CrossOver
Design Document for Airline Ticket Reservation System
Author : Mário de Sá Vera
Date : May 9th 2016

This document describes the high level design elements of a basic Airline Ticket Reservation System (from here one referenced as “the system”). Requirements will be presented and detailed. In the architecture session an overall scenario is presented and each module of the architecture is described with UML and Flow diagrams as well as a short description of its behaviour in the system and the techniques that should be applied for clarity. Along with the technical description some suggestions and references for a specific technology implementation are made in the footnotes.

1. High Level Requirement Analysis

The requirements are divided in Functional (divided in user and staff) and Operating requirements and will be attended by the architecture in the data model and use cases :

a. Functional Requirements

- REQ_FUNC_USER_LOGIN : users are able to login using Google+ , Twitter and LinkedIn accounts for system authentication in the system login user interface.
- REQ_FUNC_USER_FLIGHT_SEARCH : users are able to search for multiple flights based on multiple criterias through a the flight search user interface and after authentication create or manage their reservations from the search results.
- REQ_FUNC_USER_RESERV : authenticated users are able to place a reservation to a specific flight from a search result using a booking user interface where he will be redirected for credit card payment after selection of the desired seats in the flight. The reservation will only be accepted under payment confirmation. After reservation acceptance by the system the user should receive a Booking Confirmation Email (i.e. BCE for short) with the booking details in PDF format and links for Checkout,Cancel and My Bookings to their respective system user interface that will enable user interaction without requiring authentication at anytime.
- REQ_FUNC_USER_RESERV_CANCEL : users will be able to cancel a booking from 48 hours up to 4 hours before the flight departure either through the link provided by the BCE or directly (after authentication) through mybookings user interface. The cancellation will affect the number of seats selected for the specific reservation.

- REQ_FUNC_USER_RESERV_CHECKOUT : users will be able to checkout a booking from 48 hours up to 4 hours before the flight departure using the mybookings user interface either through the link provided by the BCE or directly after authentication. After checking out the browser will render the flight seats choice user interface and after choosing the seats the checked out reservation printing action will be enabled at the mybookings user interface.
- REQ_FUNC_USER_RESERV_SEARCH : users will be able to list all their bookings at any time accessing the mybookings user interface either through the link provided by the BCE or directly after authentication.
- REQ_FUNC_USER_CHECKOUT_ALERT : the user will receive a checkout alert email (i.e. CAE) 48 hours before flight departure with a link for that booking checkout not requiring any authentication.
- REQ_FUNC_ADMIN_LOGIN : staff members will be able to login using Microsoft Directory Credentials (i.e. MDC) from any computer connected to the internet.
- REQ_FUNC_ADMIN_RESERV_CHART_ALERT : staff members will receive reservation charts in PDF format 1h before flight departure. This will be a Departure Alert Email (i..e. DAE).
- REQ_FUNC_ADMIN_RESERV_CANCEL : staff members will be able to cancel any reservation at any time. This action should trigger a notification to the user confirming the cancellation and the payment refund. This will be a Refund Alert Email (i.e. RAE). This action can regard a specific reservation or all reservations of a specific flight and will be performed using the mybookings user interface just like a normal user or the flight search user interface that in this case will have the cancellation action available for staff members only.

b. Operating Requirements

- REQ_OPER_SPA : in order to guarantee a good performance the system should be developed as a simple page web application portal promoting a reduced network data traffic between client user interface and server side services making the system more responsive.
- REQ_OPER_SERVER_FLEX_ARCH : the server side services will form the system core and should operate over a mobile client user interface scenario without any modification.
- REQ_OPER_SERVER_CLOUD_ARCH : the server side services will be deployed in an absolutely heterogenous environment where each service can be physically installed in separate data centers anywhere in the world (i.e. Cloud Computing).
- REQ_OPER_ALERT_HAND_SHAKE : all the system alerts delivery should be confirmed.

2. High Level Data Model

The high level data model consists of two basic modules. The first module is a the CORE reservation model responsible for the reservation schedule persistence :

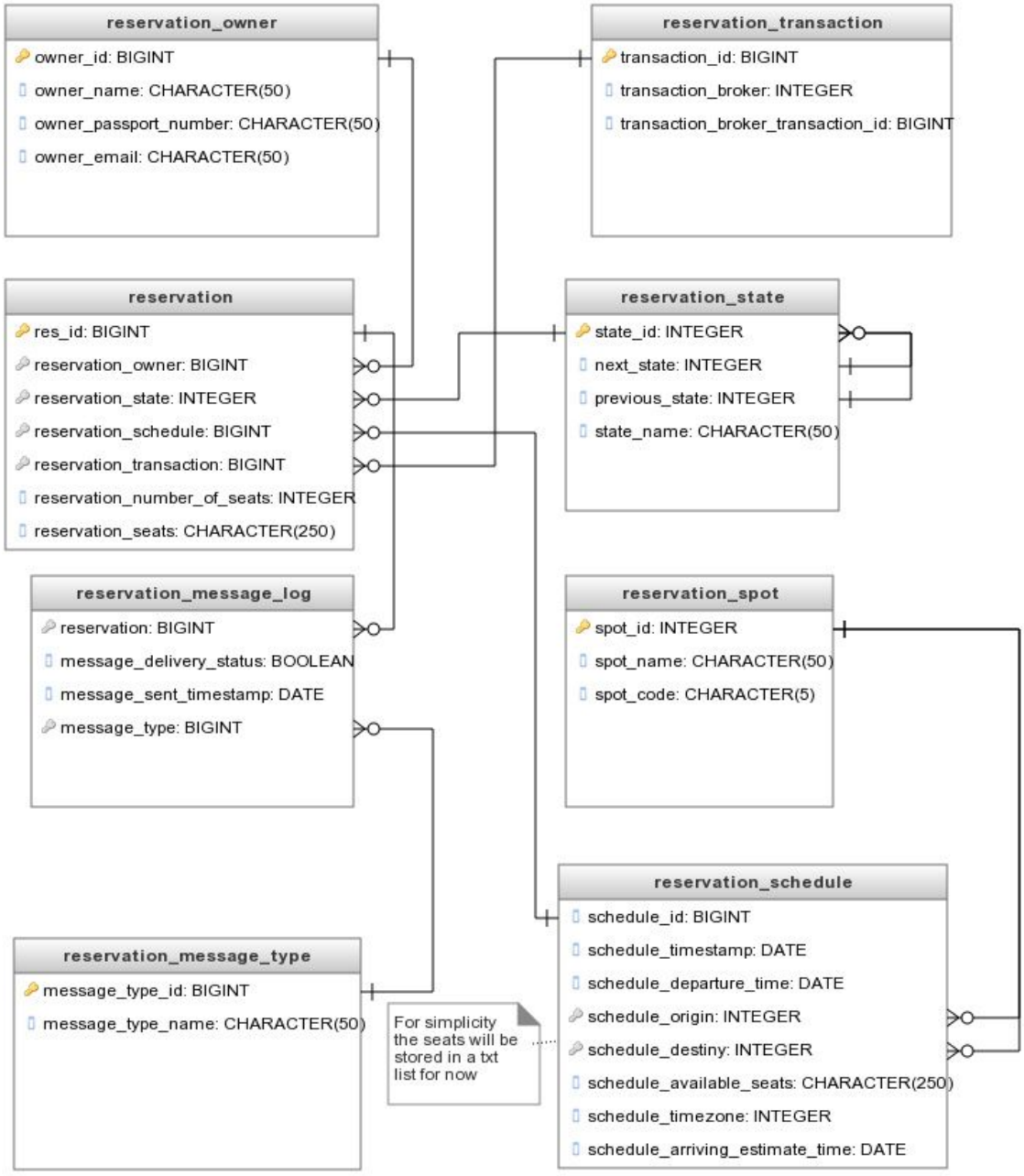

Figure 1 - CORE DATABASE MODEL

The second module is responsible for session control for user navigation on the system :

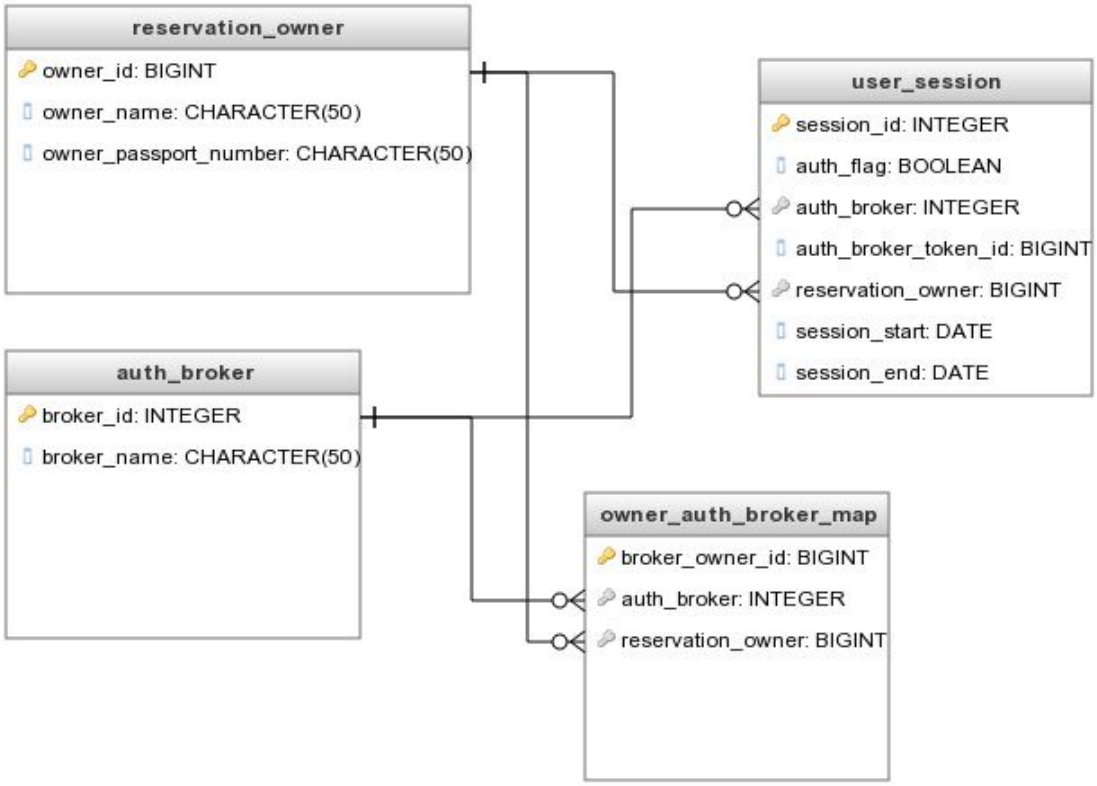

Figure 2 - USER SESSION DATABASE MODEL

3. Architecture Diagrams

The Architecture Diagrams will detail the workflow of events between the user (either normal user or staff admin) and the system showing the communication model of messages and logical infrastructure of each service module as well as key components design will be also detailed for a clear comprehension of the overall system architecture. In order to achieve this level of detail the Functional and Operating Requirements will serve as a guide being organized as system use cases and referenced such that by the end of this session there is a confidence on the architecture to be addressing each and every system requirement.

a. Overall Architecture

The system overall architecture consists of a distributed cloud based Web Services implemented in JAVA and deployed in a load balanced cluster module[1] each and connected through a Messaging BUS[2] forming the System CORE. In order to favour data integrity we will use a Relational Database running on a remote data center and a JPA provider (i.e. JAVA Persistence API) implementation for data model mapping.

The system API Gateway controls user authentication and is the first access point from client side user interfaces delegating in a synchronous fashion (i.e. through HTTP request forward) the client requests over to the proper Web Service using AJAX calls in a RESTful[3] style and receiving JSON data back to the Client browsers[4] that will render the SPA and conduct user interaction on the client side.

The Web Services communicate through messaging over the Messaging BUS in an asynchronous fashion by posting proper notifications through messaging topics. A Scheduling Watcher Service is part of the system services taking care of system alerts notifications by polling the database and posting alert calls for proper messaging listeners components.

External services are necessary for user login authentication,credit card payment and mailing and are abstracted in the 3rd Party Services API Gateway module. The conversation with the 3rd Party Service API Gateway using whatever API it might offer though being a HTTP based API is a primary requirement for service choice.

---

[1] https://aws.amazon.com/pt/about-aws/whats-new/2010/07/13/announcing-cluster-compute-instances-for-amazon-ec2/

[2] www.aws.**amazon**.com/sns

[3] http://rest.elkstein.org/

[4] JSON was chosen for it is a very clear data exchange format and is Javascript based.

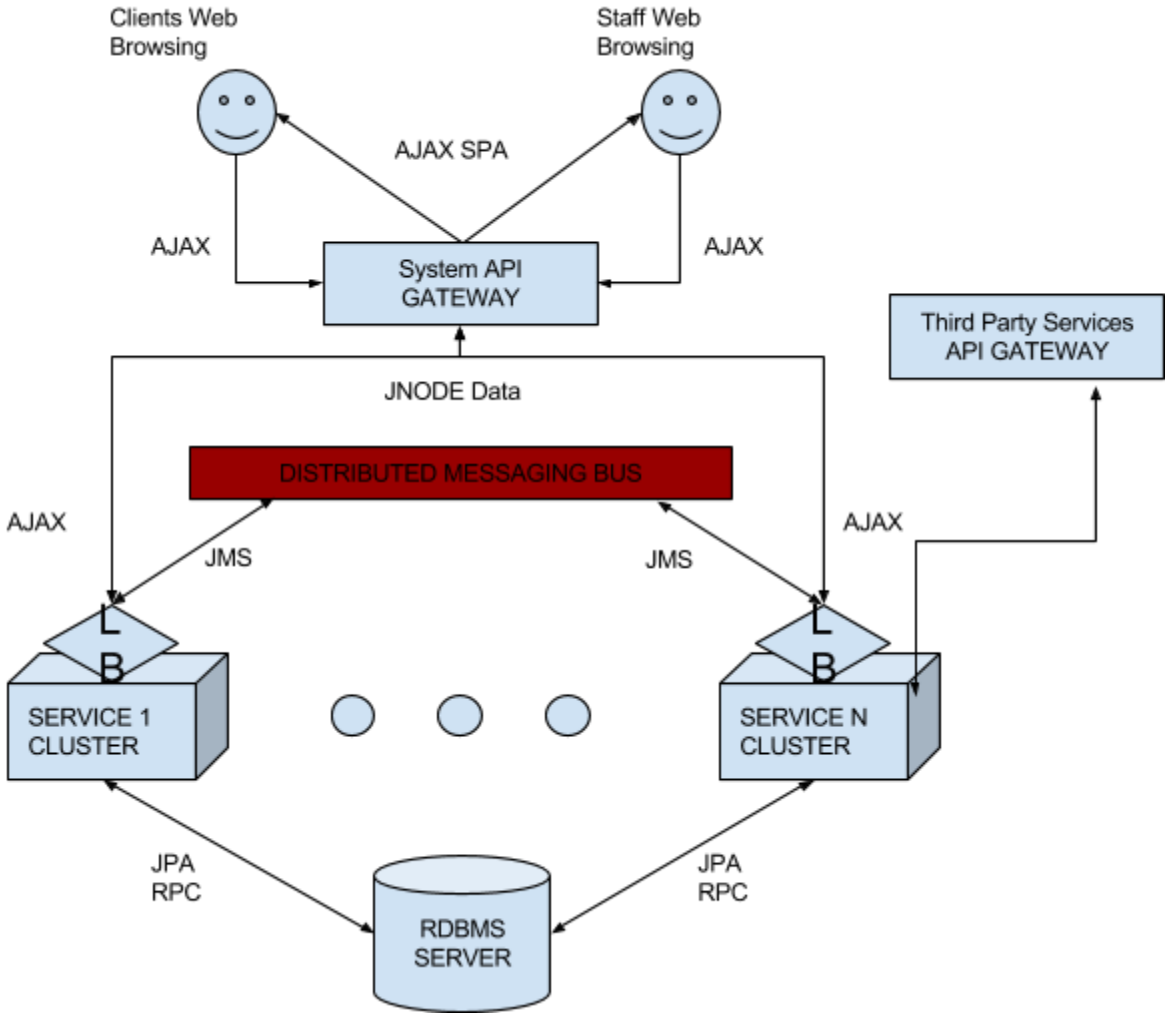

Figure 3 - Overall System Architecture (each arrow means a remote call over the net)

- Client Applications

System users will interact through Internet Browsers and in order to address REQ_OPER_SPA will be developed in Javascript (using any Javascript based library or framework supported by the more common web browsers). This choice is due to the fact that with client side processing we are able to reduce significantly data communication

between network endpoints (i.e. Client browsers and the system API Gateway) and make the user experience more responsive against old fashioned pure HTML web applications with HTTP redirections.

The client request response data from the Web Services is forwarded back to client browsers in a SPA (i.e. Single Page Application) in JSON Data format addressing REQ_OPER_SERVER_FLEX_ARCH as the whole application is loaded by browsers in each access with all logic necessary for execution of each system use case.

- The API Gateway

The System API Gateway is the system entry point responding for web browsing hits as well as the orchestrator (i.e. Proxy) of the business logic posting messages from system users browsing. It receives AJAX posts using RESTful URLs indicating the proper action and accesses the respective Web Service in order to build the SPA code with the returning JSON Data as detailed in Figure 3. From an implementation point of view it is a basic JAVA Servlet component :

Figure 4 - The system APIGateway Servlet

- The Messaging BUS

With a distributed messaging BUS in place we mean to achieve high cohesion and low coupling over the system modules. A change on a specific service module will have very low impact on the overall system with this architecture and throughput of the API Gateway is maximized.

The messaging implementation suggested is JMS[5] (i.e. JAVA Messaging Service) and there will be functional components implemented as Messaging Beans that will take care of the user alerts sending by email. These components will live in the Messaging BUS network realm and will listen to the following topics posted by the Web Services in an action conclusion as well as by the Schedule Alert Service as needed :

- CAE_ALERT : posted by Scheduling Watcher Service after polling the database after departures taking place as specified in REQ_FUNC_USER_CHECKOUT_ALERT the CAE the CAEMailerBean attends to this alert and reads from Message argument the reservation owner details and adds the internal system user token to the mybookings user interface URL link as part of the e-mail body. The 3rd party mailing service will be responsible for the e-mail delivery.
- BCE_ALERT : posted by Booking Service as soon as booking is completed (i.e. after payment confirmation). This alert is attended by BCEMailerBean which reads from Message argument all the reservation details and creates the proper mail message with the booking details in PDF format by accessing the PDFWriterBean and adds the internal system user token to the proper Services URL links with as required by REQ_FUNC_USER_RESERV.
- DAE_ALERT : posted by Scheduling Watcher Service after polling the database after departures taking place as specified in REQ_FUNC_ADMIN_RESERV_CHART_ALERT the DAEMailerBean attends to this alert and reads from Message argument the list of departures and mails all staff members queried to Authentication Service (i.e. delegating to LDAP query for AD Service) using the 3rd party mailing service.
- RAE_ALERT : posted by Booking Cancellation Service will be attended by RAEMailerBean and access proper 3rd Party Services for payment refund with data read from Message argument mailing the reservation owner afterwards.

[5] http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html

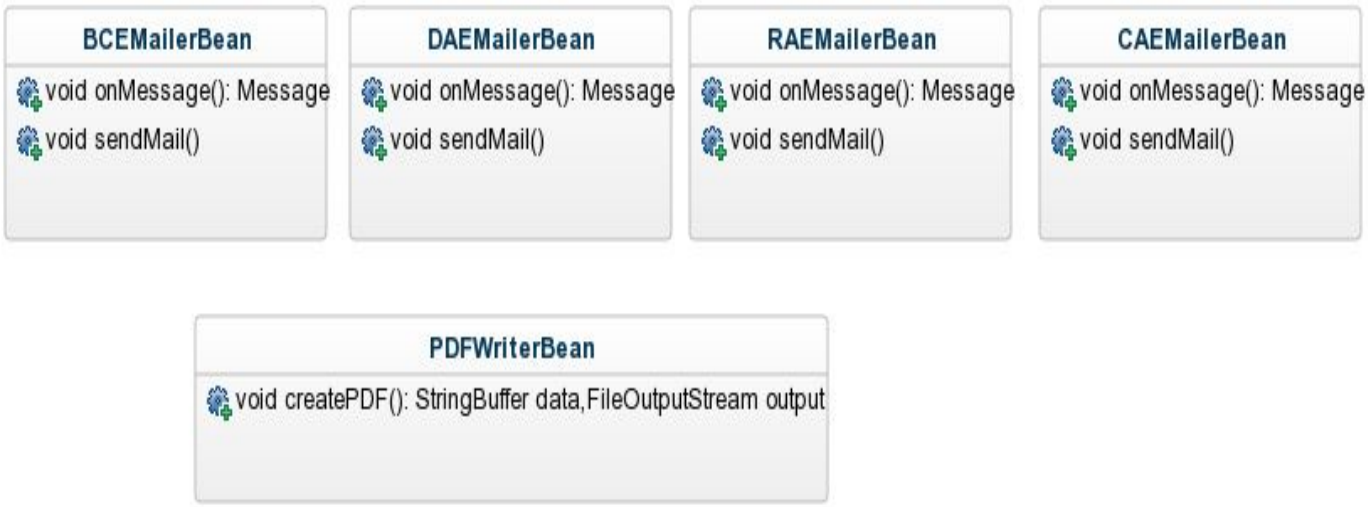

Figure 5 - Messaging Beans and the PDFWriterBean utility components

An important requirement of the system regarding alert delivery is REQ_OPER_ALERT_HAND_SHAKE and in order to address it the system will have the Alert Confirmation Service also listening to all [ALERT NAME]_ALERT POSTs and inserting a row in reservation_message_log with the alert identifier and a boolean flag for user confirmation of the alert. The Alert Confirmation Service URL will be the one registred in 3rd party guaranteed mailing service[6] that will take care of deliverability and notify back the delivery and not requiring any action from staff members. After a certain

[6] http://www.puremail.com/bundledSolutions.php

time of DAE notifications a staff member could access a non confirmed DAE list in a user interface filled up by the Alert Confirmation Service after doing an overall flight reservations failed DAE confirmation check and going after reservation owners through other communication channel rather than e-mailing.

All Services will follow the implementation template of a JAVA Servlet receiving AJAX posts in specific REST actions (according to each system use case) and accessing business logic routines and database (through JPA) for building up JSON Data responses to SPA browsers to render the proper user interface code.

Figure 6 - A System Service Servlet

The following Web Services and their respective REST URL compose the system services list :

- Authentication Service : /api-gateway-url/auth
- Booking Service : /api-gateway-url/booking
- Booking Checkout Service : /api-gateway-url/checkout
- Booking Cancellation Service : /api-gateway-url/bcancel
- Booking Search Service : /api-gateway-url/bsearch
- Flight Search Service : /api-gateway-url/fsearch
- Flight Cancellation Service : /api-gateway-url/fcancel

In order to accomplish the operating requirements some utility services were necessary :

- Alert Confirmation Service

An internal utility service responsible for alert confirmation callbacks from 3rd party mailing services.

- Scheduling Watcher Service

Alerts for user notification will be sent by the Scheduling Whatcher Service through a polling mechanism accessing the database looking for the list of alerts to be sent and using the Messaging BUS to post on respective topics.

- Network Realms

In order to have a secure and scalar architecture we recommend the placement of some services in the same network realm (or data center) : Authentication Service , AD Server, the Messaging BUS and the Database. All other Services will be deployed in separate subnets for maximum system performance.

b. User Login Use Case

In order to address REQ_FUNC_USER_LOGIN the Facebook , Twitter and LinkedIn authentication entities will be accessed (i.e. Third Party Service API Gateway module in Figure 3) and acting as identity providers following the SSO (i.e. SAML protocol for Single Sign On[7]) nomenclature and the system Authentication Service will act as the service provider and generate a user session token and persist it therefore not requiring the system to store user sensitive data. Less sensitive user data like name and some other information might be persisted at this step as well upon user explicit access grant for the system (i.e. the Application on OAuth[8] protocol nomenclature). This session token will be the one sent as part of the BCE addressing part of REQ_FUNC_USER_RESERV as every API Gateway hit using this token as part of the URL, for example, will bypass user authentication :

[7] https://developers.google.com/google-apps/sso/saml_reference_implementation
[8] https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

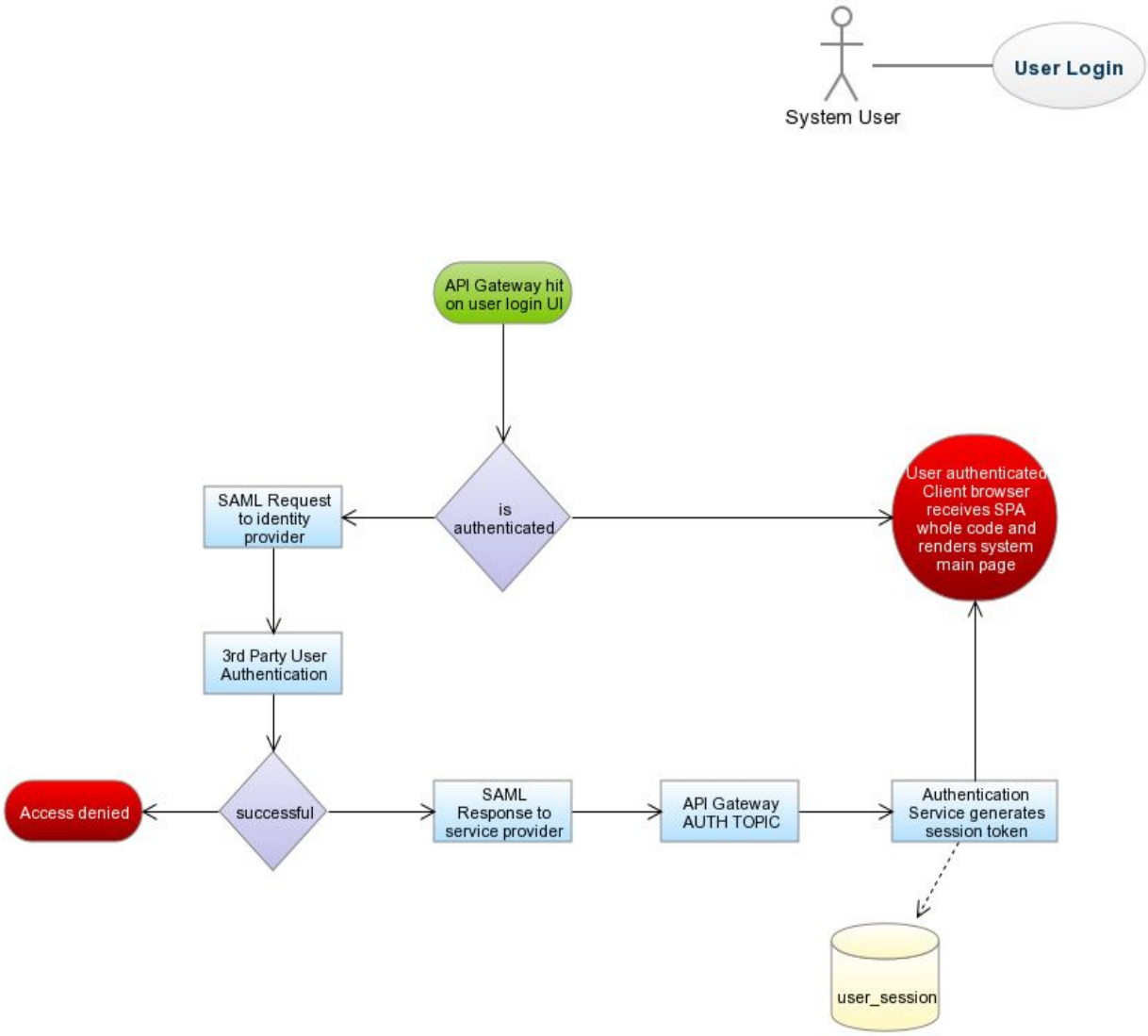

Figure 7 - User Login flow

The Authentication Service is implemented as a JAVA Servlet :

Figure 8 - The Authentication Servlet

c. Staff Admin Login Use Case

In order to address REQ_FUNC_ADMIN_LOGIN a Microsoft Active Directory server will have to be installed in the same network realm (i.e. intranet) of the Authentication Service as it makes usage of LDAP[9] and will need loca file system access. The architecture is very simple and it basically will authenticate the admin user making a simple HTTP Request to the AD server sending the password typed by the user at the login user interface validating the access or not.

d. Flight Search Use Case

In order to address REQ_FUNC_FLIGHT_SEARCH a data indexing tool[10] utility will be useful as it will cache the flight catalogue in the database (i.e. reservation_schedule) and accelerate the results as well as offering a very flexible criteria based searching engine for the system Flight Searching Service. System users will be able to search for flights in the flight search user Interface using search tags like flight destination city names and so on :

[9] https://pt.wikipedia.org/wiki/LDAP
[10] https://en.wikipedia.org/wiki/Apache_Solr

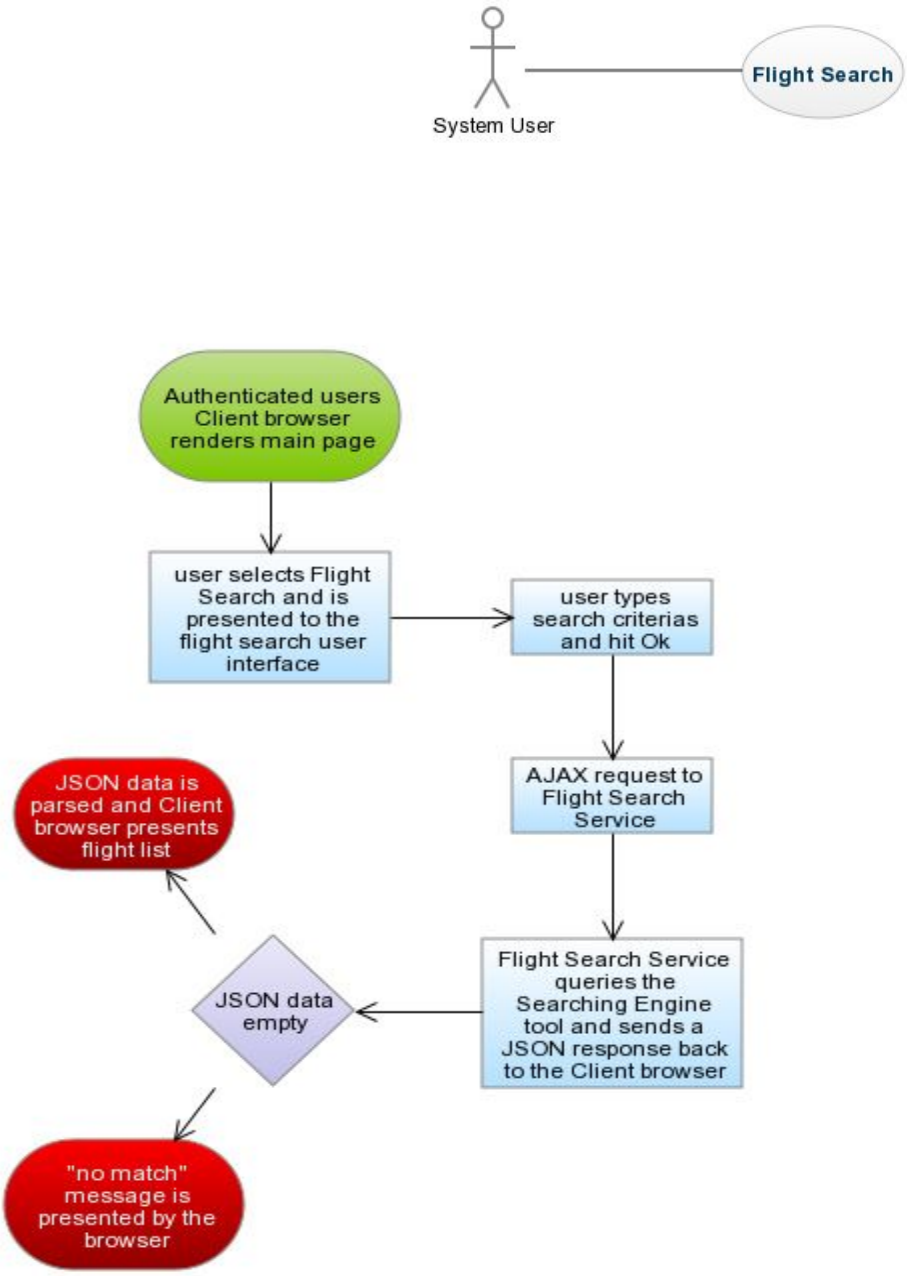

Figure 9 - Flight Search flow

e. Booking Use Case

Booking a reservation is the main system functionality and addressing REQ_FUNC_USER_RESERV authenticated users, after selecting a specific flight from Flight Search use case, will be interacting with the system through the booking user interface and be able to place a reservation with as many seats as available for that flight. The seats selection is part of the booking user interface and no server access is necessary attending to REQ_OPER_SPA the whole interface is loaded in the main page up front being necessary only “code on demand” access to the Booking Service in order to render the booking user interface properly (the system, with this approach, is assuming that only after BCE will the user be confirmed on his reservation as the seats available information is a dynamic data and can change in the meantime of user booking delay). After selecting the flight seats user is redirected to 3rd party credit card payment service and after payment confirmation is received the system will send BCE to user :

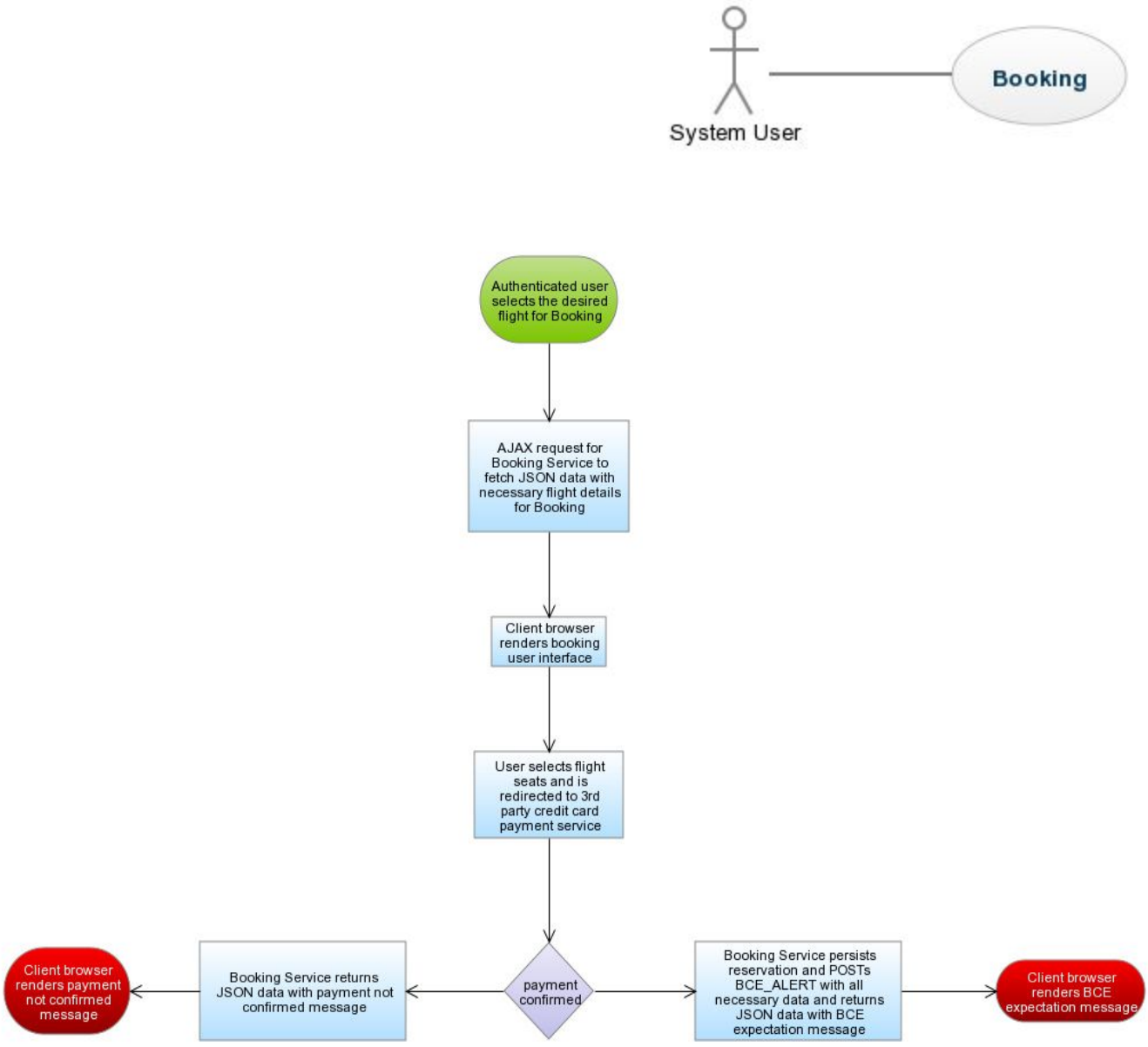

Figure 10 - Booking flow

f. Booking Cancellation Use Case

In case the user decides to cancel a reservation he will interact with the mybookings user interface and be able to cancel one or more reservations. The system will make sure the timeframes are valid following REQ_FUNC_USER_RESERV_CANCEL rules. Cancellation can also be performed by staff members and will address REQ_FUNC_ADMIN_RESERV_CANCEL rules as well :

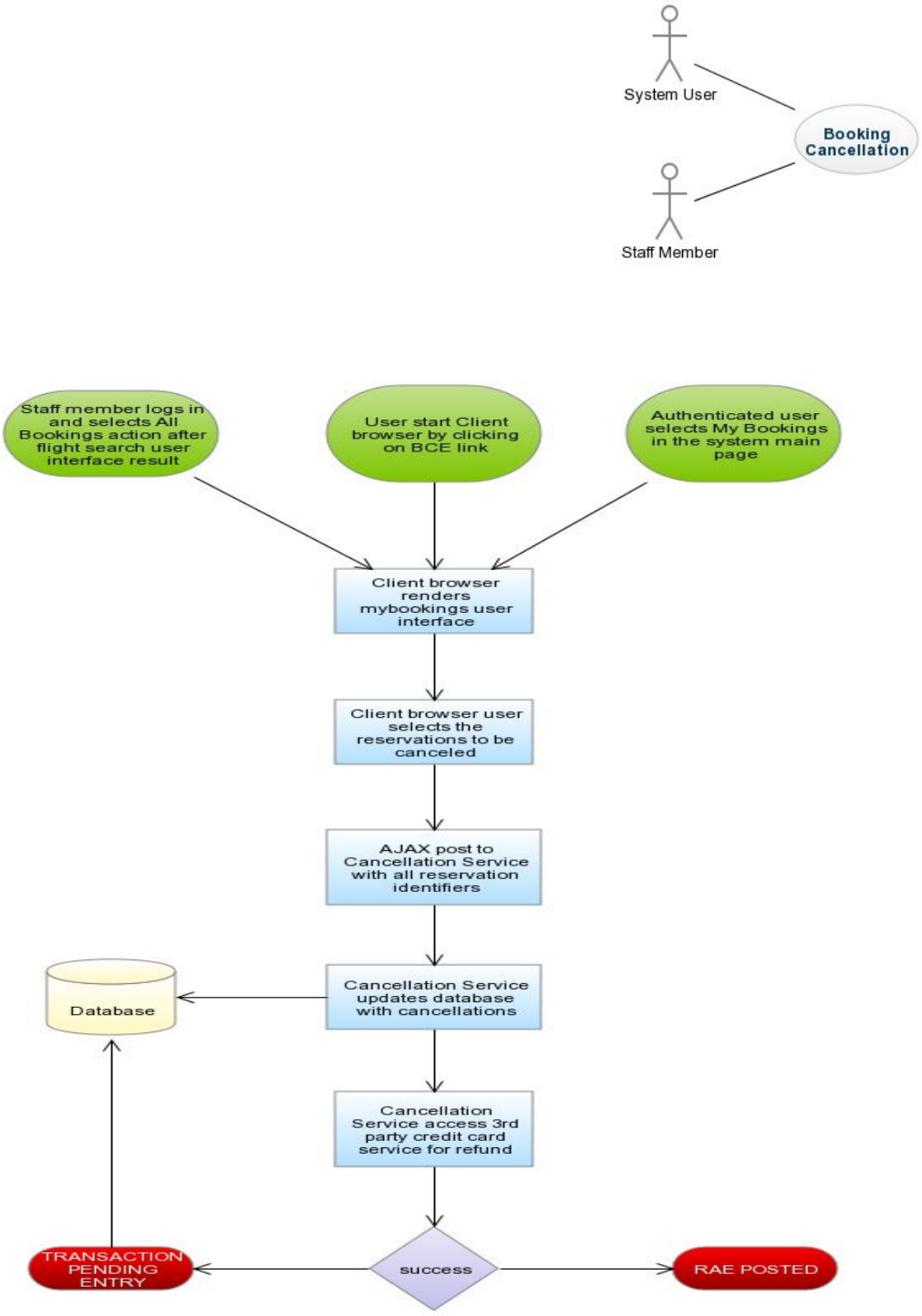

Figure 11 - Booking Cancellation flow

g. Flight Cancellation Use Case

In case a staff member must cancel all reservations from a flight the system will operate in a loop over Booking cancellation use case after browsing over flight search user interface as a staff member that will render the “Cancel All” interface action sending AJAX post to Flight Cancellation Service with the specific flight identificator.

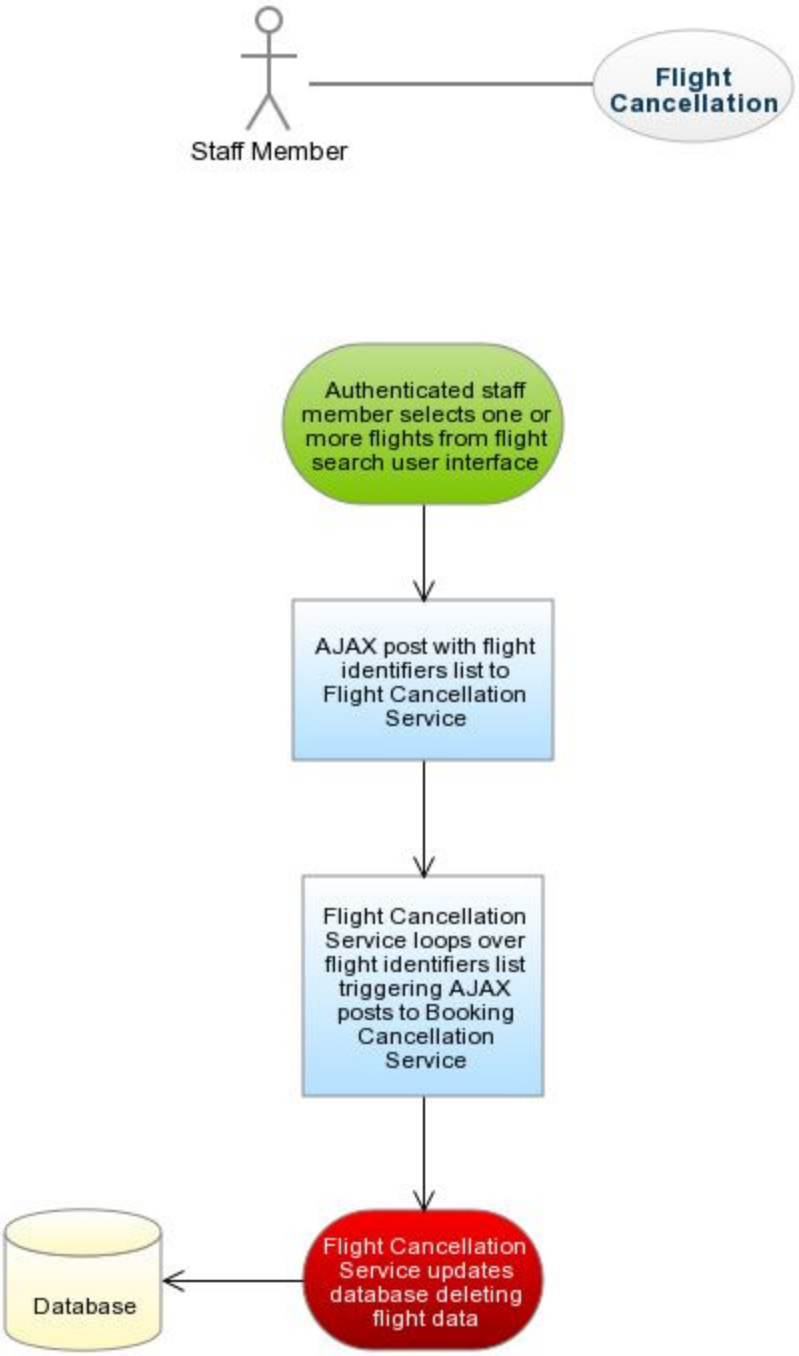

Figure 12 - Flight Cancellation flow

h. Booking Search Use Case

In order to attend REQ_FUNC_USER_RESERV_SEARCH the system will provide together as part of the SPA the mybookings user interface code. Either through direct User Login or by interacting with the BCE the user will be able to list all his reservations and their respective status. This functionality is also available for staff members and in both cases a Booking Search Service will be responsible for building up the proper JSON Data for API Gateway user interface construction and therefore attending to REQ_OPER_SERVER_FLEX_ARCH :

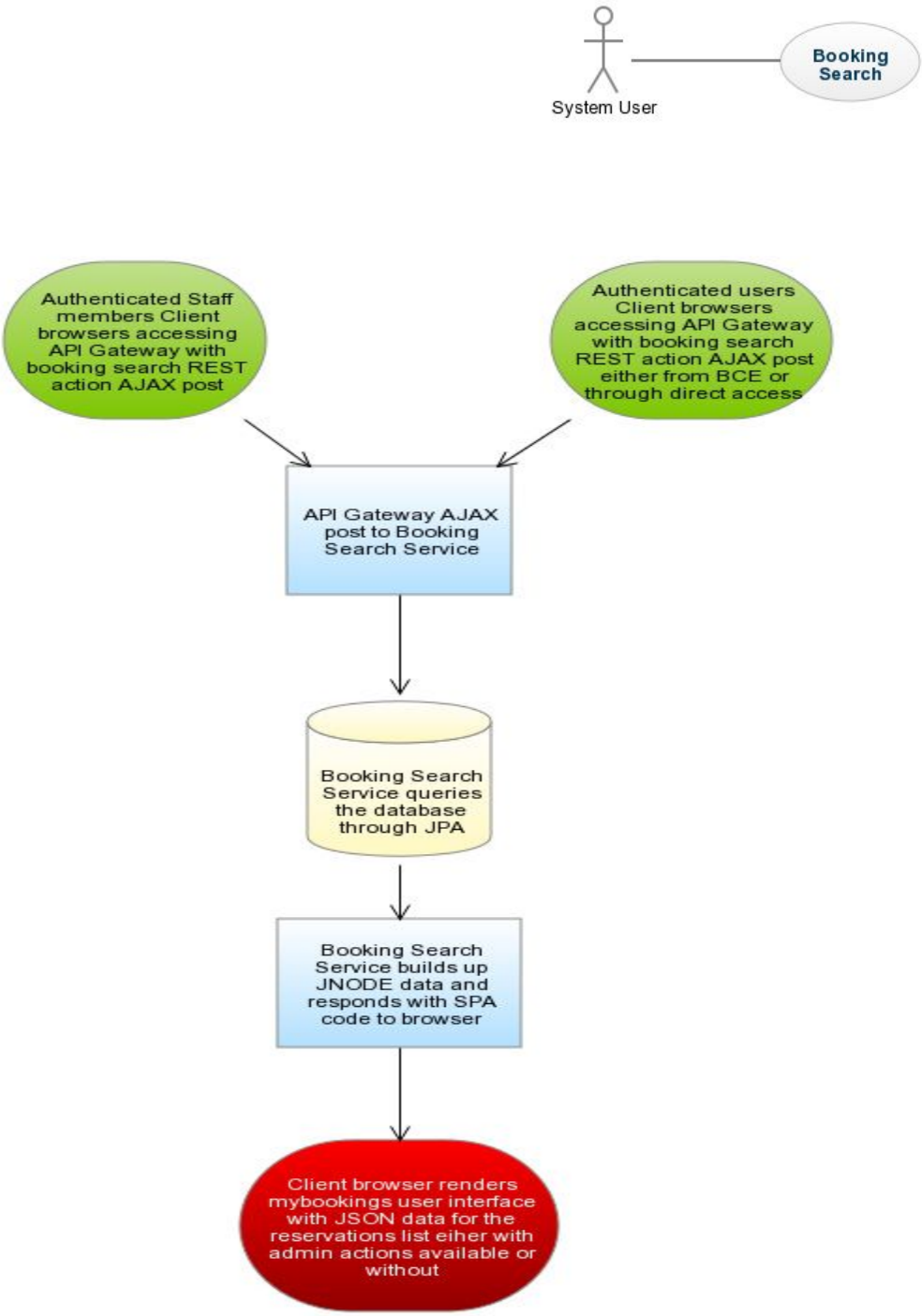

Figure 13 - Booking Search flow

i. Reservation Checkout Use Case

As specified by A REQ_FUNC_USER_CHECKOUT_ALERT , users will receive an e-mail with a link to the system API Gateway indicating the REST checkout use case action and appending the user token to this URL and therefore not requiring any user authentication. Client browser will render mybookings user interface with the specific booking identifier data enabling the user to checkout the reservation by interacting with the Booking Checkout Service :

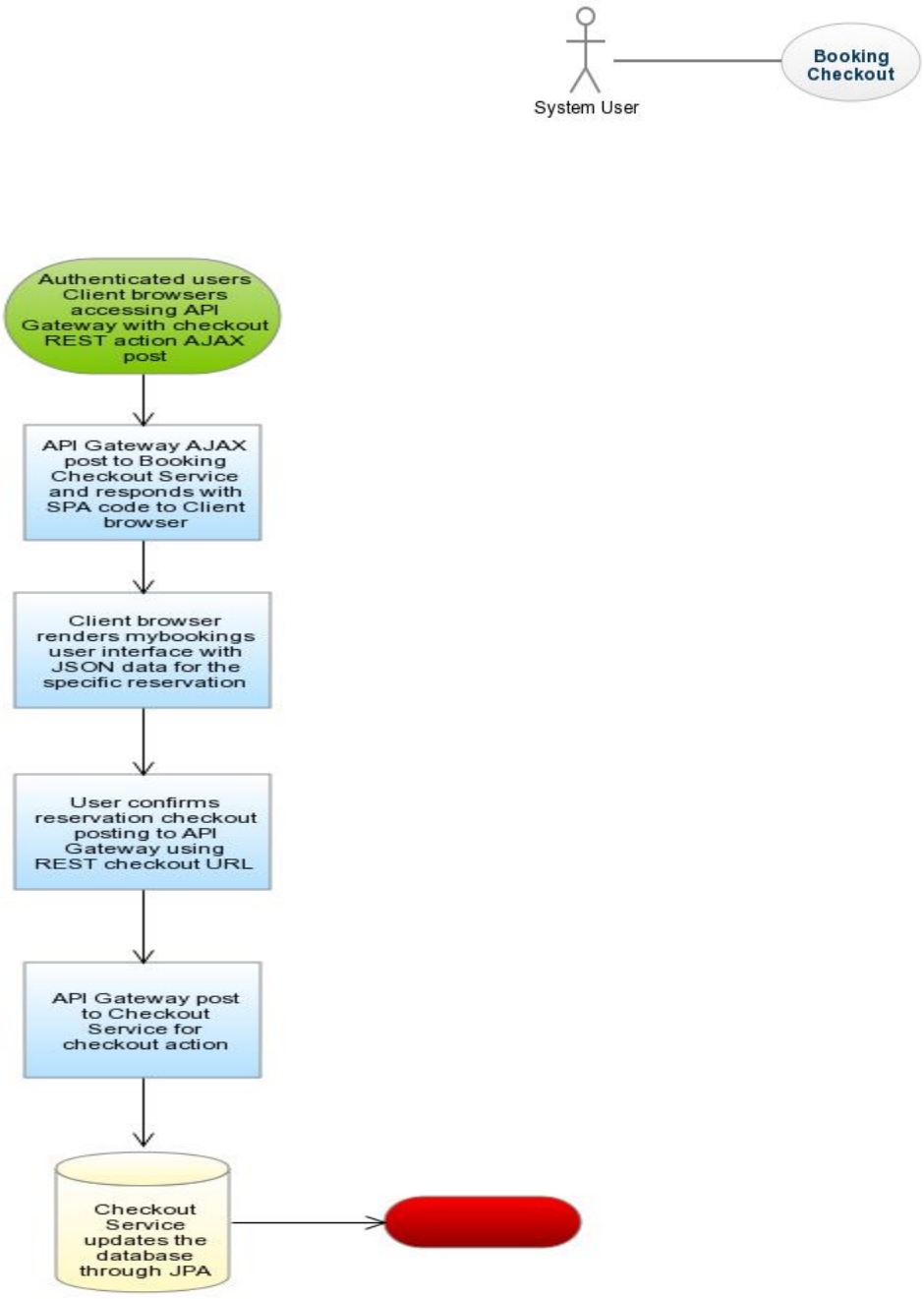

Figure 14 - Booking Checkout flow

4. UI Screens

User Interfaces were hand drawn for simplicity. In all user interfaces, the SYSTEM LOGO is a interactive image that takes the user back to the main page.

a. User Login UI

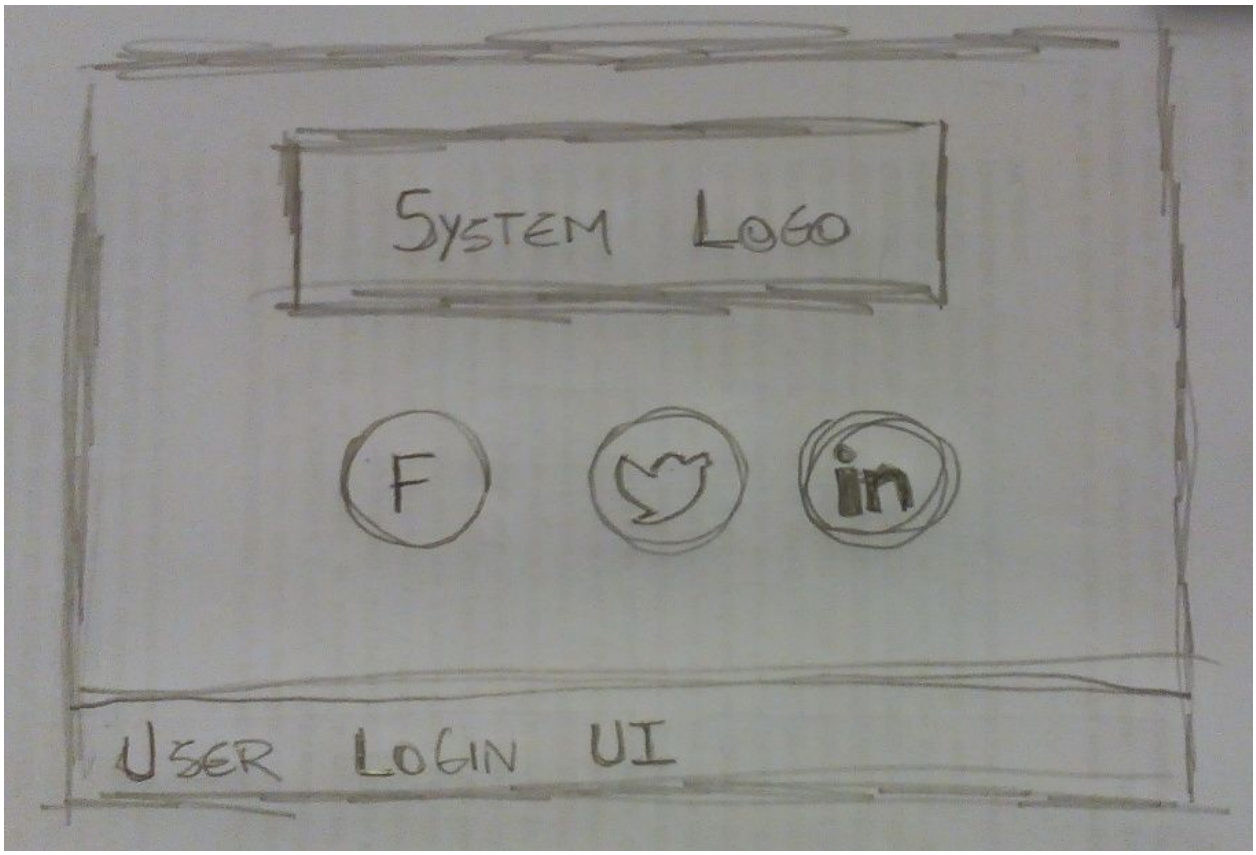

Figure 15 - User Login UI

b. Admin Staff Login UI

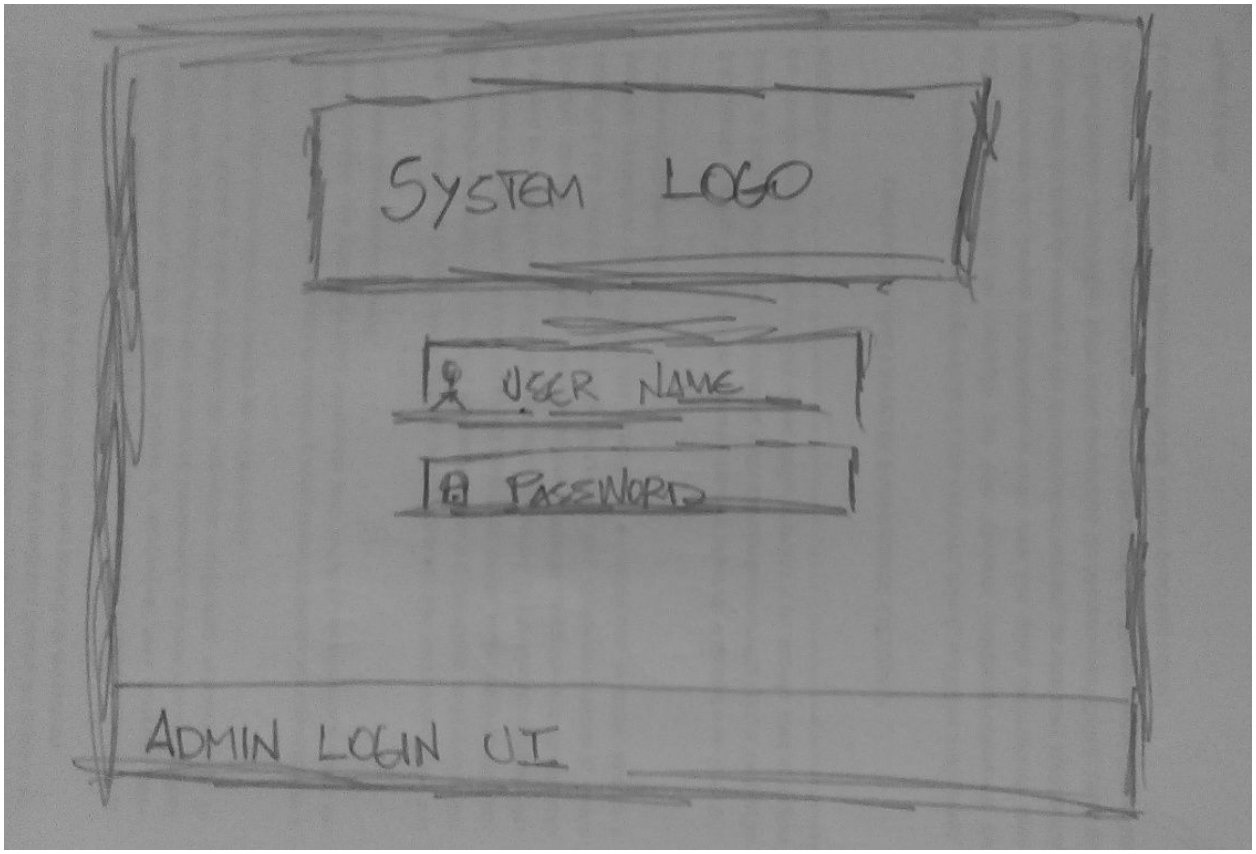

Figure 16 - Admin Staff Login UI

c. Main Page UI

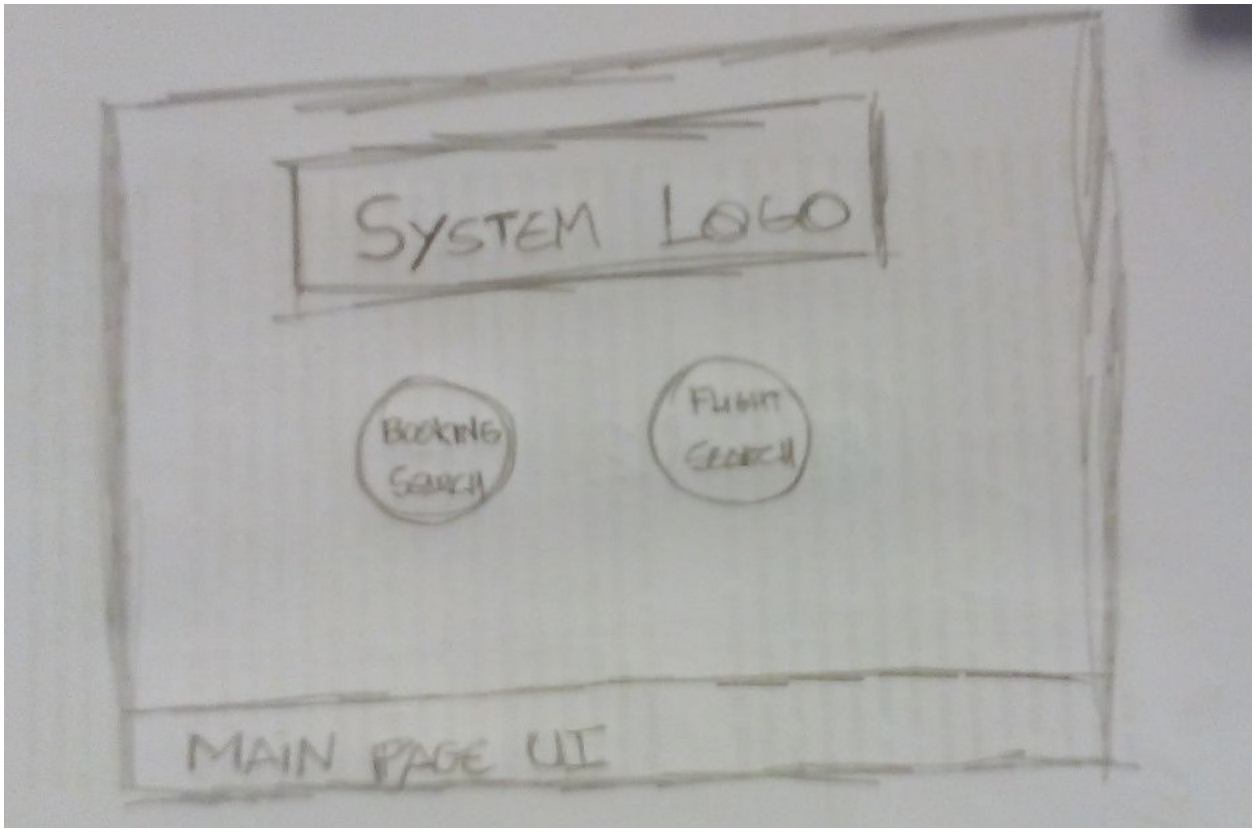

Figure 17 - Main Page UI

d. Data Searching UI

For either Booking or Flight searching a very simple user interface is suggested :

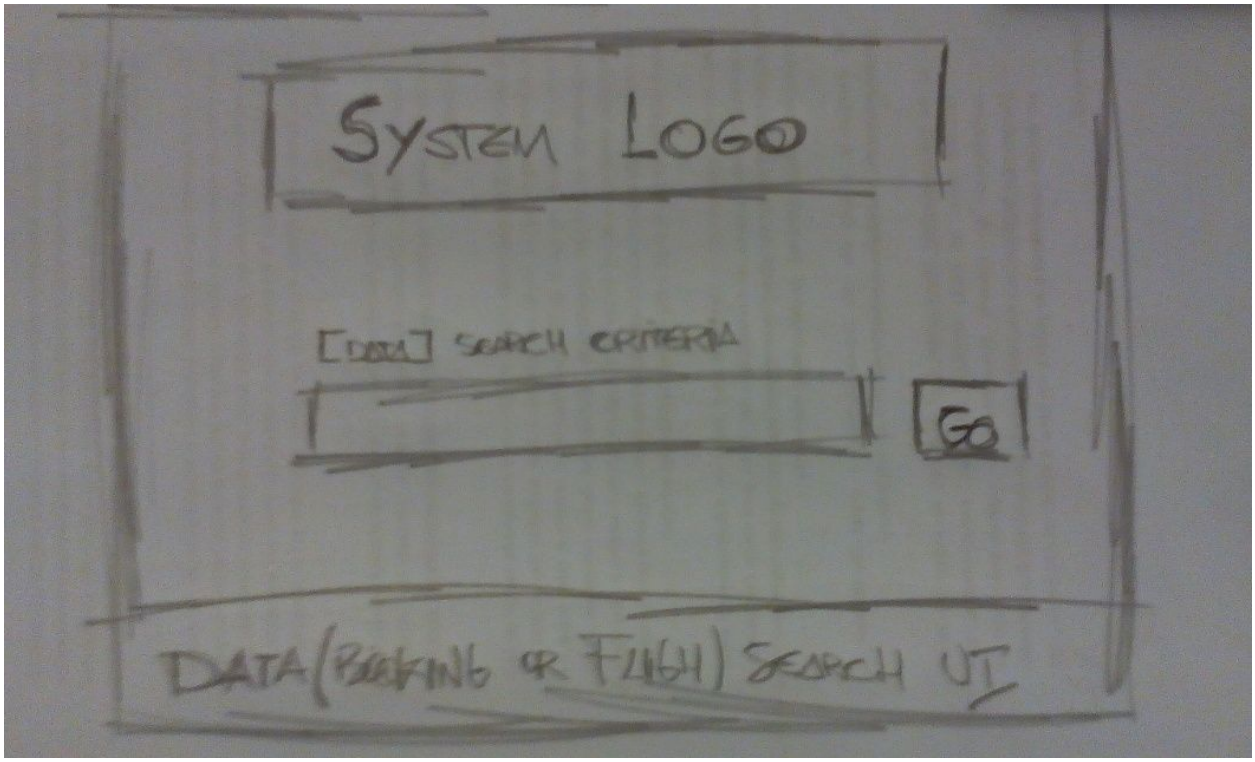

Figure 18 - Booking or Flight Search UI

e. Booking UI

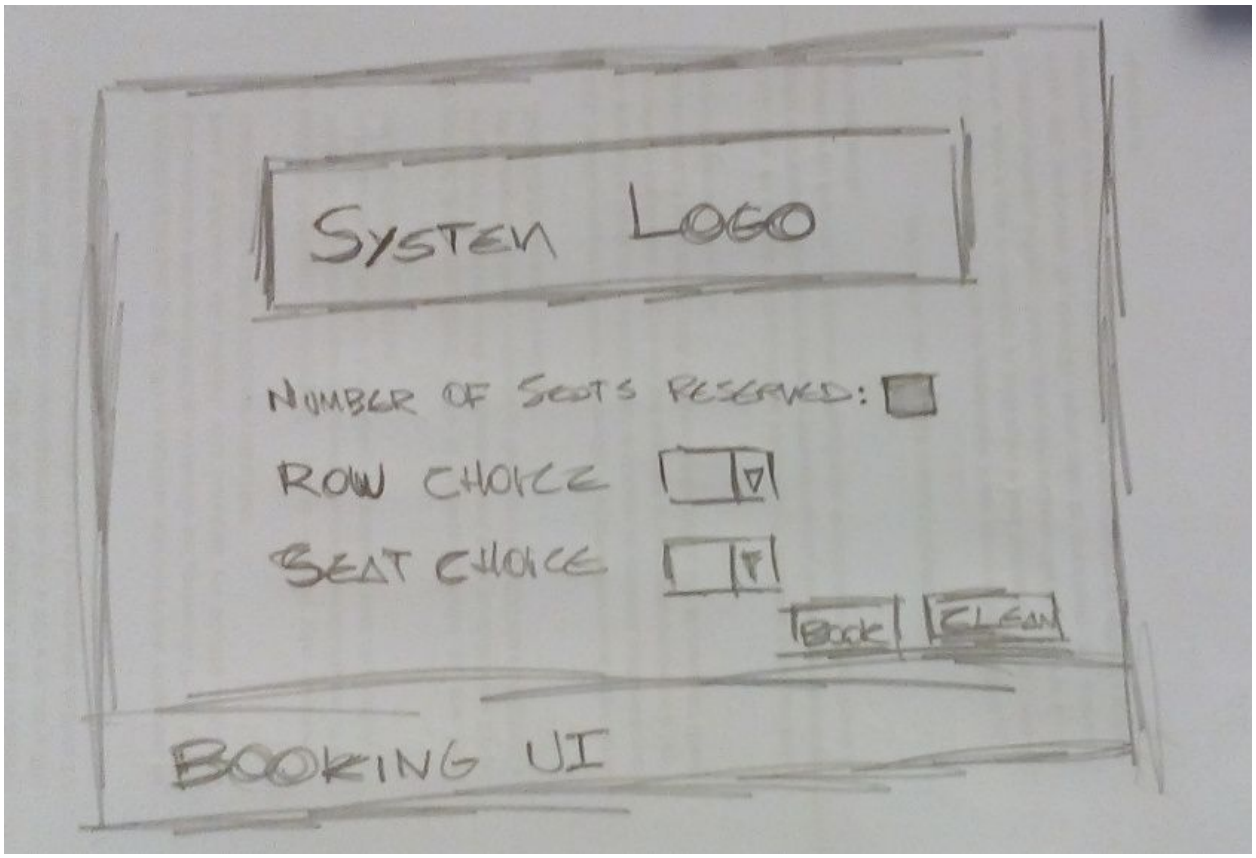

Figure 19 - Booking UI

f. Booking Listing and Managing UI

This is the mybookings user interface :

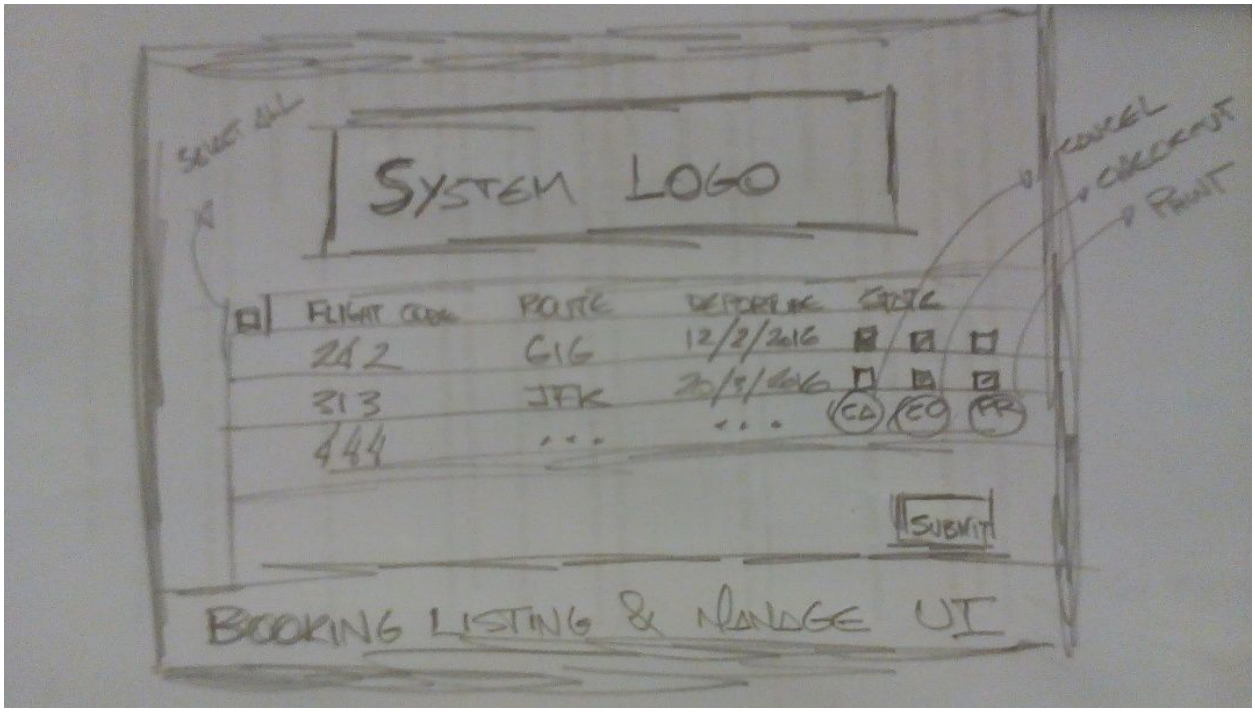

Figure 20 - Booking Listing and Manage UI

g. Flight Listing and Manage UI

This interface will also be used by common system users and staff admin. The later will have the Cancel Action button as well as the Select All :

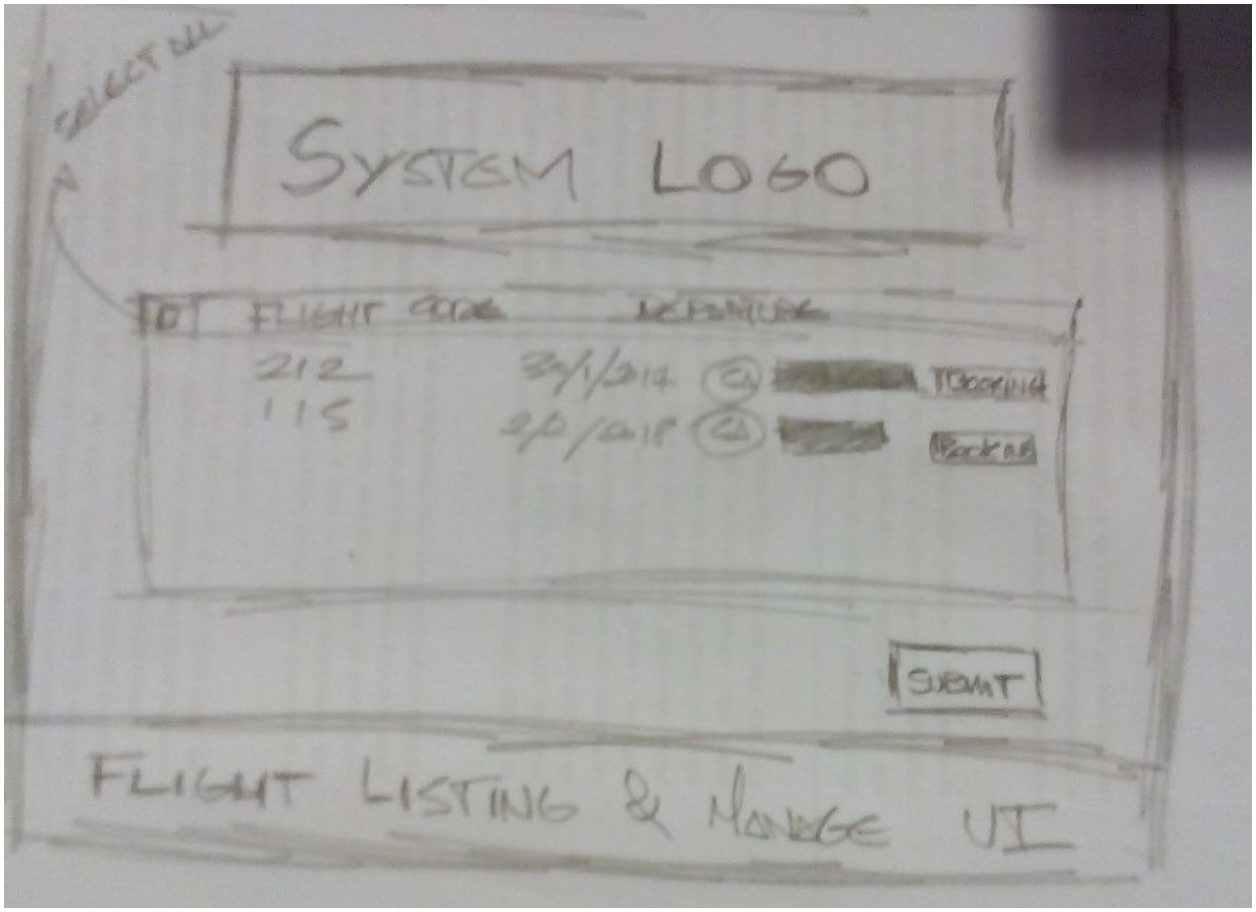

Figure 21 - Flight Listing and Manage UI

5. Conclusion

A more detailed and concise specification would require at least a week even for a simple system like this. This documentation aims to show the author capability to write proper documentation and control of up to date technologies to attend Crossover test demands.

For most of the infrastructure Amazon Web Services were the recommendation as it is the state of the art technology for cloud systems at the time of this writing and this decision attends REQ_OPER_SERVER_CLOUD_ARCH properly.