DIGITAL NARRATIVE EXPLORER

DOCUMENTATION AND WEB APPLICATION PROPOSAL

BC. ANNA KOVAŘÍKOVÁ,
ANEŽKA FORMÁNKOVÁ

1.1 Introduction

This documentation provides descriptions, explanations and suggestions concerning COST Action INDCOR (Interactive Narrative Design for Complexity Representations) project, particularly the *Interactive digital platform to enable the public dissemination of the results of the Action* deliverable. This documentation was conceptualized during a two-month internship of Anežka Formánková and Bc. Anna Kovaříková under mag Mattia Bellini.

At the beginning, the project's aim was to create a LLM-backed chatbot that would assist users with orientation in the terminology of Digital Narratives, help them make targeted materials and overall familiarize them with the project. This was deemed unrealistic due to lack of structured data that could be used to fine-tune the LLM for the needs of the project. After picking a model to experiment with and testing its responsiveness without fine-tuning, we tried to ground the model through RAG (Retrieval Augmented Generation) on the data collected from IDN Encyclopedia which is the embodiment of the efforts on creating a Shared vocabulary, but we still were not able to reach the desired results. Eventually, we opted out of the usage of LLM entirely for the moment and suggested to use a simple rule-based chatbot that would serve as an introductory guide instead. Our suggestions on how a LLM could be integrated can be found in the section Future Work – chatbot.

We have designed a graph database from the IDN Encyclopedia data which could be used to power an interactive web application that the users could explore. We present both the database structure and our proposition for the web application design in this documentation.

1. Database

2 Tools used

The tools that have been used for this project are the following:

- `Python 3.7.0` and its libraries `uuid`, `regex`,
- `Neo4j Kernel 5.12.0`, enterprise edition,
- Neo4j Desktop application,
- Neo4j Database Management System,
- and CSV files, managed though VS Code with Rainbow CSV extension.

Python, regex library and VS Code were used for structuring the data and easier manipulation with them. The uuid library was used for generating a unique identifier for each entity. Neo4j and its query language Cypher were used to create a graph from the CSV data in the local DBMS (which is described in the Neo4j documentation as *"capable of containing and managing multiple graphs contained in databases. Client applications will connect to the DBMS and open sessions against it. A client session provides access to any graph in the DBMS"*.

2.1 Database modelling

2.1.1 Building the database

The Neo4j components that are used to define the graph data model are:

- Nodes
- Labels
- Relationships
- Properties

Data modelling process

Creating a graph data model usually involves the following process:

1. Understand the **domain** and define specific **use cases** (questions) for the application.
2. Develop the initial graph data model:
    1. Model the **nodes** (entities).
    2. Model the **labels** and **properties** of the nodes.
    3. Model the **relationships** between nodes.
3. **Test the use cases against** the initial data **model**.
4. Create the graph (**instance model**) with test data using Cypher.
5. **Test the use cases** including performance **against the graph**.

6. **Refactor** (improve) the graph data model due to a change in the key use cases or for performance reasons.
7. **Implement the refactoring** on the graph and **retest** using Cypher.

(Modified from Graph Data Modelling Fundamentals Neo4j course, https://graphacademy.neo4j.com/courses/modeling-fundamentals/1-getting-started/1-what-is-modeling/)

2.1.2 Use cases

Use cases help us model what will our users need from the database.

Examples of use cases:

- How is Ideation connected to Authoring?
- What is the relationship between Authoring and Ideation?
- Who wrote Ideation?
- Who is the main author of Ideation?
- How many sections did Frank Nack write?
- Who can I contact about Authoring?
- Where can I find more about Authoring?
- What is Ideation?
- What does Ideation mean?
- How many sections does Ideation have?
- What concepts are connected to Authoring?
- What are the keywords of Authoring?

2.1.3 Data storing

Currently, the database consists of the data formatted as CSV and converted into a graph database (so far only in local DBMS) through the Neo4j Desktop application.

As of now, there are 4 CSV files:

- `concepts_node_list.csv`
- `segments_node_list.csv`
- `persons_node_list.csv`
- `edge_list.csv`

Given the lack of the "custom sections" for most of the Concepts, as of time of creating this database, file `custom_sections.csv` has not been created, but its setup is advised for future collaborators.

All nodes list files contain the information about individual nodes and their properties (Figure 1-5).

```
source,target,relationship
"d6ee5606-211c-4069-ae7b-463984e56927","127238ee-cd06-4381-991c-3fbbadba49ff",HAS_SECTION
"d6ee5606-211c-4069-ae7b-463984e56927","7655ea3d-8208-4fb9-9c5c-c80732252fe2",HAS_SECTION
"d6ee5606-211c-4069-ae7b-463984e56927","042153d7-7551-4de8-bcf4-12e3582d61c7",HAS_SECTION
"d6ee5606-211c-4069-ae7b-463984e56927","3f377a8e-7f4d-4369-b990-8f924566e390",HAS_SECTION
```

Figure 1 *Preview of the file edge_list.csv*

```
id,"taxonomy_number",name,type,alternative names,
"2c08aa89-fe56-4cf5-ae80-8933482474b5","0","Interactive Digital Narrative","Concept"
"d6ee5606-211c-4069-ae7b-463984e56927","1","Authoring","Concept","Design(ing);Narrative engineering;Writing;Interact
"127238ee-cd06-4381-991c-3fbbadba49ff","1.1","Ideation","Concept",
"f8538f9a-d13c-4ec9-b9fe-a309b9cfed3e","1.1.1","Affordances","Concept",
```

Figure 2 *Preview of the file concepts_node_list.csv*

```
id,type,text
"3995b95e-448c-40aa-bf5c-9cf4f4084887","Definiton","uri"
"ab22df28-f1a4-4fec-9464-9edd146a1746","Definiton","uri"
"cef1acbf-49c8-4ffa-85e6-1a6235df508e","Definiton","uri"
"20855d01-ab18-4713-83ac-4a02aafd32d1","Definiton","uri"
```

Figure 3 *Preview of the file segments_node_list.csv*

```
id,type,name,
"b2e987b7-1216-49c5-993b-5a3b397f1c26","Person","Frank Nack",
"e2f30fe1-a2b9-4247-9331-9bc6a3a9c791","Person","Péter Kristóf Makai",
"78c3311d-25d8-47c8-a471-13d19021b3ae","Person","Mirjam Palosaari Eladhari",
"9090f79f-a387-46ed-80f6-a4239c5f0404","Person","Shafaq Irshad",
```

Figure 4 *Preview of the file persons_node_list.csv*

2.2 Database architecture

The current database, dealing with the data from IDN Encyclopedia is displayed in *Figure 5.*

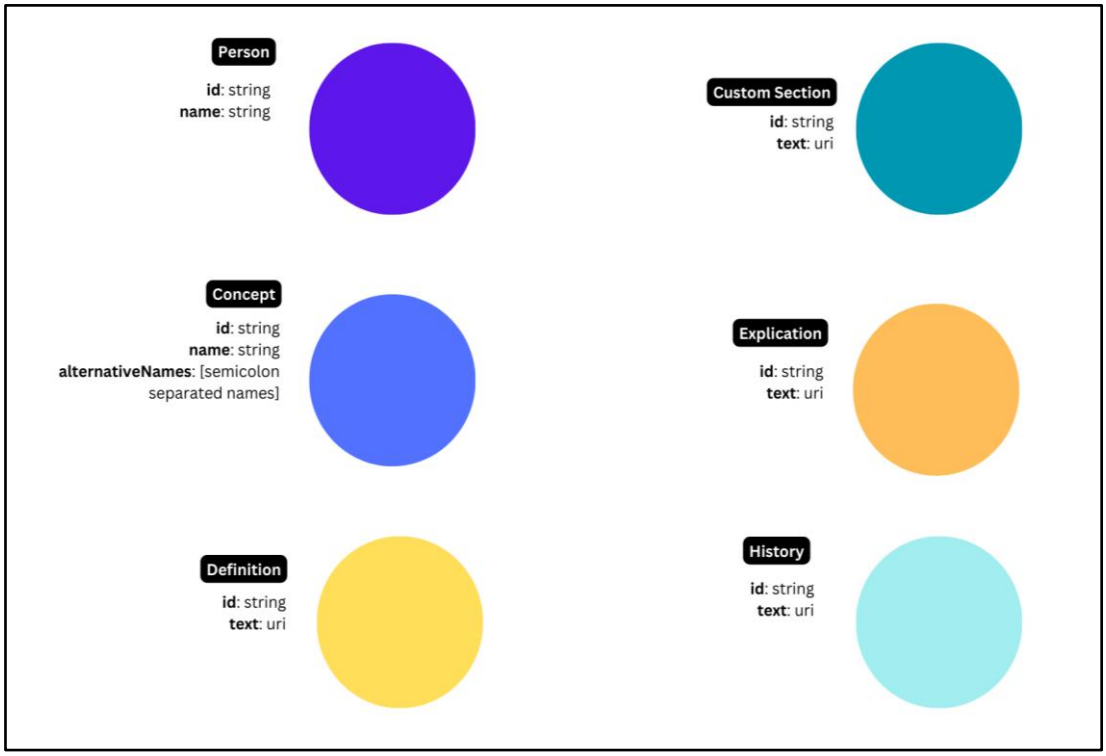


Figure 5 *Model of Node labels and properties*

The database contains the following node types: **Person**, **Concept**, **Definition**, **Explication**, and **History** (**Custom Section** has been omitted for aforementioned reasons, but we kept it in the illustration for integrity). All node types have a unique *identifier* property that serves as the node's primary key. All IDs have been randomly generated through the Python uuid library with uuid4.

Person nodes also contain property *name* in the form of a string and, in case of a contact person role, also contact information. Concept node has a property *name* and then a list of *alternative names* (if it has any) as semicolon separated values that Neo4j can split and create a list. We would also advise to add the property of "tags" or "keywords" in the future, once the Encyclopedia has been completed, for an ease of search and classification.

Node type Definition, Explication, History and Custom Section only have one other property, apart from ID, which is *text*. Text property would contain uri to another database storing the full texts of given sections from which the text could be retrieved when called upon with the uri.

The edge list file contains the information about the relationships between all the nodes stored as a list of source-target values. Sources and targets in this list are all represented by their unique ID. Third column contains the information about the type of relationship. Graph database software can then represent the relationships as shown in Figure 6.

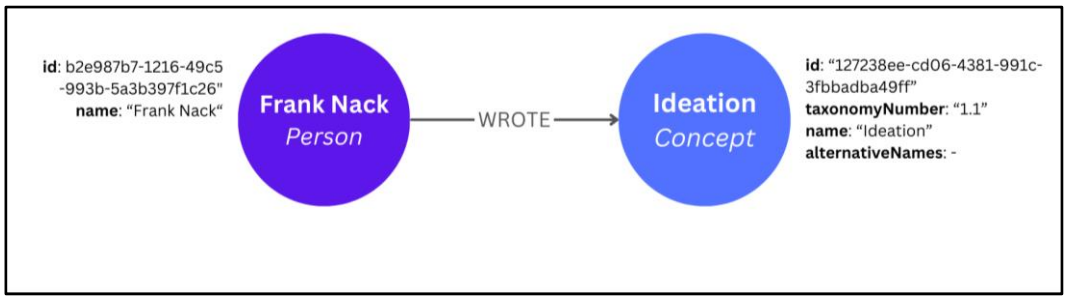


Figure 6 *Example of a relationship between two nodes*

Definition, Explication, History and Custom Section node types are all related to the Concept node type with the following relationships: `HAS_DEFINITION`, `HAS_EXPLICATION`, `HAS_HISTORY`, `HAS_CUSTOM_SECTION`, as can be seen on Figure 7.

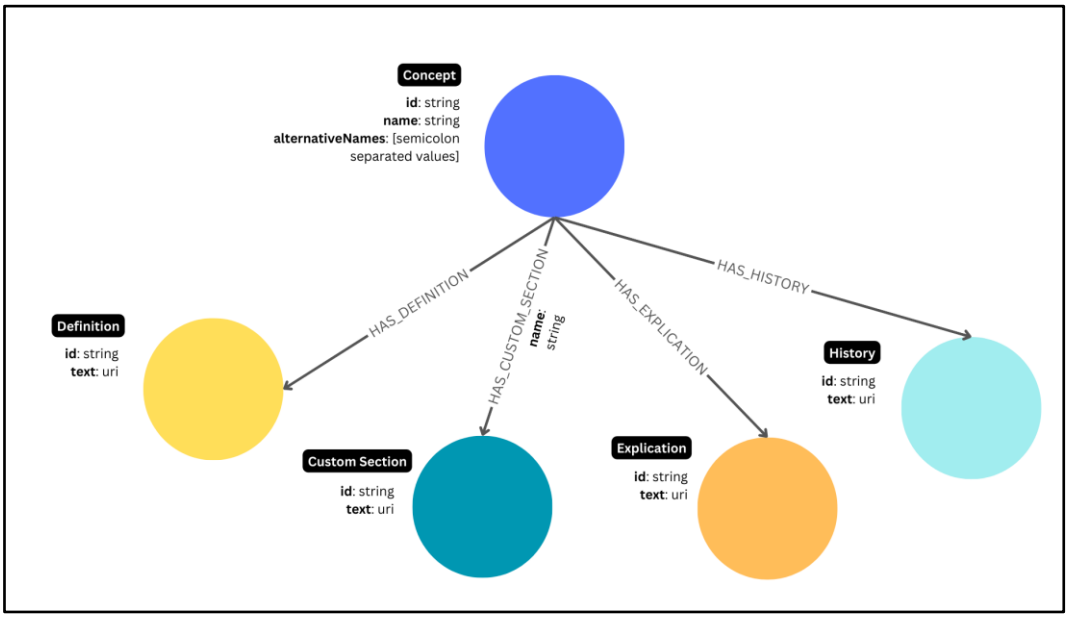


Figure 7 *Small instance model*

The full instance model is depicted by *Figure 8*.

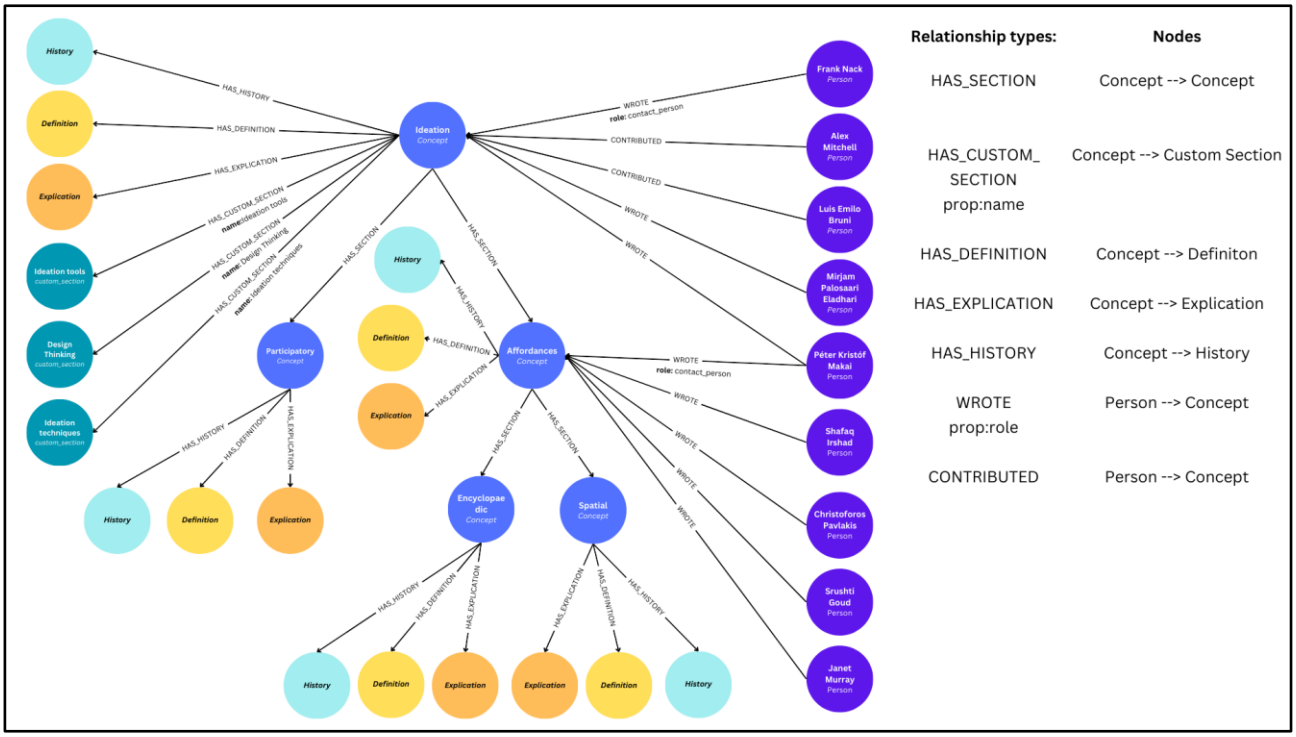
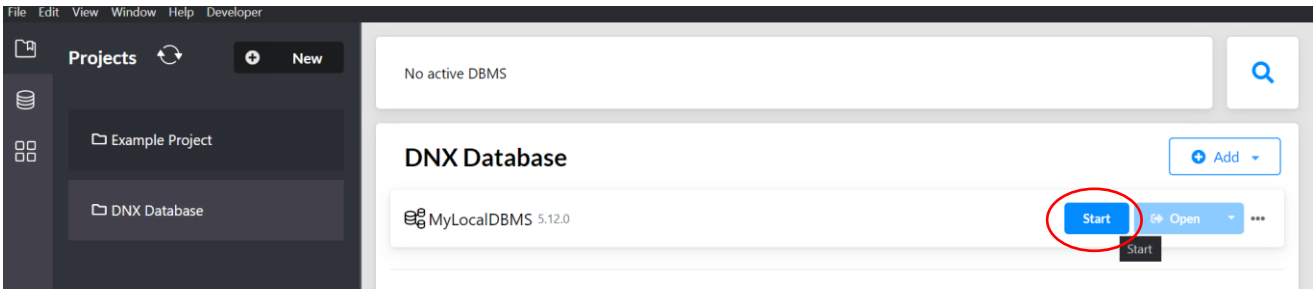


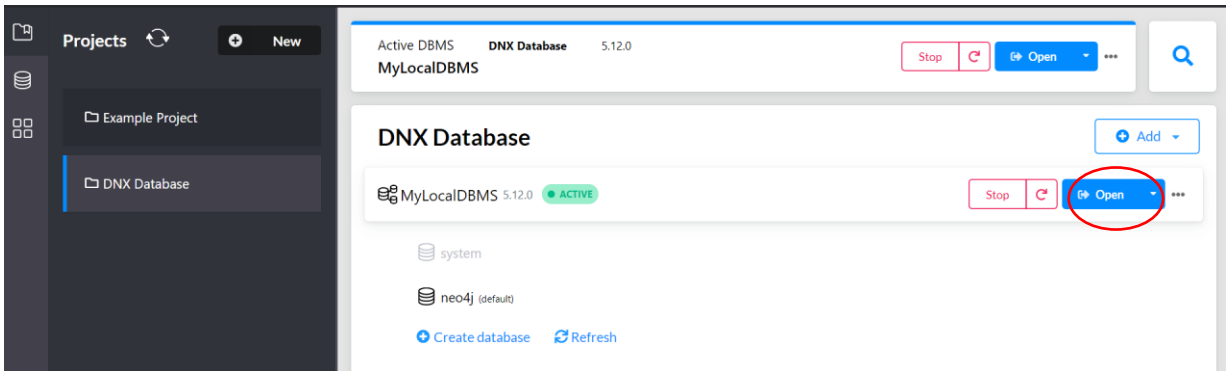
Figure 8 *Larger Instance model*

So far, the database has been designed to only deal with data from IDN Encyclopedia which is currently in the process of being written and is planned to be completed by the end of the year. Database could of course be expanded by further sources (like conference proceedings) and thus be enriched by more nodes. Expanding the database would lead to conceptualization of more nodes and relationships in a similar way to IDN Encyclopedia nodes. Further node Sources could be added which would become the top-most nodes of all the added nodes and relationships from the given source.

2.3 Running the program

The following code was used for loading the database into the local DBMS (database management system) via the Neo4j Desktop application.





```
LOAD CSV WITH HEADERS FROM 'file:///path//to/your//import//con-
cepts_node_list.csv' AS row

CALL apoc.merge.node ([row.type], {id: row.id, taxonomyNumber: row.taxon-
omy_number, name: row.name, alternativeNames :coalesce(split(row.alterna-
tive_names, ";"), "-") })

YIELD node RETURN node
```

```
LOAD CSV WITH HEADERS FROM 'file:///path//to/your//import//per-
sons_node_list.csv' AS row

CALL apoc.merge.node ([row.type], {id: row.id, name: row.name}) YIELD node RE-
TURN node
```

```
LOAD CSV WITH HEADERS FROM 'file:///path//to/your//import//seg-
ments_node_list.csv' AS row

CALL apoc.merge.node([row.type], {id: row.id, text: row.text}) YIELD node RE-
TURN node
```

```
LOAD CSV WITH HEADERS FROM 'file:///path//to/your//import//edge_list.csv' AS
row

MATCH (n), (m)

WHERE n.id = row.source AND m.id = row.target

CALL apoc.merge.relationship(n, row.relationship, {}, {}, m) YIELD rel

RETURN count(*)
```

Where `'path//to//your//import//concepts_node_list.csv'` is replaced by your true path to Neo4j imports and for files concepts_node_list.csv, segments_node_list.csv, persons_node_list.csv and edge_list.csv.

`LOAD CSV WITH HEADERS FROM`	Loads the files into Neo4j.
`CALL apoc.merge.node([row.type],`	Creates nodes with dynamic labels if they don't already exist.
`{id: row.id, taxonomyNumber: row.taxonomy_number, name: row.name`	Uses the value of *row.type* as a dynamic label, value of row.id as a primary key, value of *row.taxonomy_number* as property taxonomyNumber and value of *row.name* as property name.
`alternativeNames :coalesce(split(row.alternative_names, ";"), "-") })`	Splits the strings by their delimiters (loading from CSV doesn't support python-like lists of strings) and if the property is Null, sets it to it "-" (Neo4j can't load Null as a property).

```
MATCH (n), (m)
```

```
WHERE n.id = row.source AND
m.id = row.target
```

Finds the nodes, for which the relationship is to be defined by their id.

```
CALL apoc.merge.relation-
ship(n, row.relationship,
{}, {}, m)
```

Creates a directed relationship between n and m dynamically (using the value of row.relationship) if it doesn't already exist.

After executing the code above, the database should look like this:

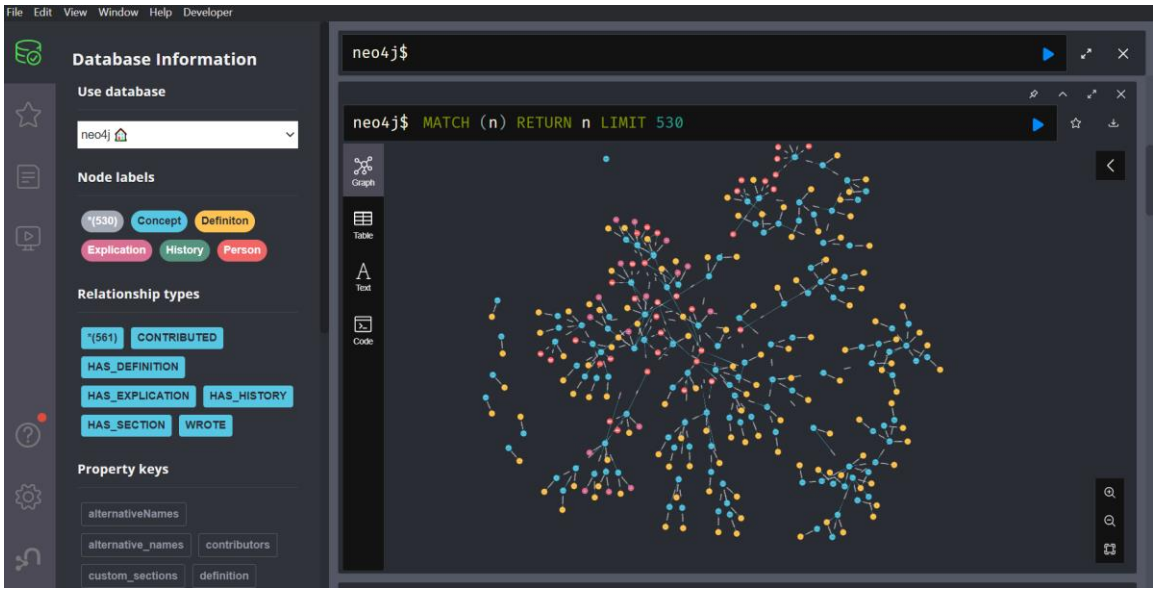


Figure 9 *IDN Encyclopedia graph in Neo4j Desktop*

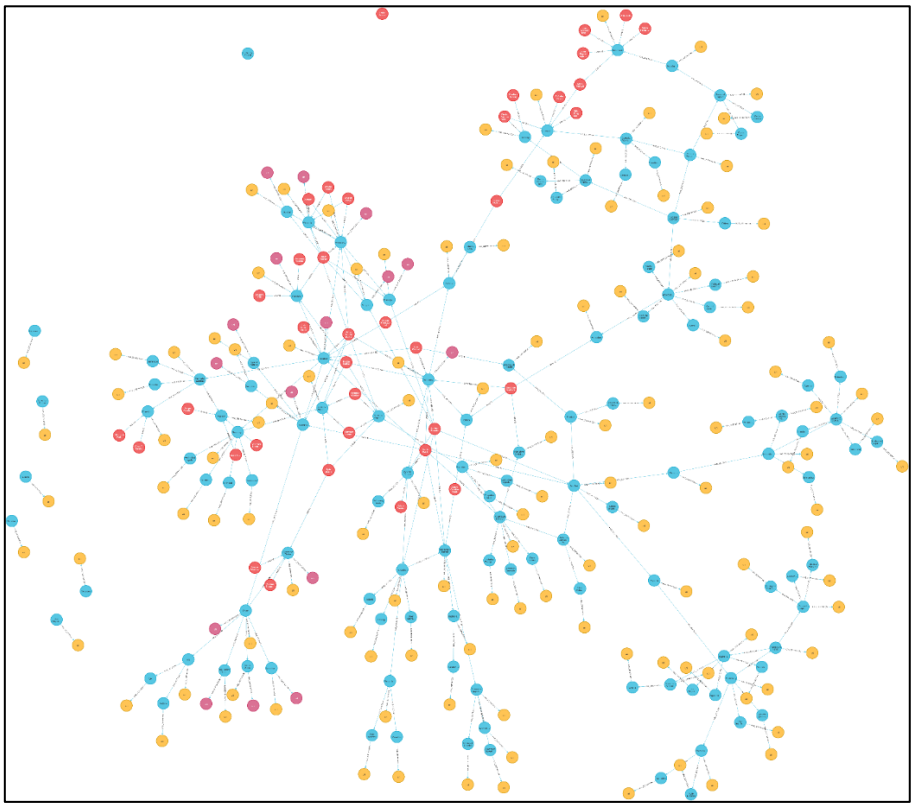


Figure 10 *IDN Encyclopedia graph*

2.4 Future work

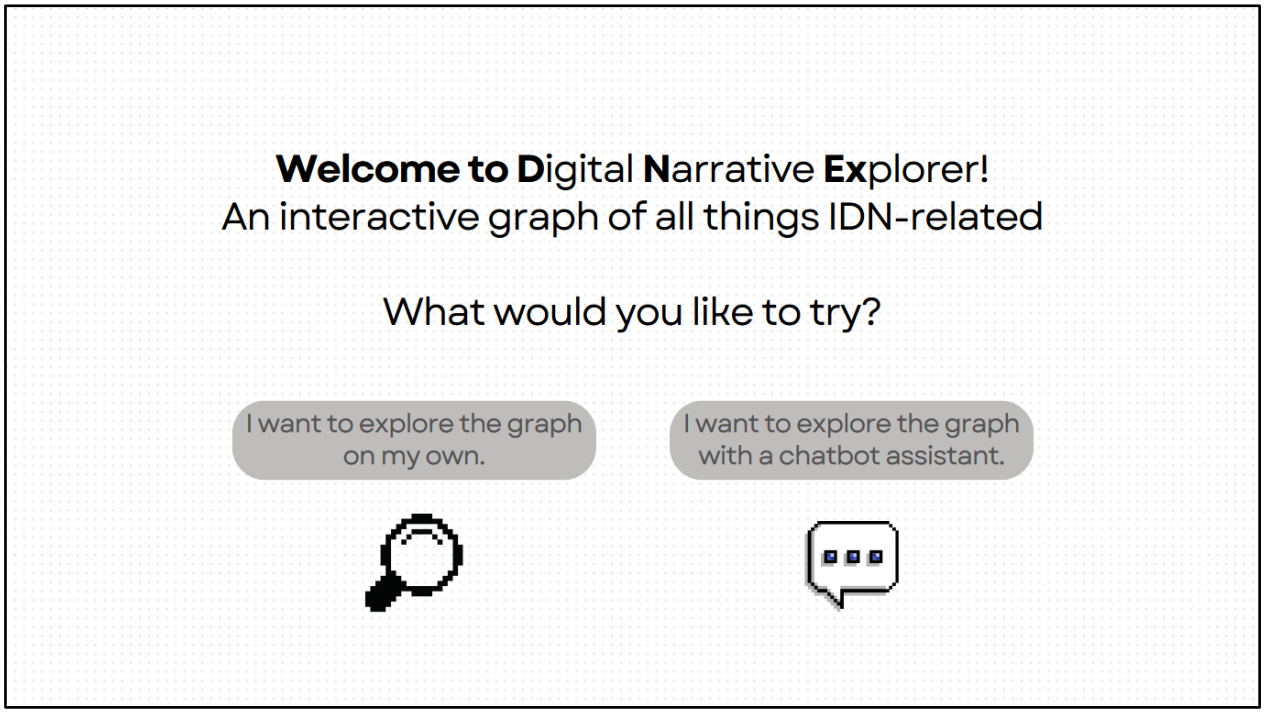
The current code does not contain the information about the contact status of a person that would be turned into the property of the WROTE relationship (because, as of writing this documentation, the Encyclopedia has not been finished yet and so the authorship of some nodes may change or expand). It also does not create any CustomSections because there has been only one Concept with Custom Section as of the time of writing.

The current hope is for IDN Encyclopedia concepts to be one of many "introductory nodes" presented to the user. Future expansion of the database might include conference proceedings, white papers, books and more, produced by the INDCOR COST action project. The mining of the data from these materials will follow a similar process – drafting of the use cases, identifying the possible entities (nodes), their properties, and the relationships that connect them, and then creating structured documents that follow this structure. Each data source will need a tailor-made solution (since each source contains different types of data and has different structures).

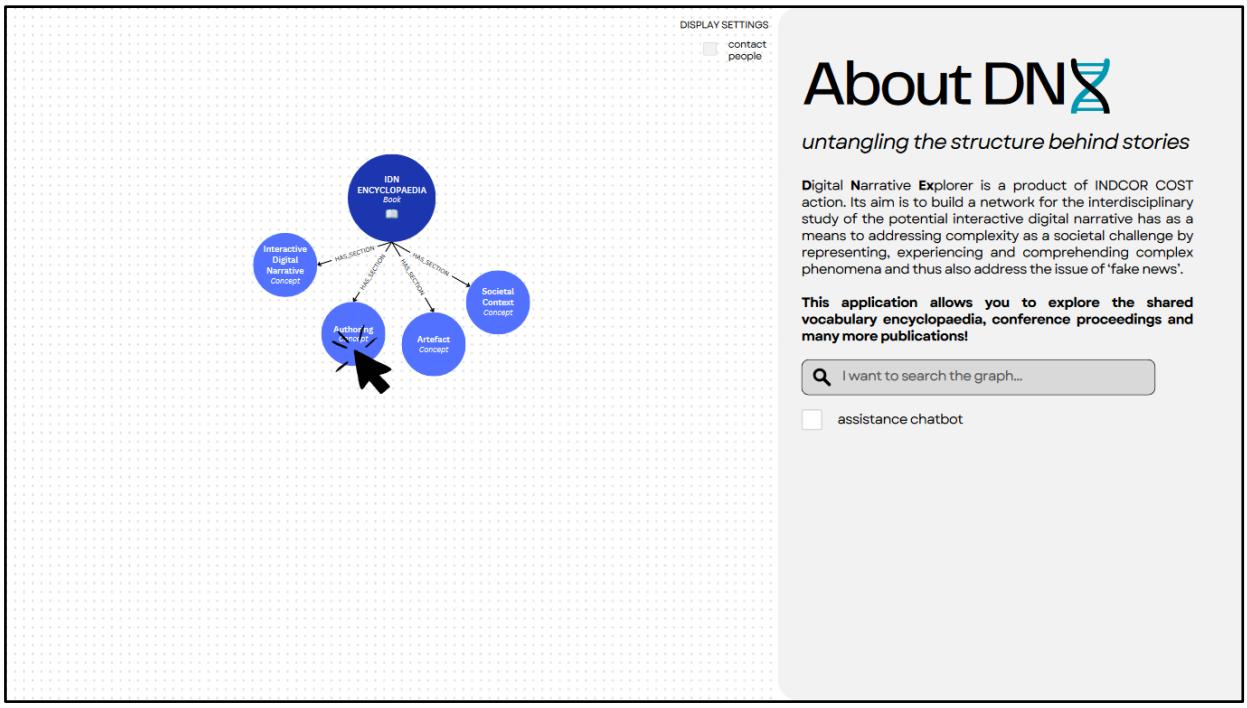
3 Web application

Digital Narrative Explorer (DNX) is a web application proposal that offers users to explore the concepts of digital narratives (and with time even conference proceedings and other materials) through an interactive knowledge graph.

Users can browse on their own, get a tutorial walkthrough and assistance from a rule-based chatbot or search the graph semantically.

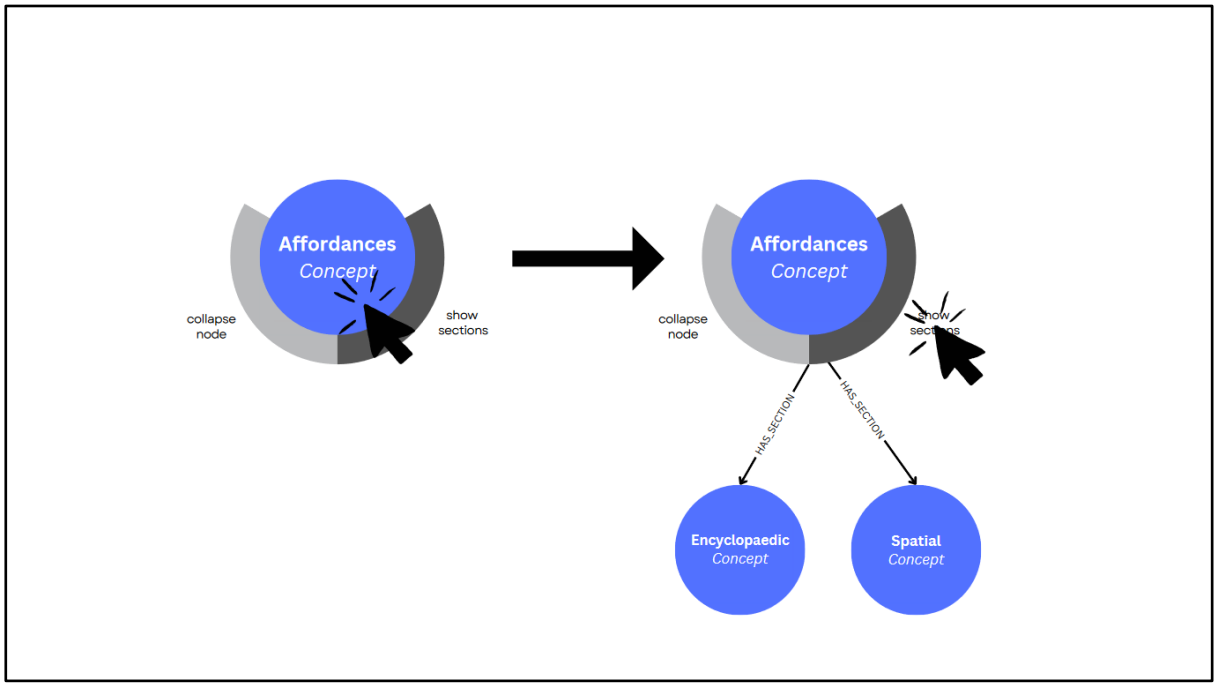


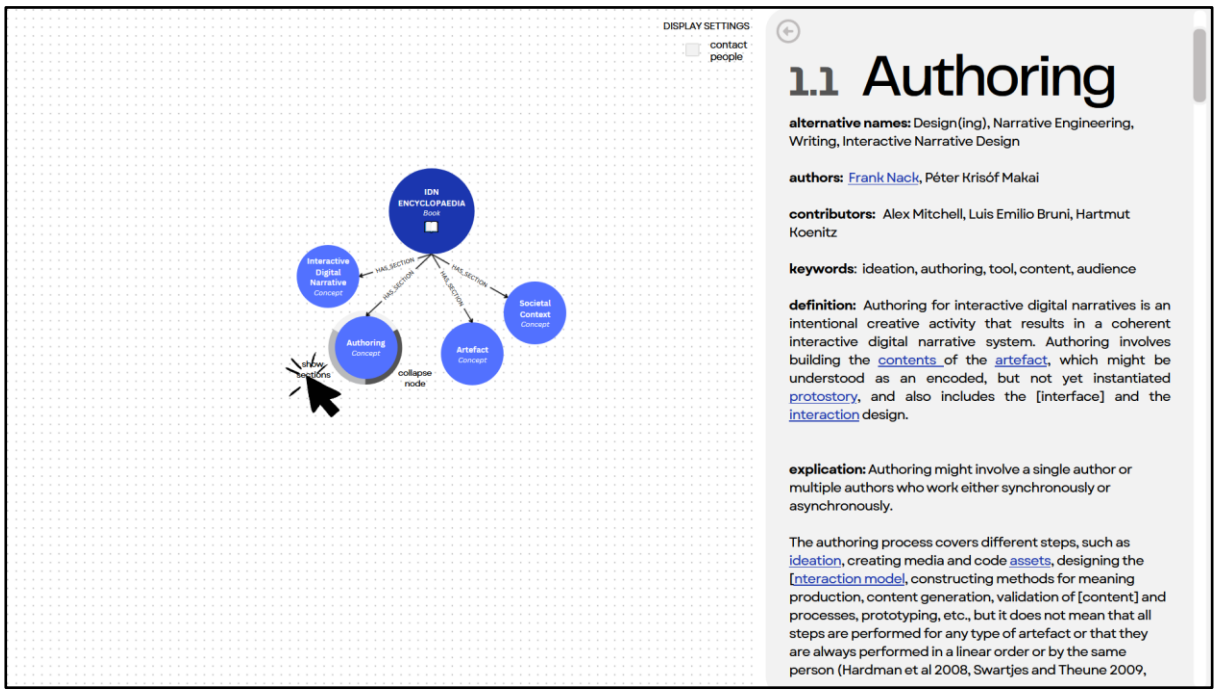
The first picture illustrates the initial option for the user to explore the graph on their own or exploring it using a chatbot assistant that will show them where to start, how the graph works and look up the nodes and for them.

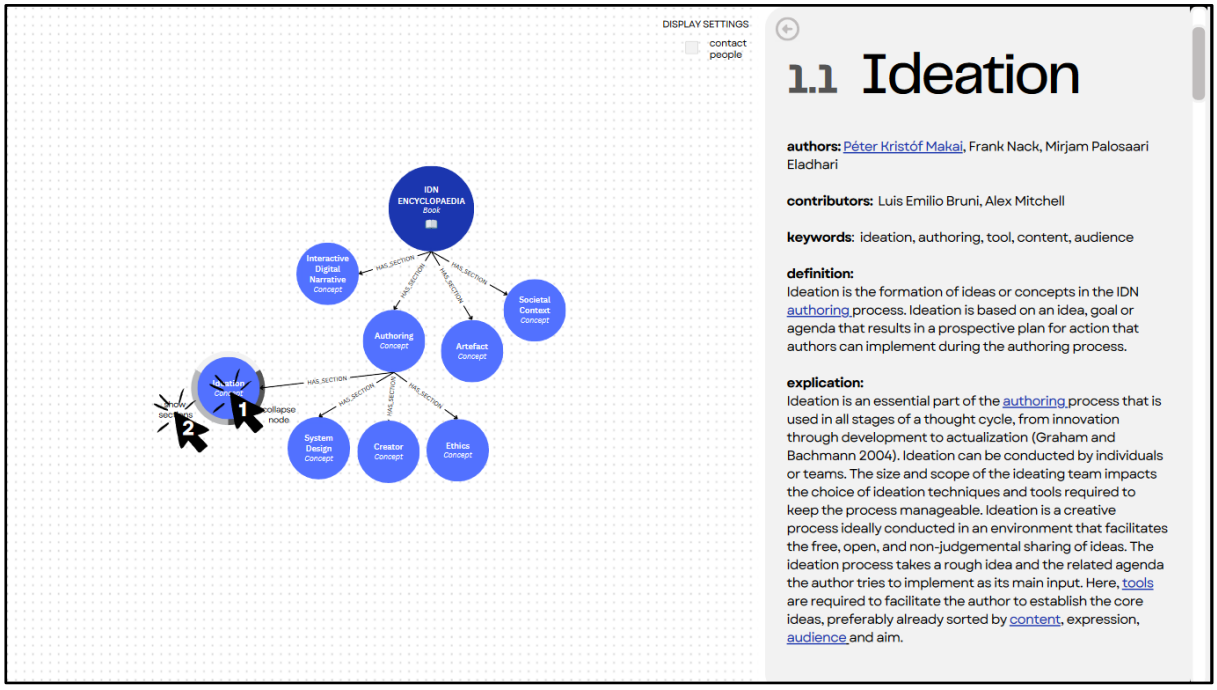


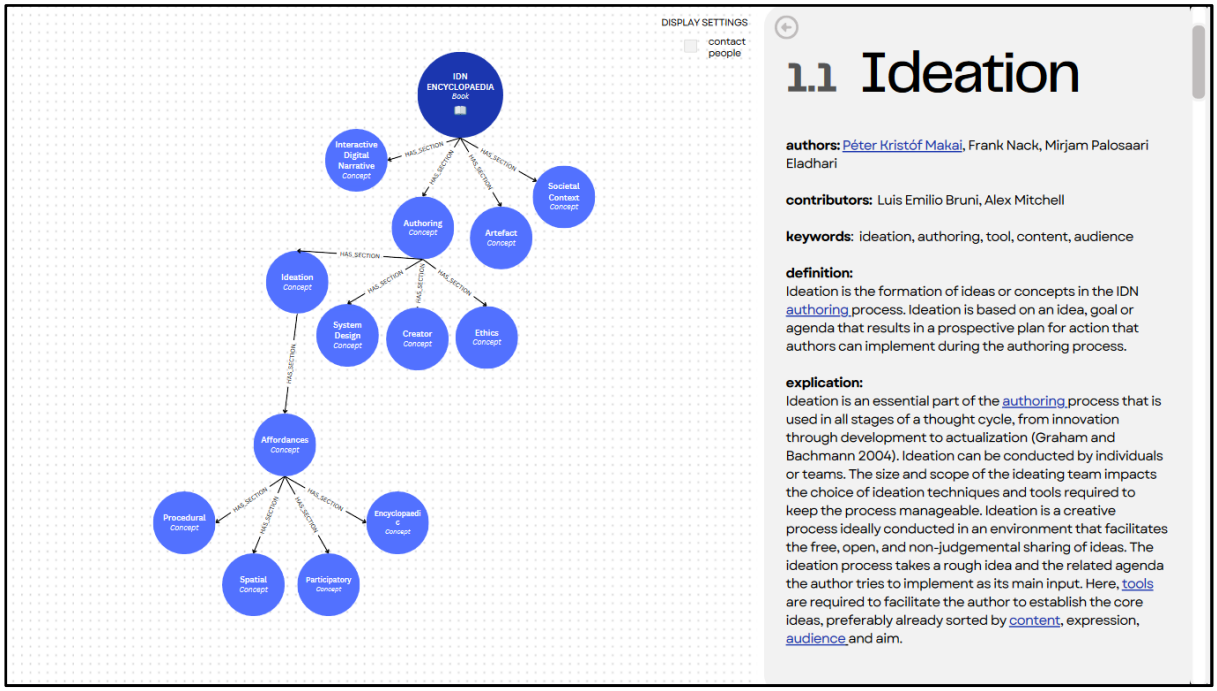
If the user chooses that they want to explore on their own, they will get an "endless whiteboard space" with the node IDN Encyclopedia and a slide-on sidebar tab that shows Introduction to the project by default.

Users can also search the graph through a search bar, looking for specific nodes. If they want, they can also turn on the assistance chatbot after all.

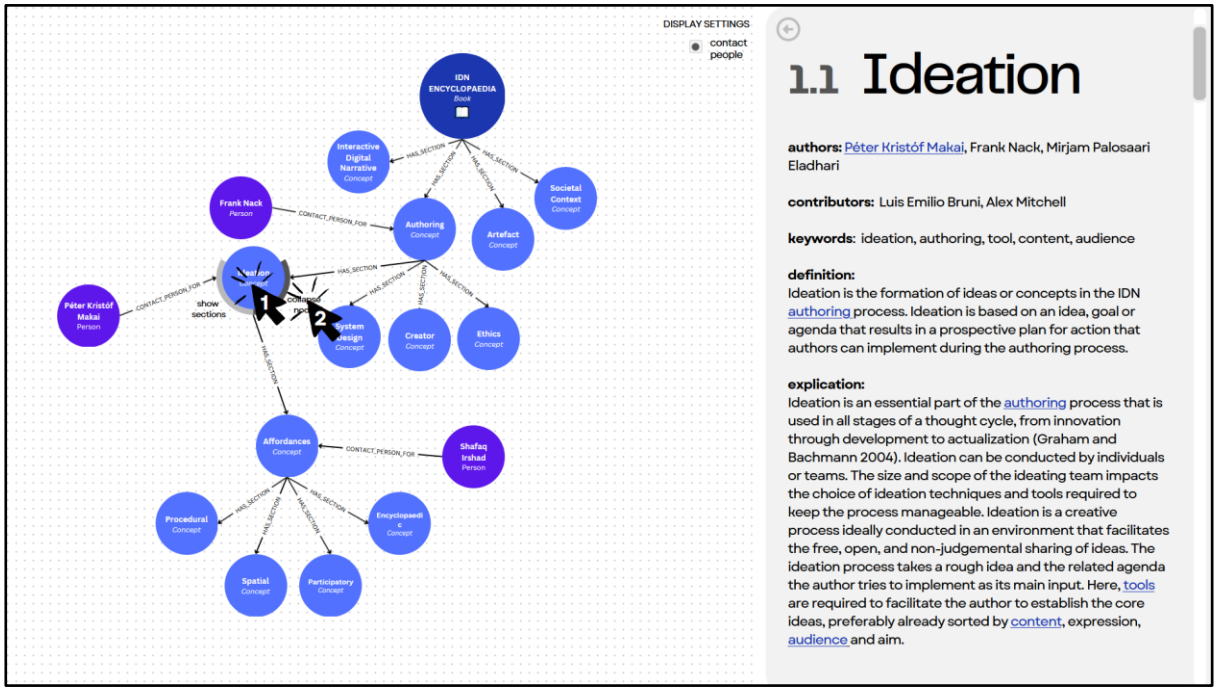


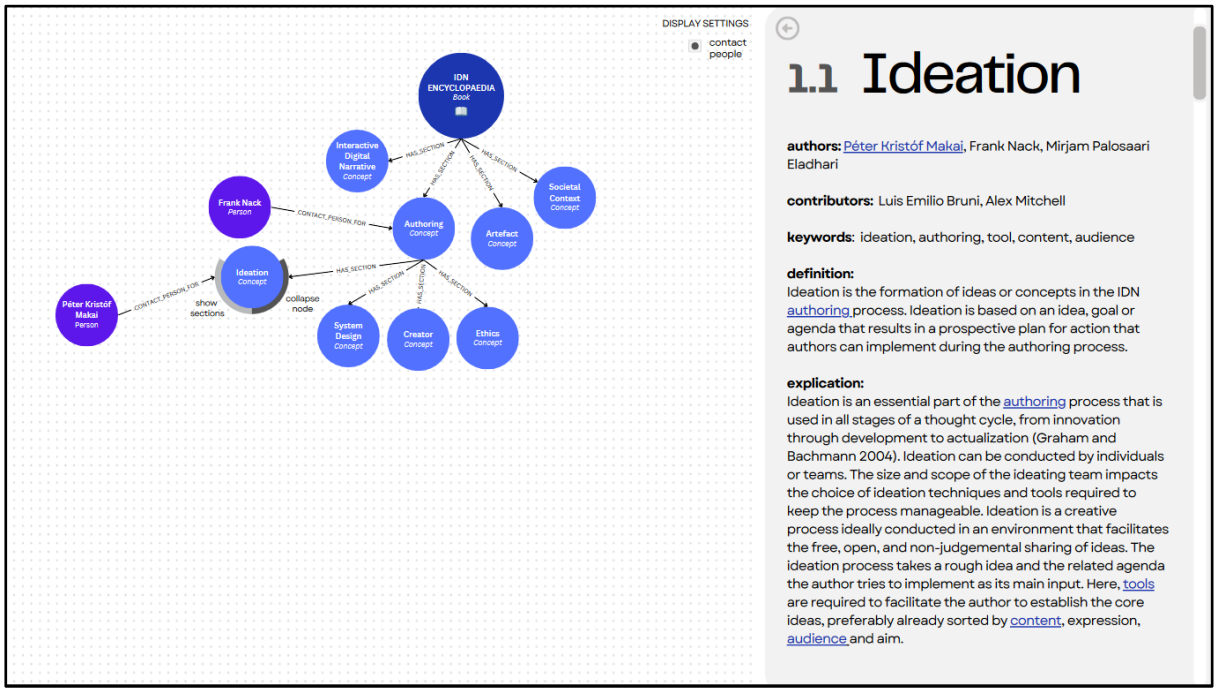






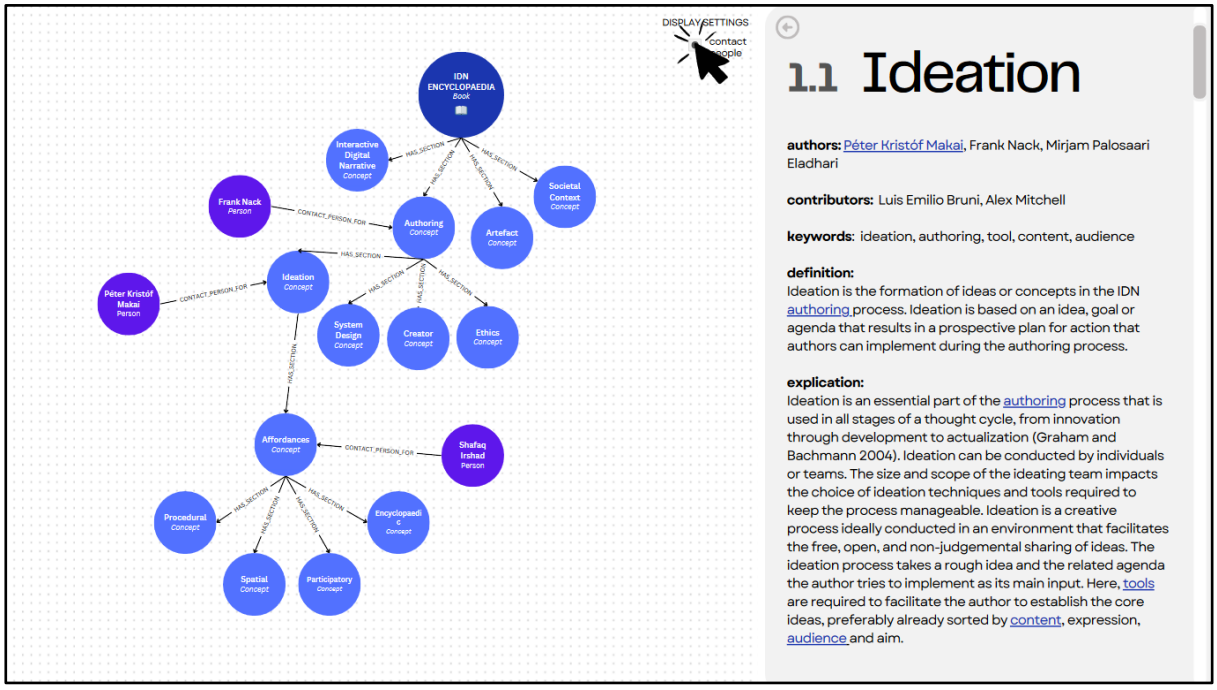
After clicking on any node, the user will get an option to either expand (by clicking on *show sections*) or collapse (by choosing the option of *collapse node*). When choosing *show sections* the chosen node will burst into other nodes (that are either other concept nodes, that are subsections ("children") of the node and/or Person nodes (depending on the chosen settings) of the contact people for that node). Choosing a *collapse node* option will lead to retraction of all the expansions in the given branch, up until the chosen node.



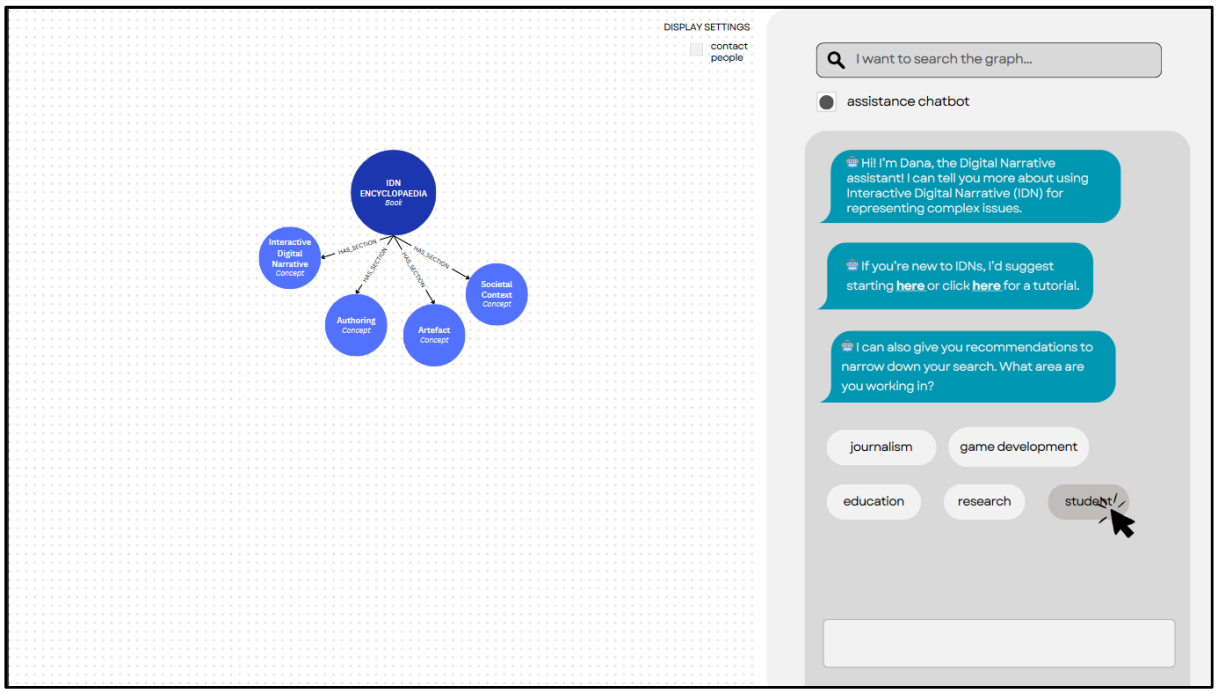


Clicking on each node also changes the slide-on sidebar menu that will display the information about the node from the database. In case of IDN Encyclopedia nodes, named Concepts, those are: concept's taxonomy number, concept's name, authors, contributors, keywords, its definition, explication, history and in some cases custom sections of content. If the sections also include allusions to other nodes (e.g. "Ideation is an essential part of the authoring process"), then clicking on them would transport the user to that node.

Clicking on the option *contact authors* in Display Settings, will also show the contact person for each node, when that node is expanded. It is turned off by default. Clicking on the Person node will show the person's name, contact and academic affiliations as well as a list of nodes they are a contact person for.



If the user chooses the chatbot option, they will be connected to the assistant right away. The assistant will introduce themselves (current working name is Dana - Digital Narrative Assistant), and then offer the tutorial on how to use the application or navigate the user to FAQ (which would include questions like: What is an IDN? And links to the respective nodes), if the user selects *starting* ***here***. If the user does not want to choose either, the assistant further asks about the user's area of interest (for example journalism, game development or education), which will then allow it to ask further questions. For example, offer the nodes that are concerned with that topic specifically, as indicated by keywords, or offer the user types of sources/programs, more suitable for their field.



3.1 Future work

3.1.1 Chatbot

In the case of wanting to improve the chatbot in the future, integrating a Large Language Model adds an option to generate answers for the user from the information in the database and summarize some texts in the database. The information from the database can be retrieved in several manners. One way is directly through the LLM. The LLM is able to generate a Cypher query which then pulls the necessary information from the database. This information is then used as context by the LLM to generate an answer.

Another way is to use semantic embeddings. If all nodes and relationships are assigned an embedding, a semantic search can be utilized using the user query, to find the most similar node or relationship using cosine similarity. The following process is to retrieve the information from the most similar node or relationship and let the LLM generate the answer using the information as context. It is necessary to note that the only difference between these two approaches is the way the information is retrieved. The decision between these approaches would depend mainly in the trust put into the LLM. This will depend on the model's ability to generate correct answers and queries which will be determined by the fine-tuning process and its prompting.

3.1.2 Semantic search

Suggest how a possible semantic search through the graph should work (list the options Neo4j offers).

Semantic search is a search that utilizes semantic embeddings into which texts and words are converted. Semantic search is based on calculating the cosine similarity between selected embeddings and returns the embedding that is the most similar to the searched embedding. The advantage of semantic embeddings is that they can express not only one word, but also longer texts.

In the case of the assistant, the process would go as follows. Assuming that the database, its nodes, and relationships have already been assigned their embeddings, the user's question is processed, and its own embedding is calculated. The question's embedding is then compared to all the other embeddings from the database using cosine similarity and the most similar embedding is returned. From the returned embedding we can access the specific node itself and then return it to the user.

For creating the embeddings (for user questions and database), it is sometimes advised to process the text in a way that gets rid of stop-words (words that do not carry significant meaning such as articles). Stop-words tend to centralize embeddings which is undesirable because the meaning in embeddings calculated from longer texts may lose its significance in the embedding if there are many stop-words in the text.

In case of using semantic embeddings in other databases, we recommend using the same process and vectors to create the embeddings of the database. This will make it easier to connect them if it is necessary in the future.