Fusillade user manual

Welcome! 👋

This user manual is here to help get you up and running with Fusillade, a load testing tool written in Go.

What is a load test?

Load tests place pressure on a system. They're used by a range of companies, like American Express and The Financial Times to ensure that their product is capable of handling many users.

What does Fusillade do?

In short, Fusillade is given some directions and goes off and tests your service. It then provides a report.

Some key terms to get started

Before diving into how to use Fusillade, it's worth covering some key terms.

`Config`: this is a file. Config files are made and edited by you, the developer, and given to Fusillade when it starts. Fusillade reads the values in a config and launches a load test based the values stored there.

`User journey`: user journeys are the steps a user takes through your website. Each step is represented with a different URL.

`Latency`: latency is the time it takes for a particular action to complete and there are many ways to measure it. Fusillade measures *response time* which is the time it takes for a request to fully download after being summoned. This is different from *service time*, which measures the time it takes for a request to download after its first byte. Fusillade measures latency in milliseconds.

⭐ How do I get started? ⭐

You run Fusillade with a binary file from the command line. You 'pass' the binary file flags. These flags help you to configure your test.

First, you need to have (or generate) a Fusillade binary.

If you don't have one, install Go on your computer, clone the Github repo to your local machine and run `go build` inside the directory. This will generate a binary.

Second, you need your JSON configuration file. Don't know how that JSON supposed to look? Not to worry. You can consult the documentation below, or run the following command and Fusillade will generate a configuration file that you can edit:

```
./fusillade
```

🚨 **IMPORTANT NOTE ABOUT BINARY NAMES:** the binary Go generates is named after the directory you're in. Therefore, if you cloned from and are inside a `msc-computer-science-project-2019-20-files-davidgodwinbirkbeck` directory, the name of the binary will be `msc-computer-science-project-2019-20-files-davidgodwinbirkbeck`.

If you cloned from the initial Fusillade GitHub repo as I had (I used a private Fusillade repo before transferring all of my work to the Birkbeck repo) then you'd be working with a `fusillade` binary.

🤔 **Why does this matter?** If you see `./fusillade` in this document and your binary is `msc-computer-science-project-2019-20-files-davidgodwinbirkbeck`, then you need to run `./msc-computer-science-project-2019-20-files-davidgodwinbirkbeck` not `./fusillade`. Other than that, nothing changes.

I have my JSON configuration file. How do I run a test?

Provide the name of your config file in the following format:

```
./fusillade -c=your_confile_file.json
```

🔍 Fusillade looks for the config file in the directory where the binary is stored. If your config file is stored somewhere else, you need to tell Fusillade. Give Fusillade a file path (relative to the current working directory or absolute from the root) and it will go and grab it. That might be `-c=../your_config_file.json` to indicate it is in the parent directory.

Not familiar with the command line? This guide might help.

I want graphs in my output. How do I do that?

Set the graph flag `-g` to `true`.

```
./fusillade -c=your_config_file.json -g=true
```

I want a JSON summary. How do I do that?

Set the JSON flag `-j` to `true`.

```
./fusillade -c=your_config_file.json -j=true
```

I want graph and JSON data. How do I do that?

Set the graph and JSON flags to `true`.

```
./fusillade -c=your_config_file.json -g=true -j=true
```

How do I structure a JSON file?

Here is an example JSON configuration file.

```
{
  "urls": ["https://www.google.com", "https://www.voguebusiness.com"],
  "user_journey_amount": 10,
  "rate": 100,
  "pause_length": 100
}
```

What does this JSON file do?

A JSON configuration file tells Fusillade:

The URLs to fire at, the number of journeys it should conduct, the speed at which journeys are launched and the length of time the tool should wait between requesting the next URL.

In more detail:

`"urls"` = the different webpages you want Fusillade to test.

`"user_journey_amount"` = the number of user journeys you want to test.

`"rate"` = the rate at which a user journey is launched in milliseconds. 100 = 1 journey every 100 milliseconds = 10 journeys every second.

`"pause_length"` = the length of time in milliseconds you want the tool to pause between each request.

What data types are needed for the JSON?

`"urls"` expects an array of strings. It cannot be empty.

> 🙂 Some advice: use full URLs. It's easier to parse and less error prone. Favour `https://www.example.com` over `example.com`

`"user_journey_amount"` expects an integer. It cannot be less than 1.

`"rate"` expects an integer. It cannot be less than 1.

`"pause_length"` expects an integer. It cannot be less than 0.

How do I know if my JSON is valid?

First, check the JSON you have made is properly formatted. This tool is useful: https://jsonlint.com/

Second, Fusillade will tell you if the values in a configuration file aren't valid before it launches a test. If there is an error, you'll need to update the JSON file and then rerun the `./fusillade` command with your updated file.

I don't want to use JSON configuration files. What do I do?

If you want to parse your own file type, you'll need to access and change Fusillade's parsing logic in the source code.

All parser-based code can be found in `parser.go`

To start, create a new type. Also, add a Translate method which takes an array of bytes and returns a *Config.

```
type name_of_parser struct{}

func (name_of_parser) Translate(bytesData []byte) *Config {
  //Parsing logic for your particular file type goes in here.
}
```

File parsing logic goes inside Translate().

When 'translating' a file, you'll need to store values in and return a `*Config`. For reference, the values held in a Config are:

```
type Config struct {
  Urls []string json:"urls"
```

```
    Count int json:"user_journey_amount"
    Rate int json:"rate"
    PauseLength int json:"pause_length"
}
```

The `json:"value_from_json_file"` is Go's way of helping a bit. These are useful if your configuration file is relatively flat, but you may find it easier to manually read your file and add different values to a Config.

Once you've implemented Translate(), add your file type to GetParser() method. When Fusillade looks at your configuration file, it will provide an appropriate parser:

```
Parser := map[string]Parser{
   ".json": JSONParser{},
  ".yourFileType": name_of_parser{},
   //Add additional parsers here. They need to implement Parser interface.
}
```

Once you're done, you'll need to create a new binary. Run `go build` to do this.

If the command line doesn't report any errors, you're good to re-run `./fusillade` with your file.