Tech trial - Air Ticket Reservation System (Mingo) - Design document

1. Synopsis

1.1 Goal -

Goal of this documentation is to offer comprehensively detailed information on how to implement a Air ticket reservation system based on the requirements outlined in another document. This document attempts to offer details for the following subsystems required,

1. **Ticket pool management - Core module**
2. **Last mile integration/Payment gateway integration**
3. **Check-in management - Isolated module**
4. **Notification management - Batch processes mostly**
5. **User management - Ancillary module**

1.2 Functional scope -

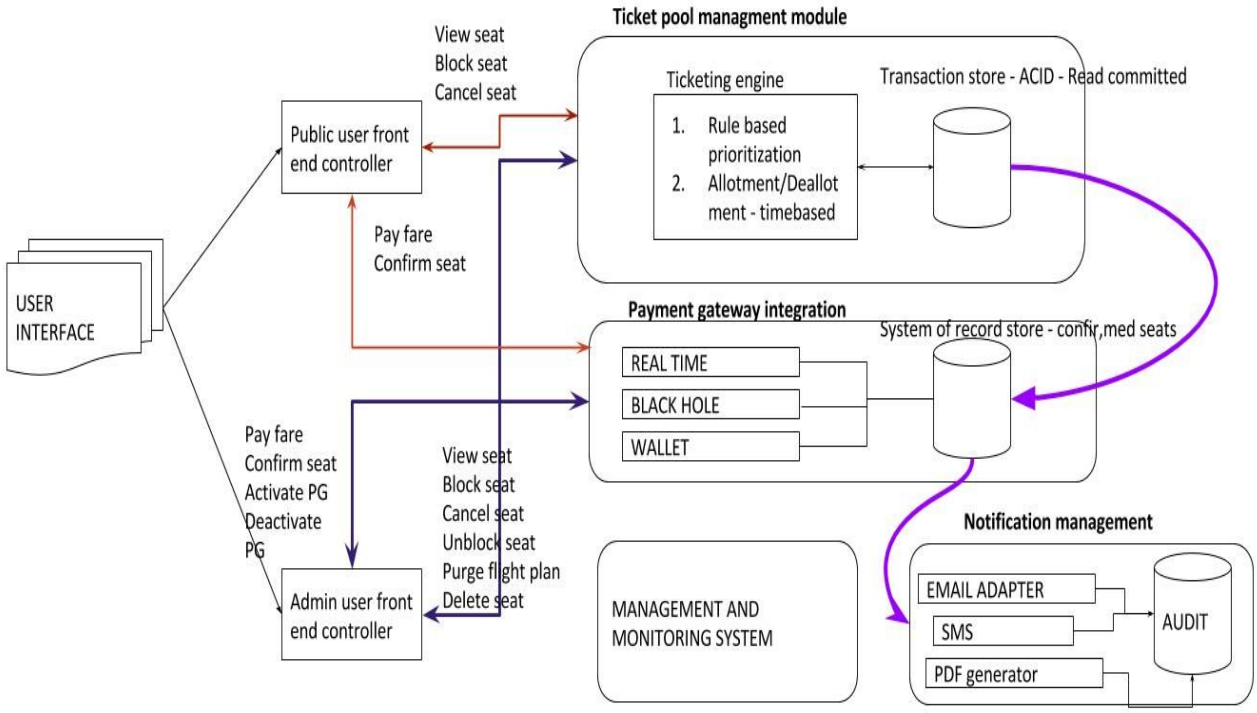
There are perceived modules such as flight plan management based on the understanding that I have currently and such modules won't be addressed part of this documentation. Supporting round-trip reservation and integration with true payment gateway options are not considered in truest essence part of this document.

1.3 Non functional constraints -

Due to time restrictions quoted for the completing the implementation of the project certain non-functional requirements are not taken into consideration. For e.g there would be occasions where peak-load support would require the system to deal with a distributed system's architecture with heavy investments made on conceiving and implementing a fast read-write transactionally safe in-memory cache and such. These considerations are not taken into account, though the interface contracts will ensure that proper abstractions are made available for future upgrades.

2. Architectural overview

Below given diagram should give a simplistic overview about the system design



TODO -

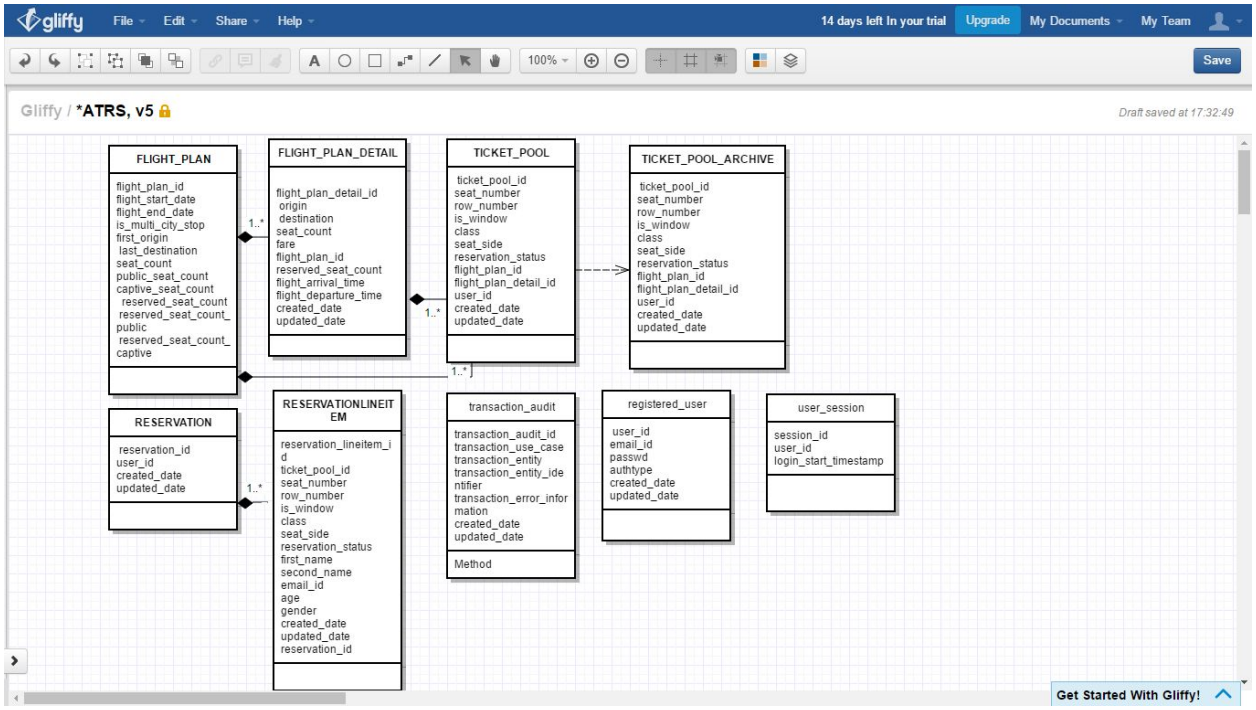
1. Need to add the online check in module interaction along with rest of the components.
2. Need to write a short description about the prescient details about the components.

3. Data design

3.1 Transactional data model - Reservation

Class diagram representing the relationship between the models

https://www.gliffy.com/go/publish/11032689



3.1.1 Flight plan

This datamodel represents the available flights on a particular date and is the parent model for flight plan detail and ticket_pool. Following are the pertinent attributes,

- **flight_start_date** - start date of the flight, user when searches for the available flight, this will help the system to filter the flights.
- **flight_end_date** - end date of the flight, for now just serves as a placeholder.
- **is_multi_city_stop** - since we are supporting multi city stop, this is a flag that tells the system if this flight has multiple stops. just a placeholder for now.
- **first_origin** - the first location from which the flight starts. essential for filtering flights.
- **last_destination** - the last location in which the flight lands and completes this flight plan.
- **seat_count** - total number of seats allocated for reservation on this flight.
- **public_seat_count** - total number of seats allocated for public consumption.
- **captive_seat_count** - total number of seats reserved for airline staff to book.

- **reserved_seat_count** - a transaction accumulator attribute that serves the purpose of a status field. meant to denote how tickets have been booked so far.
- **reserved_seat_count_public** - transaction accumulator meant to represent seats booked through public portal.
- **reserved_seat_count_captive** - transaction accumulator meant to represent seats booked through airline staff portal.

3.1.2 Flight plan line item

This data model represent the individual parts of the flight plan if it has multi city stop. Each of the line item represents an atomic unit of the flight plan.

- **origin** - the location from which this part of the flight plan starts
- **destination** - the end destination
- **seat_count** - number of seats allocated for this leg of the trip. it is important to not let all the seats be cannibalized by legs of the trip.
- **fare** - fare for this leg of the trip.
- **flight_plan_id** - referring to which flight plan it is associated to.
- **reserved_seat_count** - transaction accumulator representing number seats reserved for this leg trip.
- **flight_arrival_time** - time by which the flight will be reaching this location(origin)
- **flight_departure_time** - time by which the flight will be reaching the destination from this origin.

3.1.3 Scheduled ticket pool

This model contains pre-populated number of tickets that are made available for reservation for ticket management system to use. Pre-population of rows is done in order to have a better control of transaction management over the resource. Transactional isolation remains robust when databases are asked to comply with isolations for updates compared to inserts. Hence prepopulation. It's better to be in a place where we face phantom read problems, compared to having two transactions vying for the same ticket, if we follow non-pre-populated insert model.

- **seat_number** - the number of the seat
- **row_number** - the number of the row in the seat
- **is_window** - is it a window seat
- **class** - business class or economy class
- **seat_side** - left side of the plane or right side
- **reservation_status**
    - ***open*** - means that this ticket is available for booking
    - ***blocked*** - means that this ticket is blocked until the user completes his transaction
    - ***updated*** - means that this ticket 's passenger data is being updated,
    - ***confirmed*** - means that this ticket payment is complete,

    - ***cancelled*** - an interim status till the implementation takes the reservation data to another database. meant to represent cancellation status
    - ***open-after-cancelled*** - means that this seat is again made available as someone as relinquished this seat.
- **flight_plan_id** - refers to which flight does seat belongs to
- **flight_plan_detail_id** - refers to which leg of the flight this seat belongs to
- **user_id** - the user who is booking the ticket
- **full_name** - the passenger name
- **primary_passenger** - is he the primary passenger
- **email** - self explanatory

3.2 Transactional model - Payment/Confirmation

3.2.1 Reservation

This model represents a system or record post ticket confirmation. This model should ideally deployed in another data-store that is made highly available even if it's not as performant as the transaction store hosting the ticket_pool data. Following are the pertinent attributes,

- **user_id** - user made the reservation for 'n' number of tickets
- **credit_card_number** - card number used for future reference
- **credit_card_holder** - name of the card holder

3.2.2 Reservation line item

Represents individual tickets that are obtained from ticket_pool.

- ticket_pool_id - the ticket pool id that created this line item
- seat_number - copy of the data in ticket pool
- row_number - copy of the data in ticket pool
- is_window - copy of the data in ticket pool
- class - copy of the data in ticket pool
- seat_side - copy of the data in ticket pool
- reservation_status - can only be confirmed or cancelled. if
- first_name - copy of the data in ticket pool
- second_name - copy of the data in ticket pool
- email_id - copy of the data in ticket pool
- age - copy of the data in ticket pool
- gender - copy of the data in ticket pool
- reservation_id - refers to which reservation it belongs to

3.3 Transactional model - Online check-in

This is an isolated model that only serves the purpose giving a sense of completion to the application. Following are the pertinent attributes,

- **reservation_id** - which reservation does this check-in belongs to
- **user_id** - identity of the user who does the check-in
- **check_in_verification_id** - which identification material that user would be offering
- **check_in_date**

3.4 Transactional model - User session

This is implemented in order to override the session control that current servlet containers. The reasoning is that, the session timeouts have impact on blocked seats that are not confirmed yet. If the session goes invalid, then we need to know that as quickly as possible and relinquish the blocked ticket back to the pool. Following are the pertinent attributes,

- **user_id** - user identifier
- **login_time_stamp** - time of login
- **last_activity -** time when the last activity was performed

3.5 Audit model - Notification sent, Batch process runs

3.2.3 Transaction audit

Since the system is transactionally intensive, it is important to keep note of all the transactions happen with a management perspective, so that we can keep track of data lineage.

- **transaction_use_case** - what type of transaction is it ? - **lookup-flight, lookup-ticket, block, update-p-info, confirm, cancel, generate-ticket, generate-reservation, send-cancellation-email, send-ticket-email**
- **transaction_entity** - which model or set of models does this transaction affects
- **transaction_entity_identifier** - the identifier representing the transaction
- **transaction_status** - exception stack trace clearly stating what is the problem with the transaction if any

3.6 Queue model - Email and PDF Generation and Push

System model meant to host the batch process todo items for consumption. Following are the pertinent attributes.

- **action** - send-ticket-email, send-web-check-in-email, send-cancellation-notice, send-reservation-chart-email
- **submitted_date_time** - time of submission
- **retry_attempt -** number of times the system has tried to perform the operation
- **last_attempted_date_time** - time of last attempt

3.7 Configuration

3.7.1 Cancellation time threshold - system wide property meant for letting the user cancel his ticket, beyond which he won't be allowed to do so. Value would be 4 hours before travel.

3.7.2 Blocking threshold - how long does a ticket can be in blocked state. Even if the user is actively trying to block tickets for another travel, we ought to have a threshold beyond which the ticket needs to be pushed into the pool. Default value will be 5 minutes.

3.7.3 Active payment gateway adapter

3.7.4 Full flight plan seat percentage

3.7.5 Ticket pool rule box type - priority / fifo

4. Sub system details

4.1 Ticket pool management

4.1.1 Processing narrative

1. Selecting Travel date, Origin and Destination is mandatory
2. Upon selection system finds
    a. The seat template for the flight plan(Default 1x2)
    b. Currently Reserved Seats

c. Currently available seats available for the selected Origin and Destination
d. User selects the seat
e. System updates this seat entry in ticket pool
f. System adds a new entry in Reservation and one to many(line items) in the reservation table as per the number of tickets selected
g. User fills in the passenger details
h. User completes the payment successfully
i. System updates new entry in Reservation
j. System moves appropriate ticket pool entry to Archive table(DELETE/INSERT)
k. User acknowledged with Ticket Info
l. Increment Reserved seat count in Flight Plan and Detail
m. When another user(User2) accesses the system,
    i. User1 is within step 2.c, it shows the same info as User1
    ii. User1 is in step 2.d, then it shows the seat is blocked by User1 as "Booking In Progress"
    iii. When User1 is in step 2.h, then it shows the seat blocked by User1 as "Booking in Progress"

If Booking fails because,

1. User session times out or payment fails for both systematic and app failures
2. System will update ticket status as "Open as transaction failed"
3. There will be a transaction audit that will have history of such failed transactions.

If Booking is cancelled,

1. A new ticket entry will be added
2. Reserved seat count should be decremented
3. Reserved seat row should be moved into Archive Table(DELETE/INSERT)

4.1.2 Boundary condition behavior (application/system)

1. A user cannot select more than 4 seats
2. No past dates should be honoured

4.1.3 Interaction with other components

- It needs to interact with transactional database to obtain required information for ticket management purpose.
- It needs to interact with system of record database to obtain required information for reservation and cancellation purpose.

4.1.4 Service Interface contracts

Input contract

Output contract

4.2 Payment gateway integration

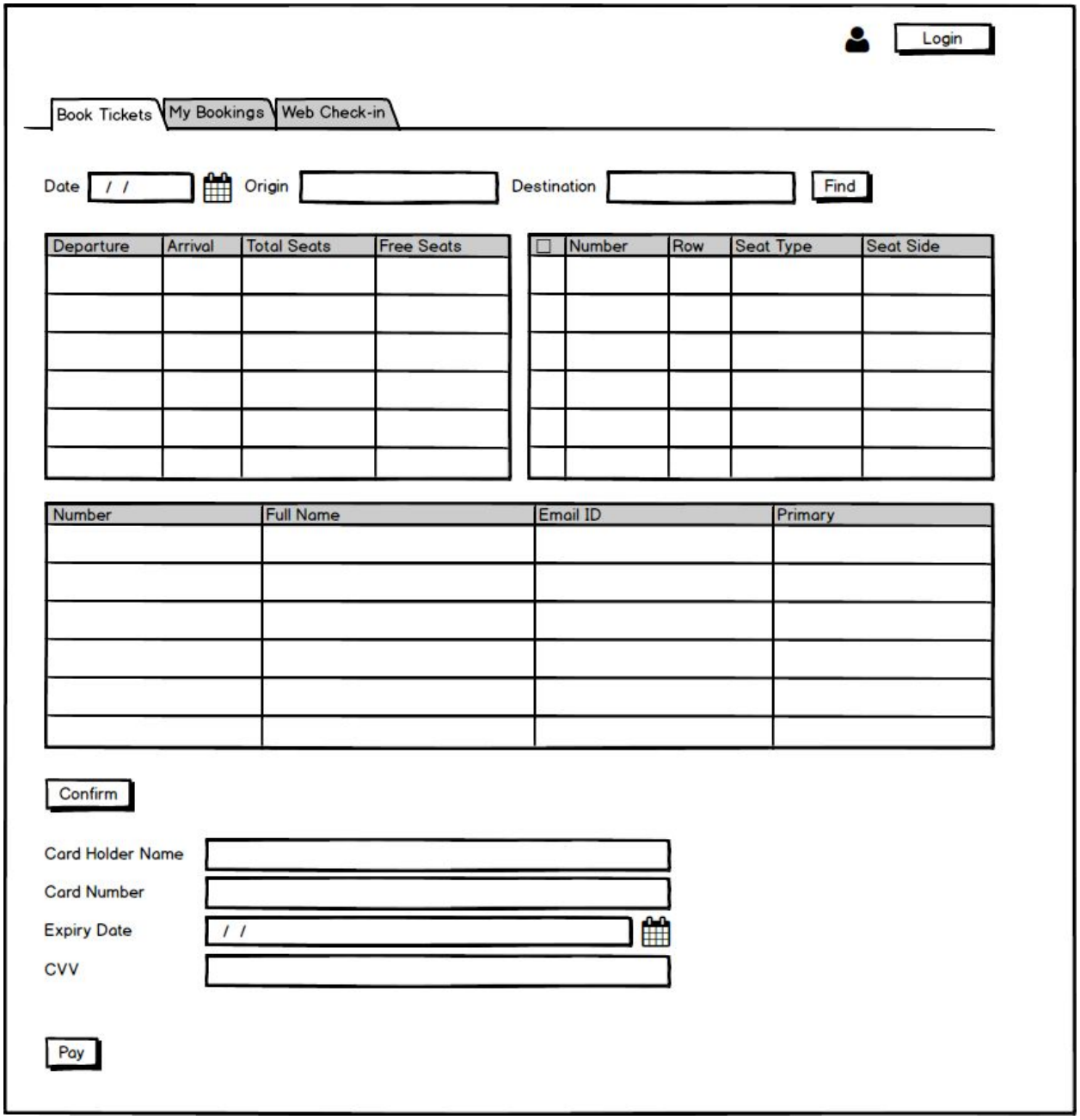
4.3 Online checking management

4.4 Notification management

4.5 User management

5. UI Design

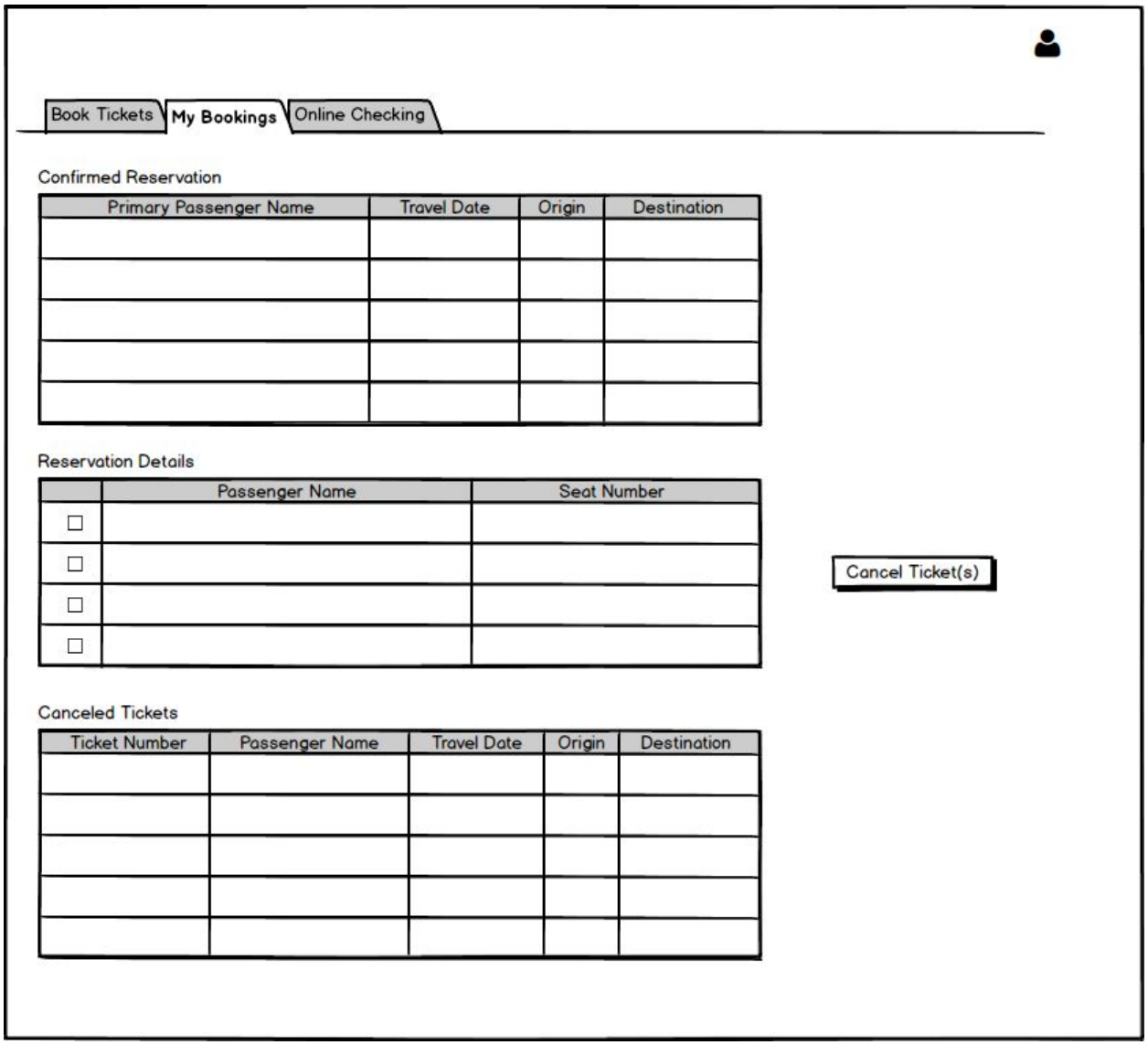
5.1 Public user - reservation screen


Login
Book Tickets
My Bookings
Web Check-in
Date
/ /
Origin
Destination
Find
Departure
Arrival
Total Seats
Free Seats
Number
Row
Seat Type
Seat Side
Number
Full Name
Email ID
Primary
Confirm
Card Holder Name
Card Number
Expiry Date
/ /
CVV
Pay

5.2 Public user authentication dialog

5.3 Public user booking confirmation email

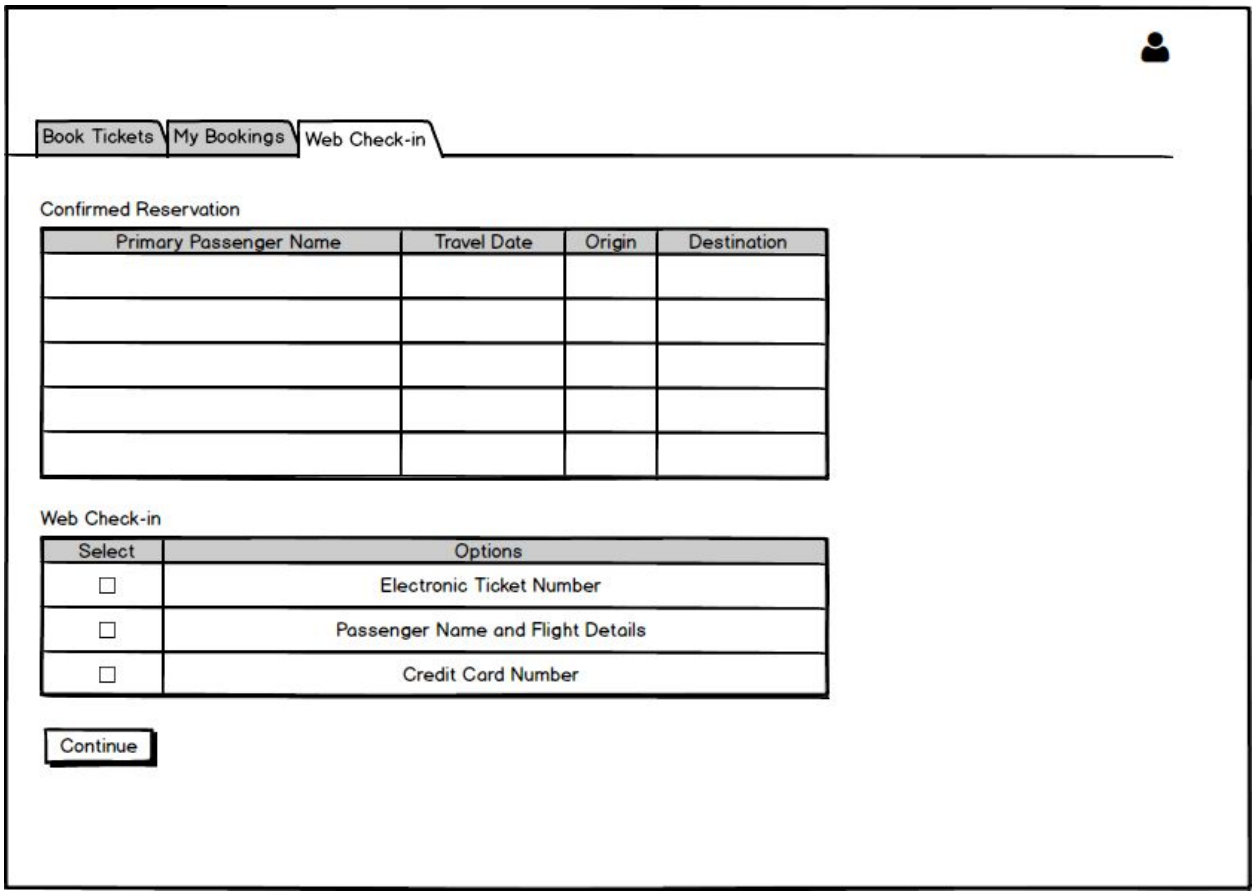
5.4 Public user cancellation screen



5.5 Public user cancellation email

5.6 Public user web-check in email

5.7 Public user web-check in screen



5.8 Airline staff reservation chart email

5.9 Airline staff ticket cancellation screen

5.10 Airline staff flight reservation chart screen

6. Resource considerations

Ticket Pool is sensitive resource(Lightweight, Durable)
Reserved Seats - System of Record(Replicated, Highly Available)

7. Deployment details

8. Instrumentation and monitoring

9. Technology stack - detailed justification

9.1 User interface - SmartGWT

9.2 Middleware - Spring IOC, JDBC template and Tomcat server

9.3 Database - MySQL

9.3.1 Innodb engine for transactional store and configuration store

9.3.2 MyISAM engine for audit store

10. Framework best practices

11. TO DO

1.SESSION TIMEOUT should be implemented as a batch/daemon process
2.The entire reservation data set needs to be made as per UTC timezone.
Need to add flight number