SE 3XA3: Test Plan
R-DB V2

Team #31, R-DB V2
Jason Tsui tsuij8
Hareem Arif arifh1
Student 3 name and macid

October 26, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Oct 25, 2018	1.0	Section 1
Oct 26, 2018	1.1	Section 4 and 6

1 General Information

1.1 Purpose

This document is a Test Plan document. This document will discuss and describe the testing, validation, and verification procedures to implement a discord chat bot, R-DB V2. As of document creation, the project is still undergoing major development and the test case procedures and descriptions disscussed are subject to change.

1.2 Scope

This project is a python based server hosted chat bot, which is able to take user input and output into the discord community server. The scope of testing for this program will cover user input, Discord API integration, and expected outputs.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
API	Aplication Programming Interface
VoIP	Voice over Internet Protocol

1.4 Overview of Document

R-DB V2 will be following the logical structure for testing purposes. Unit testing will primarily focus on handling user input and program logic. Integration testing will be devoted to integrating the bot onto the Discord API platform. System testing will be be done by observing if chat bot commands are correct. Acceptance testing will be done by a 3rd party user to evaluate non-functional requirements.

Table 3: **Table of Definitions**

Term	Definition
Discord	Freeware VoIP application. VoIP service which connects people and creates voice chat communities
VoIP	Methodology for communicating over the internet
R-DB V2	Name of project
Black Box Testing	Functional testing by probing program with inputs. Testing is concered with output of program
White Box Testing	Structural testing by understanding how program processing occurs. Testing is concered with soundess of program
Pytest	Python testing framework
Red Discord Bot	Original program which project is based off of

2 Plan

2.1 Software Description

The software for our product consists of a Discord Bot implemented in python coding language. The environment that the software is being created in is the IDLE environment provided with python by default. The testing that will be done will also be done in the same environment in the same language.

2.2 Test Team

Our group test team is comprised of all members contributing in different aspects. As the files to be implemented have been split up into the group, testing will be done both by the creator of each file of code as well as a level of testing done by other group members to ensure higher coverage.

2.3 Automated Testing Approach

As our implementation is done with python we can make use of the ability to do automated testing through unit testing and integration testing. Unit testing will be done on the individual files that make up the product to ensure that the functions within each file are performing as intended. Integration

testing will be done to ensure that the files are interacting with one another in a cohesive manner and performing their collaborative functionality as intended.

2.4 Testing Tools

The testing tools that being intended for use consist of unittest which is the built in standard library tool that is used for testing python code. Alongside that we are considering using pytest as well which is also a complete testing tool.

2.5 Testing Schedule

Gantt Chart showing the project schedule.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 File upload

1. Music upload from YouTube

   Type: Functional, Manual

   Initial State: No music playing in voice chat

   Input: Play command followed by a YouTube URL

   Output: Music from the video plays in the desired voice chat

   How test will be performed: A manual test will be utilized by entering the play music command and checking if the music from the video starts playing in the server.

2. Music from local files

   Type: Functional, Manual

   Initial State: No music sent from local files

Input: play command followed by the source to the MP3 file on the user's computer.

Output: The MP3 chosen is played in the voice channel

How test will be performed: This test will be performed manually by choosing a file from a local device and checking if it plays in the voice channel.

3.1.2 Moderation

1. Kick
   Type: Functional

   Initial state: Server full of participants

   Input: The kick method with a username

   Output: The user will be removed from the server and a success message be sent by the bot.

   How test will be performed: There will be a function that calculates the amount of users in the server before and after the kick command is called to ensure that the number of users is less after the command is called.

2. Chat cleanup

   Type: Functional

   Initial state: A chat with N number of messages.

   Input: The chat cleanup command

   Output: An empty chat log

   How test will be performed: There will be a function that returns the number of messages in a chat. This function will be called after the chat cleenup command is called to check that the amount of messages is 0.

3.2 Tests for Nonfunctional Requirements

3.2.1 Performance

Speed of responding to commands

1. Speed of responding to commands

   Type: Functional, Dynamic

   Initial State: Bot is active

   Input/Condition: Any command

   Output/Result: The amount of time it took for the command to be processed

   How test will be performed: A timer will start before the method associated with a command is called and finish when the method is done executing.

2. Reliability and availability

   Type: Functional, Dynamic

   Initial State: Bot is active

   Input: Command BotStatus is called

   Output: A boolean value

   How test will be performed: Every hour the command BotStatus will be called to ensure that the Bot is up and running around the clock.

3.2.2 Usability

1. Ease of use

   Type: Functional, Manual

   Initial State: Bot is active

   Input: Launch a server containing the bot

   Output: Commands should work as they are supposed to.

   How test will be performed: Launch the server on multiple devices with different users connect to ensure that the Bot runs for all intended environments.

4 Tests for Proof of Concept

4.1 Integration Testing for Bot

Integration Tests Testing will focus on implementing our bot and related python scripts to the Discord API and framework.

1. test-Initial Function Integration

   Type: Manual Functional System Testing

   Initial State: Bot is connected to community server, planning to implement a new functionality to bot

   Input:Command to bot on Discord server

   Output: Bot returns appropriate response or action

   How test will be performed: Blackbox testing for initial functionalities. Will test every bot command implemented.

2. test-Edge cases

   Type: Automated Equivilence Testing

   Initial State: Functionional requirement has been implemented to bot with manual testing. Testing is still not thorough to guarantee correctness.

   Input: Edge-cases

   Output: Expected output of select command

   How test will be performed: Create test cases and error handling for bot. Then select edge cases for commands and create a script to input commands for testing.

4.2 Structural System Testing for Bot

Structural System Tests Testing will focus on the structural system of the bot.

1. test-Recovery

   Type: Manual Functional Testing

   Initial State: Bot is assumed to be correct

   Input: Input incorrect commands and commands with wrong parameters

   Output: Bot handles errors and returns to ready state

   How test will be performed: Input incorrect commands and observe if bot is able to return to normal function

2. test-Stress

   Type: Automated Functional Testing

   Initial State: Bot is assumed to be correct

   Input: Bot commands

   Output: Bot output

   How test will be performed: Create a script of bot commands. Run script. Check if commands output and bot returns to ready state.

5 Comparison to Existing Implementation

With regards to comparing our implementation to the original implementation, we intend on testing the performance comparison by running a series of identical commands on both implementations to see how they perform. As the original implementation does not come with corresponding test files, we do not have a comparison for testing choices, although we have considered the possibility of running a portion of our test plan on the original implementation to see if the overall functionality is the same. Beyond that the results of testing on our own implementation and the comparison to the original will be updated as the project continues to develop.

6 Unit Testing Plan

6.1 Unit testing of internal functions

The internal functions in each of the files for our product will be tested via unit testing. That pertains providing input values and implementing the methods on them, then seeing if they provide adequate output. The inputs will range from values that should generate proper output and values that should trigger an exception and then be handled accordingly. No stubs or drivers will be required to implement this area of testing.

6.2 Unit testing of output files

For unit testing of the output, we intend on running a series of commands on the software and comparing the output results it gives to a series of expected results to see if the Bot is communicating as intended. Asides the communication aspect, the various interface aspects such as the background music and interface appearance will be tested for simple functionality, if the music plays and if the interface appears as intended.

7 Appendix

7.1 Symbolic Parameters

1. BOT_ON - Bot is on
2. BOT_OFF - Bot is off
3. BOT_EXE - Bot is executing
4. BOT_RDY - Bot is ready and not executing

7.2 Usability Survey Questions?

1. Was the bot easy to use? What are some commands that you don't understand?
2. What commands do you see yourself using the most?
3. What additional functions would you like to have implemented?