Neo4J Teamwork Documentation 2020
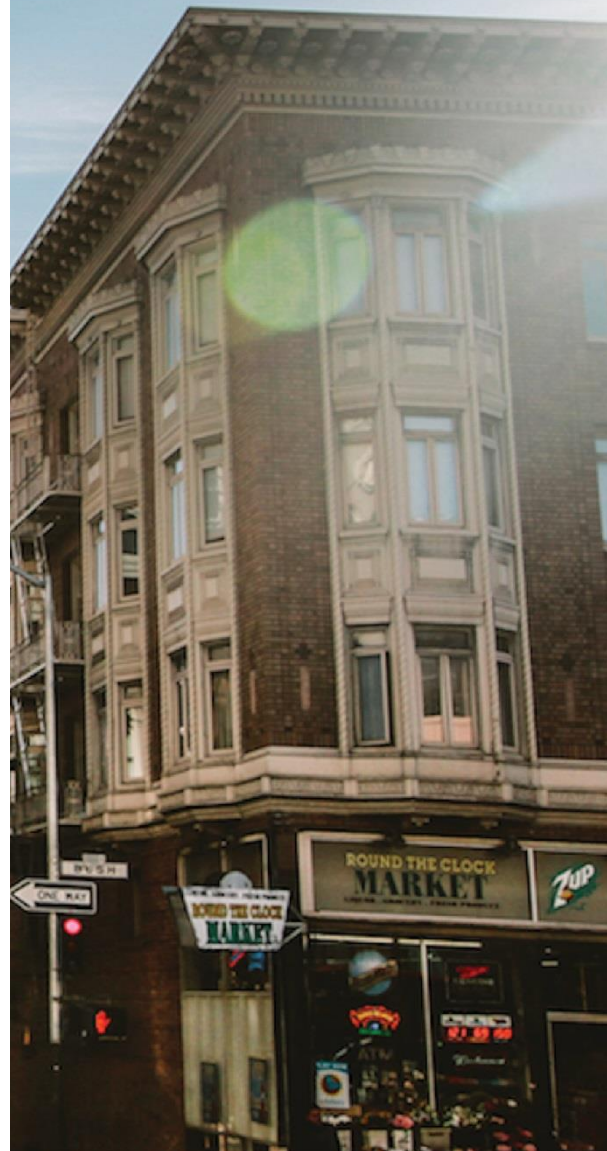
Written by: Óscar Sánchez Campo (UO265078)

Daniel Finca Martínez (UO264469)

Preface

GraphGist

As a complementary resource to this documentation, a GraphGist document has been created to serve as a self-explanatory and interactive documentary of some parts of this project's development. This documentation can be found in the site linked down below. It has been submitted for approval in the GraphGist portal so when checked out there exists the possibility that it doesn't work anymore:

https://portal.graphgist.org/graph_gist_candidates/neo4j_teamwork_graphgist_docs-candidate

In any case, this GraphGist interactive guide can be run from the Neo4j browser's console issuing the following command:

:play
https://portal.graphgist.org/graph_gists/neo4j_teamwork_graphgist_docs/graph_guide

Neo4J .conf file was modified to allow remote hosts to be accessed and so get the guide hosted in neo4J GraphGist servers. Lines 361 and 362 were added to the "neo4j.conf" file. (Followed a tutorial from the GraphGist documentation).

```
356  #***************************************************************
357  # Other Neo4j system properties
358  #***************************************************************
359  dbms.jvm.additional=-Dunsupported.dbms.udc.source=zip
360
361  #comma-separated list of base-urls, or * for everything
362  browser.remote_content_hostname_whitelist=https://portal.graphgist.org,localhost
363
```

GitHub repository

GitHub version control system has been used throughout the whole development process, just in case there is any need to check any extra files, here is the link to the public repository that holds the project files (Just in case deliverable is too big for the Campus):

https://github.com/fincamd/RI_Teamworks (Disclaimer: I have both Information Repositories teamworks held there for the sake of easy file manipulation with my colleague. Files for this concrete project are inside "NEO4J" folder).
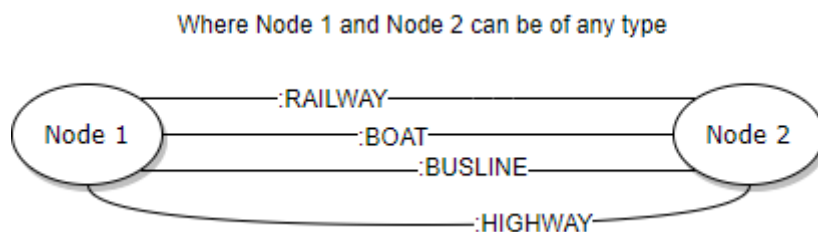
Contents

1.Motivation

Domain: Transport systems across cities, villages and valleys.

This graph database has been created based on the insight of a map. This "map" would display cities, villages and valleys which are interconnected by some conveyance means. These are: buslines, railways, highways and boats.

Where Node 1 and Node 2 can be of any type

As it can be seen from above schema, our domain allows nodes to be interrelated using three types of relationships: Railway, Boat, Highway and Busline. Although they are not represented with an arrow in the schema, they will be represented using a directed arrow in our graph. This is no problem, as they can be traversed both ways when querying. (Image included in repository path: "RI_Teamworks/NEO4J/extra_files")

Each relationship has a different set of attributes. Distributed as follows:

1. Railway:
   a. Line: Indicates the line of the linking railway.
   b. Average ride time: The arbitrarily "estimated" time to go through such relationship.
   c. Price: The arbitrarily chosen price to go through this railway relationship.
2. Busline and Boat:
   a. Average ride time: The arbitrarily "estimated" time to go through such relationships.
   b. Price: The arbitrarily chosen price to go through these relationships.
3. Highway:
   a. Average ride time: The arbitrarily "estimated" time to go through such relationship.
   b. Price: The arbitrarily chosen price to go through this highway relationship.

Moreover, nodes also have a defined set of attributes. No discrimination has been applied to different class nodes. In the sense that all of them contain the same attributes. These mentioned classes are: Valley, City and Village.

To continue, the set of attributes set for all the nodes in the domain is this:

1. City, Village and Valley:
    a. Name: Indicates the name of the place the node represents.
    b. Inhabitants: Stores the number of inhabitants populating the place. (Data taken from the internet. Disclaimer: Could be outdated).

Nodes in our domain might have one class among these: City, Village and Valley. Though our model allows it, some connections would be logically and physically impossible. Not all places represented have a dock. In addition, only some of the possible links have been represented for the sake of simplicity of the database instance.

To conclude this summary of the graph database domain, the names and data have been selected using Asturias's geography. Not every single location of the Principality has been represented because of the same reasons as stated beforehand (Not necessary to be exhaustive).
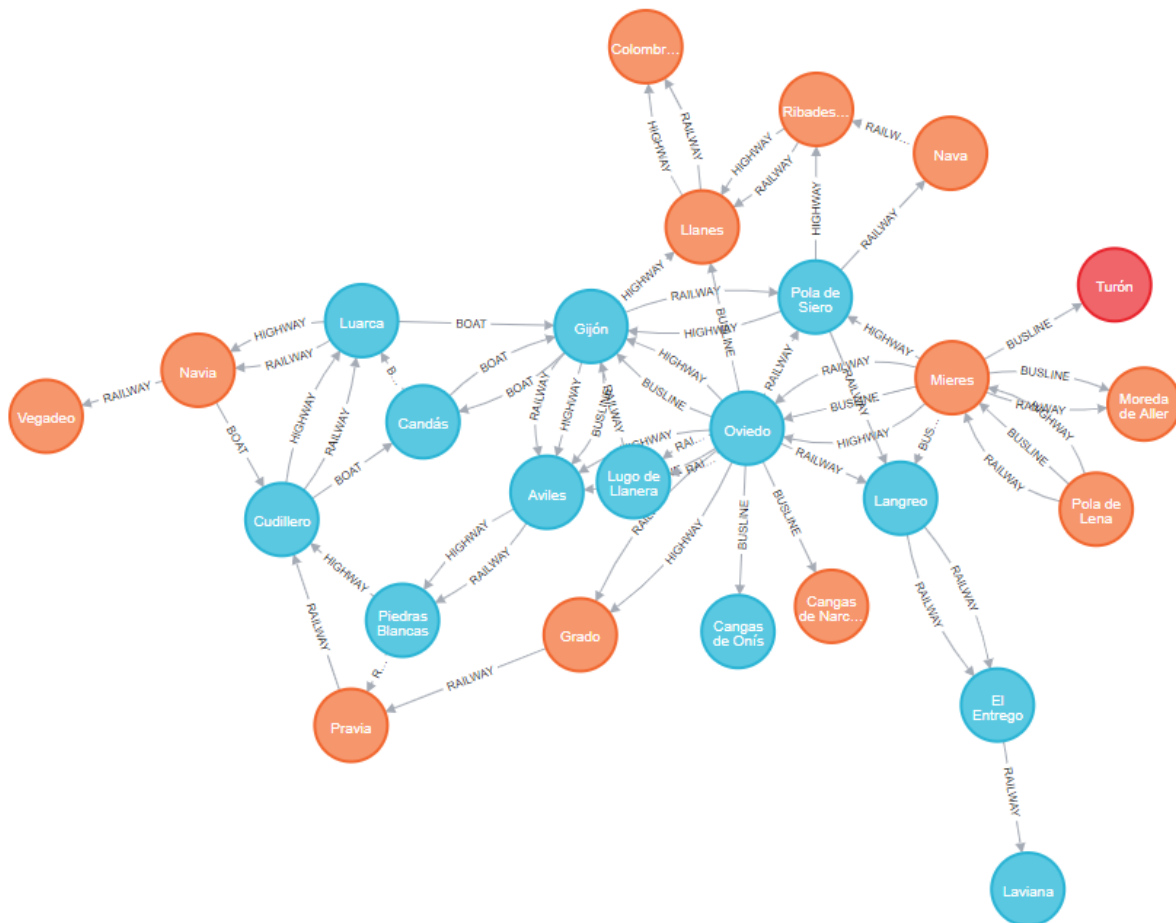
2.Database instance

This database is neither big nor small, but it is still huge when thinking about drawing it by hand. As stated in the task, a subgraph representation is allowed. However, Neo4J provides us with the tools to generate a high-quality image of the complete graph, which will be included down below just in case hand-drawn instance is not clear enough.

All the images taken for this part are included in the repository files under this path:

"RI_Teamworks/NEO4J/extra_files/*"

This database instance features a total of:

1. 26 nodes.
2. 59 relationships.
3. 4 types of relation: Railway, Highway, Busline and Boat.
4. 180 properties.

Above we saw the complete graph database instance obtained from Neo4J export system. Now we can see the hand-drawn subgraph:
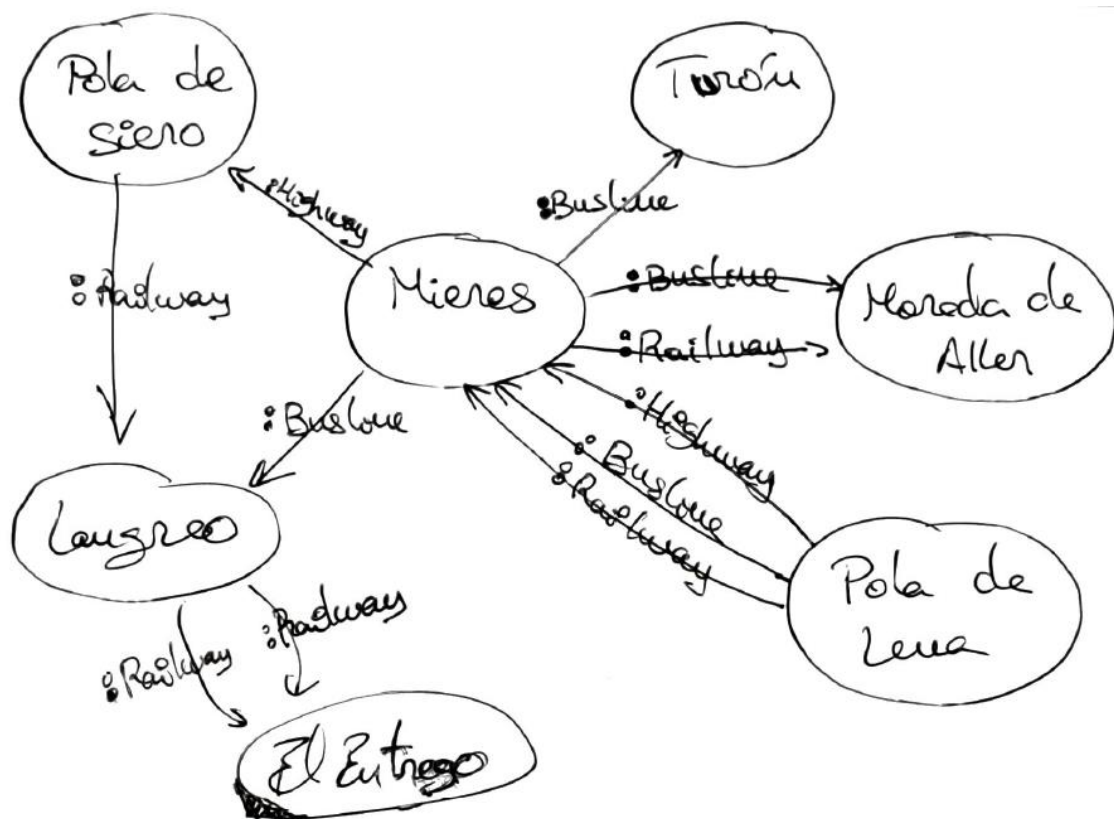
Result obtained after executing the creation script included in the delivery files. Taken from Neo4J browser:

```
$ CREATE (mieres:Village {name:'Mieres', inhabitants: 38428}), (gijon:City

Added 26 labels, created 26 nodes, set 180 properties, created 59 relationships, completed after 44 ms.
```

3.Database creation script

The script used to create the database can be found among the repository files under this path: "RI_Teamworks/NEO4J/task_txt_files/neo4jDBScript.txt"

It can also be read and executed using Neo4J Browser's console issuing the command specified at the preface of this document:

:play
https://portal.graphgist.org/graph_gists/neo4j_teamwork_graphgist_docs/graph_guide

Image taken from the browser itself:
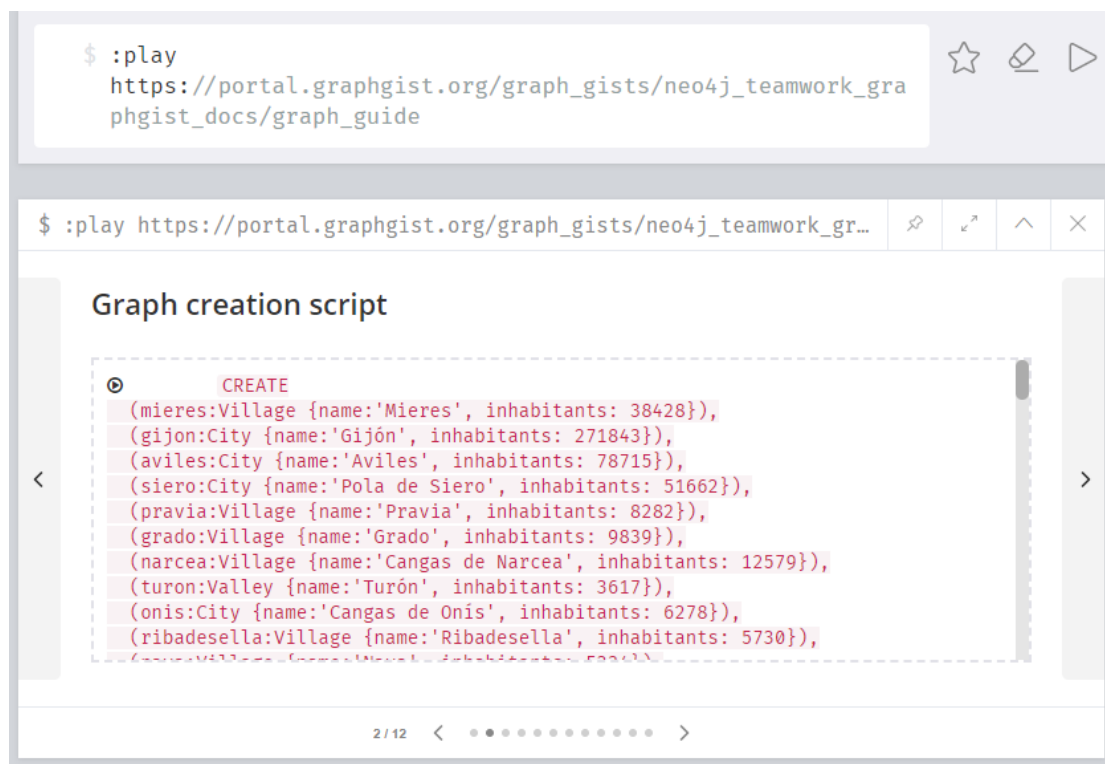
Additionally, under "RI_Teamworks/NEO4J/extra_files", the graph database folder can be found named as: "graph.db".

4.Cypher queries

Three different query difficulty levels have been produced as stated in the exercise description. Begginner, intermediate and advanced.

Every single query has been included in the complementary GraphGist documentation. As mentioned before, these can be read and even executed from the Neo4J browser's console issuing this command:

:play
https://portal.graphgist.org/graph_gists/neo4j_teamwork_graphgist_docs/graph_guide

The queries are divided in three sections, each one has its own description explaining what it retrieves/does, the code making it work and the results are obtained from its execution through browser's interface.

Below some screenshots can be seen from a couple of execution results. They have been obtained from the execution of the GraphGist custom interactive guide:

```
1  MATCH (n:City)
2    WHERE n.inhabitants > 10000
3    RETURN n.name
4    ORDER BY n.inhabitants DESC
```

$:play https://portal.graphgist.org/graph_gists/neo4j_teamwork_gr…

First

Return all cities that have over 10000 inhabitants ordered by number of inhabitants

```
        MATCH (n:City)
  WHERE n.inhabitants > 10000
  RETURN n.name
  ORDER BY n.inhabitants DESC
```

5 / 12

```
$ MATCH (n:City) WHERE n.inhabitants > 10000 RETURN n.name…
```

	n.name
Table	
	"Gijón"
	"Oviedo"
Text	
	"Aviles"
	"Pola de Siero"
Code	
	"Langreo"
	"Piedras Blancas"
	"El Entrego"
	"Lugo de Llanera"
	"Laviana"

Started streaming 9 records after 2 ms and completed after 2 ms.

```
1  MATCH path = (n)-[r1:RAILWAY]-(l)-[r2*1..3]-(t:Valley)
2    WHERE l.name='Langreo' AND n.name =~ "[aeiouAEIOU].*"
3    RETURN n, nodes(path), relationships(path)
4    LIMIT 25
```

```
$ :play https://portal.graphgist.org/graph_gists/neo4j_teamwork_gr…
```

Second

From every possible path between any node that is adjacent to Langreo by train and communicated with a Valley in 1 to 3 steps through any conveyance means, return the departure node, the nodes in the path and the relationships traversed. The first node must start with a vowel
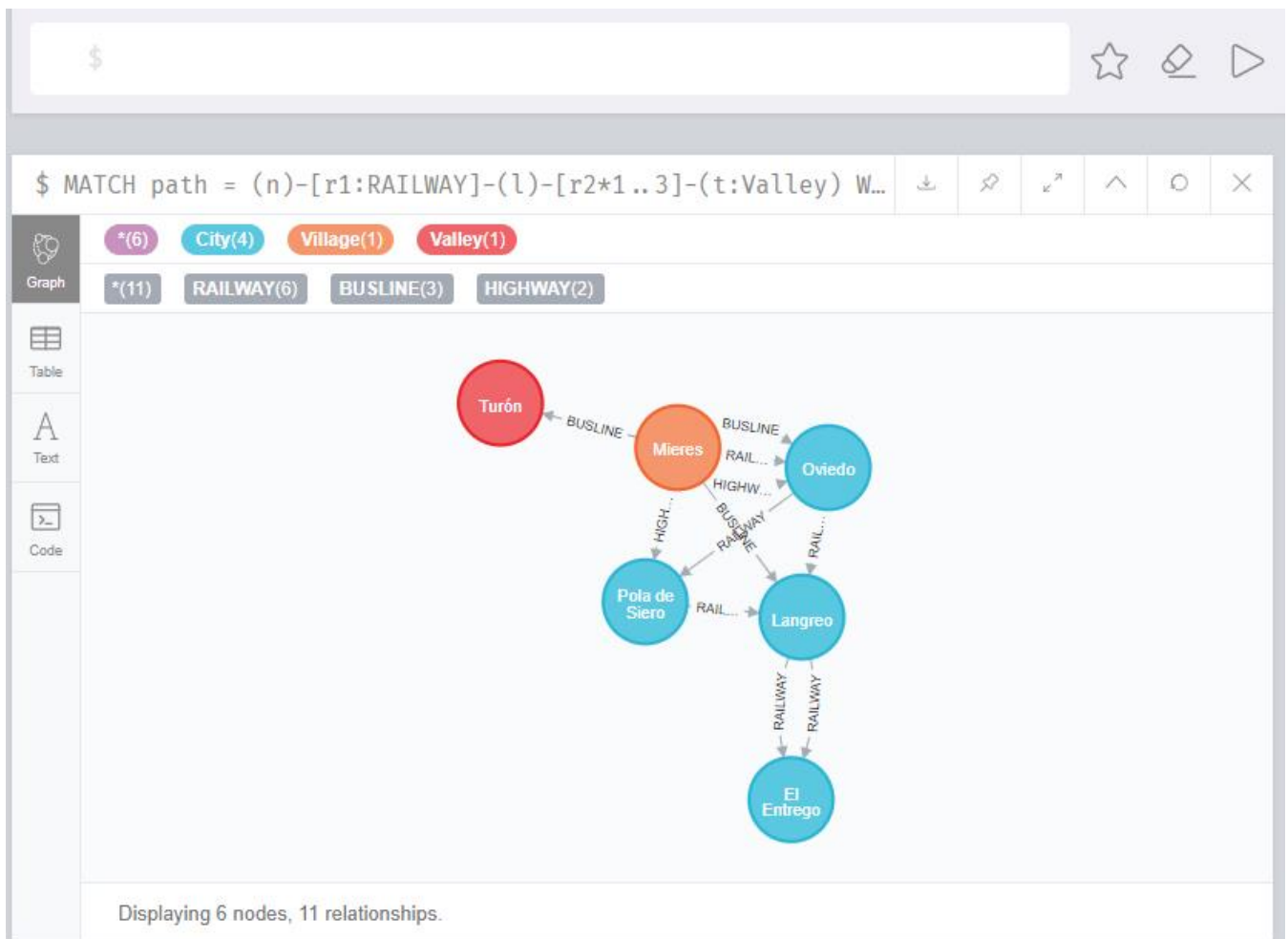
```
       MATCH path = (n)-[r1:RAILWAY]-(l)-[r2*1..3]-(t:Valley)
  WHERE l.name='Langreo' AND n.name =~ "[aeiouAEIOU].*"
  RETURN n, nodes(path), relationships(path)
  LIMIT 25
```
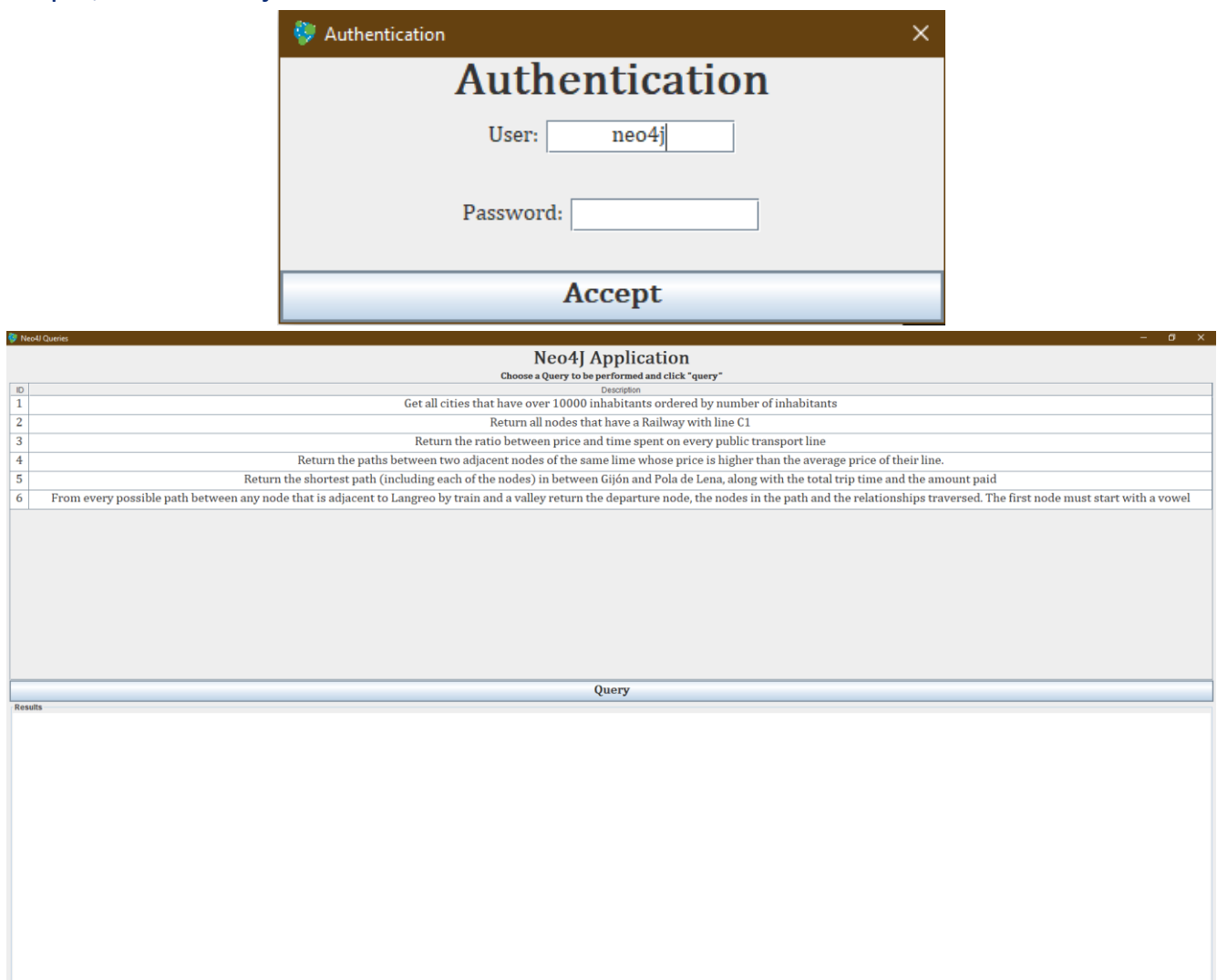
12 / 12 ‹ • • • • • • • • • • • ● ›

5.Scripts: DB Creation and queries

These two documents can be found in separate .txt files in the linked repository under the path: "RI_Teamworks/NEO4J/task_txt_files". Just in case it didn't work by the time this exercise is checked and marked, they will be included as part of the delivered file. As mentioned in the Preface, if for any reason they are not included in the deliverable, it means it was too big for the campus to handle. In such case, the repository will suffice:

1. Database creation script: "neo4jDBScript.txt"

2. Cypher queries packed together with the explanations and the code: "queries.txt"

6.Optional: Application

A Java Application has been developed to gather all queries together and allow the user to execute them over the database through a Graphical User Interface. Although it is simple, it does the job.

Authentication

User: neo4j

Password:

Accept

Neo4J Application

Choose a Query to be performed and click "query"

ID	Description
1	Get all cities that have over 10000 inhabitants ordered by number of inhabitants
2	Return all nodes that have a Railway with line C1
3	Return the ratio between price and time spent on every public transport line
4	Return the paths between two adjacent nodes of the same lime whose price is higher than the average price of their line.
5	Return the shortest path (including each of the nodes) in between Gijón and Pola de Lena, along with the total trip time and the amount paid
6	From every possible path between any node that is adjacent to Langreo by train and a valley return the departure node, the nodes in the path and the relationships traversed. The first node must start with a vowel

Query

Results

As it can be seen in the images included above, the application asks us for the credentials to log into the database management system. Once done, we are offered a selection of queries to be executed.

Displayed in a list fashion, we can select any of them and then press "Query" button to execute the selected one. Once ran and finished, the returned results will appear in the text area just below the "Query" button.

2	Return all nodes that have a Railway with line C1
3	Return the ratio between price and time spent on every public transport line
4	Return the paths between two adjacent nodes of the same lime whose price is higher than the average price of their line.
5	Return the shortest path (including each of the nodes) in between Gijón and Pola de Lena, along with the total trip time …
6	From every possible path between any node that is adjacent to Langreo by train and a valley return the departure node, …

Query

Results
-> (Type: BOAT) -> 0,13 € per min
-> (Type: RAILWAY - Line: F4) -> 0,10 € per min
-> (Type: RAILWAY - Line: C2) -> 0,10 € per min
-> (Type: RAILWAY - Line: F5) -> 0,10 € per min
-> (Type: RAILWAY - Line: F6) -> 0,10 € per min
-> (Type: RAILWAY - Line: C1) -> 0,08 € per min
-> (Type: RAILWAY - Line: F7) -> 0,08 € per min
-> (Type: RAILWAY - Line: F8) -> 0,08 € per min
-> (Type: BUSLINE) -> 0,06 € per min

To execute the application, a .jar file has been included with the deliverable named as: "neo4jdefinitiva.jar". As it has been mentioned in previous parts of this document, if the delivered file doesn't contain such referred file, it can be found in the GitHub repository under the path: "RI_Teamworks/NEO4J/optional_application". There, we can find both the .jar executable file and a .zip file with the code, binaries and eclipse project files just in case it needs checking during the marking process.

Repository link: "https://github.com/fincamd/RI_Teamworks"

The application uses the Neo4j Java Driver, along with Reactive Streams library in order to work properly. Using the methods provided in the driver's documentation we create the connection with the given user and password. Then we provide a method which receives a MyQuery (our interface for queries), creates a transaction, sends it to the database and returns a result. Once returned, it is processed, mapping the different results with the Data Types provided by the driver (Node, Relationship…) and printing them in a proper way. The result is prettier than just running the query. The application is more than extensible. To add new queries, it is as simple as creating a new class that implements the MyQuery interface and instantiating the controller.

7.Optional: GraphGist documentation

First of all, from the statement it is understood that there should be a unique file representing the documentation and that optionally could be in GraphGist format. But due to some formatting means and tools, dexterity in other text processors and other factors, as a team, we decided to include both a .docx and a .adoc (GraphGist base format) files to document this project. We hope it is enough and referenced properly.

The GraphGist documentation is hosted in the neo4j gists portal and is the complement to this document.

Postface

Appendix.Web references and documentations

During the creation process of both this documentation and the GraphGist version, some websites were checked to learn more about the .adoc format and the Neo4J extra existing configurations such as //setup, //console, //table, etc.

Below there is a list of these checked websites:

- https://neo4j.com/graphgist/how-to-create-a-graphgist#_include_a_query_console
- https://neo4j.com/developer/neo4j-browser/
- https://neo4j.com/docs/operations-manual/current/reference/configuration-settings/#config_browser.remote_content_hostname_whitelist
- https://neo4j.com/developer/graphgist/
- https://neo4j.com/docs/