Victoria University of Wellington
School of Engineering and Computer Science

SWEN438: DevOps

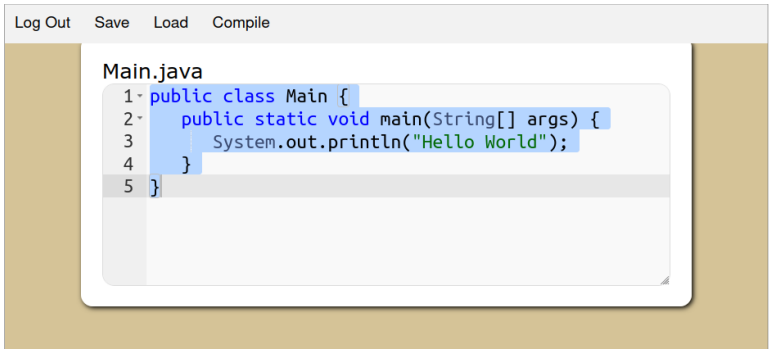
Assignment 4 — Server-Side Diagnosis

Due: 19th September @ 23:59

Overview

This assignment is focused on the *process* of diagnosing software problems on remote servers. For example, when an application running in production crashes periodically for reasons unknown. In such a setting, traditional debugging techniques are of limited use until one understands more about the problem. The typical manner in which this is done is by inspecting *log files*. Such files should provide key clues in the moments leading up to a crash, and thus allow one to recreate the issue on a local machine and apply traditional debugging.

In this assignment, you'll be working with a simple web application called *JavaFiddle* (based very loosely on http://jsfiddle.net). A screenshot of the application is given below:



The application allows users to create accounts, login, load/save files, etc. *At this stage, it should be noted that the application remains relatively unsophisticated.* A typical user session is shown in Figure 1. Having downloaded the application code, you can run the web application using the following command:

```
> mvn exec:java

[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< com.javafiddle:javafiddle >------------------
[INFO] Building javafiddle 1.0.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:3.0.0:java (default-cli) @ javafiddle ---
```

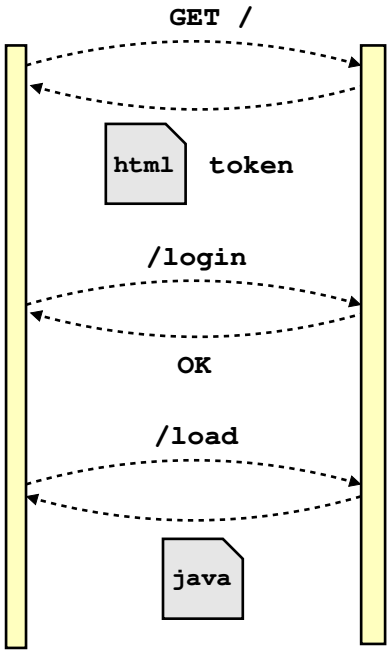


Figure 1: Illustrating part of a sessions whereby a user loads the front page, then logs in and finally loads a specific file from their account. A key aspect is the `token` which assists with authentication. Every browser session is given a unique `token` when the front page is loaded (unless it already had one). Here, a `token` can be thought of roughly speaking as a *cookie*.

As part of the application package, a suite of functional tests are provided (which should all pass). In addition, a *user simulation* is provided (see below).

NOTE: marks will be deducted for any functional tests which fail in the final submission.

HINT: the functional tests provide a *specification* regarding certain aspects of the web application (e.g. what minimal resource limits it should support). See also the Appendix on HTTP codes.

Simulation

A user simulation is provided which attempts to simulate realistic traffic to the web application. For example, the number of users can be adjusted to investigate how the application operates under load. You can run the simulation through the class `javafiddle.simulator.Simulation`. There are various options which can be configured, such as the number of steps to execute and/or whether to start a *local* instance of the application or connect to a *remote* one. The output from the simulation looks something like this:

```
Thomas ..................................... 0 resets and 0 failures.
Benjamin ................................... 0 resets and 0 failures.
Billy2 ..................................... 0 resets and 0 failures.
Brian ...................................... 0 resets and 0 failures.
--
Overall 4 users encountered 0 resets and 0 failures.
```

The primary purpose of the simulation is to help you understand how the *dark simulation* works. You may also use the simulation to check the logging code you have added, and debug your application (if you uncover problems within).

Dark Simulation

The *dark simulation* is a version of the user simulation running in the GitLab CI/CD pipeline (see appendix for details on setting it up). There are two aspects of this which differ from the normal simulation:

- **Dark**. The simulation is *dark* meaning that you cannot see console I/O from the web application. This reflects real life scenarios where remote servers (e.g. running in the cloud) cannot easily be physically inspected.
- **Malicious Users**. In addition to the typical users provided by the user simulation, the dark simulation employs a number of *malicious users*. The malicious users attempt to make requests which affect other users (e.g. crashing the server, stealing information, deleting information, etc).

Figure 2 gives sample output from the dark simulation. **The goal of this assignment is to harden the JavaFiddle web application such that malicious users cannot complete their tasks.** This requires: firstly, learning about the malicious users by *inserting logging code* into the application; and, secondly, *fixing vulnerabilities* in the application.

Unfortunately, the task of understanding the malicious users is made more challenging by the presence of other non-malicious users. You will need to analyse your log files to try and separate out malicious behaviour from normal behaviour. Observe also that the dark simulation is *non-deterministic*. That is, malicious users will not necessarily attack at the same time in every run of the simulation (though for simplicity each is guaranteed to attack at least once during every run). They will also have different usernames across different runs and may take other steps of obfuscate their behaviour.

Submission

For this assignment, you must create a repository entitled `Assignment_4` in your designated SWEN438 project area:

`https://gitlab.ecs.vuw.ac.nz/course-work/swen438/username/`

To make a submission for this assignment, you must submit a text file entitled "`submission.txt`" to the submission system. This file should contain a link to your gitlab repository. The submission system will check your repository is in the correct location and will automatically mark your assignment. **You must ensure your submission is accepted by the submission system without error**. Note, you are permitted to make as many submissions prior to the deadline as you wish.

Marking Criteria

The assignment will be assessed using both quantative and qualitative marking. The former will use an automated marking system (i.e. the dark simulation), whilst the latter will be marked by hand. The breakdown of marks is as follows:

- **Non-Functional Correctness (40 marks)**. Overall resilience of the web application will be judged using the dark simulation. As such, marks are awarded for defeating malicious users.
- **Functional Correctness (30 marks)**. Overall correctness of of the web application will be judged according to a set of *acceptance tests*. These will be similar (though not necessarily identical) to the functional tests provided with the web application.
- **Logging Code (10 marks)**. The manner in which you have extended the web application to record logging information will be assessed. Specifically, it is expected that generated logging information is appropriate for the application in question. For example, log messages should be clear and coherent and avoid. Likewise, appropriate use of *logging levels* should be made to assist administrators in subsequent analysis.
- **Commit Messages (10 marks)**. The manner in which you have used version control will also be assessed. Specifically, it is expected that all commits have sensible and coherent commit messages. As such, you are recommended to use a sensible style for your commit messages.[1] **You are recommended to employ a regexp push rule in the GitLab project for your commit messages, as done in Assignment 2.**[2]
- **Issue Tracking (10 marks)**. You are also expected to use the *GitLab issue tracker* in an appropriate fashion. In particular, it expected that every *feature* added to the program or *defect* fixed in the program has *at least one issue* raised against it (and potentially many more). The issue description is expected to clearly and coherently described the issue being addressed.

[1] See "How to Write a Git Commit Message", https://chris.beams.io/posts/git-commit/.

[2] See also "https://manual.limesurvey.org/Standard_for_Git_commit_messages"

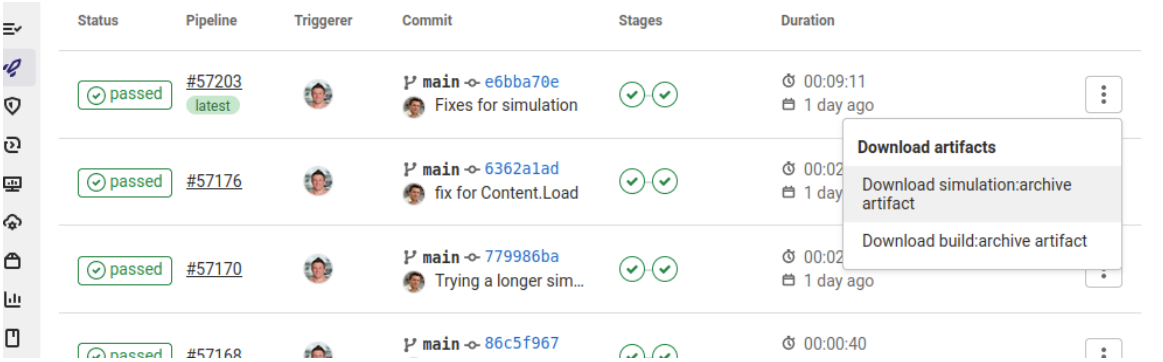
```
================================================================================
Willie ..................................................... 1 resets and 0 failures.
Jose ....................................................... 5 resets and 0 failures.
Juan ...................................................... 10 resets and 0 failures.
Jack ....................................................... 8 resets and 0 failures.
Gary6 ...................................................... 6 resets and 0 failures.
Kenneth .................................................... 5 resets and 0 failures.
Steven ..................................................... 11 resets and 0 failures.
Gary ....................................................... 0 resets and 0 failures.
Nathan ..................................................... 0 resets and 0 failures.
Wayne ...................................................... 3 resets and 0 failures.
--------------------------------------------------------------------------------
Overall 10 friendly users encountered 49 resets and 0 failures.

================================================================================
Malicious User #1 ..................................................... failed [1 marks]
Malicious User #2 ..................................................... failed [1 marks]
Malicious User #3 ..................................................... failed [1 marks]
Malicious User #4 ..................................................... failed [1 marks]
Malicious User #5 ................................................. succeeded [0 marks]
Malicious User #6 ................................................. succeeded [0 marks]
Malicious User #7 ................................................. succeeded [0 marks]
Malicious User #8 ................................................. succeeded [0 marks]
Malicious User #9 ................................................. succeeded [0 marks]
Malicious User #10 ................................................ succeeded [0 marks]
Malicious User #11 ................................................ succeeded [0 marks]
Malicious User #12 ................................................ succeeded [0 marks]
Malicious User #13 ................................................ succeeded [0 marks]
--------------------------------------------------------------------------------
Total                                                                    4 marks
```

Figure 2: Illustrating sample output from the dark simulation. We can see that thirteen malicious users attempted an unknown number of attacks on the application, of which four failed. The output shows that marks are awarded when malicious users are defeated. The amount of marks awarded varies depending on the level of sophistication employed by the malicious user. Finally, in this example simulation there were only ten friendly users. We observe that these users encountered some number of "resets" (i.e. situations where the server unexpectedly closed the connection). In this case, however, none of the users encountered any "failures" (i.e. situations where something they expected to succeed, such as loading a file they had previously saved, actually failed (e.g. because a malicious user deleted that file).

Appendix — Configuring the Dark Simulation

To configure the dark simulation, a suggested `.gitlab-ci.yml` file is given below. This assumes the generated Log4J2 log file is `log/javafiddle.log` which is the default used in the given `log/log4j2.xml` configuration (see below).

In order to see the log output from the dark simulation, you will need to download the exported artifact from the pipeline. The following illustrates:



Finally, we note that the number of steps used by the dark simulation can be configured using the command-line option (e.g. "`-steps 1000`" to limit the simulation run to 100 steps). **NOTE:** the simulation run used for marking will be 5000 steps.

Appendix — Configuring Log4J

A sample Log4J configuration file (`log/log4j2.xml`) has been provided for you. This rule should be kept in the same location, as the dark simulation expects this. Furthermore, there are few notes about using this:

- **Log4J with Maven**. When running the application from the command-line using Maven, you need to tell it where to find the Log4J configuration file. This can be done with a command-line option as follows:

  ```
  mvn exec:java -Dexec.args="-log4j2 log/log4j2.xml"
  ```

- **Log4J with Eclipse**. When running the application through eclipse, there are several options. You can modify your run configuration to pass the command-line option "`-log4j2 log/log4j2.xml`" through; alternatively, you can add as a "Class Folder" to the `CLASSPATH` via the build path settings.

When Log4J is correctly configured you should be able to see at least the following output when the server runs:

```
20:22:19.344 [main] TRACE Main - Attempting to start JavaFiddle on port 8080
20:22:19.360 [main] TRACE Main - JavaFiddle running on port 8080.
```

Looking inside the class `javafiddle.Main` and you should see the `logger.trace()` messages which generate the above output. As such, you can add logging to your server in order to get valuable information from the dark simulation.

Append — HTTP Codes

There are a number of HTTP codes which are relevant to this assignment:

- **400 "Bad Request"**. The server should return this for a malformed request (e.g. invalid JSON).
- **403 "Forbidden**. The server should return this when the user attempts to do something for which they don't have permission.
- **404 "Not Found"**. The server should return this when the user requests something which does not exist (e.g. a file).
- **413 "Payload Too Large"**. The server should return this when it is unable to accept a large request.

```
image: whileydave/javafiddle:latest

cache:
  key: maven-deps
  paths:
    - .m2/repository

default:
  tags:
   - docker

variables:
  MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"

build:
  stage: build
  script:
    - mvn test
  artifacts:
    expire_in: 1h
    paths:
      - target

simulation:
  stage: test
  script:
    - darksim
  artifacts:
    expire_in: 1d
    paths:
      - log/javafiddle.log
```

Figure 3: Suggested contents for the `.gitlab-ci.yml` file to run the dark simulation. This does several things: **(1)** it uses the provided docker image which contains the dark simulation; **(2)** it sets up caching to preventing Maven from downloading the same artifacts between different runs of the pipeline; **(3)** it configures the file `log/javafiddle.log` as an *artifact* meaning that it will be exported from the pipeline (i.e. so it can be inspected).