UNIVERSITY OF TARTU

Institute of Computer Science

Software Engineering Curriculum

Mikheil Kenchoshvili

Tool for Identifying Working Style Based on Event Logs

Master's Thesis (30 ECTS)

Supervisor(s): Marcelo Sarini

Marlon Dumas

Tartu 2018

Tool for Identifying Working Style Based on Event Logs

Abstract:

Working style is a relatively new topic in Business Informatics. It is strictly related to concepts existing in other areas of the research such as Organizational Mining, Visual analytics, Pattern recognition. While there are several tools to tackle the task of organizational mining, they are general purpose stand-alone desktop applications, and none of them concentrate solely and deeply on working style identification. To this end, this thesis introduces a domain-specific web-based tool that aids the user with working style identification. The tool allows end users to import event logs. They can explore and query patterns to answer the questions related to the working style of the human actors involved in that event logs. The thesis presents, functional and technical requirements as well as the implementation details of the tool. Also, discusses the difficulties faced during the development and the gaps that have been identified in the current state of practice of the technologies that were used for implementation. Moreover, several possible extensions of the tool are suggested that could be addressed in the future works. Finally, the thesis demonstrates the usage of the tool by importing the sample data and exploring different capabilities and aspects of the application.

Keywords:

Working Style, Working Style Identification, Organizational Mining, Event Log

CERCS: P170 - Computer science, numerical analysis, systems, control

Sündmuste logi alusel tööstiili tuvastamise tööriist

Lühikokkuvõte:

Äriinfotehnoloogia on tööstiil küllaltki uus uurimusteema. See on tihedalt seotud teistest uurimisvaldkondades tuntud kontseptsioonidega nagu näiteks organisatsiooniline kaevandamine, visuaalne analüüs ning mustrituvastus. Organisatsioonikaevet abistavaid tööriistu on küll mitmeid, kuid nad on kõik üldiseks otstarbeks eraldiseisvad programmid ning ükski ei spetsialiseeru süviti just tööstiili tuvastamisele. Antud uurimustöö tulemusena valmis domeenispetsiifiline veebipõhine tööriist mis aitab kasutajal tuvastada tööstiili identifikaatoreid. Kasutajal on antud keskkonda võimalik importida vaadeldavate isikute tegevuste ajalugu, vaadelda neid andmeid ning koostada päringuid tuvastamaks tööstiili mustreid. Lisaks funktsionaalsete ning tehnilistele nõuetele on kaetud ka tööriista teostuse üksikasjad. Lisaks arutleb autor arenduse käigus tekkinud raskuste üle ning kirjeldab tööriista valmistamisel kasutatud tehnoloogiates leitud puudujääkide hetkeseisu. Lõputöös tehakse ka mitmeid ettepanekuid tööriista täiendamiseks edasistes töödes ning demonstreeritakse näiteandmestiku abil tööriista kasutamist, uurides programmi erinevaid võimalusi ning aspekte.

Võtmesõnad:

Tööstiil, tööstiili identifikaator, organisatsiooniline kaevandamine, sündmuste logi

CERCS: P170 - Arvutiteadus, arvanalüüs, süsteemid, kontroll

Table of Content

Table of Figures

1 Introduction

Working style is a relatively new topic in Business Informatics. It is strictly related to concepts existing in other areas of the research such as Organizational Mining, Visual analytics, Pattern recognition. Working Style characterizes the nature of work, with the main focus on the relationships between human actors who perform the activities during the unfolding of a business process. For this extent, working style sheds light on the choices performers make within the constraints posed by the workplace [1].

Nowadays companies rely on process-aware information systems such as CRM, ERP, so forth to organize and track their business processes. These systems pose a certain limitation on the possible choices human actors can make while at the same time they support the unfolding of a business process. So, in the various situation, it would be useful to identify the working style of those actors to makes visible how people arrange their duties in the presence of such technologies.

1.1 The goal of the thesis

Several existing tools can analyze and visualize event logs, but most of them are general purpose and usually concentrate on control flow aspect of the event logs rather than organizational ones [2]. Moreover, a narrow field such as working style identification gets even less share of the attention.

The goal of the thesis is to create the tool based on event logs, that will help users to identify the working style. We do this by converting event logs into more comprehensible and intuitive graph-based structure that we can easily analyze and visualize. We generate and visualize WSA and WSEA to help the user with the task of working style identification.

We aim to create a domain specific web application that potentially can have multiple users that collaborate with each other. Narrowing down the domain and introducing the semantic of the logs allows us to create an intuitive web-based tool that can aid user to identify the working style of the human actors based on the event logs.

1.2 Research Questions

Given the lack of domain-specific, web-based tools that can aid user to identify working style we propose the following main research questions:

1. How to represent event logs to support working style identification?

The answer to this question will help to clarify the intermediate structure that we are going to use to store event logs efficiently to aid with querying and visualization tasks. Since we are going to base our tool on event logs, it is vital to have a deterministic way of transforming and storing them.

2. How to extract WSA from the intermediate structure and identify patterns?

To aid the user with working style identification, we should find the way to provide them with WSA (working style artifact) that are simple enough to conclude and also a way to query and identify patterns in it

3. How to visualize and compare WSAs?

This question is directly related to the ultimate goal of working style identification. Without proper visualization, the user will fail in their task.

1.3 Research Methodology

This section discusses the research approach of the thesis. This work follows the design science research methodology. In order to resolve the identified research question, we have created and evaluated IT artifact. Based on the design science research methodology the thesis consists of the following activities:

a. Design as an Artifact. The thesis presents a web-based tool for the identification of working style.
b. Problem Relevance. The paper [2] identifies the gap in the research of mining organizational aspects. While they introduce their research to try and close the gap in theoretical work, one cannot ignore the lack of practical, web-based, tools.
c. Demonstration. The thesis demonstrates the proposed solution as a web application. Section 1.5 discusses the features of the artifact.
d. Evaluation. IT artifact is evaluated using a sample event log from Appendix II. The author evaluates the application by populating the data via event log uploader and querying patterns from derived WSA. The requirements of the artifact are evaluated using three test scenarios: testing event log uploader, testing data explorer, and testing pattern creator.

e. Research Contribution. The proposed IT artifact is a tool that enables the user to upload and analyze the event logs to identify the working style of human actors involved in the execution of work. The system also supports exploration of relationships between these actors and other entities from the event log such as the activities and cases.
f. Communication of Research. The solution proposed in this thesis is available on the web. Any interested parties: end-users, software developers, and other researchers can freely utilize it.

1.4 Functional Requirements

This section contains functional requirements. We also have extracted user stories from the project related discussions to drive front-end development, track project progress and evaluate the results. Appendix II includes a complete list of user stories, while each subsequent paragraph of this section contains lists of the IDs of related user stories if there are any.

Upload event log

The web application should have the page where one can select the event log file and upload it to the server. The application should ensure the validity of the file and notify the user if anything goes wrong. The application should support CSV file similar to the one from Appendix II

Related user stories: 1, 2

Convert and store data

After the user submits the desired CSV file, the application should crunch the event logs and populate the database with relevant structures.

Visualize Data

The web application should provide a page where the user can explore the state of the database. There should be appropriate filters to hide irrelevant data and closely explore specific parts of the data.

Related user stories: 3, 4, 5

Query Patterns

The web application should provide a page where the user can indicate the query parameters (length, performers involved and so on) to identify patterns between involved performers.

Related user stories: 6, 7, 8

1.5 Technical Requirements

Database

The application should use the Neo4j [3] Graph database. We choose Neo4j over others similar technologies such as OrintDB [4] because of well documented and powerful Cypher query language. Moreover, it has built-in web-based administration tool, Neo4j Browser, that allows to connect to remote server using only a web browser and manage, update, and query the data. Also, it has database drivers for most of the common programming languages (Python, Java, .Net, JavaScript).

Nodes are the main data entities in the Neo4j database. Relationships connect two Nodes (they are directional). Both Nodes and Relationships can have properties (attributes of key: value pairs). Nodes have one or more labels that describe their role in the graph (we can refer to the group of nodes using the label).

Based on our preliminary research Neo4j is uniquely qualified for our use case and it has all necessary database entities to support the Graph structure that we propose in section 2.

Web development framework

Out of the four programming languages that support Neo4j both Python and JavaScript are equally well fit to develop web applications. We chose Python as the parties involved in the research (author and supervisor) had the personal preference towards this language.

As for web development framework we chose Flask [5]. We had Django [6] as the alternative, but because Flask is so lightweight and it is better suited for the research project such as ours.

Front-end

For the front-end component library application should use Bootstrap [7] As it is the most popular among web developers. Among other pros Bootstrap has prebuilt components, the responsive grid system, and powerful jQuery [8] plugins make it so much more useful.

As for web browsers, the application should run on the Chrome [9] without interruptions. Support of other browsers is encouraged but not required.

1.6 Limitations

This section aims to make clear the limitations that we have faced before and during the implementation of the tool. The section of technical requirements states that application should use Neo4j. However, this has its costs. We did preliminary research on the technology related to Neo4j and detected a few limitations.

Lack of PaaS providers

Neo4j is a relatively new technology, and it experiences a lack of PaaS providers. The best fit that our preliminary research found was GrapheneDB [10]. Unfortunately, they only provide the support for bare Neo4j, but they do not let the user alter the configuration of the system which is crucial if one wants to extend neo4j with plugins.

One of such plugins that we use is Graph Algorithms [11]. To operate it needs neo4j configuration to have certain parameters defined that are not there by default. So GrapheneDB cannot support it by design.

We have tried to host neo4j on Amazon EC2 [12] machine ourselves. Unfortunately, we went a little out of the scope of the project when one of the libraries failed to establish a connection because of the SSL certificate [13] issues. Moreover, Amazon is a commercial platform and not suited for scientific or open source and non-commercial projects.

So to benefit from full capabilities of Neo4j one needs to host it on a private server to be able to access and tweak all configurations manually. For these limitations, we were not able to automate the deployment process of the application. We can only manually deploy the version of the application with limited capabilities.

Visualization library

The preliminary research has identified Neovis.js as the best candidate for the visualization of Graph database content. Moreover, the official website of Neo4j reference this library multiple times. Also, the applications that rely on Neo4j frequently use this library. The primary reason for the popularity of Neovis.js is that it has a JavaScript driver for neo4j out of the box.

Unfortunately, Neovis.js is very rigid when it comes to customization of behavior and tends to make visualization decisions automatically. For this reason, there are some inconsistencies in the tool that show up from time to time. We discuss these inconsistencies in the implementation section and provide an explanation for each of them as well as the ways we can fix them in the future.

2 Related Work

This section aims to briefly overview the related theoretical work and the current state of practice.

2.1 Theoretical background

Set of choices that performers make under the constraints of the workplace characterizes the working style, this becomes more evident when process aware systems drive the unfolding of the business process as those tools pose certain constraints on how the workflows during the unfolding of the business process. [5]

Just like a painting is the visual counterpart of the painting style of the artist, working style artifact (WSA) is the visual counterpart of the working style. Moreover, just like art critics can identify a certain style from the painting, we can identify the working style base on working style artifact. WSA can serve as a tool for comparisons to identify similarities and differences about the nature of work.

We need to practically represent WSA in the way to easily reason regarding the nature of work. This thesis gets inspiration from the paper [1] but refines the structure proposed in the original work and provide a practical web-based tool that utilizes new structure to aid users with the identification of the working style.

2.2 State of Practice

This section discusses the practical tools that concentrate on process mining, and organizational mining.

ProM, Social Network Miner

Popular process mining tool ProM includes a social network miner plugin. It can import and convert data from event logs into sociograms. Converted data can serve as the starting point for execution of SNA. Using this plugin one can apply different social network mining algorithms to the data. The user can control visualization aspects, observe the data, and, filter the irrelevant information by setting thresholds. Papers [14, 15] discuss the theoretical work, as well as the implementation details.

Plugins serve as a comprehensive tool for organizational mining. While the certain results that we want to achieve can be detected using social network miner of ProM, it is still a standalone desktop application. The user has to install the whole ProM Framework and only after that they can download and benefit from appropriate plugins. In contrast, our solution is a web-based application that user can access using a web browser without downloading or installing anything extra.

Disco

Disco is a commercial process mining software that can construct process models from complex and large event logs. It is a general-purpose tool that tries to cover many aspects of process mining mainly focusing on the control flow of the event logs. Organizational mining is not Disco's primary focus, but users with a deep understanding of the tool can find a workaround to creatively swap activities and resources to build process model that can help to draw certain conclusions related to organizational structure [16].

The downside of Disco is that it is commercial, so users need to pay if they want to benefit from full capabilities. Also, it is a standalone desktop application that users need to install on their machine. While Disco can detect certain results that our application aims it requires deep knowledge of the tool to do so which can be challenging for certain users.

bupaR

BupaR [17] (business process analysis in R) is an open source project that consists of eight packages for R programming language each one of them has a different focus such as creation, exploration, and visualization of event logs. BupaR is well suited for a data scientist. Average users without the appropriate background will have troubles using the tool, but It is worth mentioning that if one has sufficient understanding of the data science and wants to work closely with the raw event logs first hand, these packages are wonderful utilities to do so [6].

BupaR has a little bit different niche than the tools discussed above or the one that we introduce in this project.

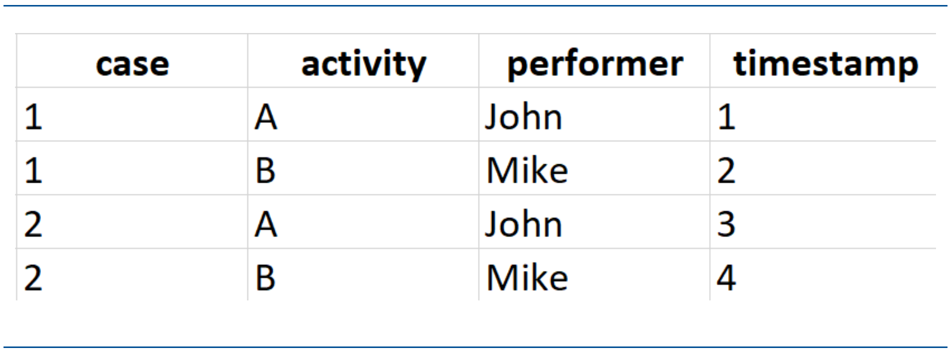
3 Proposed Solution

This section discusses the proposed solution, while the subsequent section, will discuss the implementation based on this proposal.

3.1 Event Log to Graph

We assume that the input is an event log similar to the Figure 1. To put it into words we interpret event log as follow: A case includes some activities each performed by a performer at a given timestamp.

Figure 1. Example CSV data



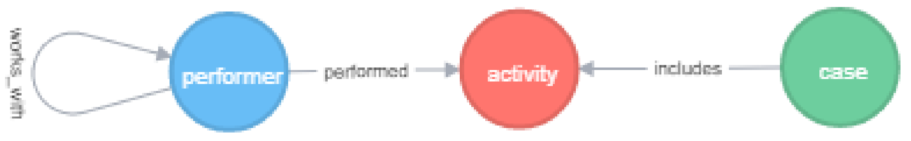
case	activity	performer	timestamp
1	A	John	1
1	B	Mike	2
2	A	John	3
2	B	Mike	4

The relationship between the two performers is very hard to identify from the CSV representation of the event log, so we propose an alternative way that makes those relationships visible. We propose first to convert the event log into a graph. As the previous work [14, 15] shows, it is very easy to mine organizational structures from Graph structures. Moreover, original theoretical work [1] also utilizes the Graph structure to define WSA.

As we can see the event log contains four columns: Case, Performer, Activity, and Timestamp. Three of those columns: Case, Performer, Activity can serve as the nodes into our graph.

Figure 2 shows the schema as a meta-graph of the proposed structure. We will motivate this structure with the example so that it is clearer why we chose to represent event log specifically with this schema.

Figure 2. Meta-graph of the proposed solution



INCLUDES relationship

The “Includes” relationship is a trivial relationship that is visible on the entry of the event log. It is between a case and an activity as in “Case one includes activity A.”

PERFORMED relationship

The “performed” relationship is a trivial relationship that is visible on the entry of the event log. It is between a performer and an activity as in “The performer John performs the activity A.”

WORKS_WITH relationship

The WORKS_WITH relationship is a non-trivial relationship that is not visible on the entry of the event log. It is between two performers as in “The performer John works with the performer Mike.” This hidden relationship connects two consecutive entries of the log to each other that is why it is so hard to deduce it by looking at a particular entry in isolation.

To identify this type of relationships first, we construct a timeline (order entries by the timestamp). Every pair of performers that directly follow one another is in the relationship of WORKS_WITH.

In its basis, WORKS_WITH relationship reflects the co-operation between two performers. We aim to uncover the nature of this relationship as it is a key to identifying the working style.

Example and Rationale

Figure 3. Graph of the example data

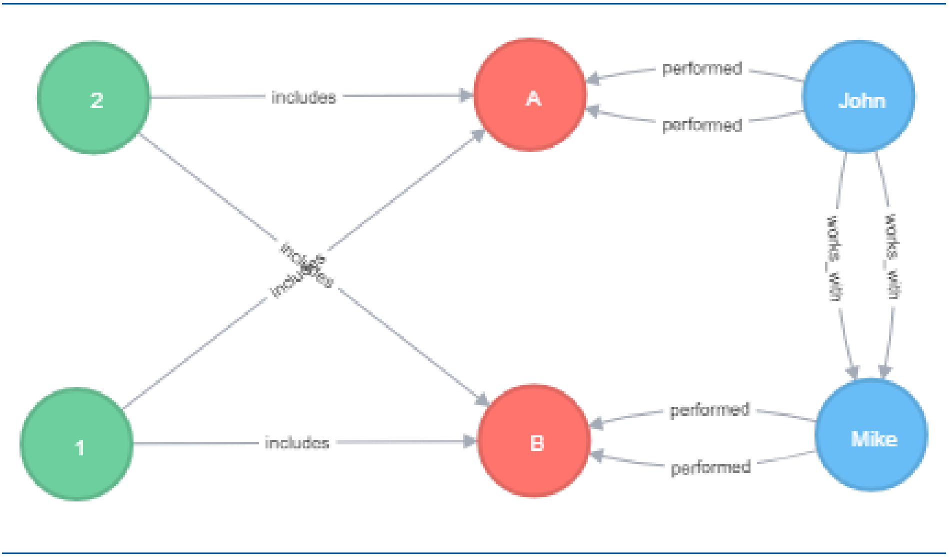


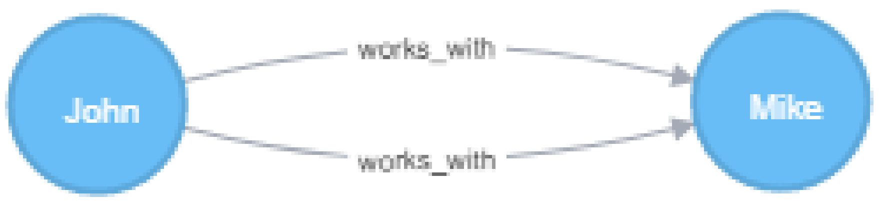
Figure 1 contains the data from a simple event log while Figure 3 contains the graph representation of that data based on the proposed schema from Figure 2.

As we can see the graph contains the same information as the event log but the relationship WORKS_WITH that we cannot see in the event log so easily, came to light and became visible in the graph representation. Also, note that columns in CSV file duplicate the values to refer the same entity while in Graph representation we have a unique node for each entity and we can have as many ingoing or outgoing relationships as we want. So while in case of CSV file every new entry duplicates the data we only need one relationship to express the same information that is of course if we already have involved parties in the Graph otherwise we create them, and that is not the duplication.

Note that method to preserve the information about the timestamp is implementation specific. We propose to save it as a property of the relationships. In other words, both INCLUDES and PERFORMED has timestamp field while WORKS_WITH has start and end fields to reflect on when (at which timestamp) has the co-operation started and when it finished.

3.2 Working Style Artifact (WSA)

Figure 4. Working Style Artifact 1

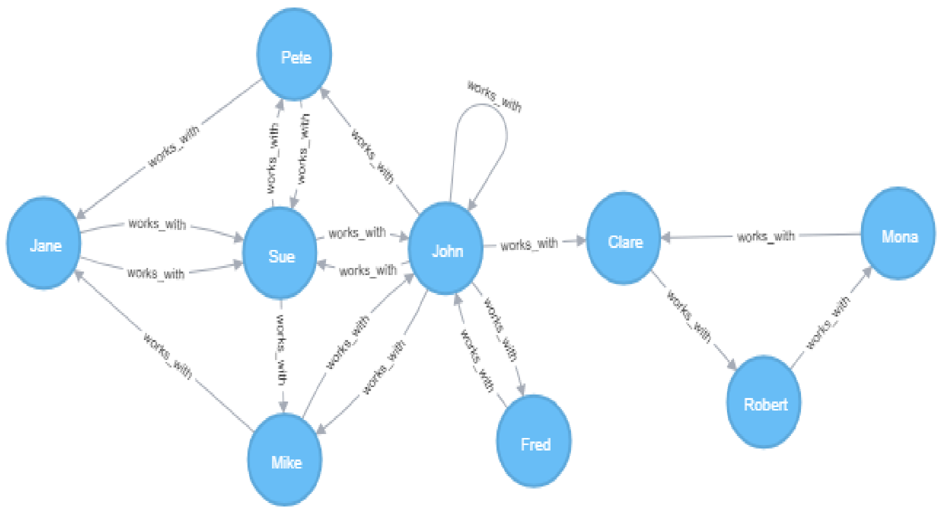


We refer to the part of the graph that exclusively contains only WORKS_WITH relationships as Working Style Artifact (Figure 4). If we enrich the WSA with context (show related activates, cases so on), we call it Working Style Extended Artifact (Figure 5).

3.3 Patterns in WSA

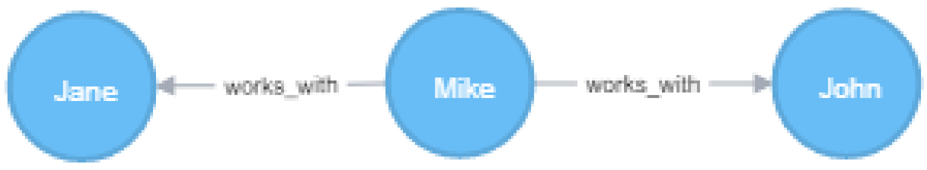
Let us discuss some of the patterns one can discover in the WSA. Figure 4 is too small of a WSA so let us take WSA from Figure 6 as an example. Depending on the use case user may want to query different patterns from WSA.

Figure 5. Working Style Artifact 2



Let’s take the following example scenario. Mike is going on the vacation and management wants to notify performers who will directly be affected by this fact. In other words, they want to identify the people who are one “works_with” away from Mike. Figure 7 contains identified pattern is a part of the WSA that contains only the information that interests the user.

Figure 6. Performers directly related to Mike from WSA 2

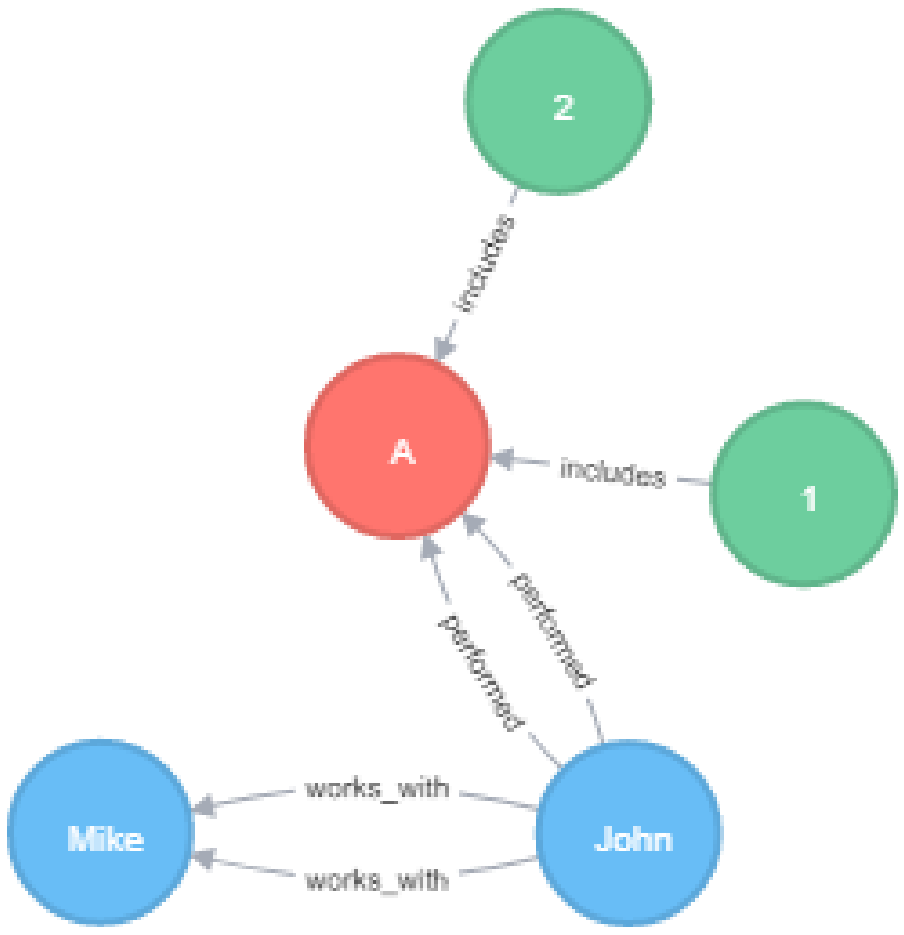


One of the goals of the tool that we develop is to let user identify the patterns in WSA. It is up to the implementing party to visualize the identified pattern in the way to be understandable for the user. We choose to bring forward (highlight) the pattern inside the WSA we call resulting visualization WSA’. The implementation section discusses more details about this.

3.4 Enriching WSA with context

The easiest enhancement we can do is to add other entities from the database as they already contain relevant information and can add useful context to WSA. Figure 7 contains extended WSA that include activities, cases, INCLUDES, and PERFORMED relationships along with performers and WORKS_WITH.

Figure 7. Working Style Extended Artifact



We can go even farther and compute parameters that can help identify the importance of the particular performer in the global scale. Also, the user will benefit from the information regarding the possible communities of performers as well as the importance of each performer in the global scale.

To detect the importance of a particular performer we propose to calculate the PageRank and to identify the communities of performers we propose to label them using Label Propagation. Later we discuss how to provide visual cues based on these parameters to increase the usefulness of the tool.

PageRank

L. Page et al. proposed the original PageRank algorithm in 1998. To put it simply it outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. The algorithm can calculate PageRank for collections of documents of any size.

Several research papers assume the evenly divided distribution among all documents in the collection at the beginning of the computational process. The Algorithm executes several iterations to adjust the approximate value of PageRank every time getting closer and closer to the true theoretical value [18].

PageRank has the same values (0 to 1) as the probability. A 0.5 probability means "50% chance" of something happening. Hence, PageRank of 0.5 means there is a 50% chance that a person clicking on a random link will appear on this document with the 0.5 PageRank.

Even though the original paper concentrates on websites and links, it has since was generalized to work on any Graph-like structure.

Lable Propagation

Semi-supervised machine learning algorithm, Label Propagation, assign labels to previously unlabeled data points. Initially, every node has a different unique community. Node changes community based on the labels of the neighboring nodes [19].

There are different metrics the most commonly used it the maximum number of labels from the direct neighbors of the node although one of the plugins of Neo4j uses different metric [20].

Every node has the unique initial label; then the labels storm throughout the network. More dense groups obtain the same label relatively quickly, thus forming the communities. Many dense (consensus) communities start to form in the network and continue to expand until it is impossible for every node to change its community label.

In contrast with other algorithms label propagation can result in various community structures from the same initial condition. Although, the range of solutions become narrow if several nodes have initial labels while others are unlabeled.

3.5 Visual Cues

Even though PageRank was originally developed in google to rank websites it has since been generalized for any Graph-based structure. Thus, we use PageRank to measure the importance of the performers in WSA. Moreover, we make the size of performer node proportional to the PageRank this will enable the user to compare performers base on their importance in WSA.

We can use Label propagation algorithm to assign labels to each performer that will indicate their community. We can use color as a visual cue to distinguish different communities on the visualization.

4 Implementation

This section describes the implementation details of the tool. Implementation satisfies all technical requirements stated in this thesis. That is, the tool uses neo4j as the database, python as the main programming language, Flask as the web development framework and Bootstrap for front-end component library.

Implementation follows the MVC pattern. HTML templates acting as views, Flask functions acting as Controllers and Neo4j database acting as a model. We separate parts of the code this way with the flexibility, extensibility, and maintainability in mind.

Implementation of the tool resides at https://github.com/MadViper/masters-project

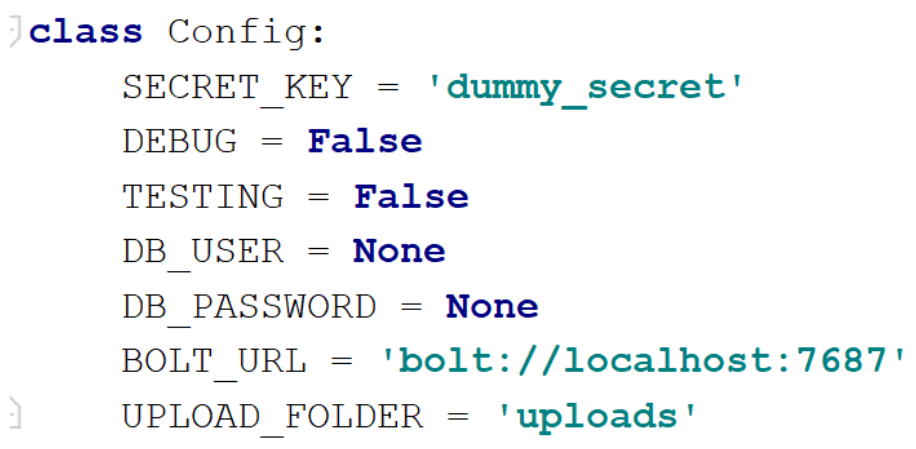
4.1 Dependencies

The application relies on the following python packages: werkzeug, gunicorn, flask, neo4j-driver. It is necessary to install the dependencies to run the application. For easy installation of dependencies, the application comes with the requirements.txt file so that it is possible to install all of the requirements at ones using the command: pip install -r requirements.txt

4.2 Configuration

Base Config class (Figure 8) defines all configurable parameters with reasonable default values. The configuration contains a secret key for Flask (it is important to set the real secret key in production configuration) debug and testing flags to indicate if the application is in either of those modes. Database user, password, and bolt URL are necessary fields for neo4j python driver to connect with the database.

Figure 8. Base configuration



```
class Config:
    SECRET_KEY = 'dummy_secret'
    DEBUG = False
    TESTING = False
    DB_USER = None
    DB_PASSWORD = None
    BOLT_URL = 'bolt://localhost:7687'
    UPLOAD_FOLDER = 'uploads'
```

Development, Testing, and Production configurations subclass the base configuration and set appropriate values for the options.

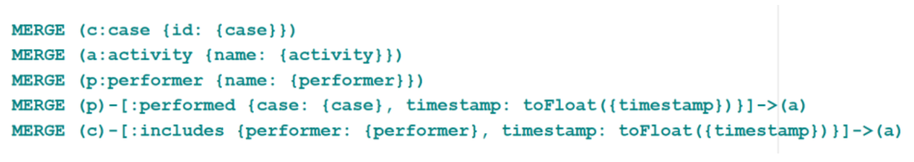
Flask supports initialization of the configuration from this kind of hierarchical structure. This structure enables the developer to create as many configuration variations as they need and later use an environment variable to provide different class references to Flask depending on the context of execution.

4.3 DBMS

Database management script is responsible for crunching the event log and creating the Graph representation from it, as well as executing graph algorithms (PageRank and Label Propagation) mentioned in the proposed solution. As we stated valid entry of the log contains case, activity, performer, and timestamp fields.

When DBMS receives the CSV file, it loops over the entries and applies Cypher from Figure 9 to initialize the nodes of the graph (cases, performers, and activities) and INCLUDES and PERFORMED relationships. It cannot initialize WORKS_WITH relationships as it retrieves entries one by one.

Figure 9. Cypher query for converting CSV data into Graph

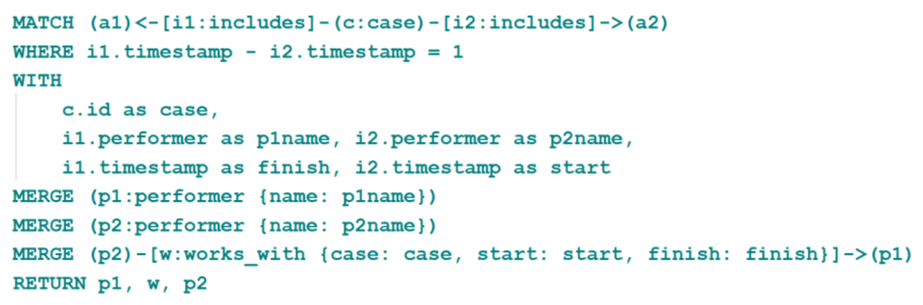


```
MERGE (c:case {id: {case}})
MERGE (a:activity {name: {activity}})
MERGE (p:performer {name: {performer}})
MERGE (p)-[:performed {case: {case}, timestamp: toFloat({timestamp})}]->(a)
MERGE (c)-[:includes {performer: {performer}, timestamp: toFloat({timestamp})}]->(a)
```

Note that, Cypher is smart enough not to duplicate the nodes if the Graph already contains it.

Cypher query language is so powerful it can create WORKS_WITH relationships from the whole data in one query, so there is no need to try and initialize WORKS_WITH along the other relationships. Figure 10 shows the query that we use to create the WORKS_WITH relationships after the creation of all nodes and other relationships.

Figure 10. Cypher query for creating WORKS_WITH relationships



```
MATCH (a1)<-[i1:includes]-(c:case)-[i2:includes]->(a2)
WHERE i1.timestamp - i2.timestamp = 1
WITH
    c.id as case,
    i1.performer as p1name, i2.performer as p2name,
    i1.timestamp as finish, i2.timestamp as start
MERGE (p1:performer {name: p1name})
MERGE (p2:performer {name: p2name})
MERGE (p2)-[w:works_with {case: case, start: start, finish: finish}]->(p1)
RETURN p1, w, p2
```

4.4 Data Access Layer

Data access layer is the class that is responsible for directly communicating with the neo4j database. Connection establishment and clean up as well as query execution is the responsibility of this class. All other entities that want to have access to the data should go through a data access layer first.

This kind of separation guarantees that even if the underlying structure of the database changes we will only have to update the implementation of this class and everything else will continue working as intended. Moreover, if we ever decide to change the backend technology, we would only have to implement a data access layer for the new one and plug it in the rest of the code.

Data access layer provides accessors to the list of cases, performers, and activities as well as the ability to run custom Cypher queries (with optional query parameters)

4.5 HTTP endpoints

This section lists the endpoints that our web application supports. We will discuss how the pages look more in-depth in the demonstration section.

HTTP GET: /uploader - renders the event log uploader page

HTTP POST: / uploader - receives the file from the user, validates it and executes DBMS functions to initialize the data.

HTTP GET: /explorer – renders data explorer page where the state of the database shows up alongside with appropriate filters

HTTP POST: /explorer – receives the form data inputted by the user, and constructs the query based on that data.

HTTP GET /creator – renders the pattern creator interface where the user can choose the desired parameters for their pattern recognition query.

HTTP POST /creator – receives the form data inputted by the user

HTTP GET /result

4.6 Cypher Query Builder

After the user enters the details of the pattern of interest, the controller function of the Flask application receives the input provided by the user and creates the object of CypherQuery-Builder class from the pattern characteristics provided. After that “build” method generates the Cypher query that can fetch the pattern that the user asked. This query is passed down to the Data Access Layer to execute and fetch results.

4.7 Jinja2 Templates

Flask supports Jinja2 template language [21] out of the box. Jinja2 enables a hierarchy of the HTML templates to reuse the common code also enables usage of python code to generate HTML. Rest of the paragraph discusses HTML templates of our web application.

base.html - Contains common functionality for all the templates such as linking CSS and JS libraries, defining the navigation bar and managing alerts.

uploader.html – Contains HTML definition of the file uploader page. Defined input fields to select the desired file and upload button see Figure 12 shows the web page rendered from this HTML.

explorer.html – Contains the HTML definition of the data explorer page. Defines area for the visualization of WSEA also input fields and drop-down lists to filter the data based on user preference. Figure 17 shows the web page rendered from this HTML

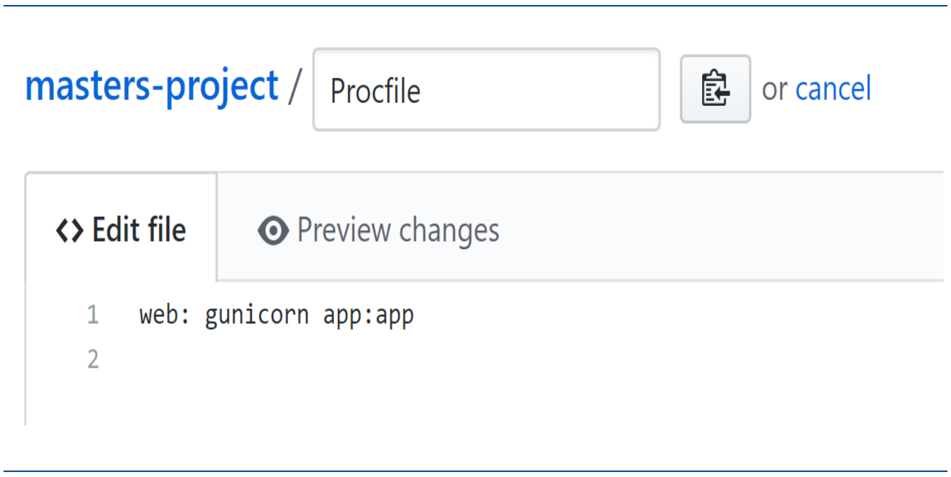
result.html – Contains HTML definition of the page with original WSA and identified pattern overlaid. Figure 21 shows the web page rendered from this HTML

4.8 Deployment

In the early stage of the development, we have been using Heroku [22] to deploy the application and make it accessible via the web. We have a Procfile (Figure 8) to enable Heroku to automate the process. At that time, we were using GrapheneDB that has a plugin for Heroku that made the deployment process a lot easier. However, after we started using Algo plugin of Neo4j GrapheneDB became useless for us because it is unable to support this plugin by design. So, our deployment cycle broke.

We tried to use Amazon EC2 in order to recover the deployment cycle. We used to maintain Amazon machine with the neo4j instance for a while, but we went out of the scope of the project once complicated system administration related issues started to show up.

Figure 11. Procfile for Heroku in the repository



```
web: gunicorn app:app
```

4.9 Logging

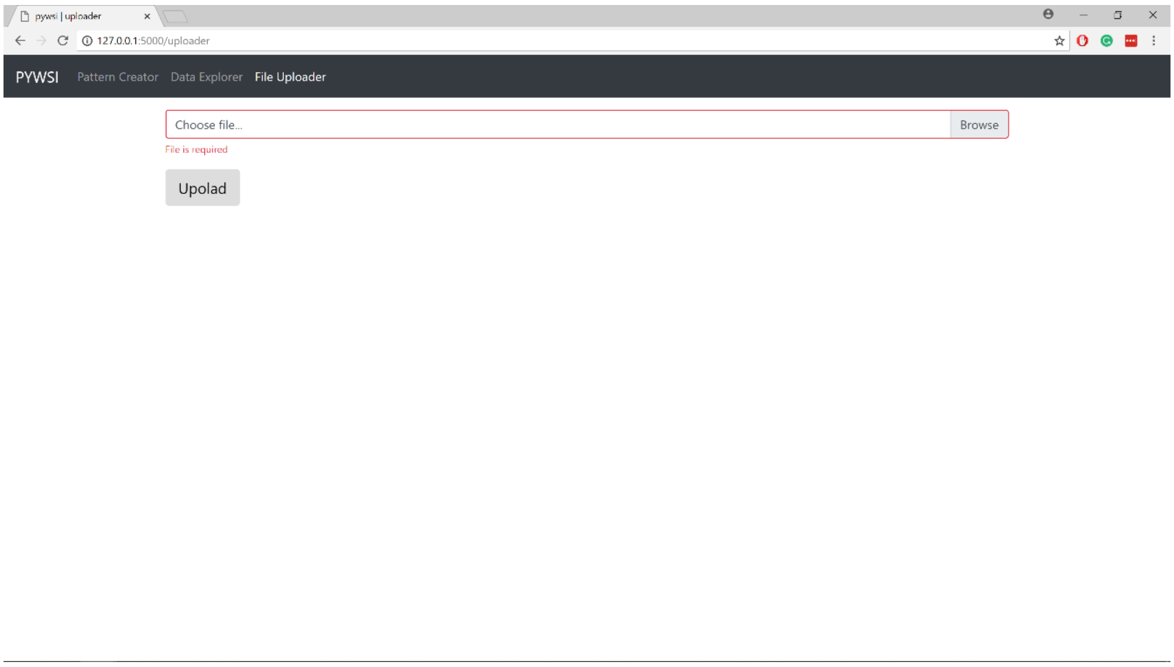
The application utilizes python’s logger module to provide descriptive logs of the execution process. It is easily configurable to log in the file, on the console, in the syslog and so forth.

5 Demonstration

This section demonstrates the developed tool in action. We explore the capabilities of the tool to evaluate and verify that all functional requirements of the thesis are satisfied. During the demonstration, we also validate the application against the user stories from Appendix II and see if we managed to provide the functionality that the user needed.

5.1 Testing Event Log Upload

Figure 12. File Uploader Page

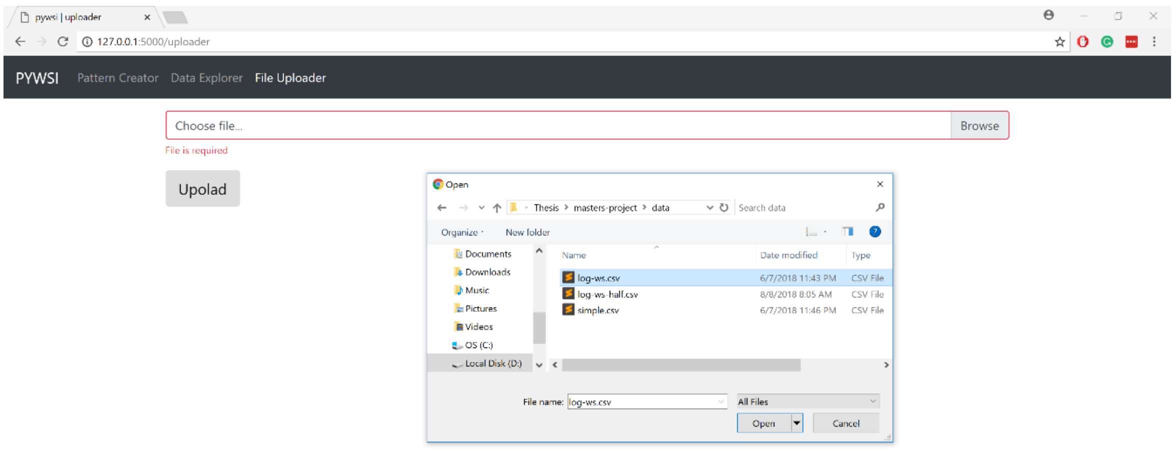


File is required

Figure 12 shows the initial state of the event log upload page. As per User Story 1, there is the browse option, and as per User Story 2, there is the upload button.

From this point, the user can follow the flow of the file upload as we discuss in this section or at any point change their mind and the page from the top navigation bar. In that case, they will end up on Pattern Creator or Data Explorer page. We will discuss their behavior in the subsequent sections.

Figure 13. Native file explorer pop-up on the upload page

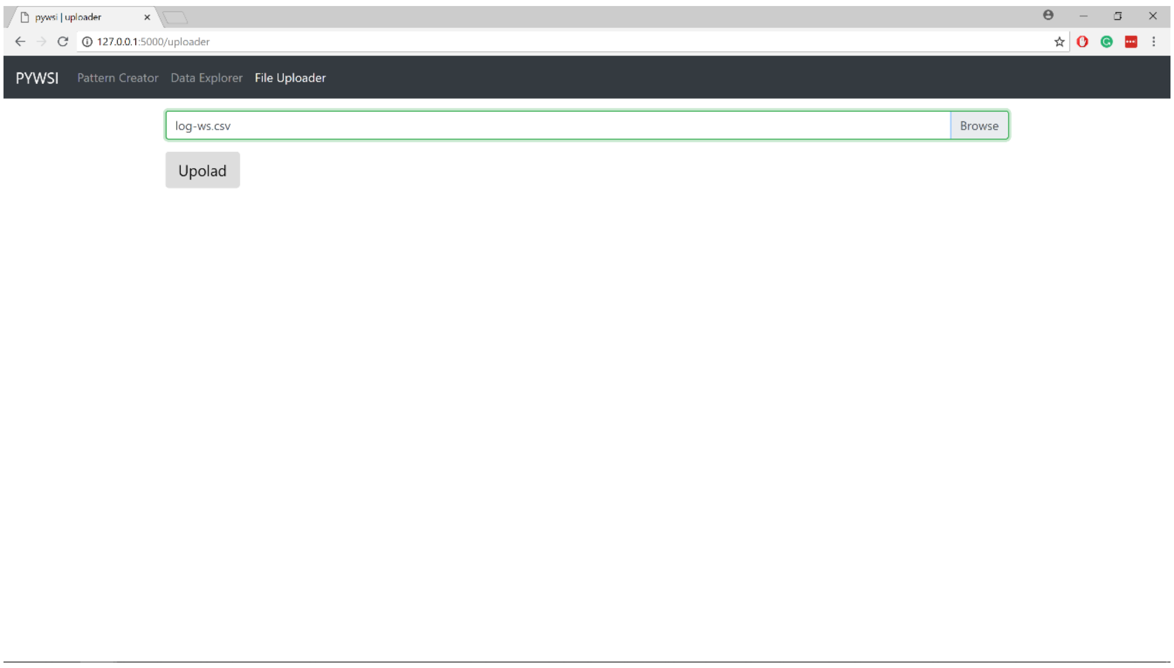


Should the user choose to follow the file upload flow, they will press on Browse (or the file field itself) operation system specific file picker dialog will pop up on the page, and they will achieve the state from Figure 13.

While in the state from Figure 13, they can search for the desired CSV file. When they find the file that they want to upload they press open.

If they reconsider at any point, they can cancel the dialog and stop the uploading attempt.

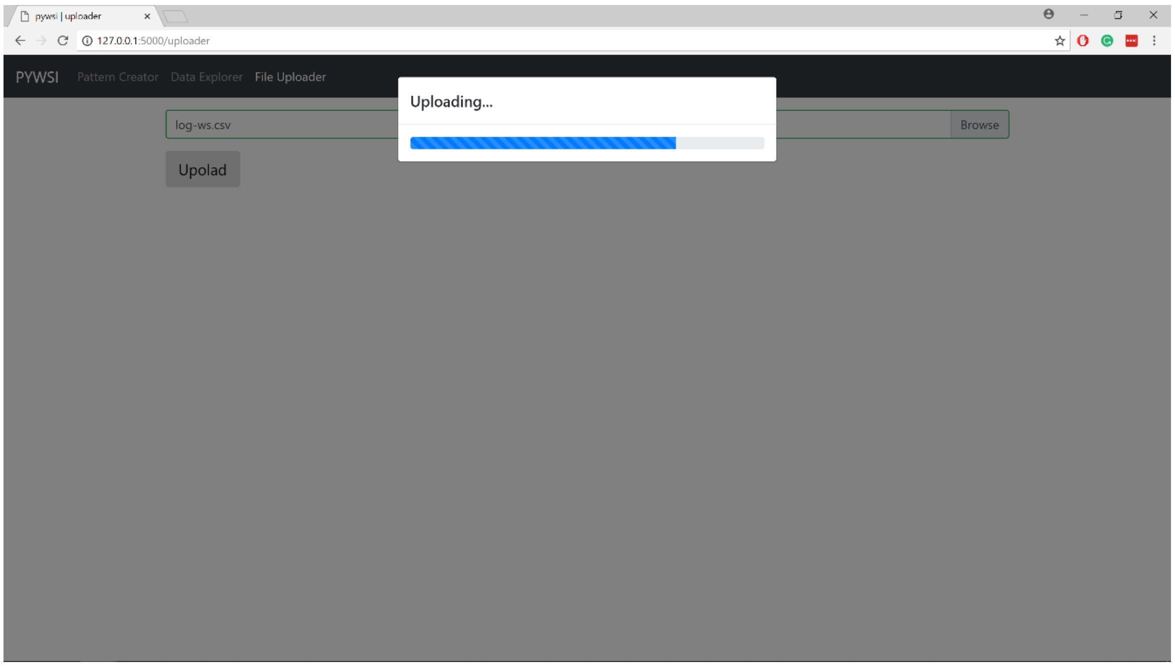
Figure 14 Upload page with the selected file



Should the user continue with uploading flow, they will reach the state in Figure 14, where the chosen file name shows up in the file field.

From this point, the user can either choose another file or upload the selected one by pressing the Upload button. If they decide to choose a different file, they will have to go through the same steps that we have discussed so far. Otherwise, they press the upload button.

Figure 15. Upload page with a progress bar

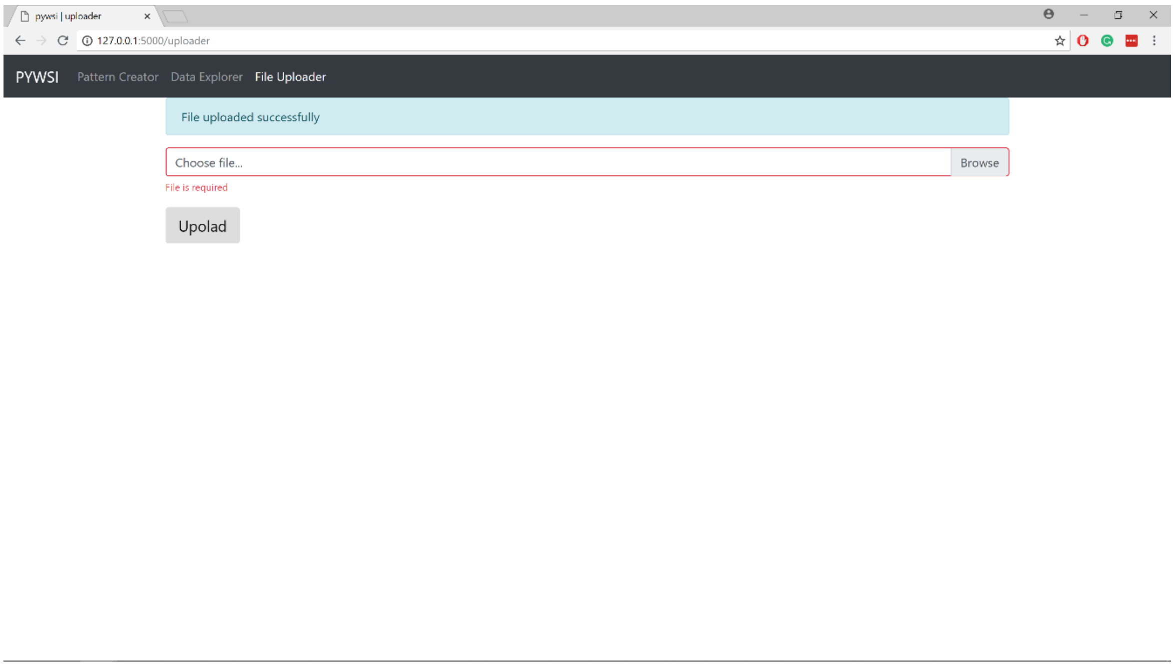


After they press the upload button Window will pop up, and the application will start tracking progress (figure 15).

Showing the progress bar to the user is important as they will not be able to do anything until the back-end crunches the data and initializes the Graph. So, for the better user experience, we show the progress bar and delay the user on the uploader page as long as it takes to initialize the Graph.

The user cannot discard the progress bar dialog. Even if they attempt to close the page entirely, the database will initialize eventually because the data processing happens on the back end.

Figure 16. Upload page with success message



File uploaded successfully

Choose file... Browse

File is required

Upolad

After the back end initializes the Graph, it will notify front end to discard the progress bar, thus bringing user in the state shown in Figure 16.

At this point, they can choose to visit other pages for example data explorer to investigate the what had they just uploaded or Pattern Creator to directly jump to working style identification.

5.2 Testing Data Explorer

Figure 17. The default state of Data Explorer

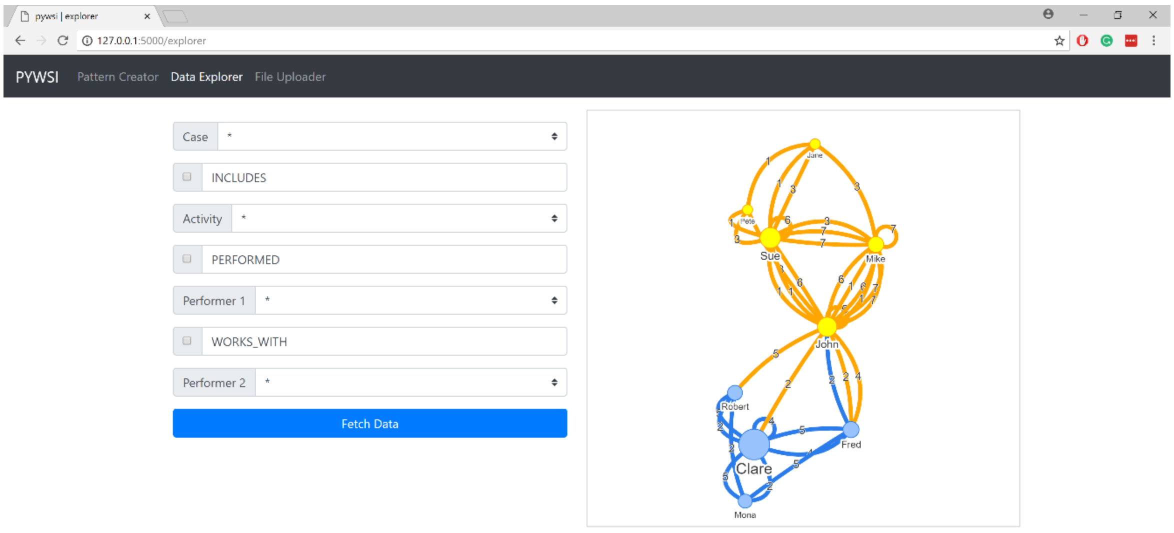
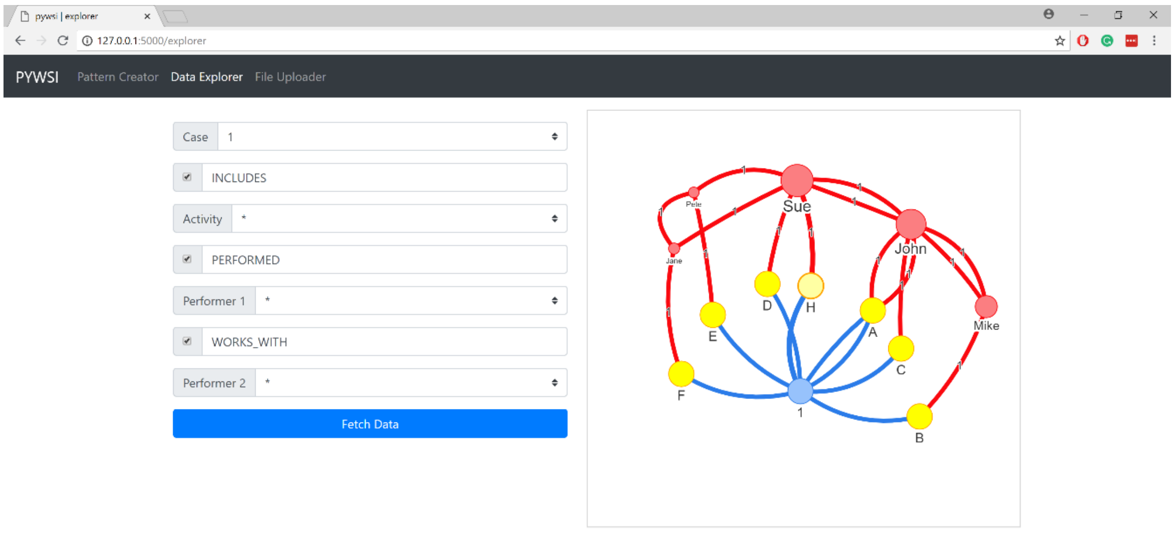


Figure 17 shows the initial state of the data explorer page. As per User Story 3, there is the fetch data button. As per User Story 4, there are control components that a user can filter the case by case ID, activity by name and performer by name. As per User Story 5, there are check boxes to exclude/include the relationships from the visualization.

At this point, if the user wants to follow data exploration flow described in this section, they can use control components (dropdown list/checkboxes) to construct different perspective of the data Figures 18, 19 and many more. They can always use the top navigation bar to go to the Pattern Creator to start working style identification or to the File Uploader to load new data.

Figure 18. View all entities related to the case with ID 1



Case: 1

☑ INCLUDES

Activity: *

☑ PERFORMED

Performer 1: *

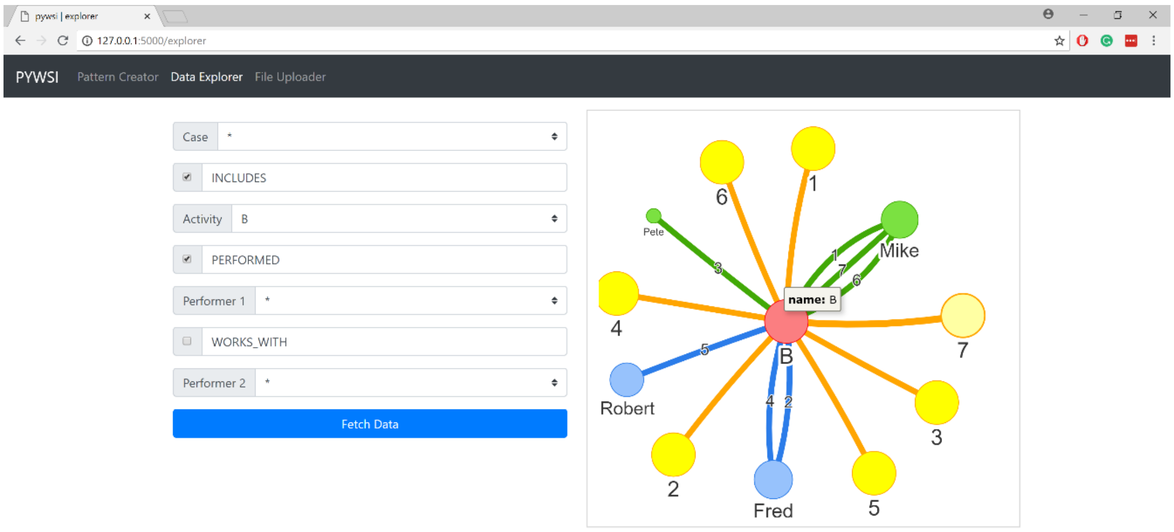
☑ WORKS_WITH

Performer 2: *

Note that Graph has visualization cues proposed in the previous section. Performer nodes have a different size proportional to the proposed PageRank property.

Moreover, they have the partition property that has the value assigned by the proposed label propagation algorithm. Performers with the same partition have the same color, and neo-vis.js automatically colors the relationships with the color of the source node.

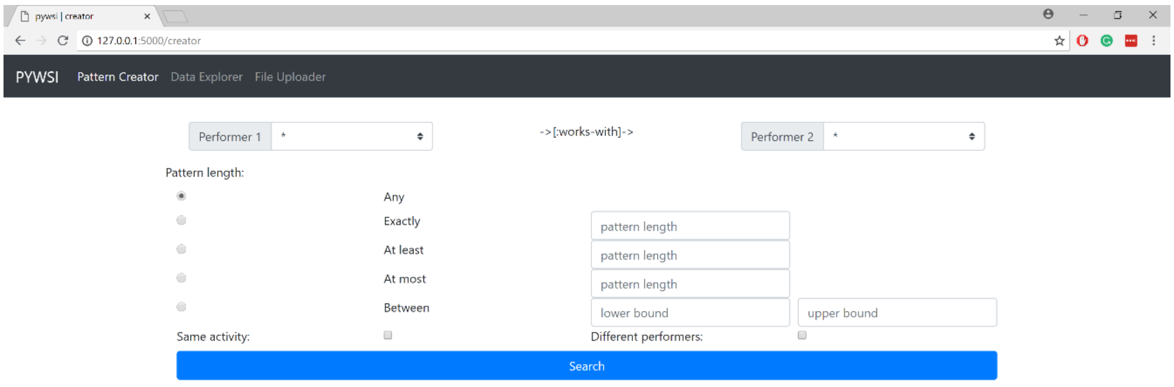
Figure 19. View all entities related to activity B



Case	*
✓	INCLUDES
Activity	B
✓	PERFORMED
Performer 1	*
☐	WORKS_WITH
Performer 2	*

5.3 Testing Pattern Creator

Figure 20. Pattern Creator Interface



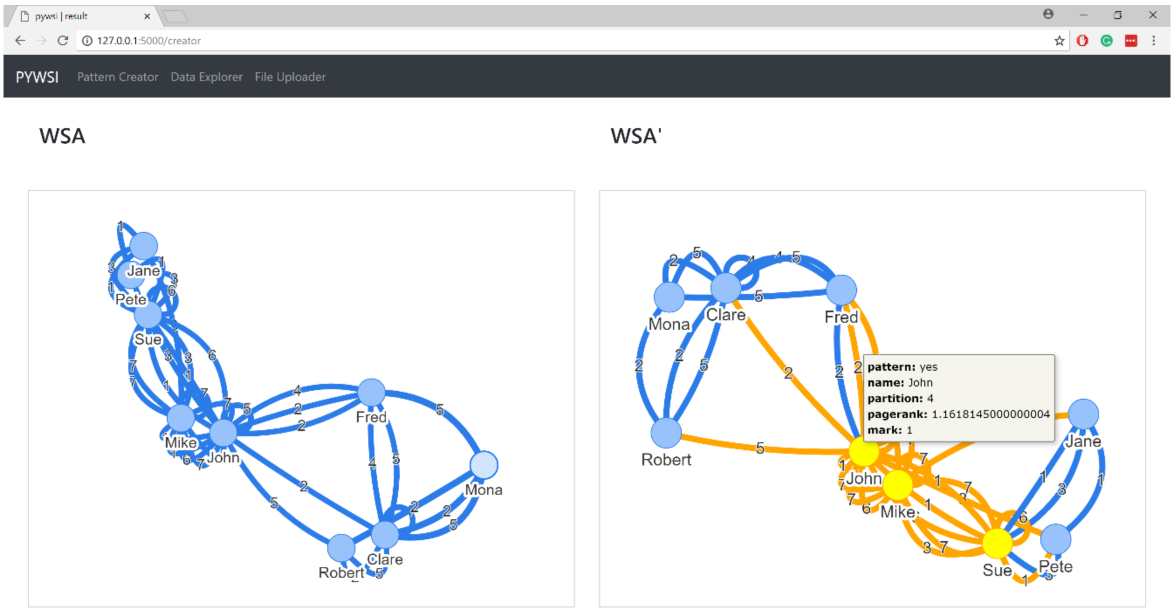
Performer 1 * ->[:works-with]-> Performer 2 *

Pattern length:

●	Any		
○	Exactly	pattern length	
○	At least	pattern length	
○	At most	pattern length	
○	Between	lower bound	upper bound
Same activity:	☐	Different performers:	☐

Figure 20 shows the pattern creator interface. As per User Story 6, there is a search button that to query for patterns. As per User Story 7, There are two drop-down lists to select the desired target and destination performers of the pattern. As per User Story 8, text fields and radio buttons to allow specification of the desired type and range of the pattern length.

Figure 21. Original WSA (left) WSA’ with the overlay of the queried pattern (right)



pattern: yes
name: John
partition: 4
pagerank: 1.1618145000000004
mark: 1

Figure 21 shows the result of the query that involves performers John and Mike and pattern length is at most 2. We can see that the application managed to identify a correct pattern. The only problem is the visualization. Because of the limitations posed by neovis.js (discussed in the limitation section), there is no consistent way to exclude all “Gateway” relationships from the pattern color. We call “Getaway” relationship those that have one end connected to the pattern and another end to the non-pattern. Noevis.js automatically assigns relationships a color of the source nodes, so some of the “Gateway” nodes show up in the color of the pattern even though they are not.

6 Conclusion and Future Work

By analyzing tests, we can assume that the goal of the thesis is reached. The outcome of the thesis is a web-based tool that aids the user with their tasks related to working style identification by consuming their event log, analyzing data and providing easy to use interface for exploring the database and searching for patterns.

Rest of this section discusses some of the future work that will enrich the tool and address current limitations.

6.1 More Graph Algorithms

Now that we have seen the power of graph algorithms and how they enrich WSA with more context and aid the visualization we propose to provide options for different Graph Algorithms and let the user decide which one they want to apply.

6.2 Customize neovis.js

One of the gaps that we identified in the current state of practice related to neo4j tools is that JavaScript visualization library neovis.js is rigid when it comes to customization of the visualization aspects, and it tends to make crucial visualization decisions automatically.

In our use case, we needed to assign a specific pre-determined color to the certain nodes (obtained using Cypher query language) and another color to the rest of the nodes. This results of choose specific color for the nodes with a specific label or to opt out from relationship coloring when we color the nodes and so forth.

The whole neo4j community will benefit if neovis.js becomes more flexible and easily customizable. As future work, we can extend the library to enable customization options. Contribute to the development of the library by extending it To address the issue Address problems with Neovis.js fix the library and contribute to help out the community.

6.3 Provide REST API

In some use cases, it will be beneficial to update the Graph data in real time. By current design, this is not possible as we only support data initialization via file uploader that takes CSV files as an input.

For example, we have stated earlier that process aware systems drive the modern workplace. These systems usually have built-in event logging and tracking functionalities. So, it will be easier for the end user if Graph database updates as soon as the system generates an event.

To support this functionality, we need to implement a way to update the Graph upon user's request dynamically. We propose to extend the tool with REST API that will have simple CRUD (create, read, update, delete) functionality and enable the client to manage the data whenever and however they want. As a result, the client can integrate their process management tool using the REST API and update it in real time.

7 References

[1] M. Sarini, "Can Working Style Be Identified?," 2017. [Online]. Available: http://ceur-ws.org/Vol-1898/paper1.pdf.

[2] A. Gao, Y. Yang, M. Zeng, J. Zhang and Y. Wang, "Organizational Structure Mining Based on Workflow Logs," *International Conference on Business Intelligence and Financial Engineering,* 2009.

[3] "Neo4j," [Online]. Available: https://neo4j.com/. [Accessed 02 08 2018].

[4] "OrientDB," [Online]. Available: https://orientdb.com/. [Accessed 02 08 2018].

[5] "Flask," [Online]. Available: http://flask.pocoo.org/. [Accessed 02 08 2018].

[6] "Django," [Online]. Available: https://www.djangoproject.com/. [Accessed 02 08 2018].

[7] "Bootstrap," [Online]. Available: https://getbootstrap.com/. [Accessed 02 08 2018].

[8] "jQuery," [Online]. Available: https://jquery.com/. [Accessed 02 08 2018].

[9] "Chrome," [Online]. Available: https://www.google.com/chrome/. [Accessed 02 08 2018].

[10] "GrapheneDB," [Online]. Available: https://www.graphenedb.com/. [Accessed 02 08 2018].

[11] "Graph Algorithms plugin for neo4j," [Online]. Available: https://neo4j.com/developer/graph-algorithms/. [Accessed 02 08 2018].

[12] "Amazon EC2," [Online]. Available: https://aws.amazon.com/ec2/.

[13] "What is SSL certificate?," [Online]. Available: https://www.globalsign.com/en/ssl-information-center/what-is-an-ssl-certificate/. [Accessed 02 08 2018].

[14] W. M. P. v. d. Aalst, H. Reijers and M. Song, "Discovering Social Networks from Event Logs," *Computer Supported Cooperative Work,* vol. 14, no. 6, pp. 549-593, 2005.

[15] M. Song and W. M. P. v. d. Aalst, "Towards Comprehensive Support for Organizational Mining," *BETA Working Paper Series,* 2007.

[16] "Disco User’s Guide," [Online]. Available: https://fluxicon.com/disco/files/Disco-User-Guide.pdf. [Accessed 02 08 2018].

[17] "bupaR," [Online]. Available: http://bupar.net/. [Accessed 02 08 2018].

[18] L. Page, S. Brin, R. Motwani and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," *Stanford Digital Libraries Working Paper,* 1998.

[19] G. Kontonatsios, A. J. Brockmeier, P. Przybyła, J. McNaught, T. Mu, J. Y. Goulermas and S. Ananiadou, "A semi-supervised approach using label propagation to support citation screening," *Journal of Biomedical Informatics,* vol. 72, pp. 67-76, 2017.

[20] "Neo4j Apoc procedures," [Online]. Available: https://neo4j-contrib.github.io/neo4j-apoc-procedures/. [Accessed 02 08 2018].

[21] "Jinja2," [Online]. Available: http://jinja.pocoo.org/docs/2.10/. [Accessed 02 08 2018].

[22] "Heroku," [Online]. Available: https://www.heroku.com/. [Accessed 02 08 2018].

Appendix

I. List of abbreviations and terms

Abbreviation	Description
CRM	Customer relationship management
ERP	Enterprise resource planning
SNA	Social Network Analyses
SSL	Secure Sockets Layer
WSA	Working Style Artifact
WSEA	Working Style Extended Artifact
PaaS	Platform as a Service
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
REST	Representational state transfer
API	Application Programming Interface
MVC	Model View Controller
JS	JavaScript

II. CSV data used for evaluation

case	activity	performer	timestamp
1	A	John	1
1	B	Mike	2
1	C	John	3
1	D	Sue	4
1	E	Pete	5
1	F	Jane	6
1	H	Sue	7
1	A	John	8
2	A	John	9
2	B	Fred	10
2	C	John	11
2	D	Clare	12
2	E	Robert	13
2	F	Mona	14
2	H	Clare	15
3	A	John	16
3	C	John	17
3	B	Pete	18
3	D	Sue	19
3	E	Mike	20
3	F	Jane	21
3	H	Sue	22
4	A	John	23
4	C	John	24
4	B	Fred	25
4	D	Clare	26
4	G	Clare	27
4	H	Clare	28
5	A	John	29
5	C	John	30
5	B	Robert	31
5	D	Clare	32
5	E	Fred	33
5	F	Mona	34
5	H	Clare	35
6	A	John	36
6	B	Mike	37
6	C	John	38
6	D	Sue	39
6	G	Sue	40
6	H	Sue	41
7	A	John	42
7	B	Mike	43
7	C	Mike	44

7	D	Sue	45
7	E	Mike	46
7	F	John	47

III. List of user stories

User Story 1	As a user, I should be able to choose CSV file from my local machine so that I can upload it to the server.
User Story 2	As a user, I should be able to upload the chosen CSV file so that I can initialize the data on the server.
User Story 3	As a user, I should be able to query the database so that I can explore and analyze the data.
User Story 4	As a user, I should be able to filter the data by case ID, activity name, performer name so that I can discard irrelevant nodes.
User Story 5	As a user, I should be able to choose which relationship types should show up on the data explorer so that I can observe the relevant connection between entities.
User Story 6	As a user, I should be able to query patterns so that I can compare WSA and WSA' to each other
User Story 7	As a user, I should be able to specify performers involved in the pattern so that I can identify the nature of co-operation between particular individuals.
User Story 8	As a user, I should be able to specify the length of the pattern so that I can identify indirect relationships between performers.

IV. License