The Itsy Bitsy Spider

Progress Report for Spider February 2023

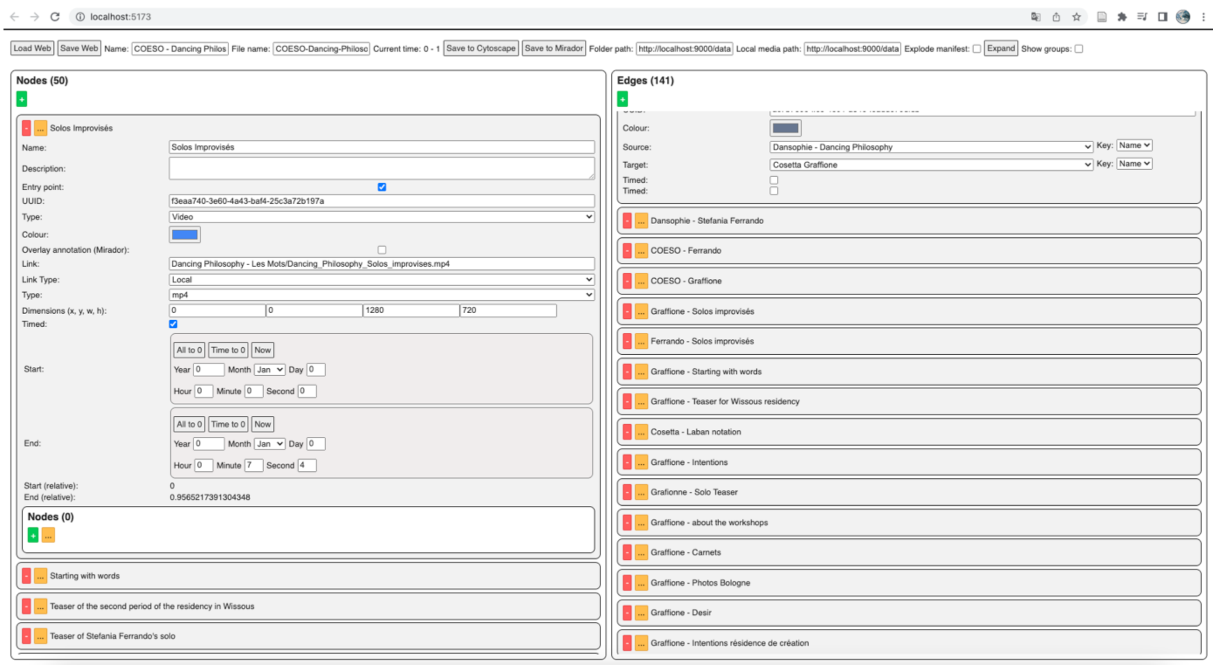
Spider refers to my system/structure/interface for creating and navigating **network structures with temporal dimensions**. Spider is a reference to **Ingold's** introduction to some of his ideas *When ANT meets SPIDER*, and is the materialisation of some of my ideas on social/network analysis that do and don't subscribe to his perspective. Spider is being developed in parallel with a new version of **MemoRekall**. They are two different entities, but greatly inform each other. In relation to the development of MemoRekall, Spider notably allows for two things:

- The fast development of **test content** and case studies (notably the **COESO Project case study**).
- **Prototyping of functionality** to be added to MemoRekall in the future.

In this progress report I shall describe development activity that has occurred and what the future holds.

spider-svelte

I created a first version of Spider in a web framework called **svelte**. Svelte notably allows for quick prototyping of **interface ideas**, and quick development of **data-driven interfaces**. As a web interface, it is fairly open, being immediately available to anyone with a web browser. For working, I run a **local web server**. The main drawback of svelte is naturally the **difficulty for accessing content on your local machine** (as is the case with anything happening in the web). This was one of the reasons I moved away from this interface later on (see below).



Name: COESO - Dancing Philos   File name: COESO-Dancing-Philoso   Current time: 0 - 1   Folder path: http://localhost:9000/data   Local media path: http://localhost:9000/data   Explode manifest: ☐   Show groups: ☐

Nodes (50)

Solos Improvisés

Name:	Solos Improvisés
Description:	
Entry point:	✓
UUID:	f3eaa740-3e60-4a43-baf4-25c3a72b197a
Type:	Video
Colour:	🟦
Overlay annotation (Mirador):	☐
Link:	Dancing Philosophy - Les Mots/Dancing_Philosophy_Solos_improvises.mp4
Link Type:	Local
Type:	mp4
Dimensions (x, y, w, h):	0 0 1280 720
Timed:	✓
Start:	Year 0 Month Jan Day 0 Hour 0 Minute 0 Second 0
End:	Year 0 Month Jan Day 0 Hour 0 Minute 7 Second 4
Start (relative):	0
End (relative):	0.9565217391304348

Nodes (0)

Starting with words

Teaser of the second period of the residency in Wissous

Teaser of Stefania Ferrando's solo

Edges (141)

Colour:	▨
Source:	Dansophie - Dancing Philosophy Key: Name
Target:	Cosetta Graffione Key: Name
Timed:	☐
Timed:	☐

Dansophie - Stefania Ferrando

COESO - Ferrando

COESO - Graffione

Graffione - Solos improvisés

Ferrando - Solos improvisés

Graffione - Starting with words

Graffione - Teaser for Wissous residency

Cosetta - Laban notation

Graffione - Intentions

Grafionne - Solo Teaser

Graffione - about the workshops

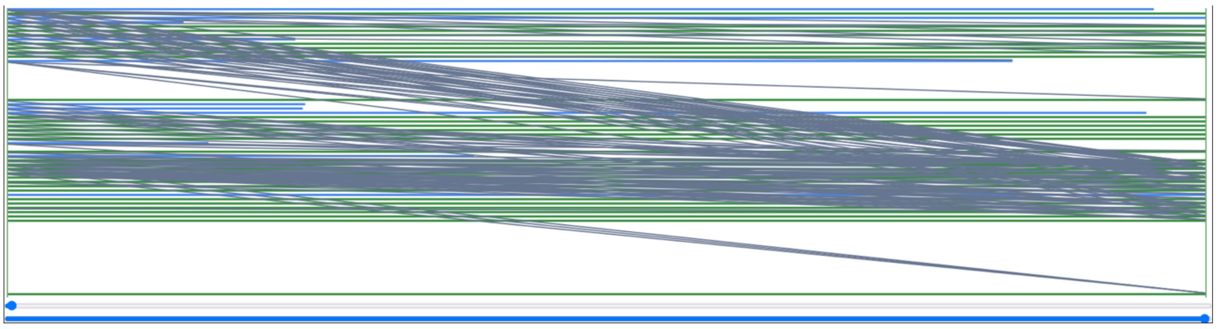
Graffione - Carnets

Graffione - Photos Bologne

Graffione - Desir

Graffione - Intentions résidence de création

In this interface, you build up a collection of two primary lists: **nodes** and **edges**. Nodes can also have any number of **nested nodes**. Each node can notably have **timed data** attached to it. The idea is that the user builds up a **simple and agnostic network structure** which is then easy to **convert** into other formats. Here, we can convert the structure either to a **Cytoscape** file or a collection of **IIIF manifests**, designed to be read in MemoRekall's fork of **Mirador**. There is also a **graphical rendering** of the current data, but as you can see, it's about as useful as a chocolate teapot.



This first prototype was great for **fleshing out ideas**, and a relatively complete version of the COESO case study was made using this interface. It was notably useful to help conceive of how a network structure should **translate** to an interface like Mirador, and into the **inter-/intra-documentary annotation methodology** that has emerged from MemoRekall (more on that below). However, there were a few **limits** that needed to be addressed:

- Not having anticipated the complexity of the code, it has grown into a **complex bowl of spaghetti**. Clearly, an **OOP-based approach** would make things a lot simpler/robust.
- For similar reasons, even with a relatively small amount of data (approximately 50 nodes and 150 edges), the interface was getting **slower and slower**.
- The data fields were not up to **Dublin Core Standards**. Also, the data fields that were needed were not clear from the beginning, and so were wonkily implemented in this system (I was able to learn from this early prototype going on).
- A **git-inspired** version control system for network creation has emerged as an important aspect of the methodology. This would be difficult to implement with this system (notably because everything is all in one-big-file).

spider-python

So, I started again. I was very quickly able to build up a system that addresses all of these issues and more as a **python package**. In order to best explain the system, I shall go through its various features.

`spider.addWeb()`

Everything in Spider happens within a **Web()**. The first thing you do is create or load the web you're working on. Under the hood, this simply refers to a **folder on your computer**, and Spider will perform operations within this folder. This is to accommodate for **a git-inspired version control** system of a web**(/capsule/collection/archive/project)**: the folder can be treated as any other git repo (on the to do list: automatize setting up the git repo).

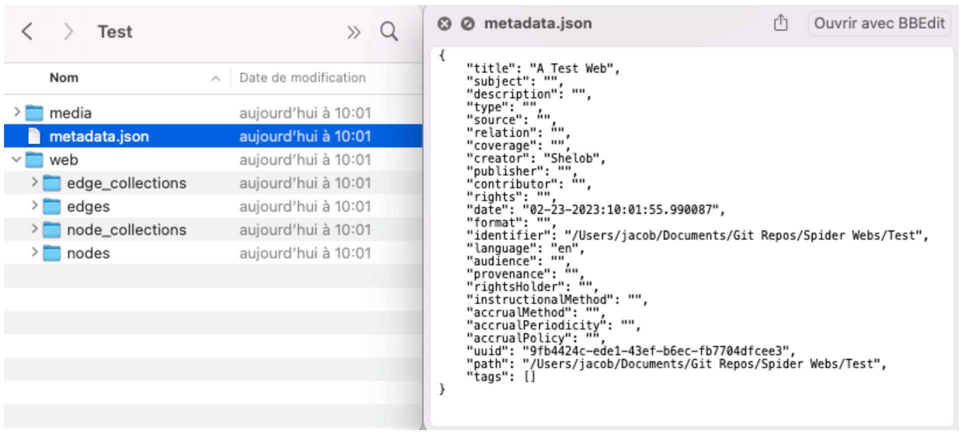
Inside this folder, there are **4 main items**:

- **Nodes**: a folder of nodes. Think of nodes as **objective items in an archive**.
- **Edges**: a folder of edges. Think of edges as the **subjective relationships we conceive of between these items.**
- **Node collections**: a folder of node collections. A collection of nodes is simply a list of nodes, nothing more, nothing less… (see below).
- **Edge collections**: same.

Every element in Spider (be it a node, an edge, a node collection, an edge collection, or the web itself) has the same number of **metadata fields** which are based on **Dublin Core**. These fields are either pure Dublin Core (like title, description etc.), or **modified** Dublin Core – a field that gets parsed into a string using a **W3C**-inspired system that's easily decodable but contains lots more information (for example the Dublin Core *relation* field will contain all the information about an edge between two nodes, the *format* field will contain all the information Mirador needs to access a media file).

Example of creating a web and the folder that gets created on your computer:

```python
# Import the spider package:
import spider as sp

# 1. Set metadata for the web:
metadata = {
    # Path on the computer where the web will be created:
    "path" : "/Path/to/my/web",

    # You can add any of the other
    # Dublin Core-derived metadata fields listed in the docs...
    "title" : "A Test Web",
    "creator" : "Shelob"
    #...
}

# 2. Create the web (provide the metadata object)
myWeb = sp.createWeb(metadata)

# 3. You can also load a web that already exists:
loadedWeb = sp.loadWeb("/Path/to/another/web")
```



```
media               aujourd'hui à 10:01
metadata.json       aujourd'hui à 10:01
web                 aujourd'hui à 10:01
  edge_collections  aujourd'hui à 10:01
  edges             aujourd'hui à 10:01
  node_collections  aujourd'hui à 10:01
  nodes             aujourd'hui à 10:01
```

```json
{
    "title": "A Test Web",
    "subject": "",
    "description": "",
    "type": "",
    "source": "",
    "relation": "",
    "coverage": "",
    "creator": "Shelob",
    "publisher": "",
    "contributor": "",
    "rights": "",
    "date": "02-23-2023:10:01:55.990087",
    "format": "",
    "identifier": "/Users/jacob/Documents/Git Repos/Spider Webs/Test",
    "language": "en",
    "audience": "",
    "provenance": "",
    "rightsHolder": "",
    "instructionalMethod": "",
    "accrualMethod": "",
    "accrualPeriodicity": "",
    "accrualPolicy": "",
    "uuid": "9fb4424c-ede1-43ef-b6ec-fb7704dfcee3",
    "path": "/Users/jacob/Documents/Git Repos/Spider Webs/Test",
    "tags": []
}
```

`web.addNode()`

Working in Python, it's very easy to **automate** things. Nodes can be added manually, but there is also functionality built in to Spider to make things easier. For example, you can use the **mediaToNode()** function to quickly add nodes to your web from a folder of media content. I also made some scripts that converted the COESO test content made with the old version of Spider into the new version. This in itself was a good case study for imagining how one could take data from a pre-existing system and feed it into Spider (imagine a **pre-existing archive database** which is converted into Spider format). Spider is agnostic and simple enough to make all of this possible.

Example of adding a node:

```python
import spider as sp
from datetime import datetime

myWeb = sp.loadWeb("/Path/to/a/web")

# 1. Set the node metadata:
nodeMetadata = {
    # You can set multiple languages:
    "title" : {
        "en" : "My amazing node",
        "fr" : "Mon noeud incroyable"
    },

    # This is a modified Dublin Core field
    "coverage" : {
        "startDateTime" : datetime.now(),
        "endDateTime" : datetime.now(),
        "region" : "Bretagne, FR"
    }
}

# 2. Add the node to the web:
myWeb.addNode(nodeMetadata)
```

Example of converting media files:

```python
import spider as sp
import utils

myWeb = sp.loadWeb("/Path/to/a/web")

# 1. Get a list of media file files to convert:
fileList = utils.collectFiles("/Path/to/media/files")

# 2. Convert files with the mediaToNode(path, copyMedia) function:
for item in fileList:
    myWeb.mediaToNode(item, True)
```

`web.addEdge()`

You can add edges to the web in the same way as nodes. As stated above, **every element can be given any of the metadata fields**, however, they may be **interpreted differently** according to the type of element. In the case of an edge, it may not make sense to add a *format* field which refers to a media file to be decoded, however it will be necessary to ass a *relation* field to describe the edge.

Example of making some edges:

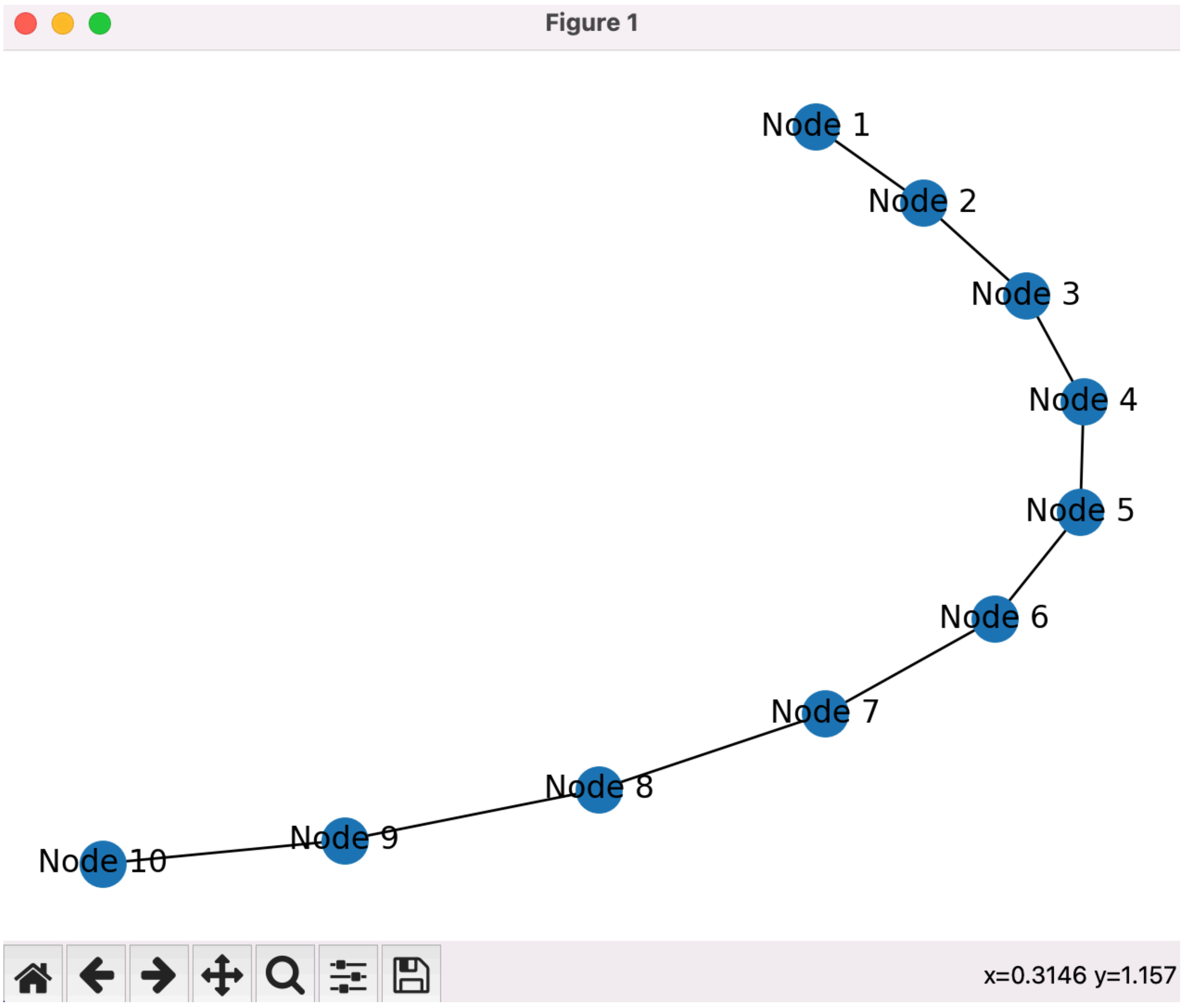
```python
import spider as sp

myWeb = sp.loadWeb("/Path/to/a/web")

# 1. Create some nodes:
nodeList = []
for i in range(10):
    nodeList.append(
        myWeb.addNode({"title" : "Node {}".format(str(i + 1))})
    )

# 2. Create an edge between each node:
for i in range(9):
    myWeb.addEdge({
        "title" : "Edge {}".format(str(i + 1)),

        # A modified Dublin Core field:
        "relation" : {
            # We have to give the UUID (unique ID) of the node
            # as a string:
            "source" : str(nodeList[i].uuid),
            "target" : str(nodeList[i + 1].uuid)
        }
    })
```

`web.convert()`

Again, this is a simple, malleable structure which can then easily be converted into other formats. Notably, **networkx** and a collection of **IIIF manifests**. There are functions built in that allow you to convert easily.

Example of converting to **networkx** and the image that is displayed:

```python
import spider as sp

myWeb = sp.loadWeb("/Path/to/a/web")

# 1. Convert the web to networkx format:
network = myWeb.convertToNetwork()

# 2. View the network:
network.display()
```

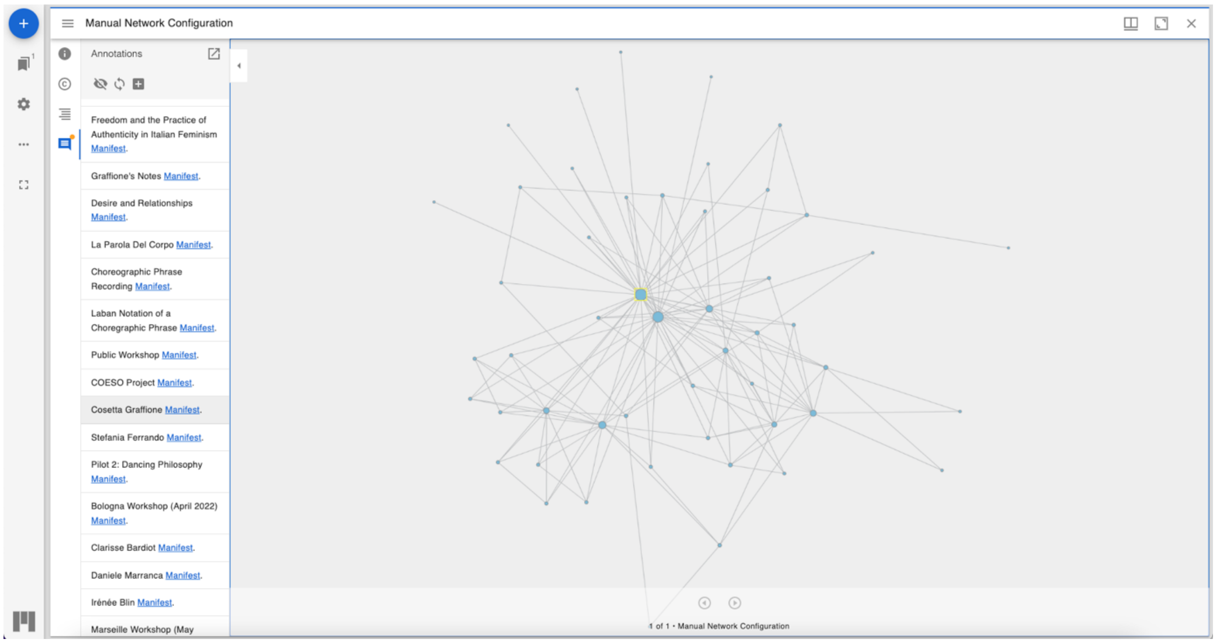


Converting to a **MemoRekall IIIF manifest network** works in the same way, you just need to give a few more fields:

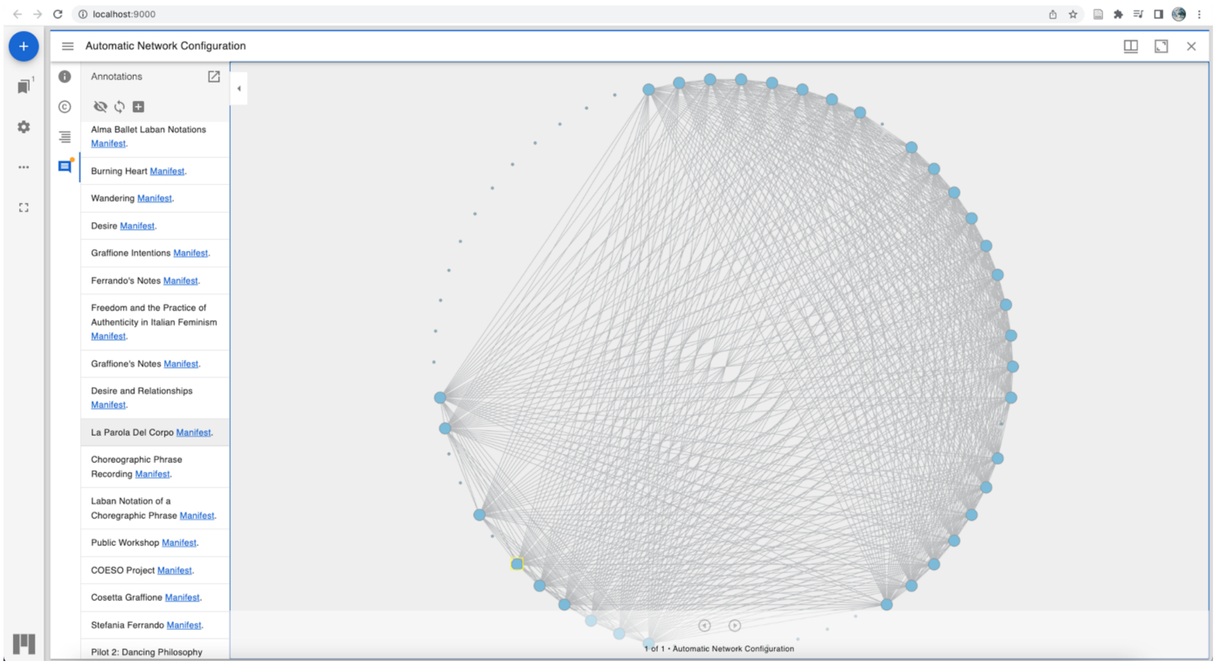
```python
import spider as sp

myWeb = sp.loadWeb("/Path/to/a/web")

# 1. Define your settings:
settings = {
    # The real path to the manifests:
    "path" : "http://localhost:9000/data/Web-Manifest-Network",

    # The place the MemoRekall network will be written to
    # (None = webPath/mirador):
    "writePath" : "/Path/to/local/folder",
}

# 2. Convert to MemoRekall:
myWeb.convertToMemoRekall(**settings)
```

`web.addCollection()`

You can also add **collections** of nodes and edges. This means that you can **configure that web differently** without having to store huge amounts of duplicated data. Taking the COESO project as an example, see the images below of manifests that were created using **two different collection configurations**. In this first example, the edges were created manually by myself:


Manual Network Configuration
Annotations
Freedom and the Practice of Authenticity in Italian Feminism Manifest.
Graffione's Notes Manifest.
Desire and Relationships Manifest.
La Parola Del Corpo Manifest.
Choreographic Phrase Recording Manifest.
Laban Notation of a Choregraphic Phrase Manifest.
Public Workshop Manifest.
COESO Project Manifest.
Cosetta Graffione Manifest.
Stefania Ferrando Manifest.
Pilot 2: Dancing Philosophy Manifest.
Bologna Workshop (April 2022) Manifest.
Clarisse Bardiot Manifest.
Daniele Marranca Manifest.
Irénée Blin Manifest.
Marseille Workshop (May
1 of 1 • Manual Network Configuration

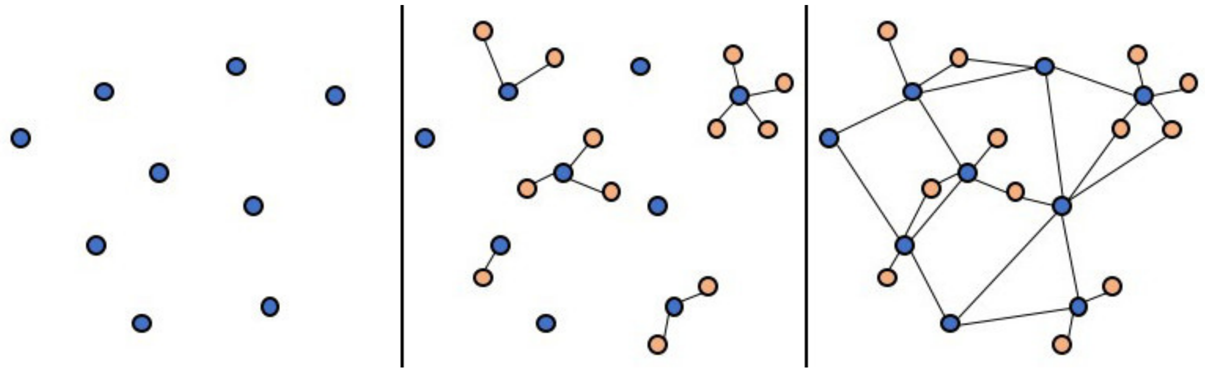
In the next example, we refer to the **same nodes**, but a **new collection of edges** was made using a simple algorithm (the algorithm looked at each node's title and description for a collection of search terms and connected nodes where the term was present).


localhost:9000
Automatic Network Configuration
Annotations
Alma Ballet Laban Notations Manifest.
Burning Heart Manifest.
Wandering Manifest.
Desire Manifest.
Graffione Intentions Manifest.
Ferrando's Notes Manifest.
Freedom and the Practice of Authenticity in Italian Feminism Manifest.
Graffione's Notes Manifest.
Desire and Relationships Manifest.
La Parola Del Corpo Manifest.
Choreographic Phrase Recording Manifest.
Laban Notation of a Choregraphic Phrase Manifest.
Public Workshop Manifest.
COESO Project Manifest.
Cosetta Graffione Manifest.
Stefania Ferrando Manifest.
Pilot 2: Dancing Philosophy
1 of 1 • Automatic Network Configuration

This gives us a **dynamic** way of **pivoting** between not only different nodes and resources, but different configurations and semantic organisations of nodes.

Methodology

The development of these interfaces, based on a concrete case study allowed a **methodology** for working in this context to emerge. It grew also from pre-existing work around MemoRekall, notably with the concept of **inter**- and **intra-documentary annotations**. Future development of both Spider and MemoRekall shall look to accommodate and develop the following ideas.



The idea is to begin with a group of **disparate resources**. In Spider, these can be thought of as the lists of **nodes**, in MemoRekall, the main element of a manifest without any annotations. These can also be thought of as items in an archive, files on a computer etc. A first **semantic operation** can be performed by making different groupings of these elements – in Spider these are conceived of as **node collections**. With the git version control model, these groupings can be subject to collaborative and cumulative growth.

Next, we can bring elements to explain and complete these items. This is a profoundly **analytical act**: not only will these subsequent operations and configurations change according to the different people making these configurations – they will also change depending on the narrative that this person is attempting to unravel from the collection. This is again where the version control approach is profoundly important. In Spider, these **intra-documentary annotations** materialize (for the moment) as **nested nodes**. Being nodes, this means that they can also be linked to with an edge (in which case we theoretically consider them as a mix between an intra- and inter-documentary annotation). With these annotations, we move from a **collection of disparate items**, to a **collection of *centred* networks** (the centre being the original main element that is being annotated).

Finally, we link these elements to each other via **edges**. These are **inter-documentary annotations**. The collection now becomes a truly **heterarchical rhizome** – we can define entry points, and there is somewhat of a hierarchy between nodes and nested nodes, however there is theoretically no innate hierarchy in the structure. MemoRekall offers a dynamic way of **navigating** this rhizomatic structure – note however, that as an agnostic structure, any kind of navigation interface could be envisioned. For now, we have also only implemented **node-local perspectives** (with links to immediate neighbours) and **global network visualizations**, but with the multitudes of metadata which can be attached to each node, many more interfaces can be imagined. For example, these visualizations tend to ignore the **temporal dimension** of nodes – an aspect that I specifically wish to address going on.

The version control approach is powerful – for example we can integrate **automated configurations** into analysis and articulate it with manual work. However, it will be of upmost importance to find **solutions for navigating the multidimensional space of these different versions**. The user interface will need to follow the advance of the underlying tools.

COESO case study

To finished, I shall briefly explain how I applied these methodological ideas. The COESO case study also has the particularity of having begun as a MemoRekall **legacy capsule**. This offers us a great opportunity to **compare** the two approaches and consider their affordances and limits.

I began by constituting the collection of **disparate elements** as described in the first step of the methodology. Essentially, I took all of the elements from the legacy capsules I was 'transposing' and added them into the system. Next, I began to create **edges** between these nodes. One idea emerged here: it was necessary to add nodes that were notably **institutional** and **intangible** in nature: nodes like *The COESO Project, Cosetta Graffione* or *Workshop in Marseille*. In **my approach** to network construction, it is difficult to weave a narrative that tells the story of a project only by configuring purely tangible nodes that are attached to a specific piece of media. This meant that I had to add nodes to the collection that have a semantic origin.

After this, I then chose some nodes that I thought would benefit from being explained with **intra-documentary annotations**. Time constrains and also issues in development meant that this aspect was not developed as fully as it could. I essentially chose some videos that could be accompanied by **Laban notations** which at least helped me understand that system a bit better and also understand what the dancers were doing.

Beyond time constraints and development issues, I do think that there is something to be said for the great majority of inter-documentary annotations that is inherent in the methodology and the interface. This system allows us to **begin with a large collection of media**, whereas with MemoRekall legacy, we are focused on **only one**. This necessarily **shifts our perspective** and the way we pursue analysis. It is symptomatic that with the IIIF, Spider-driven system, a network of edges between items emerges compared to legacy where we produce largely intra-documentary annotated capsules. This is another aspect to bear in mind going forward, especially when thinking about interface.

The COESO case study is indeed not much more than a **proof of concept** – no serious analytical work took place. It does however succeed in demonstrating how this approach could help and drive analysis. It successfully gives an overview of the project, and how its various actors (human and non-human) interact, and does offer an intriguing interface to explore these resources. It has also given us much food for thought for the future.