

Load Testing and Performance Evaluation of the Theia Online IDE

Bachelor Thesis Intermediate Presentation

Author: Tobias Klingenberg

Supervisor: Prof. Dr. Stephan Krusche

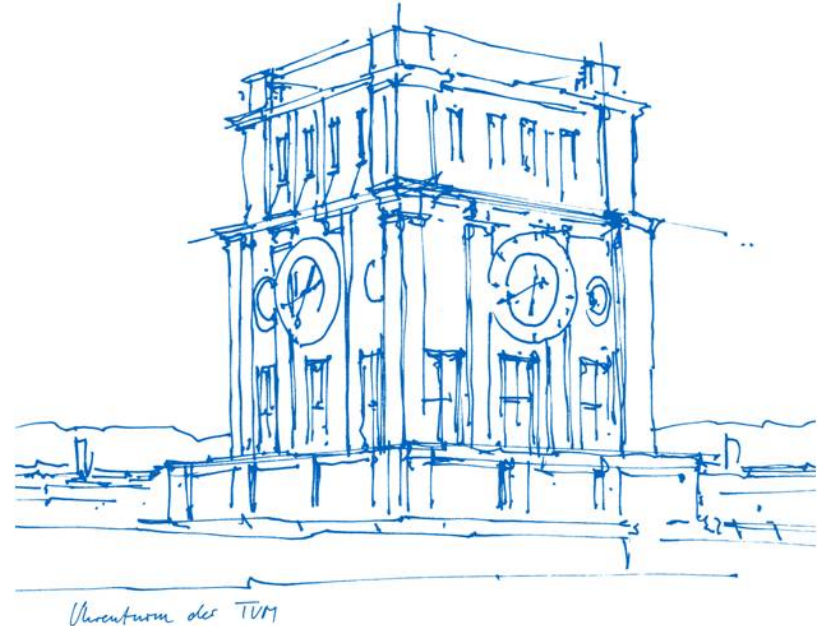
Advisor: Matthias Linhuber

Technische Universität München

TUM School of Computation, Information and Technology

Group for Applied Education Technologies

Garching, 23. October 2025



Motivation

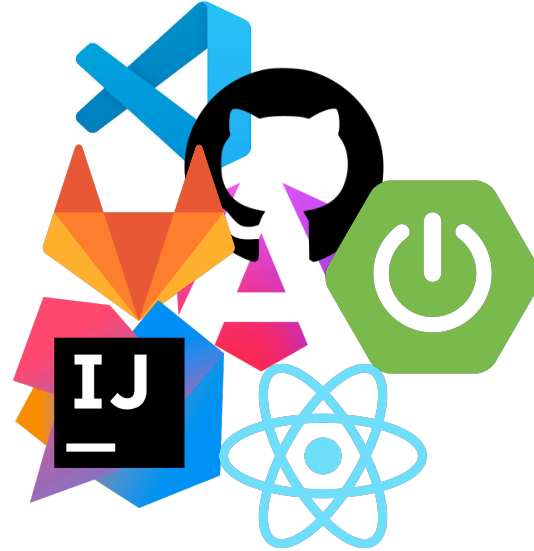


Steve

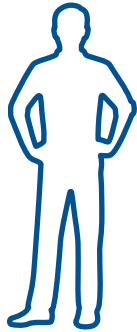
Motivation



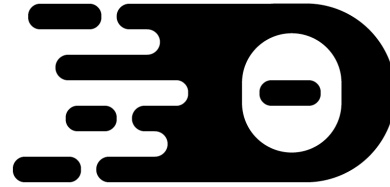
Steve



Motivation



Steve



Theia

Motivation



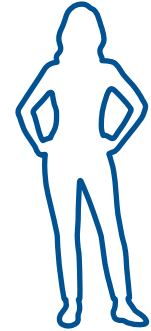
Steve
Student



Hector
Instructor

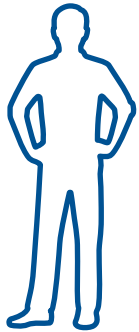


Jennifer
Developer



Edwin
Admin

Problem



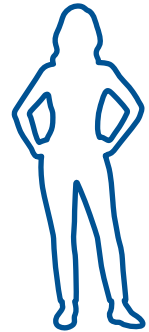
Steve



Rich Interactive
User Interface

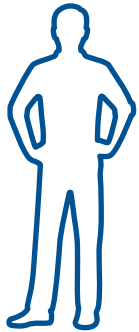


Scalable
Infrastructure



Edwin

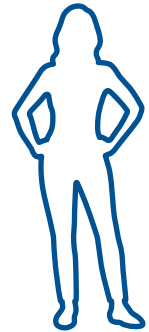
Problem



Steve



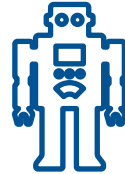
Not currently tested



Edwin

Objectives

1. Developing a **Testing Suite** that tests the usability and **functional correctness** of Theia Cloud
2. Extending the Test Suite with a **Load Testing** framework that allows developers to evaluate the performance of .
3. Creating a **MCP Testing** script that allows a LLM to interact with Theia Cloud for testing purposes




Goals

: Functional Tests 

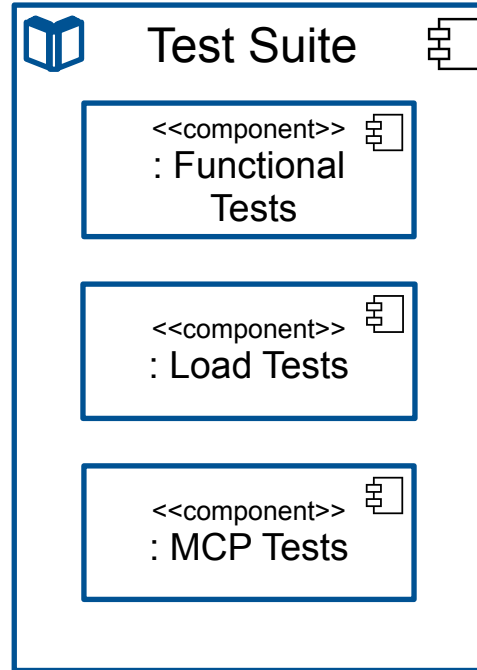
Testing **functionality**
using E2E Tests

: Load Tests 

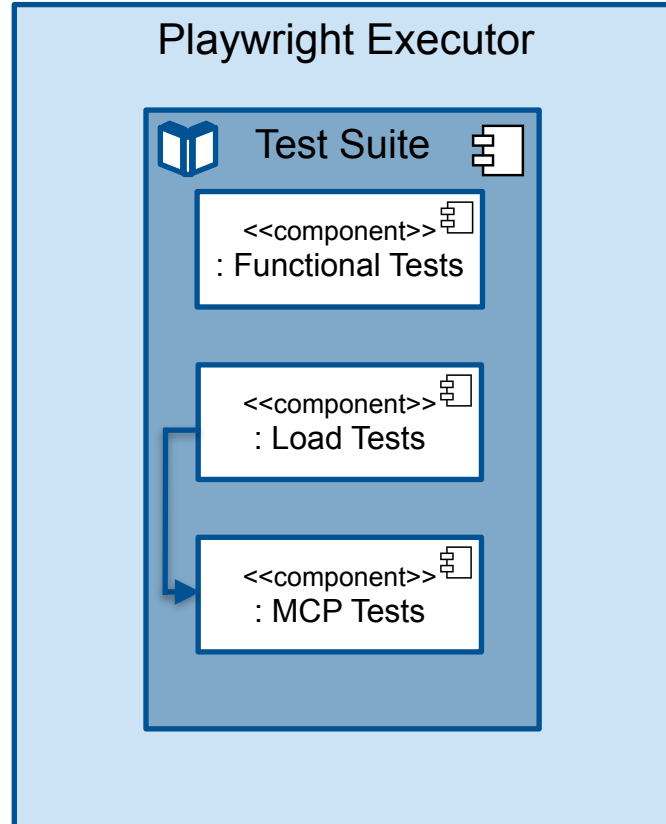
Testing **scalability**
under load

: MCP Tests 

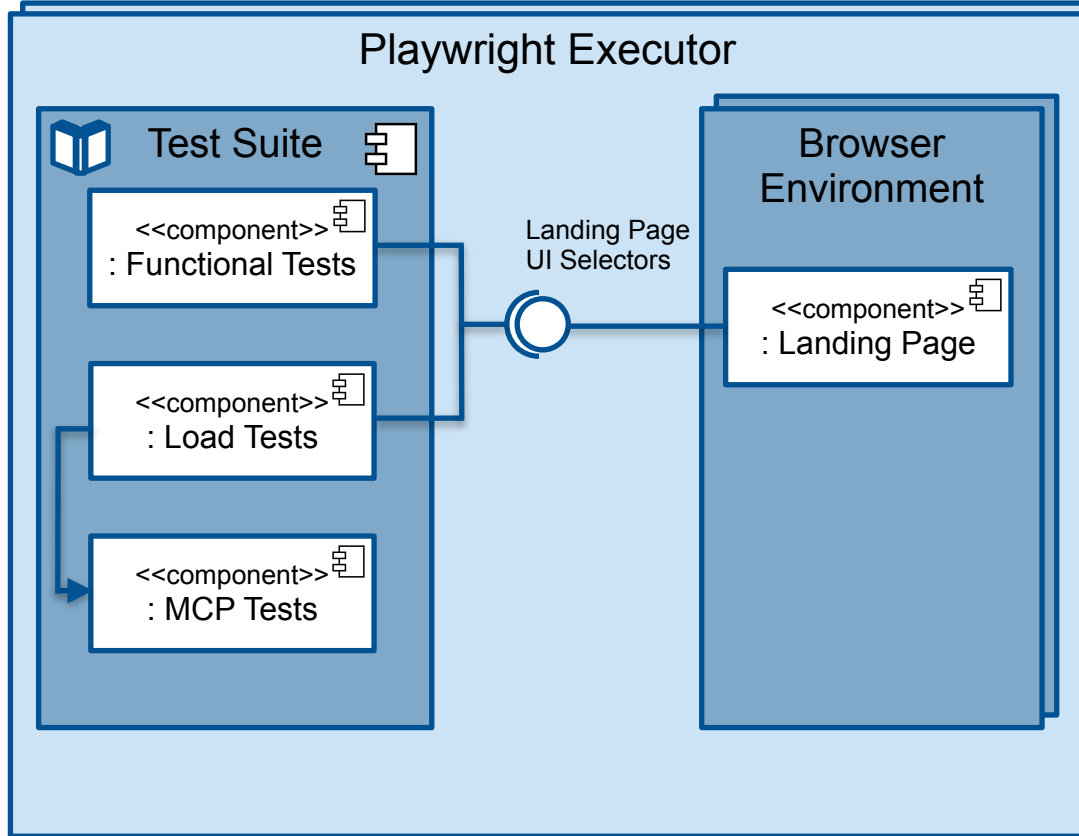
PoF: Testing using
LLMs



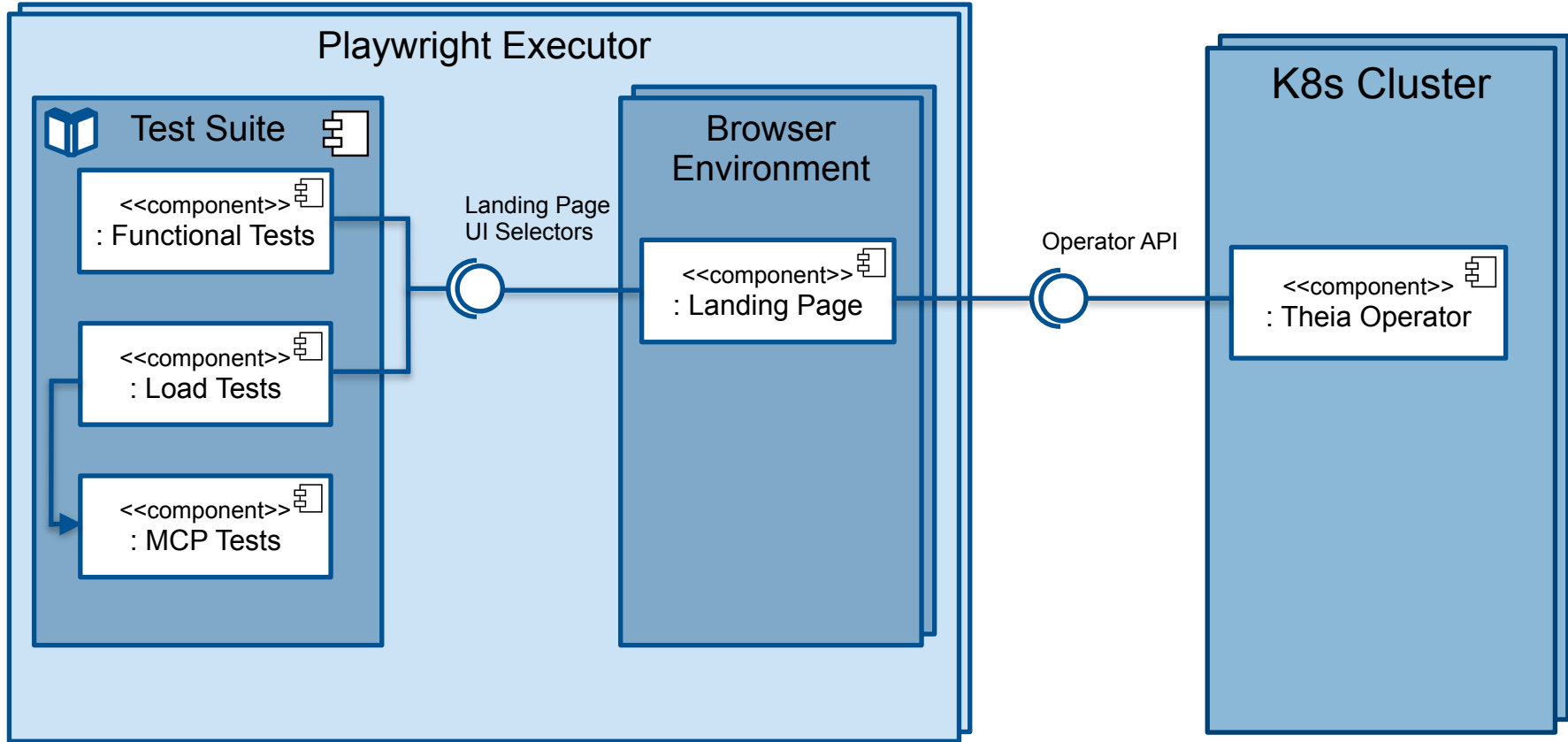
Subsystem Decomposition



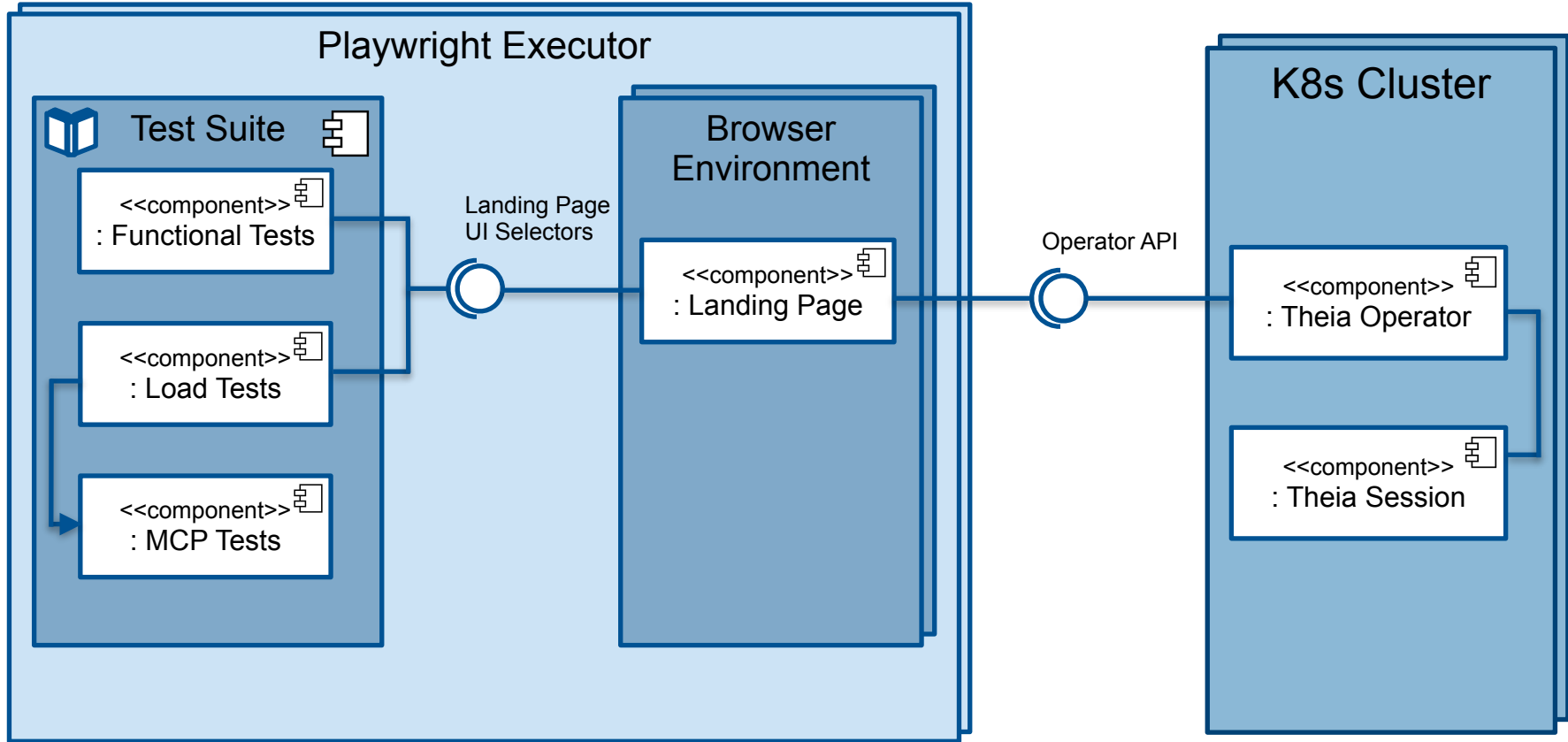
Subsystem Decomposition



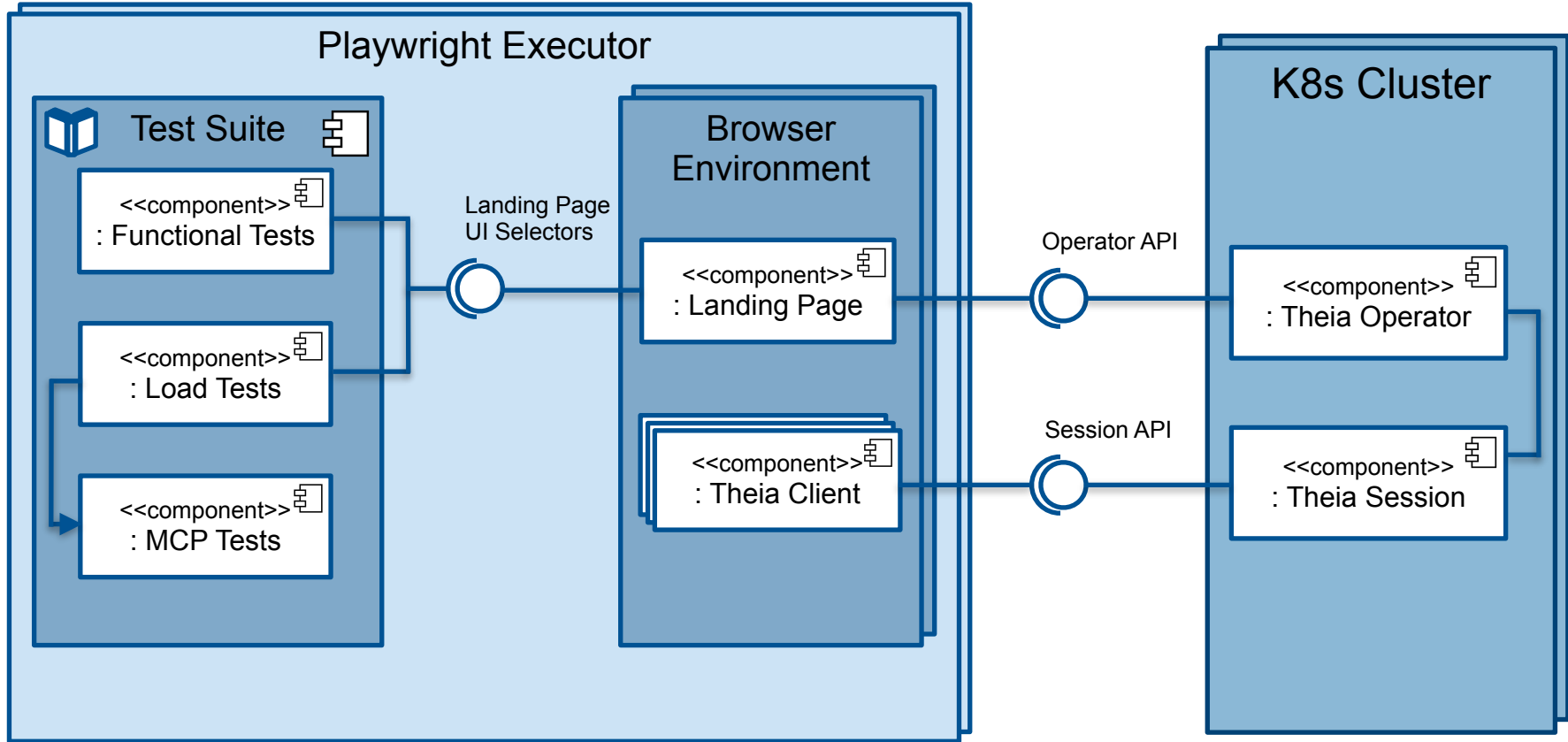
Subsystem Decomposition



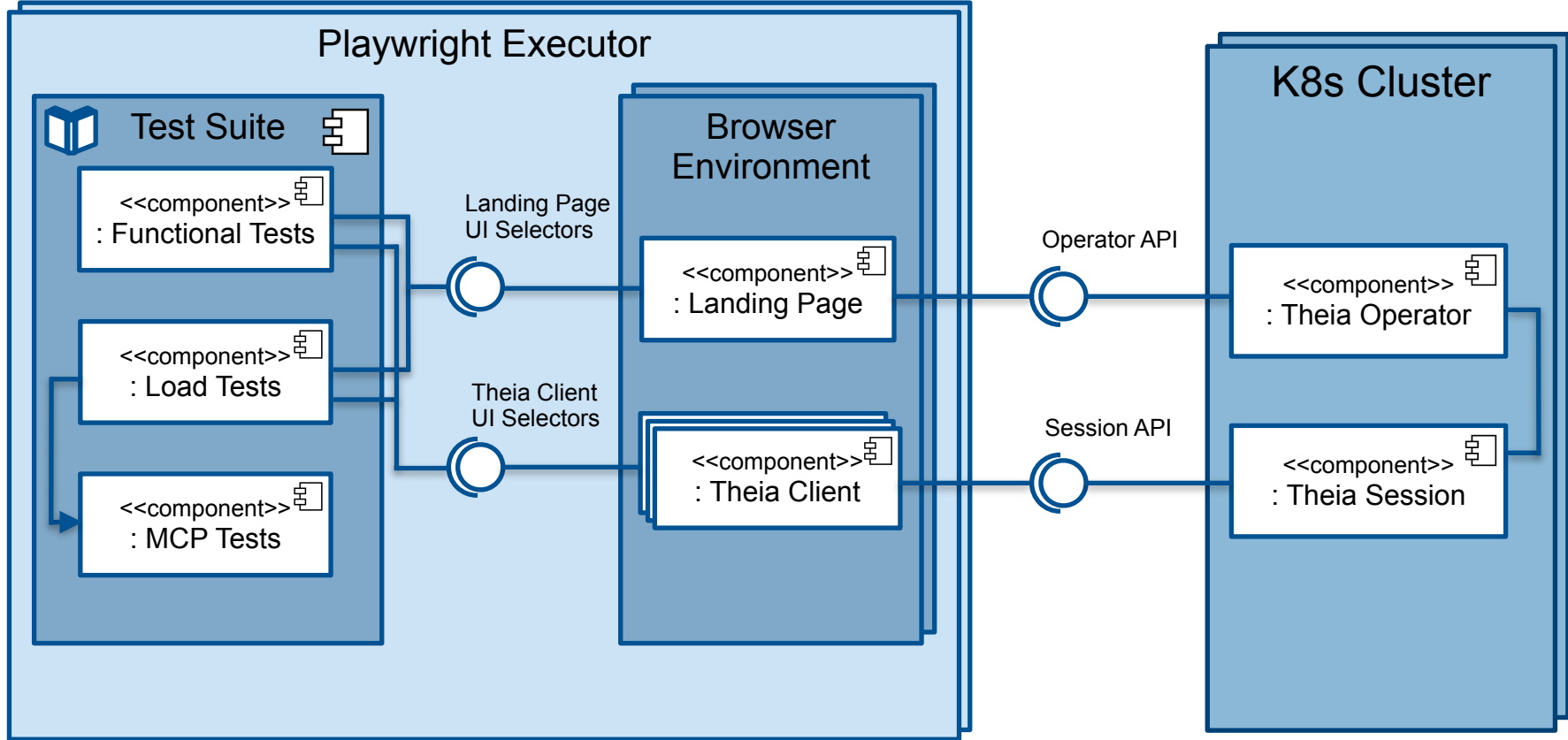
Subsystem Decomposition



Subsystem Decomposition



Subsystem Decomposition



Functionality Tests



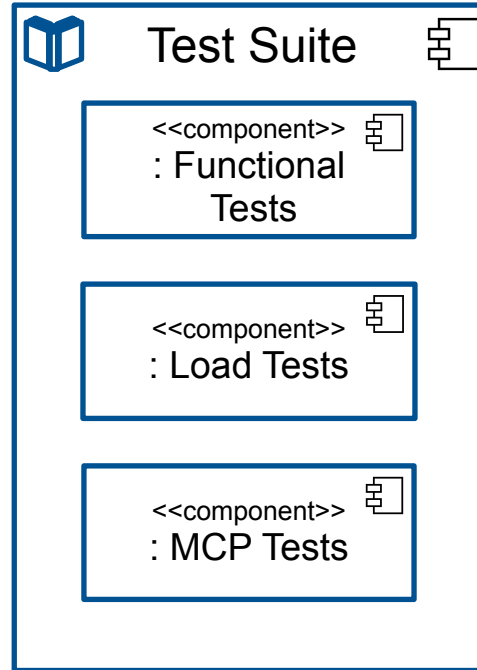
Functionality Tests



Functionality Tests



Functional Tests



E2E Tests with Playwright

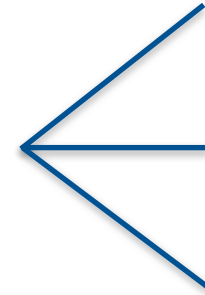


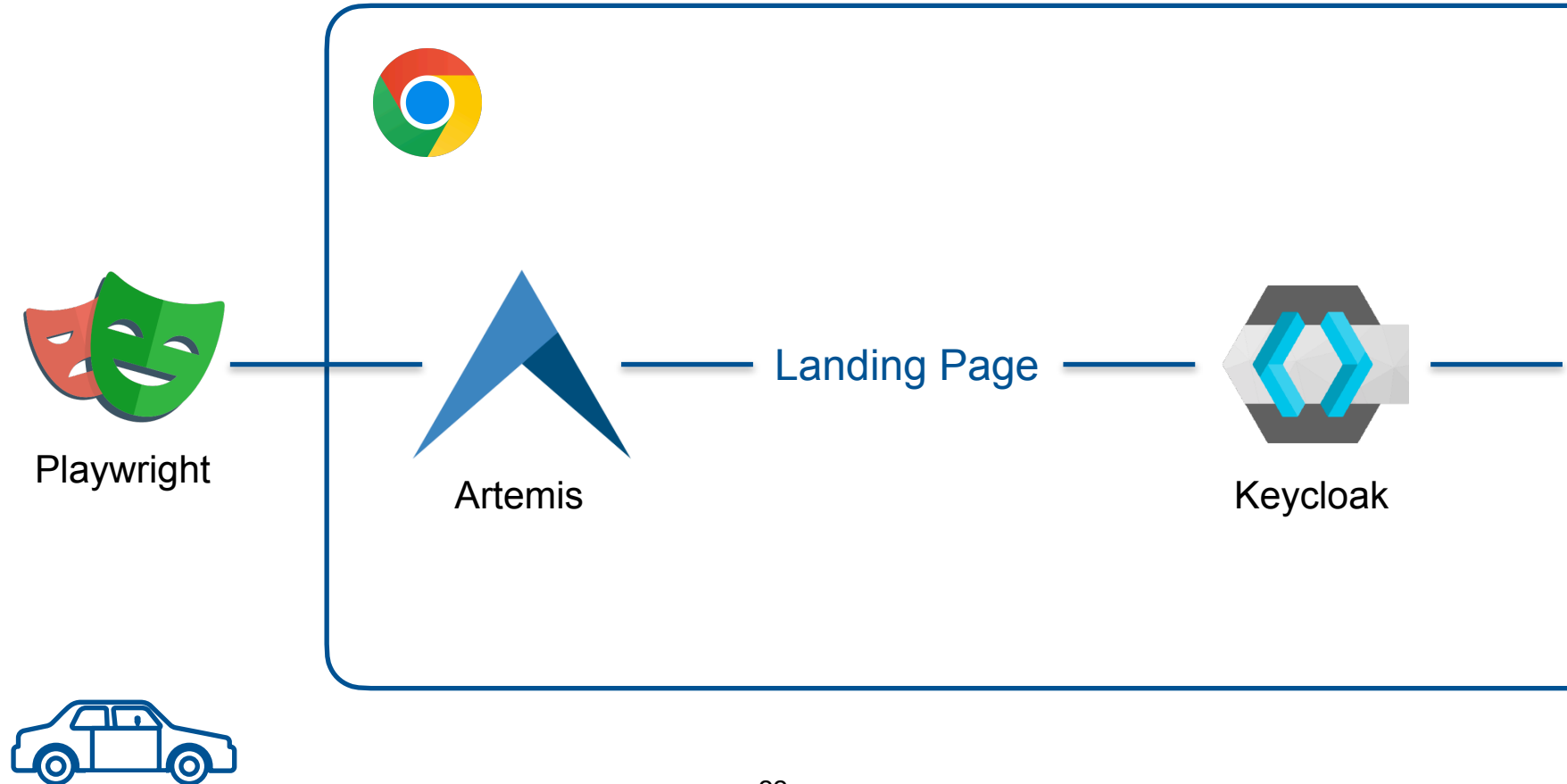
Test Suite

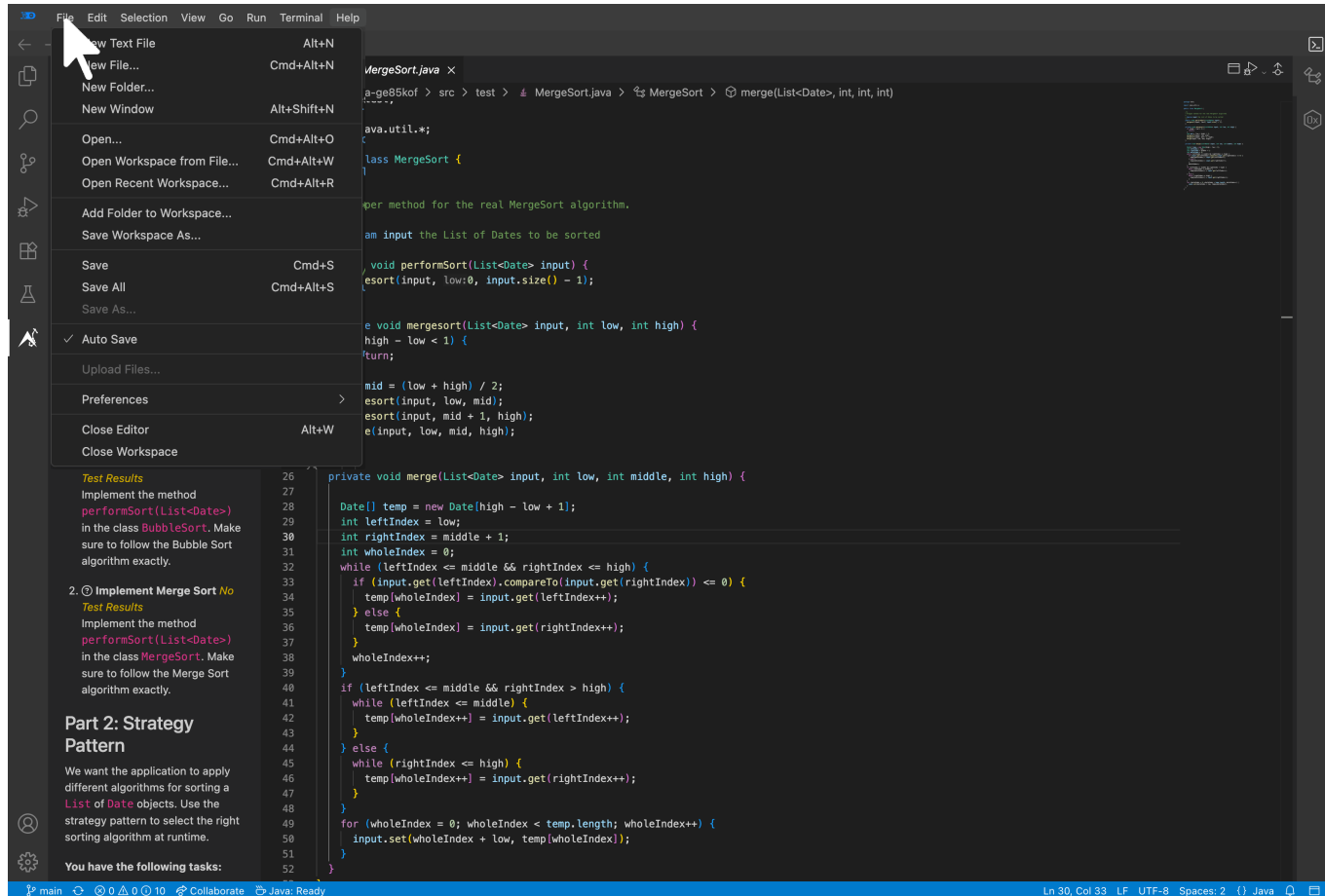
Websockets



Playwright







The screenshot shows an IDE with a dark theme. The File menu is open, showing options like New Text File, Open..., Save, and Auto Save. The main editor displays the MergeSort.java file with the following code:

```

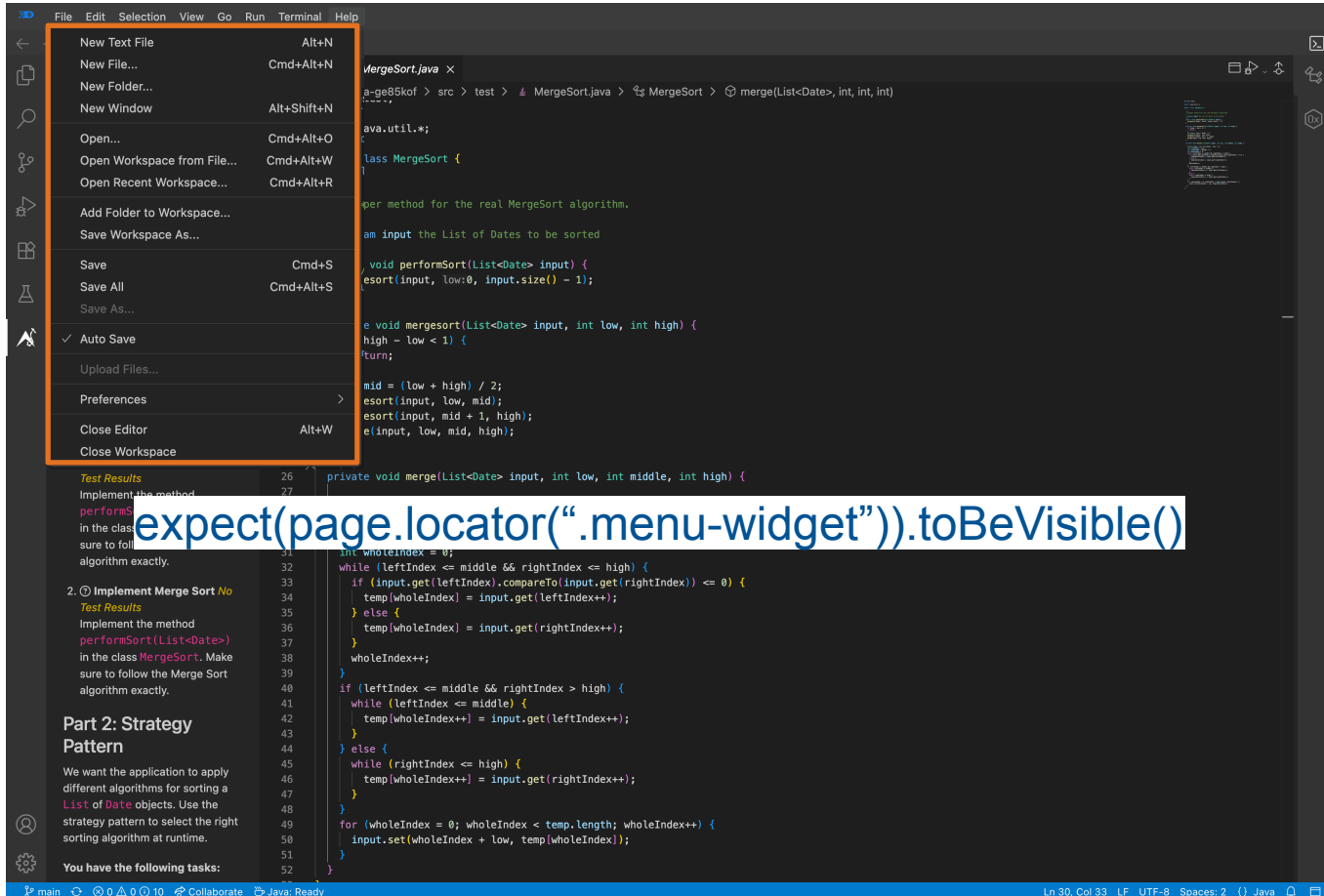
MergeSort.java
a-ge85kof > src > test > MergeSort.java > MergeSort > merge(List<Date>, int, int)
...
ava.util.*;
class MergeSort {
    // ...
    // per method for the real MergeSort algorithm.
    // an input the List of Dates to be sorted
    void performSort(List<Date> input) {
        esort(input, low:0, input.size() - 1);
    }
    void mergesort(List<Date> input, int low, int high) {
        high - low < 1) {
            turn;
        }
        mid = (low + high) / 2;
        esort(input, low, mid);
        esort(input, mid + 1, high);
        e(input, low, mid, high);
    }
    private void merge(List<Date> input, int low, int middle, int high) {
        Date[] temp = new Date[high - low + 1];
        int leftIndex = low;
        int rightIndex = middle + 1;
        int wholeIndex = 0;
        while (leftIndex <= middle && rightIndex <= high) {
            if (input.get(leftIndex).compareTo(input.get(rightIndex)) <= 0) {
                temp[wholeIndex] = input.get(leftIndex++);
            } else {
                temp[wholeIndex] = input.get(rightIndex++);
            }
            wholeIndex++;
        }
        if (leftIndex <= middle && rightIndex > high) {
            while (leftIndex <= middle) {
                temp[wholeIndex++] = input.get(leftIndex++);
            }
        } else {
            while (rightIndex <= high) {
                temp[wholeIndex++] = input.get(rightIndex++);
            }
        }
        for (wholeIndex = 0; wholeIndex < temp.length; wholeIndex++) {
            input.set(wholeIndex + low, temp[wholeIndex]);
        }
    }
}

```

The left sidebar shows a task list for "Part 2: Strategy Pattern". The tasks are:

- Test Results
- Implement the method `performSort(List<Date>)` in the class `BubbleSort`. Make sure to follow the Bubble Sort algorithm exactly.
- 2. Implement Merge Sort **No** Test Results
- Implement the method `performSort(List<Date>)` in the class `MergeSort`. Make sure to follow the Merge Sort algorithm exactly.

The bottom status bar shows the current file is `Java: Ready` and the cursor is at `Ln 30, Col 33`.



The screenshot shows an IDE window with a menu open over the `MergeSort.java` file. The menu is highlighted with an orange border. The text overlay indicates the expected behavior for the menu widget.

Menu Options:

- New Text File (ALT+N)
- New File... (Cmd+ALT+N)
- New Folder...
- New Window (ALT+SHIFT+N)
- Open... (Cmd+ALT+O)
- Open Workspace from File... (Cmd+ALT+W)
- Open Recent Workspace... (Cmd+ALT+R)
- Add Folder to Workspace...
- Save Workspace As...
- Save (Cmd+S)
- Save All (Cmd+ALT+S)
- Save As...
- ✓ Auto Save
- Upload Files...
- Preferences
- Close Editor (ALT+W)
- Close Workspace

Code Snippet (MergeSort.java):

```

1  package a.ge85kof > src > test > MergeSort.java > MergeSort > merge(List<Date>, int, int)
2  ...
3  java.util.*;
4  ...
5  class MergeSort {
6  ...
7  per method for the real MergeSort algorithm.
8  ...
9  an input the List of Dates to be sorted
10 ...
11 void performSort(List<Date> input) {
12     esort(input, low:0, input.size() - 1);
13 }
14 ...
15 e void mergesort(List<Date> input, int low, int high) {
16     high - low < 1) {
17         'turn;
18     }
19     mid = (low + high) / 2;
20     esort(input, low, mid);
21     esort(input, mid + 1, high);
22     e(input, low, mid, high);
23 }
24 ...
25 private void merge(List<Date> input, int low, int middle, int high) {
26     ...
27     while (leftIndex <= middle && rightIndex <= high) {
28         if (input.get(leftIndex).compareTo(input.get(rightIndex)) <= 0) {
29             temp[wholeIndex] = input.get(leftIndex++);
30         } else {
31             temp[wholeIndex] = input.get(rightIndex++);
32         }
33         wholeIndex++;
34     }
35     if (leftIndex <= middle && rightIndex > high) {
36         while (leftIndex <= middle) {
37             temp[wholeIndex++] = input.get(leftIndex++);
38         }
39     } else {
40         while (rightIndex <= high) {
41             temp[wholeIndex++] = input.get(rightIndex++);
42         }
43     }
44     for (wholeIndex = 0; wholeIndex < temp.length; wholeIndex++) {
45         input.set(wholeIndex + low, temp[wholeIndex]);
46     }
47 }
48 ...
49 ...
50 ...
51 ...
52 ...

```

Test Results:

Implement the method `performSort(List<Date>)` in the class `MergeSort`. Make sure to follow the Merge Sort algorithm exactly.

2. Implement Merge Sort

Implement the method `performSort(List<Date>)` in the class `MergeSort`. Make sure to follow the Merge Sort algorithm exactly.

Part 2: Strategy Pattern

We want the application to apply different algorithms for sorting a `List of Date` objects. Use the strategy pattern to select the right sorting algorithm at runtime.

You have the following tasks:

expect(page.locator(".menu-widget")).toBeVisible()

Functionality in Test



Editor
8 Tests



Search
5 Tests



Terminal
7 Tests



VCS
4 Tests



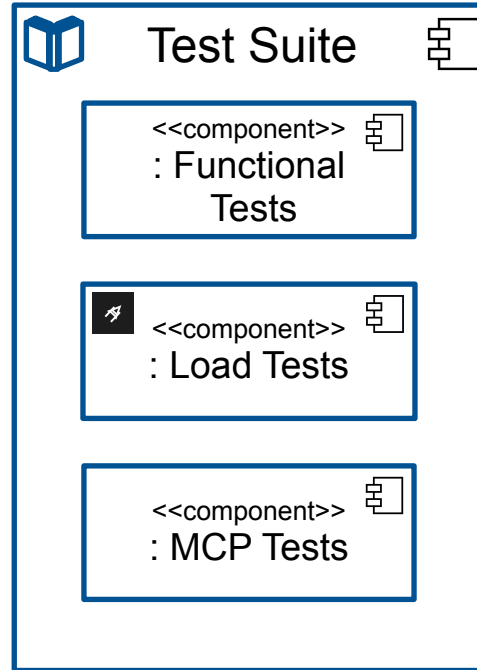
Programming Language
24 Tests



Artemis Integration
1 Test



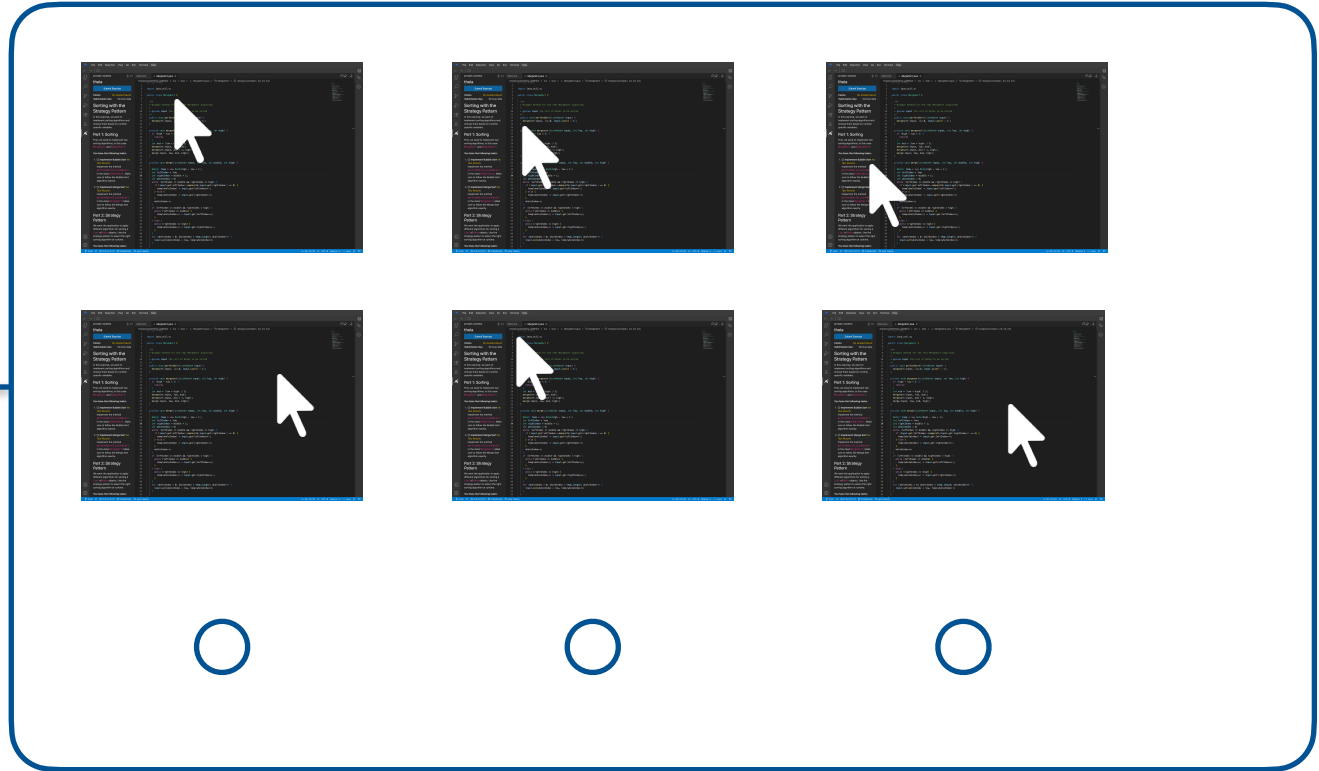
Load Testing

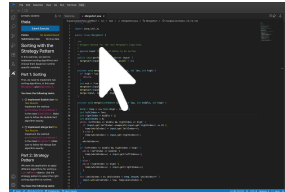
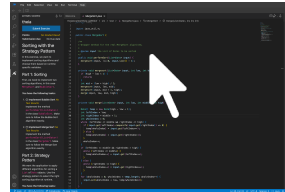
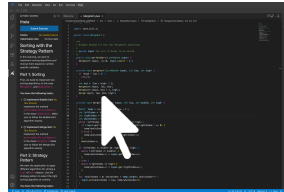
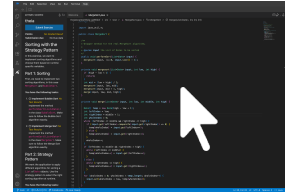
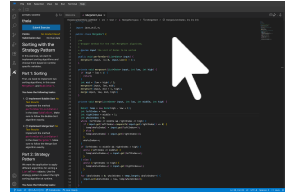
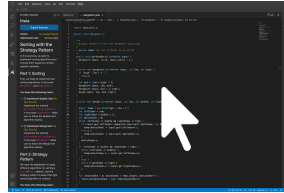




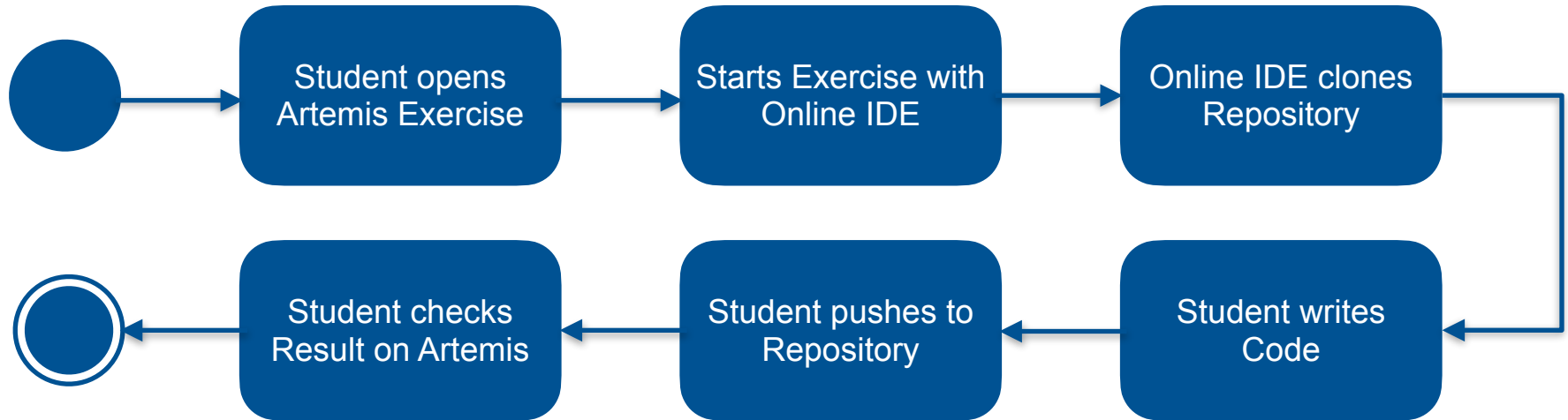
Artillery.io







Example Workflow





Steve



Static Tests

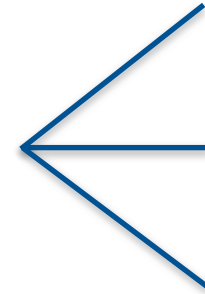
LLM Testing



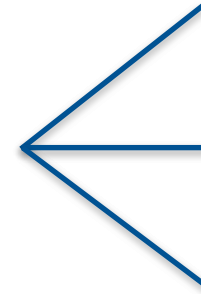
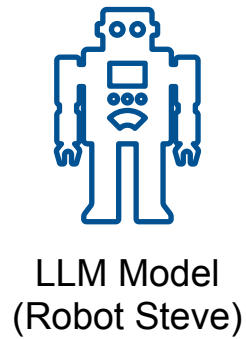
Test Suite
(Steve)



Playwright



LLM Testing

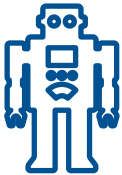




1. The functional **Test Suite** is completed and can run against the test infrastructure



2. The **Load Testing** framework can successfully spawn an arbitrary amount of instances



3. The **MCP Testing** script is in a functional state but requires further research in the topics of automation



DEMO



Questions

