**Neo4j and GraphQL Implementation**

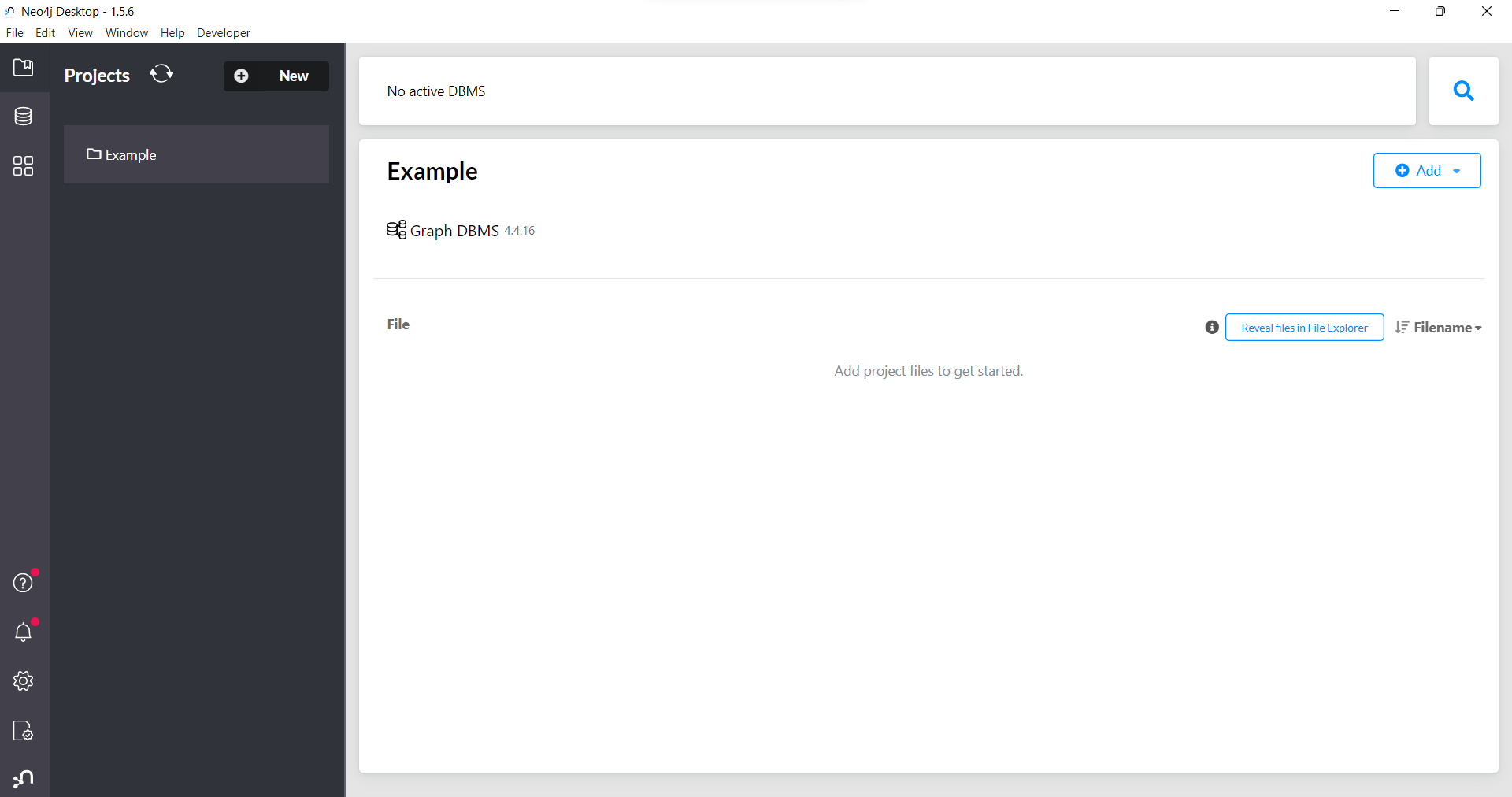
**What is Neo4j?**

Neo4j is a popular graph database management system designed to handle and process large-scale graph data. Unlike traditional relational databases that store data in tables, Neo4j stores data in a graph format, which is especially useful for representing and managing complex relationships between entities.

In a graph database, data is stored as nodes, which represent entities, and relationships, which describe connections between the nodes. Each node can have properties that contain data about the entity it represents, while each relationship can also have properties to describe attributes of the connection. This representation allows for efficient and flexible querying of data with complex relationships.

**Setup Neo4j in Desktop (**[**https://neo4j.com/docs/operations-manual/current/installation/**](https://neo4j.com/docs/operations-manual/current/installation/)**)**

**Upon successful installation we will be able to reach this UI.**



**Steps to create a new DBMS:**

1. Create a new project.

A screenshot of a computer

Description automatically generated

1. Select Add Local DMBS.

A screenshot of a computer

Description automatically generated

1. Create password and select version 4.4.x.
2. Press “Create”.

**There are two options to load data into our database.**

**1. By loading a dump file**

* + 1. Select “Add File” option.

A screenshot of a computer

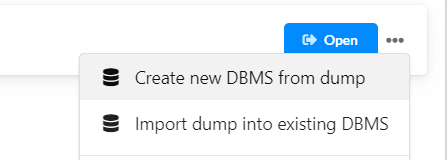
Description automatically generated

* + 1. Select *“sample.dump”* that was downloaded.

A screenshot of a computer

Description automatically generated

* + 1. Sample.dump will be shown under the file section.
    2. Select “Import dump into existing DBMS”



* + 1. Select the existing DBMS and create a new name for the database (e.g. sample)

A screenshot of a computer

Description automatically generated

* + 1. Verify the nodes were created from the left panel in Neo4j Browser.

**2. Running the codes via Neo4j Browser**

* + 1. Start Graph DBMS.
    2. Select import and copy “ship\_plan.csv” into the directory.

A screenshot of a computer

Description automatically generated

* + 1. Open Neo4j Browser.

A white background with a black and white object

Description automatically generated with medium confidence

* + 1. Copy and paste all codes from “upload.cypher” into the cell and run.

A screenshot of a computer

Description automatically generated

* + 1. Confirm nodes and properties were created from the left panel.

A screenshot of a computer

Description automatically generated

**Querying Data using Cypher**

Example:

* Query to obtain voyages related to “XYZ Express”
* *“MATCH (vo:Voyage) WHERE vo.vessel = “XYZ Exp” RETURN vo*”

**A screenshot of a computer screen

Description automatically generated**

**Obtain Schema of Database**

1. Access Neo4j Developer [Graph Apps Gallery](https://install.graphapp.io/).
2. Select GraphQL Toolbox and install.

A screenshot of a phone

Description automatically generated

1. Upon installation, access it from Neo4j Desktop.

A screenshot of a login box

Description automatically generated

1. Select Introspect.

A screenshot of a computer

Description automatically generated

1. Copy and store the schema in a .txt file for future usage.

**What is GraphQL?**

[GraphQL](https://graphql.org/) is a query language and runtime that allows clients to request exactly the data they need from a server, making it a powerful alternative to traditional REST APIs. With GraphQL, clients can specify the shape and structure of the data they want to retrieve, which reduces over-fetching and under-fetching of data, leading to more efficient and performant data interactions.

When using Neo4j as the data storage backend for a GraphQL API, you would typically have a schema that mirrors the data model in Neo4j. Nodes in the Neo4j graph database would map to object types in the GraphQL schema, while relationships between nodes would be represented as fields in those types.

Clients can then use GraphQL queries to request specific data, specifying the fields they want and the relationships they want to traverse. This gives clients the power to request a customized view of the data, fetching only what they need and reducing unnecessary data transfer.

**GraphQL Architecture**

GraphQL architecture consists of several components and layers that work together to enable efficient and flexible data querying. The architecture can vary depending on the implementation and specific requirements, below is the general overview of the key components:

Diagram

Description automatically generated with medium confidence

{Resolver}

{Schema}

* **Client** is any application that sends GraphQL queries to the server to request data.
* **Node.js** sets up the server that handles incoming requests from clients including GraphQL queries.
* **Schema** serves as a contract between the client and server, defining the available data and relationships between them.
* **Apollo server** is a middleware that specifically designed to work with GraphQL APIs by validating and processing the queries to provide requested data to client.
* **Resolver** is used to fetch data associated with fields from schema.
* **Neo4j** acts as a data source.

**Apollo GraphQL Setup**

Prerequisites:

1. Install Nodejs (<https://nodejs.dev/en/learn/how-to-install-nodejs/>)
2. Install Neo4j

Steps:

* 1. Initialize a node.js project.
  2. Install dependencies:
     1. [*@neo4j/graphql*](https://www.npmjs.com/package/@neo4j/graphql) *== 3.17.0*
     2. [*Graphql-request*](https://www.npmjs.com/package/graphql-request) *== 5.2.0*
     3. [*Apollo-server*](https://www.npmjs.com/package/@apollo/server) *== 3.11.1*
     4. [*Neo4j-driver*](https://www.npmjs.com/package/neo4j-driver) *== 5.6.0*
  3. Create an ‘index.js’ file in project folder and import packages.

Text

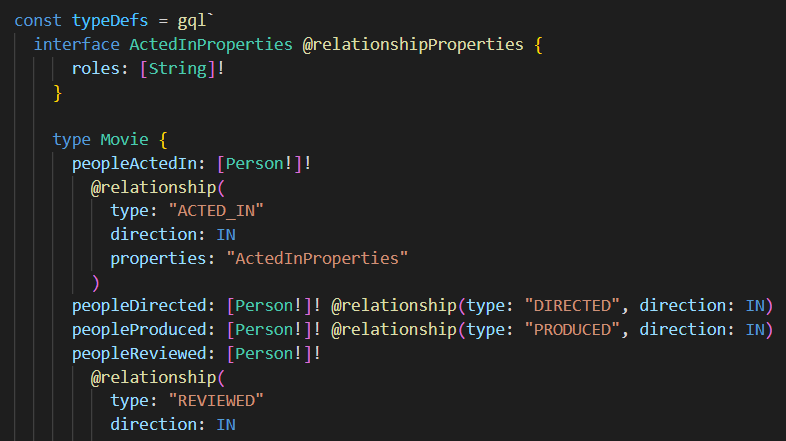
Description automatically generated

* 1. Create driver for neo4j including URL, username, and password.

Text

Description automatically generated

* 1. Copy neo4j schema that was obtained from the toolbox and store as a constant.



* 1. Combine schema and driver.



* 1. Run server.

Text

Description automatically generated



*\*Complete code (index.js) is available in the folder*

**Query using Apollo Playground**

1. Access to the apollo server (http://localhost:4000/).
2. Create a query by selecting from the provided fields.

A screenshot of a computer

Description automatically generated

1. Verify auto produced query on side screen (ensure no red curly lines in query).

Graphical user interface, application, Word

Description automatically generated

1. Run the query to obtain results in JSON format (or CSV format).

Table

Description automatically generated with low confidence

1. (Optional) Include the GQL query to client code or store as *‘\*.gql’*.

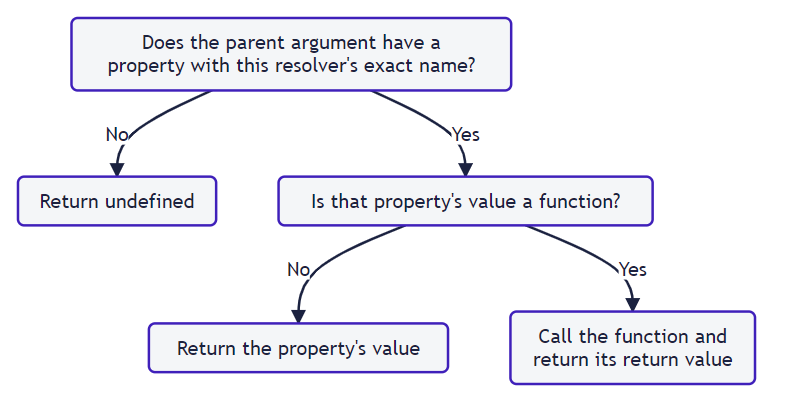
Text

Description automatically generated OR Text

Description automatically generated

**Resolvers**

As mentioned above, resolvers are functions responsible for fetching data associated with a particular field in the GraphQL schema. By default, Apollo server automatically defined a default resolver for us if not defined. The default resolver uses the logic below:



To create custom resolvers, we need to define it from the JavaScript object. This resolver is related to our schema’s type (Query) and those each resolver function belongs to a corresponding query. The function will indicate how to fetch data from the data source and return the processed output.

Below is an example of defining a distinct vessel query in GraphQL. First, we define the query name (`distinctVessel`) and the return type which is a list of non-nullable string.

A screen shot of a computer code

Description automatically generated

Then create a resolver variable to handle these defined queries as shown below.

A screen shot of a computer code

Description automatically generated

(Optional) If we are returning multiple variables, we can map them individually as well.

A computer screen shot of text

Description automatically generated

In the example below, we mapped the output from query (uniqueProducts) to multiple variables.

E.g.

* month -> month
* year -> year
* total\_unique\_product -> unique\_product\_count

Furthermore, we have implemented our own logic which checks if the total product count is returning null and thus updating the status variable (monthly status).