

MAYAN

Automaton construction for realistic website load testing

Jared o’reilly

entelect

2018

Table of Contents

[Introduction 2](#_Toc517946907)

[Plan 3](#_Toc517946908)

[User Cases 3](#_Toc517946909)

[Flow Scenarios/Options 3](#_Toc517946910)

[Critical Flow/Path 4](#_Toc517946911)

[Durations 4](#_Toc517946912)

[User Interface 5](#_Toc517946913)

[Interface Screen 5](#_Toc517946914)

[Artillery Generation 7](#_Toc517946915)

[Artillery Scenarios 7](#_Toc517946916)

[1 User Case to Scenario Conversion Technique 7](#_Toc517946917)

[2+ User Cases to Scenarios Conversion Technique 8](#_Toc517946918)

[References 9](#_Toc517946919)

# Introduction

According to Wikipedia, **load testing** is performed on a system to “determine a system's behaviour under both normal and anticipated peak load conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation.”

The Artillery toolkit, which was obtained and installed from:

[*https://artillery.io/*](https://artillery.io/)

was selected to perform the load testing actions on the Trans Atlantic Gem Sales (TAGS) diamond auction UAT website hosted by Entelect, located at:

[*http://tagsuatappservice.azurewebsites.net/Account/LogOn?ReturnUrl=%2F*](http://tagsuatappservice.azurewebsites.net/Account/LogOn?ReturnUrl=%2F)

However, to perform sufficient and effective load testing, the actions of users on the website need to be realistically replicated in the *scenarios* functionality of Artillery. The path that a user follows (the actual order of links that they click on) is hard to generate, but this is required if efficient load testing is to be done. Replicating real life users is a complex task.

The Mayan software was created to realistically replicate the actions that a user may perform on a website, in the order they are most likely to be performed in. This is done by constructing a non-deterministic finite automaton (also known as an NFA, which is basically an example of a computer science graph i.e. vertices and edges), which is set up by the user (with the help of a UI), and represents the **web pages/requests** for web pages by users as **vertices**, and the links between the pages (caused by the user applying actions onto the webpages) as the edges.

The Mayan software uses the graph set up by the user, along with *probabilities* of each action that the user may take on a certain webpage, to simulate the probable flows of users across the website. This simulation is then converted into *scenarios* in Artillery, and a total Artillery script is generated by the Mayan software (from the graph set up by the user). See below for the summary:

A website with links

↓ Mayan UI ↓

A graph representing the web pages/requests and links

↓ Mayan execution ↓

Artillery scripts to run

↓ Artillery scripts run with command *artillery run ma.txt* ↓

Load testing results summary from Artillery

Some details:

Creation Date: 27/06/2018

Author: Jared O’Reilly

Position: Intern at Entelect

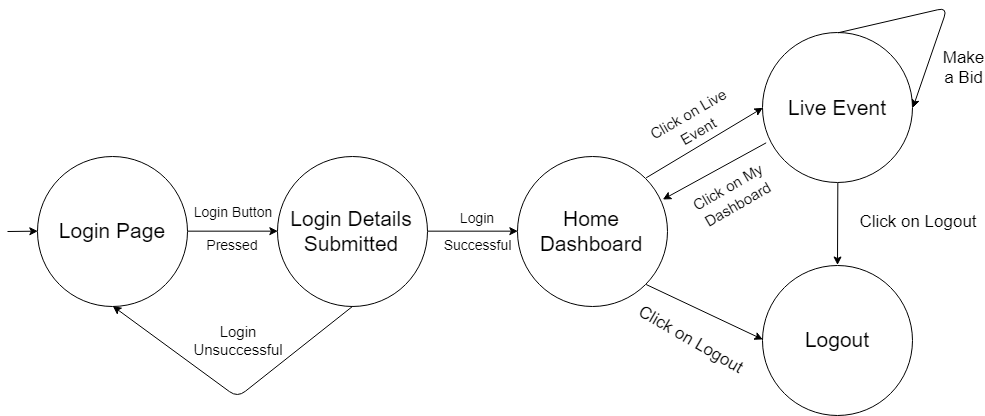
# Plan

## User Cases

The *user cases* are the possible options the user could pick for their actions as they use the website. They are also known as *flow scenarios*, as they are the scenarios that take place as a user ‘flows’ through the website, i.e. as they navigate through it. Artillery describes them as flow scenarios, and this is also included in the syntax of their toolkit and scripts. They will be described in this section in the shoes/*perspective of an eDiamond user*.

### Flow Scenarios/Options

The question that should be asked is: what are the possible flows of a user through the website? Because the real website is very complex, with many different web pages and activities, to start of our flow scenarios/options, we will use the following very simple graph of web pages/requests as vertices, and actions/links as edges, from the <http://tagsuatappservice.azurewebsites.net> website:



This is a very simple graph representation of web pages and the actions needed to traverse between them, from an eDiamond user perspective. If we follow along the directions of the arrows, we create a path, and this is known as a flow. We can see that there are many different paths possible in this simplified diagram, for example:

* **(Login Page)** > *Login Button Pressed* > **(Login Details Submitted)** > *Login Successful* > **(Home Dashboard)** > *Click on Logout* > **(Logout)**
* **(Login Page)** > *Login Button Pressed* > **(Login Details Submitted)** > *Login Unsuccessful* > **(Login Page)** > *Login Button Pressed* > **(Login Details Submitted)** > *Login Successful* > **(Home Dashboard)** > *Click on Logout* > **(Logout)**
* **(Login Page)** > *Login Button Pressed* > **(Login Details Submitted)** > *Login Successful* > **(Home Dashboard)** > *Click on My Dashboard* > **(Live Event)**
* **(Login Page)** > *Login Button Pressed* > **(Login Details Submitted)** > *Login Successful* > **(Home Dashboard)** > *Click on Live Event* > **(Live Event)** > *Make a Bid* > **(Live Event)** > *Make a Bid* > **(Live Event)** > *Click on My Dashboard* > **(Home Dashboard)** > *Click on Logout* > **(Logout)**

As we can see, there are many different options/cases/possibilities for the flow of a user through the website. To try and iteratively generate all the possible flows would be an extremely complicated and confusing task, also, there are infinite possible flows. The graph allows scenarios to be randomly generated, taking paths according to probabilities, and simplifies the flow generation.

### Critical Flow/Path

Even with all the possible flow paths, the chance that a certain flow path occurs can be very low. Based on how users actually flow through the website, it will be seen that certain flow paths are very common. However, there will be one flow path that will be taken the most out of all them, which is known as the *critical flow/path*. Based on my estimation, the primary actions of an online auction site will be to login successfully, go to an active event, make a bid, wait between possibly bidding more, and then will go back to the home page and exit (they also might logout). So, the critical flow would be:

**(Login Page)** > *Login Button Pressed* > **(Login Details Submitted)** > *Login Successful* > **(Home Dashboard)** > *Click on Live Event*> **(Live Event)** > *Make a Bid* > **(Live Event)** > Wait > **(Live Event)** > *Make a Bid* > **(Live Event)** > *Click on My Dashboard* > **(Home Dashboard)** > Exit

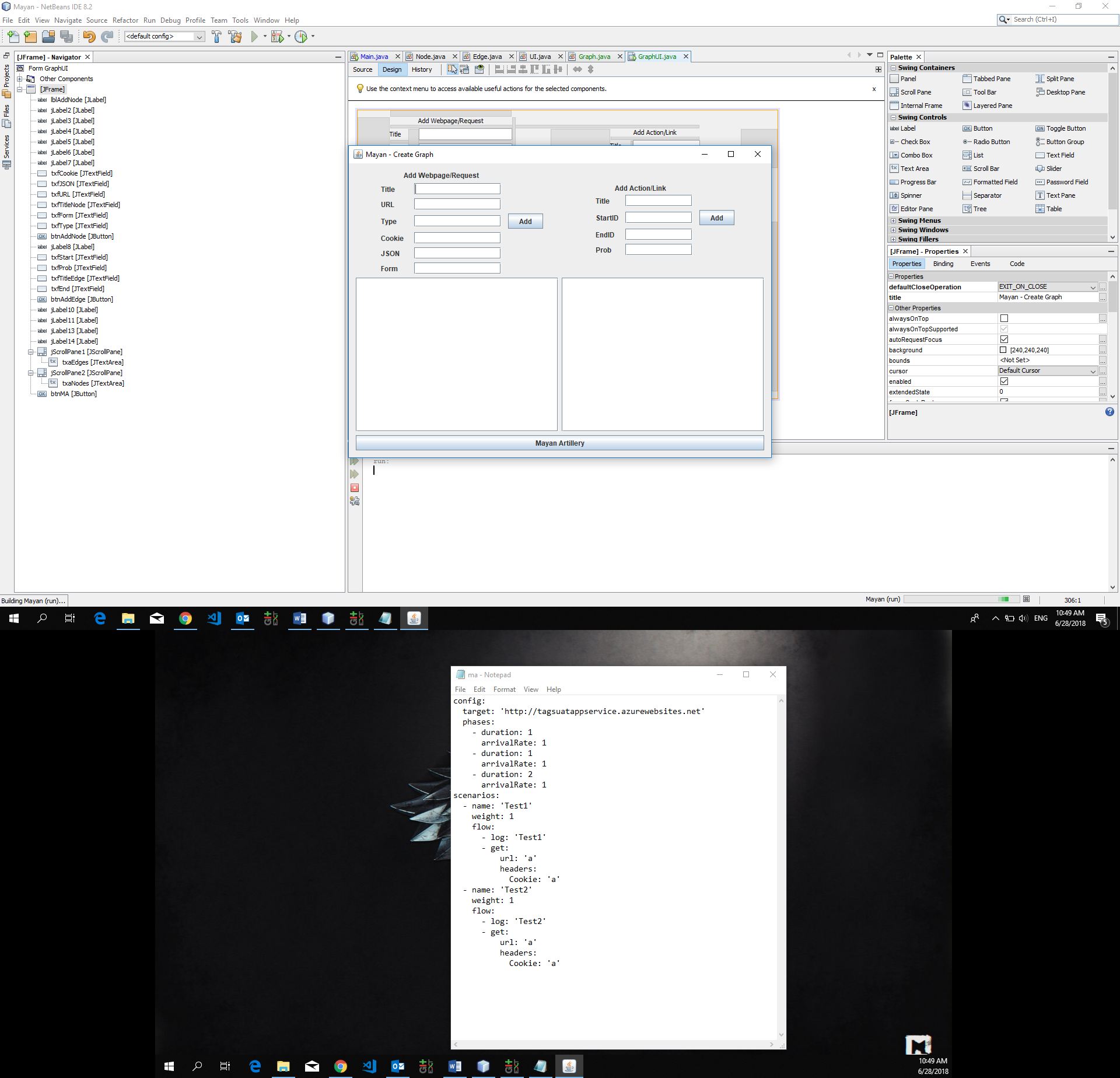
### Durations

The duration over which a user is on a website (i.e. the time they spend on that website) will affect the intensity of the load testing from that user instance. A user will often take time to process which link to press on a page, and this is time that they are not submitting requests, which will ease the load. A user may also leave a page open, e.g. to watch an auction take place, and they will not make any requests at that time. Sometimes, a user may be distracted by other events that cause their page to sit, waiting, and after that event, they will come back and refresh their page. To account for this waiting, a ‘wait’ action must be added to all web pages where a user can possibly wait, with the appropriate probability of a wait occurring at this web page factored in, and this action points to the same web page it originated from. A ‘wait and refresh’ action can also be implemented, which will request that page again too.

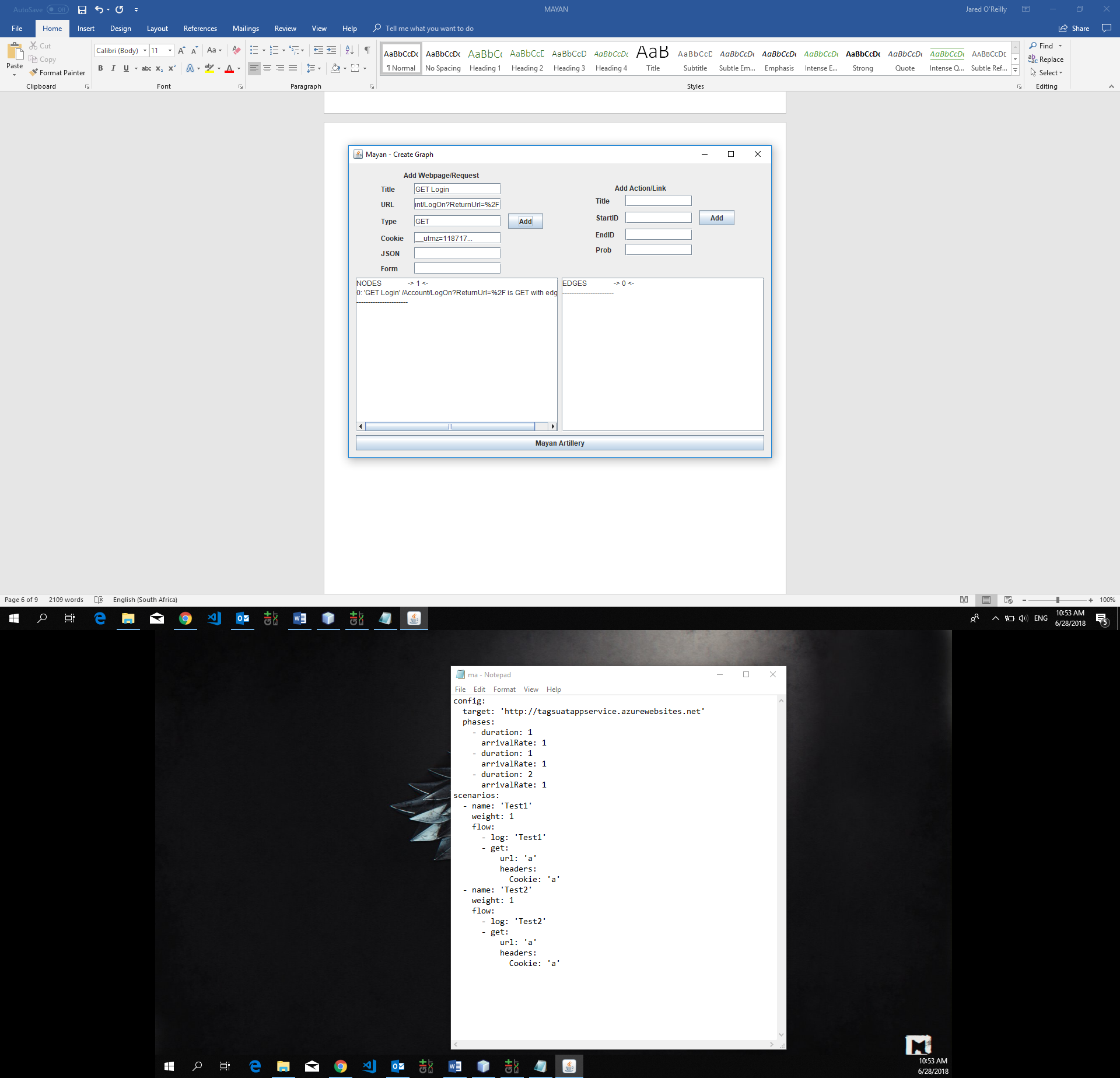
## User Interface

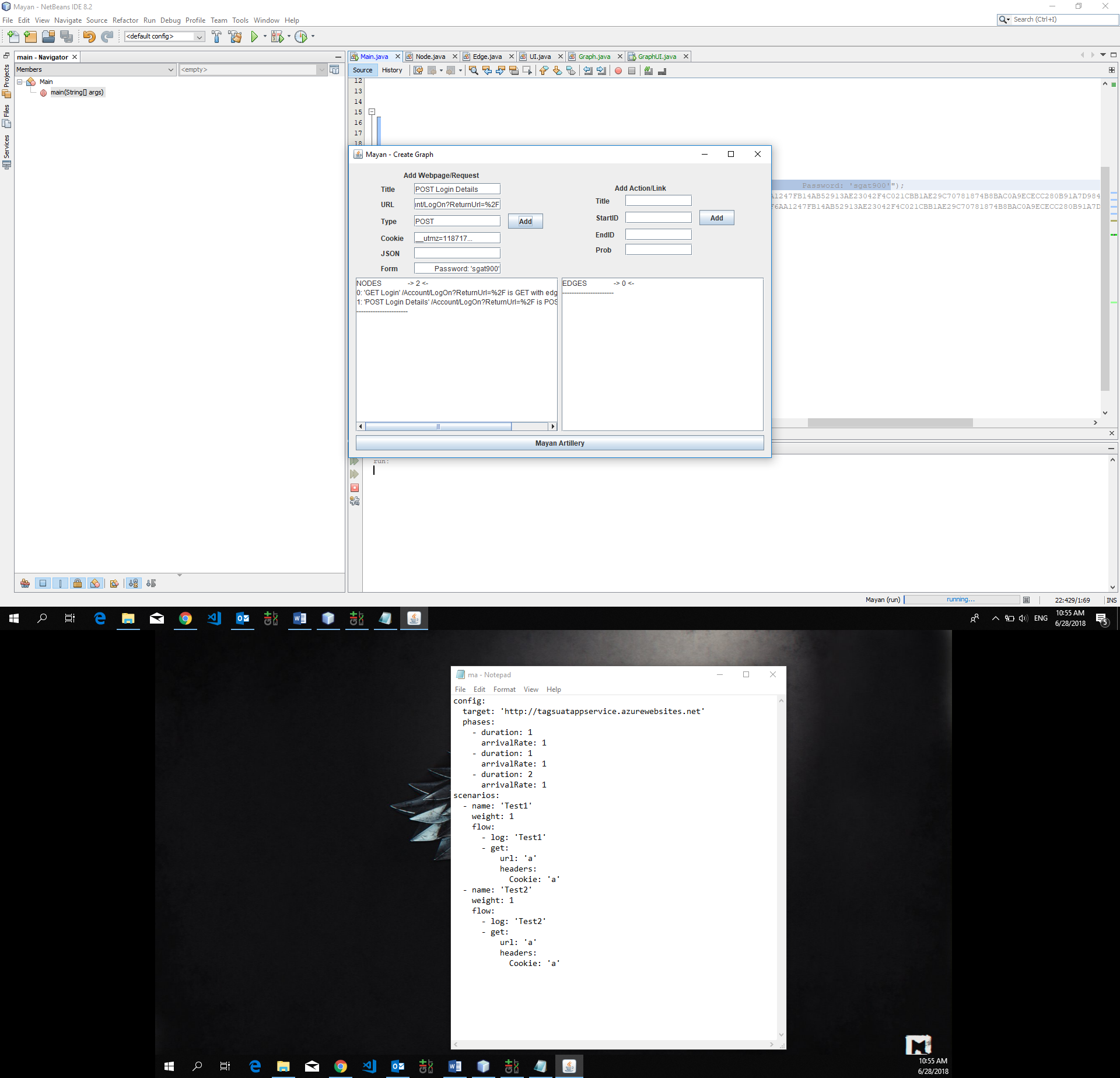
The user interface is the tool with which the user can set up the graph of webpages/requests and links. This interface should be designed so that webpages/requests can be easily created as vertices in the graph, and the links between the webpages/user actions can be easily created as edges in the graph. This interface will be designed from the load testing perspective.

### Interface Screen

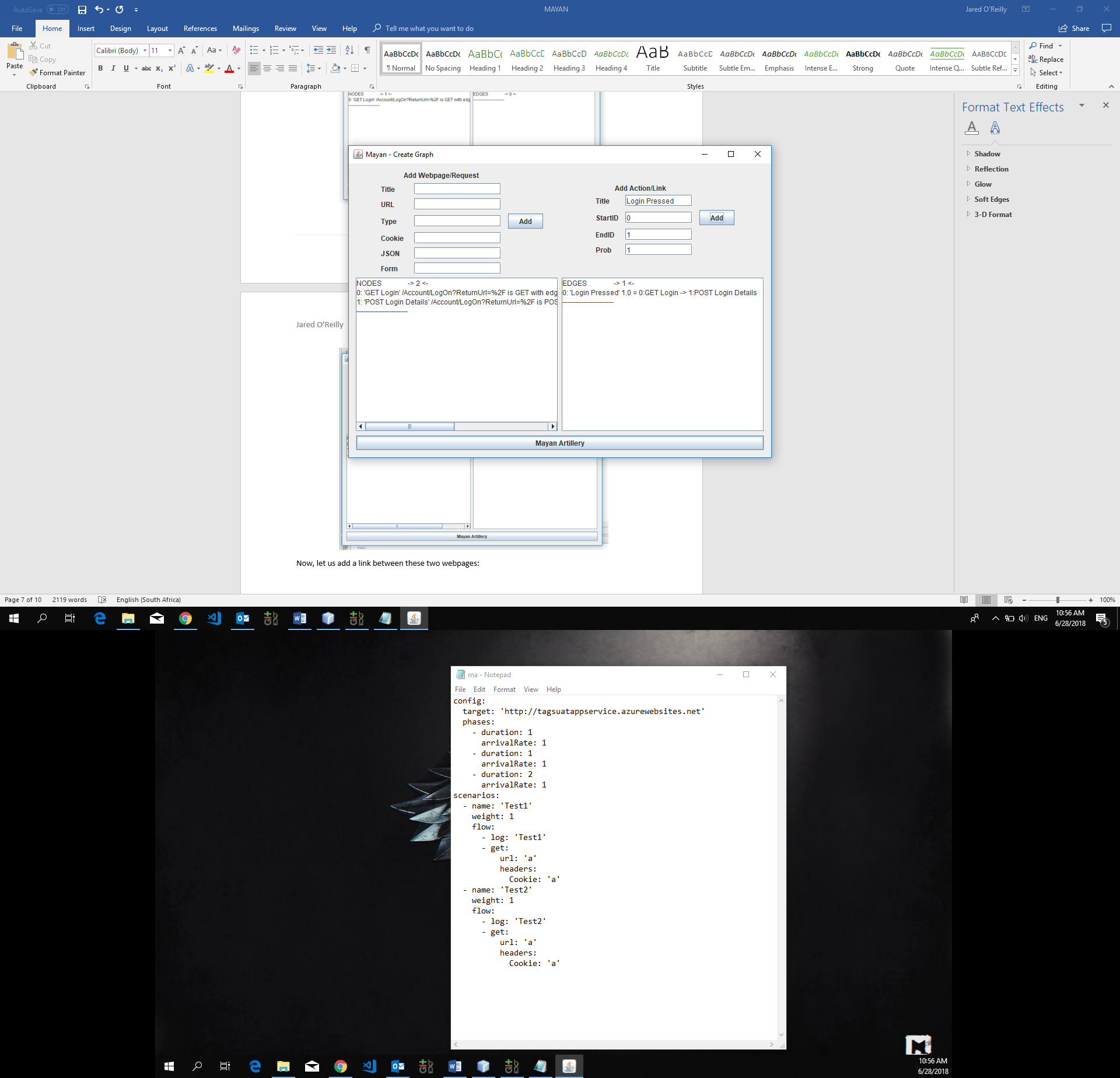


Above, the user interface screen is shown. This screen allows users to add webpages/requests to the graph, and then establish the links between them. Here is an example of adding two webpages:





Now, let us add a link between these two webpages:



The interface is still in it’s developmental stages, and the following functionality should be added:

* Add validation to all input fields
* Make the display of current nodes and edges to be more understandable to the user
* Add drop down menu for certain input fields, like Type, StartID, EndID etc
* Add functionality to update and delete nodes and edges
* General better design and make it look aesthetically pleasing
* Be able to save graphs generated before to reload them without manually inputting again, which could be done by encoding a graph into a string representation

## Artillery Generation

The Artillery scripts that need to be generated by Mayan need to be able to instantly run, without any editing or formatting, as soon as the execution of Mayan is finished. Therefore, careful attention needs to be paid to the construction of the script from the graph, to ensure the script is ready to execute immediately, and without any errors. What will follow in this section is the general approach to how user cases will be converted into Artillery scripts.

### Artillery Scenarios

The *scenarios* section of an Artillery script represents the scenario in which a user performs actions on a website. It consists of 3 subcomponents: *name*, which should be used as a ID for that scenario, *weight*, which represents how common that scenario is (higher weight = more common), and *flow*, which represents the requests that a user will make to that website for webpages. The *flow* subcomponent here needs to be the Artillery representation of the user flows that we will generate from our graph of webpages/requests and actions.

Inside of *flow*, there is an array of Artillery requests: *get* or *post*, with each get/post having a *url* field, optional *headers* fields (e.g. *Cookie*) and optional fields for *json* data, or *form* data that must be attached to the request. The order of these *get*/*post* request in this array inside of *flow* is identical to the order of requests a user may make in their flow/path through a website. **The user flows/cases must be converted into these Artillery scenarios.** What is the general technique for thus?

### 1 User Case to Scenario Conversion Technique

Let us take an example of a simple user flow, and convert this into an Artillery scenario:

**(Login Page)** > *Login Button Pressed* > **(Login Details Submitted)** > *Login Successful* > **(Home Dashboard)**

This represents the user case of them successfully logging in. The Artillery *scenarios* for this is:

scenarios:

- name: 'Test'

weight: 1

flow:

- log: 'T1'

- get:

url: '/Account/LogOn?ReturnUrl=%2F'

headers:

Cookie: '\_\_utmz=118717…’

- post:

url: '/Account/LogOn?ReturnUrl=%2F'

form:

UserName: 'user13'

Password: 'sgat900'

headers:

Cookie: '\_\_utmz=118717…’

From this, we can derive this technique:

* If a page is to be retrieved in a user case/scenario, a *get* component should be included in that flow, with a *url* field and other fields (*headers* etc), in the same position in the flow array as the page retrieval is in the actual user case/flow.
* If data is to be sent and a page retrieved based on the result, a *post* component should be included in the flow, with a *url* field and other fields (*headers*, *json*, *form* etc…), in the same position in the flow array as the data submission and page retrieval is in the user case/flow.

### 2+ User Cases to Scenarios Conversion Technique

If we have multiple possible user cases/flows, we should ensure every user case/flow has its own scenario, with *name*, *weight* and *flow* components, in the *scenarios* section of the Artillery script. This will create an array of scenarios, which is what Artillery was designed to do. Here is an example of two possible user flows for the website we represented in the graph on Page 3:

**(Login Page)** > *Login Button Pressed* > **(Login Details Submitted)** > *Login Successful* > **(Home Dashboard)**

**(Login Page)** > *Login Button Pressed* > **(Login Details Submitted)** > *Login Unsuccessful* > **(Login Page)** > *Login Button Pressed* > **(Login Details Submitted)** > *Login Successful* > **(Home Dashboard)**

These represent a successful user login, and then an unsuccessful followed by a successful user login. The Artillery *scenarios* for this is:

scenarios:

- name: 'Test1'

weight: 1

flow:

- log: 'Test1'

- get:

url: '/Account/LogOn?ReturnUrl=%2F'

headers:

Cookie: '\_\_utmz=118717…’

- post:

url: '/Account/LogOn?ReturnUrl=%2F'

form:

UserName: 'user13'

Password: 'sgat901'

headers:

Cookie: '\_\_utmz=118717…’

- get:

url: '/Account/LogOn?ReturnUrl=%2F'

headers:

Cookie: '\_\_utmz=118717…’

- post:

url: '/Account/LogOn?ReturnUrl=%2F'

form:

UserName: 'user13'

Password: 'sgat900'

↓↓↓↓↓↓↓↓↓↓↓

↓↓↓↓↓↓↓↓↓↓↓

headers:

Cookie: '\_\_utmz=118717…’

- get:

url: '/User/Index'

headers:

Cookie: '\_\_utmz=118717…’

- name: 'Test2'

weight: 1

flow:

- log: 'Test2'

- get:

url: '/Account/LogOn?ReturnUrl=%2F'

headers:

Cookie: '\_\_utmz=118717…’

- post:

url: '/Account/LogOn?ReturnUrl=%2F'

form:

UserName: 'user13'

Password: 'sgat900'

headers:

Cookie: '\_\_utmz=118717…’

- get:

url: '/User/Index'

headers:

Cookie: '\_\_utmz=118717…’

From this, we can derive this technique:

* Each user case/flow must have an element in the scenarios array, representing the flow of that user case, along with a name (ID) and a weight (how common that scenario is).
* If a page is to be retrieved in a user case/scenario, a *get* component should be included in that flow of that scenario, with a *url* field and other fields (*headers* etc), in the same position in the flow array as the page retrieval is in the actual user case/flow.
* If data is to be sent and a page retrieved based on the result in a user case/scenario, a *post* component should be included in the flow of that scenario, with a *url* field and other fields (*headers*, *json*, *form* etc…), in the same position in the flow array as the data submission and page retrieval is in the user case/flow.

# References

The picture on the front page: <https://us.123rf.com/450wm/sateda/sateda1509/sateda150900049/46107575-stock-vector-vector-of-aztec-calendar-on-white-background.jpg?ver=6>