

## Problem #01

### Configuration Problem

In real word, we cannot hard-code configuration values (DB name, passwords, timeout etc).

- Different environment need different configuration.
- For Examples
- Developer uses his local Database To Test the system, so he configures the local database to program.
- Tester may use Data on another server To test the system.
- When software is in production, it uses different data.

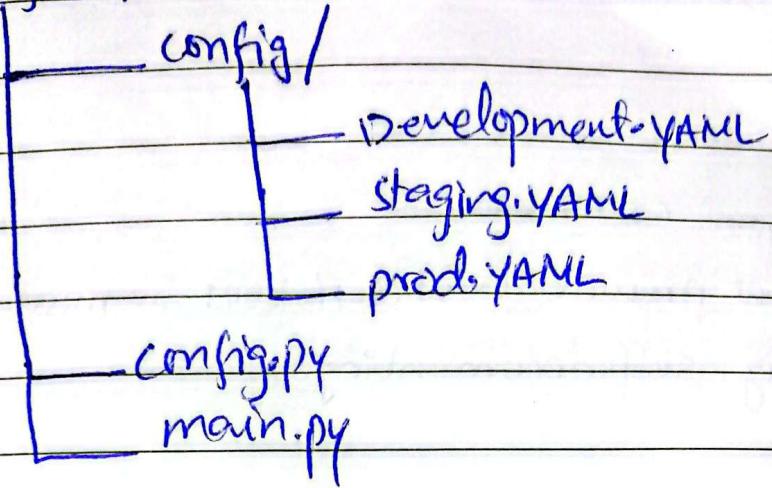
So Hardcoding the configuration Data in Development stage caused the system to crash in testing and production because dB does not exist locally there.

That is why we create single configuration loader that

- Read from YAML Files
- Start config based on environment type
- Map config values into a python object.

## Folder structure

Project /



## YAML FILE STRUCTURE

### Example (Prod.yaml)

db:

DB: "ProductionDB"

TableName: "Users"

Password: "secrets"

timeOut: 30

YAML → automatically parsed into Python dictionaries.

## Config class Implementation

import yaml

class config( ):

DB: str = " "

TableNames: str = " "

Password: str = " "

Timeouts: int = 0

- Acts As typed container
- Default values prevent crashes if keys are missing
- Makes config self-documenting

### load()

def load(self, envType) ↗ Determines which environment (development, staging, prod)

# reading YAML file

with open(f'config/{envType}.yaml', 'r')  
as file:

configData = yaml.safe\_load(file)

→ with open(...) → File auto closer (even on errors)

yaml.safe\_load() → Safe parsing parsing into python  
dict

# extracting only what we need

dbConfig = configData['db']

# Assigning values

self.DB = dbConfig.get("DB", "")

self.TableName = dbConfig.get("TableName", "")

Date

self.Password = dbconfig.get("Password", "")

self.Timeout = dbconfig.get('timeout', 15)

get() prevent keyErrors, Allows Defaults, makes config backward compatible.

return self.

# main.py

from config import config

cfg = config().load("Prod")

print(cfg.DB)

Print(cfg.TableName)

Print(cfg.Password)

Print(cfg.Timeout)