**PROJECT DOCUMENTATION**

**DISTRIBUTED FLIGHT RESERVATION SYSTEM**

**DELIVERABLE:**

A distributed system used by passengers and managers to make and manage flight reservations between different cities namely Montreal, Washington and New Delhi. Flights can be booked between these cities by passengers and the number of flights available between these cities is limited and can be controlled by the managers. There are different classes of flights such as economy class,  
premium class and the number of seats for each class from one city to other  
is fixed.

**PROJECT DESCRIPTION**:

a passenger books a flight, the passenger’s information is stored in a passenger  
record in the servers with following details: first name, last name, address, phone no.,  
destination, class of flight, date of flight. These records are placed in several lists that are  
stored in a hash map according to the first letter of the last name indicated in the records.

There are 2 types of user:

* Customer
* Manager

**Customer**:

* Can book a flight

Customers can book a fight ticket by providing the customer details and flight details such as flight date, flight origin, and flight destination. If the required flight/ flight seat is available, flight ticket will be booked for that customer and customer details will be saved in the database (hash map).

**Manager**:

* Edit flight details
* Get the count of number of seats booked
* Transfer the booked flight to another source

Managers can be identified by a unique *managerID*, which is constructed from the acronym of the city and a 4-digit number (e.g. MTL1111). Whenever a manager performs an operation, the system must identify the city that manager belongs to by looking at the *managerID* prefix and perform the operation on that server. A manager should also maintain a log file of the actions he/she performed on the system and the response from the system when available.

Manager can edit a field in the existing flight record and also get the number of seats booked from the flights from all 3 destinations

Technology Used:

* Web Service (Tomcat web server)
* UDP socket programming

**Web Service:**

* A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language--Java can talk with Perl; Windows applications can talk with Unix applications.
* Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.
* Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.
* A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

The basic web services platform is XML + HTTP. All the standard web services work using the following components

* SOAP (Simple Object Access Protocol)
* UDDI (Universal Description, Discovery and Integration)
* WSDL (Web Services Description Language)

**UDP Socket Programming**:

UDP is a simple transport-layer protocol. The application writes a message to a UDP socket, which is then encapsulated in a UDP datagram, which is further encapsulated in an IP datagram, which is sent to the destination.

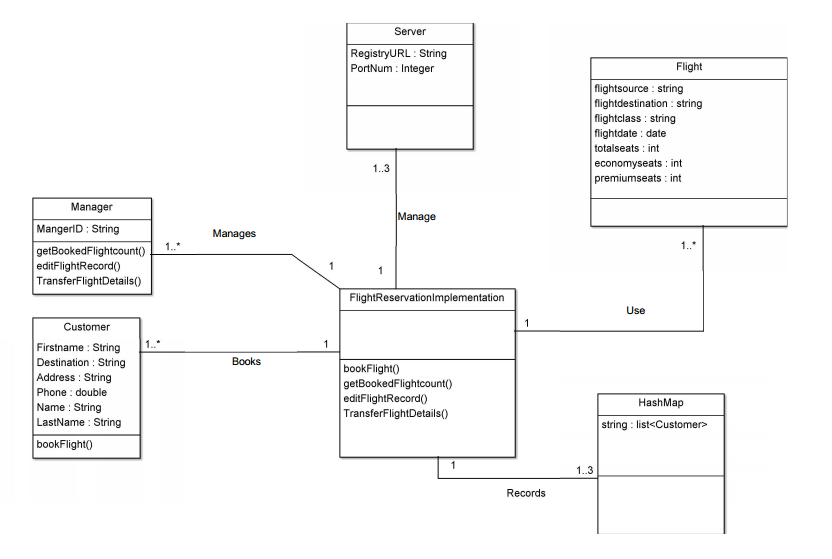
There is no guarantee that a UDP will reach the destination, that the order of the datagrams will be preserved across the network or that datagrams arrive only once.

The problem of UDP is its lack of reliability: if a datagram reaches its final destination but the checksum detects an error, or if the datagram is dropped in the network, it is not automatically retransmitted.

Each UDP datagram is characterized by a length. The length of a datagram is passed to the receiving application along with the data.

No connection is established between the client and the server and, for this reason, we say that UDP provides a connection-less service.

**UML DIAGRAM:**



**Test cases:**

**Login:**

The user can login as a customer or Manager. If the user selects customer, then he will be given with the option to book a flight. If he login as a manager he will be given with 2 options, either to edit a existing flight record or get the no of seats booked in flights across all cities.

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario** | **Expected output** | **Actual Output** | **Result** |
| User select “Customer” | Customer functionalities are provided. | Customer functionalities are provided. | Pass |
| User selects “Manager” | System asks for Manager identification | System asks for Manager identification | Pass |
| Manager provides the manager ID | Manager Id is authenticated. If it is success it directs the manager’s functionality page. | Manager Id is authenticated. If it is success it directs the manager’s functionality page. | Pass |

**Book Tickets**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario** | **Expected output** | **Actual Output** | **Result** |
| Provide user details | Given inputs are accepted without any pre requisite. | Given inputs are accepted without any pre requisite. | Pass |
| Provide flight details | Give the flight details as same as the available flight record. Flight ticket will be booked if the ticket is available in that class | Give the flight details as same as the available flight record. Flight ticket will be booked if the ticket is available in that class | Pass |
| Provide wrong flight details | Flight ticket wont be booked. Error will be shown | Flight ticket wont be booked. Error will be shown | Pass |
| Provide flight details and no ticket is available | System should tell that no more seats available in the flight | System should tell that no more seats available in the flight | Pass |

**Edit Flight Record:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario** | **Expected output** | **Actual Output** | **Result** |
| Enter as manager and select this option. Provide the flight no, field you want to edit and new value | System will get the flight record and update with new value for that record. | System will get the flight record and update with new value for that record. | Pass |
| Enter as manager and select this option. Provide the wrong flight no, field you want to edit and new value | System wont find a flight with this id. It will tell that no flight found. | System wont find a flight with this id. It will tell that no flight found. | Pass |
| Enter as manager and select this option. Provide the flight no, field that is not present in the flight record and new value | System will find the flight. But will tell that no such field is present in the flight details | System will find the flight. But will tell that no such field is present in the flight details | Pass |

**Get Booked seat count:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario** | **Expected output** | **Actual Output** | **Result** |
| Enter as manager and select this option. Based on the manager location it will call the other 2 running UDP servers and get the no of seats booked in the flights present in that city | No of seats booked in the flights of each cities will be printed | No of seats booked in the flights of each cities will be printed | Pass |

**Transfer booked flight record:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario** | **Expected output** | **Actual Output** | **Result** |
| Enter as manager and select this option. Get new source and date from manager. send it to the current server. UDP connection will be established to the new source server and ticket will be booked. simultaneously booked record from the current server will be removed | new ticket is booked in the new source server. Record from the old server will be remoed. | new ticket is booked in the new source server. Record from the old server will be remoed. | Pass |