Graph based apps with dynamic models experimentations

# Introduction

Creating an application often leads to the following data workflow:

Serialized data 🡪 memory data 🡪 view data-> memory data -> serialized data

The application deserializes data then optionally process it in memory then display it in a view.

A user can modify a data through a view, which is optionally processed in memory and then stored

Because the serialized object, the memory objects, and view objects are different entities, the code will have some code like this:

myMemoryObject.memoryPropertyName = mySerializedObject.serializedPropertyName;

myViewObject.viewPropertyName = myMemoryObject.memoryPropertyName;

…

Which means that the syncronisation between the different entities are based on a static key value system.

Unfortunately, during the development the system will change a lot for different reasons among:

* The serialization system is changing
* The view components are changed or upgraded
* The structure of the objects has changed, to handle new features, or improvements

When a change occurs, the developer must update the bindings in the code using some find and replace operations and recompile the application.

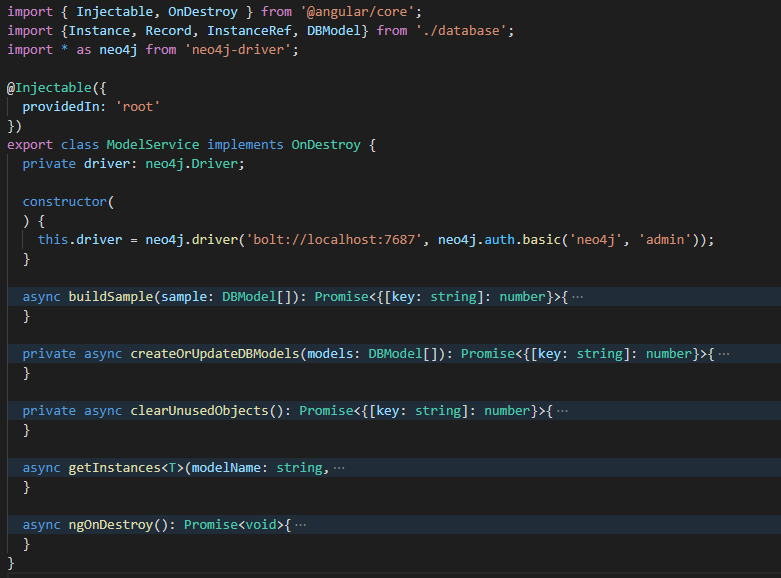
The current experimentation is trying to leverage on Graph database to have a dynamic binding system.

# Application Environment

The current sample will show a simple Angular web app, displaying data from a Neo4j Graph database.

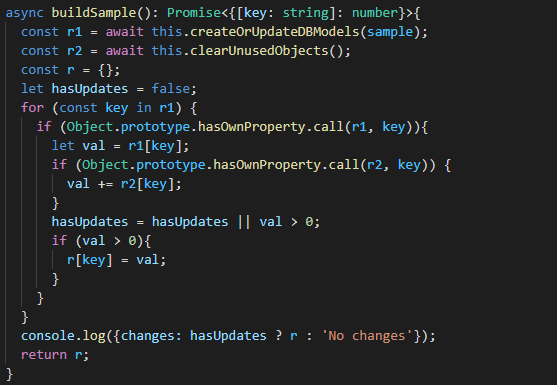
# Database Interaction

The application will communicate using the ModelService:

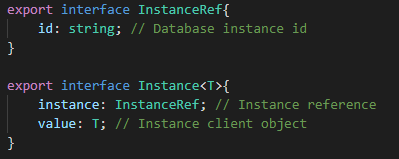


The model service has 2 public methods:

* buildSample: This will update the Database model using the private methods createOrUpdateDBModels and clearUnusedObjects providing a sample that declare the data structure and the instance. The method use as sample constant and logs the changes applied to the database

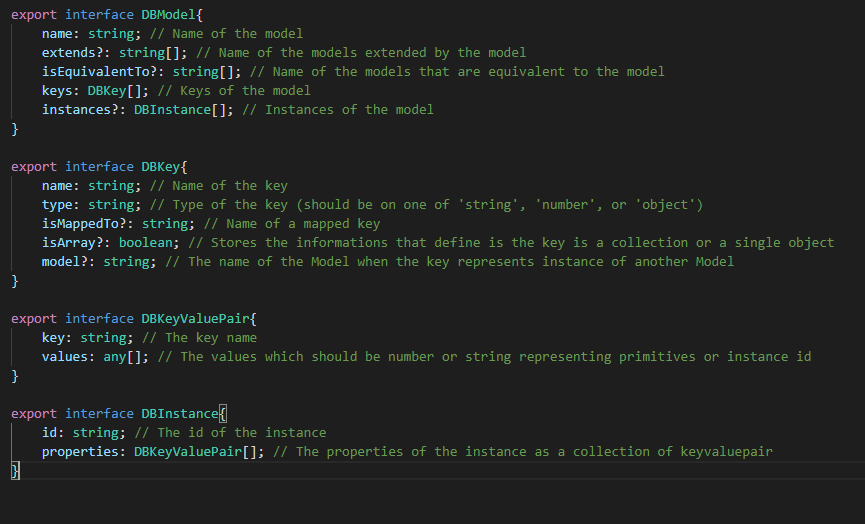


* getInstances will take a model name and will retrieve the instances for that model. A instance is defined as following:



It has also 2 private methods:

* createOrUpdateDBModels will be used to update the data structure of our project using a DBModel collection as an input which is defined as following:

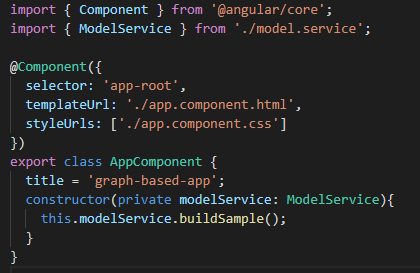


* clearUnusedObjects will removed object that have are no more used in the application

The queries will be detailed at the end of the document.

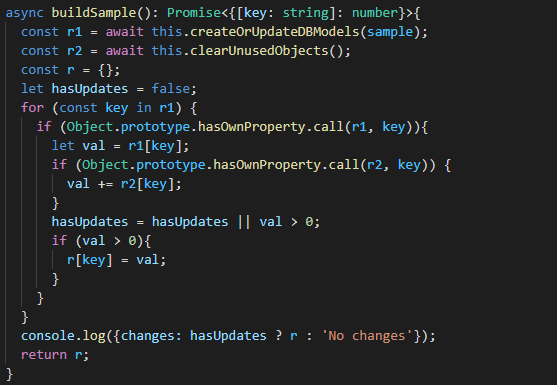
# Let’s build a simple Movie application

## Building the sample



The application will run the ModelService.buildSample() application once, when launching.

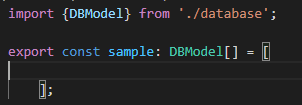
The buildSample code is detailed below:



It calls createOrUpdateDBModels method which will run a cypher query to the database providing the models then it calls clearUnusedObjects that will clean the database.

The changes of the 2 queries are aggregated and logged so we can track what are the changes made to the database

At the beginning of the project the sample const was defined like this



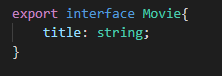
Which means that de database contains nothing and should be empty

During the evolution of the project we will modify the sample to declare what we want in the database.

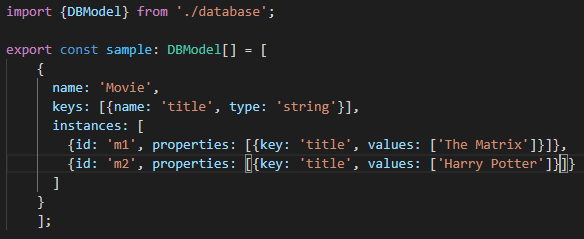
The buildsample method will apply that model to the Graph database and log the effective changes in the database.

## Displaying the movies

We want to display a list of movies and we created a simple Movie interface that represents a movie which is defined by its title



Let’s update the sample like this:



This tells the system, that we want a Model named ‘Movie’, containing the key named ‘title’ and having 2 instances ‘m1’ and ‘m2’ configured with ‘The Matrix’ and ‘Harry Potter’ as a title

We create then a MovieListComponent:

A screenshot of a cell phone

Description automatically generated

And its template

A close up of a screen

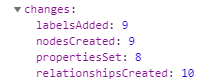
Description automatically generated

When running the application, it will display the movie list:

A picture containing drawing

Description automatically generated

When looking into the logs, the first time the application was launched we will see the following logs:

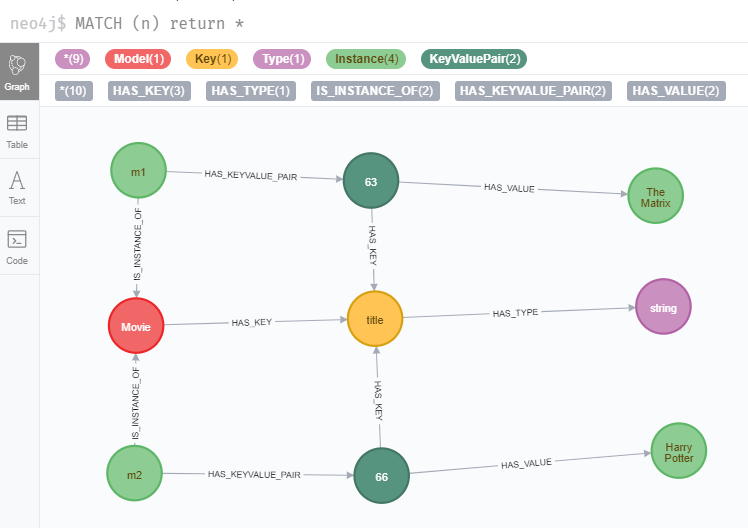


If we refresh the page or relaunch the application the logs will be:



This is because of the createOrUpdateDBModels is not incremential but differiental (the query will be detailed later)

Let’s now have a look at the database:



If you look to the labels we have:

* Model
* Key
* KeyValuePair
* Instance
* Type

If you look to the relationships, we have:

* IS\_INSTANCE\_OF
* HAS\_KEY
* HAS\_KEYVALUE\_PAIR
* HAS\_VALUE

And we can see exactly what we defined in the sample:

* A model, named ‘Movie’ which has a key named ‘title’
* 2 instances ‘m1’ and ‘m2’ having both a key/value pair binding the title to values which are instances having no model meaning that they are primitive
* The key as a type which is a string

We didn’t create a static model but a dynamic model with generic concepts

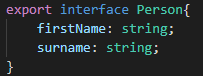
I creates more nodes that if we have chosen a static model, but it has some benefits that we will explore when improving the application

# Improving the application

A movie is not only a title he has also actors and we want to display the actors of a movie in our app.

## Creating the Person model

We create first an interface Person as following:



And a PersonComponent:

A screen shot of a smart phone

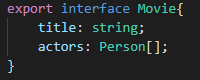
Description automatically generated

With the template:

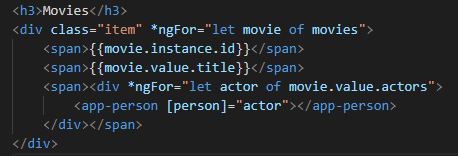


## Improving the Movie model

The movie model should know exposes an ‘actors’ property which is a Person collection:

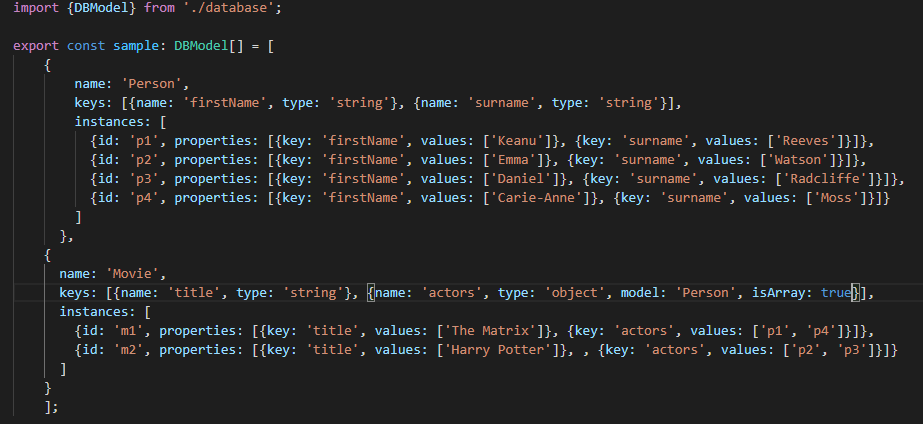


We can then improve the MovieListComponent template:



## Updating the sample

We will update the sample as following:



We have added the Model ‘Person’ and some instances of it, and added the ‘actors’ key

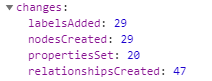
## Watching the changes

The application will now display the following:

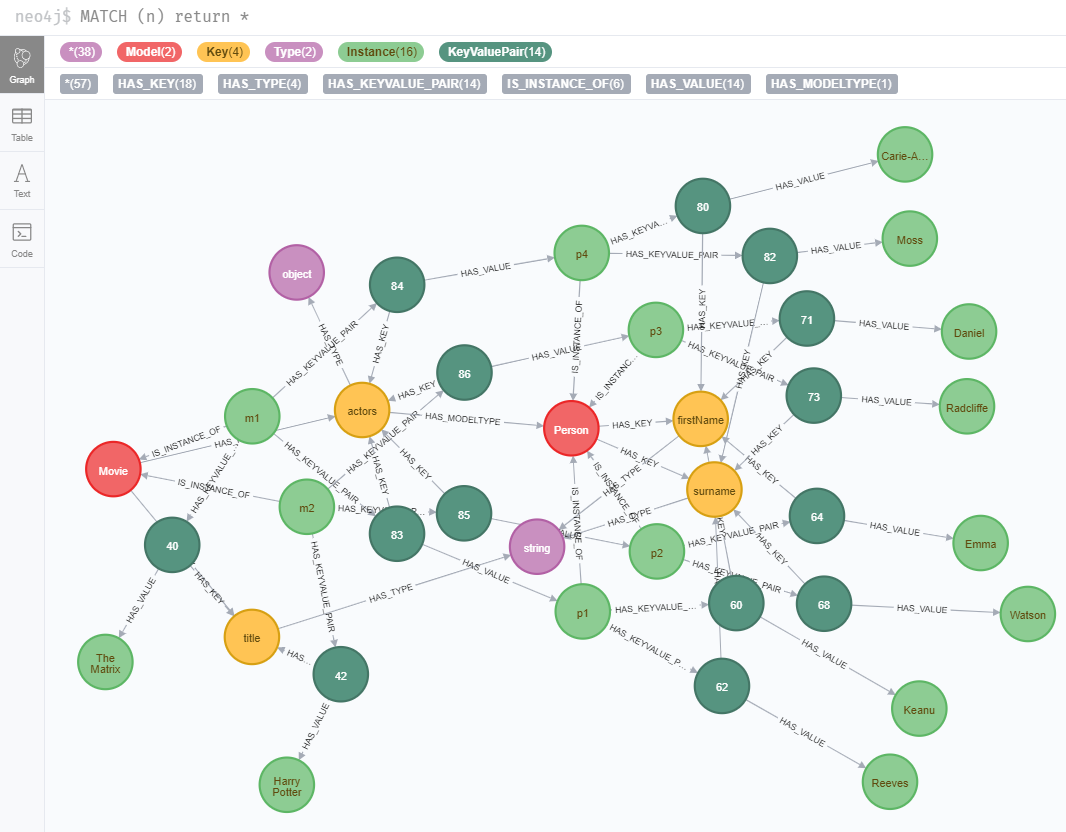
A screenshot of a cell phone

Description automatically generated

And the DB changes log will be:



In the database we have now



On the right there are all the primitives representing the ‘firstName’ and ‘surname’ which are related to the instances of Person. The movies have know new key/value pairs that stores the actors collection.

No new labels were added

One relationship was added: HAS\_MODELTYPE which tells to the system that actors are Person

Note that we didn’t have to change the modelService queries or the MovieListComponent the change of the sample has updated the database which has now all the information to return Movie instances.

What will happen if we change the data structure of the sample?

# Changing the data structure

Well, having the actors, is nice but we should have also the character played in each movie. We need to update our model to reflect that need, creating a Role model containing the actor and the character, and changing the ‘actors’ properties to ‘roles’.

## Adding Role model

This is the Role interface definition:

A picture containing sitting, holding, city, person

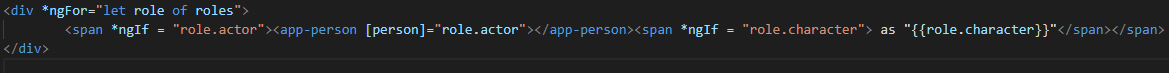
Description automatically generated

With a RoleListComponent:

A screenshot of a cell phone

Description automatically generated

And the template:



## Updating Movie model

The Movie interface is now like this:

A screenshot of a cell phone

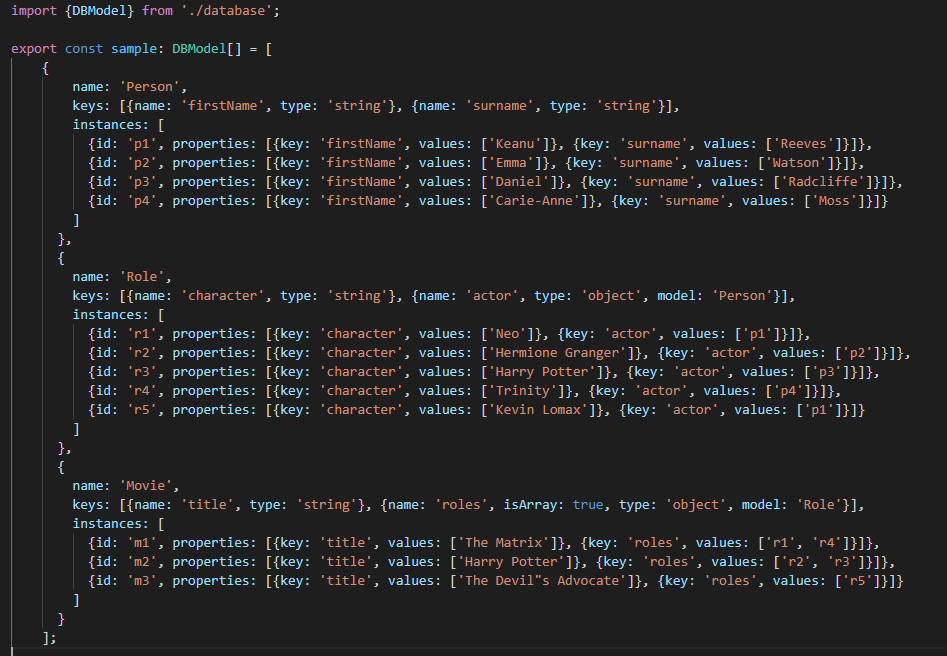
Description automatically generated

And the MovieListComponent template will be:

A screen shot of a person

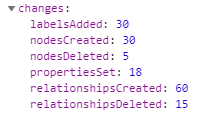
Description automatically generated

## Updating the sample



The sample reflects the changes in the model. We added the Role model, and the Movie update. We have also added a new movie.

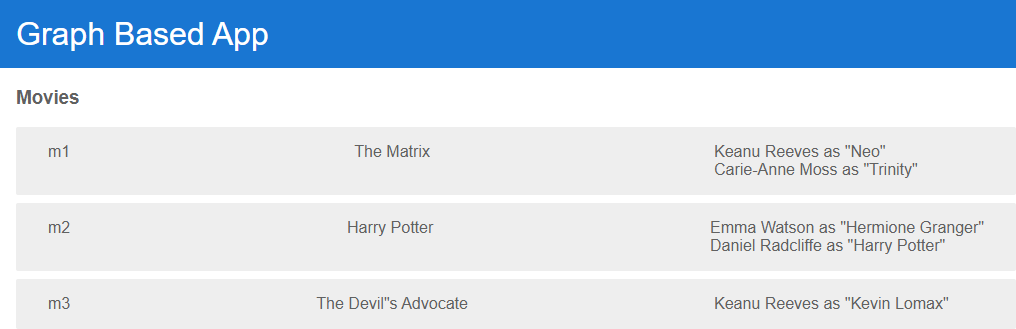
When running the application for the first time after updating the sample the changes in the databases logged by the application are:



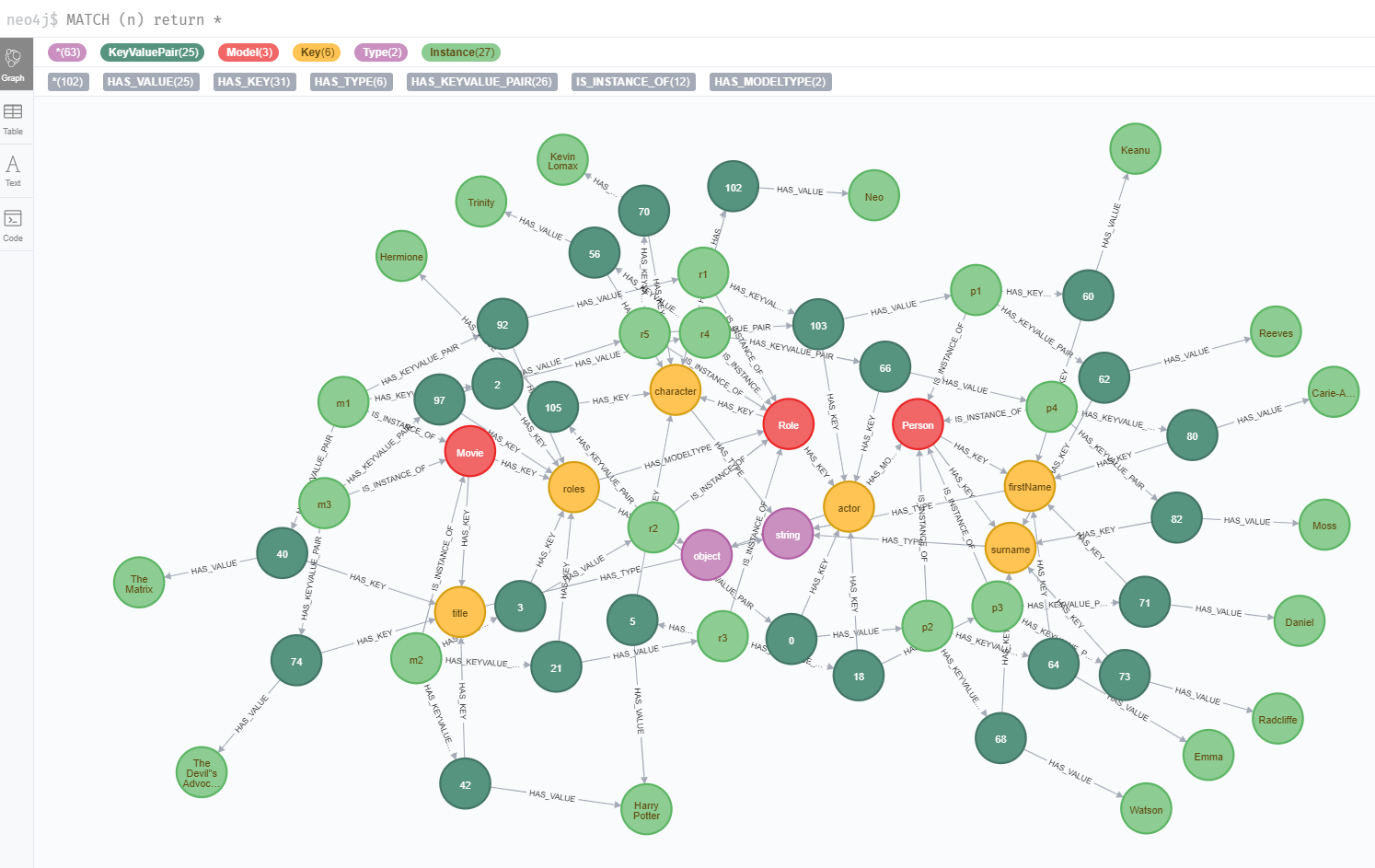
Some nodes and relationships were deleted: because the ‘actors’ properties is not used it was removed with all the key/value pairs related to it.

## Watching the changes

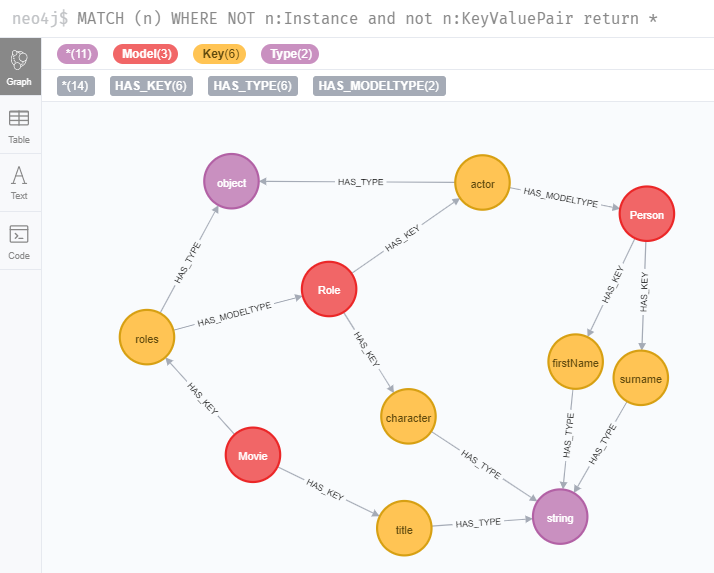
The application will now display:



The database looks now like:



Watch the entire graph beginning to become a little confusing, let’s focus on some nodes of interest

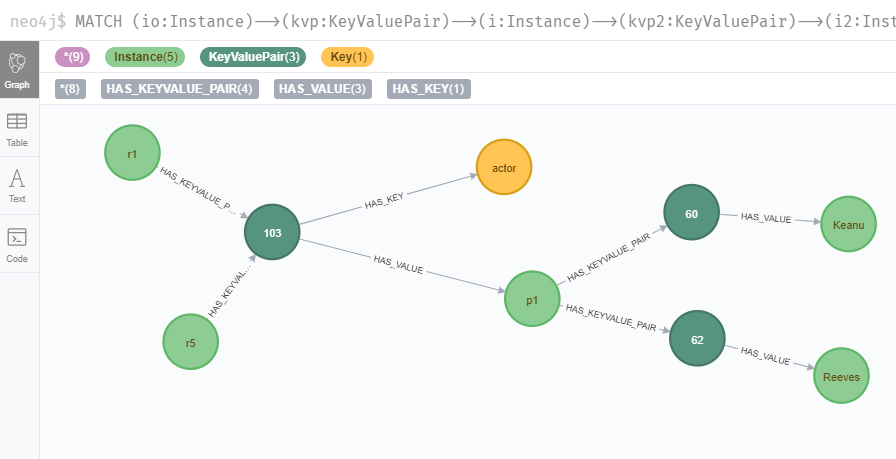


Excluding the instances and the key/value pairs we can focus on the data structure and we can see

That:

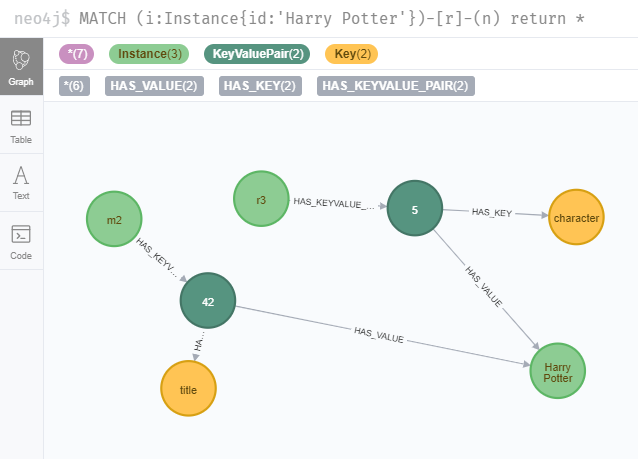
* the property ‘actors’ has disappeared from the model ‘Movie’ which has now the properties roles with ‘Role’ as model type
* The model ‘Role’ was added with 2 keys ‘character’ and ‘actor’ which is a ‘Person’
* The ‘roles’ and ‘actor’ keys are defining objects
* The other keys are defining primitives of type ‘string

Let’s have a look at Keanu Reeves now:



Because Roles ‘r1’ and ‘r5’ have both ‘p1’ as an actor (Keanu Reeves), they are sharing the same key/value pair

If we have a look at ‘Harry Potter’ primitive



Because ‘r3’ as ‘Harry Potter’ as character and ‘m2’ has ‘Harry Potter’ as title, they have both a key/value pair sharing the same primitive instance

The Model, Key, KeyValuePair, Instance system allows us to easily modify the entire structure of the data without having to destroy everything and recreate everything. We can destroy some relationships, add new ones, and then remove the nodes without specific relationships.

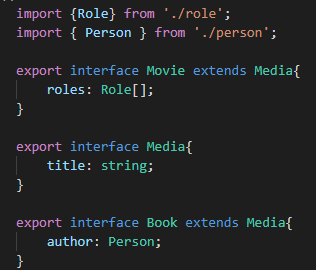
Let’s explore now how we can extend our model.

# Extending the model

We have what we want in our movie list, but there is a new requirement for the application: display movies but also books.

## Adding Media and Book model

We will create a Media interface that will store the title. The Book interface will have an ‘author’ property which is a Person. The Movie interface will be updated to remove the ‘title’ property and both Book and Movie will extend the Media interface



This will allow us to create a BookListComponent:

A screenshot of a cell phone

Description automatically generated

With its template:

A picture containing holding, sitting, person, orange

Description automatically generated

And a MediaListComponent:



With its template:

A screen shot of a person

Description automatically generated

## Adding navigation system

We added some buttons in the TopBarComponent template with different routes:

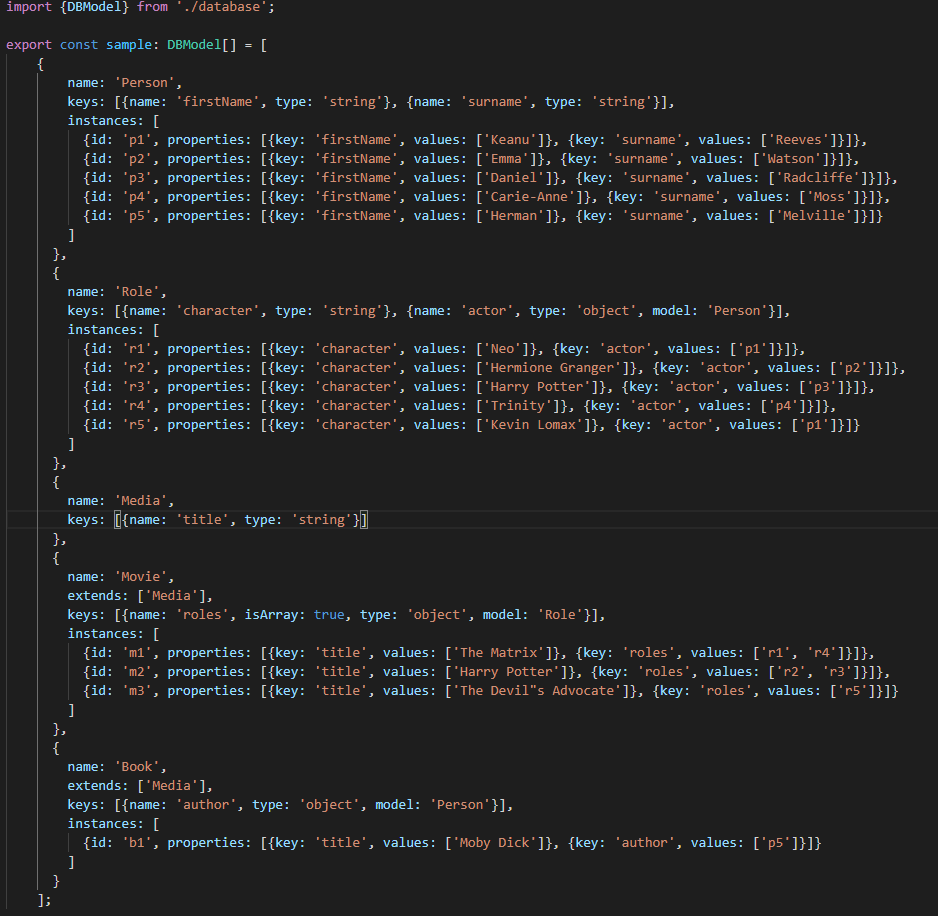
A close up of a sign

Description automatically generated

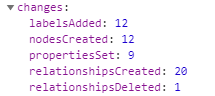
Our app having a routing module describing the following routes:



## Updating the sample

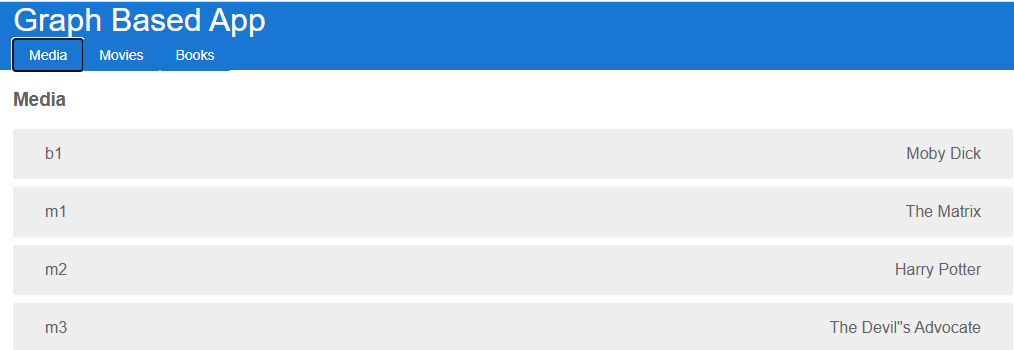


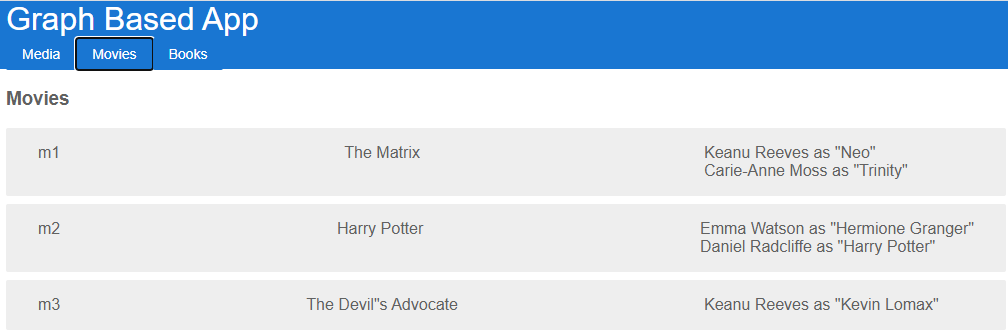
We added ‘Media’ and ‘Book’, added a ‘Person’ instance and a ‘Book’ instance and it gives us the following changes

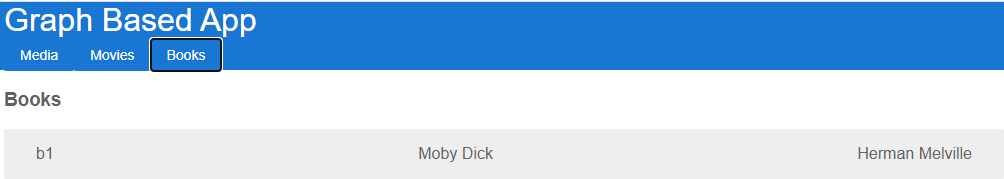


## Watching the changes

The app will display the following:

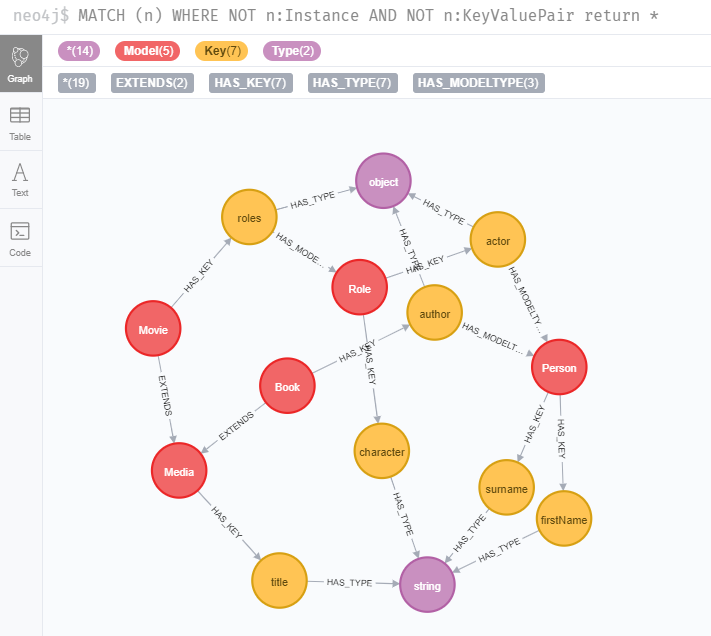






Note that the objects have been retrieved with the query getInstances, and the unique change in the query was the name of the model. Because the relation between models are described in the database the method can retrieve the appropriate json objects for each model just knowing the model name.

The database model structure looks now like this:



The new relationship: EXTENDS tells the system how some models can extends other