# Basic Configuration Management Tool

***Project Code - 194***

*Using CRUD Operations*

C – Create
R – Read
U – Update
D – Delete

***Presented by:***
A Rohan Sudhan
M Mahenoor
Priyanka V
K Pranav
Rida Arshad

# Problem Statement:

***Basic configuration management tool***

- **POC CRUD configuration files, deploy_configuration_files(config id): Deploy configuration to the servers.**

- **Track_changes_history(history_details): keep a log of all configuration changes.**

- **Code for the problem statement outlined, we will develop a simple configuration management tool using OOP principle in python.**

- **We will define classes for handling configuration files and for maintaining logs of configuration changes.**

- **Since interaction with actual servers or database is not the part o requirement. operations on data and logs will be simulated using in memory dictionary and tests.**

# INTRODUCTION:

Configuration management is a systems engineering process for establishing consistency of a product's attributes throughout its life.

In the technology world, configuration management is an IT management process that tracks individual configuration items of an IT system.

Specifications of computational hardware resource allocations for CPU, RAM, etc.

Endpoints that specify external connections to other services, databases, or domains

Secrets like passwords and encryption keys

# Skeleton :

```
from datetime import datetime

history = []
configurations = {}

class user: ...

class admin(user): ...

def main(): ...

main()
```

# Inside *class(user):*

```
def create_config(self):
    no_of_attributes = int(input('\nEnter the number of attributes you want\t'))
    for i in range(no_of_attributes):
        key = input('\nEnter the key\t')
        value = input('\nEnter the value\t')
        configurations[key] = value

    print('\nYour current list of dictionary is:')
    currenttime = datetime.now()
    print(configurations)
    history.append(f'Attribute has been created! (Created at {currenttime})')
```

```
def read_config(self):
    while(True):
        print('\nEnter the key you want to read. (Enter END to terminate the operation)\t')
        user_key = input()
        attribute = configurations.get(user_key, 'The input key was not found')
        print(attribute)
        if(user_key == 'END'):
            print('\nLoop has been exited\t')
            break
        currenttime = datetime.now()

    history.append(f'Read operation has been performed!(Read at {currenttime})')
```

# Inside *class admin(user)*

```
def update_config(self):
    while(True):
        print('\nEnter the key to be updated (Type END to kill the process)\t')
        user_key = input()
        if(user_key == 'END'):
            break
        if user_key in configurations:
            print('\nEnter the value to be updated at key\t')
            user_value = input()
            configurations[user_key] = user_value
        else:
            print('\nThe entered key was not found!\t')
            break
    currenttime = datetime.now()

    print('\nYour updated values are: ')
    print(configurations)
    history.append(f'The dictionary has been updated! (Updated at: {currenttime})')
```

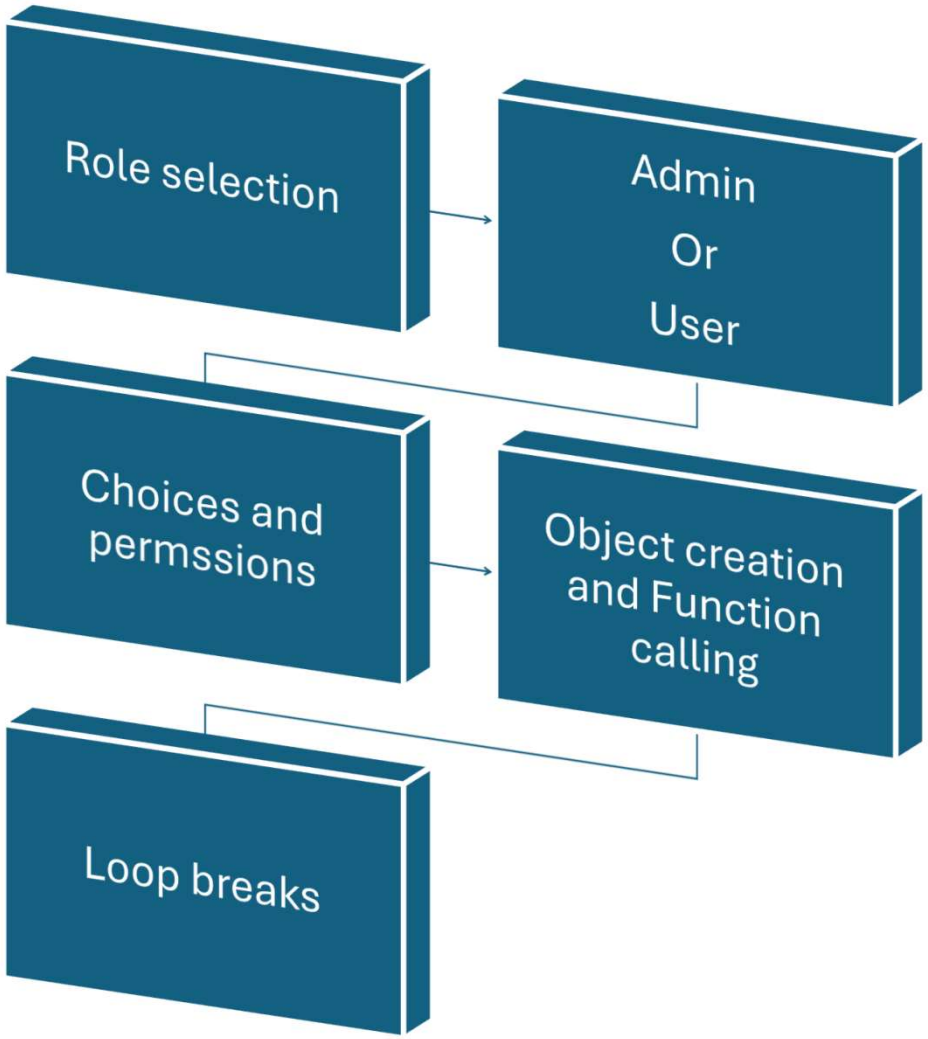
```
def delete_config(self):
    while(True):
        print('\nEnter the key to be deleted from the values (Enter END to terminate the loop)\t')
        user_key = input()
        if(user_key == 'END'):
            break
        if user_key in configurations:
            del configurations[user_key]
            print('\nThe given key is successfully deleted. Your current values are:\t')
            print(configurations)
        else:
            print('\nThe entered key was not found\t')
    currenttime = datetime.now()

    history.append(f'Delete operation has been performed. (Deleted at {currenttime})')
```

```
def history(self):
    print(history)


def view_config(self):
    print(configurations)
    currenttime = datetime.now()


    history.append(f'The dictionary has been viewed. (Viewed at {currenttime})')
```

# Inside *main()*

```
def main():
    while(True):
        print('Enter your role (enter STOP to terminate the loop!): ', end = " ")
        role = input()
        role = role.lower()
        if(role == 'user'):
            userobj = user()
            while(True):
                print('\n ')
                print('''Enter your choice!
                Press 1 for CREATE
                Press 2 to READ
                Press 0 to Terminate loop.''')
                choice = int(input())
                if(choice == 1):
                    userobj.create_config()
                elif(choice == 2):
                    userobj.read_config()
                elif(choice==0):
                    break
                else:
                    print('Enter valid input')
```

```
elif(role=='admin'):
    print('enter the password.')
    password=input()
    if(password=='admin123'):
        adminobj=admin()
        while(True):
            print('\n')
            print('''Enter your choice!
            Press 1 for CREATE
            Press 2 to READ
            Press 3 to UPDATE
            Press 4 to DELETE
            Press 5 to view history
            Press 6 to view the dictionary
            Press 0 to terminate the loop.''')

            choice=int(input('\nChoose an option from the menu\t'))
            if(choice==0):
                break
            elif(choice==1):
                adminobj.create_config()
            elif(choice==2):
                adminobj.read_config()
            elif(choice==3):
                adminobj.update_config()
            elif(choice==4):
                adminobj.delete_config()
            elif(choice==5):
                adminobj.history()
            elif(choice==6):
                adminobj.view_config()
            else:
                print('\nInvalid choice! Please enter a valid input.\t')
    else:
        print('Incorrect password.')
elif(role=='stop'):
        break
else:
    print('Enter the valid role!')
```



# Real Life Scenario:

**Dictionary Structure:**

**1.Configurations Dictionary**:

1. **Key**: Client ID (or Client Name)
2. **Value**: A dictionary containing configuration details like IP addresses, security settings, and database configurations.

**2.Change Log Dictionary**:

1. **Key**: Client ID (or Client Name)
2. **Value**: A list of strings, each string representing a change made to that client's configuration.

***Example Dictionary Representation:***

*Configurations{*
*“firewall_config” : “Allow port 20, Deny port 22” ,*
*“ Server_IP” : “1.0.11.254.123” ,*
*“firewall_status” : “Active” ,*
*}*

# Conclusion:

- Configuration management is a necessary tool for managing complex software systems. Lack of configuration management can cause serious problems with reliability, uptime, and the ability to scale a system.

- Many current software development tools have configuration management features built in.

- Bitbucket offers a powerful system for configuration management that is built around Git pull request workflows and CI/CD pipelines.

# References:


- https://realpython.com/python-gui-tkinter/
- https://youtu.be/VMP1oQOxfM0?si=pa26e_VikeImr2bo
- https://www.coursera.org/learn/python-for-applied-data-science-ai/home/module/3 {coursera coach}