SWITCHING ONE – PIT SIMULATOR

PIT SIMULATOR TECHNICAL USER GUIDE

Version:	1.0
Status:	Issued
Author:	Thomas Petty

netcompany

Document history

Version	Date	Author	Status	Comments
1.0	23-10-2019	Thomas Petty	Release	Initial Release

Reviewers

Name	Signature	Title/Responsibility	Release Date	Version Number
James Young		Delivery Manager	22-10-2019	0.3
Jason Chan		Senior Consultant	23-10-2019	0.4
G Morris		DCC Quality Assurance	23-10-2019	0.4

Approvals

Name	Signature	Title/Responsibility	Release Date	Version Number
James Young		Delivery Manager	23-10-2019	1.0

References

Reference	Title	Author	Version
1	NC-0117 Simulator Configuration Guide	Jamie Simm	1.0
2	CSS Interface Design Specification	Landmark	8.2
3	PIT Simulator - Release Note	James Young	1.0
4	Defect Management Plan	Netcompany SI	1.0

Table of contents

1 Introduction

1.1 Purpose

This document is intended to inform the use of the PIT Simulator. It specifies how inbound messages can be accepted, how to generate outbound messages using the User Interface (UI) and the viewing and downloading of logs. A high-level description of the technologies involved, along with an explanation of their function in the PIT Simulator, are included in the hope that this context might better inform its use. This document includes brief instructions on starting the PIT Simulator, however the Simulator Configuration Guide [Ref 1] should be consulted for a more detailed description as well as install.

1.2 Target Audience

This user guide is intended for use by Central Data Services (CDS) within the Switching Programme. This consists of Existing Service Providers, Licensed Parties and new Service Providers.

1.3 Definitions and abbreviations

Item	Description
Application Programming Interface (API)	An API is a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.
Central Data Services (CDS)	Central Data Services are the services included in the switching programme. This includes Existing Service Providers, Licensed Parties and new Service Providers.
Command Line Interface (CLI)	The console into which many of the following commands for this guide are typed. In Windows go to start, type “cmd” in search and select “Command Prompt”. For Mac, this is called “Terminal” and is in the Utilities folder in Applications. To open it, either open the Applications folder, then open Utilities and double-click on “*Terminal”*, or press “cmd + spacebar” to launch Spotlight and type "*Terminal*," then double-click the search result.
Container	A container is an instance of a Docker image. It allows software to be delivered alongside all the necessary libraries and configuration files. Containers are isolated from one another and run on a single operating-system kernel which makes them more lightweight than virtual machines.
Central Switching Service (CSS)	Core of the switching process, orchestrating all switching process message flows and maintaining the authoritative source of registration data.
Docker	Docker is a product that uses OS-level virtualization to deliver software in packages called containers.
Endpoint	An endpoint is the address of a resource exposed by an API. This can be in the form of a URL and forms one end of a communication channel, the other end being the user’s system.
Existing Service Providers (ESP)	Organisations who currently provide a range of Central Data Systems and services in the gas and electricity markets, for example Electralink, Gemserv, Xoserve, St Clements.

Gradle	Gradle is a build automation tool often used for languages such as Java, Groovy or Scala. Gradle can be configured to run tasks which performs functions such as compile, run tests and create documentation.
Hypertext Transfer Protocol (HTTP)	HTTP is the underlying protocol used by the World Wide Web. It defines how messages are formatted and transmitted as well as what actions should be taken in response.
JSON Web Signature (JWS)	JWS is a standard for signing arbitrary data. This signature can be used to verify a messages authenticity.
Licensed Party (LP)	A Licensed Party is a party which that will interact with the CSS upon release.
Open Application Programming Interface (API)	An Open API is an API that uses a common or universal language and structure. This makes it easier for developers to use the services provided by the API.
Pre-Integration Testing (PIT) Simulator	The PIT Simulator is the first simulator of three which aims to prove that Parties are able to receive, validate and respond to messages from the CSS. This simulator must be used for testing as a precursor to the next stage of testing.
RESTful API	A RESTful (Representational State Transfer) API (Application Programming Interface) is an API base on the REST architectural style for designing lightweight, layered web services.
Root folder	The highest-level folder of the PIT Simulator, containing every source file and directory in the software. This is currently named **CSS-Simulator**.
Swagger	Swagger is an open-source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful web services.
Uniform Resource Locator (URL)	A URL is a web address which specifies the location of a resource.

Table 1: Definitions and abbreviations

1.4 Delimitations

While this document explains interaction with the PIT Simulator in detail, it doesn't cover the method of interaction of the end user. The reader may choose some arbitrary method of interaction to suit their needs, the choice of which cannot be anticipated and consequently cannot be described. Additionally, installation and running of the PIT Simulator is not covered here, but in the Configuration Guide [Ref 1].

2 PIT Simulator

2.1 Purpose

Central Data Services (CDS) will need to test if their solution is compatible before release of the CSS. The first stage of this is the PIT Simulator which verifies their solution can send and receive messages to and from the CSS as expected. Each CDS will run a copy of the PIT Simulator configured to their own environment to test various scenarios. Successfully completing all the required scenarios will allow the CDS to leave this stage of testing and move onto the next phase of testing. The agreed subset of tests to be witnessed by the assuring parties and key stakeholders as part of PIT Execution will be made available independently of the Simulator and User Guide via the Test Working Group.

2.2 Features

Key features of the PIT Simulator are:

- Initial install and configuration of the PIT Simulator is possible across a range of different environments
- Inbound messages (to the PIT simulator):
    - Are validated according to CSS Interface Design Specification [Ref 2]
    - Send an acknowledgement message
- Outbound messages (from the PIT simulator):
    - Can be selected from a pre-built template based on the CSS Interface Design Specification [Ref 2]
    - Message data is input via UI
    - Acknowledgement of outbound message generated and displayed via UI
    - Outbound message is generated from stub and sent to CDS as defined
    - Outbound messages are signed with a JWS signature
- Logs of the system can be viewed and downloaded for debugging and proof of required scenarios

3 Launching the PIT Simulator

After following the Configuration Guide [Ref 1], there will either be a .jar file or a Docker container accessed via Docker Hub. The process of launching the simulator will be the same regardless of whether you are using a .jar file that has been shared, or the original generated in the Configuration Guide [Ref 1]. Each time the machine running the PIT simulator is shut down, the PIT simulator will need to be relaunched.

3.1 Launching from the .jar file

1. Open a command line in the same folder as **css-simulator-[VERSION]-SNAPSHOT.jar** where **[VERSION]** is the version of the PIT simulator downloaded.
2. type `java -jar css-simulator-[VERSION]-SNAPSHOT.jar` to start the PIT simulator.

3.2 Launching from Docker

Once the Docker image has been pulled from the Docker repository to which it was uploaded during installation, these commands can be used to run the PIT Simulator from it. If you have not yet retrieved the image from the Docker repository, there is a useful guide: https://ropenscilabs.github.io/r-docker-tutorial/04-Dockerhub.html. Alternatively, this guide goes into more depth: https://docs.docker.com/engine/reference/commandline/pull/.

1. To discover the Image ID of the downloaded container, open the command line and type `docker images`. A listing will appear as shown in Figure 1.

```
$ docker images
REPOSITORY                                                                  TAG                 IMAGE ID            CREATED             SIZE
docker-registry-default.40                .io/css-simulator/css-simulator-docker-image   latest              909db8733d9b        49 years ago        167MB
```

Figure 1: The Docker image for the simulator, produced as a table by the docker images command.[1]

2. To run the Docker container based on this generated image, use the command:

   `docker run -p 8080:8080 [IMAGE ID]`

   where `[IMAGE ID]` is the unique mix of letters and numbers in the table which is outputted by the `docker images` command under the IMAGE ID column.

   Image ID is unique to each Docker image. As an example, for Figure 1 the relevant command is `docker run -p 8080:8080 909db8733d9b`

3.3 Launching with Gradle

From the root folder of the PIT Simulator, type `gradle bootRun`. This will run the PIT Simulator for use. This command will need to be repeated each time the PIT Simulator service is stopped.

[1] By default, the created field will state '49 years ago'. This is default Docker behaviour and not a defect.

4 Accessing endpoints

Following the installation method outlined in the Configuration Guide [Ref 1], a number of endpoints will be exposed on the server URL in the format http:// <IP ADDRESS> : <PORT> /api/v1 , for example http://127.0.0.1:8080/api/v1.

The endpoints of messages can be found in the CSS Interface Design Specification [Ref 2]. These are added to the server URL. For example the endpoint /domain/alliance, is located at the base URL http://127.0.0.1:8080/api/v1/domain/alliance.

5 Inbound messages

Inbound messages are received in the form of HTTP requests consisting of a header and a body. The header is standard across the whole solution and contains information on message formatting. The body varies depending on the message method. These messages are received on the base URL along with any query parameters needed. The query parameter is added to the end of the base URL as shown below.

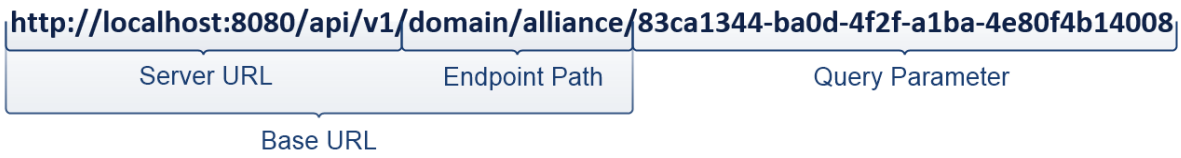

Figure 2: Breakdown of endpoint path

On receiving a request, the PIT simulator generates a response in accordance with the CSS Interface Design Specification [Ref 2]. To validate a request creation, check the logs as described in section 7.2. These logs can also be compared to logs on the user system to validate a required scenario.

6 Outbound messages

6.1 Description

Outbound messages are sent in accordance with the CSS Interface Design Specification [Ref 2]. Some of these messages are sent asynchronously back to the sender and some to systems subscribed to events. If multiple messages are queued the next message isn't sent until a response is received confirming receipt of the previous message. In these messages there are two optional parameters; the repetitions and sending rate. Repetitions defines how many times the outgoing message will be sent, and sending rate how many messages can be sent per second (assuming receipts are received quickly enough). If not specified both values default to 1.

6.2 Using the UI to generate events

1. Navigate to the server URL. By default, this is http://localhost:8080.
2. Open the test scripts folder provided within the project, an outline of the project structure can be seen in figure 3.
3. Within the [test scripts] folder open the context you wish to use, e.g. **DES**, then the file with the name of the event you wish to send, e.g. **RegistrationCancelledSynchronisation.json**. Copy the contents of this file.

```
[CSS-Simulator]
|----[test scripts]
|      |---- [DES]
|      |      |---- RegistrationCancelledSynchronisation.json
|      |      |---- RegistrationEventSynchronisation.json
|      |      |---- RegistrationPendingSynchronisation.json
|      |      |---- RegistrationSecuredActiveSynchronisation.json
|      |      |---- RegistrationSecuredInactiveSynchronisation.json
|      |      |---- RetailEnergyLocationSynchronisation.json
```

Figure 3: The folder structure path to DES example outbound messages

4. Click on the **Edit JSON** button next to the **BulkEvents** heading.

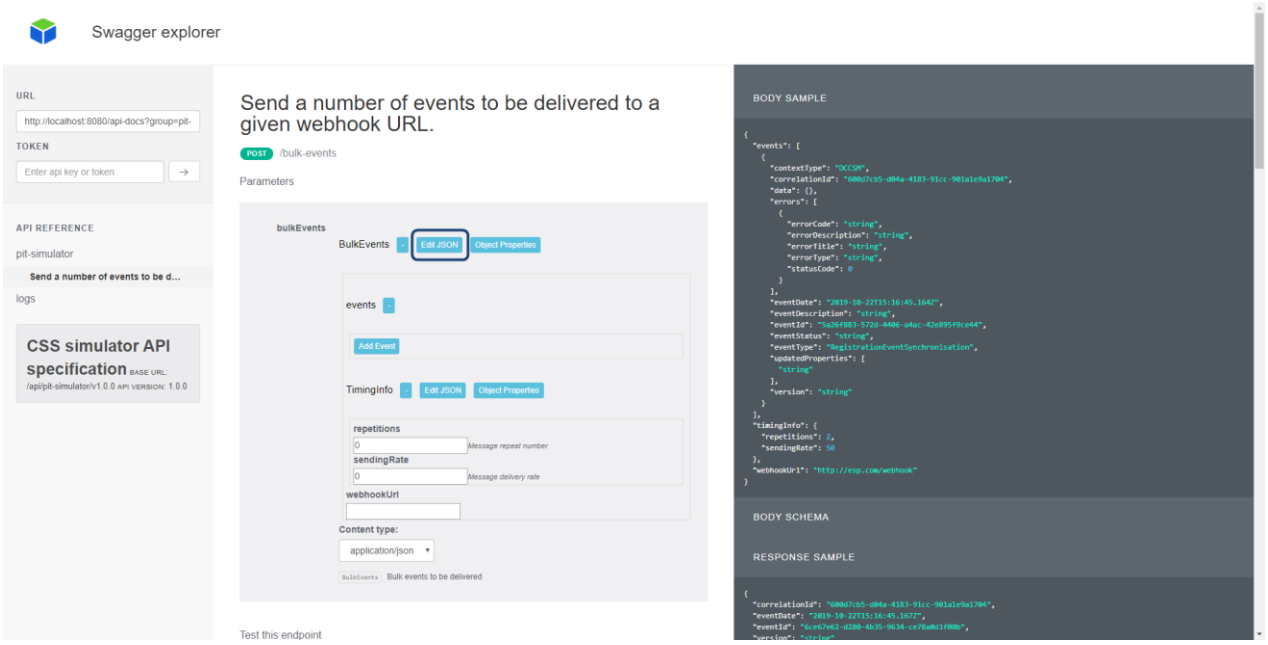

Figure 4: Outbound message UI, Edit JSON Button highlighted

5. Paste the contents of the JSON file from step 3 into the input box provided and click **Save**.

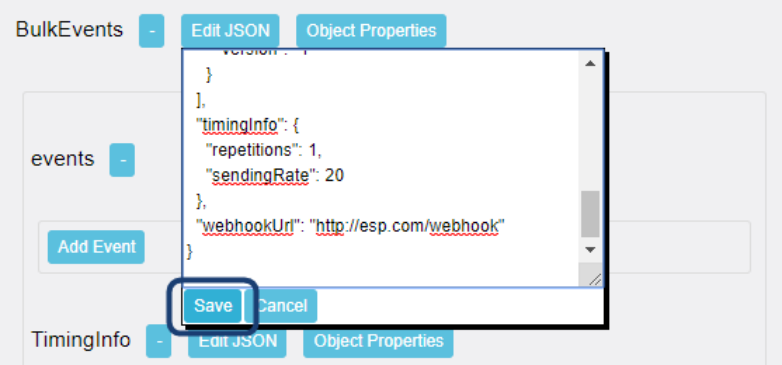

Figure 5: Edit JSON input box with Save button highlighted

6. The fields for the event have now been set up with example values and data types. These can be modified as necessary. An example of each field in the correct format are provided, for example dates have the correct date format. For more information on the format of a particular property check the CSS Interface Design Specification [Ref 2].

7. Underneath the events section there is a **TimingInfo** section which includes **repetitions** and **sendingRate**. **Repetitions** specifies how many times an event is sent to the user's system and **sendingRate** the maximum number of messages sent per second. If these are not specified these will be set to 1.

8. Define the **webhookUrl** to send the events to. This specifies the endpoint on the user's system which listens for the events.

9. Click **TRY** – see *Figure 6* - and a popup will appear showing a response – see *Figure 7*. This will either confirm the event has been accepted (response code 202) or display an error. The **errorDescription** – see *Figure 8* - gives more information about why this error has occurred.

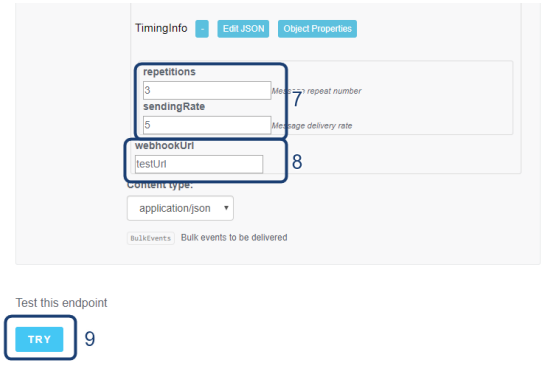

Figure 6: Defining timing information, the webhook URL and sending the event

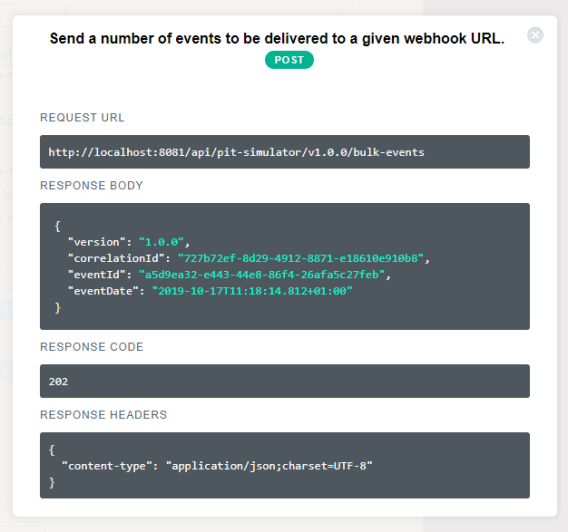

Figure 7: A valid response message

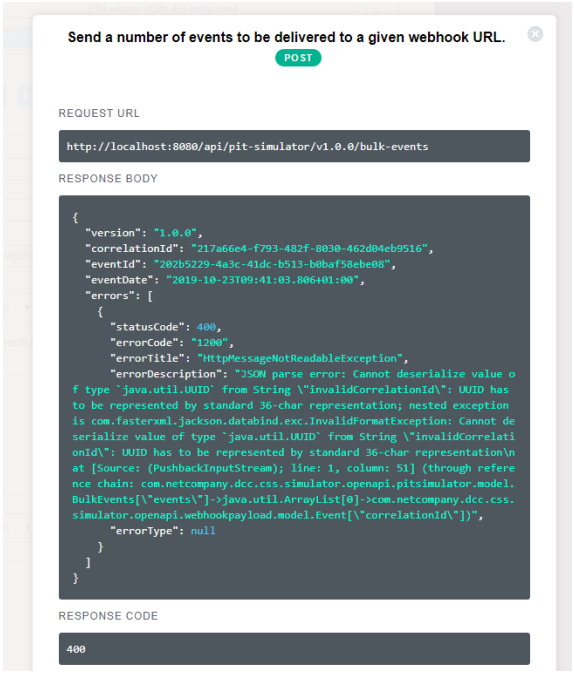

Figure 8: An error response message

7 Logging

7.1 Overview

As the application runs, log files are created for use when debugging. These are divided into two logs; the evidence.log and system.log, made up of several entries ordered by the time and date it was created. A brief breakdown of log entries can be seen in Figure 8 below. These logs can be used as test evidence for required scenarios, see section 9.

Figure 9: Breakdown of INFO and TRACE log messages

The evidence log contains entries for every HTTP request and response. Each entry contains information on the type of message (request or response), origin, method, protocols and the message content (body and header). When sending a request message, a response is generated by the API which appears as another line in the log. By searching for a message's correlation ID, you can find its corresponding message. These have the level **TRACE** and can also be found in the system log.

As well as **TRACE** messages the system log also contains **WARN** messages whenever an error is thrown containing the stack trace and **DEBUG** and **INFO** messages containing general information.

7.2 Viewing and Downloading the logs

1. Navigate to the server URL. By default, this is http://localhost:8080.
2. Click on **logs** at the left side of the page – see *Figure 10* - then **TRY** to get a list of available logs.

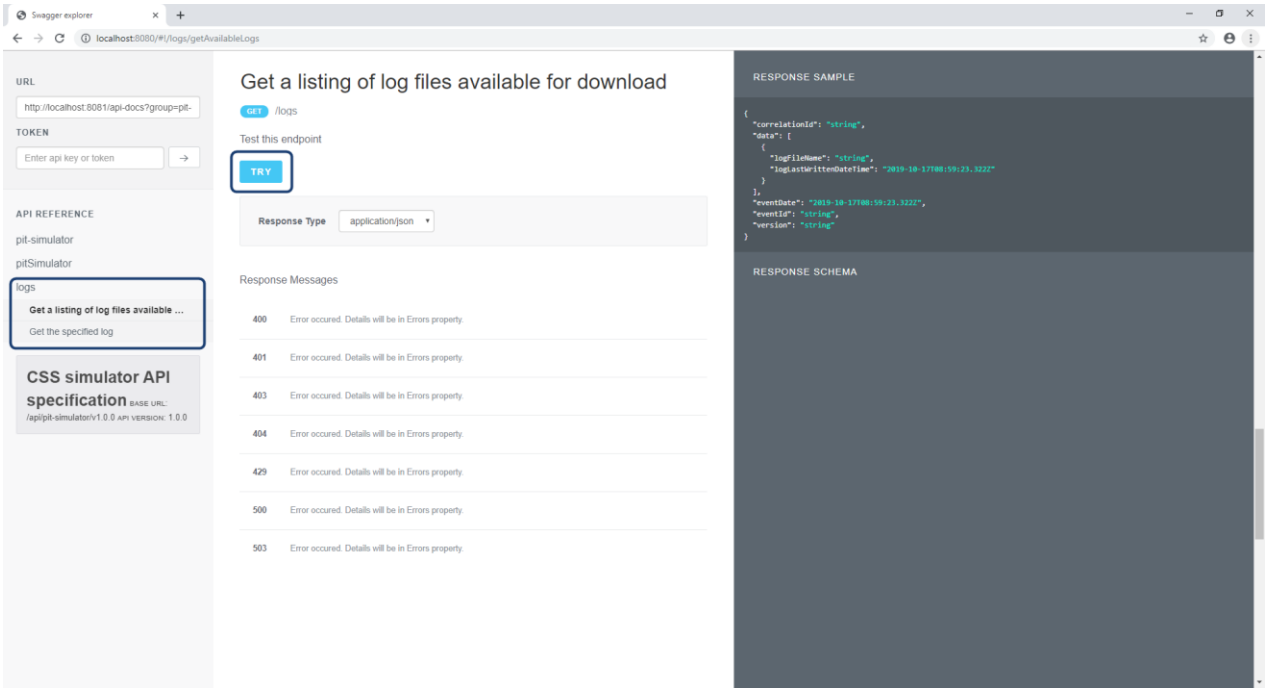

Figure 10: Getting a list of log files

3. From the response body copy the **logFileName** of the log – see *Figure 11* - to be downloaded.

Figure 11: Response list of available evidence logs

4. Click on **Get the specified log** from the left side of the page – see *Figure 12* - under the **logs** heading.

5. Paste the **logFileName** into the **logName** input box – see *Figure 12*.

6. Select **text/plain** from the response type drop-down.

7. Click on **TRY** – see *Figure 12*. Here the log can be viewed – see *Figure 13*.

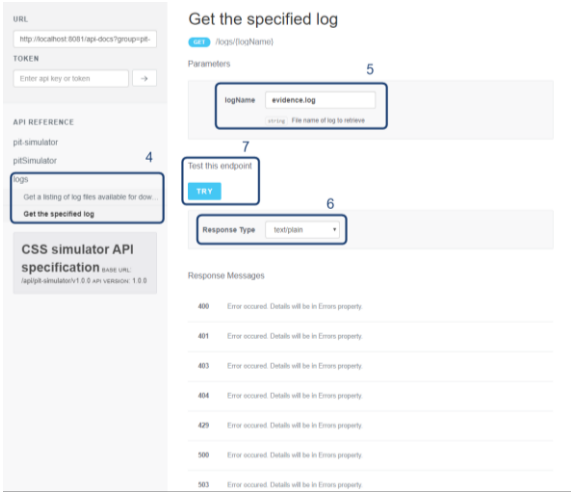

Figure 12: Getting a specified log

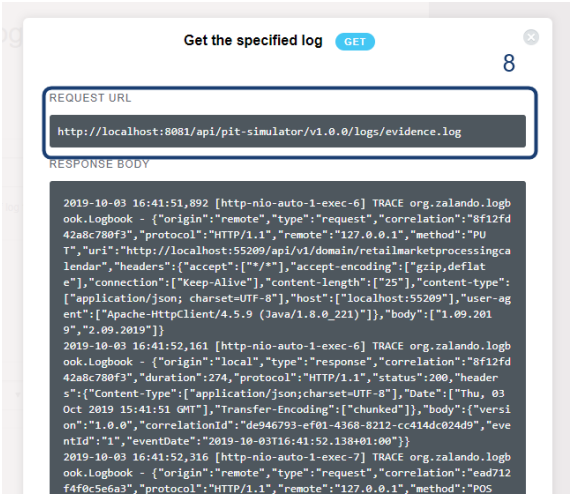

Figure 13: Response evidence log

8. To download the log, copy the **REQUEST URL** – see *Figure 13*.

9. Paste the request URL into a new tab.

10. Right click on the page and click **Save as...**

11. The log will be downloaded as a text file into the specified location.

8 Reporting Defects

In the event a defect is encountered with the PIT simulator an issue can be raised by contacting the SI team at DCCSISS@netcompany.com. They will then raise the defect on JIRA and respond accordingly with the tracking number. More information on how to contact the SI team can be found in the PIT Simulator Release Note [Ref 3] and the Defect Management Plan [Ref 4].

9 Required Scenarios

A required scenario is one that a CDS needs to pass successfully in order to move on to the next test phase. For example, passing all required scenarios in the PIT simulator will allow an ESP to move onto the SIT test phase and a LP to move onto the UIT test phase. A required scenario can be verified by sending the relevant logs to DCCSISS@netcompany.com. The agreed subset of tests to be witnessed by the assuring parties and key stakeholders as part of PIT Execution will be made available independently of the Simulator and User Guide via the Test Working Group.

END OF DOCUMENT