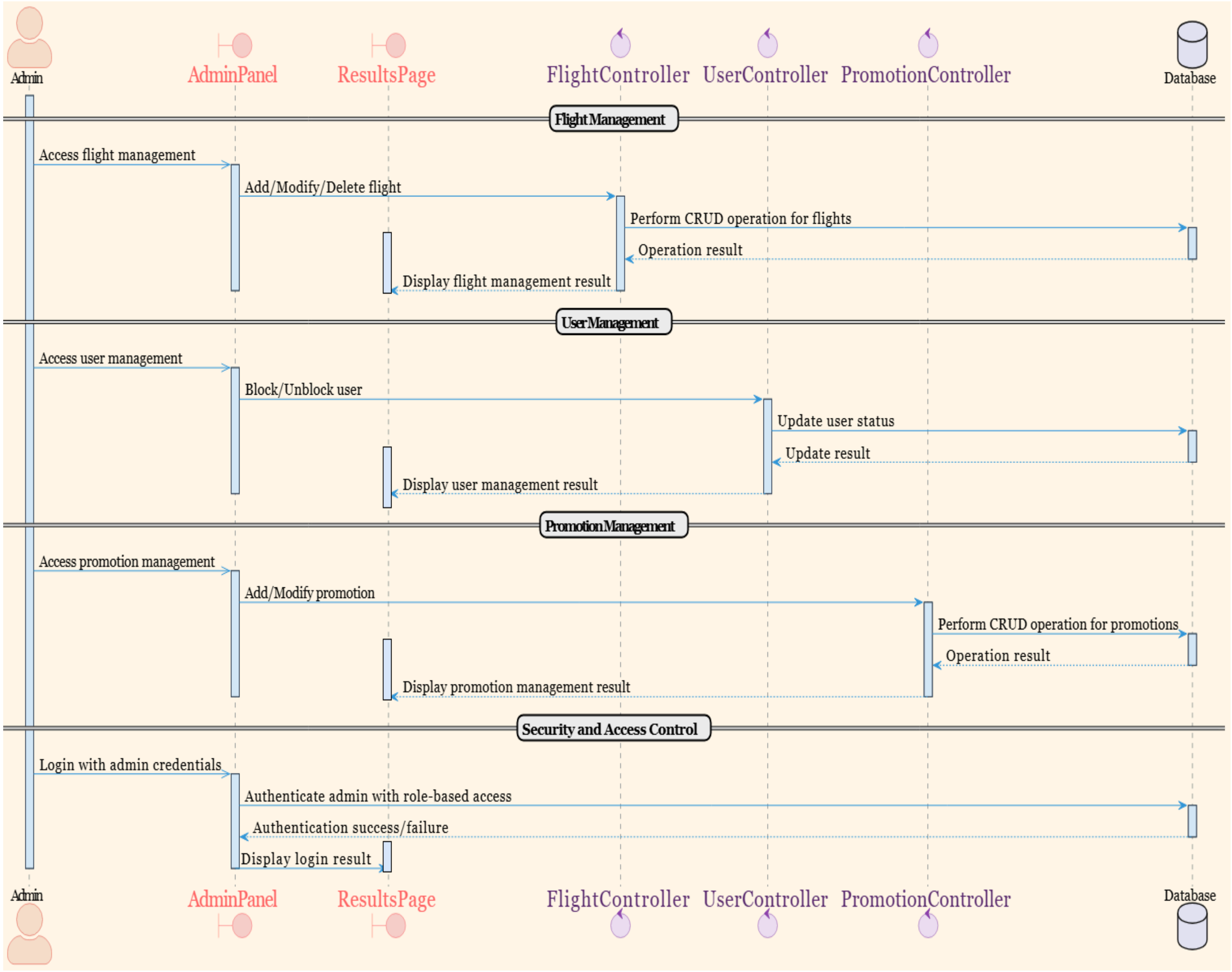
	- Implement backend for promotion management.
Testing	- Admin panel usability testing. - Security testing for admin features.

4.a) Class Diagram



Promotion
+ PromotionID: integer
+ Promocode: string
+ discountPercentage: double
+ startDate: date
+ endDate: date
+ addPromotion(promotionDetails: Promotion): boolean
+ modifyPromotion(PromotionID: integer, promotionDetails: Promotion):
+ deletePromotion(PromotionID: integer): boolean
+ getPromotionDetails(PromotionID: integer): Promotion
0..*
Handles
1
Admin
+ AdminID: integer
+ AdminName: string
+ Role: string
+ Admin_DOJ: date
+ AdminPassword: string
+ adminlogin(AdminID: integer, AdminPassword: string): boolean
+ registerAdmin(AdminID: integer, AdminName: string, AdminPassword: string): boolean
+ addFlight(flightDetails: Flight): boolean
+ modifyFlight(flightID: integer, flightDetails: Flight: boolean
+ deleteFlight(flightID:integer): boolean
+ getFlightDetails(flightID:integer): Flight
+ getUserDetails(Username:string): User
+ BlockUser(Username:string): boolean
+ UnblockUser(Username:string): boolean
1
Manages
0..*
User
+ Username: string
+ Email: string
+ Password: string
+ FirstName: string
+ LastName: string
+ DateOfBirth: date
+ Created_On: datetime
+ updateProfile(user: User): boolean
+ getProfileDetails(): User
+ registerUser(username: String, email: String, password: String): boolean
+ login(username: String, password: String): boolean
+ changePassword(oldPassword: String, newPassword: String): boolean
+ Flight Search Options: Flight Search Options
1
Controls
0..*
1
Iniates Search
0..*
Flight
+ Flight_ID: List<String>
+ Flight_Date: List<date>
+ Airline: List<String>
+ From: List<String>
+ To : List<String>
+ Class: List<String>
+ Duration: List<String>
+ Stops: List<Int>
+ Price: List<Float>
+ Departure Time: List<Time>
+ Arrival Time: List<Time>
+ Available Seats: List<Seats>
+ viewFlights() : List<Flight>
0..*
Retrieves From
1
Flight Search Options
+ From: string
+ To: string
+ Boarding_Date: date
+ No_of_passengers: integer
+ initiateSearch() : List<Flight>
1
Generates
0..*
Results
+ Results_table: List<Flight>
+ viewResults()
1
books from
0..*
Flight Filter Options
+ PriceRange: float[]
+ DateRange: float[]
+ Airlines: List<string>
+ Stops: integer
+ Departure_Timeofday: string
+ Travel_Classes: List<string>
+ applyFilters(flights : List<Flight>) : List<Flight>
Flight Sort Options
+ Departure_time(Earliest_to_Late): boolean
+ Departure_time(Late_to_Earliest): boolean
+ Price(High_to_Low): boolean
+ Price(Low_to_High): boolean
+ SeatAvailability(High_to_Low): boolean
+ SeatAvailability(Low_to_High): boolean
+ Popularity(Rating): boolean
+ applySorting(flights : List<Flight>) : List<Flight>
Flight Recommendations
+ recommendBy : String
+ generateRecommendations() : List<Flight>
Booking
+ BookingID: string
+ Flight: Flight
+ Seat_Selection: Seat[]
+ addOns: AddOn[]
+ status: string
+ createBooking(): Boolean
+ viewBooking(): Booking
+ confirmBooking(): Boolean
+ editBooking(): Boolean
1..*
reserves
1
Seat
+ SeatNumber: string
+ SeatType: string
+ isAvailable: boolean
+ reserveSeat(): Boolean
+ changeSeat(SeatNumber: string, SeatType: string, isAvailable: boolean): Boolean
1
1
includes
requires
0..*
1
AddOn
+ addOnID: string
+ description: string
+ price: double
+ applyAddOn(): boolean
+ removeAddOn(): boolean
Receipt
+ ReceiptID: string
+ booking: Booking
+ payment: Payment
+ IssuedDate: date
+ generateReceipt(): boolean
+ viewReceipt(): Receipt
1
generates
Payment
+ PaymentID: string
+ amount: double
+ PaymentMethod: string
+ PaymentStatus: string
+ processPayment(): boolean
1
uses
1
Payment Gateway
+ GatewayName: string
+ initiatePayment(PaymentID: string, Amount: double): boolean
+ confirmPayment(PaymentID:string): boolean

4.b) Sequence Diagram

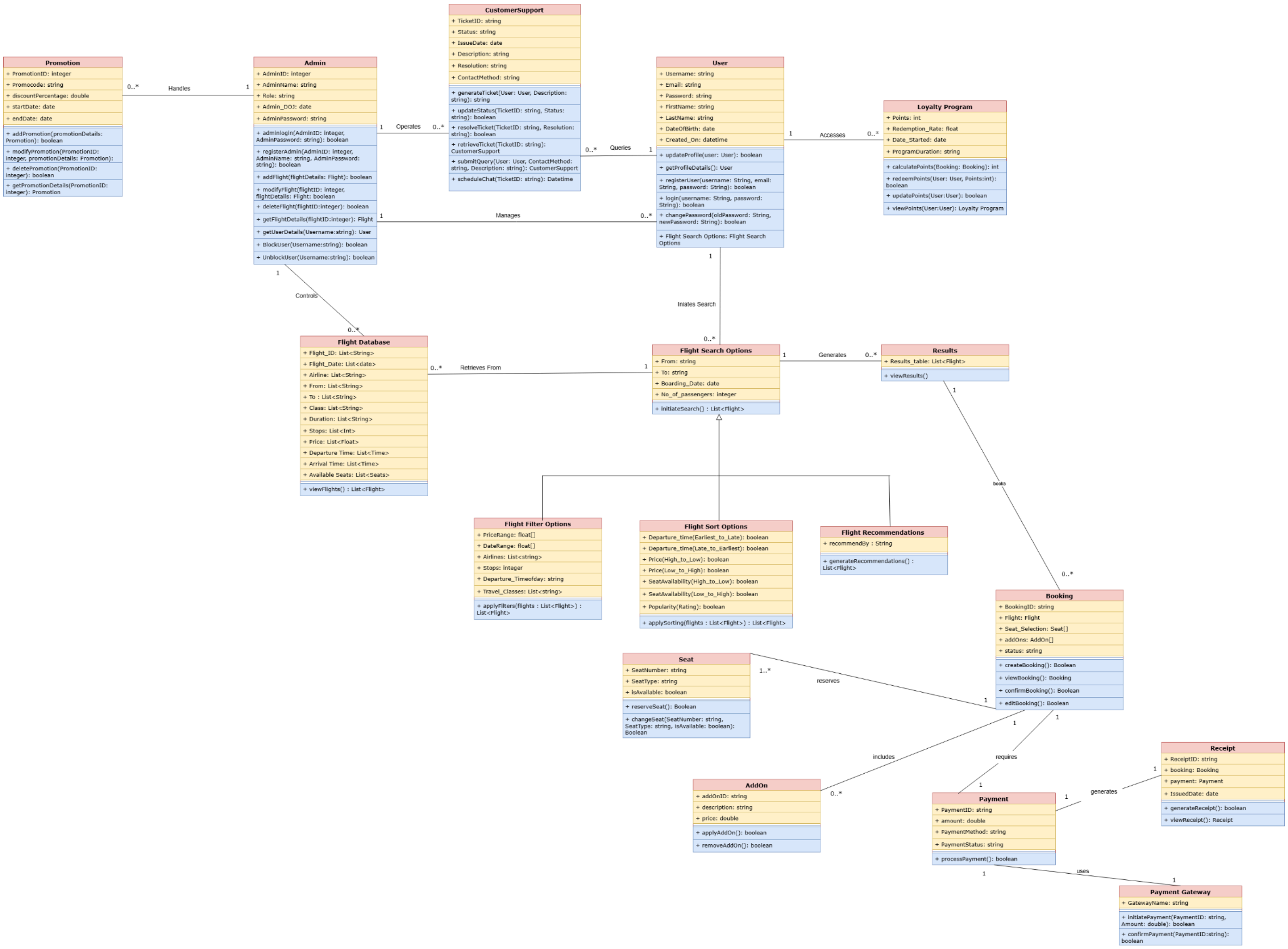


Sprint 5: Loyalty Program & Customer Service

Section	Details
Focus Points	- Implement a loyalty program (Points system, redemption). - Develop customer service contact features (Chat, Email). - Manage user queries and support tickets.
Front-end	- Design loyalty points interface in user profile. - Implement customer service contact form.

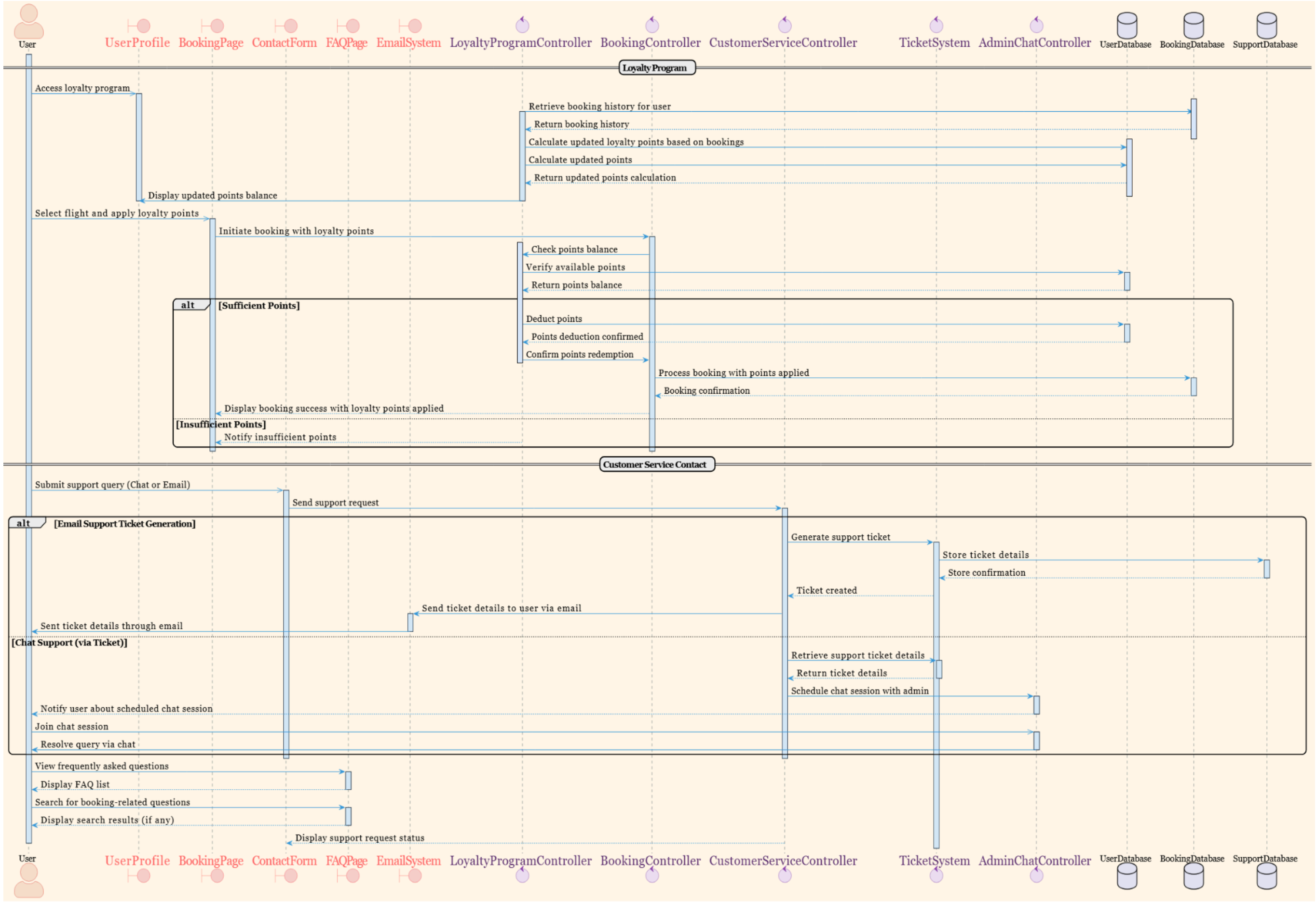
Back-end	- Develop loyalty program logic and integrate it with booking. - Implement backend support for customer queries. - Automate ticket generation and tracking.
Testing	- User acceptance testing for loyalty program. - Functional testing for customer service.

5.a) Class Diagram



Promotion
+ PromotionID: integer
+ Promocode: string
+ discountPercentage: double
+ startDate: date
+ endDate: date
+ addPromotion(promotionDetails: Promotion): boolean
+ modifyPromotion(PromotionID: integer, promotionDetails: Promotion):
+ deletePromotion(PromotionID: integer): boolean
+ getPromotionDetails(PromotionID: integer): Promotion
0..*
Handles
1
Admin
+ AdminID: integer
+ AdminName: string
+ Role: string
+ Admin_DOJ: date
+ AdminPassword: string
+ adminlogin(AdminID: integer, AdminPassword: string): boolean
+ registerAdmin(AdminID: integer, AdminName: string, AdminPassword: string): boolean
+ addFlight(flightDetails: Flight): boolean
+ modifyFlight(flightID: integer, flightDetails: Flight: boolean
+ deleteFlight(flightID:integer): boolean
+ getFlightDetails(flightID:integer): Flight
+ getUserDetails(Username:string): User
+ BlockUser(Username:string): boolean
+ UnblockUser(Username:string): boolean
1
Operates
0..*
CustomerSupport
+ TicketID: string
+ Status: string
+ IssueDate: date
+ Description: string
+ Resolution: string
+ ContactMethod: string
+ generateTicket(User: User, Description: string): string
+ updateStatus(TicketID: string, Status: string): boolean
+ resolveTicket(TicketID: string, Resolution: string): boolean
+ retrieveTicket(TicketID: string): CustomerSupport
+ submitQuery(User: User, ContactMethod: string, Description: string): CustomerSupport
+ scheduleChat(TicketID: string): Datetime
0..*
Queries
1
User
+ Username: string
+ Email: string
+ Password: string
+ FirstName: string
+ LastName: string
+ DateOfBirth: date
+ Created_On: datetime
+ updateProfile(user: User): boolean
+ getProfileDetails(): User
+ registerUser(username: String, email: String, password: String): boolean
+ login(username: String, password: String): boolean
+ changePassword(oldPassword: String, newPassword: String): boolean
+ Flight Search Options: Flight Search Options
1
Accesses
0..*
Loyalty Program
+ Points: int
+ Redemption_Rate: float
+ Date_Started: date
+ ProgramDuration: string
+ calculatePoints(Booking: Booking): int
+ redeemPoints(User: User, Points:int): boolean
+ updatePoints(User:User): boolean
+ viewPoints(User:User): Loyalty Program
1
Manages
0..*
1
Controls
0..*
Flight Database
+ Flight_ID: List<String>
+ Flight_Date: List<date>
+ Airline: List<String>
+ From: List<String>
+ To : List<String>
+ Class: List<String>
+ Duration: List<String>
+ Stops: List<Int>
+ Price: List<Float>
+ Departure Time: List<Time>
+ Arrival Time: List<Time>
+ Available Seats: List<Seats>
+ viewFlights() : List<Flight>
0..*
Retrieves From
1
1
Iniates Search
0..*
Flight Search Options
+ From: string
+ To: string
+ Boarding_Date: date
+ No_of_passengers: integer
+ initiateSearch() : List<Flight>
1
Generates
0..*
Results
+ Results_table: List<Flight>
+ viewResults()
1
books
0..*
Flight Filter Options
+ PriceRange: float[]
+ DateRange: float[]
+ Airlines: List<string>
+ Stops: integer
+ Departure_Timeofday: string
+ Travel_Classes: List<string>
+ applyFilters(flights : List<Flight>) : List<Flight>
Flight Sort Options
+ Departure_time(Earliest_to_Late): boolean
+ Departure_time(Late_to_Earliest): boolean
+ Price(High_to_Low): boolean
+ Price(Low_to_High): boolean
+ SeatAvailability(High_to_Low): boolean
+ SeatAvailability(Low_to_High): boolean
+ Popularity(Rating): boolean
+ applySorting(flights : List<Flight>) : List<Flight>
Flight Recommendations
+ recommendBy : String
+ generateRecommendations() : List<Flight>
Booking
+ BookingID: string
+ Flight: Flight
+ Seat_Selection: Seat[]
+ addOns: AddOn[]
+ status: string
+ createBooking(): Boolean
+ viewBooking(): Booking
+ confirmBooking(): Boolean
+ editBooking(): Boolean
Seat
+ SeatNumber: string
+ SeatType: string
+ isAvailable: boolean
+ reserveSeat(): Boolean
+ changeSeat(SeatNumber: string, SeatType: string, isAvailable: boolean): Boolean
1..*
reserves
1
1
includes
0..*
1
requires
1
AddOn
+ addOnID: string
+ description: string
+ price: double
+ applyAddOn(): boolean
+ removeAddOn(): boolean
Payment
+ PaymentID: string
+ amount: double
+ PaymentMethod: string
+ PaymentStatus: string
+ processPayment(): boolean
1
generates
1
Receipt
+ ReceiptID: string
+ booking: Booking
+ payment: Payment
+ IssuedDate: date
+ generateReceipt(): boolean
+ viewReceipt(): Receipt
1
uses
1
Payment Gateway
+ GatewayName: string
+ initiatePayment(PaymentID: string, Amount: double): boolean
+ confirmPayment(PaymentID:string): boolean

5.b) Sequence Diagram


User
UserProfile
BookingPage
ContactForm
FAQPage
EmailSystem
LoyaltyProgramController
BookingController
CustomerServiceController
TicketSystem
AdminChatController
UserDatabase
BookingDatabase
SupportDatabase
Loyalty Program
Access loyalty program
Retrieve booking history for user
Return booking history
Calculate updated loyalty points based on bookings
Calculate updated points
Return updated points calculation
Display updated points balance
Select flight and apply loyalty points
Initiate booking with loyalty points
Check points balance
Verify available points
Return points balance
alt
[Sufficient Points]
Deduct points
Points deduction confirmed
Confirm points redemption
Process booking with points applied
Booking confirmation
Display booking success with loyalty points applied
[Insufficient Points]
Notify insufficient points
Customer Service Contact
Submit support query (Chat or Email)
Send support request
alt
[Email Support Ticket Generation]
Generate support ticket
Store ticket details
Store confirmation
Ticket created
Send ticket details to user via email
Sent ticket details through email
[Chat Support (via Ticket)]
Retrieve support ticket details
Return ticket details
Schedule chat session with admin
Notify user about scheduled chat session
Join chat session
Resolve query via chat
View frequently asked questions
Display FAQ list
Search for booking-related questions
Display search results (if any)
Display support request status
User
UserProfile
BookingPage
ContactForm
FAQPage
EmailSystem
LoyaltyProgramController
BookingController
CustomerServiceController
TicketSystem
AdminChatController
UserDatabase
BookingDatabase
SupportDatabase

Function Point Estimation

Sprint 1: User Registration & Authentication

External Input (EI):

- Registration Form (User details input)
- Login Form
- Profile Update

External Output (EO):

- View Profile
- Login Response (Success/Failure)
- Registration Confirmation

External Query (EQ):

- Check Login Credentials

Internal Logical Files (ILF):

- User Database (Storing user info and authentication details)

External Interface Files (EIF):

- Third-Party Authentication (OAuth/JWT)

Measurement Parameter	Count	Weighting Factor	FP Count
# of Inputs (EI)	3	4	12
# of Outputs (EO)	3	5	15
# of Queries (EQ)	1	4	4
# of Internal Logical Files (ILF)	1	10	10
# of External Interface Files (EIF)	1	7	7

Unadjusted Function Count (UFC) :	**48**

Complexity Factors: 14 factors, each rated on a scale of 0 to 5, 0 being not important or applicable and 5 being absolutely essential.

SNo.	Complexity Factor	Rate
1	Backup and recovery	4
2	Data communication	4
3	Distributed processing functions	1
4	Is performance critical?	5
5	Existing operating environment	3
6	On-line data entry	4
7	Input transaction built over multiple screens	2
8	Master files updated on-line	4
9	Complexity of inputs, outputs, files, inquiries	3
10	Complexity of processing	5
11	Code design for reuse	4
12	Are conversion/installation included in design?	0
13	Multiple installations	0
14	Application designed to facilitate change by the user	2
	Total	**41**

Using Adjusted FP Count Formula:

AFPC = UFPC * [0.65 + 0.01 * (Total Rate of

Complexity Factors)] AFPC = 48* [0.65 + 0.01 * 41] =

48* [1.06] = **50.88** ≅ **51**

Sprint 2: Flight Search & Filtering

External Input (EI):

- Search Criteria (Input for date, destination, price, etc.)
- Filter Selection (Date, Destination, Price, etc.)

External Output (EO):

- View Search Results (Formatted flight results)
- Flight Recommendations

External Query (EQ):

- Flight Search Query
- Flight Filtering Query
- Flight Sorting Query

Internal Logical Files (ILF):

- Flight Database

External Interface Files (EIF):

- External Flight Data (Real-time flight data from third-party APIs)

Measurement Parameter	**Count**	**Weighting Factor**	**FP Count**
# of Inputs (EI)	2	4	8
# of Outputs (EO)	2	5	10
# of Queries (EQ)	3	4	12
# of Internal Logical Files (ILF)	1	10	10
# of External Interface Files (EIF)	1	7	7
Unadjusted Function Count (UFC) :			**47**

Complexity Factors: 14 factors, each rated on a scale of 0 to 5, 0 being not important or applicable and 5 being absolutely essential.

SNo.	**Complexity Factor**	**Rate**
1	Backup and recovery	4
2	Data communication	3
3	Distributed processing functions	2
4	Is performance critical?	5
5	Existing operating environment	3
6	On-line data entry	4
7	Input transaction built over multiple screens	2
8	Master files updated on-line	4
9	Complexity of inputs, outputs, files, inquiries	3
10	Complexity of processing	5
11	Code design for reuse	4
12	Are conversion/installation included in design?	0
13	Multiple installations	0
14	Application designed to facilitate change by the user	2
	Total	**41**

Using Adjusted FP Count Formula:

AFPC = UFPC * [0.65 + 0.01 * (Total Rate of

Complexity Factors)] AFPC = 47* [0.65 + 0.01 * 41] =

47* [1.06] = **49.82** ≅ **50**

Sprint 3: Flight Booking & Payment Integration

External Input (EI)

- Select Flight (Booking input)
- Seat Selection
- Payment Details
- Add-ons (Baggage, meals, etc.)

External Output (EO):

- Booking Confirmation
- Payment Confirmation
- Receipt

External Query (EQ):

- Flight Availability Query

Internal Logical Files (ILF):

- Booking Database

External Interface Files (EIF):

- Payment Gateway

Measurement Parameter	**Count**	**Weighting Factor**	**FP Count**
# of Inputs (EI)	4	4	16
# of Outputs (EO)	3	5	15
# of Queries (EQ)	1	4	4
# of Internal Logical Files (ILF)	1	10	10
# of External Interface Files (EIF)	1	7	7
Unadjusted Function Count (UFC) :			**52**

Complexity Factors: 14 factors, each rated on a scale of 0 to 5, 0 being not important or applicable and 5 being absolutely essential.

SNo.	Complexity Factor	Rate
1	Backup and recovery	4
2	Data communication	3
3	Distributed processing functions	2
4	Is performance critical?	5
5	Existing operating environment	3
6	On-line data entry	4
7	Input transaction built over multiple screens	2
8	Master files updated on-line	4
9	Complexity of inputs, outputs, files, inquiries	3
10	Complexity of processing	5
11	Code design for reuse	4
12	Are conversion/installation included in design?	0
13	Multiple installations	0
14	Application designed to facilitate change by the user	2
	Total	**41**

Using Adjusted FP Count Formula:

AFPC = UFPC * [0.65 + 0.01 * (Total Rate of

Complexity Factors)] AFPC = 52* [0.65 + 0.01 * 41] =

52* [1.06] = **55.12 ≅ 5**

Sprint 4: Admin Panel & User Management

External Input (EI):

- Add/Modify/Delete Flights
- Add/Modify/Delete Promotions
- Block/Unblock User

External Output (EO):

- View Admin Dashboard
- Flight Management Summary
- User Management Summary

External Query (EQ):

- Flight Query for Admin
- Promotion Query for Admin

Internal Logical Files (ILF):

- Promotion Database

External Interface Files (EIF):

- External Promotion Data

Measurement Parameter	Count	Weighting Factor	FP Count
# of Inputs (EI)	3	4	12
# of Outputs (EO)	3	5	15
# of Queries (EQ)	2	4	8
# of Internal Logical Files (ILF)	1	10	10
# of External Interface Files (EIF)	1	7	7
Unadjusted Function Count (UFC) :			**52**

Complexity Factors: 14 factors, each rated on a scale of 0 to 5, 0 being not important or applicable and 5 being absolutely essential.

SNo.	Complexity Factor	Rate
1	Backup and recovery	4
2	Data communication	3
3	Distributed processing functions	2
4	Is performance critical?	5
5	Existing operating environment	3
6	On-line data entry	4
7	Input transaction built over multiple screens	2
8	Master files updated on-line	4
9	Complexity of inputs, outputs, files, inquiries	3
10	Complexity of processing	5
11	Code design for reuse	4
12	Are conversion/installation included in design?	0
13	Multiple installations	0
14	Application designed to facilitate change by the user	2
	Total	**41**

Using Adjusted FP Count Formula:

AFPC = UFPC * [0.65 + 0.01 * (Total Rate of

Complexity Factors)] AFPC = 52* [0.65 + 0.01 * 41] =

52* [1.06] = **55.12** ≅ **5**

Sprint 5: Loyalty Program & Customer Service

External Input (EI):

- Loyalty Points Redemption
- Customer Query Submission

External Output (EO):

- View Loyalty Points
- Customer Query Response

External Query (EQ):

- Loyalty Points Query
- Customer Queries Retrieval

Internal Logical Files (ILF):

- Loyalty Program Database
- Customer Queries Database

Measurement Parameter	**Count**	**Weighting Factor**	**FP Count**
# of Inputs (EI)	2	4	8
# of Outputs (EO)	2	5	10
# of Queries (EQ)	2	4	8
# of Internal Logical Files (ILF)	2	10	20
# of External Interface Files (EIF)	0	7	0
Unadjusted Function Count (UFC) :			**46**

Complexity Factors: 14 factors, each rated on a scale of 0 to 5, 0 being not important or applicable and 5 being absolutely essential.

SNo.	Complexity Factor	Rate
1	Backup and recovery	4
2	Data communication	3
3	Distributed processing functions	2
4	Is performance critical?	5
5	Existing operating environment	3
6	On-line data entry	4
7	Input transaction built over multiple screens	2
8	Master files updated on-line	4
9	Complexity of inputs, outputs, files, inquiries	3
10	Complexity of processing	5
11	Code design for reuse	4
12	Are conversion/installation included in design?	0
13	Multiple installations	0
14	Application designed to facilitate change by the user	2
	Total	**41**

Using Adjusted FP Count Formula:

AFPC = UFPC * [0.65 + 0.01 * (Total Rate of

Complexity Factors)] AFPC = 46* [0.65 + 0.01 * 41] =

46* [1.06] = **48.76** ≅ **49**

- **Estimated Function Points completed per week = 3**
- **Number of developers = 8**
- **Estimated time of completion = Function Points/(3*8)**

Sprints	**Function Points**	**Est. time of completion**
Sprint 1	51	2
Sprint 2	50	2
Sprint 3	56	2.5
Sprint 4	56	2.5
Sprint 5	49	2
Total	**262**	**11**

- **Estimated time of completion of entire project = 262/(3*8) = 10.917 ≅ 11 weeks**