Amnesia ReST API Structure

Introduction

The backend of Amnesia is implemented in Java using the Spring framework. Its components offer a ReST API that handles anonymization requests issued by the web interface. It uses a temporary local storage for the anonymization purposes and final results are returned by the ReST interface. There are three ways to communicate with Amnesia's backend procedures:

1. Visit Amnesia's online version (https://amnesia.openaire.eu/amnesia/) provided only for test purposes for small datasets and hierarchies. To anonymize your private data, we recommend you use the following two options.

2. Run Amnesia's application and execute the web request on localhost (port 8181).

3. Clone Amnesia's repository from Github.

   To successfully build the **Amnesia** engine follow these steps:

   1. Clone the project
   2. Go to controller/AppCon.java and set the variable "os" with desired operation system ("windows" or "linux")
   3. Build the project
   4. Take the .jar from the "scr/target" directory
   5. Run via terminal java -Xms1024m -Xmx4096m -Dorg.eclipse.jetty.server.Request.maxFormKeys=1000000 -Dorg.eclipse.jetty.server.Request.maxFormContentSize=1000000 -jar "path_to_jar_file" --server.port=8181

   Then you can use the Amnesia's API as in the previous step.

All the examples below are harmonized with the last two methods on localhost using datasets and hierarchies provided in https://amnesia.openaire.eu/Datasets.zip

ReST API endpoints

getSession [POST]

- Simple HTTP request returning the session id

Web services require a field called "Session" as an argument. "Session" is the HTTP session(Cookie) of the user, which is called JSESSIONID. This call returns the JSESSIONID of the user, in order to use it.

Example via Terminal:

```
# /getSession [POST]
$ curl -X POST http://localhost:8181/getSession
> {"Status":"Success","Session_Id":"D6DF59C4A03432B79BC61961F1671EDB"}
```

loadData [POST]

- file: MultipartFile, original dataset in .txt or .csv format
- datasetType: tabular -> Simple table, set-> Sets of values, RelSet -> Table with a set-valued attribute and Disk -> Disk based simple table
- del: Delimiter of dataset
- columnsType: Json string with keys the columns' names and values the data type (int, double, string, date, set(hierarchy 's variable type must be string))
- session: Set session id as header parameter

Additional parameters

- if datasetType is RelSet
    - delSet: delimiter for the set-valued column

Examples via Terminal:

- Simple Table (tabular)

```
# /loadData [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/loadData
--form file=@"/path/to/Datasets/Scenarios/Simple Table-Disk based simple
table/data1/newData.txt"
--form del=,
--form datasetType=tabular --form
columnsType="{\"zipcode\":\"int\",\"age\":\"int\",\"gender\":\"string\",
\"salary\":\"int\"}"

> {"Status":"Success","Message":"Dataset loaded successfully!"}
```

- Sets of Values (set)
  Amnesia handles "Procedure Codes", which is a set column, as a string column.

```
# /loadData [POST]
$  curl  -X POST -H "Cookie:
JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/loadData
--form file=@"/path/to/Datasets/Scenarios/Sets of
Values/data1/setNumbers.csv"
--form del="|"
--form datasetType=set
```

```
--form columnsType="{\"Procedure Codes\": \"set\"}"

> {"Status":"Success","Message":"Dataset loaded successfully!"}
```

- Table with a set-valued attribute (RelSet)

  Amnesia handles “Procedure Codes”, which is a set column, as a string column

```
# /loadData [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/loadData
--form file=@"/path/to/Datasets/Scenarios/Table with a set-valued
attribute/data1/mixedDataSet.csv"
--form del=,
--form delSet="|"
--form datasetType=RelSet
--form columnsType="{\"Procedure Codes\": \"set\", \"salary\":\"int\",
\"age\":\"int\", \"date\":\"date\"}"

> {"Status":"Success","Message":"Dataset loaded successfully!"}
```

- Disk based simple table (Disk)

```
# /loadData [POST]
$  curl  -X POST -H "Cookie:
JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8084/amnesia/loadData
--form file=@"/path/to/Datasets/Scenarios/Simple Table-Disk based simple
table/data1/newData.txt"
--form del=,
--form datasetType=Disk
--form
columnsType="{\"zipcode\":\"int\",\"age\":\"int\",\"gender\":\"string\",
\"salary\":\"int\"}"

> {"Status":"Success","Message":"Dataset loaded successfully!"}
```

generateHierarchy [POST]

Hierarchy autogeneration requires loading the dataset initially to find the appropriate values in a specific column to generate a new hierarchy. In the examples below, we consider that the mixedDataSet.csv dataset (Table with a set-valued attribute) has been loaded.

- hierType: type of hierarchy (mask (only for string data),distinct,range}

- varType: (int,double,string,date)
- attribute: column name
- hierName: name of hierarchy
- session: Set session id as header parameter

Additional parameters for individual autogenerate conditions

- hierType is range and varType is {int or double}
    - startLimit: int
    - endLimit: int

  The parameters above declare the domain range e.g. 1-100

    - fanout: Number of children of each node (int)
    - step: Range of each node (int)

Example via Terminal:

```
# /generateHierarchy [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/generateHierarchy
--form hierType=range
--form varType=int
--form attribute=age
--form hierName=age_hier
--form startLimit=1
--form endLimit=100
--form fanout=3
--form step=5
--out ./range_hier_age.txt
```

The hierarchy is stored in the “range_hier_age.txt” text file.

- hierType is range and varType is date
    - startYear: int
    - endYear: int

  The parameters above declare the domain range in term of years e.g. 1945-2016

    - years: Range step in term of years e.g. **5** years
    - months: Range step in term of months e.g. **2** months
    - days: Range step in term of days e.g. **7** days
    - fanout: Number of children of each node (int)

Example via Terminal:

```
# /generateHierarchy [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/generateHierarchy
```

```
--form hierType=range
--form varType=date
--form attribute=date
--form hierName=date_hier
--form startYear=1940
--form endYear=2016
--form years=5
--form months=6
--form days=7
--form fanout=3
--out ./range_hier_date.txt
```

The hierarchy is stored in the “range_hier_date.txt” text file.

- hierType is distinct and varType is string
    - sorting: {alphabetical,random} (string)
    - fanout: Number of children of each node (int)

Example via Terminal:

```
# /generateHierarchy [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/generateHierarchy
--form hierType=distinct
--form varType=string
--form attribute=”Procedure Codes”
--form hierName=codes_hier
--form sorting=alphabetical
--form fanout=3
--out ./hier_p_codes.txt
```

The hierarchy is stored in the “hier_p_codes.txt” text file.

- hierType is mask and varType is string
    - length: The length of mask (int)

Example via Terminal:

```
# /generateHierarchy [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/generateHierarchy
--form hierType=mask
--form varType=string
--form attribute=”Procedure Codes”
```

```
--form hierName=codes_hier
--form length=3
--out ./mask_hier_p_codes.txt
```

The hierarchy is stored in the “mask_hier_p_codes.txt” text file.

- hierType is distinct and varType is {int or double}
    - sorting: {numeric,alphabetical,random} (string)
    - fanout: Number of children of each node (int)

Example via Terminal:

```
# /generateHierarchy [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/generateHierarchy
--form hierType=distinct
--form varType=int
--form attribute=salary
--form hierName=salary_hier
--form sorting=numeric
--form fanout=3
--out ./distinct_hier_salary.txt
```

The hierarchy was saved in the “distinct_hier_salary.txt” text file.

loadHierarchies [POST]

- hierarchies: MultipartFile[], list of hierarchies files in .txt format
- session: Set session id as header parameter

Example via Terminal:

```
# /generateHierarchy [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/loadHierarchies
--form hierarchies=@"path/to/Datasets/Scenarios/Simple Table-Disk based
simple table/data1/distinct_hier_salary.txt"
--form hierarchies=@"path/to/Datasets/Scenarios/Simple Table-Disk based
simple table/data1/distinct_hier_age.txt"

> {"Status":"Success","Message":"Hierarchies have been successfully
> loaded!"}
```

anonymization [POST]

- bind: json string {colName1:hierName1,colName2:hierName2,..,colNameN:hierNameN}
- k: at least k same records (int)
- m: optional, depends on the dataset type.(Sets, RelSet) (int)
- session: Set session id as header parameter

Returns a json string of possible solutions if the executed algorithm is Flash (simple table dataset) e.g. {sol1:{[1,1], unsafe},sol2:{[1,2], safe}} or the anonymized dataset.

Example via Terminal for simple table dataset:

```
# /anonymization [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/anonymization
--form bind="{\"age\":\"age\",\"salary\":\"salary\"}"
--form k=3

> {"Solutions":{
> "sol5":{"result":"safe","levels":"[0, 2]"},
> "sol4":{"result":"unsafe","levels":"[1, 1]"},
> "sol7":{"result":"safe","levels":"[1, 2]"},
> "sol6":{"result":"safe","levels":"[2, 1]"},
> "sol1":{"result":"unsafe","levels":"[1, 0]"},
> "sol0":{"result":"unsafe","levels":"[0, 0]"},
> "sol3":{"result":"safe","levels":"[2, 0]"},
> "sol2":{"result":"unsafe","levels":"[0, 1]"},
> "sol8":{"result":"safe","levels":"[2, 2]"}
> }}
```

Example via Terminal for table with with a set-valued attribute:

```
# /anonymization [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/anonymization
--form bind="{\"age\":\"age\",\"salary\":\"salary\", \"Procedure
Codes\": \"myNumHier\" }"
--form k=3
--form m=2
--out ./anonymized_mixedData.csv
```

The anonymized data are stored in “anonymized_mixedData.csv” file

getAnonRules [POST]

- sol: json string array which contain the anonymize level of each quasi-identifier e.g. [1,2] only for simple table data
- suppresed: boolean for only for simple table data
- session: Set session id as header parameter

The current endpoint is not supported for disk-based data (Disk based simple table) because the algorithm is executed using clustering and similarities that do not generate anonymization rules.

Examples via Terminal for simple table data:

```
# /getAnonRules [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getAnonRules
--form sol="[1,2]"
--form suppressed=true

> {"Status":"Fail","Message":"The solution [1,2] satisfies 3-anonymity
so > it can not be suppressed!"}
```

```
# /getAnonRules [POST]
curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getAnonRules
--form sol="[1,2]"
--form suppressed=false
--out ./anonRules.txt
```

The anonymized rules are stored in the “anonRules.txt” file.

Example via Terminal for table with with a set-valued attribute:

```
# /getAnonRules [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getAnonRules
--out ./anonRules_mixed.txt
```

The anonymized rules are stored in the “anonRules_mixed.txt” file.

loadAnonRules [POST]

- rules: MultipartFile, the of anonymized rules in .txt format
- session: Set session id as header parameter

We suppose that a simple table dataset has already been imported to Amnesia, and we have exported the anonymization rules in the “anonRules.txt” as in the previous example.

```
# /loadAnonRules [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/loadAnonRules
--form rules="path/to/anoRules.txt"
--out ./apply_rules.txt
```

The dataset which was produced from the anonymization rules is stored in the “apply_rules.txt” file.

clearSession [POST]

- session: Set session id as header parameter (setHeader("Cookie", "JSESSIONID="+ SessionID)

Example via Terminal:

```
# /clearSession [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/clearSession

> {"Status":"Success","Message":"Session is cleared!"}
```

Additional endpoints for Flash algorithm

getSolution [POST]

- sol: json string array which contain the anonymize level of each quasi-identifier e.g. [1,2]
- session: Set session id as header parameter

Example via Terminal:

```
# /getSolution [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getSolution
--form sol="[1,2]"
--out ./anonymized_simple_table_data.txt
```

The anonymized dataset is stored in the “anonymized_simple_table_data.txt” text file.

getSuppressPercentage [POST]

- sol: json string array which contain the anonymize level of each quasi-identifier e.g. [1,2]
- session: Set session id as header parameter (setHeader("Cookie", "JSESSIONID="+ SessionID)

Examples via Terminal:

```
# /getSuppressPercentage [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getSuppressPercentage
--form sol="[1,2]"

> {"Status":"Success","percentageSuppress":0.0,"Message":"The solution:
> [1,2] satisfies k=3 anonymity","k":3}
```

For unsafe solution e.g. [1,1]

```
# /getSuppressPercentage [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getSuppressPercentage
--form sol="[1,1]"

> {"Status":"Success","percentageSuppress":0.3,"Message":"To produce a
k=3 anonymity solution, it must be suppressed by 0.3%","k":3}
```

getSuppressedSolution [POST]

- sol: json string array which contain the anonymize level of each quasi-identifier e.g. [1,2]
- session: Set session id as header parameter (setHeader("Cookie", "JSESSIONID="+ SessionID)

Examples via Terminal:

```
# /getSuppressedSolution [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getSuppressedSolution --form sol="[1,2]"
```

```
> {"Status":"Fail","Message":"The solution [1,2] satisfies 3-anonymity
so > it can not be suppressed!"}
```

For unsafe solution e.g. [1,1]

```
# /getSuppressedSolution [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getSuppressedSolution
--form sol="[1,1]"
--out ./anonymized_suppressed_dataset.txt
```

The suppressed anonymized dataset is stored in the "anonymized_suppressed_dataset.txt" text file.

getStatistics [POST]

- sol: json string array which contain the anonymize level of each quasi-identifier e.g. [1,2]
- quasi_ids: String[], list of columns which are quasi-identifiers
- session: Set session id as header parameter (setHeader("Cookie", "JSESSIONID="+ SessionID)

Examples via Terminal:

For age column

```
# /getStatistics [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getStatistics
--form sol="[1,2]"
--form quasi_ids=age

> {"Status":"Success",
> "AnonymizedStats":[
> {"numberOfValues":"142","value":"[102]"},
> {"numberOfValues":"134","value":"[101]"},
> {"numberOfValues":"131","value":"[105]"},
> {"numberOfValues":"122","value":"[103]"},
> {"numberOfValues":"119","value":"[107]"},
> {"numberOfValues":"110","value":"[108]"},
> {"numberOfValues":"104","value":"[106]"},
> {"numberOfValues":"103","value":"[104]"},
> {"numberOfValues":"34","value":"[109]"},
> {"numberOfValues":"0","value":"problematic records"},
> {"numberOfValues":"0","value":"Records with support less than 3%"}],
```

```
> "TotalRecords":999,
> "k":3}
```

For salary column

```
# /getStatistics [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getStatistics --form sol="[1,2]" --form
quasi_ids=salary

> {"Status":"Success",
> "AnonymizedStats":[
> {"numberOfValues":"999","value":"[10011]"},
> {"numberOfValues":"0","value":"problematic records"},
> {"numberOfValues":"0","value":"Records with support less than 100%"}],
> "TotalRecords":999,
> "k":3}
```

For age and salary combined

```
# /getStatistics [POST]
$ curl  -X POST -H "Cookie: JSESSIONID=D6DF59C4A03432B79BC61961F1671EDB"
http://localhost:8181/getStatistics
--form sol="[1,2]"
--form quasi_ids=age
--form quasi_ids=salary

> {"Status":"Success",
> "AnonymizedStats":[
> {"numberOfValues":"142","value":"[102, 10011]"},
> {"numberOfValues":"134","value":"[101, 10011]"},
> {"numberOfValues":"131","value":"[105, 10011]"},
> {"numberOfValues":"122","value":"[103, 10011]"},
> {"numberOfValues":"119","value":"[107, 10011]"},
> {"numberOfValues":"110","value":"[108, 10011]"},
> {"numberOfValues":"104","value":"[106, 10011]"},
> {"numberOfValues":"103","value":"[104, 10011]"},
> {"numberOfValues":"34","value":"[109, 10011]"},
> {"numberOfValues":"0","value":"problematic records"},
> {"numberOfValues":"0","value":"Records with support less than 3%"}],
> "TotalRecords":999,
> "k":3}
```