# Use Cases:

## Read YAML Files:

* Application reads multiple YAML Configuration files.
* Primary Actor: Application or User
* Goal: Load configuration files.
* Preconditions: YAML files should be exist, accessible and valid.
* Postconditions: YAML files are successfully read.

## Parse YAML Files:

* Application parses the contents of the YAMLs.
* Primary Actor: Application
* Goal: Get YAML content for manipulating DOM elements.
* Preconditions: YAML files are read by system.
* Postconditions: YAML content is parsed and converted as an object.

## Manipulate DOM Elements:

* Application applies actions to DOM elements.
* Primary Actor: Applicaiton
* Goal: Modify the DOM according to the configuration.
* Preconditions: YAML is successfully parsed.
* Postconditions: DOM is modified according to YAML.

### Remove Element

* + Primary Actor: Application
  + Goal: Remove DOM elements
  + Preconditions: There should be deleting action in configuration.
  + Postconditions: Related elements removed from the DOM.

### Replace Element

* + Primary Actor: Application
  + Goal: Replace elements
  + Preconditions: There should be replacing action in configuration.
  + Postconditions: Related elements replaced to the DOM.

### Insert Element

* + Primary Actor: Application
  + Goal: Insert elements
  + Preconditions: There should be inserting action in configuration.
  + Postconditions: Related elements inserted to the DOM.

### Alter Element

* + Primary Actor: Application
  + Goal: Alter elements
  + Preconditions: There should be altering action in configuration.
  + Postconditions: Related elements altered into the DOM.

## Handle Conflicts

* Apply a logic to minimize the conflicts when there is multiple YAML affect same DOM element.
* Primary Actor: Application
* Goal: Ensure concsistency and there is no conflict between configurations.
* Preconditions: There is multiple configurations effecting same DOM element.
* Postconditions: Conflicts are resolved based on priority logic.

# High-Level Design Document:

## YAML Configuration Parsing:

* Read and parse YAML files
* Handle multiple YAML files

## Action Types:

* Insert, Alter, Replace, Remove

## Multiple Configuration Handling:

* Implement priority logic to resolve conflicts.
* Consistency.

## Basic DOM Manipulation:

* Use JS to manipulate the DOM elements according to YAML configurations.

## Specific Configuration:

* Create a YAML file specifying configurations for specific pages.

## Workflow:

1. Initialization: Application initialized by reading config.yaml file then decide on which YAML files will be read according to current page’s url then related YAML files will be read.
2. Parsing YAML Files: Application parses each YAML file and converts the content into the objects.
3. Resolving Conglicts: Conflicts is checked between actions by their priorty logic.
4. Applying Actions: Actitons will be appliced to DOM elements

## Assumptions and Limitations:

YAML files are correctly formatted.

Basic error handling for invalid YAML formats or unsupported actions.

## Challenges and Solutions:

1. Challenge: Handling conflicts between multiple configurations.

Solution: Implement a priority logic to ensure consistency.

# Low Level Desing:

## fetchConfigFile(configFile):

* config.yaml includes configurations related which yaml file will be used in which url.
* Assumptions:
  + Since it is bonus part, it is assumed that validations are okey.
* Workflow:
  + Application will fetch the config.yaml file and create a JS object.
  + If there is no file named config.yaml raise an error.
  + It will check the current page url then arrange the related yaml files.

## fetchYamlFile(file):

* This function will fetch yaml files and convert them to a JS object.
* Assumptions:
  + There could be 2 different ways to read YAML files. First one is getting file via input tag from user. Second way is getting from same directory. In this project reading from same directory is used.
  + If there is no file named inside of config.yaml raise an error.
* Workflow:
  + Applicaiton will fetch the files and convert them to JS object.
  + If the format of YAML is wrong log error.
  + Send converted JS object to validateYaml(parsedYaml) function to validate the elements of JS object.
  + If validation successfully completed return file name and object.

## validateYaml(parsedYaml):

* This function validates the elements and structure of given yaml file.
* Assumptions:
  + Format of every YAML is fixed. There are 4 types: remove, replace, insert and alter. Also their elements will be fixed. Example formats can be seen inside of the yaml files.
* Workflow:
  + Get an converted yaml to a JS object.
  + There should be an “actions” array.
  + Check the indexes of “actions” array. First elements are should be named as “type” and they should be string. If there is not an element named type and it is not a string error will raise.
  + After that validate each types format.
    - For remove:
      * validateRemoveAction method will be used.
      * It will check if the selector element is presend and is string
    - For replace:
      * validateReplaceAction method will be used.
      * It will check if the selector element and newElement is presend and is string
    - For insert:
      * validateInsertAction method will be used.
      * It will check if the position, target, element items is presend and is string
    - For alter:
      * validateAlterAction method will be used.
      * It will check if the oldValue and newValue element is presend and is string
  + If the formats are okey and there is no error dont raise an error and do nothing. Continue.

## processYamlFiles(yamlFilesData):

* The purpose of this funciton is manipulating DOM elements.
* Assumptions:
  + There can be a different elements inside of types. For example remove type just needs selector element. If selector does not exists there will be an error. However, if there is also an oldValue element which alter is use, there wont be an error. It will be a redundant data and it wont be used for system.
* Workflow:
  + If there is an priority parameter inside of yamls it will sort the files according to their priority level. “1” is highest priority and it will append it first. If there is not a priority parameter inside of a yaml, program implements the first file that read is firstly.
  + After arranging the list of files. Start implementing file by file according to action types.
  + Action types:
    - removeElement():
      * By using the selector get all elements.
      * Remove each element.
    - replaceElement():
      * By using the selector get all elements.
      * Change each element with the new one.
    - insertElement():
      * By using target variable get all elements.
      * Check position variables value.
      * According to position’s value, add a new element before or after the target
    - alterText():
      * Get the all bodyHTML
      * Split the old value and add new ones.

## Suggestions and Questions feedback:

* Using a backend system while fetching can be implemented. While fetching files, file extensions can be validated. Only “.yaml” files can be fetched. Naming convention/structure can be added to files and it can be use as a filter. File size can be checked. If the size is too much to fetch file may not fetched or file can be zipped then fetched then unzip.
* If the yaml files are user supplied, there should be an xss filter to increate the security and prevent xss attacks.