Report

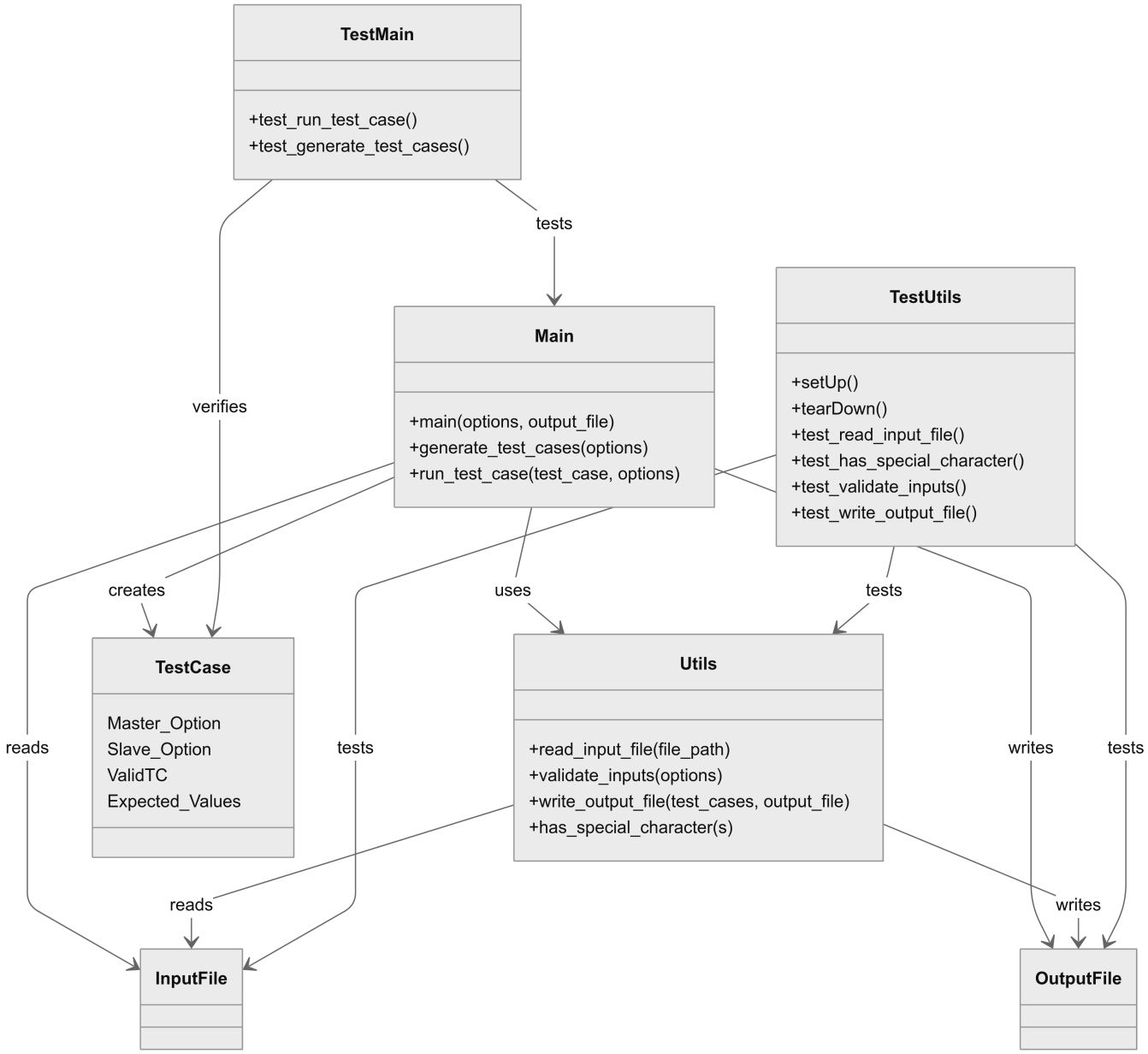
Test Case Generator Documentation

Overview

This Task implements a test case generator that creates combinations of test cases based on master and slave options. The system generates all possible combinations of True/False/NA values for each option pair and validates them according to the specificed rules.

System Design / Approach

The system is in some sort a brute force testing that covers all possible combinations of the input parameters and thier expected results.



Generation of test cases

Based that all server options are might have 1 of 3 values (True, False, NA). and for every server-option we have two desired inputs one for the master-client and one for the slave-client. So the number of test-cases will be calculated as follows:
3^(2*serverOptions)
for the sample input we have 2 server options so we will have 81 test-cases and so on.

Key Components

Main Module (`main.py`)

```
run_test_case(test_case: dict, options) -> None
```

- **Purpose**: Validates and processes individual test cases
- **Logic**:
  - Checks if a test case is valid based on specified constrains
  - A test case is invalid if:
    - Master option is "NA" and slave option is "False"
    - Master and slave options have different values (when master is not "NA")
  - Sets expected values for each option
  - If test case is valid, expected values match master options and NA option is replaced with the default value which i assumed to be true across all the options.

`generate_test_cases(options: list[str]) -> list`

- **Purpose**: Generates all possible test case combinations
- **Logic**:
  - Creates combinations of "True", "False", "NA" for all master and slave options
  - Uses `itertools.product` to generate all possible combinations
  - Processes each combination through `run_test_case`
  - Returns list of all generated test cases with validation results

`main(options: list, output_file = "./output.csv") -> None`

- **Purpose**: Main entry point for the application
- **Features**:
  - Validates input options
  - Generates test cases
  - Writes results to CSV file
  - Handles errors

Utilities Module (`utils.py`)

`read_input_file(file_path: str) -> list[str]`

- **Purpose**: Reads options from input file
- **Features**:
  - Reads one option per line
  - Skips empty lines

`validate_inputs(options: list) -> None`

- **Purpose**: Validates input options

- **Validation Rules**:
  - No empty options list
  - No duplicate options
  - No special characters in option names

`write_output_file(test_cases: list[dict], output_file) -> None`

- **Purpose**: Writes test cases to CSV file
- **Features**:
  - Adds Test Case ID column
  - Writes all test case data in CSV format

File Structure

```
project_root/
│
│
│
├── main.py
├── utils.py
├── test_all.py
├── input.txt
├── putput.csv
│
├── tests/
│   ├── __init__.py
│   ├── test_main.py
│   └── test_utils.py
│
├── test/
│   ├── inputs/
│   └── outputs/
│
├── docs/
│   └── report.pdf
│   └── sample_output.csv
│
└── README.md
```

Assumptions

- I assumed that if the test case is invalid, the expected output will be NA for all the expected values.
  this is decided from this test case in the document
  `2 TRUE FALSE NA TRUE NO NA NA`
- I assumed that if master is NA for some option it is equivalent to True when comparing to none NA value for the slave ,(The default is assumed to be true).

Testing

Unit Tests Coverage

- Input validation (`utils.py`)
  - Empty input check
  - Duplicate options check
  - Special characters check
- test case generation logic (`main.py`)
  - Test case generation
  - Test case execution
  - Expected value calculation
- File I/O operations (`utils.py`)
  - Reading input files
  - Writing CSV output files

Run Small Set of Tests

```
python test_all.py
```

- the above script runs some selected test cases specifed in the `./test/inputs` folder and generate the corresponding outputs in `./test/outpt`

the test cases:

1. duplicate_test.txt : test for duplicate server options

- input:

```
    BufferData
```

```
BufferData
```

- output:
  `Repeated options name is invalid.`

2. no_test.txt :

- input:
  empty file
- output:
  `Empty options is invalid.`

2. special_char_test.txt : for invalid input

- input:

```
BufferDat$$
TimeOut
```

- output:
  `Option 'BufferDat$$' contains special characters, which is invalid.`

4. one_test.txt
5. two_test.txt
6. three_test.txt
   and outputs to 4,5,6 are save as files in `./test/outputs`