Speaker
SÖREN KLEIN

EMBER NEXUS API
For Knowledge Graphs

29TH JAN 2024

10:00 PT | 19:00 CET

https://bit.ly/graphgeeks

ONLINE

GraphGeeks.org

Ember Nexus

Knowledge Graph API

1. The motivation behind Ember Nexus API
2. Requirements and lessons learned
3. Frontend: How would I like to work with data?
4. API: Current state with examples
5. Plans for the future

1. The motivation behind Ember Nexus API

As a child, I was always fascinated by data. I was curious about outliers that broke simplified rules, leading to a more complete understanding of our world. As I grew older, my interest in data deepened.

However, I faced a challenge when it came to storing all this data. Maintaining the schemas of databases required more and more time, especially if I wanted to push the capabilities further.

I made several attempts to build an API geared towards data storage and retrieval, but failed often and learned from my mistakes. Eventually, I stumbled upon Neo4j, and after an amazing initial experiment, I was hooked! :D

This was the start of my graph journey, and half a year later, I began developing another data API, which eventually became Ember Nexus API.

2. Requirements and lessons learned

Simplicity is important.

Limited features can still result in
complex results, e.g. Conway's Game of Life.

Standards are great.

Standards are great.

Not every standard works for everybody;
being able to change defaults is important.

How do I want to implements these requirements and lessons?

Atomic data.

The API should not enforce how I store data within the smallest indivisible element.

Normalization is a powerful tool, but requiring to normalize everything is a burden.

Global identifier.

Every element should be accessible in a single step.

Access should not require knowing the element's type, just its identifier. The identifier is basically globally unique.

Open types.

User can define types themself.
Elements can be of any type.

Visualizations require context.

Elements should be displayed and visualized in different ways, depending on their type and context.

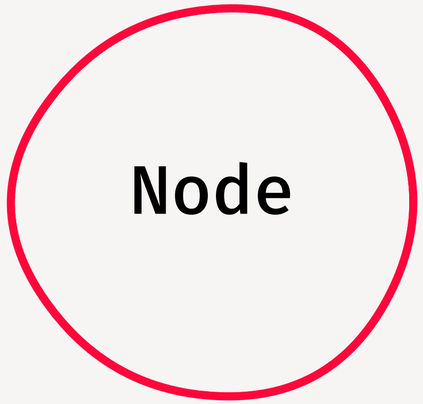
Great defaults, easy modification.

The default visualizations should be easily useable and understandable by the average user.

Power users should be able to modify, replace, and add their own visualizations.

3. Frontend: How would I like to work with data?

Components can visualize individual nodes and edges.



Name of the element

`Type`

Description Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam...

Name of the element

`StartNodeType -- RELATION_TYPE → EndNodeType`

Components are available in different shapes and sizes to adapt to different environments & contexts.

Icon

A B C 

Inline Text

Name of the user

Inline Pill

Name of the user

Thumbnail

Card

Name of the user

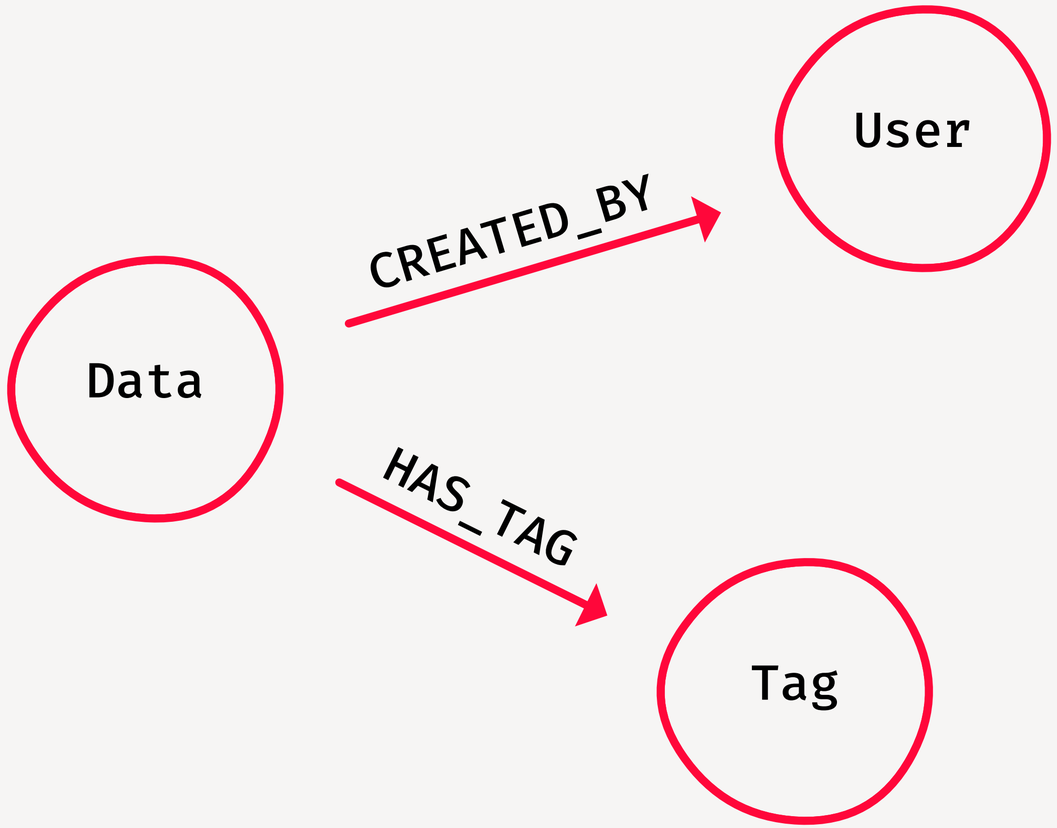
User

Frameless

Name of the user

User

Components are composable, i.e. they can contain other components to display related data.



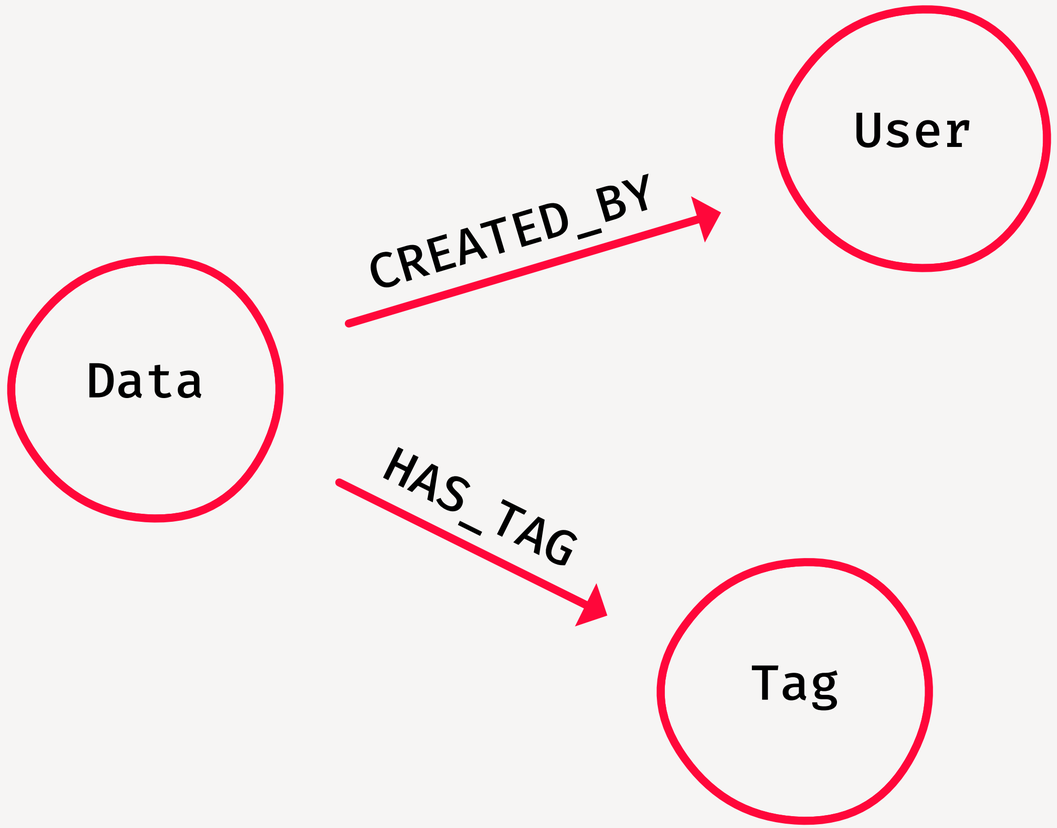
Name of the element

Type

some blue tag

Description Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam...

Components can try to render more data if available.



Name of the element

Type

Description Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam...

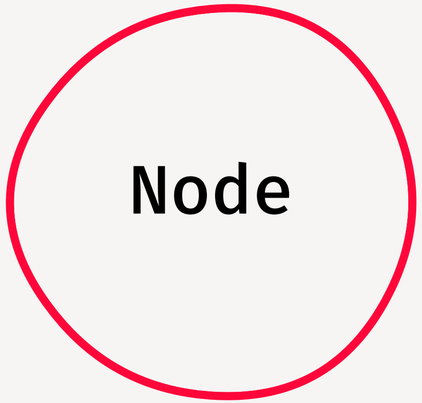
Name of the element

Type

some blue tag

Description Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam...

In the absence of specialized components, elements will be visualized by fallback components.



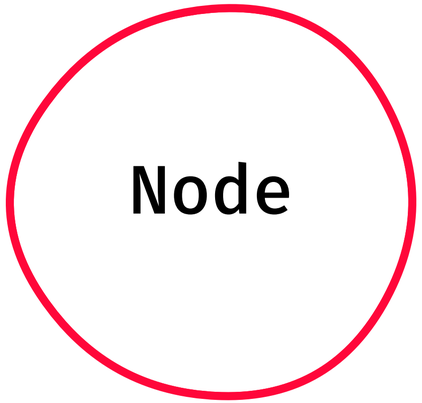
Name of the element

Type

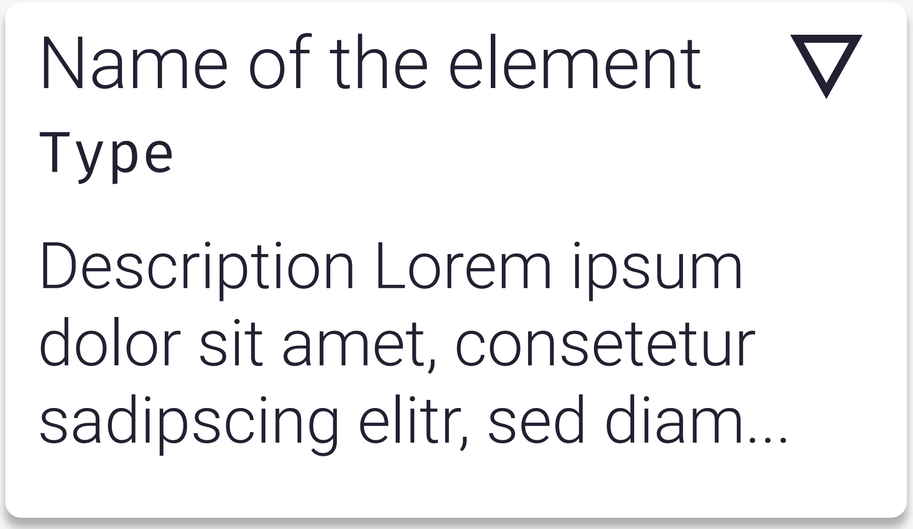
Description Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam...

Now that we can display data as components, what can we do with them?

Single data node:



Card visualization:

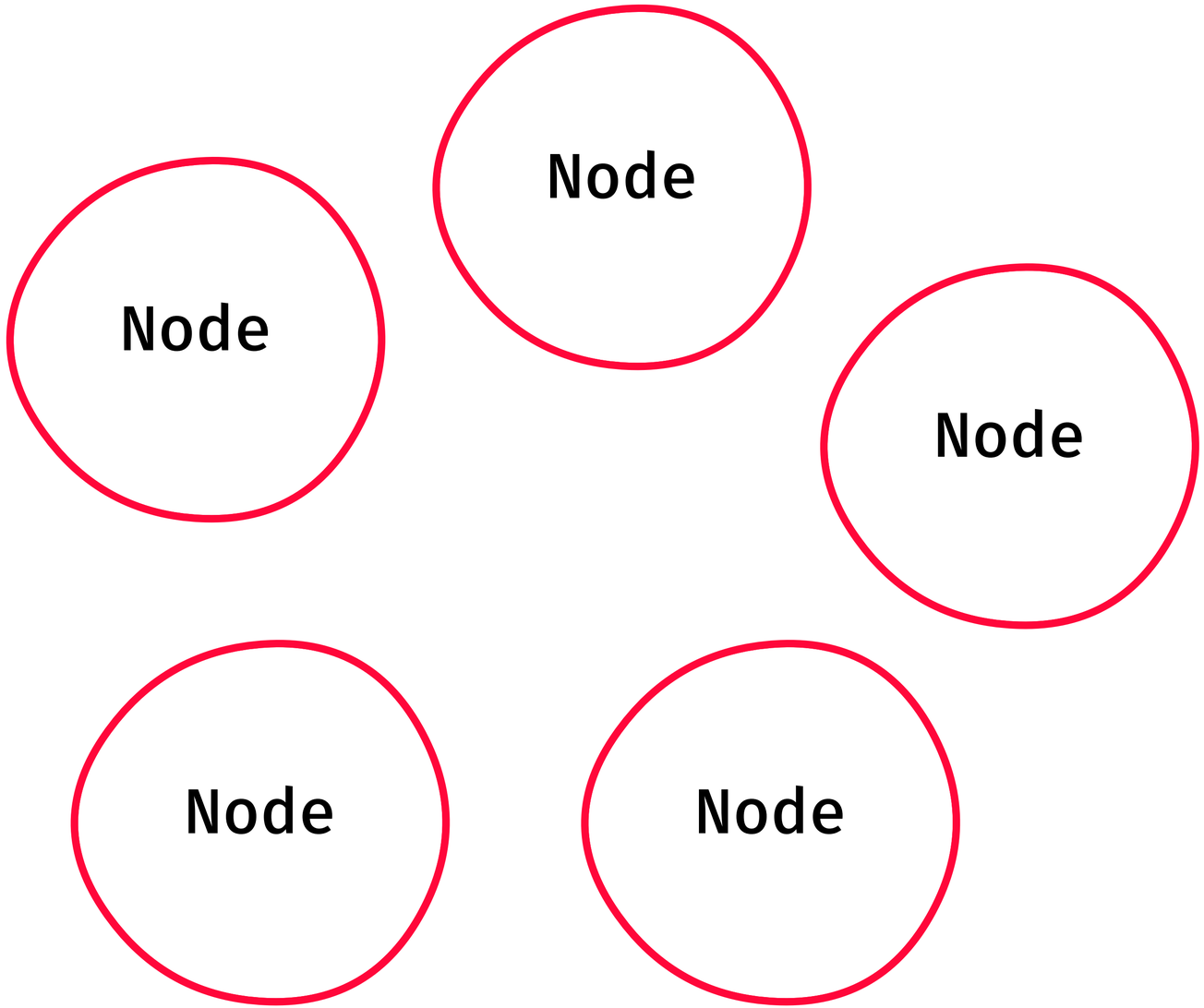


Name of the user

`User`

Full screen frame:

Collection of data nodes:



List:

Name of the element

Type

Name of the element

Type

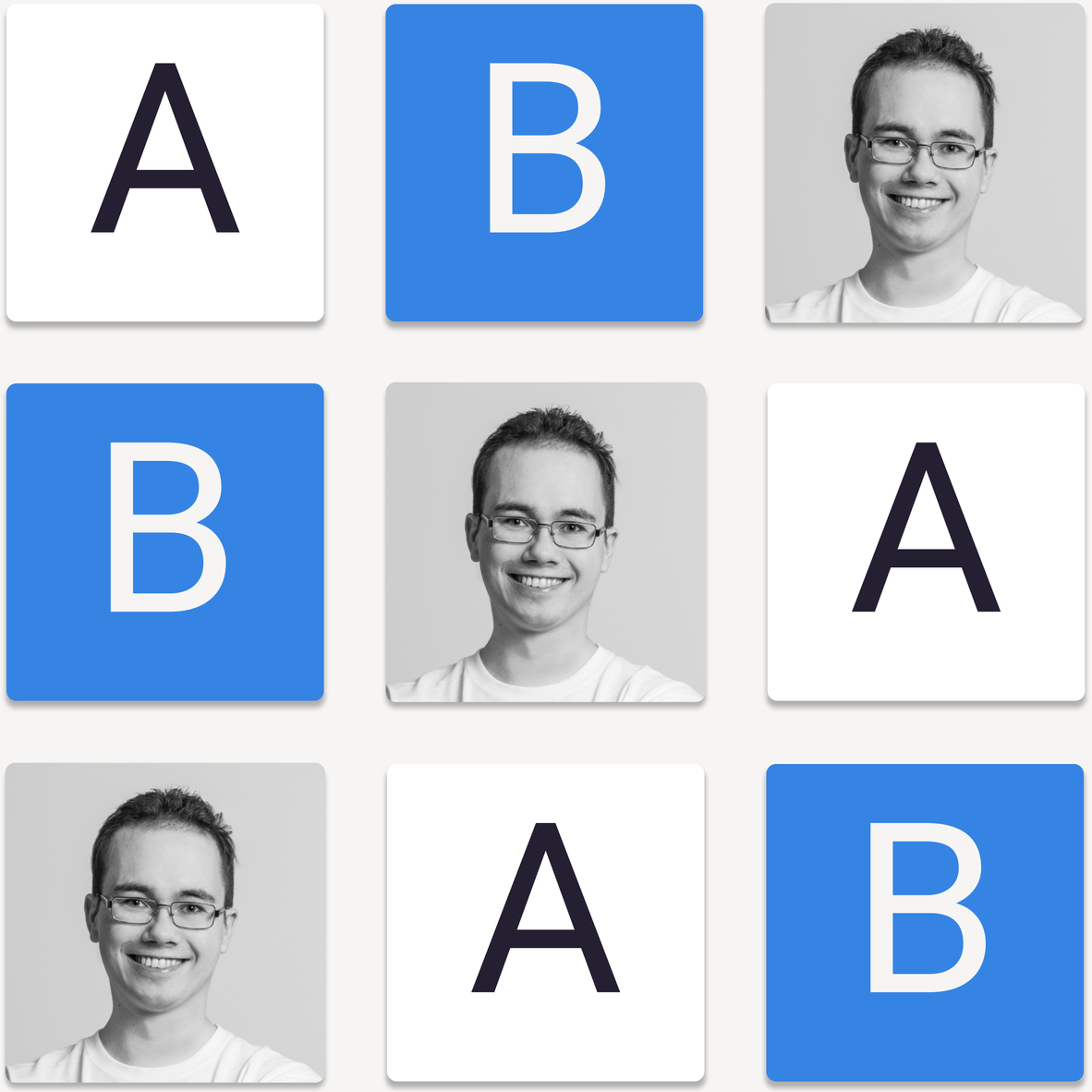
Name of the element

Type

Name of the element

Type

Grid:



4. API: Current state with examples