Major Project Report

On

WebLogic Advanced Feature Tool (WAFT)

By

Biswadeep Sarkar (201900322)

In partial fulfilment of requirements for the award of degree in

Bachelor of Technology in Information Technology

(2023)

Under the Guidance of

Internal Guide	*External Guide*
Mr. Nirmal Rai (Assistant Professor, Dept. of Information Technology)	Satheesh Kumar Thupa (Principal Software Engineer, Dell Technologies)

DEPARTMENT OF INFORMATION TECHNOLOGY

(A constituent college of Sikkim Manipal University)

MAJITAR, RANGPO, EAST SIKKIM – 737136

ACKNOWLEDGEMENT

I wish to express my sincere thanks to External guide **Satheesh Kumar Thupa, Principal Software Developer, Test Environment Management (TEM)** of **Dell Technologies** for providing me an opportunity to carry out project work on “WebLogic Advanced Feature Tool” and their unlisted encouragement and guidance carrying out this project work.

I sincerely thank my Internal guide **Ms. Nirmal Rai, Assistant Professor,** Department of Information Technology, Sikkim Manipal Institute of Technology for his guidance and encouragement in carrying out this project till the completion.

I would like to express my sincere thanks to **Dr. Udyan Baruah**, HOD, Information technology for allowing me to carry out my project from 16th Jan 2023 to 26th May 2023 and valuable support and guidance during the project period.

Lastly, I wish to avail myself of this opportunity, express a sense of gratitude and love to all our teachers, staff of the department of Information Technology, friends and fellow for their support and help.

Biswadeep Sarkar (Reg no. 201900322)

To whomsoever it may concern

This is to certify that **Sarkar, Biswadeep** from Sikkim Manipal Institute of Technology, was working with Dell Technologies for his Internship Project.

Project Details	
Project Name	WebLogic Advanced Feature Tool
Duration	16-Jan-23 to 26-May-23
Location	Hyderabad
Reporting Manager Name	Mattaparthi, Bhargav

He has successfully completed the project. His findings in course of the project has been found to be practical and relevant. Some of the recommendations will be incorporated on approval from the business.

His performance during the tenure of the internship has been found to be satisfactory.

Thanking You,

Regards,

Santosh

Santosh TK
Talent Acquisition Director
Dell Technologies

LIST OF CONTENTS

ABSTRACT **1**

1. INTRODUCTION **2**

2. DESIGN STRATEGY **8**

3. IMPLEMENTATION DETAILS **12**

4. RESULTS AND DISCUSSIONS **16**

5. GANTT CHART **17**

6. SUMMARY AND CONCLUSION **18**

7. REFERENCES **19**

ABSTRACT

Oracle WebLogic Server Architectures are known for a relatively more complex structure than today’s new server and cloud architectures. As such, operations and maintenance activities on WebLogic servers are challenging and time consuming with multiple instances where functionalities and existing safety mechanisms can break if there are human-errors due to the manual operations.

The need for removing the human-errors and reducing operations time for the Test Environment Management Teams was becoming of paramount importance given the time and resources developers had to use in handling and managing the WebLogic servers. Dell Technologies Test Environment Management teams must handle and manage more than 300 server applications across multiple global, test and production environments. Besides, they must engage in daily operational tasks as well.

Thus, there was a need to design robust software that could handle heavy loads for seamlessly performing WebLogic server maintenance and application management across multiple environments, without human intervention or direct interaction with the servers.

The WebLogic Advanced Feature Tool reduces the time elapsed for the operations from 24+ hours to less than 3hrs, without compromising on the quality of operations, integrity of server data health of the servers. It implements all operations remotely and securely without directly controlling server operating systems. It provides relevant diagnostics post the operations for further development and debugging activities.

1. INTRODUCTION

General Overview of the Problem

WAFT or The WebLogic Advanced Feature Tool is created specifically for automating, handling, and managing WebLogic server operations and activities.

There four teams under the Test Environment Management Domain which engage in handling and operating on WebLogic application servers on a day-to-day basis. They must ensure that the servers ae working correctly and they are exactly serving the purposes they are meant to serve.

The ever-expanding number of applications across multiple environments are increasing the task in hand for the developers of the teams. Every developer is in charge of on application server under a give environment, but the number of applications and environments keep increasing. As such a software interface is highly required for simplifying the tedious tasks of server operations.

To overcome these challenges, whenever there is an issue with an application server, an automated service that can perform operations and management of the servers on a large scale, with proper accuracy and security measures, eliminating the need for human intervention, was very much needed and that is what the WebLogic Advanced Feature Tool aims at doing.

Literature Survey

Sl no.	**Author**	Paper and Publication Details	Findings	Relevance to the project
1.	**Dean Jacobs**	Distributed Computing with BEA WebLogic Server by Jacobs, D., 2003, January. Distributed Computing with BEA WebLogic Server. In CIDR.	BEA WebLogic Server is a distributed implementation of the Java platform for application servers, the Java2 Enterprise Edition (J2EE). WebLogic Server supports a variety of application programming interfaces, including ones for servlets, components, messaging, database access, and naming.	Understanding the comprehensive architecture of the WebLogic Servers is essential in implementing the algorithms in the software.
2.	**Pantazis Deligiannis, Alastair F. Donaldson, Jeroen Ketema, Akash Lal, Paul Thomson**	Asynchronous Programming, Analysis C and Testing with State Machines, Deligiannis, P., Donaldson, A.F., Ketema, J., Lal, A. and Thomson, P., 2015, June. Asynchronous programming, analysis and testing with state machines. In Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (pp. 154-164).	In programming asynchronous systems the main concern is responsiveness. To achieve responsiveness, long running sequential activities are often split into multiple shorter tasks that are scheduled asynchronously. If the tasks are not coordinated properly, the interleaving between tasks from different activities can be a source of bugs.	Asynchronous programming is essential in large scale systems where the processing time needs to be reduced to resolve time redundancies and increase max CPU utilisation

Table: Literature Review

Problem Definition

To design software for seamlessly performing WebLogic server maintenance and application management across multiple environments, without human intervention or direct interaction with the servers.

Reduction of the time elapsed for the operations from 20+ hours to less than 3hrs, without compromising on the quality of operations, integrity of server data health of the servers.

Implementation of all operations remotely and securely without directly controlling server operating systems.

Provision of user panel with the options of selecting the applications and respective environments on which the operations will be performed.

Implementation of security blackout measures for uninterrupted operations without raising INCIDENTS.

Testing all components and final deployment.

Proposed Solution Strategy

- Implementing the python FastAPI framework for developing Microservices and RESTful APIs for implementing the operations on WebLogic servers.
- Managing and handling data and credentials from Dell Technologies' repositories, for making multiple remote connections and security verifications.
- Using asynchronous/concurrent computation for handling the bulk applications across multiple environments, to bring down the operations time and increase efficiency.
- Using concurrent secure shell protocol to handle multiple servers together.
- Using meticulously designed algorithms and logic to reduce processing time and implement security blackout mechanism for error proofing from the default server security mechanism.
- Providing a comprehensive menu/dashboard for the selection of applications and environments on which the operations are to be performed.
- Writing unit test cases for all modules and deploying through CICD pipelines.

Software Requirements Specifications

Functional Requirements:

1. Taking in server operation specifications from UI
2. Connect securely to TEM Repositories for fetching important credentials.
3. Parsing and validating credentials for further steps.
4. Connecting to remote server Linux box with proper authentication
5. Invocation of wlst.sh tool with self-developed Jython scripts for performing a user-specified server operation.
6. Making appropriate JSON responses for showing operation details to the user.

Non-Functional Requirements:

1. **Performance:** The application must be able to handle and manage multiple application servers across different environment at once without compromising on data integrity and server security. It should be able to take heavy operation loads and avoid deadlocks and race conditions.

2. **Usability:** To provide users with a simple and useful UI with proper user experience for taking in the business segments and operation specifications as input.

3. **Availability**: The application will be hosted on Cloud foundry servers to be available 24/7 for assistance to developers.

4. **Interoperability:** The application made on AngularJs, will be supported by all web browsers thus increasing usability and flexibility.

5. **Security:** The requests made to the backend APIs are secure and have SSO implemented together with CORS and SSL security to prevent external security breaches.

Hardware Requirements

1. RAM: 8GB
2. Processor: Intel / AMD processor
3. Hard Disk: 50GB or more
4. Speed: 2GHz or more

Software Requirements

1. Operating System: Windows

2.Tools/Scripting Languages used: Python, WebLogic Scripting Tool (WLST), Bash/Shell scripting, Jython, Angular Js.

3. Important Python Libraries Used: asyncio, aiohttp, asyncssh

2. DESIGN STRATEGY

Overall Architectural Diagram

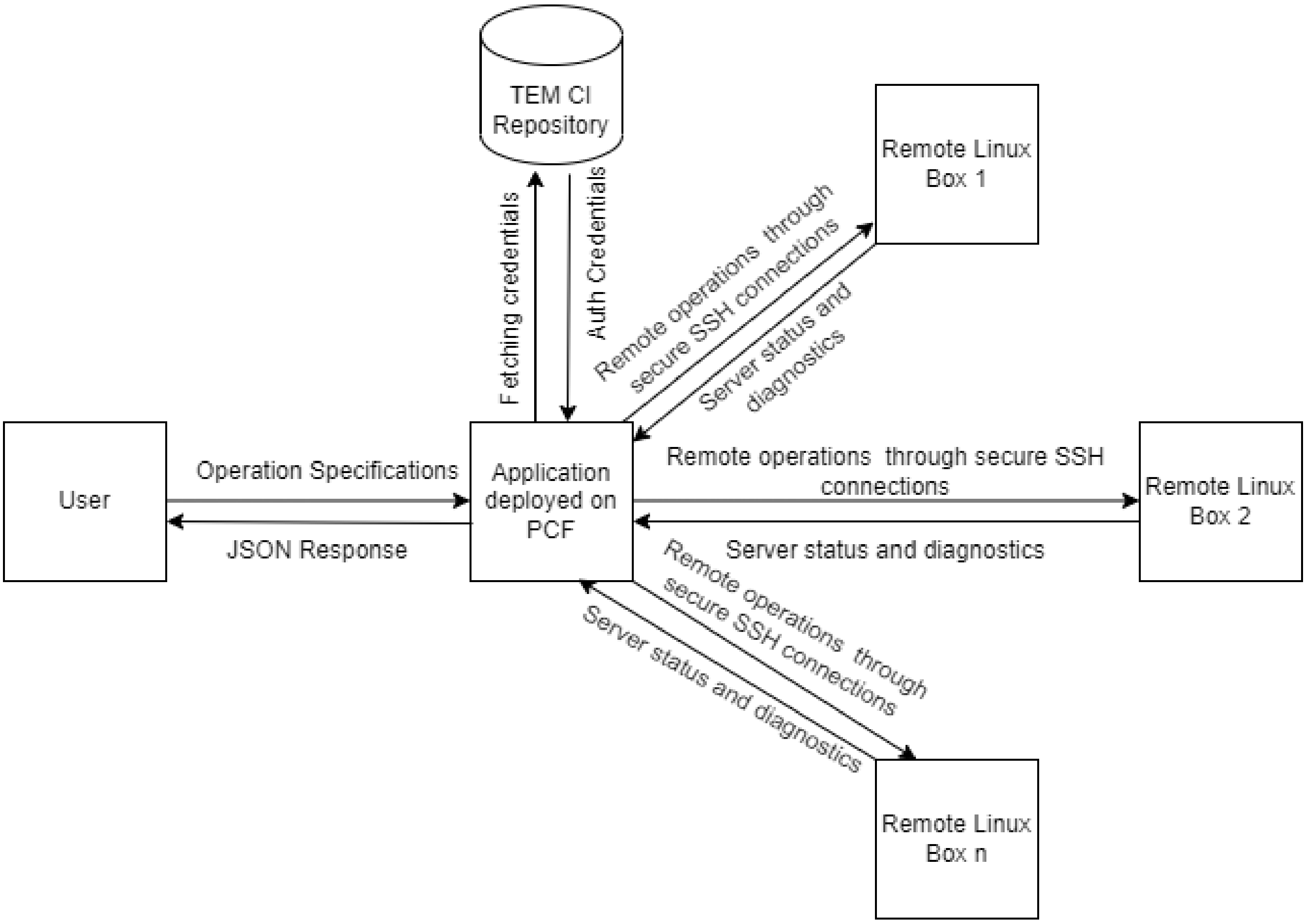
TEM CI Repository
Fetching credentials
Auth Credentials
Remote Linux Box 1
Remote operations through secure SSH connections
Server status and diagnostics
User
Operation Specifications
JSON Response
Application deployed on PCF
Remote operations through secure SSH connections
Server status and diagnostics
Remote Linux Box 2
Remote operations through secure SSH connections
Server status and diagnostics
Remote Linux Box n

Fig: Overall system architecture

User Interface: The user interface is the front-end component of the application that users interact with. Made with Angular Js it provides a simple yet robust interface for taking operation specifications as input.

Backend: Microservices and APIs are developed using the FastAPI framework, that performs the actual operations on the WebLogic servers.

WebLogic Scripts: WebLogic scripts have been developed for running the same with the WebLogic Scripting Tool for executing the operations on WebLogic servers.

Data Flow Diagrams

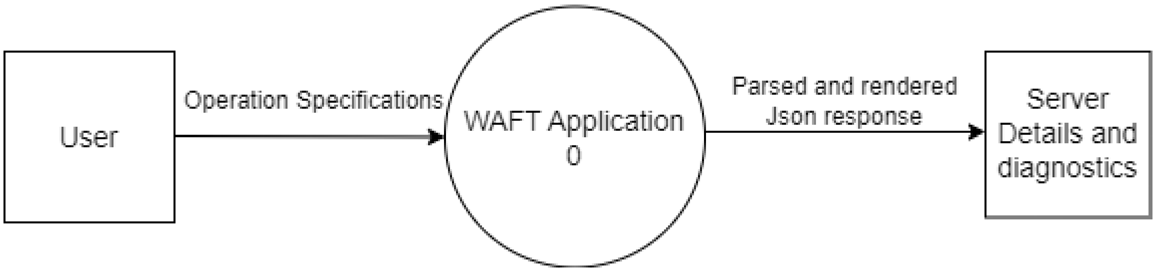

Fig: Level 0 DFD

User signs in through SSO on the WAFT Application and begins the process for WebLogic server operations and gets the server details and other server diagnostics details upon completion of the process.

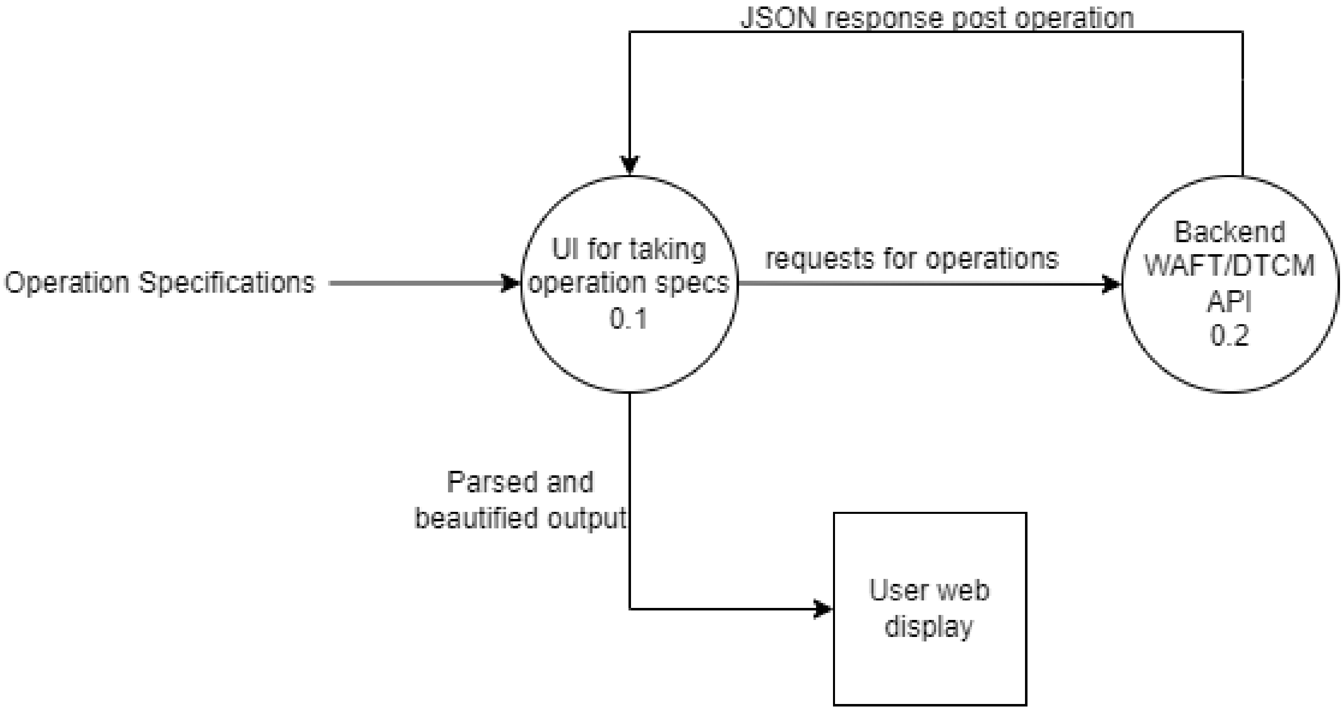

Fig: Level 1 DFD

The operation specifications are taken through the User Interface of the application and then from the UI a POST request is made to the backend microservices/APIs of the application for performing the WebLogic Operations.

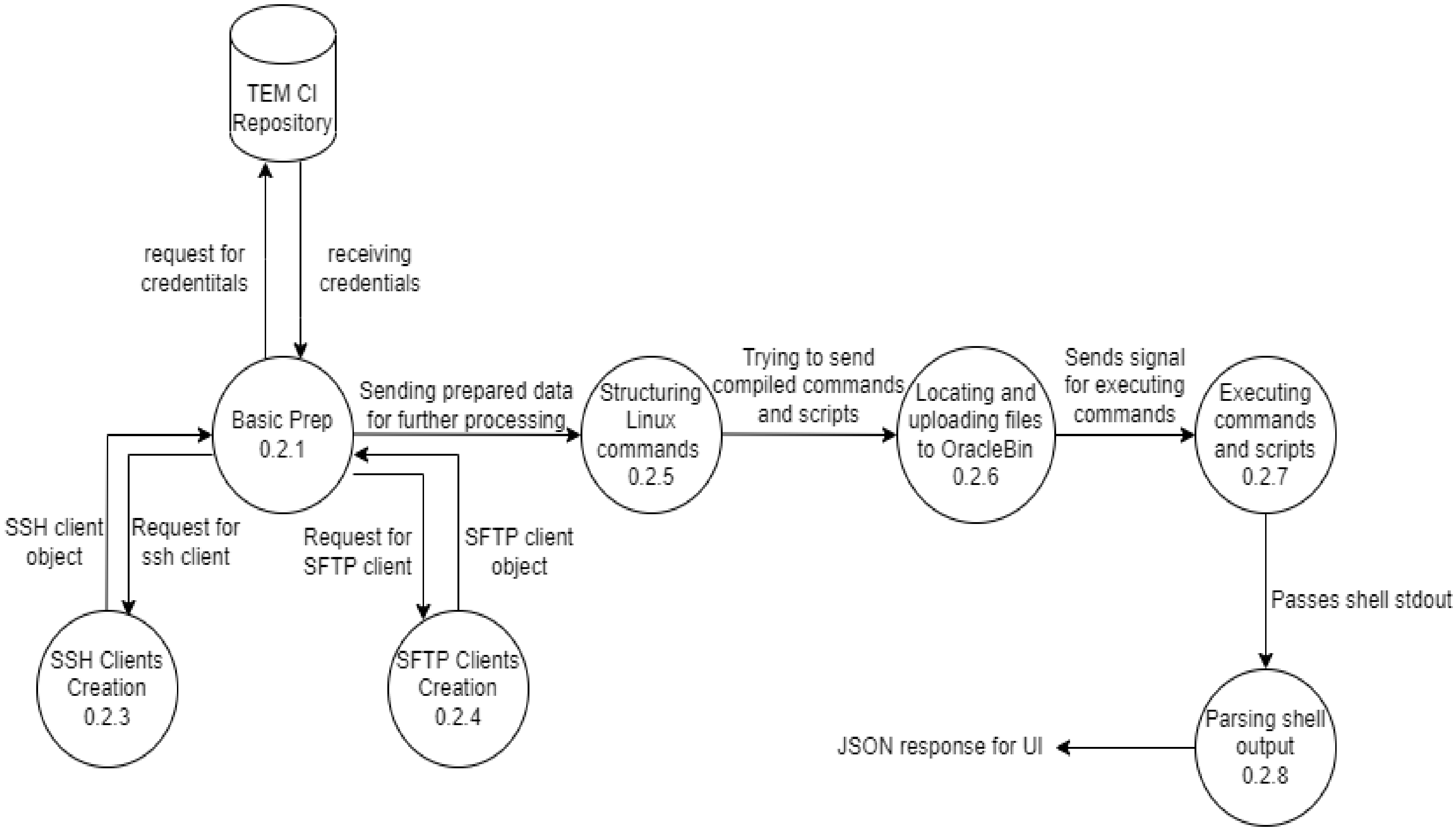
TEM CI Repository
request for credentitals
receiving credentials
Basic Prep 0.2.1
Sending prepared data for further processing
Structuring Linux commands 0.2.5
Trying to send compiled commands and scripts
Locating and uploading files to OracleBin 0.2.6
Sends signal for executing commands
Executing commands and scripts 0.2.7
SSH client object
Request for ssh client
Request for SFTP client
SFTP client object
Passes shell stdout
SSH Clients Creation 0.2.3
SFTP Clients Creation 0.2.4
Parsing shell output 0.2.8
JSON response for UI

Fig: Level 2 DFD

Upon the Backend API being called, the initial step is to start the basic preparations. Under the basic preparations, the SSH clients are created, the SFTP clients are created, credentials are fetched from the TEM CI repository and then further processes are performed by forwarding the data received to the next steps.

The Linux commands are then structured based on the basic preparations and WebLogic scripts are crated and compiled for being uploaded to remote Linux Boxes later.

The Oracle Bin is located in the remote server and the files or the WebLogic scripts and commands are uploaded to the location in that server. Finally, the WebLogic scripting tool is invoked together with the WebLogic scripts to perform the specified operations and the shell stdout is received.

Finally, the shell output is parsed and a JSON output is prepared for sending to the UI.

Flowchart of the system

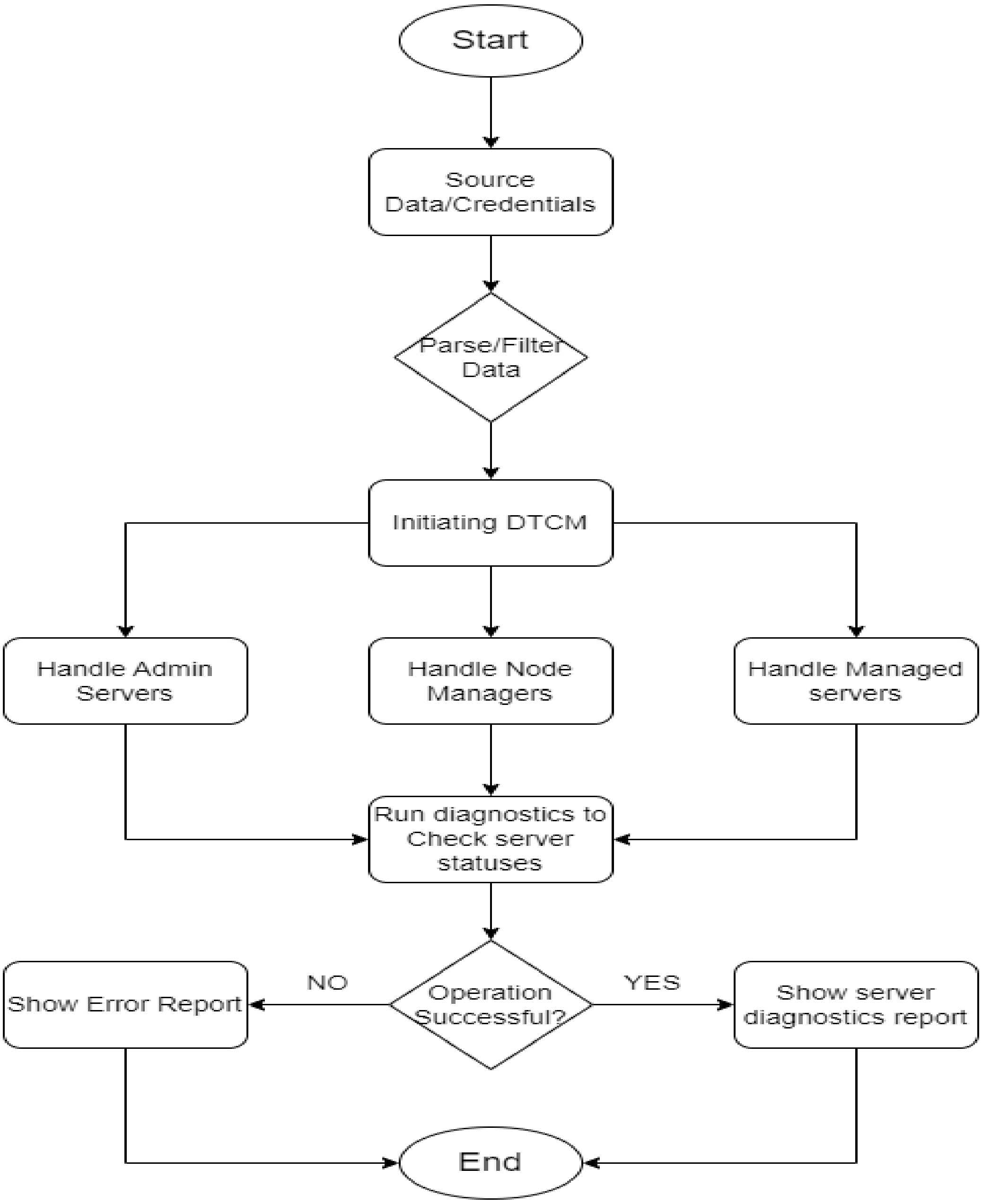

Fig: Flowchart of the system

3. IMPLEMENTATION DETAILS

Logical Details

1. Operation Specifications Input from UI

- Simple and user-friendly UI made with Angular Js is used to capture the user input regarding the operation specifications and other important parameters.
- Makes a POST request to the backend APIs developed on Python's FastAPI framework

2. Fetching Credentials from TEM Repository

- Makes asynchronous HTTP GET requests to an internal API for fetching relevant credentials as is required for authentication ahead.
- Verifies SSL certificate for making the API GET requests.
- Parses and verifies the data received and stores comprehensive credential information in a data dictionary.

3. Secure connection to remote Linux Boxes

- Using suitable SSH Credentials fetched, the application makes secure SSH connections to the remote shells, simultaneously with multiple Linux boxes as mentioned/specified by the user.

4. Implementing WebLogic Operations through WL Scripts

- Locates and invokes the WebLogic Scripting Tool.
- Uploads the Jython wl scripts
- Executes the Linux and Jython commands with wlst.sh to perform the operations
- Sends the output back to the application for processing and parsing.

5. Parsing and extracting data

- The shell outputs received from the remote Linux Boxes are parsed and meaningful information are extracted.
- The server information and different diagnostics details are also extracted and a meaningful JSON is passed to the front end, which eventually displays a message to the user in a human-readable format.

6. Closing connections

- All existing connections like the open SSH connections, SFTP connections etc are safely closed
- Interactive communication channels, if open, are closed as well.

High-Level General Codes/Algorithms applied

1. **Taking input from UI**

```
   var app = angular.module('myApp', []);

   app.controller('myCtrl', function($scope) {
     $scope.formData = {}; // Create an object to store the form data

     // Function to handle the form submission
     $scope.submitForm = function() {
       // Access the form data through $scope.formData
       console.log($scope.formData);
       // You can perform further processing or send the data to a server
     };
});
```

2. **Making asynchronous requests to fetch credentials**

```
   import asyncio
   import aiohttp
   async def fetch(session, url):
       async with session.get(url) as response:
           return await response.text()
   async def make_get_requests():
       urls = [
           'https://example.com/page1',
           'https://example.com/page2',
           'https://example.com/page3'
       ]
   async with aiohttp.ClientSession() as session:
       tasks = []
       for url in urls:
           task = asyncio.ensure_future(fetch(session, url))
     tasks.append(task)
```

(continued …)

```
# Wait for all the tasks to complete
    responses = await asyncio.gather(*tasks)

    # Process the responses
    for response in responses:
        print(response)

# Run the asynchronous code
loop = asyncio.get_event_loop()
loop.run_until_complete(make_get_requests())
```

3. Making asynchronous SSH connections

```
import asyncio
import asyncssh

async def run_command(hostname, username, password, command):
    async with asyncssh.connect(hostname, username=username, password=password) as conn:
        result = await conn.run(command)
        return result.stdout

async def make_ssh_connections():
    hosts = [
        {
            'hostname': 'example.com',
            'username': 'your_username',
            'password': 'your_password',
            'command': 'ls -l'
        },
        {
            'hostname': 'example.net',
            'username': 'your_username',
            'password': 'your_password',
            'command': 'df -h'
        }]
```

(continued …)

```
    tasks = []
    for host in hosts:
        task = asyncio.ensure_future(run_command(
            host['hostname'], host['username'], host['password'], host['command']
        ))
        tasks.append(task)

    # Wait for all the tasks to complete
          results = await asyncio.gather(*tasks)
```

4. One of the WebLogic Operations performed (starting managed nodes)

```
connect('admin_username', 'admin_password', 't3://admin_server_host:admin_server_port')

    start(block='true', blockTimeout=600000, blockIncrement=60000)

    servers = cmo.getServers()
    for server in servers:
        server_name = server.getName()
        print("Starting server: " + server_name)
        start(server_name, 'Server')

    disconnect()
```

4. RESULTS AND DISCUSSIONS

Input Example:

```
{
   app_id: [x,y,z….],
   env_id: [a,b,c,…],
}
```

Output Example:

```
{
   "Data":{
      "ServerXYZ": "Successfully Started",
      "Server status": "RUNNING",
      "Server health": "OK",
      "Server machine": "abc@",
      "Server port": "1234",
      "Server report": "diagnostics report...",
   },
   "Error":{
      "message": "No error logged"
   }
}
```

Brief Overview of the Results

1. The application successfully performs the WebLogic Operations post which the API returns a relevant JSON to the front end
2. Using the data received in the front end, a table of the servers and their respective details and statuses are displayed to the developer/user.
3. It brings down the processing time by more than 100% from 24+ hours to 4 hours maximum
4. Ensured proper handling of coroutines to prevent race conditions and deadlocks in asynchronous events
5. Handled several errors and exceptions that occurred at various points in the process. Further, through thorough trial and testing of the application, the application was made 99% error-free and reduced manual tasks by 60%..

5. GANTT CHART

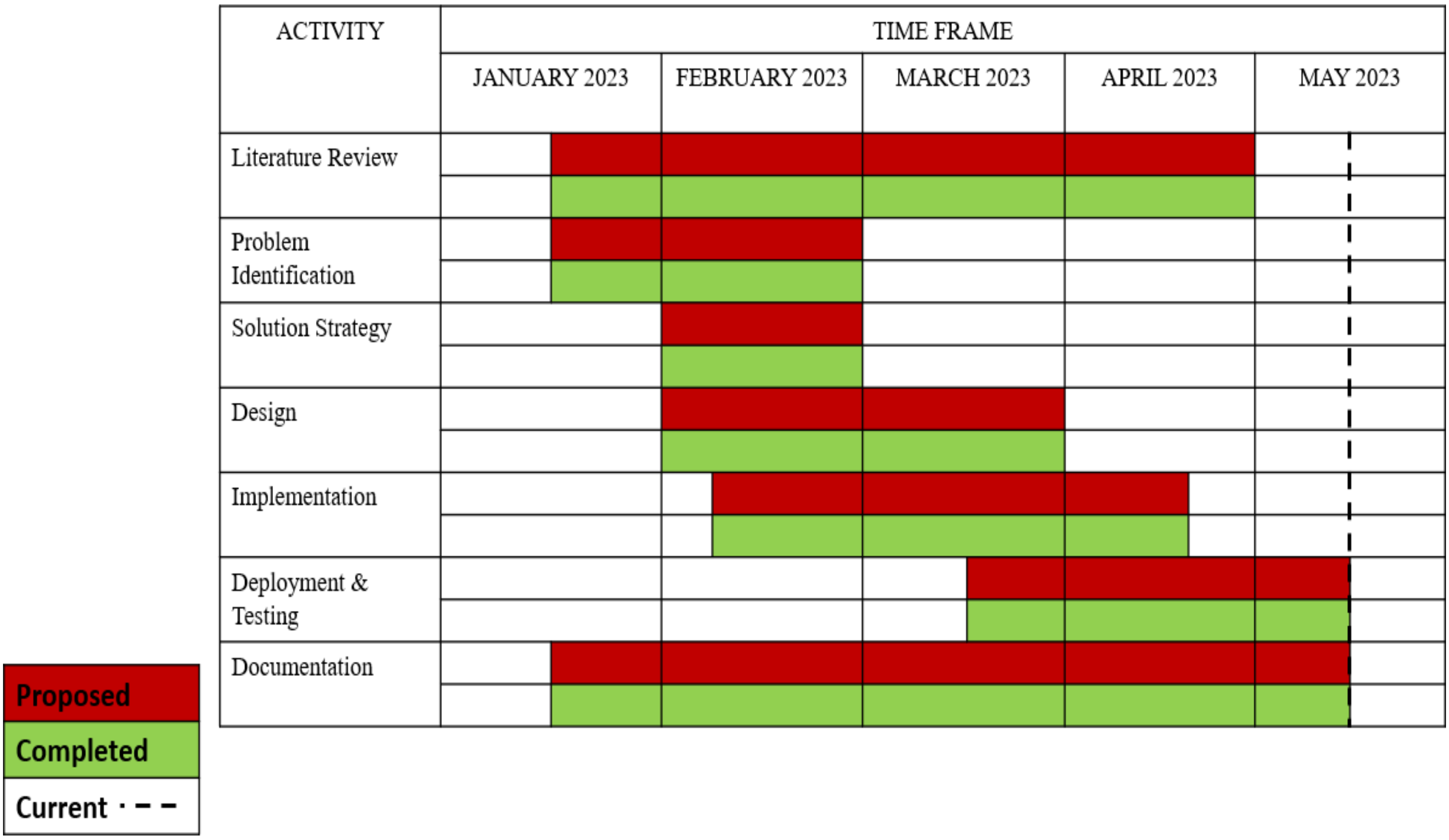

Fig: Gantt Chart for the Application

6. SUMMARY AND CONCLUSION

Summary of Achievements

Developed general APIs and DTCM under WAFT software using FastAPI Framework to automate WebLogic server operations, such as bulk start-up, shutdown, and restart without human intervention

Created Micro Services and RESTful APIs to make a connection to the admin WLST console in a given server, execute Linux commands, and invoke scripts for server operations

Reduced operations time from **24+** hours to a maximum of **4** hours, achieving over **100%** efficiency by implementing asynchronous/concurrent computation to handle bulk servers across multiple environments simultaneously

Ensured proper handling of coroutines to prevent race conditions and deadlocks in asynchronous events

Implemented Blackout Operations to temporarily disengage the Incident Tracking System and prevent false alarms of system failures; logged potential errors during this period and safely re-engaged the tracking system after the process.

Limitations of the Project

The application has been developed keeping in mind that it targets only Oracle WebLogic server structures that have a single admin server. However in case there are multiple admins sharing the same server, then only one of them can be operated on, at a time. So two admins in a given server cannot be handled simultaneously together.

Challenges Faced

The architecture of Oracle WebLogic servers is difficult to surf through and performing operations on them is a tedious and sensitive task. Thus implementation of proper safety mechanisms was challenging and of paramount importance. Further, implementing remote command line operations and handling coroutines were challenging, since performance would break very often and several trial and error operations were used to get them to be perfect and 100% efficient.

Future Scope of Work

The handling of two admins in a server can be implemented in a safe manner. UI design can be modified as per requirements that may arise going forward.

7. REFERENCES

- https://docs.oracle.com/en/middleware/standalone/weblogic-server/
- https://docs.oracle.com/en/middleware/fusion-middleware/12.2.1.4/wlstm/wlst-command-reference.html
- https://www.jython.org/docs/index.html
- https://www.programcreek.com/python/example/10109/javax.script.ScriptEngine
- http://tldp.org/LDP/abs/html/index.html
- https://www.gnu.org/software/bash/manual/
- https://devhints.io/bash
- https://docs.angularjs.org/api
- https://docs.angularjs.org/guide
- https://fastapi.tiangolo.com/advanced/extending-openapi/
- https://fastapi.tiangolo.com/