Evaluation test: Build a simple data visualization dashboard

By: Darius Sas (darius.sas@arcan.tech)

Description

The goal of this test is to build a small data visualization dashboard and visualize an interactive **dependency graph**. The dashboard must retrieve the data from a remote endpoint using GraphQL, visualize it to the user, and provide a few basic controls to allow the user to inspect and navigate the graph.

In particular, the controls should allow the user to filter/show only edges and nodes with a specific type of **label**. Additionally, the user should be able to pick different **layouts** for the graph to be displayed.

The dashboard must be written in TypeScript using React. To draw and layout the graph, use Cytoscape.js. In the section below, you can find the details on how to fetch the data.

The **duration of the test is 3-4 hours** (hackathon style). You are evaluated based on the quality and skill to understand the data and how you plot it. Note that a bare plot that does its job is desirable; don't spend time on making it beautiful, but rather on making it useful to the user.

The output of the project should be a public GitHub repository containing all the code and instructions on how to run it. Please add a Dockerfile (or docker compose) to run the app.

The dependency graph

A dependency graph is a graph representing source code constructs and the relationships between them as nodes and edges respectively. For example, if we have two classes A and B, with A declaring a method with a parameter of type B, then we say that A **depends on** B. Thus, the corresponding dependency graph will look like this: A is a node, B is node, and a directed edge connects A to B.

In a graph, one can distinguish between different types of nodes/edges using the **label** property.

There are two different types of source code constructs (nodes) represented in the dependency graph:

- Containers, such as packages or folders (label 'container')
- Units, such as classes and interfaces (label 'unit')

Typically, when visualized, nodes with different labels are mapped to different colors.

There are different types of relationships (edges) represented in a dependency graph:

- Dependency relationship, as in the previous example (label ‘dependsOn’, ‘containerIsAfferentOf’)
- Membership relationship, as in a Class/File belonging to a Package/Folder (label ‘belongsTo’)
- Hierarchy relationship, as in a class inheriting from a superclass

Some types of edges have a ‘Weight’ property, which indicates the strength of the relationship. This is typically mapped to the width of the edge.

Fetching the data

To fetch the data, you can use the following information.

Endpoint: https://api-demo.arcan.tech/graphql

GraphQL Variables:

```
{
    "projectId": 287,
    "versionId": "2e718ebd3f968a675dfbc36bb4a126e13186eddf"
}
```

GraphQL Query:

```
query getdependencyGraph($projectId: Int!, $versionId: String!) {
  projectById(projectId: $projectId) {
    dependencyGraph(versionId: $versionId) {
      allUnits {
        id
        label
        name
        simpleName
        relativeFilePath
        properties {
          key
          value
          __typename
        }
        constructType {
```

```
    name
    prettyName
    __typename
  }
  __typename
}
allContainers {
  id
  label
  name
  simpleName
  relativeFilePath
  constructType {
    name
    prettyName
    __typename
  }
  properties {
    key
    value
    __typename
  }
  __typename
}
membershipEdges {
  id
  label
  member {
    id
    __typename
```

```
  }
  parent {
    id
    __typename
  }
  __typename
}
dependencyEdges {
  id
  label
  weight
  dependedUpon {
    id
    __typename
  }
  dependant {
    id
    __typename
  }
  __typename
}
hierarchyEdges {
  id
  label
  parent {
    id
    __typename
  }
  children {
    id
```

```
          __typename
        }
        __typename
      }
      __typename
    }
    __typename
  }
}
```