Department of Computer Science and Information Systems
Birkbeck, University of London

Project Report

A Graph-Based Database Application for Cartel Investigations

A dissertation submitted in partial fulfilment of the requirements for the MSc in Computer Science

by

Eugenio Ruiz-Tagle Wiegand

Supervisor: Mr. Peter T. Wood

Abstract

*This document describes the motivations, background research, design, and realisation of an Application For Cartel Investigations (***AFCI***). The application aims to leverage the power of a graph database management system for the purpose of understanding and representing complex relationships between entities in the context of competition law enforcement investigations. The exposition spans from the theoretical and technical concepts that underpin the development of this idea, through the process of database definition and the program's implementation in accordance with conventional software design principles. Finally, this report addresses the main challenges and potential limitations of this endeavour.*

The Project was supervised by Mr. Peter T. Wood.

Table of Contents

I. INTRODUCTION ... 6

I.1. A BRIEF NOTE ON CARTEL INVESTIGATIONS AND THE USE OF SOCIAL NETWORK ANALYSIS ... 6
I.2. MOTIVATION ... 6
I.3. OBJECTIVES OF THE PROJECT ... 7
I.4. REPORT STRUCTURE ... 8

II. BACKGROUND ... 10

II.1 CARTEL INVESTIGATIONS. THE NEED FOR FLEXIBILITY, SCALABILITY AND ROBUST ANALYTICS ... 10
II. 1. (a) Flexibility ... 10
II. 1. (b) Scalability ... 10
II.1. (c) Robust analytics ... 11
II. 2. WHY GRAPH DATABASES? ... 11
II. 2. (a) Intuitive representation of data ... 12
II. 2. (b) Managing complex relationships ... 12
II. 2. (c) Adaptability ... 12
II. 2. (d) Built-in analytical functions ... 12
II. 2. (e) Scalability ... 13
II. 2. (f) Downsides of Graph DBMS ... 13
II. 2. (g) Other NoSQL solutions ... 13
II. 2. (h) Conclusion ... 13
II. 3. BACKGROUND RESEARCH ... 14
II. 3. (a) Data Modelling ... 14
II. 3. (b) Cypher & Neo4j Libraries ... 15
II. 3. (c) Development ... 15
II. 3. (d) The Neo4j Driver ... 15
II. 3. (e) User interface ... 16

III. PROGRAM DESIGN ... 17

III.1. GENERAL SPECIFICATIONS ... 17
III.1. (a) Intended users ... 17
III.1. (b) Functional Requirements ... 17
III.2. TECHNICAL SPECIFICATIONS ... 18
III.2. (a) Graph-based DBMS: Neo4j ... 18
III.2. (b) Programming Language & Framework ... 19
III.2. (c) User interface (UI): ... 21

IV. DATABASE DESIGN ... 23

IV.1. WHY NOT REAL DATA? ... 23
IV.2. DATABASE DEFINITIONS ... 24
IV.2. (a) Firm ... 24
IV.2. (b) Person ... 24
IV.2. (c) Email & Message ... 26
IV.2. (d) Social Media Account (SMA) ... 28
IV.2. (e) Call ... 29
IV.3. EXAMPLE OF DATABASE DEFINITION PROCEDURE ... 30
IV.4. ON THE PAGERANK AND BETWEENNESS CENTRALITY ALGORITHMS ... 32
IV.4. (a) Creation of "COMMUNICATION" relationships ... 32
IV.4. (b) Named-graph projection ... 33

IV.4. (c) PageRank *34*
IV.4. (d) Betweenness Centrality *34*

V. IMPLEMENTATION **36**

V.1. *Development methodology* 36
V.2. *Setting up the program* 36
V.3. *Overview of the program architecture* 37
V.4. *Design principles* 38
V.4. (a) Dependency injection *39*
V.4.(b) Model-View-Controller (MVC) *39*
V.4.(c) Single Responsibility *39*
V.4.(d) Open/Closed *40*
V. 5. *Program architecture* 40
V.5. (a) Entity Classes *40*
V.5. (b) Repository Interfaces *40*
V.5. (c) Data Transfer Objects (DTOs) *41*
V.5. (d) Service Classes *41*
V.5.(e) Controller Class *41*
V.5. (f) CartelAppApplication Class *42*
V.6. *User Interface (UI): Thymeleaf* 42

VI. TESTING **43**

VI.1. *Running the application* 43
VI.2. *Authentication* 44
VI.3. *Creating / Deleting a Person node* 44
VI.4. *Email/Message queries* 46
VI.5. *PageRank* 47
VI.6. *Betweenness Centrality* 48
VI.7. *Unit Tests* 49

VII. LIMITATIONS AND FUTURE DEVELOPMENTS **51**

VII. 1. *Limitations* 51
VII.1.(a) Domain-specific nature *51*
VII.1.(b) Limited queries *51*
VII.1.(c) Neo4j dependence *51*
VII.1.(d) Metrics from "Weight" properties *51*
VII.2. *Future developments* 52
VII.2.(a) Improved user interface *52*
VII.2.(b) Nodes/Properties *52*
VII.2.(c) Other GDS algorithms *52*
VII.2.(d) Large Language Model capabilities *53*

VIII. CONCLUSIONS **54**

REFERENCES **55**

APPENDIXES **58**

APPENDIX 1: USER MANUAL **59**

I. Introduction

I.1. *A brief note on cartel investigations and the use of social network analysis*

Horizontal agreements between businesses to raise prices, share markets, rig bids, or other similar illicit aims, harm consumers and are detrimental to both ethical norms and the global economy. Such kind of corporate criminality, almost always carried out outside public scrutiny, requires Competition Agencies to develop comprehensive investigative strategies. The success of these investigations is contingent on the meticulous analysis of digital evidentiary elements, from emails and social media platform messages to spreadsheets and network traffic, and their interconnections.

As budgets for crime prevention stay on the rise, graph-based database management systems ("Graph DBMS") are increasingly being utilised to enhance cybersecurity defences, risk management, and anti-money laundering measures. Indeed, Graph DBMS showcase features that intuitively allow for the targeting of key individuals or uncovering hidden structures within criminal organisations. This is why companies such as Paypal, MasterCard and Amazon are using this technology to analyse transactional data, customer behaviour and prevent fraud.[1]

With so much interest in the realm of crime prevention, is there a case for the use of Graph DBMS when a complex economic offence has already been committed?

I.2. *Motivation*

The idea for this project comes from an apparently insignificant passage from the International Competition Network's Anti-Cartel Enforcement Manual.[2] The Manual recommends "Social Network Analysis" methodologies for the scrutiny of evidence. The construction of visual networks that interconnect pertinent entities, be they individuals under investigation, intra-or-inter organisational communications or metadata associated to relevant documents, it is suggested, can offer a nuanced understanding of complex relationships that may arise in a given case.

[1] As reported in the following link: https://www.finextra.com/blogposting/24684/harnessing-the-power-of-graph-technology-for-fraud-prevention .

[2] International Competition Network, 2021. 'Anti-Cartel Enforcement Manual (2014, updated 2021), Chapter 3: Management of Electronically Stored Information (ESI) in searches, raids and inspections', pp. 27.

During my time serving as Deputy Head of the Anti-Cartel Division of the National Economic Prosecution Office in Chile, I had the chance to work in dozens of cartel investigations and participate in several webinars and international summits. I noted that most Competition Agencies rely primarily on digital forensic platforms and data analytics solutions like Relativity, Summation, or Nuix for evidence extraction and processing. Without a doubt, these technologies have experienced impressive advances in their capabilities. Yet they typically operate under the relational database paradigm,[3] which although powerful, may have limitations in capturing intricate or non-linear relationships among disparate entities.

I.3. *Objectives of the project*

Throughout this project I have aimed to create an **Application For Cartel Investigations** (henceforth "**AFCI**"), a program that connects with a Neo4j graph-database management system, taking advantage of some of its functionalities to assist users interacting with information that pertains to evidence retrieved in a (fictional, but mimicking a real) cartel investigation.

In embarking on the Project, my objectives were threefold.

First, to construct a prototype graph database that encapsulates the essential relations and data types commonly encountered as evidence in a real-world competition law infringement.

Second, to develop an application that enables users to perform CRUD (Create, Read, Update, Delete) operations and extract insights from a connected Neo4j database, without requiring familiarity with the Cypher Query Language.[4] This is significant considering the fact that lawyers, paralegals, and economists that work in competition law cases are not necessarily trained in database programming languages.

Finally, drawing from my experience in this area of law enforcement, I aimed to explore how working with graph databases in this context can offer valuable results to investigative teams.

[3] For instance, Relativity allows for "Structured Analysis" (identifying similarities/difference between documents or other objects), and Analytic profile for email threads. See: https://help.relativity.com/RelativityOne/Content/Site_Resources/Products.htm .
[4] Cypher is bound to be replaced, sooner or later. Indeed, there is an ongoing international industry committee working on a new formal Graph Query Language (GQL) standard that will be built as a full database language. See information available at: https://www.gqlstandards.org/ .

I.4. *Report Structure*

After the Introduction, this report is structured as follows:

Section II (Background) reflects on the challenges posed by cartel investigations and considers what features are desirable of any information management system that is used in this context. Moreover, some discussion is offered about the potential of Graph DBMS to serve the Project's objectives. Further, a detailed account on the different aspects of the research that was conducted for the purpose of this project is given.

Section III (Program Design) covers the specifications, objectives, and the primary target audience of the project. Subsequently, it explains the functional requirements and the technical specifications of the **AFCI** for setting up the database, the programming language that is used, the framework, and user interface.

Section IV (Database Design) provides the reasons why the graph database formulated for the Project is not composed of data extracted from a real-world case. Then it describes the principles underscoring the model definitions for the creation of a synthetic dataset. Emphasis is given to the key nodes and their relationships within the context of a cartel investigation. Lastly, this section explains the procedure through which the pageRank and betweenness centrality algorithms were projected and written to a named-graph in the database.

Section V (Implementation) presents the development methodology defined for the Project's execution. After some remarks on how the program was set up, this segment delves on the program architecture and the software design principles that were followed during the coding process. The definition of the different objects, and the interaction between the Entity Classes, Data Transfer Object Classes, Service Classes and Repository Interfaces is explained, along with some brief notes on the implementation of the user interface.

Section VI (Testing the program) showcases the working prototype, offering previews of the user interface that illustrate some of the program capabilities and its potential. Next, some paragraphs are dedicated to analysing the outcomes of the application of the algorithms introduced in Section IV, together with some insights elaborated upon the context of the application's objectives.

Section VII (Limitations and future developments) reflects on the perceived limitations of the AFCI and discusses some possible avenues for improvement and expansion, principally in the form of improved interface and user interaction possibilities.

Finally, **Section VII** (Conclusions) summarises the main achievements and challenges of the Project.

II. Background

II.1 *Cartel Investigations. The need for flexibility, scalability and robust analytics*

Cartels are complex, evolving organisations with intricate interpersonal dynamics that often endure changing market conditions, mergers or vertical integrations, entry of new participants or exit of existing members, among other events. And even though microeconomic theory teaches us that cartels are naturally unstable (due to individual incentives to cheat), empirical estimations have demonstrated that, on average, cartels tend to last for several years.[5]

As such, any information management system used in the context of uncovering or prosecuting a cartel agreement needs to be flexible, scalable, and provide robust analytics, for the reasons I briefly outline:

II. 1. (a) Flexibility

Cartel investigations are seldom linear. The investigative process usually demands commanding iterative revisions of hypotheses and dynamically changing trajectories. This creates the need to model relationships, reconfigure them, or even sometimes reorient the direction of the entire inquiry. A flexible environment can help managing the workload without having to go through cumbersome restructuration processes.

II. 1. (b) Scalability

Amidst globalisation and the ascent of multinational enterprises, cartels increasingly operate across borders, affecting multiple jurisdictions. As investigative teams deepen their understanding of the magnitude and scope of the conduct under scrutiny, they will often encounter escalating volumes of data. In this context, a database management system must be able to scale both horizontally and vertically to maintain acceptable performance levels.

[5] Harrington, J. and Wei, Y. (2016) 'What can the duration of discovered cartels tell us about the duration of all cartels?', *The Economic Journal*, 12.

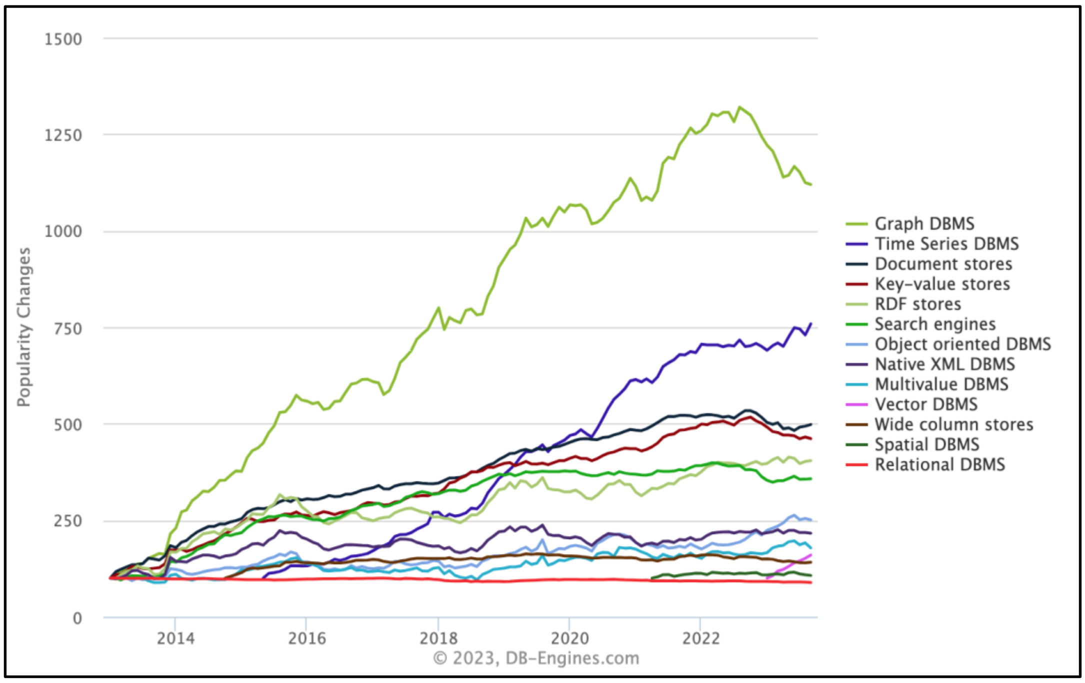
II.1. (c) Robust analytics

To avoid being detected by the authorities, modern cartelists resort to increasingly sophisticated means for coordinating, implementing, and monitoring their illegal activities.[6] At the core of effective cartel investigations resides the capacity to understand the structural logic and the dynamics behind a conspiracy through relational data, that is, interactions and connections between different categories of entities. Presently, analytical tools are less a luxury and more a *prerequisite* for conducting thorough, effective investigations.

II. 2. *Why graph databases?*

The fact that graph-based database management systems have garnered a lot of attention in the last decade is hard to dispute. The following chart shows changes in popularity per category of database model throughout the years, since January 2013:

Image 1
DBMS Popularity Changes per Category[7]



[6] For instance, potential wrongdoers have at their disposal untraceable blockchain-powered "smart contracts" to enforce illegal cartel arrangements. This is only one example of the paramount challenges that new technologies pose to governance of the market economy. See Matsushima, H., 2019. *Blockchain Disables Real-World Governance*. Kyoto University, Institute of Economic Research Working Papers, 1017.
[7] Source: DB-Engines ranking, available at: https://db-engines.com/en/ranking_categories .

The chart shows a clear upward trend for Graph DBMS (at least until mid 2022), while the popularity of relational-based systems, by far the most widely used DBMS, has remained stagnant.[8] In ever-evolving technological landscapes and business requirements, several features of Graph DBMS account for this tendency, some of which, together with its disadvantages, are discussed in the following paragraphs.

II. 2. (a) Intuitive representation of data

Unlike relational databases, which utilise tables to represent data-points and connections, Graph DBMS use nodes, edges, and properties to depict relationships, closely mirroring real-world systems and thereby making the data model more intelligible. By all means, this is not merely an aesthetic perk; it serves as a sort of 'intellectual catalyst' for investigators navigating complex corporate networks.

II. 2. (b) Managing complex relationships

While relational databases can certainly represent relationships, this may come at the expense of complexity and computational efficiency, especially when the relationships are many-to-many, recursive, asymmetric or hierarchical. On the other hand, Graph DBMS allow users to work with rich semantic properties, as nodes and directed edges can be designed to address the handling of complex, multifaceted relationships.

II. 2. (c) Adaptability

New emerging evidence can take the form of different data types and relationships (social media interactions, GPS locations, financial transactions, etc.). A salient feature of Graph DBMS is that they can be used to integrate new sources of information harmonically and with minimal disruptions.

II. 2. (d) Built-in analytical functions

Mainstream Graph DBMS are compounded with data analytics tools which can lead to pathfinding or identifying centrality of entities, among other functions. These tools are built specifically to be applied on relationships, which can offer substantial leverage in grasping the roles of various entities and/or help unearth hidden relationships or communication dynamics.

[8] If we look at DBMS popularity broken down by Database Model, Relational DBMS has 71.7%, while Graph DBMS has less than 2%. See chart available at: https://db-engines.com/en/ranking_categories .

II. 2. (e) Scalability

Graph DBMS provide native storage and processing engines for direct traversal between nodes without having to perform joins or indexing.[9] Also, mainstream solutions allow partitioning single logical databases into smaller contained databases or "shards" to manage larger datasets.[10]

II. 2. (f) Downsides of Graph DBMS

The above-mentioned features should be weighed against the fact that Graph DBMS are not as mature, or widely adopted, as relational databases. As it will be noted later in this work, **AFCI** will require connection and interaction with a Graph DBMS. There is a learning curve to this technology, and experienced database administrators can be hard to find for some organisations, which can definitely represent a setback against the traditional and well-known SQL query language. Other technical considerations, such as Graph Databases being memory-intensive, may also be in order.

II. 2. (g) Other NoSQL solutions

For this Project, the possibility of working with another general-purpose NoSQL DBMS was considered, particularly with MongoDB, a very well-used system that offers the possibility of storing cartel-investigation related data in a JSON-like format (BSON) while allowing the handling of a wide range of data types.

While it is true that the document model can represent complex relationships in a way that is arguably more semantically rich, say, than the flat tables of a relational database, it seems incontestable that MongoDB doesn't have the same native capabilities as a graph database system for traversing and analysing interconnected relationships, or to handle hierarchies between different entities.

II. 2. (h) Conclusion

Notwithstanding the potential downsides and the alternatives available, the decision to work with Graph DBMS appears appropriately grounded on what I think is a comprehensive understanding of the specific use-case at hand: Corporate criminality

[9] However, indexing (often on a single property) is still a standard good practice to optimise query performance for large datasets in Graph DBMS.
[10] See Neo4j documentation on scaling a graph database, available at: https://neo4j.com/product/neo4j-graph-database/scalability/#:~:text=Neo4j%20scales%20out%20as%20data,for%20a%20very%20large%20graph
.

and fraud investigations are by their very nature relationship-heavy, ridden with intricate, changing associations between entities that more often than not are non-linear.[11] Thus, this Project has attempted to direct the power of graph databases to approach this problem.

II. 3. *Background research*

Among the literature underscoring the instrumental role of Graph DBMS in the area of law enforcement, Sadowski and Rathle (2015) have demonstrated that topological analysis of connected data of bank or insurance clients yields more accurate results and fewer false positives than traditional, discrete data analyses. Concurrently, work by Carnaz, Nogueira, and Antunez (2021) operationalised this theoretical framework by using Cypher queries to scrutinise criminal reports in Portugal. PageRank and centrality algorithms, originally developed for network analysis in other contexts, have been successfully adapted in graph database environments to identify key nodes and connections in criminal rings and illicit financial networks.

This subsection provides an overview of the most important research aspects undergone during the design and development of the Project.

II. 3. (a) Data Modelling

Having established that the use of graph databases in this context is well-justified, I was confronted with the task of ingesting a synthetic database[12], which called for scrutiny of the principles and best practices for data modelling in view of the specific entities and relationships pertinent to the Project.[13] Time was devoted to analyse the modelling process, so as to determine the degree of granularity needed in order to create information-rich nodes without indulging in excessive complexities, or to discern where to keep consistency among attributes and relationships to facilitate data retrievals.[14]

[11] This is the reason why fraud-detection is often considered one of the top use cases for graph databases. See, for example, this article from the 'techtarget' webpage, available at: https://www.techtarget.com/searchdatamanagement/feature/Graph-database-vs-relational-database-Key-differences .
[12] The reasons behind the need for a synthetic dataset are set forth in Section IV of the Report.
[13] Robinson, I., Webber, J. & Eifrem, E., 2015. 'Graph Databases'. 2nd ed. O'Reilly, chapter 4.
[14] More details on the data modelling process are provided on section IV of this report.

II. 3. (b) Cypher & Neo4j Libraries

For the purpose of modelling the database, I reinforced my foundations on the use of the Cypher query language.[15] This step was critical, because some of the queries that needed to be performed for data definition resulted to be quite complex. Research was also dedicated to learning about the Neo4j environment and its libraries, such as the APOC and Graph Data Science (GDS) libraries.

II. 3. (c) Development

To understand the frameworks employed by other developers for Neo4j-based applications, I engaged in a review of online forums, consulted the Neo4j Training Series, and examined relevant GitHub repositories.[16] This investigation led to Spring and Spring Boot, as articulated in a later section of this report.[17]

This aspect of the development process conveyed learning about the installation, setup, and adding Spring Boot dependencies to a Maven project. A certain degree of familiarity with Spring containers, which must be instantiated, configured, and assembled according to XML files or annotations, was also necessary. Finally, I got acquainted with the implementation of the Inversion of Control (IoC) principle, commonly referred to as dependency injection, and the Bean Factory interface within the Spring ecosystem.[18]

II. 3. (d) The Neo4j Driver

There is also some essential knowledge to be obtained prior to working with the Neo4j driver to integrate the database with a Java Spring Boot program.[19] For this end, I took advantage of the material provided by Neo4j to build applications with Java[20] and other materials and resources available online.[21]

---

[15] For this purpose, I built on the course materials provided at the Advances in Data Management MSc Module (BBK_COIY025H7_2223).

[16] Available at https://www.youtube.com/@neo4j .

[17] Spring documentation for "Accessing data with Neo4j", available at: https://spring.io/guides/gs/accessing-data-neo4j/ .

[18] For this purpose I used material provided at the Software Design and Programming (COIY062H7) module of the MSc, such as the Spring IoC container documentation, available at: https://docs.spring.io/spring-framework/reference/core/beans.html .

[19] See the Neo4j Java Driver Manual v5.12.0, available at: https://neo4j.com/docs/java-manual/current/.

[20] See Neo4j official tutorial for using the Neo4j Driver, available at: https://graphacademy.neo4j.com/courses/app-java/?ref=guides&_ga=2.70499024.1713065781.1694686358-1831086811.1694686358 .

[21] I took some inspiration from the materials in the Udemy course: "Graph Database: Neo4j with Spring Boot NoSQL: Neo4j with Java and Spring Framework. Node, Relationship with CRUD Operations & AND,

II. 3. (e) User interface

Lastly, I aimed to identify a suitable user interface (**UI**). As will be elaborated upon in the Design section, Thymeleaf emerged as the most fitting Java template engine for the Project's specific requirements.

As part of this area of development, given the fact that Thymeleaf is a server-side Java templating engine for rendering Hyper Text Markup Language views, I consulted and accessed different resources available online in order to grasp the basics concepts on HTML tags, attributes and formatting.[22]

OR, IN Cypher Queries". Available at: https://www.udemy.com/course/graph-database-neo4j-with-java-spring-boot-nosql-cypher-query-graphdb/ .

[22] There are plenty of resources online, for example, the JavaTPoint tutorial available at: https://www.javatpoint.com/html-tutorial .

III. Program Design

III.1. *General Specifications*

AFCI is envisioned as a prototype software for investigative teams, allowing them to work and interact with a graph database and perform CRUD operations without getting entangled in the specifics of the underlying query language (Cypher), while also providing insights that are potentially valuable in the exploratory phase of data analysis.

The **AFCI** is neither intended or designed as an autonomous or standalone program. It rather functions as a case-specific database client, dependent upon an external database server, specifically, a Neo4j Aura instance. The client will display an interface of its own, and, if desired, can be used simultaneously with Neo4j's intrinsic graph manipulation capabilities.

III.1. (a) Intended users

The application was conceived as an analytical tool targeting users engaged in analysis of evidentiary data measurable in terms of its relevance, usually in the context of a legal process, corporate audits or internal investigations.

As explained above, **AFCI** has been modelled on a specific kind of corporate criminality (collusion) and tailored after the needs of specific users inside Competition Law Enforcement Agencies. Yet many of the principles underpinning inquiries of this nature can be amenable to other manifestations of corporate criminality, provided that the underlying database is modified, and some adjustments are made accordingly.

The intended users need not be acquainted with Cypher or any other query language. However, were it to be used by an organisation in a real case scenario, it is reasonable to expect there to be a database administrator to interact directly with the Neo4j instance.

III.1. (b) Functional Requirements

(i) Graph-Database integration

Users should be able to connect to a running Neo4j instance when running the program and be derived to a web-based interface.

(ii) Security access

Only authorised users can access the database connection, by inputting their username and password. Credentials for login to the **AFCI** are distinct from the Neo4j instance user and password.

(iii) Graph-Database management

Users should be able to interact with the database, create, and safely delete new nodes without any knowledge of the Cypher query language. To exemplify the potentiality of this endeavour, **AFCI** will create a new “Person” node if the corresponding properties of a Person in the “Create Person” form are entered and the “Submit” button is pressed. All Person nodes will be listed and can be deleted by pressing a “Delete” button (no Cypher queries needed). Updated nodes will be reflected on the connected Neo4j instance.

(iv) Data Analytics

Users should also be able to generate insights from the data, in the form of predefined queries that are directly accessible through a web interface. Some of the insights will include results from the implementation of centrality algorithms.

III.2. *Technical Specifications*

III.2. (a) Graph-based DBMS: Neo4j

Neo4j has been acclaimed as the “first and best” Graph DBMS by key analysts.[23] It has an active community of developers and data scientists engaged in finding different use-cases and building novel applications with graph databases. It also offers impressive capabilities for visualisation and interactivity with the graph base, both in its desktop and web platforms.[24]

Neo4j is suited with several libraries and toolkits, of which APOC and GDS where particularly relevant for this project:

[23] Rebika, R. & Chettri, P., 2018. 'NoSQL Hands On'. In: P. Raj & G. Chandra Deka, eds. *Advances in Computers*, Volume 109. A Deep Dive into NoSQL Databases: The Use Cases and Applications. Elsevier, ch.6.
[24] Nadime, F. et al., 2018. 'Cypher: An Evolving Query Language for Property Graphs'. SIGMOD’18 *Proceedings of the 2018 International Conference on Management of Data*, pp. 1433–1445.

(i) **Awesome Procedures on Cypher (APOC)** is an extensive toolkit for data manipulation and transformation which 'amplifies' Cypher's native capabilities. It can be deployed into a Neo4j instance and be called directly using Cypher, providing mechanisms to import and export data in diverse formats, execute multiple Cypher queries in the same transaction, and shortcuts to refactor entities, among other functions.[25]

For the Project, I was especially interested in delving into APOC's randomisation functions for large-scale data generation for prototyping information related to a cartel case.

(ii) The **Neo4j Graph Data Science (GDS)** library is a specialised in-memory graph analytics engine, which can be installed as an extension to the database.[26] It provides a robust set of graph algorithms, such as Page Rank, community detection and centrality metrics, which can be called by developers and database administrators on a previously loaded 'named graph'.

In light of the robustness of these features and the integration capabilities of the Neo4j Driver[27], other available solutions like Amazon Neptune[28] or TigerGraph[29] were discarded.

As a word of caution, it is important to bear in mind Neo4j's architecture is primarily conceived for single-instance deployments, and not for distributed datasets. There are also licensing costs to consider if Neo4j is to be used for medium-scaled applications or production environments, which may certainly represent a burden for some organisations.

III.2. (b) Programming Language & Framework

After conducting the research described in the Background Section, it was decided that this project was going to be developed using Java 17 programming language and Spring Boot, a popular framework for creating stand-alone applications.[30]

---

[25] APOC documentation available at: https://neo4j.com/labs/apoc/ .

[26] GDS documentation available at: https://neo4j.com/docs/graph-data-science/current/algorithms/ .

[27] For instance, every year Neo4j organises "NODES", a 24-hour community gathering for knowledge and skill development worldwide, covering the latest topics (such as LLM integration and graph neural networks) and best practices for graph-powered applications. The program is available at: https://neo4j.com/blog/nodes-2023/ .

[28] Amazon AWS documentation available at: https://docs.aws.amazon.com/neptune/latest/userguide/intro.html .

[29]Available at: https://www.tigergraph.com/ .

[30] See Spring Boot documentation available at:

Notwithstanding that based on its extensible, out-of-the box enterprise grade features I was somewhat biased towards working with Java, the option of switching to Python with the Django application framework was also considered. This alternative appeared to be especially compelling when I acknowledged that the best user interface solution for the project was using a web engine: The Python/Django framework is known for its simple syntax, versatility, and it is widely used for the visualisation and managing access of information stored in databases.[31]

However, according to my research, the use of Spring Data Neo4j seems to be the most up to date solution for Neo4j-based application development, and in fact a step-by-step workshop for developers on this topic was recently introduced as part of the Neo4j Training series.[32]

Besides from the community support I found for this solution, the decision for Java/Spring Boot was grounded on the following features:

(i) **Configuration/integration:** Once the Spring dependencies and the Neo4j drivers are appropriately integrated with the project file and the corresponding annotations are declared, integration with the Java environment can be injected directly by modifying the 'application properties' file.

(ii) **Map annotated entity classes and query support:** By extending the Neo4jRepository in repository interfaces in the Java program, it is possible to define nodes, relationships and query methods 'behind the scenes', allowing for easy performance of CRUD operations that are mapped to the Neo4j instance.

(iii) **Authentication plugin:** The integration with Spring Security allows the implementation of an access control framework for Spring-based applications, which is one of the functionalities expected for the **AFCI**.[33]

(iv) **Interface compatibility**: Spring Boot has the Tomcat servlet embedded to it, so the application can be deployed and accessible through HTTP, handling requests and responses without adding configuration overhead.[34]

---

https://spring.io/projects/spring-boot .

[31] Manikanta Vamsi, K. et al. (2021) 'Visualization of Real World Enterprise Data using Python Django Framework', *IOP Conf. Ser.: Mater. Sci. Eng.*, 1042, 012019.

[32]Neo4j Training Series Youtube Channel available at: https://www.youtube.com/watch?v=laAOQLKWy0o&t=1250s .

[33] Spring Security documentation available at: https://spring.io/projects/spring-security .

[34] Spring documentation on embedding web servers available at: https://docs.spring.io/spring-boot/docs/2.0.6.RELEASE/reference/html/howto-embedded-web-servers.html#:~:text=Spring%20Boot%20ships%20by%20default%20with%20Tomcat%208.5.

III.2. (c) User interface (UI):

The **AFCI** was primarily designed to serve as a companion to the Neo4j sandbox, which inherently possesses comprehensive graphical capabilities for data visualisation, allowing the users to easily interact with nodes and edges.

Hence, the UI was devised as a vehicle to showcase the possibility of streamlining some database operations without having to produce code in Cypher, while displaying case-specific results that focus on producing inferences particularly useful in a cartel investigation. This strategic simplification needs not be comprehensive to every functionality of the **AFCI**, but sufficiently workable to let the users experience the power of producing out-of-the-box results from a preloaded database.

(i) Alternatives

The alternatives that were considered for the user interface were the following:

- Window-based desktop interfaces, such as JavaFX or Swing.
- Neo4j Bloom.
- Front end technologies (e.g., React).
- Use a Java template engine to create and process HTML.

(ii) Selection

The criteria of selection were:

- Compatibility with Java-based environments and Spring Boot.
- Ease of integration.
- Minimal complexity in the process of passing data objects from server-side controllers to the client side.
- Not too verbose.

After reviewing several options, it was decided to use Thymeleaf, a straightforward server-side Java template engine designed for rendering web-based content.[35]

[35] See Thymeleaf official webpage and resources available at: https://www.thymeleaf.org/ .

(iii) Why Thymeleaf?

Various alternatives exist that provide server-side templating for Java-based web applications.[36] The decision to use this particular engine was basically the result of installing the corresponding dependency to the Project and engaging in experimentation. Thymeleaf’s syntax is relatively intuitive and accessible, it didn’t carry a steep learning curve that could elongate the development process in detriment of more interesting features I intended to incorporate to the program.

Mostly, the election was made by its natural compatibility with the Spring Boot framework, which results from the mapping of methods in the Controller Class (annotated as '*@Controller*') to the Thymeleaf managed template.

[36] For instance, the Apache Velocity Project. More information available at: https://velocity.apache.org/.

IV. Database Design

The dummy database created for this Project is loosely inspired by actual events: A price-fixing violation that involved online sellers of wall posters and prints, who offered their products through Amazon's UK website.[37] Data entities in this database are inspired by a fictionalised variant of such events, but instead of posters, the sellers here are boxing apparel companies.

In this section, the logic underlying the database design and definitions will be explained. But first, I will address an important question: Why not feed the graph database with data extracted from an existing, real-world cartel case, instead of creating it from scratch?

IV.1. *Why not real data?*

In many developed countries, including the UK, cartel investigation case files are considered to be extremely confidential. Only general facts, either disclosed through statements by the authorities or decisions by the courts, enter the public domain. Even the possibility of obtaining redacted versions of documents pertaining to case files via procedures established in transparency or freedom of information laws, is very limited.[38]

This prohibition is grounded in the fact that such case files normally contain 'commercially sensitive information' and 'sensitive personal information' about businesses and/or individuals, the divulgation of which could not only have negative consequences for the potential infringers, but could also affect third parties such as competitors, suppliers, clients, or others.[39]

---

[37] In broad terms, cartel participants agreed to use a pricing software with the object to avoid undercutting each other's prices. Competition and Markets Authority notes on the case available at: https://www.gov.uk/government/case-studies/online-sellers-price-fixing-case-study.

[38] For example, the UK's Competition and Market Authority, while pledging to the transparency of its procedures, also seeks to maintain (as appropriate) the "*confidentiality of information it obtains in the exercise of its functions*". See: Competition and Markets Authority (UK), 2014. 'Transparency and disclosure: Statement of the CMA's policy and approach'. Available at: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/270249/CMA6_Transparency_Statement.pdf .

[39] According to the shared experiences from several members of the Organisation for Economic Co-operation and Development (OECD) in a recent roundtable. See: Organisation for Economic Cooperation and Development (OECD), 2019. Access to the case file and protection of confidential information. *Working Party No. 3 on Co-operation and Enforcement*, December. pp. 9. Available at: https://one.oecd.org/document/DAF/COMP/WP3(2019)6/en/pdf .

While acknowledging that working with real data would have been the best-case scenario, in view of the author of this report, such impediment did not constitute an unsurpassable hurdle. For the demonstrative purposes of the Project, it is perfectly possible to build a low-scale representation that comprises the main entities present in a cartel investigation.

IV.2. *Database definitions*

Nodes are intended to represent objects in a domain of interest that can be labelled and grouped.[40] In cartel Investigations, business executives are individuals (a "Person") that act on behalf of their employer (a "Firm"). Usually, several people in different roles are subjects of interest for their potential involvement in an infringing conduct. So, our database design must start by defining Firm and Person.

IV.2. (a) Firm

The database was designed to have 3 fictional firms that are vendors of boxing-related products. Thus, the following nodes:

(i) "PunchGear Co."
(ii) "K.O. Inc."
(iii) "TysonJab Ltd."

Each Firm node, as well as every node in the database, has a unique <id> property, a literal Long number that is used to identify and distinguish each node and provide for easy retrieval.

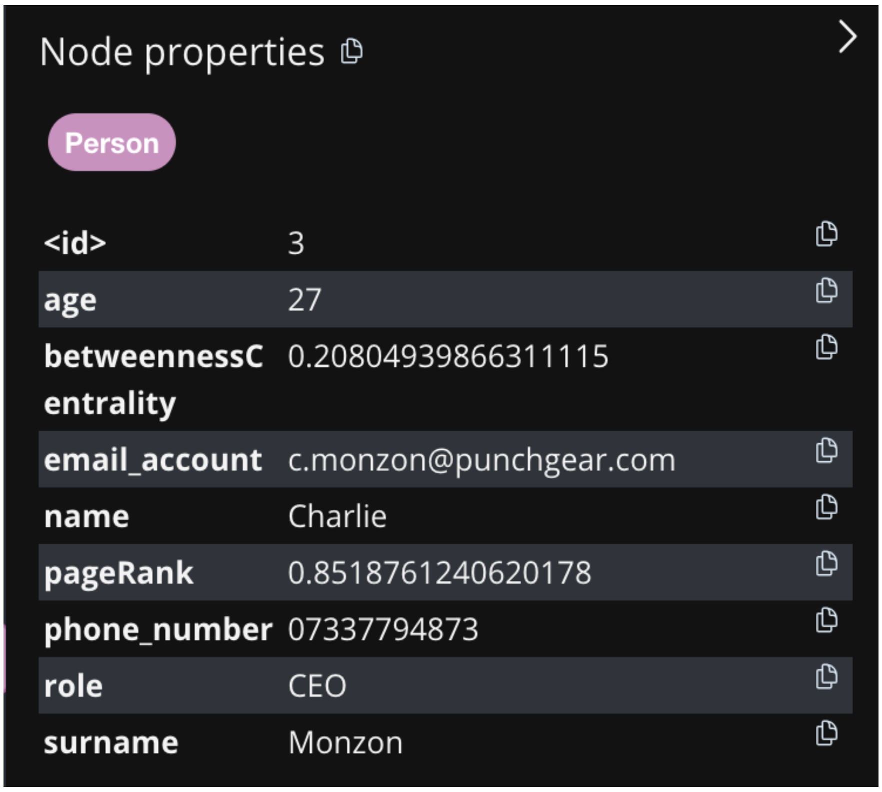
IV.2. (b) Person

Person nodes represent different business executives at different levels of hierarchy, labelled for different "CEO", "Sales VP", "Zone Manager", "Sales Agent". For simplicity, we assumed all three firms follow the same organisational structure.

In the image below, we show the entity attributes pertaining to a random Person node from the Neo4j database:

[40] Robinson, I., Webber, J. & Eifrem, E., 2015. 'Graph Databases'. 2nd ed. O'Reilly, pp. 67.

Image 2
Person node


Node properties

Person

<id>	3
age	27
betweennessCentrality	0.20804939866311115
email_account	c.monzon@punchgear.com
name	Charlie
pageRank	0.8518761240620178
phone_number	07337794873
role	CEO
surname	Monzon

Every person is identified with a name, surname, age, role, phone number and email account, each with a different data type to offer a rich characterisation. Each of these nodes has also a 'pageRank' and a 'betweennessCentrality' figure associated to them. We will delve into these two properties separately, in the final subsection of this chapter.

A relationship was created to represent affiliation or employment from every Person node to a Firm Node:

Person -- [:WORKS]--> Firm.

Similarly, the structure of roles within Persons that work for a Firm is represented by directed MANAGES relationships:

Person -- [:MANAGES]--> Person.

These relationships are directed from the Person in a higher role to other Person(s) in an immediately inferior layer of hierarchy. Hence, the CEO manages the VP of Sales, who in turn manages the Zone Manager, who then oversees the performance of the Sales Agents.

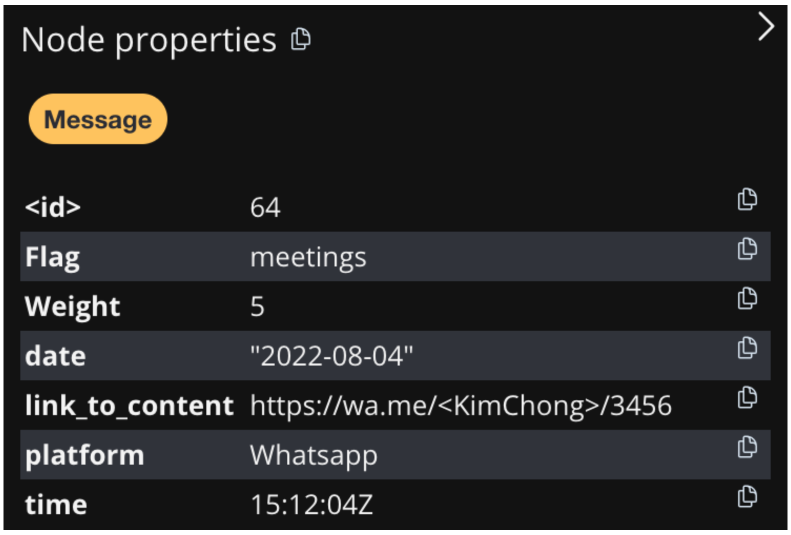
As with other directed relationships in the database, these are intended to express semantically contextualised connections, a quintessential characteristic of the property graph.[41]

IV.2. (c) Email & Message

In any cartel conspiracy, firms and their agents need to communicate to plan and carry out the wrongdoing. Written proofs of these contacts can amount to valuable evidence, provided they have been properly analysed, contextualised, and categorised.

For this prototype it was decided to represent every single Email and Message as an individual node, to provide extensive granularity to the database.

Image 3
Message node

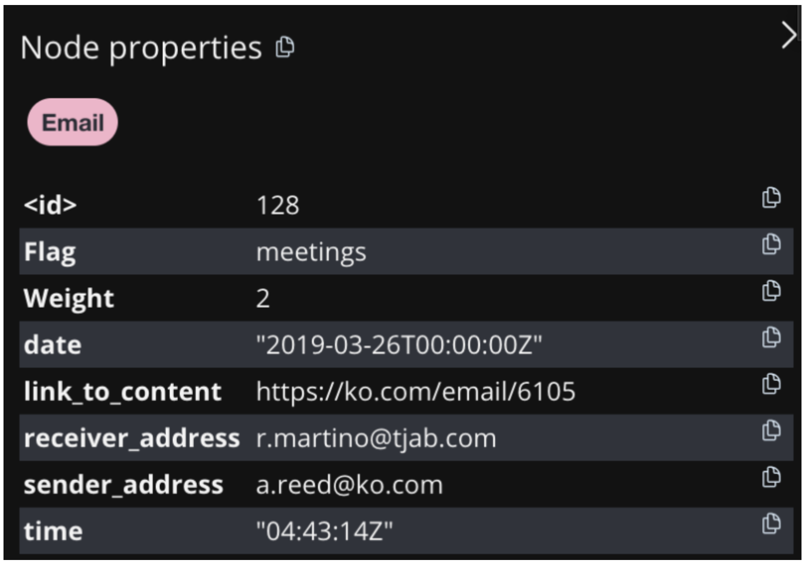

Node properties

Message

<id>	64
Flag	meetings
Weight	5
date	"2022-08-04"
link_to_content	https://wa.me/<KimChong>/3456
platform	Whatsapp
time	15:12:04Z

[41] Robinson et al (2015).

Image 4
Email node


Node properties

Email

<id>	128
Flag	meetings
Weight	2
date	"2019-03-26T00:00:00Z"
link_to_content	https://ko.com/email/6105
receiver_address	r.martino@tjab.com
sender_address	a.reed@ko.com
time	"04:43:14Z"

The database was fed with a few thousand nodes similar to the random Message and Email nodes, the properties of which are exhibited above as examples (Image 4 and Image 5).

Such nodes are compounded with a mix of numerical, textual, and date-time types, to include a standardised format that allows Emails and Messages to be easily sorted.[42]

Both Messages and Email nodes have a '*link_to_content*' property. This is conceived as a unique value that references a specific firm or platform domain. However, in this version of the prototype, the links do not actually point to any specific documents. The purpose is to show the potential of providing direct access to a server-based textual content of each Email or Message represented in the database. While this could be eventually cumbersome to assemble manually, it could significantly enhance the experience of investigators working with Neo4j when navigating the graph DBMS.

Sender and receiver addresses were also included in the Email node properties, with domain-based inference like '@ko.com' or '@tjab.com', which can provide ground to analytical queries to discern between inter-firm and intra-firm communications. Message nodes also identify a specific platform through which the communication was conducted ("Whatsapp", "Telegram", etc.).

[42] Temporal data types in Cypher are based on the ISO 8601 format. Cypher supports date, time and timezone. See the Neo4j developer blog in this subject, available at: https://neo4j.com/developer-blog/cypher-sleuthing-dealing-with-dates-part-1/ .

Furthermore, “Flag” and “Weight” properties were added, which are of capital importance for the program analytics. These properties were modelled mimicking a real-world setting where the relevant communications were previously reviewed and categorised, and the graph database is then set up for the analytical process.

(i) “**Flag**” is a property attributed to a communication marked as significant under a domain of predefined concepts. Labels such as "*meetings*" , "*pricing*" and "*agreement*" represent notions typically relevant in a cartel investigation, making it possible to filter emails or messages that are more likely to contain critical information based on a given set of subjects defined by investigative teams.

(i) “**Weigh**t” is a value assigned to a piece of evidence aimed to measure the importance of a certain Email or Message, aiding in the identification of high-value pieces of evidence. For simplification reasons, weight values are set from 1 to 5, in ascending order of significance.

Finally, relationships SENT and RECEIVED are directed from Person nodes to all written communication nodes, so the following directed connections apply:

Person --[: SENT]--> (Message) <--[: RECEIVED]-- Person
Person --[: SENT]--> (Email) <--[: RECEIVED]-- Person

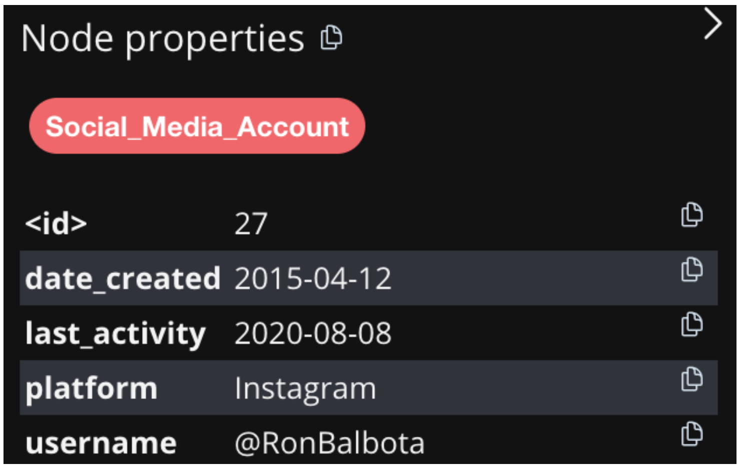
A bi-directional relationship from Person nodes to every Email or Message was created using the common syntax SENT and RECEIVED, thereby simplifying queries aiming to filter any type of written communications.

IV.2. (d) Social Media Account (SMA)

Even though the current version of the **AFCI** does not perform computations in relation to SMAs, given the widespread use of these platforms for law enforcement purposes, they were also identified as nodes.[43]

[43] The use of social media apps for antitrust related conspiracies is mentioned in an internal guidance by the United States Department of Justice, Antitrust Division, 2022. An Antitrust Primer for Federal Law Enforcement Personnel, pp 4, available at: https://www.justice.gov/atr/page/file/1091651/download .

Image 5
Social_Media_Account node


Node properties

Social_Media_Account

<id>	27
date_created	2015-04-12
last_activity	2020-08-08
platform	Instagram
username	@RonBalbota

SMA nodes include properties of "date" type to register the moment when a specific account was created as well as the last activity. This could aid in identifying accounts that were active at the time of the investigation. Likewise, identifying the platform (Instagram, Twitter, etc.) could also be important, and in larger datasets platform usage could lead to material inferences.[44]

Regarding the relationships for this node:

(i) **Person--[: HAS_ACCOUNT]-->SMA** : Links a Person with a social media account (SMA), the use of which is attributed to the latter (though this is sometimes a matter of contention).

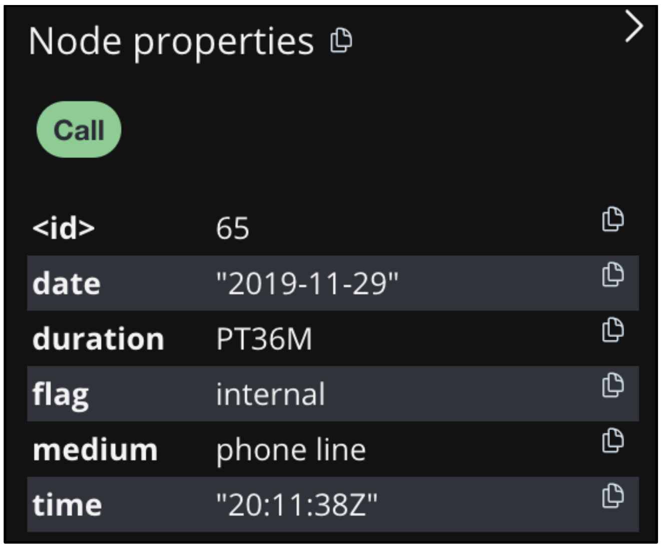
(ii) **SMA--[: FOLLOWS]-->SMA** : Links a SMA that follows (or 'is friends' with another) SMA. This factual relation can in some cases be one-directional (one person follows another) and in others bi-directional (two accounts follow each other).

IV.2. (e) <u>Call</u>

Call nodes gather information on individual calls made through the telephone line or through any platform or app. It is assumed that the Competition Agency has access to information on the existence of the calls from users, but not to the content. This is commonly the case, as new technologies normally use encryption, and it is not usual to have access to recordings derived from phone tapping monitoring techniques.

[44] For example, if all non-conspiracy related personal communications are conducted through one platform and another service is preferred for communications related to the conspiracy, this could lead to important contextual interpretation of the communications therein.

Image 6
Call node


Node properties

Call

<id>	65
date	"2019-11-29"
duration	PT36M
flag	internal
medium	phone line
time	"20:11:38Z"

Call logs are sometimes relevant to prove the existence of contacts within two or more competitors. However, this Project has been focused on providing demonstrative analysis of written communications (Emails and Messages).

IV.3. *Example of database definition procedure*

Before covering the implementation of the algorithms used in this project, it can be illustrative to give one example on the formulation logic that was necessary for the synthetic database to be created.[45]

For this purpose, an example is provided of the Cypher expression used to create a random number of emails (in the range 20-120) between two specific Person executives from the database ("Tony" and "Apollo"):

[45] A text file with a collection of many of the queries used for database definition can be found in Appendix 3.

Image 7

Cypher formulation of a random number (range 20/120) of Email nodes

```
// Define strings defined as "Flags" inside an array

WITH ['meetings', 'pricing', 'agreement', 'other'] AS flags

// Generate a random number of emails (minimum 20, maximum 120)

WITH toInteger(20 + rand() * 101) AS randomEmailNumber
UNWIND range(1, randomEmailNumber)

// Using this information, generate a random date for each Email (mailDate)

WITH flags, randomEmailNumber,
 datetime({
year: toInteger(rand() * 4) + 2019,
month: toInteger(rand() * 12) + 1,
day: toInteger(rand() * 31) + 1
 }) AS mailDate

// Create Email nodes with random attributes
// Sender/Receiver email addresses will remain fixed, addresses were added manually

CREATE (e:Email {
 date: mailDate,
 time: time({hour: toInteger(rand() * 24), minute: toInteger(rand() * 60), second: toInteger(rand()
* 60)}),
 sender_address: 't.josher@ko.com',
 receiver_address: 'a.reed@ko.com',

//Set weight values: random number from 1 to 5
//However, if the flag is "agreement", the value is set to the highest (5).
//link to content integer identifier also works with a randomised function

 Weight: CASE flags[toInteger(rand() * size(flags))] WHEN 'agreement' THEN 5 ELSE
toInteger(rand() * 5) + 1 END,
 Flag: flags[toInteger(rand() * size(flags))],
 link_to_content: 'https://ko.com/email/' + toInteger(rand() * 10000)
})

// Pass the newly created email node (e) to the next part of the query

WITH e

// Match Person nodes with specific names 'Tony' and 'Apollo'

MATCH (sender:Person {name: 'Tony'}), (receiver:Person {name: 'Apollo'})

// Create SENT and RECEIVED relationships
//Both relationships are directed to the email node (e)

CREATE (sender)-[:SENT]->(e)<-[:RECEIVED]-(receiver);
```

Making the corresponding adjustments, this process was iterated for other Person nodes communication interactions throughout the database.

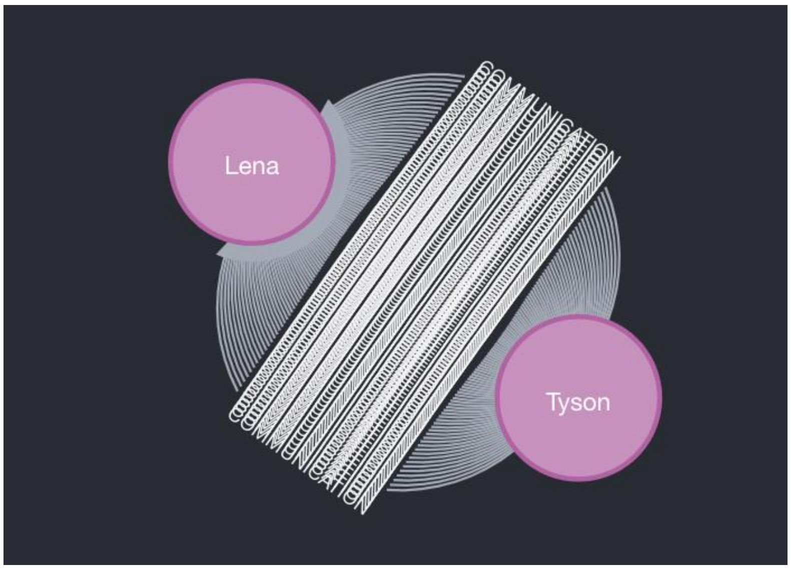
IV.4. *On the pageRank and betweenness centrality algorithms*

One of the objectives set out for this Project is to show the advantages that can be accrued by making use of the utilities available at the Graph Data Science (GDS) library in addition to the core Neo4j capabilities, and to explore some of the graph-related intuitions that can be derived from them to the benefit of a competition law infringement inquiry. For this purpose, we will be exploring two graph algorithms techniques that aim to tackle the "Node Importance" in terms of critical nodes and information spread within the network.[46]

IV.4. (a) Creation of "COMMUNICATION" relationships

For the purposes of the implementation of said algorithms, direct relationships between nodes needed to be analysed, as opposed to indirect relationships between two Person nodes that are connected to an Email or a Message node through SEND and RECEIVE relationships. This is because these algorithms are fundamentally designed to assign a metric to the significance or to evaluate the centrality of direct connections in a graph, without intermediary nodes. Hence, it was necessary to create a new set of relationships in the database, which were identified as "**COMMUNICATION**":

Image 8
Communication relationships between two Person nodes



[46] Scifo, E., 2020. Hands-On Graph Analytics with Neo4j. Packt Publishing.

(i) COMMUNICATION relationships directly connect two Person nodes in every instance there is an Email or Message interchange between said Persons. This entails that direct relationships among two nodes grow constantly with the number of Emails or Messages sent and received among them.[47]

(ii) Every COMMUNICATION relationship carries the "Weight" property of the corresponding Email or Message node it represents. This property is incorporated into the named graph, making it possible to calculate the importance attributed to every single communication from nodes.

IV.4. (b) Named-graph projection

As mentioned in the Background section, for any algorithm to run it is necessary to project a named-graph by calling the corresponding GDS library function. A 'named graph' is conceptually akin to a materialised view: an in-memory abstraction layer that captures a subset of nodes and relationships previously defined through Cypher queries.[48] Once created, the named graph serves as the computational basis for graph analytics within the GDS library.

In a graph projection call, it is necessary to identify:

(i) The nodes we are interested in projecting.
(ii) The relationship type for the projection.
(iii) The orientation of the relationships (directed or undirected).
(iv) The properties of the relationships included in the named graph.

By referencing the name of a previously created graph, it is possible to execute algorithms that yield results restricted to this specific subset of entities and relationships defined therein. With this objective, a "**communication_graph**" was created including all Person nodes and their COMMUNICATION relationships in any direction, including the "Weight" property of said relationships.

[47] For simplification, we have assumed that every email or message only involves two Persons on the receiving end (no CC or CCOs).
[48] Neo4j documentation on projecting graphs available at: https://neo4j.com/docs/graph-data-science/current/common-usage/projecting-graphs/ .

Finally, another function call was done for the corresponding algorithms to be applied by writing the “pageRank” and “betweeness_centrality” properties, which are viewable in the properties of the Person nodes.[49]

IV.4. (c) PageRank

PageRank uses a recursive formula to weigh the transitive influence of a node on the communication network. A Person, based on the weight of its incoming weighted connections (in this case, Emails and Messages), is assigned a pageRank score, which is then successively adjusted by adding to the equation a fraction of the PageRank scores, which are calculated over the relative importance of the nodes connecting to said person.[50]

In essence, each Person's importance compared to other Persons. For example, John's PageRank score will tend to be higher if the total PageRank score of other Persons communicating with John is also high. The implementation of PageRank is an iterative process, where the rank of a given node is updated repeatedly based on the rank of its neighbours in the previous operation.[51]

For the implementation of this algorithm, a “damping ratio” or “damping factor” was set into the computation - which, in the initial formulation of the algorithms, can be represented by the ratio of random clicks. In our case, this can take the form of the likelihood of a COMMUNICATION relationship having an overestimated “Weight” property, as investigators can be overly enthusiastic about the real importance of a given piece of evidence. Following convention, it was set to 0.85.[52]

IV.4. (d) Betweenness Centrality

The "Betweenness Centrality" algorithm quantifies the brokerage role of each Person in the context of a network of communications within the nodes.[53] The idea is to rank the executives showing their individual scores, so those who are more influential or

[49] This means the function should be called and properties should be written into the nodes when the algorithms are run.
[50] Neo4j documentation on the pageRank algorithm: https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/ .
[51] Scifo, E., 2020. 'Hands-On Graph Analytics with Neo4j'. Packt Publishing. In our implementation, 20 iterations were performed (which is the default number provided in the GDS library.
[52] Brin, S. & Page, L., 1998. 'The Anatomy of a Large-Scale Hypertextual Web Search Engine'. *Computer Networks and ISDN Systems 30*, pp. 107-117.
[53] This is a modified version of the canonical problem of solving the “single-source shortest-paths (SSSP)” between nodes. Brandes, U. & Pich, C., 2007. 'Centrality Estimation in Large Networks'. *International Journal of Bifurcation and Chaos*, 17(07), pp.2303-2318.

“central” within the network would be ranked higher, as more weighted-valued information flows through them in the form of Emails and Messages.[54]

Betweenness Centrality it is used to identify those individuals who have the greatest influence on the flow and structure of the network. This is done by computing the “weighted shortest path”, or more accurately, the “heaviest path” between nodes. The GDS employs a sampling technique that adjusts the number of source nodes to optimise execution time.[55]

As deployed in the GDS library, betweenness centrality can also compute the centrality of distribution, that is, how the scores are distributed among different nodes, of which it can be inferred whether there are particularly important nodes in the database.[56]

The results of the implementation of these algorithms will be shown in the “Testing” section of this report.

[54] Brandes, U., 2001. 'A Faster Algorithm for Betweenness Centrality'. *Journal of Mathematical Sociology*, 25(2), pp. 163-177.
[55] Neo4j documentation on the betweenness centrality algorithm: https://neo4j.com/docs/graph-data-science/current/algorithms/betweenness-centrality/ .
[56] This property is not associated with any particular node, and it is not analysed in the program, yet it could be taken into account in a real-world scenario.

V. Implementation

V.1. *Development methodology*

For this project the methodology was based on the principles of Iterative Agile Development. However, it's worth noting that the traditional mechanism of incorporating customer feedback to the development sprints was excluded, given the case-specific nature of the Project.[57]

Initially, a prototype Neo4j synthetic database was designed in a markdown text file.[58] This was followed by populating it through Cypher queries, a process that underwent through several successive refinements. Once a skeleton of the database was completed, the initial codebase for the **AFCI** was added.

As the Project made progress through the implementation of entities, services, and other resources, both the database schema and the application were incrementally updated in parallel, with each iteration contributing to refine the code or to add new functionalities.[59] Once the basic architecture of the Spring Boot project was working, the user interface was introduced by installing the appropriate files to the project.

Overall, this strategy was directed to attain a basic level of coherence between the data layer and the application logic, while also adhering to certain consistency by repeatedly checking that the queries and processes developed in connection with Neo4j were outputting the expected results.

V.2. *Setting up the program*

The Project was set up using the IntelliJ IDE as a Spring Project, using Java 17 and Apache Maven for project management.[60] The first step was to configure the Project Object Model file (pom.xml).[61] This was achieved by adding the respective dependencies for the most updated versions of the Neo4 Java driver, the Spring Boot framework, and Spring Security.

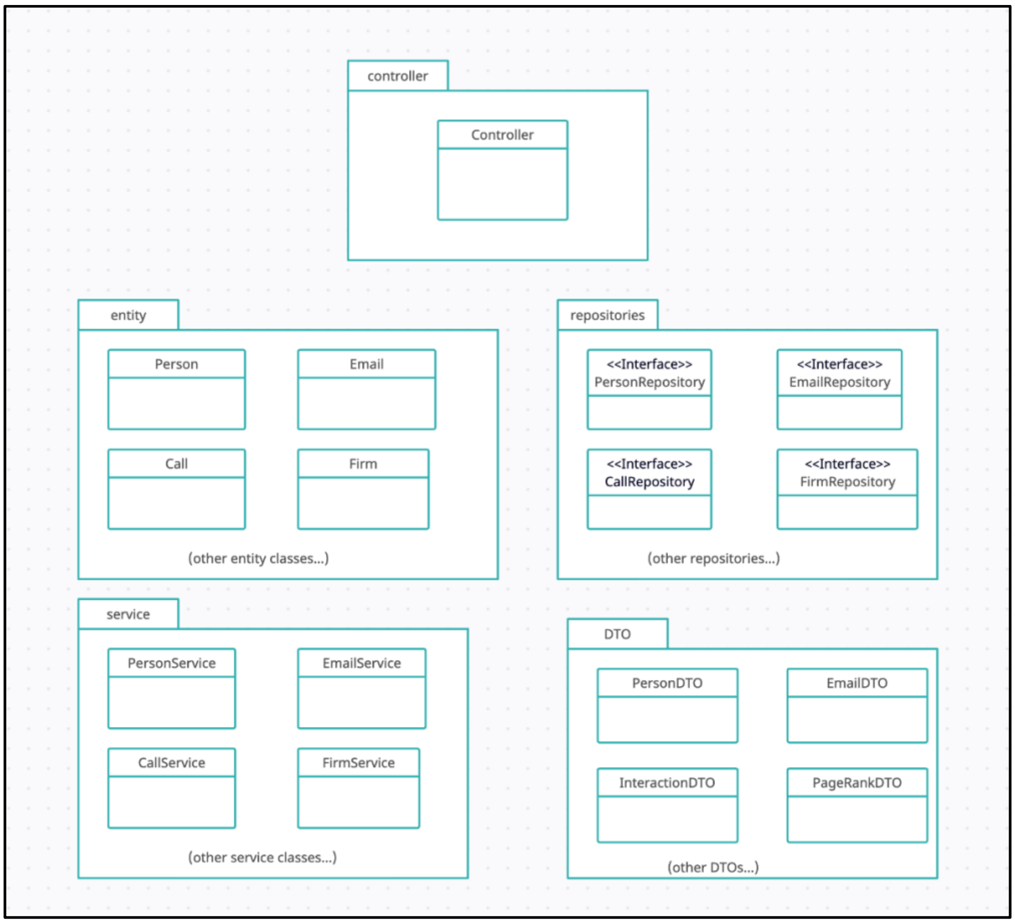
[57] Salo, O. & Abrahamsson, P., 2007. 'An Iterative Improvement Process for Agile Software Development'. *Softw. Process Improve. Pract.*, Wiley.
[58] The initial database definition is a README.md file that is part of the Appendixes of this Report.
[59] Needless to say, this strategy was not always successful as intended, and several iterations were completely redone.
[60] See the Maven Apache documentation at https://maven.apache.org/ .
[61] Introductory guide to the POM file is available at: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html .

Next, it was necessary to define the application properties providing the authentication details for the Spring Neo4j and for the Spring Security credentials, which are requested to the user once the program runs.[62]

V.3. *Overview of the program architecture*

The following diagram shows the main program architecture of the different classes and the corresponding packages to which they belong, showing the organisation of the different classes:

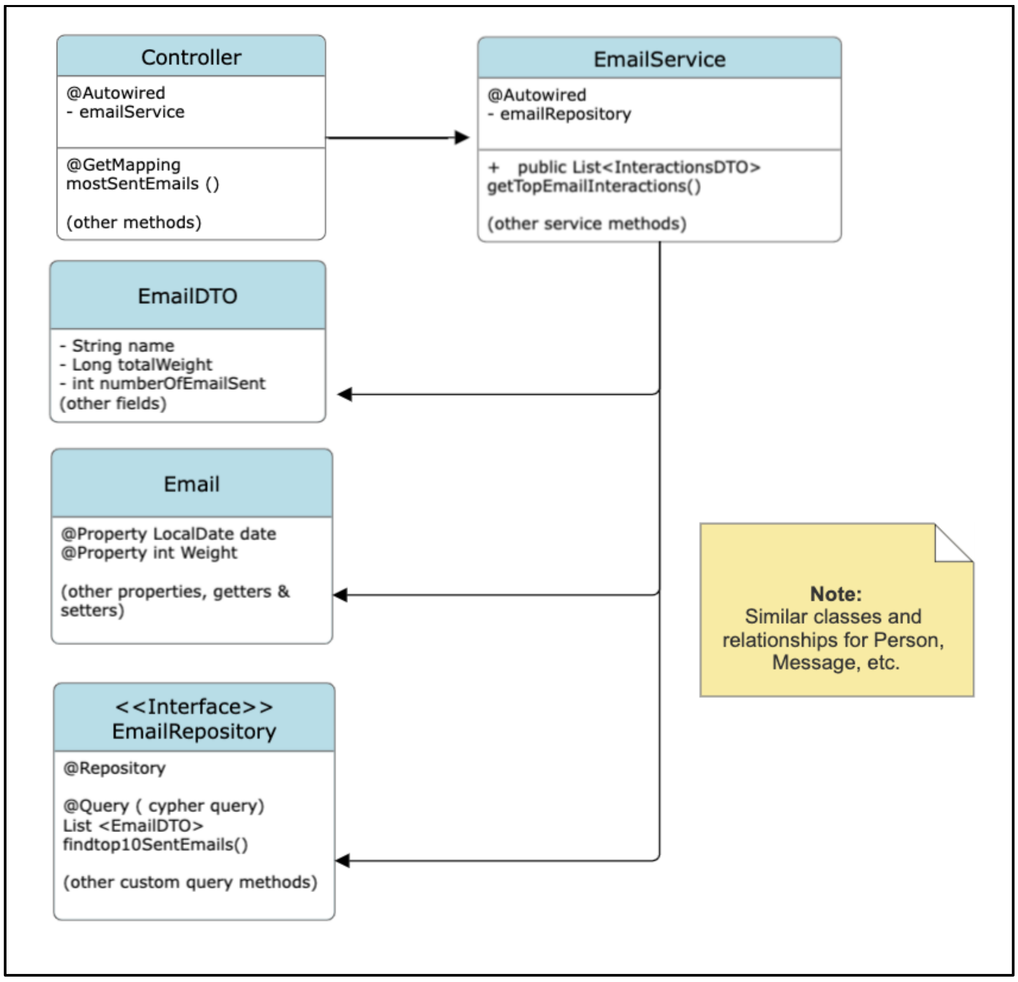
Image 9
AFCI Package Organisation



Similarly, Image 10 below shows an UML diagram showing the different components of the AFCI architecture, converging on the relationships and roles of each one of them. For clarity, the diagram focuses on only one of the entities and its related classes and interfaces (Email), though the same associations apply to other entities:

[62] Application properties are located in src/main/resources.

Image 10
UML Diagram (Email-related entities and Classes)



After presenting the design principles that guided the architecture design process, an in-depth explanation of these objects and their interaction is provided in subsection (V.5).

V.4. *Design principles*

In the following paragraphs, attention is drawn to the principal design paradigms that were followed for the design and deployment of the code in the Project.

V.4. (a) Dependency injection

Inversion of Control, also known as the Dependency Injection principle, is supported natively by Spring framework[63], and in few words, is manifested by the existence of Service Classes and Repository Interfaces[64] that are "injected" at runtime using the '*@Autowired*' annotation (See image 10 above).[65] This annotation redirects the compiler to a bean of the corresponding type within the Spring container.[66] Thanks to this model, components of the program are decoupled thereby avoiding excessive boilerplate code that would be the result of instantiation of repository classes for data access operations.

V.4.(b) Model-View-Controller (MVC)

Three separate components are distinguishable in this architecture: A "Model", in charge of handling data, the "View", that displays the user interface for the end users, and the "Controller", which is a map-like object in charge of handling the data that will later be displayed to the users through the View.[67]

V.4.(c) Single Responsibility

The architecture of the **AFCI** aims for separation of concerns between the different program modules, so each of them has no more than one reason to be modified.[68] Concerns among the program module are divided and encapsulated: Entity Classes represent the domain objects, DTOs serialise the data, Service Classes contain the methods that implement the business logic, while the Repositories control access to the connected database.

[63] See documentation for the IoC container available at: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html .

[64] Fowler, Martin. 2004. 'Inversion of Control Containers and the Dependency Injection pattern'. Available at: https://martinfowler.com/articles/injection.html .

[65] "*Marks a constructor, field, setter method, or config method as to be autowired by Spring's dependency injection facilities*." See Spring documentation. https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/annotation/package-summary.html .

[66] Discussion of beans exceeds the scope of the Project. In short, beans can be defined as the essential building block objects of the Spring framework.

[67] See the interaction between HTML Templates and Spring MVC, available at this article: https://medium.com/@nandaras0103/spring-mvc-and-thymleaf-fa48c000f68a .

[68] Martin, Robert (Uncle Bob), 'The Single Responsibility Principle', in Henney, K. (ed.), 2010. *97 Things Every Programmer Should Know*. O'Reilly, pp. 152.

V.4.(d) Open/Closed

This architecture is extensible for adding new types of entities, relationships, and queries pursuant to the needs of the clients, with minimal changes to the basic architecture of the program. For example, a new node to analyse financial transactions between Firms or Persons could be defined as “Financial_Transactions”, adding the corresponding Repository interface, Service and DTO classes and incorporating a few lines of code to the Controller class and the HTML file, without need to change the existing classes or to override methods.

V. 5. *Program architecture*

V.5. (a) Entity Classes

Entity classes define the different entities like Person, Message, Email, etc. with their respective attributes in the form of fields, getter, and setter methods. These entities are annotated with Neo4j-specific annotations like '*@Node*' and '*@Property*', thereby serving as an object mapping module which, in simple words, translates the Java objects into the entity elements present in the graph database.

Crucially, these classes enable the Neo4j repository to perform CRUD operations seamlessly in the Java application - provided that the entities are defined in terms that perfectly match the nodes and properties that already exist in the database.[69]

V.5. (b) Repository Interfaces

The Repository interfaces (annotated '*@Repository*') extend the Neo4j Repository and act as the isolated data-access layer that connects the program to the underlying Neo4j graph database.[70] The Neo4j repository includes plenty of methods for performing CRUD operations, interacting with the database, and performing Cypher queries, according to the needs of the different entity types.

[69] The mapping from Neo4j to Java will not recognise and entity and will not perform CRUD operations if there are inconsistencies with the property definition, for instance, if the “surname” property in the Person entity class is defined as “surName”, or as an integer instead of a String.
[70] See Spring Data Neo4j documentation, available at: https://docs.spring.io/spring-data/data-neo4j/docs/current/api/org/springframework/data/neo4j/repository/Neo4jRepository.html .

Indeed, Cypher queries can be incorporated inside quotes in the form of abstract methods, by using the annotation '*@Query*'. In this design, such methods return a Data Transfer Object (or a List of such objects).

V.5. (c) Data Transfer Objects (DTOs)

Given that any modification to the entity classes could imply an alteration of the data layer (which may provoke undesirable results), these classes define a subset of attributes for manipulation without affecting the data model.

Not only entities amenable to modification are defined as DTOs, but also the implemented algorithms. PageRankDTO and BetCenDTO represent specialised DTO classes that are defined to query the score associated with each as a Double value (of data type Double in Java).

V.5. (d) Service Classes

These classes (annotated '*@Service*') work as a middle-layer between the repositories, the DTOs and the controller, and oversee the implementation of the business logic. They usually have only one instance variable - the corresponding repository that is injected with '*@Autowired*'.

Service Classes include methods that interact with the data layer (repositories) invoking the respective methods therein to perform different functionalities, such as finding the person that sent the greatest number of Emails or creating a Person and saving it into the database.

V.5.(e) Controller Class

The Controller Class, duly annotated for classpath scanning, is essential for handling GET and POST HTTP requests ('*@GetMapping*' and '*@PostMapping*'). [71] It is injected with repository dependencies and Service classes to receive input and manipulate the data objects (using the Model interface)[72] to render the view for the user, according to the HTML mappings.

[71] Spring documentation on the "Annotation Interface Controller" available at: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/stereotype/Controller.html .

[72] The Model component is an interface that defines a placeholder for data attributes, enabling data to be transferred between the Controller and the View. The documentation for this interface is available at: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/ui/Model.html.

For example, in the method *getTop5PageRankedPersons(*Model model), the model adds the list of top 5 PageRanked persons as an attribute to the Model object, allowing the frontend users to access the new (or updated) data.

V.5. (f) CartelAppApplication Class

Lastly, the CartelAppApplication Class is the entry point to the Spring Boot application. It sets up the repositories by scanning the different components of the application in the same package, while holding the actions to be executed when the program runs, like pointing the user to open the web browser at the default local port.

V.6. *User Interface (UI): Thymeleaf*

The user interface was deployed by adding a template file (in this case the document type is HTML[73]) into the "resources" directory of the program. This HTML file was then iteratively fine-tuned, corrected and customised to be synched with the Controller class. Inline CSS elements were added into the file to create single HTML elements, such as tables and buttons.

The Thymeleaf engine works by parsing this template file ("cartelApp.html")[74], interpreting the different tags and expressions contained therein. As the program runs, it outputs an HTML page that is delivered to the client browser.[75]

[73] Thymeleaf can process XML, TEXT, JAVASCRIPT, CSS and RAW data types.
[74] Located in the Project at the following directory: src/main/resources/templates/cartelApp.html .
[75] The official Thymeleaf guide for different templates is available here: https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html#what-kind-of-templates-can-thymeleaf-process .

VI. Testing

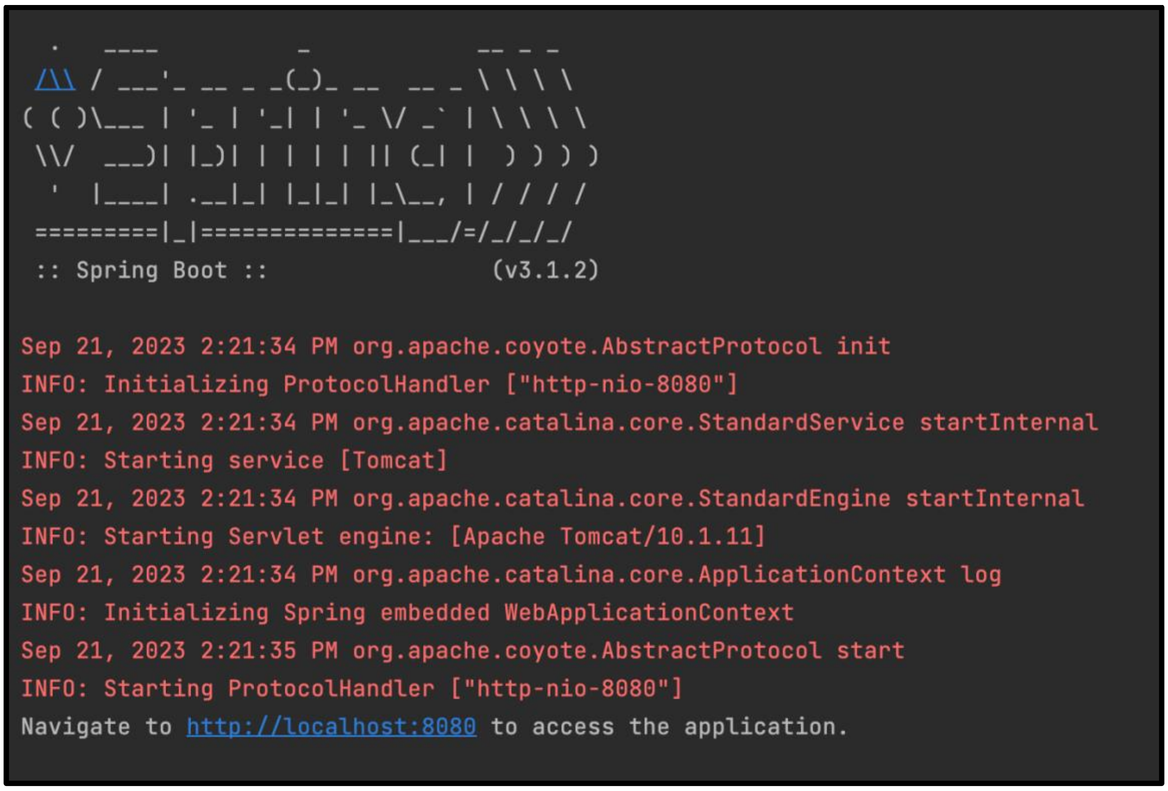
In this chapter, the program's functionality and the interaction with the Neo4j instance database layer is demonstrated, as well as an example of the unit tests that were performed in the code to ensure its correctness.

VI.1. *Running the application*

To run the program, it is necessary to have the Neo4j instance open and running, with the appropriate credentials so the database is successfully connected.[76]

Once the Java program runs, the user is asked to Navigate to the default port http://localhost:8080 to access the application.

Image 11
Console display when running the AFCI



```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.1.2)

Sep 21, 2023 2:21:34 PM org.apache.coyote.AbstractProtocol init
INFO: Initializing ProtocolHandler ["http-nio-8080"]
Sep 21, 2023 2:21:34 PM org.apache.catalina.core.StandardService startInternal
INFO: Starting service [Tomcat]
Sep 21, 2023 2:21:34 PM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet engine: [Apache Tomcat/10.1.11]
Sep 21, 2023 2:21:34 PM org.apache.catalina.core.ApplicationContext log
INFO: Initializing Spring embedded WebApplicationContext
Sep 21, 2023 2:21:35 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8080"]
Navigate to http://localhost:8080 to access the application.
```

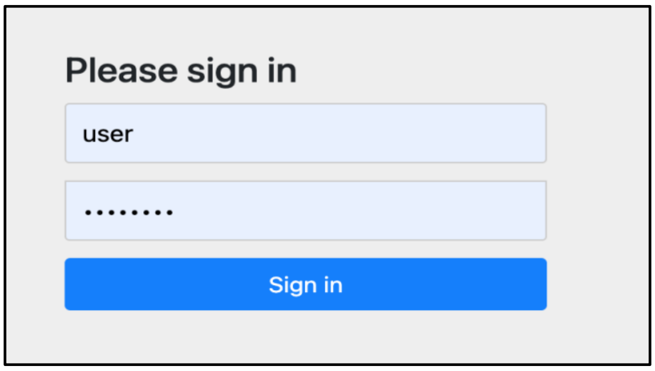
This shortcut is offered for convenience purposes, as it is possible to access just inputting the port address inside the web browser once the program runs to access the interface.

[76] This is explained further in Appendix 1, "User Manual".

VI.2. *Authentication*

The program then directs the user to a web login page, and it is asked to sign in with username and password:

Image 12
Web-based user authentication feature

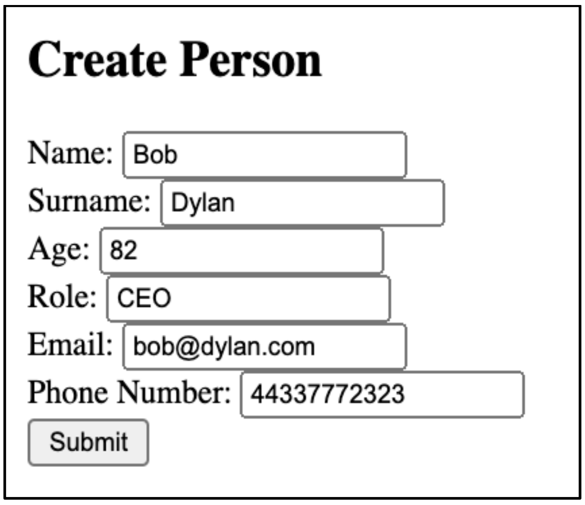


By default, the credentials are "**user**", and the password is "**bbk12345**". These are not the same as the user/password credentials defined for a given instance, and they can be modified in the application properties file of the program. After running the program, the user is directed to the program interface, where several queries and CRUD operations are allowed to show the capabilities of the program.

VI.3. *Creating / Deleting a Person node*

Under the label "Create Person", the user is presented with a form with all the fields for the properties and a "**Submit**" button.

Image 13
'Create Person' form with 'Submit' button

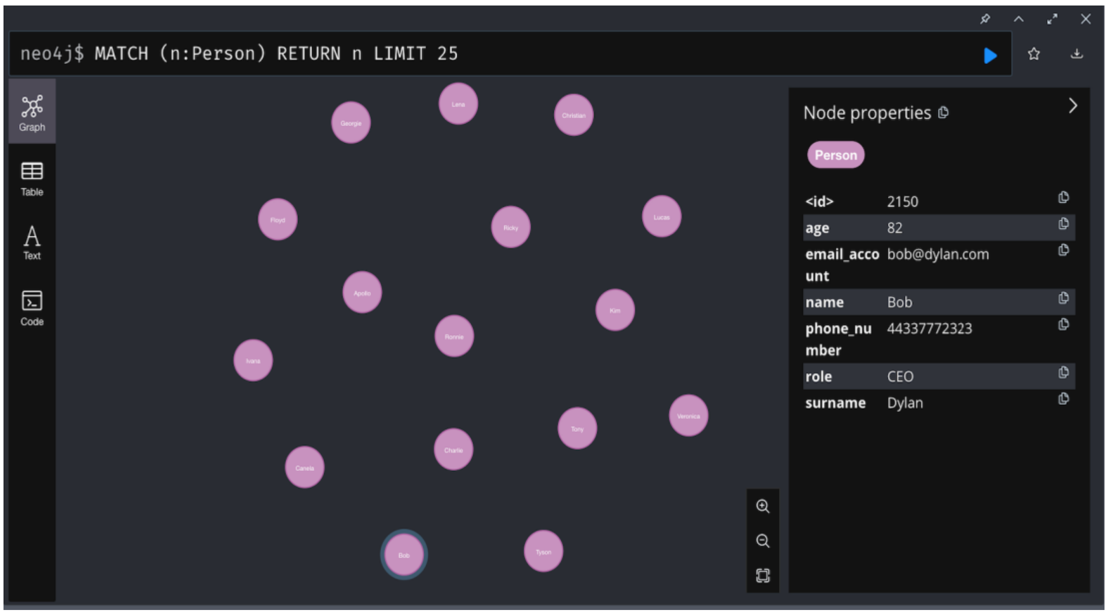


Create Person

Name:	Bob
Surname:	Dylan
Age:	82
Role:	CEO
Email:	bob@dylan.com
Phone Number:	44337772323

Once the fields are completed and a Person is created, this will be reflected in the Neo4j instance (see "Node Properties" on the right of the image).

Image 14
AFCI created Person node shown in Neo4j instance



```
neo4j$ MATCH (n:Person) RETURN n LIMIT 25
```

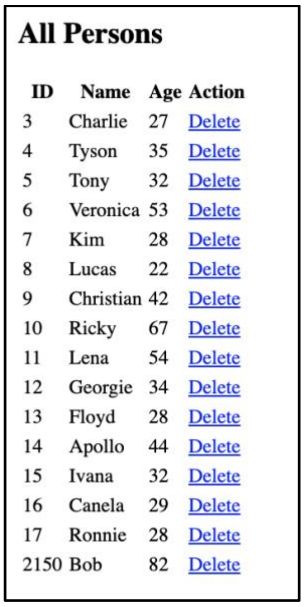
Node properties

Person

<id>	2150
age	82
email_account	bob@dylan.com
name	Bob
phone_number	44337772323
role	CEO
surname	Dylan

For this prototype, all Person nodes appear listed in the user interface. A new Person will thus appear in the interface after being created, allowing the user to delete it (or any other Person node) by pressing the “**Delete**” button next to their name, which will also reflect in the underlying database.

Image 15
List of "All Persons" and "Delete" button



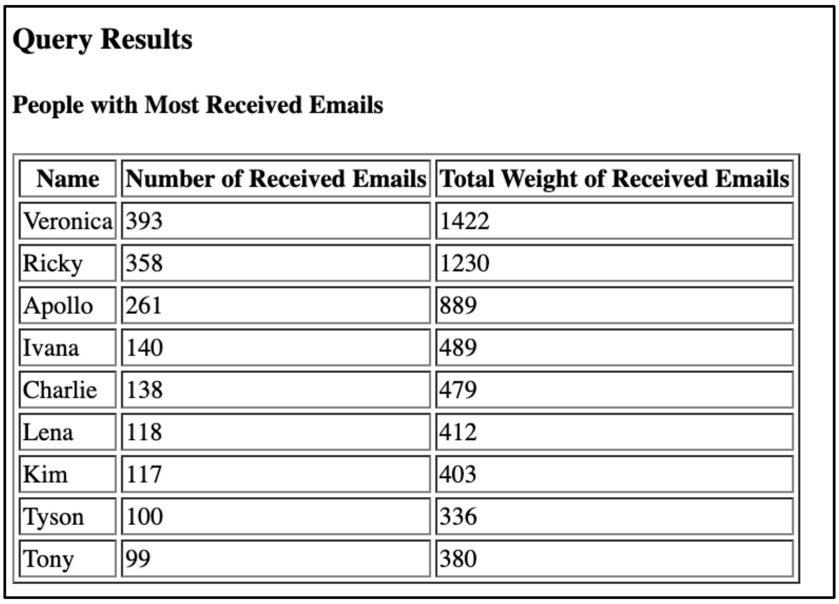
All Persons

ID	Name	Age	Action
3	Charlie	27	Delete
4	Tyson	35	Delete
5	Tony	32	Delete
6	Veronica	53	Delete
7	Kim	28	Delete
8	Lucas	22	Delete
9	Christian	42	Delete
10	Ricky	67	Delete
11	Lena	54	Delete
12	Georgie	34	Delete
13	Floyd	28	Delete
14	Apollo	44	Delete
15	Ivana	32	Delete
16	Canela	29	Delete
17	Ronnie	28	Delete
2150	Bob	82	Delete

VI.4. *Email/Message queries*

To test its functionality, queries were added into the EmailRepository and MessageRepository classes to show the result for simple operations, such as finding the top 10 persons that received the greatest number of Emails, including their total weight. Such queries are presented with a "**Find**" button in the interface. Here is how the results are displayed in the prototype:

Image 16
Query result for Person with most received Emails



Query Results

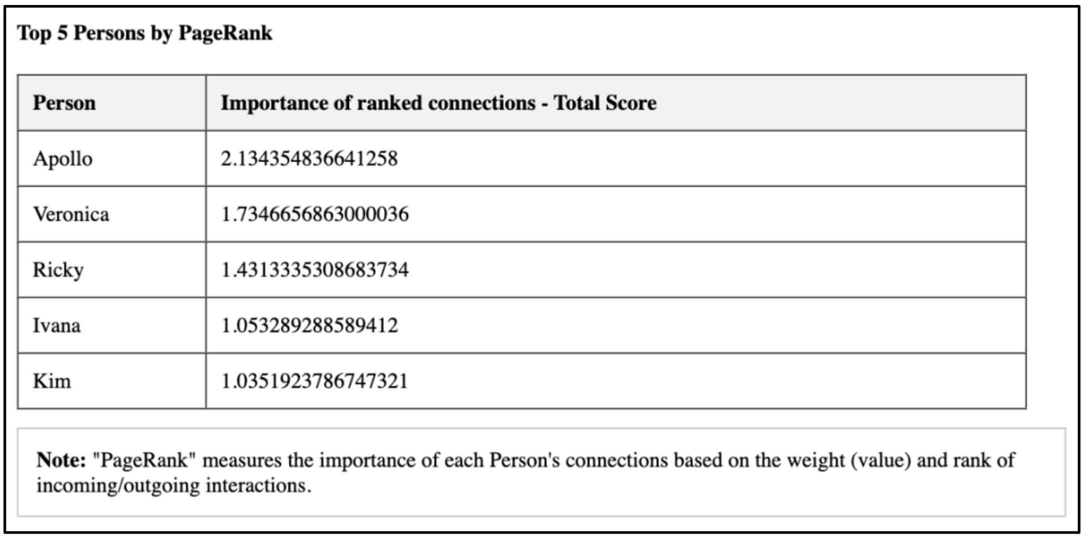
People with Most Received Emails

Name	Number of Received Emails	Total Weight of Received Emails
Veronica	393	1422
Ricky	358	1230
Apollo	261	889
Ivana	140	489
Charlie	138	479
Lena	118	412
Kim	117	403
Tyson	100	336
Tony	99	380

VI.5. *PageRank*

The program allows the user to find the top 5 pageRanked persons according to the PageRank attribute written into the Person node properties after running said algorithm. The results are displayed as follows:

Image 17
Query result for the 5 top PageRanked Persons



Top 5 Persons by PageRank

Person	Importance of ranked connections - Total Score
Apollo	2.134354836641258
Veronica	1.7346656863000036
Ricky	1.4313335308683734
Ivana	1.053289288589412
Kim	1.0351923786747321

Note: "PageRank" measures the importance of each Person's connections based on the weight (value) and rank of incoming/outgoing interactions.

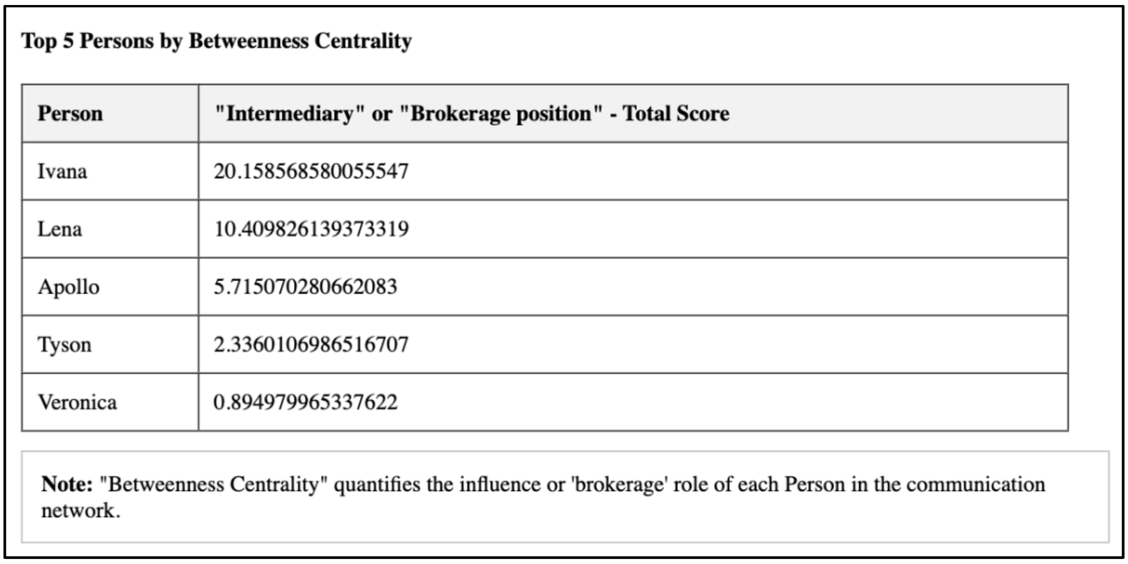
The table shows a hierarchy of interactions in the communication network based on "PageRank" scores. The score attributed to each Person node represents aggregated incoming weights and iteratively considers into the equation the fact that the top ranked people, like **Apollo**, **Veronica**, and **Ricky**, are communicating among themselves and/or with other Persons that, in turn, also have high incoming communication weights.

This bird's eye view of the data suggests that the evidence against the listed individuals might be stronger, from which it may follow that special investigative measures could be taken in their respect. For instance, these results could be explained and compounded with other evidentiary items to demonstrate the need of a dawn raid in one of the Firm's premises, or to conclude that one or more of the subjects could be targeted to be interrogated and/or asked to cooperate in the investigation in exchange for a more lenient treatment.

VI.6. *Betweenness Centrality*

Finally, the interface allows the users to find the Betweenness Centrality nodes in accordance with the implementation of that algorithm in the projected named graph defined as "**communication_graph**"[77]:

Image 18
Query result for the 5 top Betweenness Centrality ranked Persons



Top 5 Persons by Betweenness Centrality

Person	"Intermediary" or "Brokerage position" - Total Score
Ivana	20.158568580055547
Lena	10.409826139373319
Apollo	5.715070280662083
Tyson	2.3360106986516707
Veronica	0.894979965337622

Note: "Betweenness Centrality" quantifies the influence or 'brokerage' role of each Person in the communication network.

From the centrality scores shown in the table above, it may be inferred that the highest-ranked executives, **Ivana** and **Lena**, might be acting as the principal conduits or hubs for the passing of information within the presumptive cartel, and that the evidence relating to them might deserve further scrutiny and analysis.

Beyond, these results could be confronted with other facts to extract further insights in the form of a comparative analysis. For example, take the following facts:

(i) **Ivana** WORKS at '**K.O. Inc.**' and has the role of 'Zone Manager' .

(ii) **Lena** WORKS at '**TysonJab Ltd.**' and has the role of 'Zone Manager'.

(iii) **Apollo** WORKS at '**K.O. Inc.**' and has the role of 'VPSales' (Vice President of Sales).

[77] See section IV.4. (b) of the Report.

So, inside the hierarchy of the Firm 'K.O. Inc.', the relationship

Person -- [:MANAGES]--> Person

directed from **Apollo** to **Ivana** exists. We recall that **Apollo**, the top-scored Person in pageRank, is only third on the Betweenness Centrality ranking.

A possible hypothesis that could be drawn from this information is that **Apollo** may be acting as a *decision-maker* of some sort. While very influential in running the conspiracy, he might be "shielded" from being exposed by having personnel below him in the corporate hierarchy (in this case **Ivana**) to arrange direct communications with people in similar roles in the competition (**Lena**).

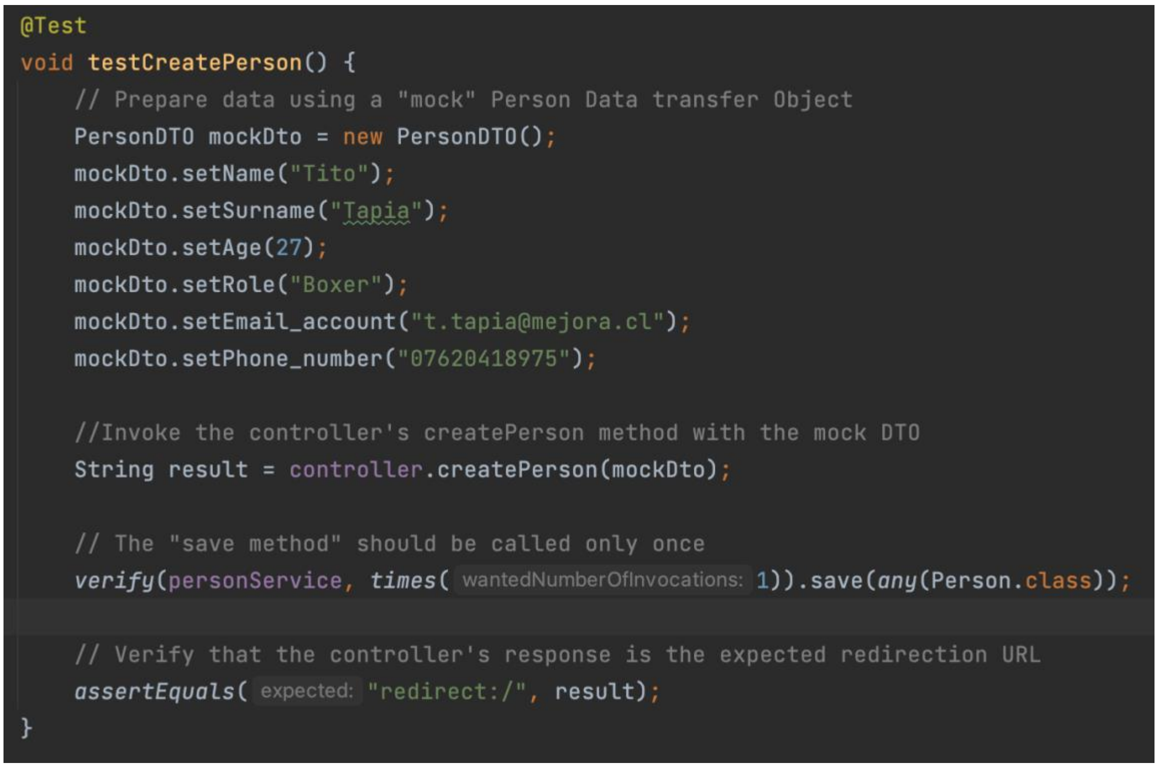
The information provided in the previous sections can also help narrow down the targets of the investigation. People that do not appear in neither these rankings may perhaps not know of the illegal scheme at all, or otherwise might not be involved to the same extent and might be more inclined to cooperate.

VI.7. *Unit Tests*

For unit testing, I have used JUNIT and Mockito[78] to inject simulated behaviours from different components (DTOs, Repositories, etc.). As an example, below I show the test for the createPerson() method from the Controller class:

[78] https://site.mockito.org/ .

Image 19
Unit test for createPerson() method from the Controller class[79]



```
@Test
void testCreatePerson() {
    // Prepare data using a "mock" Person Data transfer Object
    PersonDTO mockDto = new PersonDTO();
    mockDto.setName("Tito");
    mockDto.setSurname("Tapia");
    mockDto.setAge(27);
    mockDto.setRole("Boxer");
    mockDto.setEmail_account("t.tapia@mejora.cl");
    mockDto.setPhone_number("07620418975");

    //Invoke the controller's createPerson method with the mock DTO
    String result = controller.createPerson(mockDto);

    // The "save method" should be called only once
    verify(personService, times( wantedNumberOfInvocations: 1)).save(any(Person.class));

    // Verify that the controller's response is the expected redirection URL
    assertEquals( expected: "redirect:/", result);
}
```

A "mock" Person DTO was populated with fictional Person data - fictional, in the sense that such Person was not retrieved from the database. After triggering the createPerson() method with the newly created mock PersonDTO, the test checks that the save method from the PersonService class is properly invoked, that it is invoked only once and with a Person object[80]. Lastly, the test checks that the user is correctly redirected to the View.

Other unit tests for other methods in the Controller class and Service Classes are available in the corresponding directory.[81]

[79] Test in the ControllerTest.java class. This class is located in the following path in the Repository: src/test/java/mscproject/cartelapp/ControllerTest.java.
[80] Using Mockito's verify() method.
[81] Tests available at this path pf the Project: src/test/java/mscproject/cartelapp .

VII. Limitations and future developments

VII. 1. *Limitations*

VII.1.(a) Domain-specific nature

The AFCI is designed and tailored to a specific use-case, which ensures its precision and optimisation for its designated domain. However, this speciality also constrains the program to the domain of the elements and entities defined in the database, in the form that it is currently structured. This means the AFCI will not work with other datasets without significant modifications to the codebase.

VII.1.(b) Limited queries

As it is currently structured, users are confined to pre-curated, statically defined queries. This configuration narrows the functionality of the AFCI to a subset of its possibilities, so as users interact with the tool, more queries need to be successively integrated taking advantage of the extensibility qualities of the current program architecture.

VII.1.(c) Neo4j dependence

AFCI is designed to work as a companion to Neo4j, which inevitably falls into a “lock-in”. Many features, even the entire GDS library, could eventually be deprecated. As mentioned above, there are some other downsides to working with Neo4j: It will be necessary to have at least one person in the organisation that is knowledgeable in interacting with the Neo4j instance, and there are licensing costs to consider.

VII.1.(d) Metrics from "Weight" properties

In its current version, the most significant insights extracted by the graph analytics powers of Neo4j for the AFCI, stem from the COMMUNICATION relationships and the “Weight” property assigned to them. There are two issues to consider: (i) In this case, we have identified significance of a piece of evidence by assigning a randomised value between 1 and 5, being 5 the highest value. This is an obvious simplification of reality, as usually there are many factors and nuances to consider when analysing evidence that cannot be translated into a single digit. (ii) Relying on human evaluation can be burdensome and error-prone, though, as explained above, the pageRank algorithm takes some of this “noise” into account by adding a damping factor into the computation.

VII.2. *Future developments*

VII.2.(a) Improved user interface

In **AFCI**, the user interface was conceived as a vehicle to show the capabilities of integrating a graph database to a backend framework. The UI can certainly be upgraded to a more robust client-side library (like React, Angular or other) to allow for enhanced query manipulation or user-customised representations of the data.

VII.2.(b) Nodes/Properties

Beyond the basic nodes created for the dataset as described above, many other entities could be created according to the needs of the investigation: Meetings[82], Financial Transactions, Documents or even Metadata of documents or communications gathered as evidence.[83]Other very interesting area of development is to use Product nodes to represent sales, which could be used to analyse price trends or to implement screening techniques to identify patterns that could flag anticompetitive behaviour.[84]

VII.2.(c) Other GDS algorithms

The Project has demonstrated that applying data analytics capabilities from the GDS suite to an evidence dataset can lead to powerful conclusions. There are plenty of other similarity, pathfinding, or centrality algorithms that could be implemented by selecting the corresponding properties that should be used and projecting named graphs in the database accordingly.

[82] "Meeting" nodes were initially considered but not implemented in the database due to time constrained. See README file (Appendix 2).

[83] The amount of digital evidence in a normal case where the amount of information available amounts to several Terabytes. This can amount to real 'data lakes' where metadata can be analysed and used to make connections between different types of entities.

[84] The Java codebase has a Product entity class and a ProductService class with the method *createProduct(MultipartFile file)* to upload products from an Excel document to the Neo4j database. Due to time constraints, I was not able to embed this functionality of this method into the HTML view.

VII.2.(d) Large Language Model capabilities

Although outside the scope of this endeavour, it seems clear that in the next few years all evidence analysis and categorisation in addressing corporate criminality will incorporate Large Language Model capabilities by embedding evidence data to a generative pre-trained transformer API.[85] This could enable, for example, that queries written in natural language can be translated to Cypher (or other database management language) under the hood, or that anomalies in certain data patterns could be detected and then explained to the user.

[85] For example, the Open AI API is already available for developers. Documentation available at: https://platform.openai.com/docs/introduction . See a hands-on example of an academic knowledge graph using Neo4j and OpenAI API available at: https://medium.com/@yu-joshua/building-an-academic-knowledge-graph-with-openai-graph-database-12b320f08ef0 .

VIII. Conclusions

During the development of this project, two main challenges were encountered:

First, the complexity of populating the database, a very time-intensive process which was grossly underestimated when embarking in the Project. Second, defining and completing the UI, not only due to my lack of experience working with HTML, but because Neo4j inherently has a very rich interface. Hence, it was a hard task discerning between different possible functionalities to add to the interface.

The result has attempted to provide value to the end users by allowing a non-specialised audience to interact with a mainstream Graph DBMS and to take advantage of its analytical prowess in a straightforward manner. The outlook of categorising communications with some predefined values or parameters, to then find critical nodes under different definitions of importance using graph algorithms, seems a particularly promising area of future research.

Rudimentary forms of visual representations have long been used by investigators fighting organised crime to make connections between apparently isolated data entities. All setbacks and limitations considered, this project has aimed to establish a groundwork for further academic exploration working with Graph DBMS for corporate crime-related evidence analysis, showcasing how a graph database can be designed, and what kind of interactions and data analytics are possible. As developers, it is our role to craft the next generation of “Crazy Walls”, so they are increasingly intuitive, insightful, and user-friendly.

REFERENCES

1. Budur, E., Lee, S. & Kong, V., 2015. 'Structural Analysis of Criminal Network and Predicting Hidden Links using Machine Learning'.

2. Brandes, U., 2001. 'A Faster Algorithm for Betweenness Centrality'. *Journal of Mathematical Sociology*, 25(2), pp.163-177.

3. Brandes, U. & Pich, C., 2007. 'Centrality Estimation in Large Networks'. *International Journal of Bifurcation and Chaos*, 17(07), pp.2303-2318.

4. Brin, S. & Page, L., 1998. 'The Anatomy of a Large-Scale Hypertextual Web Search Engine'. *Computer Networks and ISDN Systems* 30, pp.107-117.

5. Calderoni, F., 2014. 'Social Network Analysis of Organized Criminal Groups'.

6. Carnaz, G., Nogueira, V. & Antunes, M., 2021. 'A Graph Database Representation of Portuguese Criminal-Related Documents'. *Informatics*, 8, p.37.

7. Carrington, P., 2011. 'Crime and Social Network Analysis'. In: J. Scott & P. Carrington, eds. *Sage Handbook of Social Network Analysis*, Ch.17.

8. Competition and Markets Authority (UK), 2014. 'Transparency and disclosure: Statement of the CMA's policy and approach'.

9. Faisal, et al., 2015. 'Role of graph databases in social networks'.

10. Fowler, M., 2004. 'Inversion of Control Containers and the Dependency Injection pattern'.

11. Gleich, F., 2014. 'Pagerank beyond the web'. *Siam Review*, 57(3).

12. Henney, K. (ed.), 2010. *97 Things Every Programmer Should Know*. O'Reilly.

13. Hunt, S., 2022. 'The technology-led transformation of competition and consumer agencies: The Competition and Markets Authority's experience'. Discussion paper.

14. International Competition Network, 2021. 'Anti-Cartel Enforcement Manual (2014, updated 2021), Chapter 3: Management of Electronically Stored Information (ESI) in searches, raids and inspections'.

15. Kang, Y. et al., 2011. 'How can Visual Analytics Assist Investigative Analysis? Design Implications from an Evaluation'. *IEEE Transactions on Visualisation and Computer Graphics*, 17(5).

16. Lal, M., 2015. *Neo4j Graph Data Modelling*. Packt Publishing.

17. Manikanta Vamsi, K. et al., 2021. 'Visualization of Real World Enterprise Data using Python Django Framework'. *IOP Conf. Ser.: Mater. Sci. Eng.*, 1042, 012019.

18. Matsushima, H., 2019. 'Blockchain Disables Real-World Governance'. Kyoto University, *Institute of Economic Research Working Papers*, 1017.

19. Moreselli, C. & Roy, J., 2008. 'Brokerage Qualifications in Ringing Operations'. *Criminology*, 46(1).

20. Nadime, F. et al., 2018. 'Cypher: An Evolving Query Language for Property Graphs'. *SIGMOD'18 Proceedings of the 2018 International Conference on Management of Data*.

21. Organisation for Economic Cooperation and Development (OECD), 2019. 'Working Party No. 3 on Co-operation and Enforcement. Access to the case file and protection of confidential information', December.

22. Rebika, R. & Chettri, P., 2018. 'NoSQL Hands On'. In: P. Raj & G. Chandra Deka, eds. *Advances in Computers, Volume 109. A Deep Dive into NoSQL Databases: The Use Cases and Applications*. Elsevier.

23. Robinson, I., Webber, J. & Eifrem, E., 2015. 'Graph Databases'. 2nd ed. O'Reilly.

24. Sadowski, G. & Rathle, P., 2015. 'Fraud Detection: Discovering connections using Graph Databases'. Neo4j White Paper.

25. Salo, O. & Abrahamsson, P., 2007. 'An Iterative Improvement Process for Agile Software Development'. *Softw. Process Improve. Pract.*, Wiley.

26. Scifo, E., 2020. *Hands-On Graph Analytics with Neo4j*. Packt Publishing.

27. Sarvari, H. et al., 2014. 'Constructing and Analyzing Criminal Networks'. *IEEE Security and Privacy Workshops*.

28. United States Department of Justice, Antitrust Division, 2022. 'An Antitrust Primer for Federal Law Enforcement Personnel'.

29. Wish, R., 2012. Competition Law. 7th ed. OUP.

APPENDIXES

APPENDIX 1: USER MANUAL

APPENDIX 2: README.md . A markdown file with the original database design (Available at the Github repository).

APPENDIX 3: Example Database Definition Queries.txt. A text file that shows part of the iterative process of graph-database definition embarked for the project.

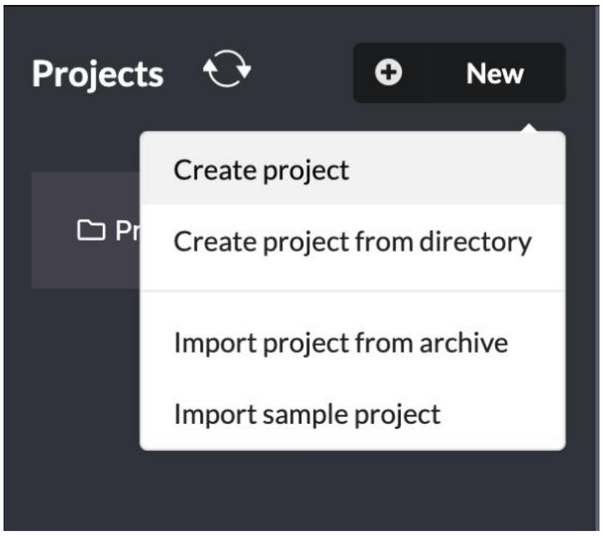
APPENDIX 4. JAVA CODE for the AFCI (Available at the Github repository).

APPENDIX 1: USER MANUAL

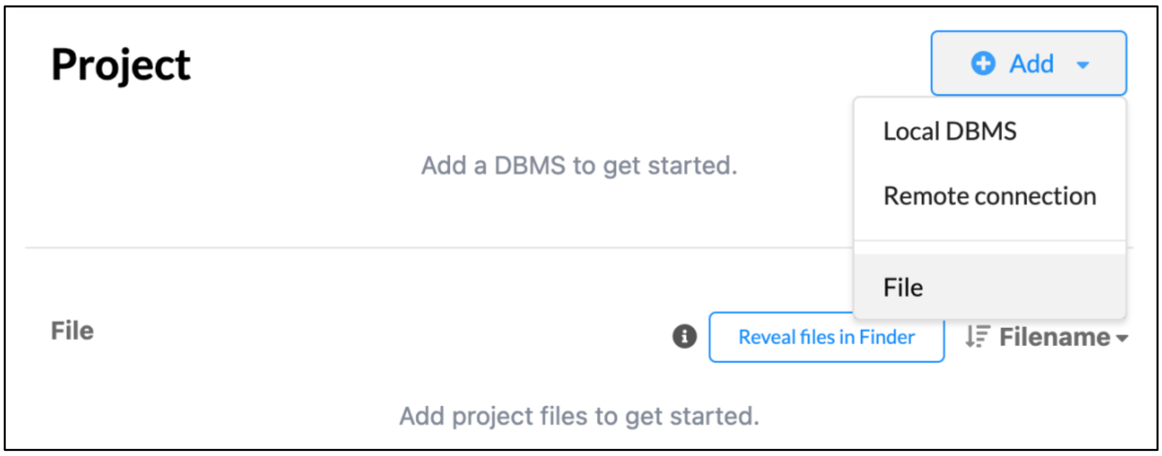
I. Installing the Neo4j desktop application and importing the database.

a) Download and install the Neo4j desktop application. The application includes a "Developer's Licence" for free.[86]

b) Once installed, open the application and choose "Create project":



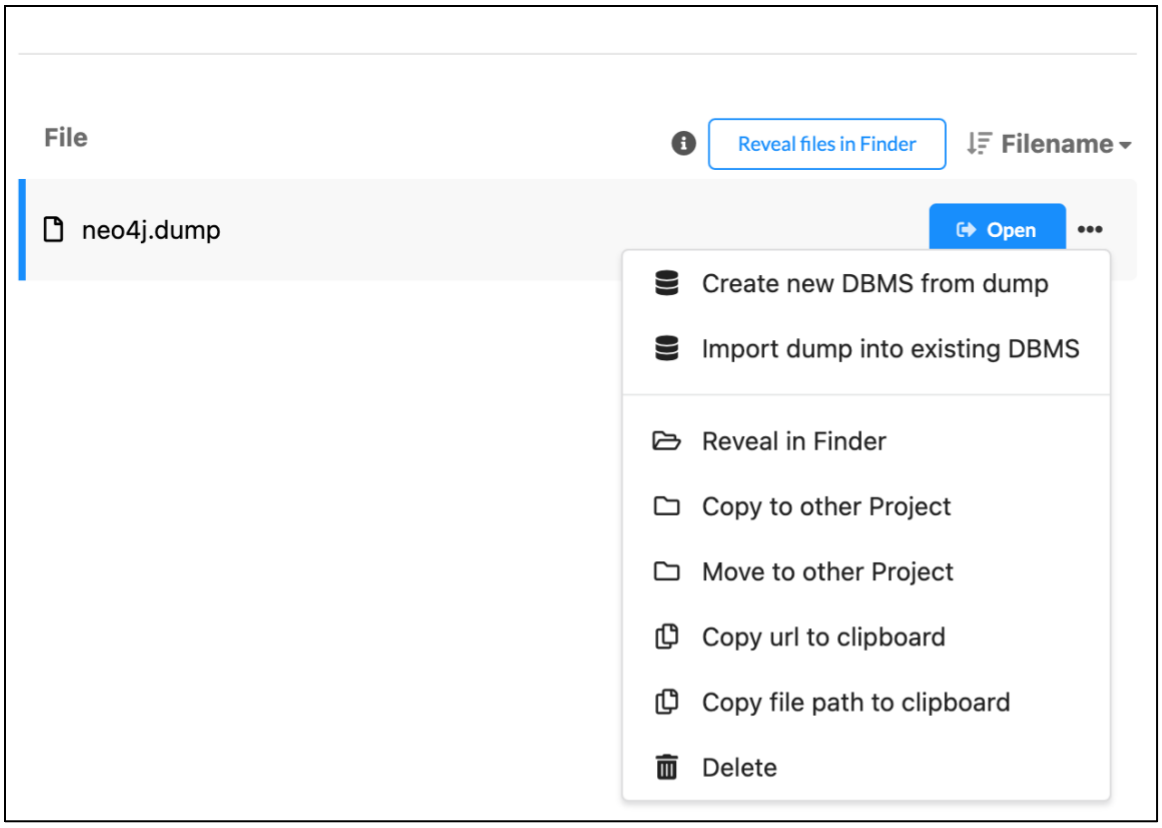
c) Inside the project, click "⊕ Add" and choose "File".



[86] Installation manual available at: https://neo4j.com/docs/desktop-manual/current/installation/ . See further information available at: https://neo4j.com/docs/getting-started/ .

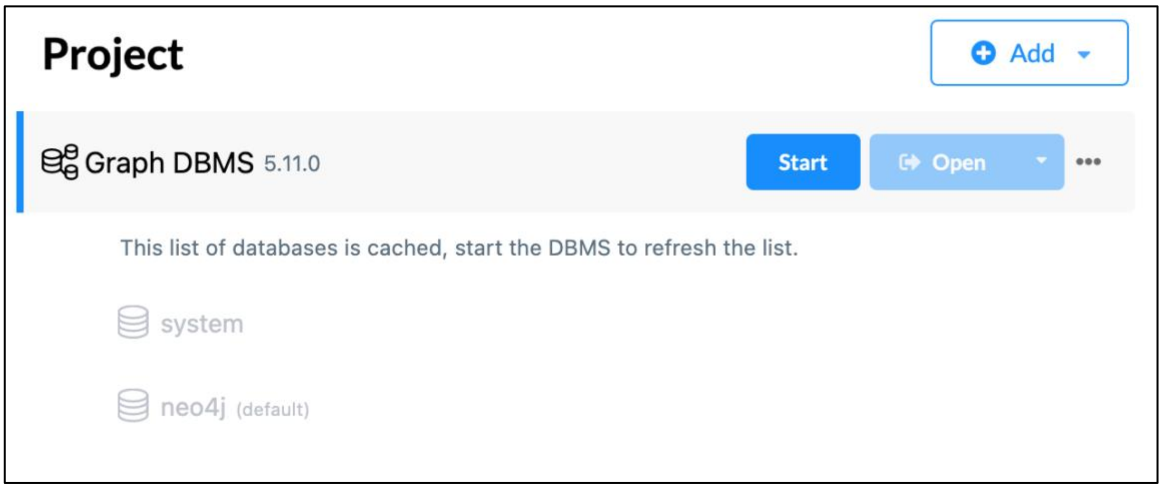
d) Once prompted, choose the "dump" file "**Neo4j.dump**" from dump_file directory of the project's GitHub repository.

e) Click the ••• symbol next to the "Open" button (see below) and choose "**Create new DBMS from dump**".



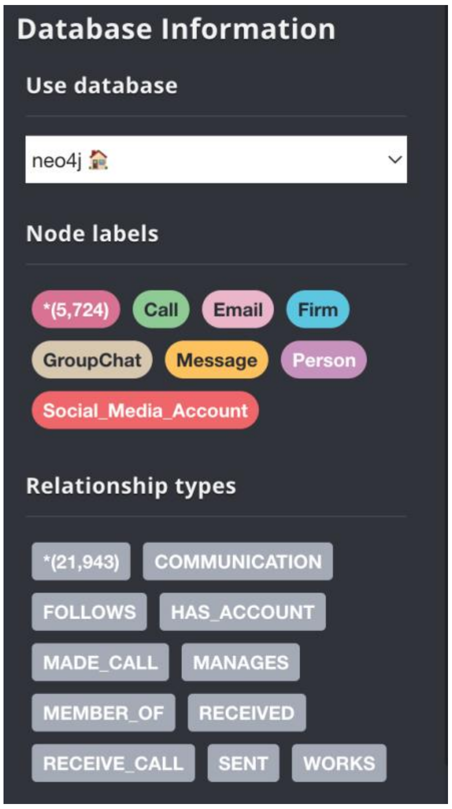
f) Choose a password for the Graph DBMS that is created. For simplicity, use "bbk12345", (otherwise it will be necessary to change the application properties).

g) Click "Start" and then "Open" the database:



This list of databases is cached, start the DBMS to refresh the list.

Once open, the database information should be available and the database ready for interaction:


Database Information

Use database

neo4j 🏠

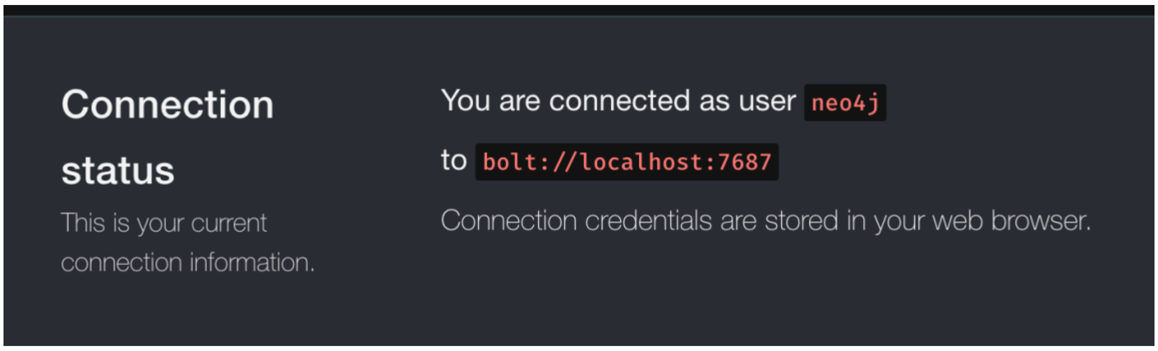
Node labels

*(5,724) Call Email Firm GroupChat Message Person Social_Media_Account

Relationship types

*(21,943) COMMUNICATION FOLLOWS HAS_ACCOUNT MADE_CALL MANAGES MEMBER_OF RECEIVED RECEIVE_CALL SENT WORKS

If necessary, take a look at the information related to the connection status that are displayed when opening the database. This is important because the user and port should match the "application properties" in the Java program.


Connection status

This is your current connection information.

You are connected as user `neo4j`

to `bolt://localhost:7687`

Connection credentials are stored in your web browser.

II. Running the AFCI

a) Import the project files from Version Control (GitHub) from IntelliJ or other IDE.

b) Check the "application properties" in the following path:

src/main/resources/application.properties

c) To connect the database, the following information should be correctly set up:

```
spring.neo4j.uri = neo4j://localhost:7687
spring.neo4j.authentication.username= neo4j
spring.neo4j.authentication.password= bbk12345
server.port=8080
spring.security.user.name=user
spring.security.user.password=bbk12345
```

i) spring.neo4j.uri : The URI for the Neo4j Database should match the Neo4j Bolt protocol. By default, this is set to **7687**.

ii) spring.neo4j.authentication.username: Username identified in the Neo4j database.

iii) spring.neo4j.authentication.password: Password for the Neo4j database.

iv) server.port: Sets the port where the Spring Boot app will run. '8080' is normally the default port for web applications.

iv) spring.security (user.name and user.password): These are the username and password credentials that should be entered when the web application is run.

d) Finally, run the Java program and connect to the local port http://localhost:8080 by clicking the link in the console or manually entering the address into a web browser.