System Architecture for Vacation Booking

Table of Contents

1. Executive summary

Using the Vacation Booking Demo Environment, you can demonstrate the capabilities of IBM® Rational® Integration Tester.

The Vacation Booking Demo Environment has been built with the intention to demonstrate how the IBM Rational Integration Tester can be used to test a real product. Vacation Booking is constructed around a set of technologies that are common in the setup IBM clients use in their production environment.

The Demo Environment is tailored around technologies like web services, REST APIs, JDBC database communication, Mainframe transactions, Mobile First applications and JMS/MQ transport.

Web services and REST are used to link the user interface with the backend web application server. In the example scenarios covered in this document, these technologies are used in the communication between Apache® Tomcat® and IBM WebSphere® Application Server. In the context of Rational Integration Tester, this helps to highlight features like recording different technology transports and building tests or stubs based on messages from a real system.

JDBC technology is used in Vacation Booking to link the front- and backend with database technologies. This demo environment is using IBM DB2® as a database, but the scenarios covered in this document are relevant for any particular database technology that provides a JDBC proxy for connectivity. In Rational Integration Tester this will enable the monitoring and recording of database communication and the creation of a virtual database as a stub.

Mainframe transaction is used in Vacation Booking to link the WAS application and CICS transaction server running on z/OS. In Rational Integration Tester this allows us to demonstrate how RIT supports the recording and virtualization of CICS® programs running on z/OS®.

Mobile First application is used in Vacation Booking to allow users to checkin for flights using their mobile device. In the context of Rational Test Workbench, this scenario helps us to create, edit, and perform tests for native, web, and hybrid applications on mobile devices. In this particular case we are using a hybrid application.

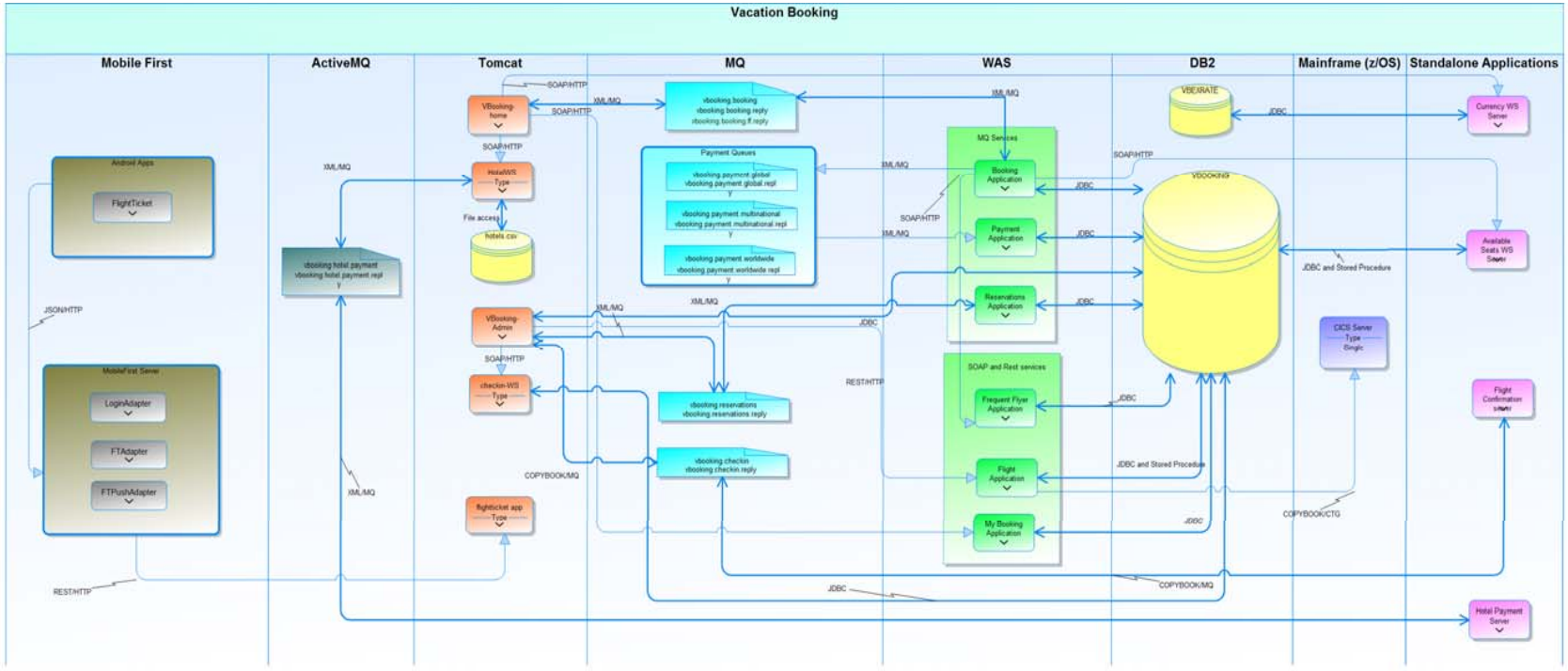
Lastly, JMS/MQ technology is used as the main communication transport backbone of the Vacation Booking system. The software used to implement the MQ technology is IBM WebSphere MQ® and it connects to the other service providers of the system – Tomcat and WAS. The MQ allows the entire system to exchange messages from one service provider to the next mostly in an asynchronous way (for example, Tomcat sends a message onto an MQ queue and then another relevant service from WAS picks up the message and does further processing). In Rational Integration Tester, the MQ transport is going to be recorded and monitored for changes and later it can be simulated using stubs.

1.1. Intended Audience

The System Architecture Plan is intended to be used by Client Technical Professionals, trainers and consultants as the primary audience. Secondly, because it contains information about a complex demonstration platform it can be used by the developers and testers of Rational Integration Tester.

Anyone interested in knowing how to use the Rational Integration Tester can use this document to help them set-up a demonstration environment that will highlight the key features of the software on an actual system with real data.

This document provides a high-level overview of a common system that would resemble to setups that clients have in their organisations. Thus, following the Vacation Booking will facilitate the understanding of concepts that Rational Integration Tester is based on.



2. System overview

The Vacation Booking Demo Environment is a purposely-built fictitious flight and hotel booking service. The user interacts with the service through a web page developed in java (jsp pages) which is served by Apache Tomcat. In the background there are different technologies that connect to the web interface to provide very advanced and complex functionality. The main technologies are IBM WebSphere Application Server, IBM MQ and IBM DB2. Every technology used in this demonstration environment is linked with specific features built inside the Vacation Booking system (for example, Booking a Flight or Passenger Check-in). Above you can find a system diagram of the entire configuration.

In order to make use of the technologies implemented inside the Vacation Booking system, this document proposes the use of scenarios. Each scenario consists of a normal process that a user would undertake in order to use a flight/hotel booking system. This is the list of scenarios that are supported by Vacation Booking:

- Use a currency web service that provides the system with exchange rates based on information found in a database.
- Book a flight, there are two different approaches:
    - In this scenario the user fills in a form and then there are web services that deal with the request, do the validation and pass the message to other parts of the system for processing the booking.
    - The second method is very similar to the first, but it adds a new component, called the Frequent Flyer Number in which the user is awarded loyalty points based on the destination he is flying to.
- Amend a flight booking based on a reference number, this talks with the WAS web service responsible for the bookings and alters the database after a booking has been made.
- Booking a hotel, very similar to the booking of flights, but this scenario explores the use of a web service hosted in Tomcat, ActiveMQ queues and uses the currency and hotel payment standalone applications.
- Checking-in a passenger is the scenario that an airline employee would go through at the check-in desk in which it confirms the validity of the booking and signs you in for your flight. This scenario is one of the most complex ones and uses most parts of the Vacation Booking system.
- Boarding a passenger onto the aircraft based on a valid passport which is processed as a stored procedure in the database or as a mainframe transaction.
- Immigration of the passenger once he has arrived at destination, again a service the airline would provide to the client.
- Flight maintenance, where the company would be able to alter flight information, for example, flight numbers.
- Mobile Flight Ticket, where users can checkin for flights using a hybrid mobile application.

3. System configuration

4. Use cases

For each feature built inside the Vacation Booking Demo Environment, there is a specific technological path in the backend that supports it. The use cases are useful when you want to demonstrate a particular technology to the client, for example how WebSphere Application Server can connect to a JDBC enabled database and how to record the database tables.

Below we will detail the available scenarios built inside the Demo Environment.

4.1. Book a Flight Scenario

In this scenario the user will use the main frontend interface to book a flight for a particular destination using one of the buttons on the right-hand side on the page (flights to Barcelona and Edinburgh). There are multiple components active in this scenario and you can follow them in the

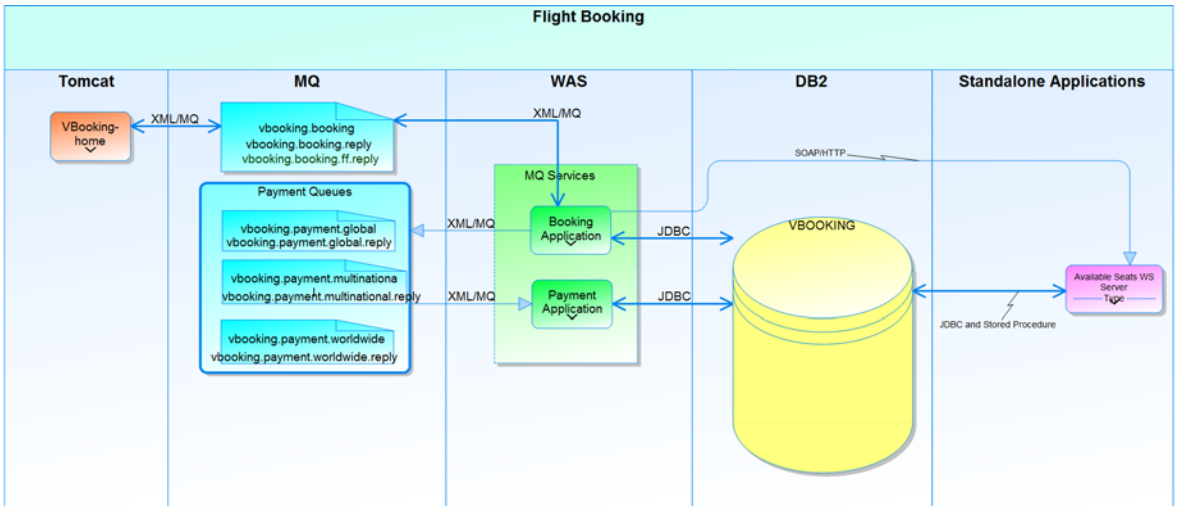


diagram below:

1		17:35:55....	MakeBooking	vbooking.booking (vbooking.booking.reply)
2		17:35:56....	ProcessMulti...	vbooking.payment.multinational (vbooking.payment.multinational.reply)
3		17:35:56....	VBooking XA...	SELECT MAX(RESERVATION_NUMBER) FROM RESERVATION
4		17:35:57....	ProcessMulti...	vbooking.payment.multinational.reply
5		17:35:58....	VBooking XA...	INSERT INTO RESERVATION (gender, last_name, middle_name, first_name, week_number, flight_number, reservat...
6		17:35:58....	MakeBooking	vbooking.booking.reply

The flight booking process uses components from every part of the Vacation Booking system and is one of the most complex scenarios.

Starting with Tomcat, the scenario makes use of the JSP web application vbooking-home that deals with the User Interface. (URL: http://localhost:8089/vbooking-home/index.jsp)

Following along, there are two web services hosted by WAS that deal with the flight booking process. The two components, Booking Application and Payment Application, both are MQ services applications that communicate directly with the MQ queues. In this scenario, the components communicate asynchronously. Both Tomcat and WAS applications publish messages on the MQ queues and then wait for a reply on the corresponding reply queue.

The six MQ queues (two for each operation) are used to pass messages from one part of the system to another and their use is detailed below. The DB2 database holds the information about the passengers, flights and destinations. Finally, the standalone service AvailableSeats is used to verify if there are seats available on this flight for the requested week.

When you are booking a flight, the following steps are happening in the backend:

1. When the user books a flight using one of the links on the page, he is presented with a form to complete. Once the user clicks Proceed, Vacation Booking main app uses XML/MQ to publish a new booking request on the vbooking.booking queue and waits for a reply on the vbooking.booking.reply queue.

2. The message published by the vbooking-home app is picked up by the WebSphere Application Server Booking Application. This service validates the input and calls the standalone service 'AvailableSeats'. This standalone service will access the database, calling a stored procedure. The stored procedure counts the number of people booked on this flight for the week requested, and compare it to the number of seats advertised on the plane. If the number of seats available is more than 1 then the booking proceeds else it fails.

3. Once the seats are verified then depending on the card used for payment when completed the form (there are three options: global, multinational or worldwide), the Booking service then publishes a message with the payment details to the appropriate MQ queue (vbooking.payment.*).

4. The WAS Payment Application then picks up the message published on the vbooking.payment.* queue and processes the payment information.

5. The Payment service then send a reply back on the appropriate queue vbooking.payment.*.reply. For example if a booking was made using a Global card type, then the payment information would have been posted on the vbooking.payment.global queue and the reply would be posted on vbooking.payment.global.reply.

6. The Booking service picks up the reply on the vbooking.payment.*.reply queue. If there are no errors in the payment processor, then the Booking service updates the database with the information using JDBC.

7. Once a database has been updated with the new booking, the Booking service publishes a booking confirmation on the vbooking.booking.reply queue.

8. Tomcat then picks up the message and displays the reservation number to the user.

4.1.1. Book a Flight Scenario (with the Frequent Flyer interface)

The second version of the flight booking process involves another WAS application to return frequent flyer points.

There are two ways to invoke this process

1) Edit the vbooking-home web.xml file in Tomcat so that the FFVersion parameter appears as below

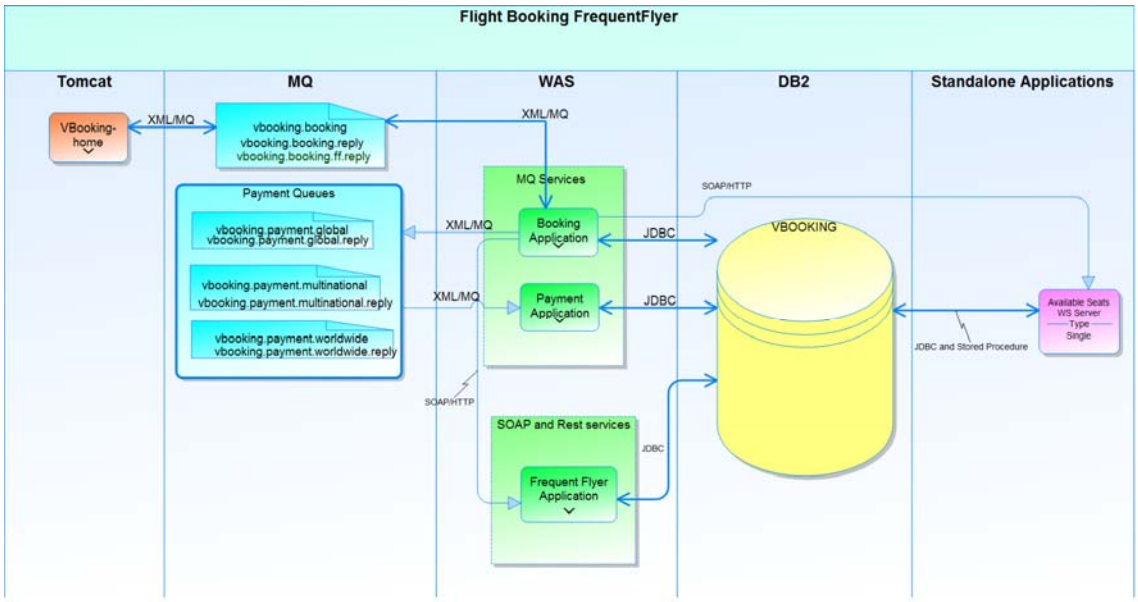
```
<context-param>
        <param-name>FFVersion</param-name>
        <param-value>YES</param-value>
</context-param>
```

   You will then need to restart Tomcat for this setting to take effect. Then when you use the url http://localhost:8089/vbooking-home the frequent flyer variant will be used.

2) You can also use the url http://localhost:8089/vbooking-home/indexFF.jsp to invoke this variant without making it permanent. This can be useful for testing and quick demos where you want to switch from one to the other.

The process and the background logic is the same as in the previous subchapter (Book a Flight Scenario). The few differences are that in this scenario the system is making use of the Frequent Flyer Application and you need to provide a frequent flyer number during the booking process. In addition to this the vbooking-home app waits for the booking confirmation reply on vbooking.booking.ff.reply queue. There are multiple components active in this scenario and you can follow them in the diagram below:

The Frequent Flyer application is a web service hosted on WAS that receives SOAP/XML format messages over the HTTP transport. It then checks the passenger start and destination airports in the database and awards travel points based on the result.



1		17:34:02....	MakeBooking	vbooking.booking (vbooking.booking.reply)
2		17:34:02....	VBooking XA...	SELECT DEP_AIRPORT, ARR_AIRPORT FROM FLIGHTS WHERE FLIGHT_NO =?
3		17:34:02....	frequentflyer	POST /frequentflyer
4		17:34:02....	VBooking XA...	SELECT AIR_MILES_AVAILABLE FROM FLIGHTMILES WHERE DEP_AIRPORT = ? AND DEST_AIRPORT = ?
5		17:34:02....	frequentflyer	200 - OK
6		17:34:03....	ProcessMulti...	vbooking.payment.multinational (vbooking.payment.multinational.reply)
7		17:34:03....	VBooking XA...	SELECT MAX(RESERVATION_NUMBER) FROM RESERVATION
8		17:34:04....	ProcessMulti...	vbooking.payment.multinational.reply
9		17:34:05....	VBooking XA...	INSERT INTO RESERVATION (gender, last_name, middle_name, first_name, week_number, flight_number, reservat...
10		17:34:05....	MakeBooking	vbooking.booking.reply

9. When the user books a flight using one of the links on the page, he is presented with a form to complete. Once the user clicks Proceed, Vacation Booking main app uses XML/MQ to publish a new booking request on the vbooking.booking queue and waits for a reply on the vbooking.booking.ff.reply queue.

10. The message published by the vbooking-home app is picked up by the WebSphere Application Server Booking Application. This service validates the input message and calls the standalone service 'AvailableSeats'. This standalone service will access the database, calling a stored procedure. The stored procedure counts the number of people booked on this flight for the week requested, and compare it to the number of seats advertised on the plane. If the number of seats available is more than 1 then the booking proceeds else it fails.

11. Once the seats are verified then if a Frequent Flyer number has been provided, it sends a request to the Frequent Flyer application hosted in WAS over SOAP/HTTP.

12. The Frequent Flyer app checks that the destination provided is valid and looks up the reward in the VBOOKING database based on the result.

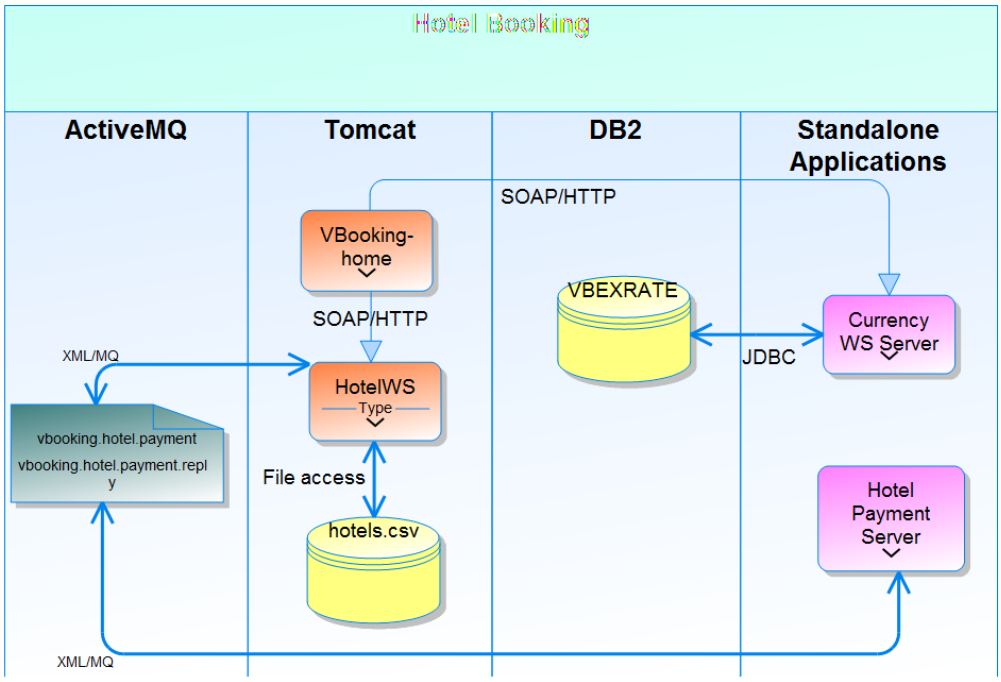
13. After the Frequent Flyer returns its result, the Booking service finishes the validation. Depending on the card used for payment when completed the form (there are three options:

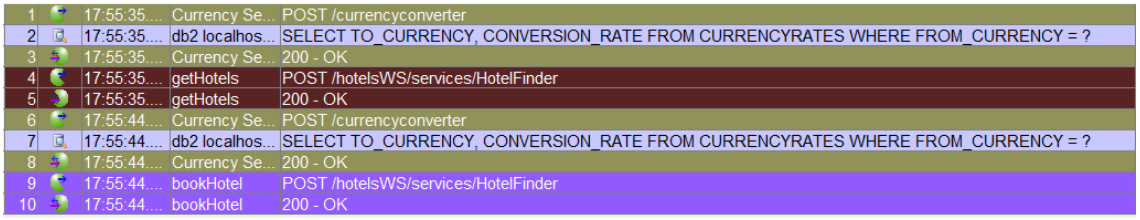
global, multinational or worldwide), the Booking service then publishes a message with the payment details to the appropriate MQ queue (vbooking.payment.*).

14. The WAS Payment Application then picks up the message published on the vbooking.payment.* queue and processes the payment information.
15. The Payment service then send a reply back on the appropriate queue vbooking.payment.*.reply. For example if a booking was made using a Global card type, then the payment information would have been posted on the vbooking.payment.global queue and the reply would be posted on vbooking.payment.global.reply.
16. The Booking service picks up the reply on the vbooking.payment.*.reply queue. If there are no errors in the payment processor, then the Booking service updates the database with the information using JDBC communication.
17. Once a database has been updated with the new booking, the Booking service publishes a booking confirmation on the vbooking.booking.ff.reply queue.
18. Tomcat then picks up the message and displays the reservation number together with the number of points to be earnt to the user.

4.2. Book a Hotel Scenario

In this scenario the user will use the main frontend interface to search and book a hotel. There are multiple components active in this scenario and you can follow them in the diagram below:





The hotel booking process uses various different components from the Vacation Booking system.

Starting with Tomcat, the scenario makes use of the JSP web application vbooking-home that deals with the User Interface. (URL: http://localhost:8089/vbooking-home/index.jsp). Also, Tomcat is the web container that hosts the HotelFinder Service that deals with the booking request.

Following along, the HotelFinder Service uses a flat CSV file to provide the user with the available hotels for his destination of choice.

The Currency Standalone application is a standalone service that receives XML format messages over HTTP transport. It then checks the exchange rates in the database and returns the conversion rate to the HotelFinder service.

The Hotel Payment Server validates the payment details of the customer which are exchanged using XML type messages through ActiveMQ queues.

The final component used in the hotel booking process is the Vacation Booking Exchange Database (VBEXRATE Database on the diagram).

When you are booking a hotel, the following steps are happening in the backend:

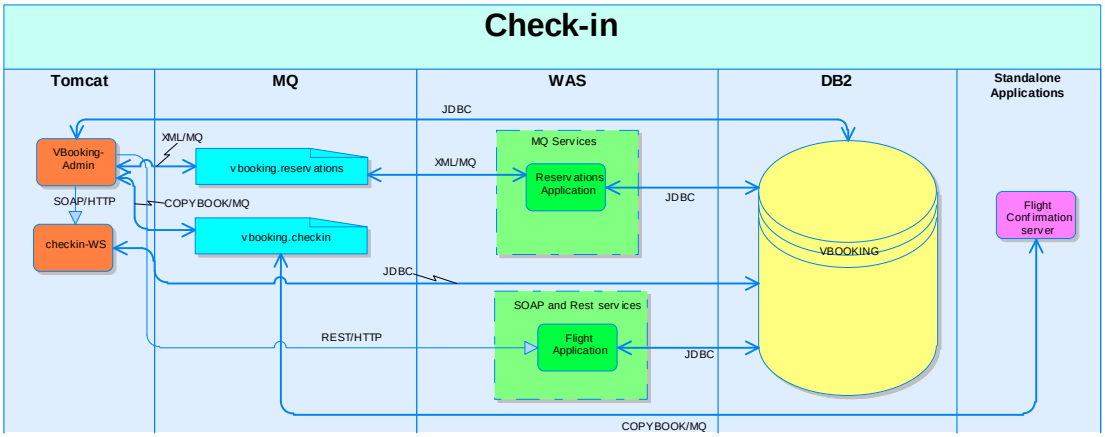
When the user wants to book a hotel, the User Interface allows two separate methods. The first method is to select "Hotel only" from the left-hand side of the interface and search for a location, for example, Barcelona. The second method is to book your hotel as a bundle, including the flight as part of the booking. If you are interested in the second method, please see the Note box below, this subchapter will detail the first method.

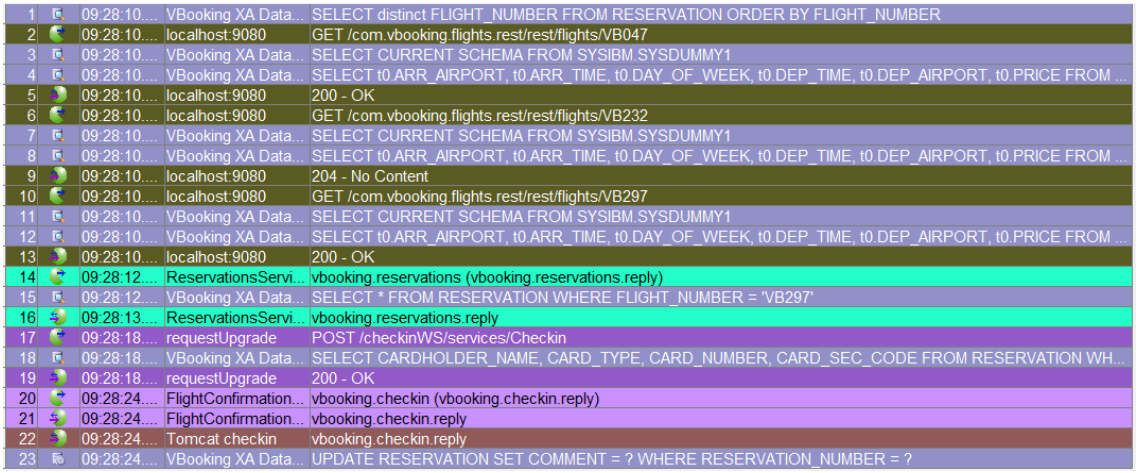
1. The user selects "Hotel only" and provides the city name, the reservation period and clicks Search. This will send a request to the HotelFinder Service in SOAP format using the HTTP transport.
2. The hotel finder service will read the CSV file containing the hotels name in the location provided by the user.
3. The service will also send a SOAP message to the Currency Standalone App to ask for the exchange rate selected by the user.
4. The currency service will interrogate the VBEXRATE Database using the JDBC connection and return the exchange rate to the hotel finder service.
5. The hotel finder service will compile a list of hotels available for the searched location and the correct currency and present the result to the user.
6. The user will select a hotel and provide the information for the booking and click Proceed. This will again send a request to HotelFinder Service with all the details entered by the user.

7. This time the hotel finder service will post a XML message on the ActiveMQ vbooking.hotel.payment queue and will wait for a reply on the reply queue.
8. The standalone hotel payment service picks up the published messaged and validates the payment details of the customer and post a reply back on the reply queue (for now the payment validation always succeeds).
9. The confirmation message on the reply queue is read by the hotel finder service and is forwarded to the vbooking-home app. The User Interface is updated with the confirmation for the booking, but this is not stored in any database, so it is just a temporary plain text.

4.3. Passenger Check-in Scenario

In this scenario the user will have to use the Administration interface to check a passenger in for a particular flight that was already booked. There are multiple components active in this scenario and you can follow them in the diagram below:





1		09:28:10....	VBooking XA Data...	SELECT distinct FLIGHT_NUMBER FROM RESERVATION ORDER BY FLIGHT_NUMBER
2		09:28:10....	localhost:9080	GET /com.vbooking.flights.rest/rest/flights/VB047
3		09:28:10....	VBooking XA Data...	SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1
4		09:28:10....	VBooking XA Data...	SELECT t0.ARR_AIRPORT, t0.ARR_TIME, t0.DAY_OF_WEEK, t0.DEP_TIME, t0.DEP_AIRPORT, t0.PRICE FROM ...
5		09:28:10....	localhost:9080	200 - OK
6		09:28:10....	localhost:9080	GET /com.vbooking.flights.rest/rest/flights/VB232
7		09:28:10....	VBooking XA Data...	SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1
8		09:28:10....	VBooking XA Data...	SELECT t0.ARR_AIRPORT, t0.ARR_TIME, t0.DAY_OF_WEEK, t0.DEP_TIME, t0.DEP_AIRPORT, t0.PRICE FROM ...
9		09:28:10....	localhost:9080	204 - No Content
10		09:28:10....	localhost:9080	GET /com.vbooking.flights.rest/rest/flights/VB297
11		09:28:10....	VBooking XA Data...	SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1
12		09:28:10....	VBooking XA Data...	SELECT t0.ARR_AIRPORT, t0.ARR_TIME, t0.DAY_OF_WEEK, t0.DEP_TIME, t0.DEP_AIRPORT, t0.PRICE FROM ...
13		09:28:10....	localhost:9080	200 - OK
14		09:28:12....	ReservationsServi...	vbooking.reservations (vbooking.reservations.reply)
15		09:28:12....	VBooking XA Data...	SELECT * FROM RESERVATION WHERE FLIGHT_NUMBER = 'VB297'
16		09:28:13....	ReservationsServi...	vbooking.reservations.reply
17		09:28:18....	requestUpgrade	POST /checkinWS/services/Checkin
18		09:28:18....	VBooking XA Data...	SELECT CARDHOLDER_NAME, CARD_TYPE, CARD_NUMBER, CARD_SEC_CODE FROM RESERVATION WH...
19		09:28:18....	requestUpgrade	200 - OK
20		09:28:24....	FlightConfirmation...	vbooking.checkin (vbooking.checkin.reply)
21		09:28:24....	FlightConfirmation...	vbooking.checkin.reply
22		09:28:24....	Tomcat checkin	vbooking.checkin.reply
23		09:28:24....	VBooking XA Data...	UPDATE RESERVATION SET COMMENT = ? WHERE RESERVATION_NUMBER = ?

The check-in process uses components from every part of the Vacation Booking system and is one of the most complex scenarios.

Starting with Tomcat, the scenario makes use of the JSP web application vbooking-admin that deals with the User Interface. (URL: http://localhost:8089/vbooking-admin/index.jsp) Also, Tomcat provides a check-in web service that the admin interface uses through XML messages on the HTTP transport.

Following along, there are two web services hosted by WAS that deal with the check-in process. The first one on the diagram, the flight service is a REST application that communicates with the admin app to provide it with information on the flight details, like the destination of the airplane. The second web service from WAS is the MQ service Reservation service that communicates with the MQ service to perform the check-in procedure.

The Flight Confirmation Server is a standalone MQ service that receives Copybook format messages and return an altered message to confirm the check-in.

The four MQ queues (two for each operation) are used to pass messages from one part of the system to another and their use is detailed below. Finally the DB2 database holds the information about the passenger and flight.

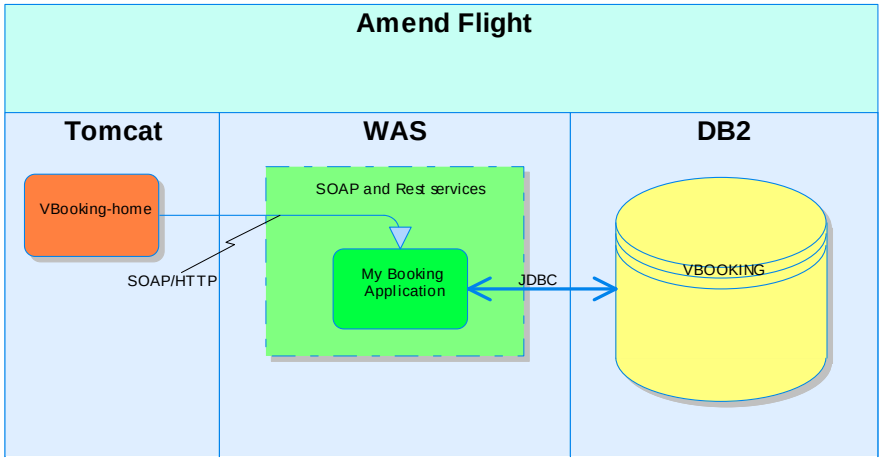
When you are doing the check-in process, the following steps are happening in the backend:

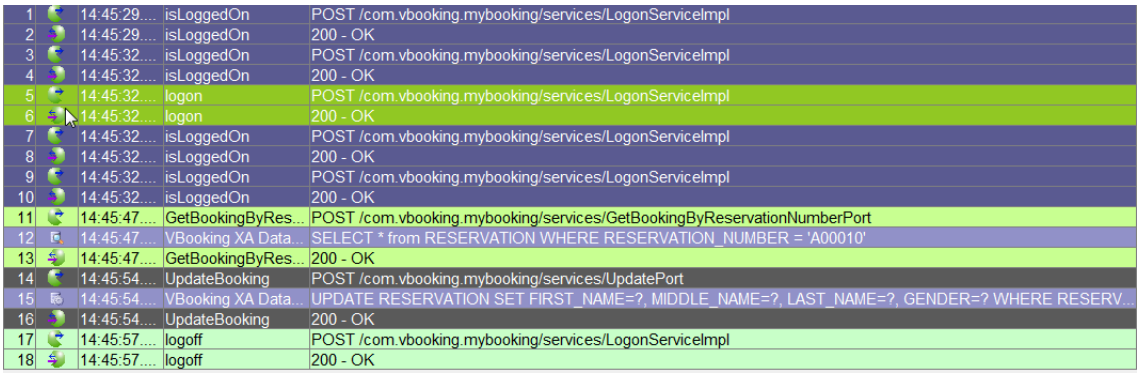
1. Vacation Booking Admin app uses JDBC to get a list of flights with passengers.
2. The Admin app communicates through REST over HTTP with the flights application hosted by WAS. The Flights service provides the flight destination for a single flight with each request; the admin app may make multiple requests.
3. The flight service uses JDBC to access the database for each flight in order to retrieve the destination.
4. The admin app puts an MQ request on the vbooking.reservations queue and waits for the reply - this is to get a list of passengers on that flight.
5. The Reservations web service stored also in WAS is listening to the vbooking.reservations queue and receives the message.

6. The reservations service gets a passenger list via JDBC from the database.
7. Then the reservation app puts the list of passengers on the MQ vbooking.reservations.reply queue.
8. Vacation Booking admin app receives the reply from the MQ vbooking.reservations.reply queue and displays the passenger list to the user.
9. The user selects a passenger and then clicks Request Upgrade, Vacation Booking admin sends a soap message to Check-in service.
10. The Check-in service returns the upgrade string.
11. The user clicks Check-in and Vacation Booking sends an MQ message to vbooking.checkin queue and waits for the reply on vbooking.checkin.reply queue.
12. The Flight Confirmation Server receives the message, modifies it and posts the result to the MQ queue vbooking.checkin.reply.
13. Vacation Booking uses a separate thread to read the result from the vbooking.checkin.reply queue. It then uses JDBC to update the database with the check-in information. Lastly, the User Interface is updated with the confirmation for the check-in.

4.4. Amend Flight Scenario

In this scenario the user will have to use the main home interface to change the details for a particular flight that was already booked. There are multiple components active in this scenario and you can follow them in the diagram below:





1	14:45:29....	isLoggedOn	POST /com.vbooking.mybooking/services/LogonServiceImpl
2	14:45:29....	isLoggedOn	200 - OK
3	14:45:32....	isLoggedOn	POST /com.vbooking.mybooking/services/LogonServiceImpl
4	14:45:32....	isLoggedOn	200 - OK
5	14:45:32....	logon	POST /com.vbooking.mybooking/services/LogonServiceImpl
6	14:45:32....	logon	200 - OK
7	14:45:32....	isLoggedOn	POST /com.vbooking.mybooking/services/LogonServiceImpl
8	14:45:32....	isLoggedOn	200 - OK
9	14:45:32....	isLoggedOn	POST /com.vbooking.mybooking/services/LogonServiceImpl
10	14:45:32....	isLoggedOn	200 - OK
11	14:45:47....	GetBookingByRes...	POST /com.vbooking.mybooking/services/GetBookingByReservationNumberPort
12	14:45:47....	VBooking XA Data...	SELECT * from RESERVATION WHERE RESERVATION_NUMBER = 'A00010'
13	14:45:47....	GetBookingByRes...	200 - OK
14	14:45:54....	UpdateBooking	POST /com.vbooking.mybooking/services/UpdatePort
15	14:45:54....	VBooking XA Data...	UPDATE RESERVATION SET FIRST_NAME=?, MIDDLE_NAME=?, LAST_NAME=?, GENDER=? WHERE RESERV...
16	14:45:54....	UpdateBooking	200 - OK
17	14:45:57....	logoff	POST /com.vbooking.mybooking/services/LogonServiceImpl
18	14:45:57....	logoff	200 - OK

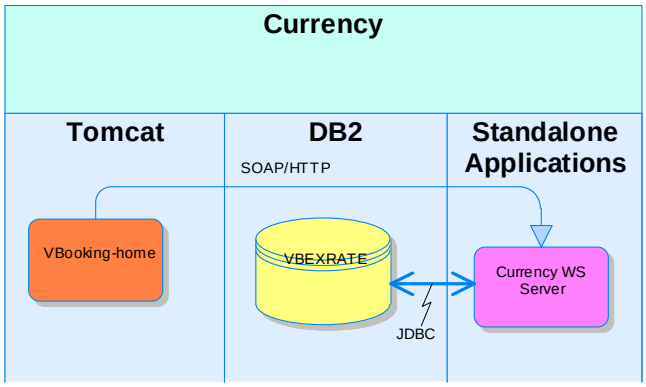
The flight amending process is straightforward and includes only a few components from the Vacation Booking Demonstration Environment. The first component used in this scenario is the Tomcat JSP application vbooking-home. This interface is the main point of contact with the user. Secondly, the request is passed to the My Booking WAS Application for processing and once the details are validated by the service, finally, the VBOOKING Database is updated with the new information provided by the user.

When you are amending a flight, the following steps are happening in the backend:

1. In the main interface (URL: http://localhost:8089/vbooking-home/index.jsp) of Vacation Booking home app, the user will click on “Amend my flight booking”. After the user signs in (no username or password), a booking reference must be provided. For example, use A00005 as a booking reference. This will send a request to the My Booking Application hosted in WAS. The message format is SOAP over HTTP transport.
2. The My Booking web service will get the request from vbooking-home and interrogate the database for the matching booking reference, and return the result to the User Interface.
3. The user can then make the changes as needed, and submit them back to the My Booking web service.
4. The web service will validate the input and update the database using JDBC communication.

4.5. Currency application scenario

This scenario details the capabilities built into the Currency Standalone application. There are multiple components active in this scenario and you can follow them in the diagram below:



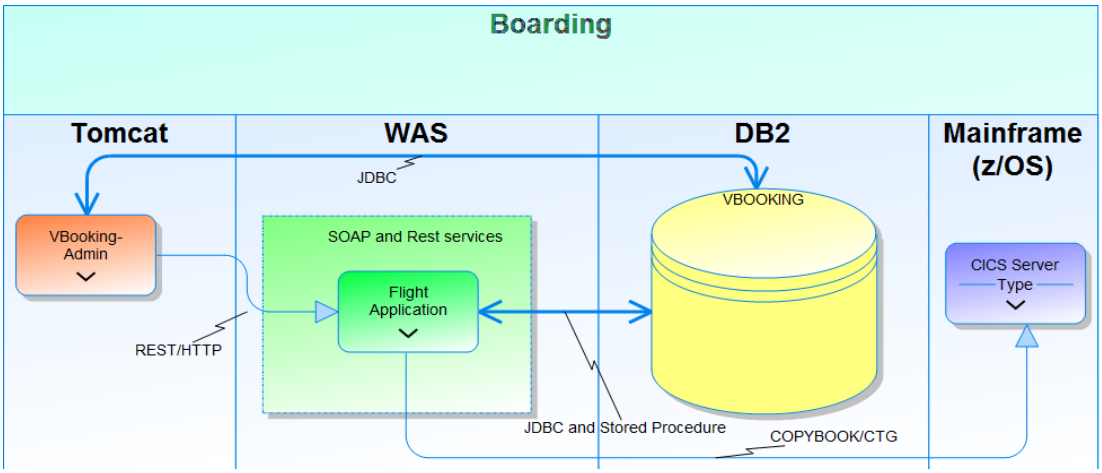
The Currency Standalone application is used during the booking process for both the hotel and the flight. The service receives a SOAP message format from the Tomcat User Interface vbooking-home. That then triggers the web service to access the VBEXRATE Database and return the exchange rate.

When you are making a booking, the following steps are happening in the backend:

1. The Tomcat JSP interface vbooking-home will also send a SOAP message to the Currency Standalone App to ask for the exchange rate selected by the user.
2. The currency service will interrogate the VBEXRATE Database using the JDBC connection and return the exchange rate to the main interface.
3. The vbooking-home interface will update the price according to the user currency preferences.

4.6. Boarding scenario

In this scenario the user will act as a flight operator and will be able to board a passenger onto the aircraft for departure. There are multiple components active in this scenario and you can follow them in the diagram below:



1		09:23:03....	VBooking XA...	SELECT distinct FLIGHT_NUMBER FROM RESERVATION ORDER BY FLIGHT_NUMBER
2		09:23:03....	FlightsRestS...	GET /com.vbooking.flights.rest/rest/flights/VB047
3		09:23:03....	db2 9.81.39....	SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1
4		09:23:03....	db2 9.81.39....	SELECT t0.ARR_AIRPORT, t0.ARR_TIME, t0.DAY_OF_WEEK, t0.DEP_TIME, t0.DEP_AIRPORT, t0.PRICE FROM FLIGHTS t0 WHER...
5		09:23:03....	FlightsRestS...	200 - OK
6		09:23:03....	FlightsRestS...	GET /com.vbooking.flights.rest/rest/flights/VB232
7		09:23:03....	db2 9.81.39....	SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1
8		09:23:03....	db2 9.81.39....	SELECT t0.ARR_AIRPORT, t0.ARR_TIME, t0.DAY_OF_WEEK, t0.DEP_TIME, t0.DEP_AIRPORT, t0.PRICE FROM FLIGHTS t0 WHER...
9		09:23:03....	FlightsRestS...	204 - No Content
10		09:23:03....	FlightsRestS...	GET /com.vbooking.flights.rest/rest/flights/VB297
11		09:23:03....	db2 9.81.39....	SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1
12		09:23:04....	db2 9.81.39....	SELECT t0.ARR_AIRPORT, t0.ARR_TIME, t0.DAY_OF_WEEK, t0.DEP_TIME, t0.DEP_AIRPORT, t0.PRICE FROM FLIGHTS t0 WHER...
13		09:23:04....	FlightsRestS...	200 - OK
14		09:23:04....	FlightsRestS...	GET /com.vbooking.flights.rest/rest/flights/withReservations
15		09:23:04....	db2 9.81.39....	SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1
16		09:23:04....	db2 9.81.39....	SELECT F.FLIGHT_NO, F.DEP_AIRPORT, F.DEP_TIME, F.ARR_AIRPORT, F.ARR_TIME, F.DAY_OF_WEEK, F.PRICE FROM FLIGHT...
17		09:23:04....	FlightsRestS...	200 - OK
18		09:23:06....	FlightsRestS...	GET /com.vbooking.flights.rest/rest/flights/VB297/reservations
19		09:23:06....	db2 9.81.39....	SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1
20		09:23:06....	db2 9.81.39....	SELECT R.FIRST_NAME, R.LAST_NAME, R.WEEK_NUMBER, R.FLIGHT_NUMBER, R.RESERVATION_NUMBER, R.COMMENT, R.PA...
21		09:23:06....	FlightsRestS...	200 - OK
22		09:23:11....	Reservation...	GET /com.vbooking.flights.rest/rest/reservations/A00010
23		09:23:11....	db2 9.81.39....	SELECT R.FIRST_NAME, R.LAST_NAME, R.WEEK_NUMBER, R.FLIGHT_NUMBER, R.RESERVATION_NUMBER, R.COMMENT, R.PA...
24		09:23:11....	Reservation...	200 - OK
25		09:23:19....	Reservation...	GET /com.vbooking.flights.rest/rest/reservations/A00010
26		09:23:19....	db2 9.81.39....	SELECT R.FIRST_NAME, R.LAST_NAME, R.WEEK_NUMBER, R.FLIGHT_NUMBER, R.RESERVATION_NUMBER, R.COMMENT, R.PA...
27		09:23:19....	Reservation...	200 - OK
28		09:23:19....	db2 9.81.39....	SELECT R.FIRST_NAME, R.LAST_NAME, R.WEEK_NUMBER, R.FLIGHT_NUMBER, R.RESERVATION_NUMBER, R.COMMENT, R.PA...
29		09:23:20....	db2 9.81.39....	{call doFlyCheck(?,?,?,?,?,?,?,?)}
30		09:23:20....	db2 9.81.39....	SELECT R.FIRST_NAME, R.LAST_NAME, R.WEEK_NUMBER, R.FLIGHT_NUMBER, R.RESERVATION_NUMBER, R.COMMENT, R.PA...
31		09:23:20....	db2 9.81.39....	UPDATE RESERVATION SET PASSPORT_ID=?, BOARDED=? WHERE RESERVATION_NUMBER = ? AND LOWER('m')=LOWER...
32		09:23:20....	FlightsRestS...	GET /com.vbooking.flights.rest/rest/flights/VB297/reservations
33		09:23:20....	db2 9.81.39....	SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1
34		09:23:20....	db2 9.81.39....	SELECT R.FIRST_NAME, R.LAST_NAME, R.WEEK_NUMBER, R.FLIGHT_NUMBER, R.RESERVATION_NUMBER, R.COMMENT, R.PA...
35		09:23:20....	FlightsRestS...	200 - OK

The boarding process is accessed through the vbooking-admin administration interface. (URL: http://localhsot:8089/vbooking-admin/index.jsp) This interface is provided by the Tomcat web server. Following along, in the background the vbooking-admin service communicates with the Flight Application WAS service which is a RESTful service. The last component in this process is the VBOOKING Database which stores all the information about passengers, flights and destinations.

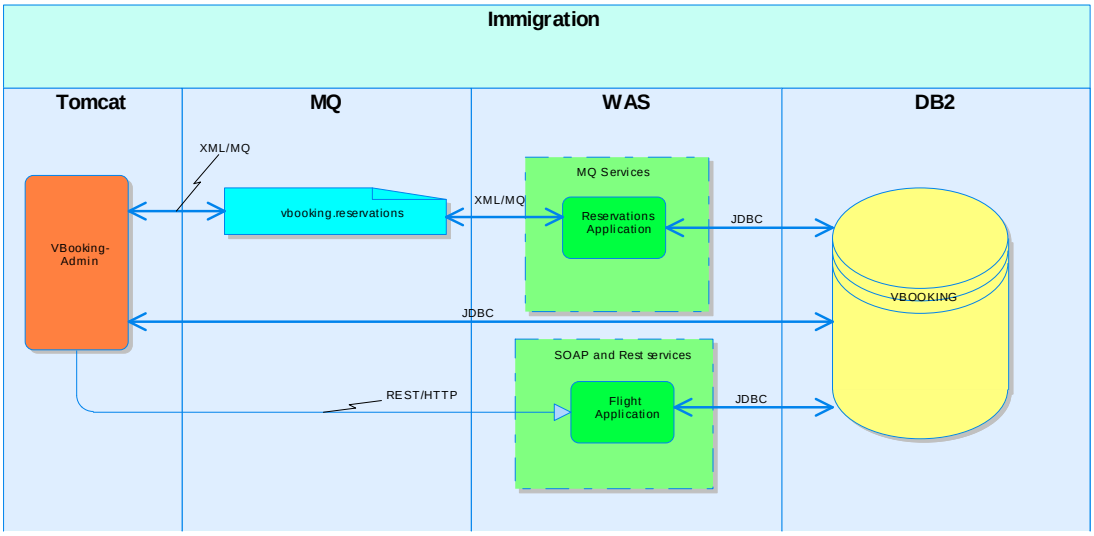
When you are boarding a passenger, the following steps are happening in the backend:

1. The Tomcat JSP interface vbooking-admin will access the database through JDBC communication to get a list of flights.
2. The vbooking-admin web app will also send a REST request to the Flight Application REST service hosted in WAS to access a specific flight.

3. The user will provide a passport number on the boarding view of the User Interface.
4. Now depending on the method selected for flyCheck there two different approaches:
    a. If the flyCheck method selected is SPROC then the database stored procedure is used to validate the passport number (contains 7 or more characters)
    b. If the flyCheck method selected is CICS then the CICS transaction is run using Copybook format to validate the passport number (contains 1 or more characters)
5. To configure which method to use to perform fly check a System Property needs to be defined in WAS – *flightrest.flyCheckMethod.* Its value can be set to either *CICS* (to use CICS interface) or *SPROC* (to use database stored procedure). If this property is not defined then the default (i.e. *SPROC*) will be used.
6. Finally, if the Passport number is valid, the boarding information is stored inside the database and the user interface is updated.

4.7. Immigration scenario

In this scenario the user will act as an airport officer and will be able to do the immigration for all the boarded passengers in the said aircraft. There are multiple components active in this scenario and you can follow them in the diagram below:



1		12:58:13....	VBooking XA...	SELECT distinct FLIGHT_NUMBER FROM RESERVATION ORDER BY FLIGHT_NUMBER
2		12:58:13....	FlightsRestS...	GET /com.vbooking.flights.rest/rest/flights/VB047
3		12:58:13....	FlightsRestS...	200 - OK
4		12:58:13....	FlightsRestS...	GET /com.vbooking.flights.rest/rest/flights/VB297
5		12:58:13....	FlightsRestS...	200 - OK
6		12:58:14....	Reservation...	vbooking.reservations (vbooking.reservations.reply)
7		12:58:14....	Reservation...	vbooking.reservations.reply
8		12:58:14....	Reservation...	vbooking.reservations.reply
9		12:58:14....	Reservation...	vbooking.reservations.reply
10		12:58:14....	Reservation...	vbooking.reservations.reply

In the Immigration scenario there are multiple parts used from the Vacation Booking Demo Environment. Firstly, the user will interact with the vbooking-admin JSP webapp hosted by Tomcat.

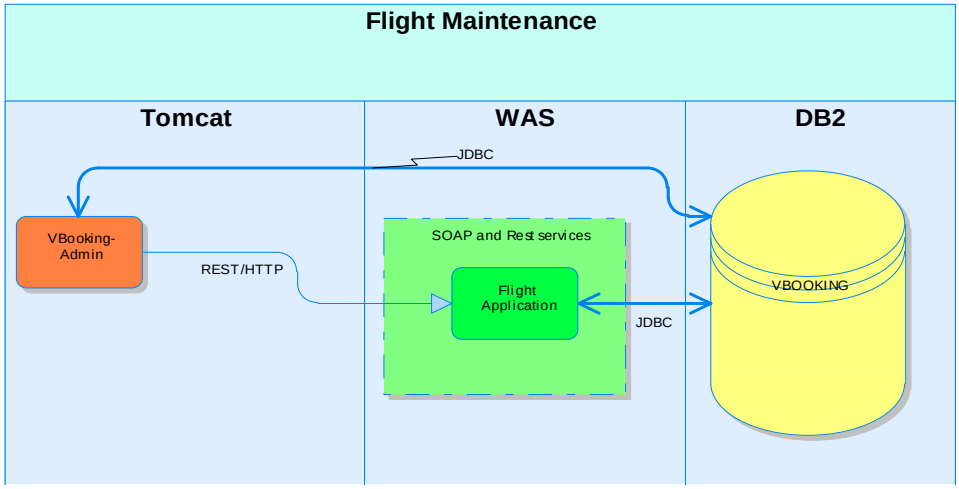
Secondly, the scenario make use of two WAS services: Flight Application and Reservations Application. The first one provides the process with the flight details through REST calls. The second WAS service deals with the processing of passengers that are posted on the MQ queue vbooking.reservations.*. The last component is the VBOOKING Database which stores all the information in this process.

When you are doing immigration, the following steps are happening in the backend:

1. The Tomcat JSP interface vbooking-admin will access the database through JDBC communication to get a list of flights.
2. For each flight returned to the vbooking-admin, the web app will make a REST request to the Flight Application WAS service for the destination.
3. After a user chooses a flight, vbooking-admin will publish a message on the vbooking.reservations queue with the flight chosen.
4. The Reservations Application WAS service will pick up the message on the queue and access the database through JDBC and return a MQ message for each boarded passenger on the vbooking.reservations.reply queue.

4.8. Flight Maintenance scenario

In this scenario the user will be able to edit details about flights and add new flights. There are multiple components active in this scenario and you can follow them in the diagram below:



The flight maintenance process is accessed through the vbooking-admin administration interface. (URL: http://localhsot:8089/vbooking-admin/index.jsp) This interface is provided by the Tomcat web server. Following along, in the background the vbooking-admin service communicates with the Flight Application WAS service which is a RESTful service. The last component in this process is the VBOOKING Database which stores all the information about passengers, flights and destinations.

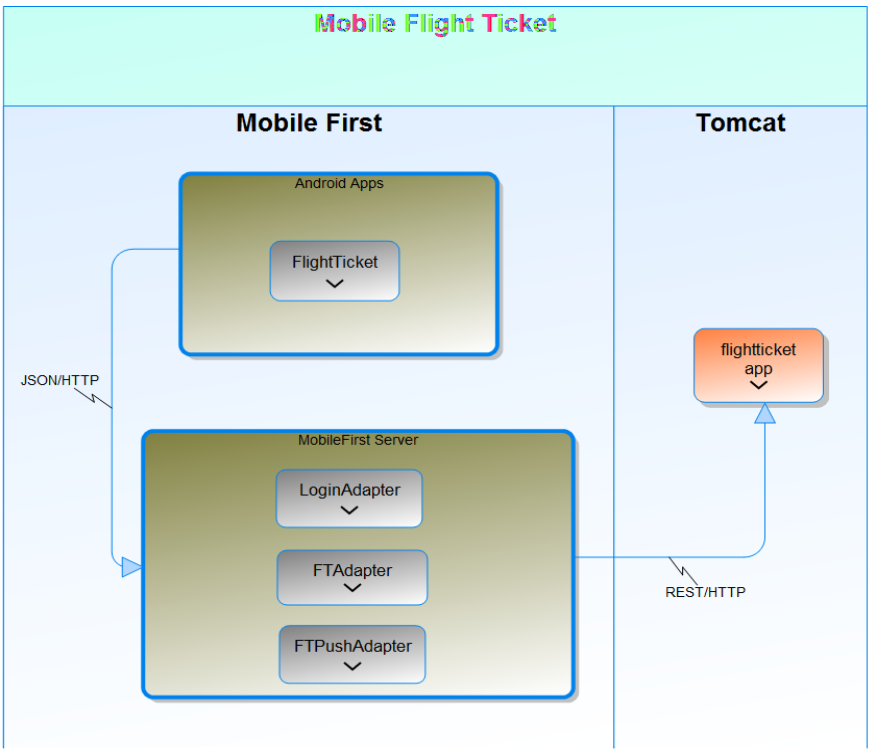
When you are performing a flight maintenance, the following steps are happening in the backend:

1. The Tomcat JSP interface vbooking-admin will access the database through JDBC communication to get a list of flights.

2. The vbooking-admin web app will also send a REST request to the Flight Application REST service hosted in WAS to access a specific flight.
3. The user will provide new details for a flight or update an existing one.
4. If the input is valid, the Flight Application WAS service will update the database using JDBC communication.

4.9. Mobile Flight Ticket scenario

This scenario uses a mobile first application running on Android to connect to a Mobile First server and then to the Vacation booking REST service. The mobile application allows a passenger to check in for a flight. There are multiple components active in this scenario and you can follow them in the diagram below:



The FlightTicket application is run on an Android device and provides the mobile user interface to allow the passenger to checkin for a flight.

Following along, the adapters run in MobileFirst Server and they mediate between the FlightTicket mobile application and the back-end service running on Tomcat. They expose a set of services to the mobile app, called procedures. The FlightTicket application invokes these procedures using Ajax requests. The procedure retrieves information from the back-end application.

The flightticket app running on Tomcat is a REST application that communicates with the adapters and provide them with information on the list of flights that the passenger is booked on..

When you checkin for a flight using the mobile application, the following steps are happening in the backend:

1. The user launches the FlightTicket mobile application and logs in using a username and password. This username and password is the first and last name of the user in Vacation Booking.
2. The users selects the button “My Flights” and this invokes the appropriate procedure in the adapters.
3. The adapter communicates through REST over HTTP with the flightticket app hosted by Tomcat. The flightticket app returns the list of flights that this user is booked on.
4. The data that is retrieved by an adapter is presented to the FlightTicket application as a JSON object.
5. The FlightTicket application processes the JSON result and displays the list of flights to the user. The user selects a flights that he wants to checkin.
6. This invokes another service through the adapters and they communicate the request over to the flightticket app. The flightticket app receives the request and updates the check-in information. The confirmation for check-in is then displayed back to the user.