Software Test Plan

Cloud Connect

2/26/2018
Project Sponsor: Tony Pallas
Faculty Mentor: Ana Paula Chaves Steinmacher
Version 1

Team Members:

Parth Patel
Abraham Ramirez
Jacob Serafin
Steven Strickland

Table of Content

1. Introduction

Within the hospitality industry, hotels are equipped with hardware on site to help produce an efficient and secure stay for customers. Devices on site transfer data to systems throughout the Internet where it can be used as a way to enhance the customer experience and keep track of various information. One example is the devices that are in place to organize reservation data to make sure customers can reserve rooms and services. Other devices work in similar ways, facilitating many services that are seen as essential for the modern hospitality industry.

Choice Hotels is a company that owns and operates multiple chains of hotels with up to 6,500 hotels deployed globally. They are a billion dollar company that operates, in aggregate, approximately 15.5 million available rooms each night. Each hotel is set up with a wide variety of sophisticated electronic systems. These systems provide security, conveniences, and luxuries for every one of the 15.5 million rooms every day. SkyTouch Technology is a Software as a Service (SaaS) company that provides an operating system for which they can move data from these electronic systems to online databases, allowing all of this information to be collected and used. Team CloudConnect has partnered with both SkyTouch Technology and Choice Hotels to help provide a more modern solution to their current systems.

The goal of our team is to improve the capability of SkyTouch's solution to access and manage the many varied hardware systems installed in today's modern hotels. Many of these hotels rely on old standard RS232 serial port connections to send and receive information from these systems on site at hotels. Figure 1 is a modeled representation of SkyTouch's current solution to proxy devices from any given hotel to and from their own database located within an online AWS database. Within the "on-site" section to the left represents any given hotel. The devices run through their own systems and, currently, send serial data to a PC at the hotel that

runs custom made software called JEDI (Java Enabled Device Interface). The JEDI will then translate the
incoming device data and from here, the JEDI then runs a proxy from the hotel to an online AWS database which is provided by SkyTouch's Amazon Web Services account.

Overall, this structure has been kept intact since 1998 when JEDI was first implemented into Choice Hotels' locations. This leads to several key flaws that cause massive waste in both time and money. Each JEDI must be custom setup and developed, wasting both time and money. In addition, the use of the old standard RS-232 connections cause these connections to be slow, further wasting time. To make this situation even worse, to keep this structure at any given hotel it costs a lot of money to maintain and develop as time continues. SkyTouch has begun to move their systems into a cloud-based solution, but have yet to resolve the issue of how to produce a cloud based system which would not need JEDI at each location.

Our proposed solution is to eliminate the need for JEDI by installing a server at each hotel to replace the current JEDI. This server will communicate serial data through IP/TCP protocols to a custom application hosted inside AWS, that will translate the incoming data and then send it to the current database within AWS. This solution will eliminate the need for a custom JEDI at each hotel, and will allow centralized management and development within AWS.

2. Unit Testing

2.1 Overview

Our unit testings consist of back end testing only as front end is a non-required part for our project. In our case our back end uses java and will be tested in a AWS environment that was provided by our client. To ensure our software is well tested, we plan on testing as many independent parts of the project with AWS and ensure files are being parsed and stored accordingly. We will begin testing by using our local client and server that we created to pass a file and parse accordingly. This will give us a local testing environment to begin with incase AWS has any issues. We will then fully transfer to AWS components and use the database included to store, extract and parse files.

Our testing will to an extent be divided into two parts. Locally and AWS. Since AWS components and database are more complex we will transfer to it once we have everything functioning locally in client, server and database. This will give us more freedom to run tests and correct them.

2.2. Local Backend

First focusing on the local backend we will break our implementation into different modules which will ensure they all work efficiently and respond accordingly to different files. Our Local back end will be broken into four modules these are client-server, XML File, Translation Thread and the Message Listener.

2.2.1 Client-Server

Overall Functionality

- The client-server Module is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters called clients. The Server Host runs one or more server programs which will share their resources with clients.

Details- The Client-server will ensure we can locally run our server (our own computer outside NAU's network) with no issues while we have multiple clients connecting to the server.

- Establishing Server
    - Ensure that we can start server in various computers(different networks) and possibly test within NAU's network and see if we can run it.
- Starting multiple clients

- Client side needs to connect to server accordingly and echo out that a connection was successfully made. We will be testing multiple clients in different computers to ensure multiple connections can be made. This will ensure that when testing we can pass in multiple files and loop through them accordingly and avoid errors later within the AWS server.

2.2.3 XML File

Overall functionality

- This module is a file, formatted in such a way that the Translation Thread can read it and use it to properly translated according to the instructions contained within this file. The format for this file and its instructions is described in the next section of this document.

Details- Files will follow a generic structure but will vary depending on what device the file is meant to be for. Each device will have a specific file written for it that the parser will use to understand how to parse the data and what to do with it. Each file will only need to be created once, after which they can be edited to reflect any changes in the device's configuration.

- ID Field
  - Contains a string
  - Identify validity of message
  - Parsing instructions

- Save Field
  - Contain Bits to be saved in message for Final message

- Combine Field
  - Contains Final values to be sent according to save field instructions
- Send Field
  - Send the Final Message from Combine field

2.2.4 Translation Thread

Overall Functionality

- This module takes the data from its connection and reads through each XML File until it finds a XML File that contains instructions for the type of data the module is trying to translate. Once it finds the correct XML File, it will then follow the instructions in that file. Once completed, the Translation Thread will then exit, releasing the resources it was using.

Details-Each translation thread is created specifically to handle one connection. It starts by reading through the current directory, generating a list of xml files. It then selects the first xml file, opening the file and searching for the files "ID" string in the connection's message. If the

“ID” string is not found in the message, the translation thread closes the file, removes it from the list, and repeated with the next xml file.

- Find ID String
    - Checks Validity of String and procedes

- Search Key Strings
    - Looks for message containing “Key” in the Save Field
    - Saves “Key” for later use
- Finalize Message
    - Finalizes all “Key” Fields and creates message whose format is specified in the “COMBINE” field

- Send Message
    - Send Message to database and quit

2.2.5 Message Listener

Overall Functionality

- This module is a loop that constantly cycles, listening for any incoming connections. When an incoming connection is successfully connected, the Message Listener passes the connection into a new Translation Thread. After passing the connection, the message listener returns to cycling and waiting from new incoming connections.

Details-The Message Listener module constantly loops, checking for any incoming connections. When an incoming connection is found, it will create a new translation thread, and give that thread the connection. Once the thread has started, the server continues to loops, starting the cycle over again.

- Secure Connection
    - Message Listener is always looping and listening for incoming connections
    - Connection stays running
- Creates translation thread
    - When incoming connection is found a translation thread is created and give that thread the connection
    - Connection stays listening once thread is started

2.3 AWS Backend

Our AWS backend consists of the AWS Server and three modules implemented fully together in the AWS Server. Since AWS provides the Server side the client-server module won’t be needed. Only the XML file, Translation thread and Message Listener will be implemented in the AWS virtual machine. They will work together and stay fully functional; storing data passed into the given database. Once we have all of our Local Backend working we can start testing with

the AWS Server and Database. This will allow us to see if connections can be made to the Server and files being passed are stored accordingly.

2.3.1 Connection to AWS Server

Overall Functionality

- This Service is responsible to have all connections connecting to the AWS server. The server will have our Module code being implemented here instead of locally and store the Data to the database.

Details- Make connections and have the Files stored accordingly in the AWS database.

- Client Connect successfully to server

2.3.2 Implementation of modules in the AWS Server

Overall Functionality

- This service will have all of our modules implemented together to be functioning on the AWS Server. The Code should be fully functional and do the steps mentioned in the Local backend. Implementation of the modules in AWS Server should store the parsed message and store it accordingly in the database.

2.3.3 AWS Database

Overall Functionality

- In practice, our database is a mock of what SkyTouch's own database would look like. The database should be have tables integrated for the two use cases we are providing and will have cells for the data each incoming message provides.

Details- Store the message that was parsed by our XML file.

- Store message
- Retrieve data for a message's function

3. Integration Testing

3.1 Overview

Our focus of the integration testing will be testing the connectivity between the client and AWS Server. We have to ensure that our Code is working correctly and parsing files accordingly in the AWS server. We have to also ensure that we can have multiple client connected to the server and at the same time sending xml files to it. This will allow us to make sure the XML file, the Translation thread, and the Message Listener modules are working as intended.

3.2 Backend

All of our testing will rely on backend. We will check XML files are being read correctly and parsed correctly. If multiple files are being sent to the server we have to check that they are being held in line within the Message Listener module. If a bad XML file is sent we have to ensure the Translation thread handles this exception.

Here are some of the tests we will conduct :

3.2.1 AWS Server Integration

Code implementation

Our main focus will be implementing the main modules into the AWS server which are the XML file, the Server Threading and the Message Listener. We will be conducting multiple tests to ensure everything is working as expected.

- Client connections
    - We want to have various clients connected to the server and establishing a valid connection. This will need to be tested for:
        - Multiple established connections working as intended
        - Ensure that correct ports are open and implemented correctly
        - Different network connections that can connect

- Client XML File Transfer
    - We will be testing different XML files that are going to be transferred to the server, this includes Files with an invalid ID to ensure the server threading module is working correctly, A Valid ID to ensure it actually works as intended and multiple clients waiting to ensure the Message Listener is listening to incoming connections. This will help us test:
        - Message Listener
        - Server threads
        - Message Parser

3.2.2 AWS Database Integration

Data Mapping Test

- Test to check if the data that is being sent back and forth within AWS is accurate for On-Premises devices and SkyTouch Database with in AWS

Data mapping will also check the following:

- Device is recognizable with its given id
- Establishing connection to database
- Data is accurately sent back

- Device can not receive different kind of data
- Device keep the connection during the transition period

4. Usability Testing

4.1 Overview

Usability testing is used to make sure users get the experience intended for them when using the technology. We represent different “devices” as our users. For example, usability tests will check if we can send different types of XML files over the server and back. Different user will have different third party devices so they would all have different XML files. We will also test different security levels for different devices. We will use this opportunity to test our Java socket/port communication between AWS and different “Hotel Premises”. This is help us with how many different devices can be on the server at once. Usability tests check if the software can run and be used.

4.2 End to End Testing

Once we get our project hosted on the AWS virtual machine, we will use AWS as our testing platform for both ends since Skytouch has Hotel OS and device specs on AWS. AWS already has some built-in security feature so we will be testing that for our port forwarding. It will give us a real time feedback.

In addition, Our sponsor will be setting up a mock device that would act as our client device with several use cases. This test will be able to gather information about different devices and will be able to provide feedback. Since SkyTouch is partnered with many devices, this would help them determine which devices will need future updates so that SkyTouch could get up to data on the Cloud Connect Technology.

4.3 Use Case

We have use cases for each type of devices because we want to provide the technology that can be use able for all of the current SkyTouch partners as well as future partners. Since our product is heavily geared within the back end, these tests will not be seen by the actual SkyTouch front end, but this will help revolutionize their back-end Architecture. With that in

mind, we would like to provide some cases with steps for Back-end Architecture and Non-Technical Front end users:

We will have two different types of users for each case: Back-End Architecture and Non-Technical Front end users. Each user will be given list of different cases such as if charges is going to decline, room is empty, wrong name, wrong room number and such. This tests will be done live as well as locally since AWS may prevent us from testing live for security purposes.

4.3.1 One way communication(Third Party Device to SkyTouch Software)

Back End Architecture will see this on a command line when they run our code:

- Device connects with AWS with TCP connection(Port forwarding)
- Devices prepares appropriate XML files
- Sends XML files to AWS (Cloud Connect Plug-in)
- Cloud Connect Plug-in will phase the data and sends different data fields to SkyTouch's database within AWS
- Now appropriate data field will be updated within Database

Non-Technical Front End users:

- User will use PBX device to enter a certain code to Hotel OS

*Back-End Architecture will see if it worked or not.

4.3.2 One way communication(SkyTouch Software to Third Party Device)

Back End Architecture will see this on command line when they run our code:

- Hotel OS connects with AWS with TCP connection(Port forwarding)
- Devices prepares appropriate XML files and sends to AWS
- Cloud Connect Plug-in will phase the data and sends different data fields to SkyTouch's database within AWS
- Now appropriate data field will be updated within Database

Non-Technical Front End users:

- User will use Hotel OS to enter certain code to PBX

*Back-End Architecture will see if it worked or not.

4.3.3 Two ways communication(Third Party Device to SkyTouch Software)

Back End Architecture will see this on command line when they run our code:

- Device connects with AWS with TCP connection(Port forwarding)
- Devices prepares appropriate XML files
- Sends XML files to AWS (Cloud Connect Plug-in)
- Cloud Connect Plug-in will phase the data and sends different data fields to SkyTouch's database within AWS
- Cloud Connects Plug-in will create appropriate message and sends it back to appropriate device
- Device will receive the appropriate message

Non-Technical Front End users:

- User will use POS device that will prompt user to charge certain amount to certain room number
- User will be notify if certain cases description
- User will be notify almost instantly if their case passes or fails

This is the tests we hope most carries our since a number of different user cases will be created to make sure we cover all of the scenarios. This will help us restructure our technology based on users feedback.

5. Conclusion

SkyTouch Technology is a multi-million dollar software company. The new VADER technology that we have developed aims to help SkyTouch Technology save money by: avoiding their custom made java application for each hotel, and avoiding reliance on slow RS-232 connections. All the VADER modules have been integrate and now we need to carry out our testing to validate functionality and make sure our technology is free of errors, faults, and failures. We plan on carrying out unit testing, integration testing and usability testing. We believe that our plan discussed above will be able to prove whether or not out technology is ready for operation within SkyTouch's production environment.